INIT

```
!pip install ta
```

```
Collecting ta
  Downloading ta-0.11.0.tar.gz (25 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packag
Building wheels for collected packages: ta
  Building wheel for ta (setup.py) ... done
  Created wheel for ta: filename=ta-0.11.0-py3-none-any.whl size=29413 sha256=b8
  Stored in directory: /root/.cache/pip/wheels/5f/67/4f/8a9f252836e053e532c6587a
Successfully built ta
Installing collected packages: ta
Successfully installed ta-0.11.0
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
Start coding or generate with AI.
```

RUN ONLY A SINGLE TIME(only when you want to clear the processed_symbols csv or create it for the first time) AND CLEAR THE MODELS FOLDER IN GOOGLE DRIVE(be very careful with this)

```
import pandas as pd
processed_symbols = pd.DataFrame(columns= ["processed_symbols"])
processed_symbols.to_csv(f"/content/drive/MyDrive/processed_symbols_50.csv")
```

```
import os
folder = "models_nifty"
os.chdir(f'/content/drive/MyDrive/{folder}')
for filename in os.listdir():
    os.remove(filename)
```

MODEL CREATOR

```
import yfinance as yf
import pandas as pd

def get_stock_symbols(exchange):
    df = pd.read_csv(f"/content/drive/MyDrive/{exchange}.csv")
    print(df)
    # Extract the 'Symbol' column as a list
    symbols = df['SYMBOL \n'].to_list()
    return list(symbols)

# Get NASDAQ symbols
symbols = get_stock_symbols('nifty')


# Combine the lists if needed
all_symbols = symbols

# Print or use the list as needed
print(all_symbols)
print(len(all_symbols))
```

```
     SYMBOL \n    OPEN \n    HIGH \n     LOW \n PREV. CLOSE \n     LTP \n  \
0     NIFTY 50  19,674.75  19,806.00  19,667.45      19,765.20  19,731.80
1      SBILIFE   1,354.00   1,434.45   1,354.00       1,359.95   1,413.70
2   APOLLOHOSP   5,332.20   5,493.35   5,306.15       5,338.75   5,485.00
3     HDFCLIFE     632.85     660.00     632.00         634.95     652.30
4           LT   3,050.00   3,115.45   3,045.15       3,051.15   3,106.25
5   TATACONSUM     922.00     934.00     921.00         920.05     934.00
6     DIVISLAB   3,550.00   3,619.00   3,550.00       3,548.25   3,600.00
7   HINDUNILVR   2,465.00   2,533.50   2,465.00       2,491.20   2,527.00
8       GRASIM   1,941.05   1,983.60   1,940.35       1,942.60   1,970.45
9   BAJAJ-AUTO   5,556.90   5,674.95   5,479.35       5,550.90   5,620.25
10  HEROMOTOCO   3,289.70   3,368.80   3,280.05       3,280.05   3,320.00
11  ASIANPAINT   3,150.00   3,228.25   3,145.65       3,130.30   3,167.00
12     DRREDDY   5,565.00   5,639.50   5,545.00       5,544.25   5,609.00
13       CIPLA   1,236.00   1,256.10   1,235.50       1,234.80   1,248.20
14   NESTLEIND  24,100.00  24,484.90  24,064.60      24,082.90  24,335.00
15   POWERGRID     207.70     210.70     206.90         207.20     209.00
16         M&M   1,575.00   1,596.00   1,572.65       1,569.50   1,582.00
17   EICHERMOT   3,824.90   3,898.00   3,816.90       3,836.75   3,859.10
18      MARUTI  10,500.00  10,606.80  10,499.15      10,484.50  10,529.35
19   SUNPHARMA   1,189.80   1,198.40   1,187.95       1,189.55   1,192.45
```

20	TATASTEEL	124.70	125.75	124.30	124.70	125.00	
21	ADANIENT	2,205.90	2,237.95	2,201.00	2,205.90	2,209.00	
22	TATAMOTORS	682.50	687.65	679.60	680.40	681.25	
23	TITAN	3,339.00	3,366.00	3,326.70	3,335.95	3,337.50	
24	ITC	439.25	441.50	438.30	438.65	438.85	
25	TCS	3,497.85	3,524.50	3,492.55	3,497.85	3,499.00	
26	ULTRACEMCO	8,785.00	8,869.60	8,771.65	8,775.05	8,777.85	
27	BRITANNIA	4,703.00	4,745.60	4,690.10	4,702.10	4,703.45	
28	UPL	560.35	565.70	558.65	560.85	561.00	
29	COALINDIA	348.00	348.55	342.30	345.80	345.60	
30	LTIM	5,540.00	5,548.00	5,486.50	5,519.50	5,510.00	
31	HDFCBANK	1,490.00	1,507.50	1,489.00	1,508.35	1,505.00	
32	HCLTECH	1,313.40	1,325.75	1,306.40	1,311.05	1,308.00	
33	INDUSINDBK	1,490.55	1,509.00	1,485.55	1,498.65	1,495.00	
34	RELIANCE	2,352.90	2,373.25	2,352.05	2,360.70	2,354.85	
35	BHARTIARTL	948.80	954.00	946.00	949.55	946.75	
36	JSWSTEEL	772.00	774.90	766.00	769.15	766.50	
37	ADANIPORTS	811.70	821.95	809.05	813.60	810.00	
38	NTPC	253.00	253.10	251.00	252.35	251.20	
39	BAJAJFINSV	1,593.05	1,620.65	1,581.60	1,620.65	1,613.05	
40	TECHM	1,212.20	1,213.95	1,198.00	1,207.20	1,200.85	
41	INFY	1,441.05	1,451.00	1,435.05	1,444.90	1,437.20	
42	WIPRO	398.00	399.00	394.50	397.10	394.90	
43	KOTAKBANK	1,760.00	1,784.25	1,759.30	1,772.85	1,760.60	
44	HINDALCO	503.05	506.45	496.85	504.30	498.30	
45	ICICIBANK	926.55	938.40	920.15	936.00	920.80	
46	BAJFINANCE	7,140.50	7,260.00	7,108.05	7,362.45	7,221.75	
47	BPCL	400.00	407.10	389.25	398.65	390.40	
48	ONGC	200.85	201.25	195.55	201.80	196.80	
49	AXISBANK	1,014.90	1,020.00	991.85	1,026.35	992.65	
50	SBIN	574.50	574.50	562.10	584.65	562.90	

