Project Title: Technical Analysis for Stock Price Prediction
https://github.com/anjanshrestha123/Technical-Analysis-For-Stock-Price-Prediction

Participants:

- Anjan Shrestha (anjanshrestha@my.unt.edu)
- Krishna Sathvik Mantripragada (KrishnaSathvikMantripragada@my.unt.edu)
- Sai Tarun Gunda (saitarungunda@my.unt.edu)
- Tharun Digajari (tharundigajari@my.unt.edu)

Abstract:

As we know the stock market is very dynamic and volatile, it's extremely hard and challenging to make accurate predictions. There are a lot of factors that impact stock prices such as news, events, financial performance, people's sentiment and so on. Basically, analysis of stock has been divided into three parts i.e., Fundamental Analysis, Technical Analysis and Sentimental Analysis. In this project, we will be focusing on technical analysis to make stock price predictions.

Technical Analysis uses historical stock prices, returns and volume of trades to perform the prediction. It basically captures the pattern of stock market movement and finds different trading signals out of it. This kind of analysis is mainly used in short term trading that can be daily, weekly, or monthly which can give high returns in a short amount of time if we are able to predict it properly.

Our main objective is to extract the precious stock prices by using different python modules, to train and test our model out of those data and to predict the price of various stocks. In other words, our model should be able to provide the future price of different stocks based on technical analysis.

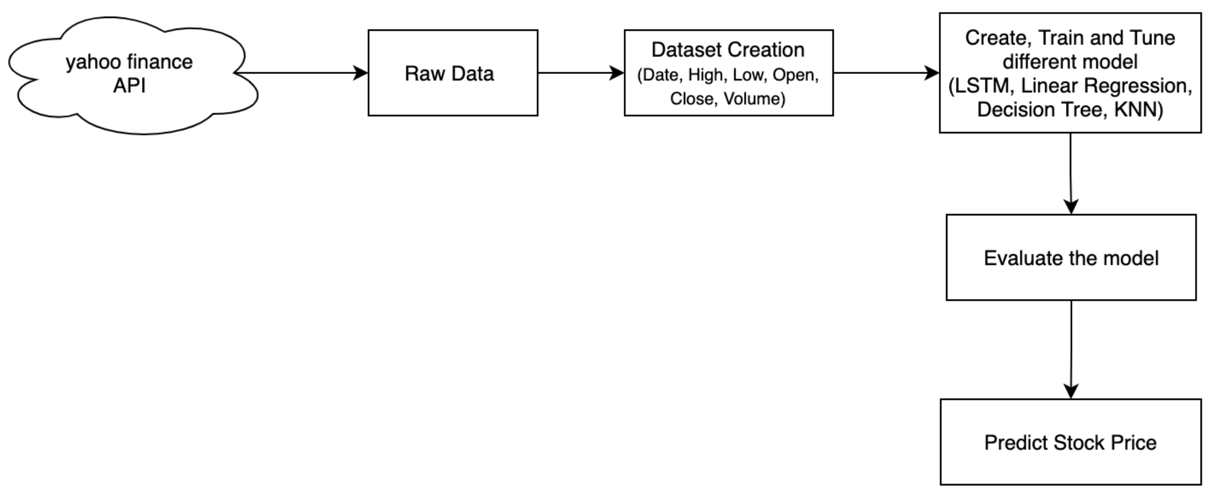
ML Problem specification:

Predicting stock price is a supervised ML problem where dataset is a time-series data that is recorded over consistent intervals of time. Data was extracted from yahoo finance API using pandas_datareader module that directly converts response from the API to pandas dataframe. The response consists of different columns such as Date, High, Low, Open, Close and Volume.

	Date	High	Low	Open	Close	Volume
0	2015-01-02	27.860001	26.837500	27.847500	27.332500	212818400.0
1	2015-01-05	27.162500	26.352501	27.072500	26.562500	257142000.0
2	2015-01-06	26.857500	26.157499	26.635000	26.565001	263188400.0
3	2015-01-07	27.049999	26.674999	26.799999	26.937500	160423600.0
4	2015-01-08	28.037500	27.174999	27.307501	27.972500	237458000.0

