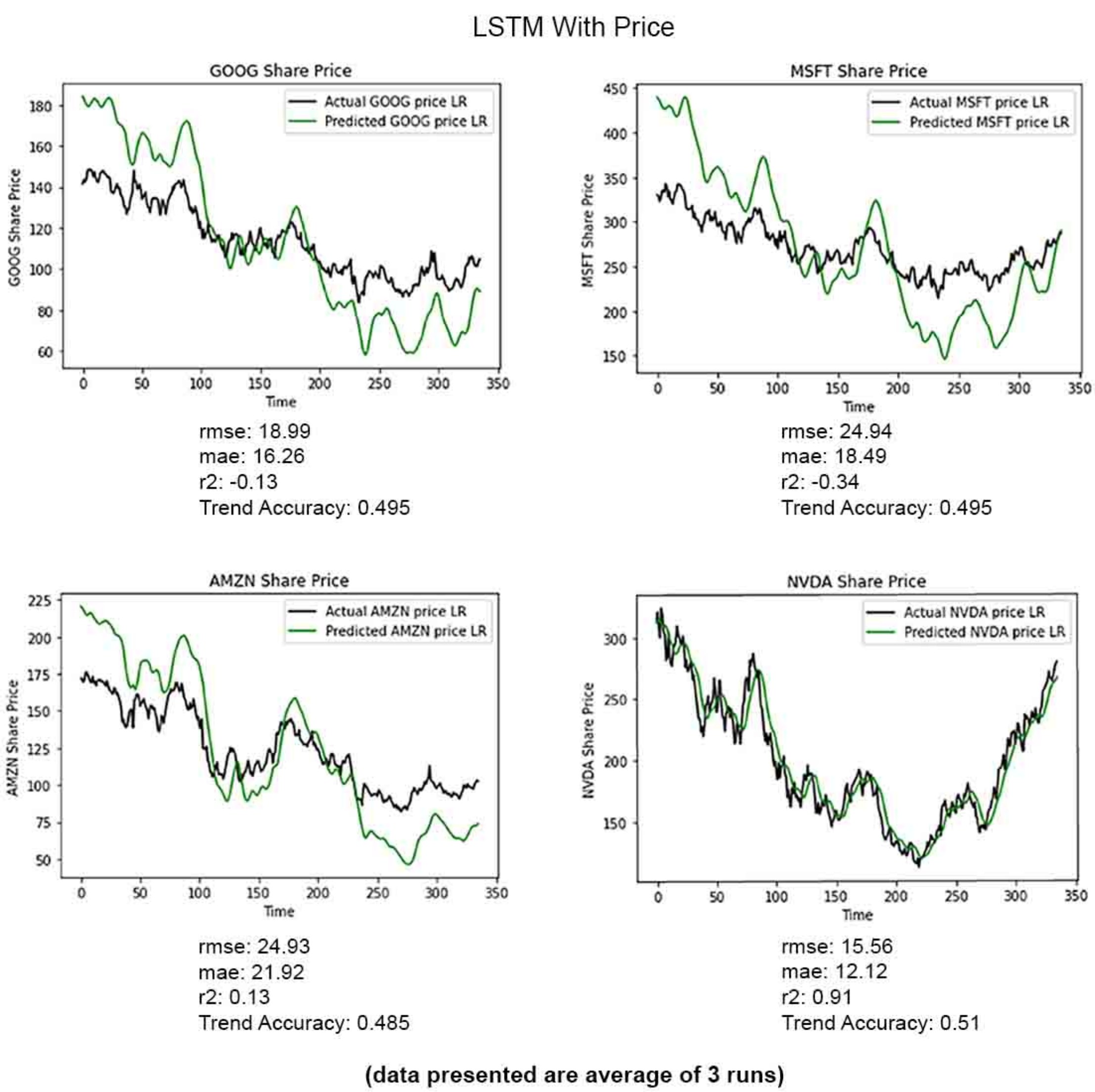
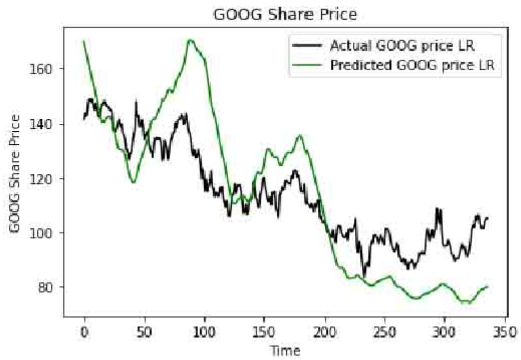
Appendix A

Prediction results

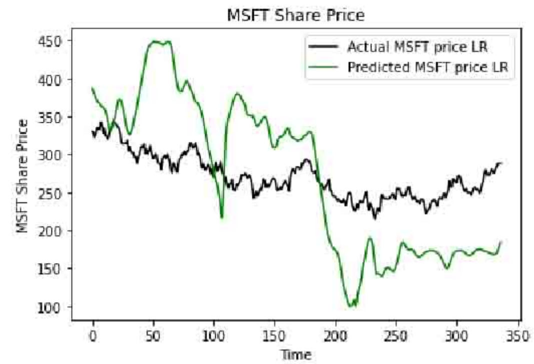
A.1 LSTM



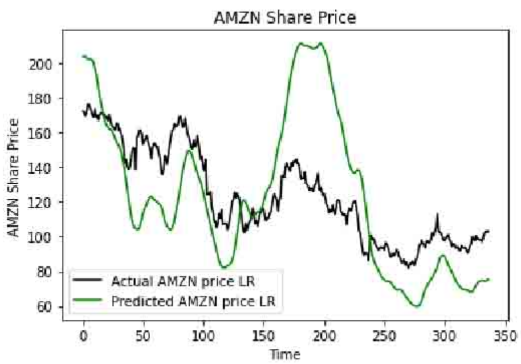
LSTM With Indicators



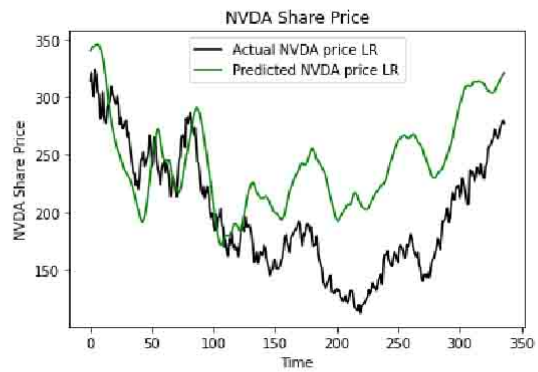
rmse: 17.23
mae: 14.62
r2: 0.07
Trend Accuracy: 0.623