	CHNG \n	%CHNG \n	VOLUME \n(shares)	VALUE \n (₹ Billions)	52W H \n	\
0	-33.40	-0.17	23,68,27,462	216.81	20,222.45	
1	53.75	3.95	31,92,944	4.53	1,434.45	
2	146.25	2.74	7,35,541	4.00	5,493.35	

```
symbols = all_symbols[1:51]
print(symbols)
print(len(symbols))
```

```
['SBILIFE', 'APOLLOHOSP', 'HDFCLIFE', 'LT', 'TATACONSUM', 'DIVISLAB', 'HINDUNILV
50
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error
import yfinance as yf
import ta
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.optimizers import Adam
from tqdm.notebook import tqdm
from IPython.display import HTML, clear_output
import datetime

count = 0
total_symbols = len(symbols)
processed_symbols = pd.read_csv(f"/content/drive/MyDrive/processed_symbols_50.c
initial_value = len(processed_symbols)
progress_bar_sites = tqdm(total=total_symbols, initial = initial_value, desc="P
for symbol in symbols:
    # Check if the specified Site ID already exists in the DataFrame
    existing_row = processed_symbols[processed_symbols['processed_symbols'] ==
    if not existing_row.empty:
      print(f"Site ID {symbol} already processed, skipping to the next one.....
      continue
    # Define the stock symbol and the date range you're interested in

    stock_symbol = f"{symbol}.NS"
    start_date = '1950-01-01'
    # Get the current date
    current_date = datetime.datetime.today()

    # Format the current date as a string
    end_date = current_date.strftime('%Y-%m-%d')

    # Download historical stock data
    data = yf.download(stock_symbol, start=start_date, end=end_date)

    # Save the data to a CSV file
    data.to_csv('/content/close_price.csv', index=True)

    # Display the first few rows of the dataset
    print(data.head())
    print(data.tail())

    # Check for missing values
    print(data.isnull().sum())

    # Visualize the data
    plt.figure(figsize=(12, 6))
    plt.plot(data['Close'], label='Closing Price')
    plt.xlabel('Date')
    plt.ylabel('Closing Price')
    plt.title(f'{stock_symbol} Stock Price Over Time')
    plt.legend()
    plt.show()

    # Reset the index to make Date a regular column
```

```
data.reset_index(inplace=True)

# Convert the 'Date' column to datetime format
data['Date'] = pd.to_datetime(data['Date'])

# Set 'Date' as the index again (useful for time-series analysis)
data.set_index('Date', inplace=True)

# Display the updated dataset
print(data.head())

# Add various technical indicators
data = ta.add_all_ta_features(data, open='Open', high='High', low='Low', cl
data.to_csv('/content/close_price.csv', index=True)
# Display the updated dataset with technical indicators
print(data)

# Remove the open, high, low, and adjusted close prices from the dataframe
data = data.drop(columns=['Open', 'High', 'Low', 'Adj Close'])
# Print the dataframe
print(data)

# Define the percentage for training set
train_percentage = 1

# Calculate the index to split the data
split_index = int(len(data) * train_percentage)

# Split the data into training and testing sets
train_data = data.iloc[:split_index]
#test_data = data.iloc[split_index:]

# Display the shapes of the training and testing sets
print("Training set shape:", train_data.shape)
#print("Testing set shape:", test_data.shape)


# Initialize the scaler
scaler = StandardScaler()

# Fit and transform the scaler on the training data
train_scaled = scaler.fit_transform(train_data.drop('Close', axis=1))

# Transform the testing data using the same scaler
#test_scaled = scaler.transform(test_data.drop('Close', axis=1))

# Display the first few rows of the scaled training data
print(pd.DataFrame(train_scaled, columns=train_data.columns[:-1]))

# Define the input features and target variable for training set
X_train = train_scaled
```

```
        y_train = train_data['Close'].values

        # Define the input features and target variable for testing set
        #X_test = test_scaled
        #y_test = test_data['Close'].values

        # Display the shapes of the input and output for both training and testing
        print("Training set - Input shape:", X_train.shape, " Output shape:", y_tra
        #print("Testing set - Input shape:", X_test.shape, " Output shape:", y_test




        # Reshape data for LSTM input (assuming a time series sequence)
        X_train_reshaped = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
        #X_test_reshaped = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

        # Define the LSTM model
        model = Sequential()

        # LSTM layer with 50 units and 'relu' activation function (you can experime
        model.add(LSTM(64, activation='relu', input_shape=(X_train_reshaped.shape[1

        # Dense output layer with linear activation (common for regression tasks)
        model.add(Dense(1, activation='linear'))

        # Compile the model with the Adam optimizer and Huber loss function
        model.compile(optimizer=Adam(learning_rate=0.001), loss='huber_loss')

        # Print the model summary
        model.summary()

        # Train the model
        model.fit(X_train_reshaped, y_train, epochs=200, batch_size=32, verbose=1)

        model.save(f"/content/drive/MyDrive/models_nifty/model_{symbol}.h5")
        # Add the processed symbol to the DataFrame
        processed_symbols = processed_symbols.append({"processed_symbols": symbol},
        processed_symbols.to_csv("/content/drive/MyDrive/processed_symbols_50.csv")

        count = count+1
        print("Number of models trained: ", count)
        progress_bar_sites.update(1)

    progress_bar_sites.close()
```

```
Processing.....: 100%                                    50/50 [00:00<?, ?it/s]
Site ID SBILIFE already processed, skipping to the next one.....
Site ID APOLLOHOSP already processed, skipping to the next one.....
Site ID HDFCLIFE already processed, skipping to the next one.....
Site ID LT already processed, skipping to the next one.....
Site ID TATACONSUM already processed, skipping to the next one.....
Site ID DIVISLAB already processed, skipping to the next one.....
Site ID HINDUNILVR already processed, skipping to the next one.....
Site ID GRASIM already processed, skipping to the next one.....
Site ID BAJAJ-AUTO already processed, skipping to the next one.....
Site ID HEROMOTOCO already processed, skipping to the next one.....
Site ID ASIANPAINT already processed, skipping to the next one.....
Site ID DRREDDY already processed, skipping to the next one.....
Site ID CIPLA already processed, skipping to the next one.....
Site ID NESTLEIND already processed, skipping to the next one.....
Site ID POWERGRID already processed, skipping to the next one.....
Site ID M&M already processed, skipping to the next one.....
Site ID EICHERMOT already processed, skipping to the next one.....
Site ID MARUTI already processed, skipping to the next one.....
Site ID SUNPHARMA already processed, skipping to the next one.....
Site ID TATASTEEL already processed, skipping to the next one.....
Site ID ADANIENT already processed, skipping to the next one.....
Site ID TATAMOTORS already processed, skipping to the next one.....
Site ID TITAN already processed, skipping to the next one.....
Site ID ITC already processed, skipping to the next one.....
Site ID TCS already processed, skipping to the next one.....
Site ID ULTRACEMCO already processed, skipping to the next one.....
Site ID BRITANNIA already processed, skipping to the next one.....
Site ID UPL already processed, skipping to the next one.....
Site ID COALINDIA already processed, skipping to the next one.....
Site ID LTIM already processed, skipping to the next one.....
Site ID HDFCBANK already processed, skipping to the next one.....
Site ID HCLTECH already processed, skipping to the next one.....
Site ID INDUSINDBK already processed, skipping to the next one.....
Site ID RELIANCE already processed, skipping to the next one.....
Site ID BHARTIARTL already processed, skipping to the next one.....
Site ID JSWSTEEL already processed, skipping to the next one.....
Site ID ADANIPORTS already processed, skipping to the next one.....
Site ID NTPC already processed, skipping to the next one.....
Site ID BAJAJFINSV already processed, skipping to the next one.....
Site ID TECHM already processed, skipping to the next one.....
Site ID INFY already processed, skipping to the next one.....
Site ID WIPRO already processed, skipping to the next one.....
Site ID KOTAKBANK already processed, skipping to the next one.....
Site ID HINDALCO already processed, skipping to the next one.....
Site ID ICICIBANK already processed, skipping to the next one.....
Site ID BAJFINANCE already processed, skipping to the next one.....
Site ID BPCL already processed, skipping to the next one.....
Site ID ONGC already processed, skipping to the next one.....
Site ID AXISBANK already processed, skipping to the next one.....
Site ID SBIN already processed, skipping to the next one.....
```