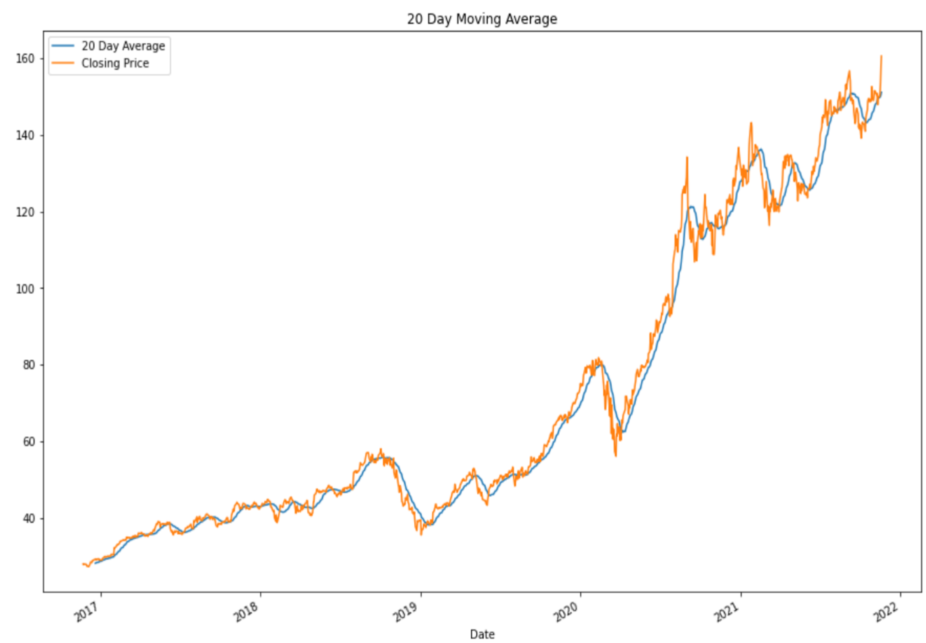
Among different columns, only Date and Closing price are used as to extract features to run our model. As a part of feature engineering, we initially selected 50 days of closing price which is later tuned to predict the next day price. So, after feature engineering, our features contain 50 columns of previous day's closing price and target would be the next day closing price. After getting the features, we used 70% of data to train machine learning model and remaining 30% to test it.

Design:



Stock closing price had been collected from yahoo finance API along with its date with the help of datareader module and then features had been created using last 50 days of time-steps initially. Different models had been used, trained, and tuned such as LSTM, Linear Regression, Decision Tree Regression and KNN Regression. Finally, after hyperparameter tuning, the model with the best root mean square error (RMSE score) had been used to predict both the stock trend for next 30 days as well as stock closing price for next day.

In this project, we used the python programming language since it has lots of modules that have support for machine learning. Some of the examples of those modules were sklearn, matplotlib, pandas, pandas dataframe, numpy, yahoo finance and so on. Also, we wrote the code in a jupyter lab.

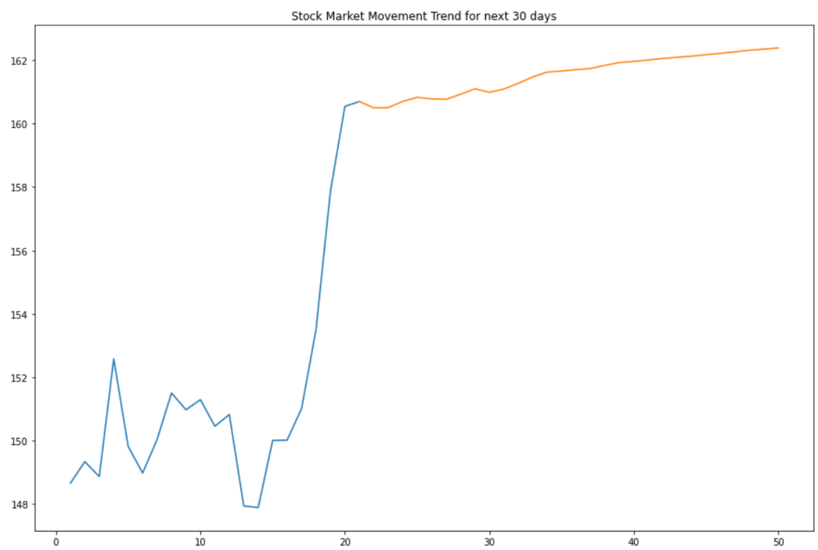


Exploratory Data analysis had been performed before running the model to know more about data. For this analysis, we have created a 20 Day Moving Average for stock closing price vs date on Apple company. From this graph, we see the stock is moving in an upward trend, so there is high probability for this stock to increase in long run.