rmse: 84.47
mae: 76.45
r2: -7.46
Trend Accuracy: 0.642



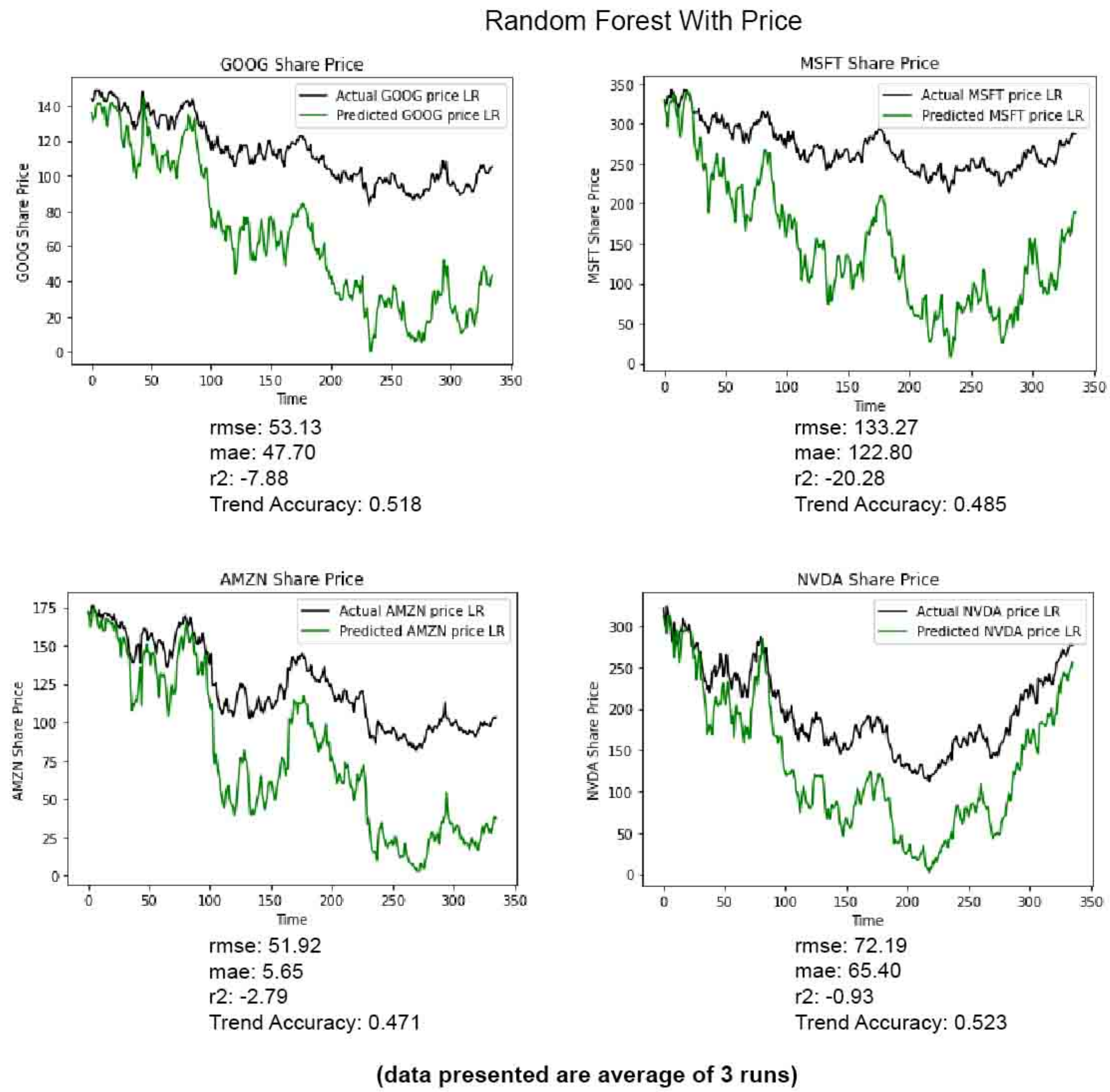
rmse: 35.41
mae: 28.86
r2: -0.75
Trend Accuracy: 0.652



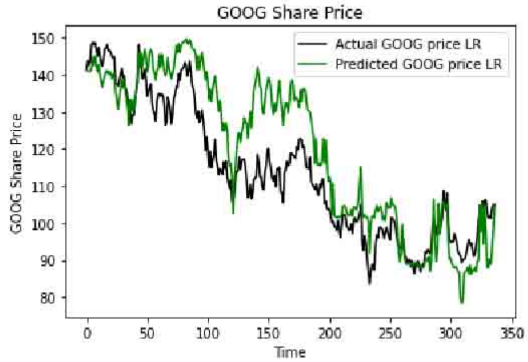
rmse: 54.50
mae: 47.92
r2: -0.09
Trend Accuracy: 0.629

(data presented are average of 3 runs)

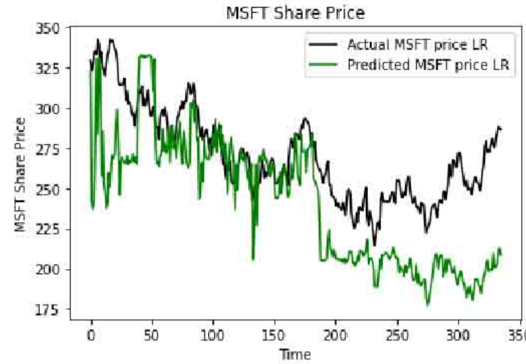
A.2 Random Forest



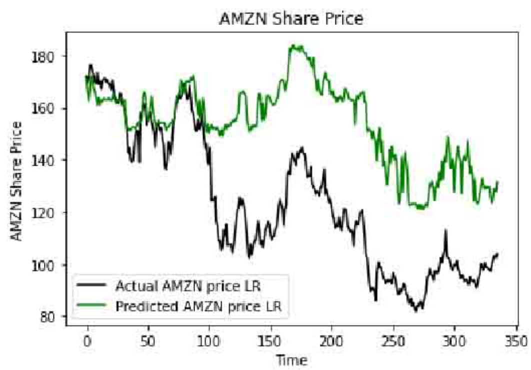
Random Forest With Indicators



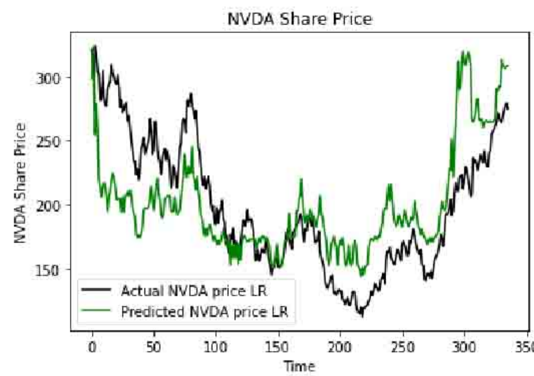
rmse: 12.53
mae: 10.31
r2: 0.58
Trend Accuracy: 0.518