ENSEMBLE

PRE-PROCESSING

```
import pandas as pd
processed_symbols = pd.read_csv(f"/content/drive/MyDrive/processed_symbols_50.c
symbols = list(processed_symbols["processed_symbols"])
print(symbols)
print(len(symbols))
```

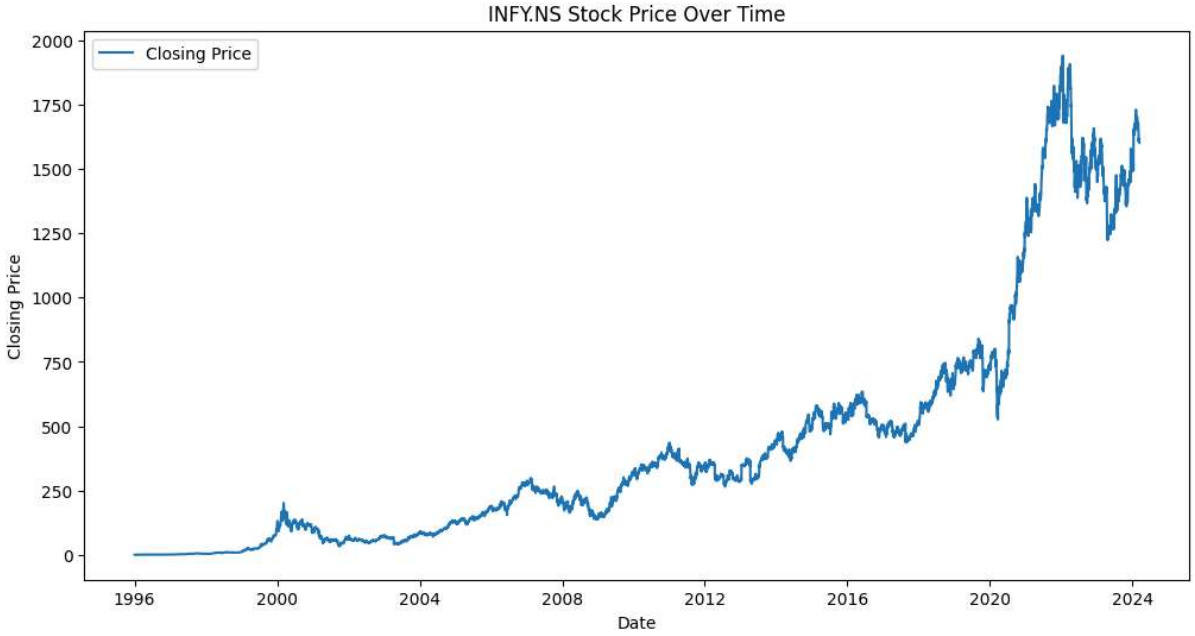
```
['SBILIFE', 'APOLLOHOSP', 'HDFCLIFE', 'LT', 'TATACONSUM', 'DIVISLAB', 'HINDUNILV
50
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error
import yfinance as yf
import ta
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.optimizers import Adam
from sklearn.ensemble import RandomForestRegressor
import datetime
# Define the stock symbol and the date range you're interested in
stock_symbol = "INFY.NS"
start_date = '1950-01-01'
# Get the current date
current_date = datetime.datetime.today()

# Format the current date as a string
end_date = current_date.strftime('%Y-%m-%d')
# Download historical stock data
data = yf.download(stock_symbol, start=start_date, end=end_date)

# Save the data to a CSV file
data.to_csv('/content/drive/MyDrive/test/close_price.csv', index=True)

# Display the first few rows of the dataset
print(data.head())
print(data.tail())

# Check for missing values
```

```
print(data.isnull().sum())

# Visualize the data
plt.figure(figsize=(12, 6))
plt.plot(data['Close'], label='Closing Price')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.title(f'{stock_symbol} Stock Price Over Time')
plt.legend()
plt.show()

# Reset the index to make Date a regular column
data.reset_index(inplace=True)

# Convert the 'Date' column to datetime format
data['Date'] = pd.to_datetime(data['Date'])

# Set 'Date' as the index again (useful for time-series analysis)
data.set_index('Date', inplace=True)

# Display the updated dataset
print(data.head())

# Add various technical indicators
data = ta.add_all_ta_features(data, open='Open', high='High', low='Low', close=
data.to_csv('/content/drive/MyDrive/test/close_price.csv', index=True)

# Remove the open, high, low, and adjusted close prices from the dataframe
data = data.drop(columns=['Open', 'High', 'Low', 'Adj Close'])
# Display the updated dataset with technical indicators
print(data)




# Define the percentage for training set
train_percentage = 1

# Calculate the index to split the data
split_index = int(len(data) * train_percentage)

# Split the data into training and testing sets
test_data = data.iloc[:split_index]


# Display the shapes of the training and testing sets
print("Test set shape:", test_data.shape)



# Initialize the scaler
```

```
scaler = StandardScaler()

# Fit and transform the scaler on the training data
test_scaled = scaler.fit_transform(test_data.drop('Close', axis=1))



# Display the first few rows of the scaled training data
print(pd.DataFrame(test_scaled, columns=test_data.columns[:-1]))

# Define the input features and target variable for training set
X_test = test_scaled
y_test = test_data['Close'].values



# Display the shapes of the input and output for both training and testing sets
print("Test set - Input shape:", X_test.shape, " Output shape:", y_test.shape)
#print("Testing set - Input shape:", X_test.shape, " Output shape:", y_test.sha



# Reshape data for LSTM input (assuming a time series sequence)
X_test_reshaped = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
```

```
[**********100%%**********]  1 of 1 completed
                 Open      High       Low     Close  Adj Close  Volume
Date
1996-01-01  0.794921  0.796679  0.791015  0.796679   0.517830  204800
1996-01-02  0.794921  0.798828  0.793457  0.793457   0.515736  204800
1996-01-03  0.798828  0.798828  0.798828  0.798828   0.519227  102400
1996-01-04  0.791015  0.794921  0.791015  0.793554   0.515799  307200
1996-01-05  0.784179  0.784179  0.784179  0.784179   0.509705   51200
                  Open         High          Low        Close    Adj Close  \
Date
2024-03-05  1634.000000  1637.349976  1602.000000  1606.500000  1606.500000