	Model	Default Hyperparameters	Best Hyperparameters	Stock Price Data (No. of Years)	Time Series Step (No. of Days)	RMSE
0	LSTM	{'epochs': 1, 'batch_size': None}	{'epochs': 25, 'batch_size': 30}	15	20	84.960178
1	Linear Regression	{}	{}	15	20	1.663047
2	Decision Tree Regression	{'min_samples_split': 2}	{'min_samples_split': 7}	5	20	50.896809
3	KNN Regression	{'n_neighbors': 5}	{'n_neighbors': 5}	5	20	52.029079

Hyperparameter tuning were performed on different four models to find the best hyperparameters. These parameters were epochs and batch size for LSTM, neighbors for KNN and min samples split for Decision Tree. Since there are no hyperparameter for Linear

Regression, there were no need to do the tuning. Also, we selected different number of years to fetch the stock price for dataset creation and different time series interval to create feature out of it and ran the model to see which one gives more accurate prediction. As shown in the figure above, after performing the tuning, the best model was found to be Linear Regression with 15 years of stock price data and 20 days of time series step.



After getting the best model i.e., Linear Regression, stock movement trend was prediction for next 30 days as shown in the figure above.

8. Predict Stock Closing Price for Next Day Using Best Model (Linear Regression)

```
# Current Date of Model Run
print('Current Date:                ', dt.date.today())
print('Last Stock Closing Price:    ', best_test_data_lr[-1])

# Next day price using Linear Regression
last_n_days_data = best_test_data_lr[len(best_test_data_lr) - best_no_of_days_lr:]
next_day = best_model_lr.predict([last_n_days_data])[0]

print('Next Day Stock Closing Price: ', next_day)
```

```
Current Date:                2021-11-21
Last Stock Closing Price:    160.5500030517578
Next Day Stock Closing Price: 160.702096334333
```

Also, using Linear Regression, stock closing price had been calculated for the next day which was found to be 160.70.

To sum all of it, we have fetched data from yahoo finance API, performed exploratory data analysis, created, and trained four different regression models, performed hyperparameter tuning, compared, and evaluated models and selected best model with best hyperparameter and finally calculated the stock movement for next 30 days as well as next day stock closing price using the best model. We have not used any ensemble method in this project. Also, we can improve this model by adding fundamental and sentimental features to the dataset and adding more hyperparameters to fine tune the models even better. Moreover, we can try with other machine learning models or use ensemble methods to get better prediction.

Milestones:

Throughout the five milestones, we utilized github as a version control that helped us to share and work in the central codebase.

Milestones and its incremental feature:

Milestone	Date	Incremental Feature
1	10/22/2021	Analysis, Research, Planning, Design and Raw data collection
2	11/05/2021	Dataset creation, Feature engineering and Model training
3	11/19/2021	Model validation and Testing (hyperparameter tuning and cross validation)
4	11/03/2021	Analyze and Improve model accuracy
5	11/20/2021	Monitor outputs from model