rmse: 40.55
mae: 33.54
r2: -0.97
Trend Accuracy: 0.521



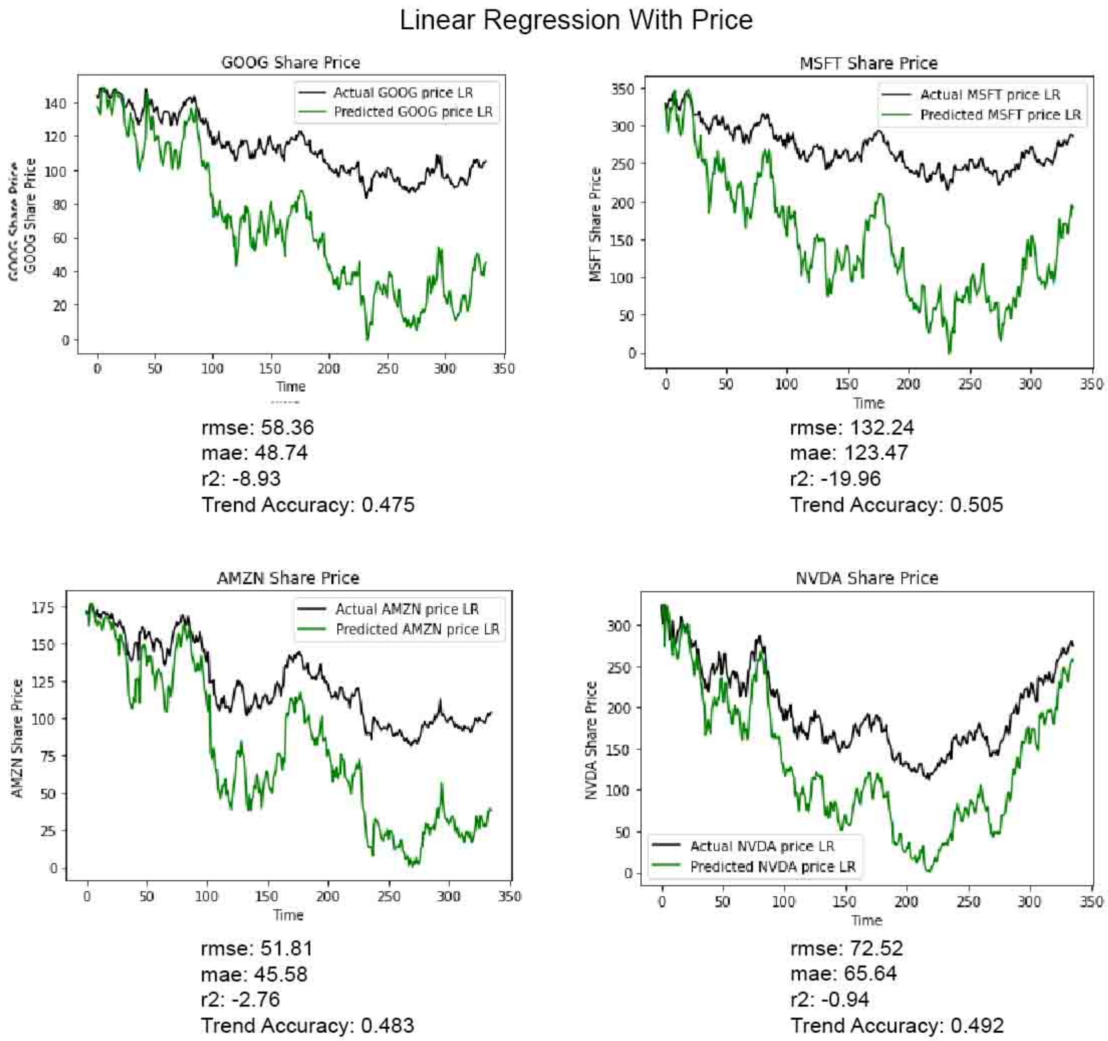
rmse: 36.06
mae: 31.20
r2: -0.83
Trend Accuracy: 0.506



rmse: 41.55
mae: 33.54
r2: -0.37
Trend Accuracy: 0.493

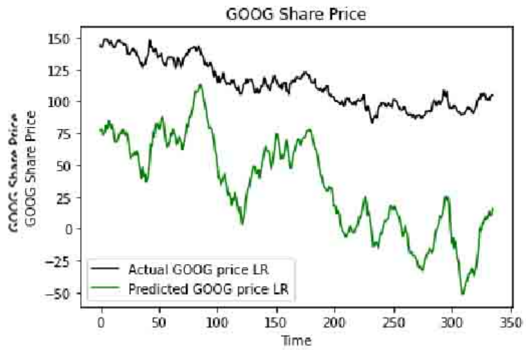
(data presented are average of 3 runs)

A.3 Linear Regression

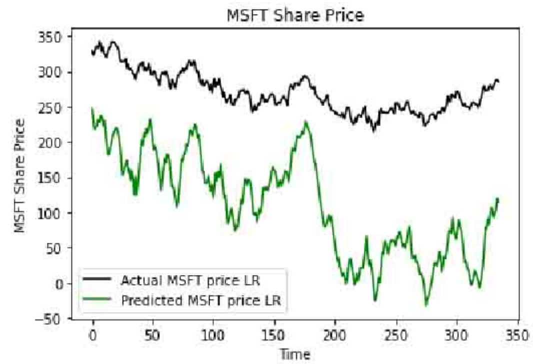


(data presented are average of 3 runs)

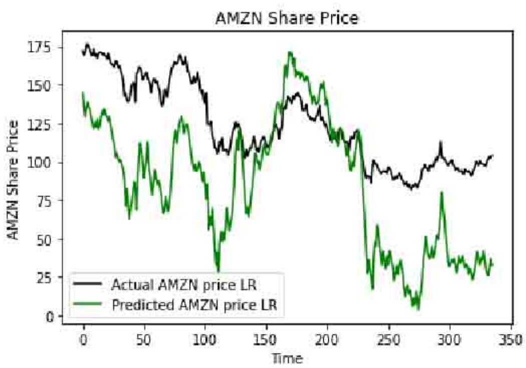
Linear Regression With Indicators



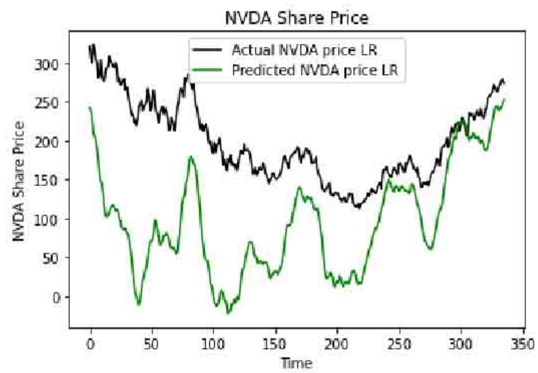
rmse: 83.10
mae: 79.45
r2: -20.72
Trend Accuracy: 0.482



rmse: 168.56
mae: 159.92
r2: -32.64
Trend Accuracy: 0.47



rmse: 49.14
mae: 43.41
r2: -2.41
Trend Accuracy: 0.47



rmse: 120.56
mae:103.42
r2: -4.36
Trend Accuracy: 0.497

(data presented are average of 3 runs)

Appendix B

User Guide

B.1 Introduction

The project is simple to set up, as consists of 6 Python files that run independently of each other.

B.2 Set Up

The version of Python needed to run these scripts are Python 3.9.7. The scripts are written using Spyder IDE 5.1.5, on Windows 10 OS, however, they should be able to be run in any other environments with the same Python version installed, or command line.

First, install Python 3.9.7 on the system if it's not yet installed.

There are a few Python packages that need to be installed before executing the files.

Navigate to the folder of the project in the command prompt or in the IDE.

```
C:\Users\ASUS>cd C:\Users\ASUS\Desktop\Stock Predict
```

Figure B.1:

install numpy 1.24.2 using:

pip install numpy

install pandas 1.3.4 using:

pip install pandas==1.3.4

install pandas-datareader 0.10.0 using:

pip install pandas-datareader

install matplotlib 3.7.1 using:

pip install matplotlib

install yfinance 0.2.4 using:

pip install yfinance==0.2.4

install tensorflow 2.12.0 using:

pip install tensorflow

install scikit-learn 1.2.2 using:

pip install scikit-learn

The files should now be executable using the command:
python (file name)

The output should then be produced, where the graph is displayed on a separate pop-up window, while the numerical output is displayed on the command prompt.
The graph pop-up window window may have to be closed for the model to continue to running the trend prediction.

All 6 of these files can be run separately, producing the results of their respective model.

Note: if using Spyder IDE, the above process can be replicated within Spyder, and the ”Run file” button on top can be clicked to run the code:

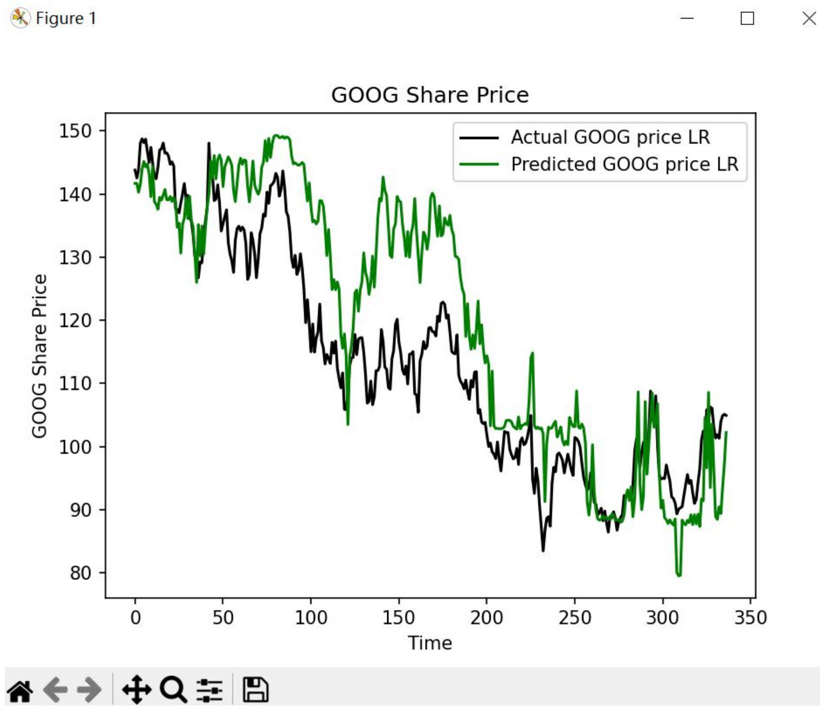


Figure B.2: graphical output

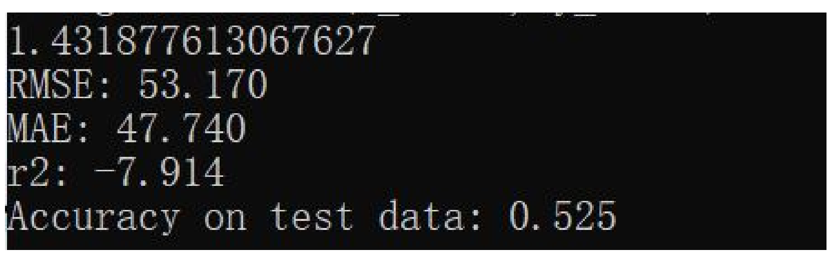


Figure B.3: numerical data outputted

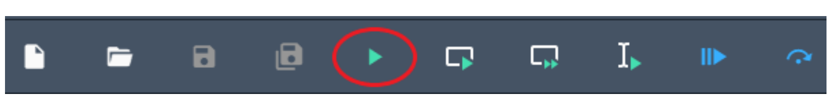


Figure B.4: Run file button

Appendix C

Source Code

C.1 Table of Content

1. LSTM_with_prices.py
2. LSTM_with_indicators.py
3. random_forest_with_prices.py
4. random_forest_with_indicators.py
5. linear_regression_with_prices.py
6. linear_regression_with_indicators.py

C.2 Statement

I verify that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary

Zebin Liao

2023/4/3

LSTM_with_prices.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 21 05:27:43 2023

@author: ASUS
"""

# -*- coding: utf-8 -*-
"""
Created on Thu Mar 16 00:22:01 2023

@author: ASUS
"""
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pandas_datareader as web
from pandas_datareader import data as pdr
import datetime as dt

import yfinance as yf
yf.pdr_override()

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDClassifier

from sklearn.preprocessing import MinMaxScaler

#load data
company = "AMZN"

start = dt.datetime (2012,1,1)
end = dt.datetime.now()

train_start = dt.datetime (2012,1,1)
train_end = dt.datetime(2021,12,1)

data = pdr.get_data_yahoo(company, start, end)

data['change_in_price'] = data['Close'].diff()

#print (data.index)

data.at['2012-01-03 00:00:00-05:00','change_in_price']=0

###
```

```
# Copy the `Close` column.
close_groups = data['Close']

# Apply the lambda function which will return -1.0 for down, 1.0 for up and 0.0 for no change.
close_groups = close_groups.transform(lambda x : np.sign(x.diff()))

# add column to data set
data['Prediction'] = close_groups

data.loc[data['Prediction'] == 0.0] = 1.0
data['Prediction'] = data['Prediction'].shift(-1)

###

data.at['2012-01-03 00:00:00-05:00','Prediction']= 1

data.fillna(0, inplace=True)

data_train = data.loc[:train_end,:]

x_train_data = data_train[['Open', 'High', 'Low', 'Volume']]
y_train_data = data_train['Close'].values.reshape(-1,1)

scaler_x = MinMaxScaler(feature_range = (0,1))
scaler_y = MinMaxScaler(feature_range = (0,1))
x_train_data = scaler_x.fit_transform(x_train_data[['Open', 'High', 'Low', 'Volume']])
scaled_data = scaler_y.fit_transform(data['Close'].values.reshape(-1,1))

prediction_days = 20

x_train = []
y_train = []

for x in range (prediction_days, len(x_train_data)):
    x_train.append(x_train_data[x - prediction_days: x])
    y_train.append(scaled_data[x , 0])

x_train = np.array(x_train)
y_train = np.array(y_train)

test_start = dt.datetime(2021,12,1)
test_end = dt.datetime.now()

data_test = data.loc[test_start:,:]

total_dataset = pd.concat((data_train[['Open', 'High', 'Low', 'Volume']], data_test[['Open', 'High', 'Low',
'Volume']]), axis = 0)

model_inputs = data[len(data) - len(data_test) - prediction_days:]

x_test_data = model_inputs[['Open', 'High', 'Low', 'Volume']]
y_test_data = model_inputs['Close'].values.reshape(-1,1)

x_test_data = scaler_x.fit_transform(x_test_data[['Open', 'High', 'Low', 'Volume']])

x_test = []
y_test = []

for x in range (prediction_days, len(x_test_data)):
    x_test.append(x_test_data[x - prediction_days: x])
    y_test.append(y_test_data[x , 0])

x_test = np.array(x_test)
y_test = np.array(y_test)

y_train = np.reshape(y_train, (-1, 1))
y_test = np.reshape(y_test, (-1, 1))

import time
```

```
start = time.time()

model = Sequential()

model.add(LSTM(units = 50, return_sequences = True, input_shape = (x_train.shape[1],4)))
model.add(Dropout(0.2))
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
model.add(LSTM(units = 50))
model.add(Dropout(0.2))
model.add(Dense(units = 1))

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, epochs = 40, batch_size = 20)

end = time.time()
print(f'Time Taken: {end-start:.3f}')

predicted_prices = model.predict(x_test)
predicted_prices = scaler_y.inverse_transform(predicted_prices)

actual_prices = data_test['Close'].values

difference_list = []

for i in range(len(predicted_prices)):
    difference_list.append(predicted_prices[i] - actual_prices[i])

difference = abs(sum(difference_list)/len(difference_list))

for i in range(len(predicted_prices)):
    predicted_prices[i] = predicted_prices[i] + difference

plt.plot(actual_prices, color = 'black', label = f"Actual {company} price LR")
plt.plot(predicted_prices, color = 'green', label = f"Predicted {company} price LR")
plt.title(f"{company} Share Price")
plt.xlabel('Time')
plt.ylabel(f"{company} Share Price")
plt.legend()
plt.show()

mse = mean_squared_error(actual_prices, predicted_prices)
mae = mean_absolute_error(actual_prices, predicted_prices)
r2 = r2_score(actual_prices, predicted_prices)

rmse = math.sqrt(mse)
print(f'RMSE: {rmse:.3f}')
print(f'MAE: {mae:.3f}')
print(f'r2: {r2:.3f}')

y_train_data = data_train['Prediction'].values.reshape(-1,1)
y_test_data = model_inputs['Prediction'].values.reshape(-1,1)
y_train = np.array(y_train_data)
y_test = np.array(y_test_data)

y_train = y_train[:-20]
y_test = y_test[:-20]

model.fit(x_train, y_train, batch_size=64, epochs=10, validation_split = 0.1, verbose = 1)
y_pred = model.predict(x_test)

y_pred = list(map(lambda x: -1 if x<0 else 1, y_pred))