2024-03-06  1602.800049  1620.150024  1576.000000  1617.550049  1617.550049
2024-03-07  1616.599976  1628.449951  1604.199951  1616.449951  1616.449951
2024-03-11  1608.150024  1613.800049  1588.000000  1600.599976  1600.599976
2024-03-12  1600.000000  1625.000000  1597.650024  1612.949951  1612.949951

              Volume
Date
2024-03-05  4596579
2024-03-06  6420684
2024-03-07  6348359
2024-03-11  6752895
2024-03-12  4614222
Open          0
High          0
Low           0
Close         0
Adj Close     0
Volume        0
dtype: int64
```




```
                 Open      High       Low     Close  Adj Close  Volume
Date
1996-01-01  0.794921  0.796679  0.791015  0.796679   0.517830  204800
1996-01-02  0.794921  0.798828  0.793457  0.793457   0.515736  204800
1996-01-03  0.798828  0.798828  0.798828  0.798828   0.519227  102400
1996-01-04  0.791015  0.794921  0.791015  0.793554   0.515799  307200
1996-01-05  0.784179  0.784179  0.784179  0.784179   0.509705   51200
```

```
                 Close   Volume    Volume_adi  Volume_obv  Volume_cmf  \
Date
1996-01-01    0.796679   204800  2.048000e+05      204800    1.000000
1996-01-02    0.793457   204800  0.000000e+00           0    0.000000
1996-01-03    0.798828   102400  0.000000e+00      102400    0.000000
1996-01-04    0.793554   307200  9.217735e+04     -204800    0.112521
1996-01-05    0.784179    51200  9.217735e+04     -256000    0.105902
...                ...      ...           ...         ...         ...
2024-03-05  1606.500000  4596579  2.017739e+08  3951254220   -0.066059
2024-03-06  1617.550049  6420684  2.074383e+08  3957674904    0.028522
2024-03-07  1616.449951  6348359  2.075038e+08  3951326545    0.042268
2024-03-11  1600.599976  6752895  2.073467e+08  3944573650    0.059851
2024-03-12  1612.949951  4614222  2.078950e+08  3949187872    0.069817

               volume_fi     volume_em  volume_sma_em    volume_vpt  \
Date
1996-01-01  0.000000e+00      0.000000       0.000000  0.000000e+00
1996-01-02 -6.598755e+02      0.006020       0.006020 -8.282827e+02
1996-01-03 -4.870370e+02      0.000000       0.003010 -1.351212e+02
1996-01-04 -6.489134e+02     -0.007451      -0.000477 -2.163308e+03
1996-01-05 -6.247832e+02     -0.000000      -0.000358 -2.768184e+03
...                 ...           ...            ...           ...
2024-03-05 -3.488755e+07 -22590.834028   -3024.616128  1.063031e+08
2024-03-06 -1.976806e+07 -14852.614604   -4943.256855  1.063473e+08
2024-03-07 -1.794174e+07   6971.266431   -5178.511497  1.063430e+08
2024-03-11 -3.066909e+07  -5893.233399   -5651.272171  1.062768e+08
2024-03-12 -1.814700e+07   6179.225049   -4320.412915  1.063124e+08

            volume_vwap  ...  momentum_ppo  momentum_ppo_signal  \
Date                     ...
1996-01-01     0.794791  ...      0.000000             0.000000
1996-01-02     0.795019  ...     -0.032272            -0.006454
1996-01-03     0.795781  ...     -0.003392            -0.005842
1996-01-04     0.794799  ...     -0.033554            -0.011384
1996-01-05     0.794175  ...     -0.150880            -0.039284
...                 ...  ...           ...                  ...
2024-03-05  1667.849710  ...      0.054225             0.710564
2024-03-06  1661.405186  ...     -0.137147             0.541021
2024-03-07  1654.667508  ...     -0.291375             0.374542
2024-03-11  1647.421739  ...     -0.486422             0.202349
2024-03-12  1643.436619  ...     -0.574207             0.047038

            momentum_ppo_hist  momentum_pvo  momentum_pvo_signal  \
Date
1996-01-01           0.000000      0.000000             0.000000
1996-01-02          -0.025818      0.000000             0.000000
1996-01-03           0.002450     -4.142012            -0.828402
1996-01-04          -0.022170      0.906596            -0.481403
1996-01-05          -0.111597     -5.528671            -1.490856
...                       ...           ...                  ...
2024-03-05          -0.656339     -5.498958            -8.602915