Code:

```
# For dealing with dataframe
import pandas as pd
```

```
# For dealing with np array
import numpy as np

# For calling yahoo finance to get stock price
import pandas_datareader as pdr
import datetime as dt
from datetime import timedelta

# For plotting
import matplotlib.pyplot as plt

# For model
import math
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor

# For upgrading pandas datareader module
!pip install --upgrade pandas_datareader

# Dataset Properties
DATE = 'Date'
CLOSE = 'Close'
VOLUME = 'Volume'

# Stock Properties
STOCK_TICKER = 'AAPL' # Stock ticker name to run the model

# Target
NUMBER_OF_DAYS_TO_PREDICT = 30 # Target Stock Trend

# Custom Hyperparameters
NUMBER_OF_YEARS_TO_FETCH_PRICE_DATA = [5, 15]
NUMBER_OF_DAYS_FOR_PRICE_PREDICTION = [20, 50]

# LSTM Hyperparameters
NUMBER_OF_EPOCH_LIST = [25, 50]
BATCH_SIZE_LIST = [30, 60]

# KNN Hyperparameters
KNN_NEIGHBORS_LIST = [1, 3, 5, 7, 9]

# Decision Tree Hyperparameters
MIN_SAMPLES_LIST = [2, 3, 5, 7, 9]

# Function to Create Dataset
def extract_raw_data(number_of_years_to_fetch_price_data):
  # Getting start and end date for stock data
  end_date = dt.date.today()
  start_date = end_date - timedelta(days=number_of_years_to_fetch_price_data * 365)  # Getting start
date as last 'n' number of years from now
```

```python
    # Calling Yahoo Finance API for last 7 years of stock data
    df = pdr.get_data_yahoo(STOCK_TICKER, start = start_date, end = end_date)

    return df

# Plotting the graph visualizing price change with date with 15 years of data
df = extract_raw_data(15)
df.plot(y=[CLOSE],figsize=(15,10), ylabel='Stock Price', title='Stock Price Movement for last 15 years')

plt.figure(figsize=(15,10))
df = extract_raw_data(5)
df[CLOSE].rolling(window=20).mean().plot(label='20 Day Average')
df[CLOSE].plot(label='Closing Price', title='20 Day Moving Average')
plt.legend()

# Function to extract Stock Closing Price as a Feature from Dataframe
def select_features(df):
    return df.reset_index()[CLOSE]

def split_data_into_train_test_lstm(model_df):
    train_index = 0.7 * model_df.shape[0]
    train_data = model_df[:int(train_index)]
    test_data = model_df[int(train_index):]
    return train_data, test_data

# Function to create dataset into feature and target
def create_dataset_lstm(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - time_step-1):
        a = dataset[i:(i+time_step), 0]
        dataX.append(a)
        dataY.append(dataset[i+time_step, 0])
    return np.array(dataX), np.array(dataY)

def split_data_into_train_test(model_df):
    train_index = 0.7 * model_df.shape[0]
    train_data = list(model_df[:int(train_index)])
    test_data = list(model_df[int(train_index):])
    return train_data, test_data


# For other model
def create_dataset(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - time_step-1):
        a = dataset[i:(i+time_step)]
        dataX.append(a)
        dataY.append(dataset[i+time_step])
    return np.array(dataX), np.array(dataY)


# Function to tune the model
def create_tune_model(model_name, X_train, y_train, X_test, y_test, number_of_days=20, scaler=None):
    if model_name == 'KNN':
        return create_tune_knn(X_train, y_train, X_test, y_test)
```

```
    elif model_name == 'DT':
        return create_tune_dt(X_train, y_train, X_test, y_test)
    elif model_name == 'LR':
        return create_tune_lr(X_train, y_train, X_test, y_test)
    elif model_name == 'LSTM':
        return create_tune_lstm(X_train, y_train, X_test, y_test, number_of_days, scaler)


# Function to tune lstm based on specific hyperparameter
def create_tune_lstm(X_train, y_train, X_test, y_test, number_of_days, scaler):
    tuned_test_set_rmse = None
    tuned_hyperparameters = None
    tuned_model = None

    for no_of_epoch in NUMBER_OF_EPOCH_LIST:
        for batch_size in BATCH_SIZE_LIST:
            model = Sequential()
            model.add(LSTM(50, return_sequences=True, input_shape=(number_of_days,1)))
            model.add(LSTM(50, return_sequences=True))
            model.add(LSTM(50))
            model.add(Dense(1))
            model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])

            model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=no_of_epoch,
batch_size=batch_size, verbose=0)

            y_pred = model.predict(X_test)
            y_pred = scaler.inverse_transform(y_pred)
            y_test = scaler.inverse_transform(np.array(y_test).reshape(-1,1))
            current_rmse = math.sqrt(mean_squared_error(y_pred, y_test))
            current_rmse = math.sqrt(mean_squared_error(model.predict(X_test), y_test))

            if tuned_test_set_rmse is None or tuned_test_set_rmse > current_rmse:
                tuned_test_set_rmse = current_rmse
                tuned_hyperparameters = {'epochs': no_of_epoch, 'batch_size': batch_size}
                tuned_model = model

    return tuned_model, tuned_hyperparameters, tuned_test_set_rmse


# Function to run Linear Regression - no hyperparameter available in this model
def create_tune_lr(X_train, y_train, X_test, y_test):