accuracy = accuracy_score(y_test, y_pred, normalize=(True))
print(f'Accuracy on test data: {accuracy:.3f}')
```

LSTM_with_indicators.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 16 00:22:01 2023

@author: ASUS
"""
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pandas_datareader as web
from pandas_datareader import data as pdr
import datetime as dt

import yfinance as yf
yf.pdr_override()

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDClassifier

from sklearn.preprocessing import MinMaxScaler

#load data
company = "NVDA"

start = dt.datetime (2012,1,1)
end = dt.datetime.now()

train_start = dt.datetime (2012,1,1)
train_end = dt.datetime(2021,12,1)

data = pdr.get_data_yahoo(company, start, end)

data['change_in_price'] = data['Close'].diff()

data.at['2012-01-03 00:00:00-05:00','change_in_price']=0

###

n=14

up_df, down_df = data[['change_in_price']].copy(), data[['change_in_price']].copy()

# For up days, if the change is less than 0 set to 0.
up_df.loc['change_in_price'] = up_df.loc[(up_df['change_in_price'] < 0), 'change_in_price'] = 0

# For down days, if the change is greater than 0 set to 0.
down_df.loc['change_in_price'] = down_df.loc[(down_df['change_in_price'] > 0), 'change_in_price'] = 0
```

```
    down_df['change_in_price'] = down_df['change_in_price'].abs()

    # Calculate the Exponential Weighted Moving Average
    ewma_up = up_df['change_in_price'].transform(lambda x: x.ewm(span = n).mean())
    ewma_down = down_df['change_in_price'].transform(lambda x: x.ewm(span = n).mean())

    relative_strength = ewma_up / ewma_down

    relative_strength_index = 100.0 - (100.0 / (1.0 + relative_strength))

    data['down_days'] = down_df['change_in_price']
    data['up_days'] = up_df['change_in_price']
    data['RSI'] = relative_strength_index

    data.at['2012-01-04 00:00:00-05:00','RSI']=25.3

    ###

    ###

    low_14, high_14 = data[['Low']].copy(), data[['High']].copy()

    low_14 = low_14['Low'].transform(lambda x: x.rolling(window = n).min())
    high_14 = high_14['High'].transform(lambda x: x.rolling(window = n).max())

    # Calculate the Stochastic Oscillator.
    k_percent = 100 * ((data['Close'] - low_14) / (high_14 - low_14))

    # Add the info to the data frame.
    data['low_14'] = low_14
    data['high_14'] = high_14
    data['k_percent'] = k_percent

    r_percent = ((high_14 - data['Close']) / (high_14 - low_14)) * - 100
    data['r_percent'] = r_percent

    ###

    ###

    # Calculate the MACD
    ema_26 = data['Close'].transform(lambda x: x.ewm(span = 26).mean())
    ema_12 = data['Close'].transform(lambda x: x.ewm(span = 12).mean())
    macd = ema_12 - ema_26

    # Calculate the EMA
    ema_9_macd = macd.ewm(span = 9).mean()

    # Store the data in the data frame.
    data['MACD'] = macd
    data['MACD_EMA'] = ema_9_macd

    ###

    ###

    # Calculate the Price Rate of Change
    n = 9

    # Calculate the Rate of Change in the Price, and store it in the Data Frame.
    data['Price_Rate_Of_Change'] = data['Close'].transform(lambda x: x.pct_change(periods = n))

    ###

    ###

    def obv(group):
```

```
    change = data['Close'].diff()
    volume = data['Volume']

    # intialize the previous OBV
    prev_obv = 0
    obv_values = []

    # calculate the On Balance Volume
    for i, j in zip(change, volume):

        if i > 0:
            current_obv = prev_obv + j
        elif i < 0:
            current_obv = prev_obv - j
        else:
            current_obv = prev_obv

        prev_obv = current_obv
        obv_values.append(current_obv)

    # Return a panda series.
    return pd.Series(obv_values, index = data.index)


# apply the function
obv_groups = data.apply(obv)['Open']

# add to data
data['On Balance Volume'] = obv_groups

###

###

close_groups = data['Close']

close_groups = close_groups.transform(lambda x : np.sign(x.diff()))

# add the data to the main dataframe.
data['Prediction'] = close_groups

data.loc[data['Prediction'] == 0.0] = 1.0
data['Prediction'] = data['Prediction'].shift(-1)

###

data.at['2012-01-03 00:00:00-05:00','Prediction']= 1

data.fillna(0, inplace=True)

data_train = data.loc[:train_end,:]

x_train_data = data_train[['RSI','k_percent','r_percent','MACD','On Balance
Volume','Price_Rate_Of_Change']]
y_train_data = data_train['Close'].values.reshape(-1,1)

scaler_x = MinMaxScaler(feature_range = (0,1))
scaler_y = MinMaxScaler(feature_range = (0,1))
x_train_data = scaler_x.fit_transform(x_train_data[['RSI','k_percent','r_percent','MACD','On Balance
Volume','Price_Rate_Of_Change']])
scaled_data = scaler_y.fit_transform(data['Close'].values.reshape(-1,1))

prediction_days = 20

x_train = []
y_train = []

for x in range (prediction_days, len(x_train_data)):
```

```
        x_train.append(x_train_data[x - prediction_days: x])
        y_train.append(scaled_data[x , 0])

    x_train = np.array(x_train)
    y_train = np.array(y_train)

    test_start = dt.datetime(2021,12,1)
    test_end = dt.datetime.now()

    data_test = data.loc[test_start:,:]

    total_dataset = pd.concat((data_train[['RSI','k_percent','r_percent','MACD','On Balance
Volume','Price_Rate_Of_Change']], data_test[['RSI','k_percent','r_percent','MACD','On Balance
Volume','Price_Rate_Of_Change']]), axis = 0)

    model_inputs = data[len(data) - len(data_test) - prediction_days:]

    x_test_data = model_inputs[['RSI','k_percent','r_percent','MACD','On Balance
Volume','Price_Rate_Of_Change']]
    y_test_data = model_inputs['Close'].values.reshape(-1,1)

    x_test_data = scaler_x.fit_transform(x_test_data[['RSI','k_percent','r_percent','MACD','On Balance
Volume','Price_Rate_Of_Change']])

    x_test = []
    y_test = []

    for x in range (prediction_days, len(x_test_data)):
        x_test.append(x_test_data[x - prediction_days: x])
        y_test.append(y_test_data[x , 0])

    x_test = np.array(x_test)
    y_test = np.array(y_test)

    y_train = np.reshape(y_train, (-1, 1))
    y_test = np.reshape(y_test, (-1, 1))

    #time the execution time for the model
    import time
    start = time.time()

    #build the model
    model = Sequential()

    model.add(LSTM(units = 50, return_sequences = True, input_shape = (x_train.shape[1],6)))
    model.add(Dropout(0.2))
    model.add(LSTM(units = 50, return_sequences = True))
    model.add(Dropout(0.2))
    model.add(LSTM(units = 50))
    model.add(Dropout(0.2))
    model.add(Dense(units = 1))

    model.compile(optimizer='adam', loss='mean_squared_error')
    model.fit(x_train, y_train, epochs = 40, batch_size = 20)

    predicted_prices = model.predict(x_test)
    predicted_prices = scaler_y.inverse_transform(predicted_prices)

    end = time.time()
    print(f'Time Taken: {end-start:.3f}')

    actual_prices = data_test['Close'].values

    difference_list = []


    for i in range(len(predicted_prices)):
        difference_list.append(predicted_prices[i] - actual_prices[i])
```

```
difference = abs(sum(difference_list)/len(difference_list))

for i in range(len(predicted_prices)):
    predicted_prices[i] = predicted_prices[i] + difference

# plot the graph
plt.plot(actual_prices, color = 'black', label = f"Actual {company} price LR")
plt.plot(predicted_prices, color = 'green', label = f"Predicted {company} price LR")
plt.title(f"{company} Share Price")
plt.xlabel('Time')
plt.ylabel(f"{company} Share Price")
plt.legend()
plt.show()

#calculate the evaluation of the results
mse = mean_squared_error(actual_prices, predicted_prices)
mae = mean_absolute_error(actual_prices, predicted_prices)
r2 = r2_score(actual_prices, predicted_prices)

rmse = math.sqrt(mse)
print(f'RMSE: {rmse:.3f}')
print(f'MAE: {mae:.3f}')
print(f'r2: {r2:.3f}')

y_train_data = data_train['Prediction'].values.reshape(-1,1)
y_test_data = model_inputs['Prediction'].values.reshape(-1,1)
y_train = np.array(y_train_data)
y_test = np.array(y_test_data)

y_train = y_train[:-20]
y_test = y_test[:-20]

#make prediction for trend prediction
model.fit(x_train, y_train, batch_size=64, epochs=10, validation_split = 0.1, verbose = 1)
y_pred = model.predict(x_test)

y_pred = list(map(lambda x: -1 if x<0 else 1, y_pred))

accuracy = accuracy_score(y_test, y_pred, normalize=(True))
print(f'Trend prediction accuracy: {accuracy:.3f}')
```