2024-03-06          -0.678169     -2.864967            -7.455326
2024-03-07          -0.665917     -0.932147            -6.150690
2024-03-11          -0.688771      1.136279            -4.693296
2024-03-12          -0.621245     -0.360291            -3.826695

            momentum_pvo_hist  momentum_kama  others_dr  others_dlr  \
```

```
            Date
            1996-01-01           0.000000       0.796679   0.000000    0.000000
            1996-01-02           0.000000       0.795152  -0.404435   -0.405255
            1996-01-03          -3.313609       0.796899   0.676916    0.674635
            1996-01-04           1.387999       0.795342  -0.660217   -0.662406
            1996-01-05          -4.037815       0.790245  -1.181399   -1.188433
            ...                       ...            ...        ...         ...
            2024-03-05           3.103957    1642.919023  -1.875150   -1.892954
            2024-03-06           4.590359    1641.589853   0.687834    0.685479
            2024-03-07           5.218543    1637.449521  -0.068010   -0.068033
            2024-03-11           5.829575    1630.656724  -0.980542   -0.985381
            2024-03-12           3.466404    1629.079665   0.771584    0.768623

                           others_cr
            Date
            1996-01-01      0.000000
            1996-01-02     -0.404435
            1996-01-03      0.269743
            1996-01-04     -0.392255
            1996-01-05     -1.569019
            ...                  ...
            2024-03-05  201549.592849
            2024-03-06  202936.606760
            2024-03-07  202798.521329
            2024-03-11  200809.015494
            2024-03-12  202359.197597

            [7087 rows x 88 columns]
            Test set shape: (7087, 88)
                     Close    Volume  volume_adi  volume_obv  volume_cmf  volume_fi  \
            0    -0.913056 -1.809267   -2.000692    6.370841    0.024325  -0.013527
            1    -0.913056 -1.810574   -2.000860   -0.098900    0.024317  -0.013526
            2    -0.919578 -1.810574   -2.000776   -0.098900    0.024319  -0.013527
            3    -0.906535 -1.809986   -2.001028    0.629083    0.024318  -0.013528
            4    -0.922838 -1.809986   -2.001070    0.586260    0.024318  -0.013527
            ...        ...       ...         ...         ...         ...        ...
            7082 -0.633361 -0.523403    1.245505   -0.526285   -0.383194  -3.114381
            7083 -0.517191 -0.487268    1.250780    0.085630   -0.206584  -2.052220
            7084 -0.521797 -0.486850    1.245564    0.174564   -0.185251   0.943360
            7085 -0.496034 -0.487852    1.240016    0.288323   -0.333918  -0.822441
            7086 -0.632237 -0.484355    1.243807    0.352799   -0.187649   0.834643

                  volume_em  volume_sma_em  volume_vpt  volume_vwap  ...  momentum_roc  \
            0     -0.052378      -2.686002   -0.913268    -0.237836  ...     -0.244729
            1     -0.052375      -2.686025   -0.913268     2.764655  ...     -0.256307
            2     -0.052376      -2.686006   -0.913266     2.764655  ...     -0.245946
            3     -0.052378      -2.686061   -0.913268    -0.231645  ...     -0.256767
            4     -0.052378      -2.686078   -0.913270    -0.460224  ...     -0.298858
            ...         ...            ...         ...          ...  ...           ...
            7082  -1.576237       0.202410    2.748967    -0.264580  ...     -0.225276
            7083  -2.542885       0.203610    2.734809    -0.748910  ...     -0.293931
            7084  -2.661411       0.203493    2.720008    -0.689306  ...     -0.349261
            7085  -2.899597       0.201693    2.704090    -1.098127  ...     -0.419235
            7086  -2.229085       0.202661    2.695336    -0.752149  ...     -0.450729

                  momentum_ppo  momentum_ppo_signal  momentum_ppo_hist  momentum_pvo  \
```

```
display(data)
```