~\Desktop\Stock Predict\random_forest_with_prices.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar 11 22:15:18 2023

@author: ASUS
"""
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pandas_datareader as web
from pandas_datareader import data as pdr
import datetime as dt

import yfinance as yf
yf.pdr_override()

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import SGDClassifier

#load data
company = "GOOG"

start = dt.datetime (2012,1,1)
end = dt.datetime.now()

train_start = dt.datetime (2012,1,1)
train_end = dt.datetime(2021,12,1)

data = pdr.get_data_yahoo(company, start, end)

scaler_x = MinMaxScaler(feature_range = (0,1))
scaler_y = MinMaxScaler(feature_range = (0,1))

# Group by the `Symbol` column, then grab the `Close` column.
close_groups = data['Close']

data['value_prediction'] = close_groups.shift(-1)

close_groups = close_groups.transform(lambda x : np.sign(x.diff()))

data['Prediction'] = close_groups

data.loc[data['Prediction'] == 0.0] = 1.0
data['Prediction'] = data['Prediction'].shift(-1)

data.fillna(0, inplace=True)

data_train = data.loc[:train_end,:]

x_train_data = data_train[['Open','High','Low','Volume']]
y_train_data = data_train ['value_prediction']
```

```
x_train_data = scaler_x.fit_transform(x_train_data)
y_train_data = scaler_y.fit_transform(y_train_data.values.reshape(-1,1))

x_train = []
y_train = []

x_train = np.array(x_train_data)
y_train = np.array(y_train_data)

test_start = dt.datetime(2021,12,1)
test_end = dt.datetime.now()

data_test = data.loc[test_start:,:]

x_test_data = data_test[['Open','High','Low','Volume']]
y_test_data = data_test ['value_prediction']

x_test_data = scaler_x.fit_transform(x_test_data)
y_test_data = scaler_y.fit_transform(y_test_data.values.reshape(-1,1))

total_dataset = pd.concat((data[['Open', 'High', 'Low', 'Volume']], data_test[['Open', 'High', 'Low',
'Volume']]), axis = 0)

x_test = []
y_test = []

x_test = np.array(x_test_data)
y_test = np.array(y_test_data)

x_train = np.reshape(x_train, (x_train.shape[0], -1))
x_test = np.reshape(x_test, (x_test.shape[0], -1))

y_test = y_test[:-1]

x_test = x_test[:-1]

import time

start = time.time()

regressor = RandomForestRegressor(n_estimators=150, random_state=35)
regressor.fit(x_train, y_train)
predicted = regressor.predict(x_test)

end = time.time()
print(end-start)

predicted = predicted.reshape(-1,1)
predicted = scaler_y.inverse_transform(predicted)

actual_prices = data_test['value_prediction']
actual_prices = actual_prices[:-1].values


plt.plot(actual_prices, color = 'black', label = f"Actual {company} price LR")
plt.plot(predicted, color = 'green', label = f"Predicted {company} price LR")
plt.title(f"{company} Share Price")
plt.xlabel('Time')
plt.ylabel(f"{company} Share Price")
plt.legend()
plt.show()

mse = mean_squared_error(actual_prices, predicted)

mae = mean_absolute_error(actual_prices, predicted)
r2 = r2_score(actual_prices, predicted)

rmse = math.sqrt(mse)
```

```
print(f'RMSE: {rmse:.3f}')
print(f'MAE: {mae:.3f}')
print(f'r2: {r2:.3f}')


classifier = RandomForestClassifier(n_estimators=150, random_state=50)

y_train_data = data_train['Prediction']
y_test_data = data_test['Prediction']
y_train = np.array(y_train_data)
y_test = np.array(y_test_data)

classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)

x_train_data = np.array(x_train_data)

y_test_data = y_test_data[:-1]

accuracy = accuracy_score(y_test_data, y_pred, normalize=(True))
print(f'Accuracy on test data: {accuracy:.3f}')
```

random_forest_with_indicators.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 16 00:22:01 2023

@author: ASUS
"""
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pandas_datareader as web
from pandas_datareader import data as pdr
import datetime as dt

import yfinance as yf
yf.pdr_override()

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score

#load data
company = "GOOG"

start = dt.datetime (2012,1,1)
end = dt.datetime.now()

train_start = dt.datetime (2012,1,1)
train_end = dt.datetime(2021,12,1)

data = pdr.get_data_yahoo(company, start, end)

data = pdr.get_data_yahoo(company, start, end)

data['change_in_price'] = data['Close'].diff()

data.at['2012-01-03 00:00:00-05:00','change_in_price']=0

###

n=14

up_df, down_df = data[['change_in_price']].copy(), data[['change_in_price']].copy()

# For up days, if the change is less than 0 set to 0.
up_df.loc['change_in_price'] = up_df.loc[(up_df['change_in_price'] < 0), 'change_in_price'] = 0

# For down days, if the change is greater than 0 set to 0.
down_df.loc['change_in_price'] = down_df.loc[(down_df['change_in_price'] > 0), 'change_in_price'] = 0

down_df['change_in_price'] = down_df['change_in_price'].abs()

# Calculate the Exponential Weighted Moving Average
ewma_up = up_df['change_in_price'].transform(lambda x: x.ewm(span = n).mean())
ewma_down = down_df['change_in_price'].transform(lambda x: x.ewm(span = n).mean())
```

```
relative_strength = ewma_up / ewma_down

relative_strength_index = 100.0 - (100.0 / (1.0 + relative_strength))

data['down_days'] = down_df['change_in_price']
data['up_days'] = up_df['change_in_price']
data['RSI'] = relative_strength_index

data.at['2012-01-04 00:00:00-05:00','RSI']=25.3

###

###

low_14, high_14 = data[['Low']].copy(), data[['High']].copy()

low_14 = low_14['Low'].transform(lambda x: x.rolling(window = n).min())
high_14 = high_14['High'].transform(lambda x: x.rolling(window = n).max())

# Calculate the Stochastic Oscillator.
k_percent = 100 * ((data['Close'] - low_14) / (high_14 - low_14))

# Add the info to the data frame.
data['low_14'] = low_14
data['high_14'] = high_14
data['k_percent'] = k_percent

r_percent = ((high_14 - data['Close']) / (high_14 - low_14)) * - 100
data['r_percent'] = r_percent

###

###

# Calculate the MACD
ema_26 = data['Close'].transform(lambda x: x.ewm(span = 26).mean())
ema_12 = data['Close'].transform(lambda x: x.ewm(span = 12).mean())
macd = ema_12 - ema_26

# Calculate the EMA
ema_9_macd = macd.ewm(span = 9).mean()

# Store the data in the data frame.
data['MACD'] = macd
data['MACD_EMA'] = ema_9_macd

###

###

# Calculate the Price Rate of Change
n = 9

data['Price_Rate_Of_Change'] = data['Close'].transform(lambda x: x.pct_change(periods = n))

###

###

def obv(group):

    change = data['Close'].diff()
    volume = data['Volume']

    # intialize the previous OBV
    prev_obv = 0
    obv_values = []
```

```
    # calculate the On Balance Volume
    for i, j in zip(change, volume):

        if i > 0:
            current_obv = prev_obv + j
        elif i < 0:
            current_obv = prev_obv - j
        else:
            current_obv = prev_obv

        prev_obv = current_obv
        obv_values.append(current_obv)

    # Return a panda series.
    return pd.Series(obv_values, index = data.index)


# apply the function
obv_groups = data.apply(obv)['Open']

# add to data
data['On Balance Volume'] = obv_groups

###

###

# Group by the `Symbol` column, then grab the `Close` column.
close_groups = data['Close']

data['value_prediction'] = close_groups.shift(-1)

# Apply the lambda function which will return -1.0 for down, 1.0 for up and 0.0 for no change.
close_groups = close_groups.transform(lambda x : np.sign(x.diff()))

# add the data to the main dataframe.
data['Prediction'] = close_groups

# for simplicity in later sections I'm going to make a change to our prediction column. To keep this as a
binary classifier I'll change flat days and consider them up days.
data.loc[data['Prediction'] == 0.0] = 1.0
data['Prediction'] = data['Prediction'].shift(-1)


###

data.at['2012-01-03 00:00:00-05:00','Prediction']= 1

data.fillna(0, inplace=True)

data_train = data.loc[:train_end,:]

x_train_data = data_train[['RSI','k_percent','r_percent','MACD','On Balance
Volume','Price_Rate_Of_Change']]
y_train_data = data_train['value_prediction']

x_train = []
y_train = []

x_train = np.array(x_train_data)
y_train = np.array(y_train_data)

test_start = dt.datetime(2021,12,1)
test_end = dt.datetime.now()

data_test = data.loc[test_start:,:]

total_dataset = pd.concat((data_train[['RSI','k_percent','r_percent','MACD','On Balance
Volume','Price_Rate_Of_Change']], data_test[['RSI','k_percent','r_percent','MACD','On Balance
```

```
Volume','Price_Rate_Of_Change']]), axis = 0)

model_inputs = total_dataset[len(total_dataset) - len(data_test):].values

x_test = []
y_test = []

x_test_data = data_test[['RSI','k_percent','r_percent','MACD','On Balance Volume','Price_Rate_Of_Change']]
y_test_data = data_test['value_prediction']

x_test = np.array(x_test_data)
y_test = np.array(y_test_data)

x_train = np.reshape(x_train, (x_train.shape[0], -1))
x_test = np.reshape(x_test, (x_test.shape[0], -1))

y_test = y_test[:-1]

x_test = x_test[:-1]

import time
start = time.time()

regressor = RandomForestRegressor(n_estimators=150, random_state=30)
regressor.fit(x_train, y_train)
predicted = regressor.predict(x_test)

end = time.time()
print(f'Time Taken: {end-start:.3f}')

actual_prices = data_test['value_prediction']
actual_prices = actual_prices[:-1].values

plt.plot(actual_prices, color = 'black', label = f"Actual {company} price LR")
plt.plot(predicted, color = 'green', label = f"Predicted {company} price LR")
plt.title(f"{company} Share Price")
plt.xlabel('Time')
plt.ylabel(f"{company} Share Price")
plt.legend()
plt.show()

mse = mean_squared_error(actual_prices, predicted)
mae = mean_absolute_error(actual_prices, predicted)
r2 = r2_score(actual_prices, predicted)

rmse = math.sqrt(mse)
print(f'RMSE: {rmse:.3f}')
print(f'MAE: {mae:.3f}')
print(f'r2: {r2:.3f}')

classifier = RandomForestClassifier(n_estimators=150, random_state=45)

y_train_data = data_train['Prediction']
y_test_data = data_test['Prediction']
y_train = np.array(y_train_data)
y_test = np.array(y_test_data)

classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)

x_train_data = np.array(x_train_data)

y_test_data = y_test_data[:-1]

accuracy = accuracy_score(y_test_data, y_pred, normalize=(True))
print(f'Accuracy on test data: {accuracy:.3f}')
```

linear_regression_with_prices.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar 11 22:15:18 2023

@author: ASUS
"""
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pandas_datareader as web
from pandas_datareader import data as pdr
import datetime as dt

import yfinance as yf
yf.pdr_override()

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import SGDClassifier

#load data
company = "NVDA"

start = dt.datetime (2012,1,1)
end = dt.datetime.now()

train_start = dt.datetime (2012,1,1)
train_end = dt.datetime(2021,12,1)

data = pdr.get_data_yahoo(company, start, end)

scaler_x = MinMaxScaler(feature_range = (0,1))
scaler_y = MinMaxScaler(feature_range = (0,1))

# Group by the `Symbol` column, then grab the `Close` column.
close_groups = data['Close']

data['value_prediction'] = close_groups.shift(-1)

# Apply the lambda function which will return -1.0 for down, 1.0 for up and 0.0 for no change.
close_groups = close_groups.transform(lambda x : np.sign(x.diff()))

# add the data to the main dataframe.
data['Prediction'] = close_groups

data.loc[data['Prediction'] == 0.0] = 1.0
data['Prediction'] = data['Prediction'].shift(-1)

data.fillna(0, inplace=True)

data_train = data.loc[:train_end,:]

x_train_data = data_train[['Open','High','Low','Volume']]
y_train_data = data_train ['value_prediction']
```

```
x_train_data = scaler_x.fit_transform(x_train_data)
y_train_data = scaler_y.fit_transform(y_train_data.values.reshape(-1,1))

x_train = []
y_train = []

x_train = np.array(x_train_data)
y_train = np.array(y_train_data)

test_start = dt.datetime(2021,12,1)
test_end = dt.datetime.now()

data_test = data.loc[test_start:,:]

x_test_data = data_test[['Open','High','Low','Volume']]
y_test_data = data_test ['value_prediction']

x_test_data = scaler_x.fit_transform(x_test_data)
y_test_data = scaler_y.fit_transform(y_test_data.values.reshape(-1,1))

total_dataset = pd.concat((data[['Open', 'High', 'Low', 'Volume']], data_test[['Open', 'High', 'Low',
'Volume']]), axis = 0)

x_test = []
y_test = []

x_test = np.array(x_test_data)
y_test = np.array(y_test_data)

x_train = np.reshape(x_train, (x_train.shape[0], -1))
x_test = np.reshape(x_test, (x_test.shape[0], -1))

y_test = y_test[:-1]

x_test = x_test[:-1]

import time
start = time.time()

regressor = LinearRegression()
regressor.fit(x_train, y_train)
predicted = regressor.predict(x_test)

end = time.time()
print(f'Time Taken: {end-start:.3f}')

predicted = scaler_y.inverse_transform(predicted)

actual_prices = data_test['value_prediction']
actual_prices = actual_prices[:-1].values


plt.plot(actual_prices, color = 'black', label = f"Actual {company} price LR")
plt.plot(predicted, color = 'green', label = f"Predicted {company} price LR")
plt.title(f"{company} Share Price")
plt.xlabel('Time')
plt.ylabel(f"{company} Share Price")
plt.legend()
plt.show()

mse = mean_squared_error(actual_prices, predicted)
mae = mean_absolute_error(actual_prices, predicted)
r2 = r2_score(actual_prices, predicted)

rmse = math.sqrt(mse)
print(f'RMSE: {rmse:.3f}')
print(f'MAE: {mae:.3f}')
print(f'r2: {r2:.3f}')
```