Date	Close	Volume	volume_adi	volume_obv	volume_cmf	volume_fi
1996-01-01	0.796679	204800	2.048000e+05	204800	1.000000	0.000000e+00
1996-01-02	0.793457	204800	0.000000e+00	0	0.000000	-6.598755e+02
1996-01-03	0.798828	102400	0.000000e+00	102400	0.000000	-4.870370e+02
1996-01-04	0.793554	307200	9.217735e+04	-204800	-0.112521	-6.489134e+02
1996-01-05	0.784179	51200	9.217735e+04	-256000	0.105902	-6.247832e+02
...	...	...	...	...	...	...
2024-03-05	1606.500000	4596579	2.017739e+08	3951254220	-0.066059	-3.488755e+07
2024-03-06	1617.550049	6420684	2.074383e+08	3957674904	0.028522	-1.976806e+07
2024-03-07	1616.449951	6348359	2.075038e+08	3951326545	0.042268	-1.794174e+07
...	...	...	...	...	...	...
2024-03-11	1609.599976	6752895	2.073467e+08	3944573650	0.059851	-3.066909e+07
2024-03-12	1612.949951	4614222	2.078950e+08	3949187872	0.069817	-1.814700e+07

7087 rows × 88 columns

	momentum_pvo_signal	momentum_pvo_hist	momentum_kama	others_dr	...	others_dlr
2	-0.498359	-0.912989	0.224852	0.235456	...	-0.913348
3	-0.209213	-0.912993	-0.331434	-0.319565	...	-0.913355
4	-0.607349	-0.913004	-0.548260	-0.537925	...	
...	...	...	...	...	...	
7084	0.785693	2.681112	-0.085059	-0.072834	...	
7085	0.877651	2.666195	-0.464698	-0.453636	...	
7086	0.969509	2.623195				

```
[7087 rows x 87 columns]
Test set - Input shape: (7087, 87)  Output shape: (7087,)
```

```
# Import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error
from tensorflow.keras.models import load_model

# Load saved models
models = {}
index = "nifty"
for symbol in symbols:
    model = load_model(f"/content/drive/MyDrive/models_{index}/model_{symbol}.h
```

```
    models[symbol] = model
```