```
classifier = SGDClassifier()

y_train_data = data_train['Prediction']
y_test_data = data_test['Prediction']
y_train = np.array(y_train_data)
y_test = np.array(y_test_data)

classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)

x_train_data = np.array(x_train_data)

y_test_data = y_test_data[:-1]

accuracy = accuracy_score(y_test_data, y_pred, normalize=(True))
print(f'Accuracy on test data: {accuracy:.3f}')
```

linear_regression_with_indicators.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar 25 12:11:50 2023

@author: ASUS
"""
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pandas_datareader as web
from pandas_datareader import data as pdr
import datetime as dt

import yfinance as yf
yf.pdr_override()

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import SGDClassifier

#load data
company = "NVDA"

start = dt.datetime (2012,1,1)
end = dt.datetime.now()

train_start = dt.datetime (2012,1,1)
train_end = dt.datetime(2021,12,1)

data = pdr.get_data_yahoo(company, start, end)

data['change_in_price'] = data['Close'].diff()

data.at['2012-01-03 00:00:00-05:00','change_in_price']=0

###

n=14

up_df, down_df = data[['change_in_price']].copy(), data[['change_in_price']].copy()

# For up days, if the change is less than 0 set to 0.
up_df.loc['change_in_price'] = up_df.loc[(up_df['change_in_price'] < 0), 'change_in_price'] = 0

# For down days, if the change is greater than 0 set to 0.
down_df.loc['change_in_price'] = down_df.loc[(down_df['change_in_price'] > 0), 'change_in_price'] = 0

down_df['change_in_price'] = down_df['change_in_price'].abs()

# Calculate the Exponential Weighted Moving Average
ewma_up = up_df['change_in_price'].transform(lambda x: x.ewm(span = n).mean())
ewma_down = down_df['change_in_price'].transform(lambda x: x.ewm(span = n).mean())

relative_strength = ewma_up / ewma_down
```

```
relative_strength_index = 100.0 - (100.0 / (1.0 + relative_strength))

data['down_days'] = down_df['change_in_price']
data['up_days'] = up_df['change_in_price']
data['RSI'] = relative_strength_index

data.at['2012-01-04 00:00:00-05:00','RSI']=25.3

###

###

low_14, high_14 = data[['Low']].copy(), data[['High']].copy()

low_14 = low_14['Low'].transform(lambda x: x.rolling(window = n).min())
high_14 = high_14['High'].transform(lambda x: x.rolling(window = n).max())

# Calculate the Stochastic Oscillator.
k_percent = 100 * ((data['Close'] - low_14) / (high_14 - low_14))

# Add the info to the data frame.
data['low_14'] = low_14
data['high_14'] = high_14
data['k_percent'] = k_percent

r_percent = ((high_14 - data['Close']) / (high_14 - low_14)) * - 100
data['r_percent'] = r_percent

###

###

# Calculate the MACD
ema_26 = data['Close'].transform(lambda x: x.ewm(span = 26).mean())
ema_12 = data['Close'].transform(lambda x: x.ewm(span = 12).mean())
macd = ema_12 - ema_26

# Calculate the EMA
ema_9_macd = macd.ewm(span = 9).mean()

# Store the data in the data frame.
data['MACD'] = macd
data['MACD_EMA'] = ema_9_macd

###

###

# Calculate the Price Rate of Change
n = 9

# Calculate the Rate of Change in the Price, and store it in the Data Frame.
data['Price_Rate_Of_Change'] = data['Close'].transform(lambda x: x.pct_change(periods = n))

###

###

def obv(group):

    change = data['Close'].diff()
    volume = data['Volume']

    # intialize the previous OBV
    prev_obv = 0
    obv_values = []

    # calculate the On Balance Volume
```

```
    for i, j in zip(change, volume):

        if i > 0:
            current_obv = prev_obv + j
        elif i < 0:
            current_obv = prev_obv - j
        else:
            current_obv = prev_obv

        prev_obv = current_obv
        obv_values.append(current_obv)

    # Return a panda series.
    return pd.Series(obv_values, index = data.index)


# apply the function
obv_groups = data.apply(obv)['Open']

# add to data
data['On Balance Volume'] = obv_groups

###

###

close_groups = data['Close']

# Apply the lambda function which will return -1.0 for down, 1.0 for up and 0.0 for no change.
close_groups = close_groups.transform(lambda x : np.sign(x.diff()))

# add the data to the main dataframe.
data['Prediction'] = close_groups

data.loc[data['Prediction'] == 0.0] = 1.0
data['Prediction'] = data['Prediction'].shift(-1)

###

data.at['2012-01-03 00:00:00-05:00','Prediction']= 1

data.fillna(0, inplace=True)

scaler_x = MinMaxScaler(feature_range = (0,1))
scaler_y = MinMaxScaler(feature_range = (0,1))

# Group by the `Symbol` column, then grab the `Close` column.
close_groups = data['Close']

data['value_prediction'] = close_groups.shift(-1)

# Apply the lambda function which will return -1.0 for down, 1.0 for up and 0.0 for no change.
close_groups = close_groups.transform(lambda x : np.sign(x.diff()))

data.fillna(0, inplace=True)

data_train = data.loc[:train_end,:]

x_train_data = data_train[['RSI','k_percent','r_percent','MACD','On Balance
Volume','Price_Rate_Of_Change']]
y_train_data = data_train ['value_prediction']

x_train_data = scaler_x.fit_transform(x_train_data)
y_train_data = scaler_y.fit_transform(y_train_data.values.reshape(-1,1))

x_train = []
y_train = []

x_train = np.array(x_train_data)
```

```
y_train = np.array(y_train_data)

test_start = dt.datetime(2021,12,1)
test_end = dt.datetime.now()

data_test = data.loc[test_start:,:]

x_test_data = data_test[['RSI','k_percent','r_percent','MACD','On Balance Volume','Price_Rate_Of_Change']]
y_test_data = data_test ['value_prediction']

x_test_data = scaler_x.fit_transform(x_test_data)
y_test_data = scaler_y.fit_transform(y_test_data.values.reshape(-1,1))

total_dataset = pd.concat((data[['RSI','k_percent','r_percent','MACD','On Balance
Volume','Price_Rate_Of_Change']], data_test[['RSI','k_percent','r_percent','MACD','On Balance
Volume','Price_Rate_Of_Change']]), axis = 0)

x_test = []
y_test = []

x_test = np.array(x_test_data)
y_test = np.array(y_test_data)

x_train = np.reshape(x_train, (x_train.shape[0], -1))
x_test = np.reshape(x_test, (x_test.shape[0], -1))

y_test = y_test[:-1]

x_test = x_test[:-1]

import time
start = time.time()

regressor = LinearRegression()
regressor.fit(x_train, y_train)
predicted = regressor.predict(x_test)

predicted = scaler_y.inverse_transform(predicted)

end = time.time()
print(f'Time Taken: {end-start:.3f}')

actual_prices = data_test['value_prediction']
actual_prices = actual_prices[:-1].values


plt.plot(actual_prices, color = 'black', label = f"Actual {company} price LR")
plt.plot(predicted, color = 'green', label = f"Predicted {company} price LR")
plt.title(f"{company} Share Price")
plt.xlabel('Time')
plt.ylabel(f"{company} Share Price")
plt.legend()
plt.show()

mse = mean_squared_error(actual_prices, predicted)
mae = mean_absolute_error(actual_prices, predicted)
r2 = r2_score(actual_prices, predicted)

rmse = math.sqrt(mse)
print(f'RMSE: {rmse:.3f}')
print(f'MAE: {mae:.3f}')
print(f'r2: {r2:.3f}')

classifier = SGDClassifier()

y_train_data = data_train['Prediction']
y_test_data = data_test['Prediction']
y_train = np.array(y_train_data)
y_test = np.array(y_test_data)
```

```
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)

x_train_data = np.array(x_train_data)

y_test_data = y_test_data[:-1]

accuracy = accuracy_score(y_test_data, y_pred, normalize=(True))
print(f'Accuracy on test data: {accuracy:.3f}')
```