STACKING

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# Initialize an array to store predictions from individual models
individual_predictions = []

# Initialize an array to store the weights of each model
model_weights = []

for symbol, model in models.items():
    # Assuming the input shape is correct for your models
    individual_pred = model.predict(X_test_reshaped)
    individual_predictions.append(individual_pred)

    # Calculate the mean absolute error for each model
    mae = mean_absolute_error(y_test, individual_pred)

    # Assign weights inversely proportional to the mean absolute error
    weight = 1 / (mae + 1e-8)  # Adding a small value to avoid division by zerc
    model_weights.append(weight)

# Convert the list of individual predictions and weights into numpy arrays
individual_predictions = np.array(individual_predictions)
```

```
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 3ms/step
222/222 [==============================] - 1s 3ms/step
222/222 [==============================] - 1s 3ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 3ms/step
222/222 [==============================] - 1s 3ms/step
222/222 [==============================] - 1s 2ms/step
```

```
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 3ms/step
222/222 [==============================] - 1s 3ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 3ms/step
```

```
# Reshape individual predictions for stacking
individual_predictions_stacked = individual_predictions.reshape((individual_pre
```

```
from sklearn.ensemble import RandomForestRegressor
meta_model = RandomForestRegressor(n_estimators = 100, random_state = 42, verbo

# Fit the meta-model on individual predictions
meta_model.fit(individual_predictions_stacked, y_test)

# Make predictions with the meta-model
ensemble_predictions_stacked = meta_model.predict(individual_predictions_stacke

ensemble_predictions_stacked = ensemble_predictions_stacked.reshape((individual
```

```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:   25.8s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.1s
```

```
"""
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense

# Define a neural network regressor
nn_regressor = Sequential([
    Dense(2, activation='sigmoid', input_shape=(individual_predictions_stacked.
    Dense(1)
])

# Compile the neural network regressor
nn_regressor.compile(optimizer='adam', loss='mean_squared_error')

# Fit the neural network regressor on individual predictions
nn_regressor.fit(individual_predictions_stacked, y_test, epochs=200, batch_size

# Make predictions with the neural network regressor
ensemble_predictions_stacked = nn_regressor.predict(individual_predictions_stac

ensemble_predictions_stacked = ensemble_predictions_stacked.reshape((individual
"""
```

```
'\nfrom tensorflow.keras.models import Sequential\nfrom tensorflow.keras.layers
import Dense\n\n# Define a neural network regressor\nnn_regressor = Sequential
([\n    Dense(2, activation='sigmoid', input_shape=(individual_predictions_stac
ked.shape[1],)),\n    Dense(1)\n])\n\n# Compile the neural network regressor\nn
n_regressor.compile(optimizer='adam', loss='mean_squared_error')\n\n# Fit the n
eural network regressor on individual predictions\nnn regressor.fit(individual
```

WEIGHTED AVERAGING

```
model_weights = np.array(model_weights)

# Normalize weights to sum to 1
model_weights /= model_weights.sum()

# Calculate ensemble predictions using weighted average boosting
ensemble_predictions_boosted = np.average(individual_predictions, axis=0, weigh
```

```
from sklearn.metrics import mean_absolute_error
model_weights = np.array(model_weights)

# Normalize weights to sum to 1
model_weights /= model_weights.sum()

# Calculate ensemble predictions using weighted average
ensemble_predictions_dynamic = np.average(individual_predictions, axis=0, weigh
```

BAYESIAN BELIEF-BASED WEIGHTED AVERAGING

```
from scipy.stats import beta

# Initialize an array to store predictions from individual models
individual_predictions = []

# Initialize an array to store the weights of each model
model_weights = []

for symbol, model in models.items():
    # Assuming the input shape is correct for your models
    individual_pred = model.predict(X_test_reshaped)
    individual_predictions.append(individual_pred)

    # Calculate the Bayesian weight for each model
    alpha = 1  # prior hyperparameter
    beta_val = 1 # prior hyperparameter
    mae = mean_absolute_error(y_test, individual_pred)

    # Adjust alpha and beta_val if they would result in negative posterior valu
    while alpha + mae <= 0 or beta_val + len(y_test) - mae <= 0:
        alpha *= 2
        beta_val *= 2

    posterior_alpha = alpha + mae
    posterior_beta = beta_val + len(y_test) - mae

    weight = beta.mean(posterior_alpha, posterior_beta)

    model_weights.append(weight)

# Convert the list of individual predictions and weights into numpy arrays
individual_predictions = np.array(individual_predictions)
model_weights = np.array(model_weights)

# Normalize weights to sum to 1
model_weights /= model_weights.sum()

# Calculate ensemble predictions using weighted average
ensemble_predictions_bayesian = np.average(individual_predictions, axis=0, weig
```

```
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
```

```
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 3ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 2ms/step
222/222 [==============================] - 1s 3ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 0s 2ms/step
222/222 [==============================] - 1s 2ms/step
```

ASSIGNMENT AND OPTIMIZATION OF WEIGHTS FOR EACH ENSEMBLE METHOD

```
from scipy.optimize import minimize
from sklearn.metrics import mean_squared_error

# Assuming you have validation data (X_val, y_val)
```

```
# Define the objective function to minimize (in this case, mean squared error)
def objective_function(weights):
    ensemble_predictions_combined = (
        weights[0] * ensemble_predictions_boosted +
        weights[1] * ensemble_predictions_bayesian +
        weights[2] * ensemble_predictions_stacked +
        weights[3] * ensemble_predictions_dynamic

    )
    print(ensemble_predictions_combined)
    mse = mean_squared_error(y_test, ensemble_predictions_combined)
    print("Weights:", weights, "MSE:", mse)
    return mse


# Initialize weights
initial_weights = [1/4, 1/4, 1/4, 1/4]  # Starting with equal weights

# Define constraints (weights should sum to 1)
constraints = ({'type': 'eq', 'fun': lambda w: 1 - sum(w)})

# Define bounds for each weight (assuming weights are between 0 and 1)
bounds = [(0, 1), (0, 1), (0, 1), (0, 1)]

# Use the minimize function to find the optimal weights
result = minimize(objective_function, initial_weights, method='trust-constr', b
#For example, you can try 'L-BFGS-B' or 'trust-constr' or 'SLSQP' methods.

# Get the optimal weights from the result
optimal_weights = result.x

# Use the optimal weights to combine ensemble predictions
ensemble_predictions_optimal = (
    optimal_weights[0] * ensemble_predictions_boosted +
    optimal_weights[1] * ensemble_predictions_bayesian +
    optimal_weights[2] * ensemble_predictions_stacked +
    optimal_weights[3] * ensemble_predictions_dynamic

)
print(optimal_weights)
```

```
[[ 161.54487611]
 [ 146.79949946]
 [  98.27187101]
 ...
 [2377.12316479]
 [2360.36208372]
 [2369.32020604]]
Weights: [0.25 0.25 0.25 0.25] MSE: 111257.87721945273
[[ 161.54487635]
 [ 146.79949985]
 [  98.27187129]
```

```
 ...
 [2377.12318846]
 [2360.36210724]
 [2369.32022969]]
Weights: [0.25000001 0.25       0.25       0.25      ] MSE: 111257.88312081005
[[ 161.54488346]
 [ 146.79950582]
 [  98.27187529]
 ...
 [2377.12324121]
 [2360.36215887]
 [2369.32028247]]
Weights: [0.25       0.25000001 0.25       0.25      ] MSE: 111257.89709049143
[[ 161.54487793]
 [ 146.79950108]
 [  98.27187204]
 ...
 [2377.12318273]
 [2360.36210223]
 [2369.32022354]]
Weights: [0.25       0.25       0.25000001 0.25      ] MSE: 111257.88250875247
[[ 161.54487635]
 [ 146.79949985]
 [  98.27187129]
 ...
 [2377.12318846]
 [2360.36210724]
 [2369.32022969]]
Weights: [0.25       0.25       0.25       0.25000001] MSE: 111257.88312081005
[[  25.53967799]
 [  31.94208013]
 [  20.66748349]
 ...
 [1232.31365192]
 [1244.45077862]
 [1220.35686429]]
Weights: [ 0.34720256 -0.06078299  0.36637786  0.34720256] MSE: 17706.64073643
[[  25.53967822]
 [  31.94208052]
 [  20.66748377]
 ...
 [1232.31367559]
 [1244.45080214]
 [1220.35688794]]
Weights: [ 0.34720258 -0.06078299  0.36637786  0.34720256] MSE: 17706.63846350
[[  25.53967064]
```

```
# Assuming you have obtained ensemble_predictions_boosted, ensemble_predictions

# Assign weights to each ensemble prediction method
weight_boosted = optimal_weights[0]
weight_bayesian = optimal_weights[1]
weight_stacked = optimal_weights[2]
weight_dynamic = optimal_weights[3]
```

```
# Combine ensemble predictions using weighted average
ensemble_predictions_combined = (
    weight_boosted * ensemble_predictions_boosted +
    weight_bayesian * ensemble_predictions_bayesian +
    weight_stacked * ensemble_predictions_stacked +
    weight_dynamic * ensemble_predictions_dynamic)

# Normalize the combined ensemble predictions
ensemble_predictions_combined /= (
    weight_boosted + weight_bayesian + weight_stacked + weight_dynamic
)
```

POST-PROCESSING

```
# Assuming you have a time index for your testing data (adjust as needed)
time_index = test_data.index

# Plot individual predictions
plt.figure(figsize=(14, 7))
for i, symbol in enumerate(symbols):
    plt.plot(time_index, individual_predictions[i], label=f'{symbol} Predictior

plt.xlabel('Time')
plt.ylabel('Stock Price Prediction')
plt.title('Individual Model Predictions')
plt.legend()
plt.show()
```

```
# Plot ensemble predictions
plt.figure(figsize=(99, 50))
plt.plot(time_index, ensemble_predictions_combined, label='Ensemble Prediction'
plt.plot(test_data.index, y_test, label='Actual Closing Price',color = 'blue')
plt.xlabel('Time')
plt.ylabel('Stock Price Prediction')
plt.title('Ensemble Model Prediction')
plt.legend()
plt.show()
```