    model = LinearRegression()
    model.fit(X_train, y_train)
    current_rmse = math.sqrt(mean_squared_error(model.predict(X_test), y_test))

    return model, {}, current_rmse


# Function to tune Decision Tree Regression based on specific hyperparameter
def create_tune_dt(X_train, y_train, X_test, y_test):
    tuned_test_set_rmse = None
    tuned_hyperparameters = None
    tuned_model = None

    for min_samples_split in MIN_SAMPLES_LIST:
```

```
        model = DecisionTreeRegressor(min_samples_split=min_samples_split)
        model.fit(X_train, y_train)
        current_rmse = math.sqrt(mean_squared_error(model.predict(X_test), y_test))

        if tuned_test_set_rmse is None or tuned_test_set_rmse > current_rmse:
            tuned_test_set_rmse = current_rmse
            tuned_hyperparameters = {'min_samples_split': min_samples_split}
            tuned_model = model

    return tuned_model, tuned_hyperparameters, tuned_test_set_rmse


# Function to tune KNN Regression based on specific hyperparameter
def create_tune_knn(X_train, y_train, X_test, y_test):
    tuned_test_set_rmse = None
    tuned_hyperparameters = None
    tuned_model = None

    for n_neighbors in KNN_NEIGHBORS_LIST:
        model = KNeighborsRegressor(n_neighbors=n_neighbors)
        model.fit(X_train, y_train)
        current_rmse = math.sqrt(mean_squared_error(model.predict(X_test), y_test))

        if tuned_test_set_rmse is None or tuned_test_set_rmse > current_rmse:
            tuned_test_set_rmse = current_rmse
            tuned_hyperparameters = {'n_neighbors': n_neighbors}
            tuned_model = model

    return tuned_model, tuned_hyperparameters, tuned_test_set_rmse


# Function to run the model
def run_model(model_name):
    best_model = None
    best_df = None
    best_X_train = None
    best_X_test = None
    best_test_data = None
    best_test_set_rmse = None
    best_hyperparameters = None
    best_no_of_years = None
    best_no_of_days = None

    # Looping through number of years of data to find accurate data selection
    for no_of_years in NUMBER_OF_YEARS_TO_FETCH_PRICE_DATA:

        # Extract Raw Dataset
        df = extract_raw_data(no_of_years)

        # Feature Selection - selecting stock closing price
        model_df = select_features(df)

        # Spliting data into train and test
        if model_name == 'LSTM':
            # Tranforming value to 0-1 since lstm are sensitive to the scale of the data
            scaler = MinMaxScaler(feature_range=(0,1))
```

```
        model_df_lstm = scaler.fit_transform(np.array(model_df).reshape(-1,1))

        train_data, test_data = split_data_into_train_test_lstm(model_df_lstm)
      else:
        train_data, test_data = split_data_into_train_test(model_df)

      # Looping through number of days for price prediction to find best time-steps
      for no_of_days in NUMBER_OF_DAYS_FOR_PRICE_PREDICTION:

        # Creating dataset out of it using time-steps
        if model_name == 'LSTM':
          X_train, y_train = create_dataset_lstm(train_data, no_of_days)
          X_test, y_test = create_dataset_lstm(test_data, no_of_days)

          # reshape input to be [samples, time steps, features] which is required for LSTM
          X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
          X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

          # Traning, Evaluating and performing Hyper Parameter Tuning for the LSTM Model
          tuned_model, tuned_hyperparameters, tuned_test_set_rmse =
create_tune_model(model_name, X_train, y_train, X_test, y_test, no_of_days, scaler)
        else:
          X_train, y_train = create_dataset(train_data, no_of_days)
          X_test, y_test= create_dataset(test_data, no_of_days)

          # Traning, Evaluating and performing Hyper Parameter Tuning for the Other Model
          tuned_model, tuned_hyperparameters, tuned_test_set_rmse =
create_tune_model(model_name, X_train, y_train, X_test, y_test)

        # Update hyperparameters and its test accuracy
        if best_test_set_rmse is None or best_test_set_rmse > tuned_test_set_rmse:
          best_test_set_rmse = tuned_test_set_rmse
          best_hyperparameters = tuned_hyperparameters
          best_no_of_years = no_of_years
          best_no_of_days = no_of_days
          best_model = tuned_model
          best_df = model_df
          best_X_train = X_train
          best_X_test = X_test
          best_test_data = test_data

  return best_model, best_df, best_X_train, best_X_test, best_test_data, best_hyperparameters,
best_test_set_rmse, best_no_of_years, best_no_of_days


# Defining default values for LSTM for displaying RMSE score later since it takes time to build in case we
don't want to run LSTM model
best_hyperparameters_lstm = None
best_no_of_years_lstm = None
best_no_of_days_lstm = None
best_test_set_rmse_lstm = None

# Running LSTM Model
best_model_lstm,best_df_lstm, best_X_train_lstm, best_X_test_lstm, best_test_data_lstm,
best_hyperparameters_lstm, \
  best_test_set_rmse_lstm, best_no_of_years_lstm, best_no_of_days_lstm = run_model('LSTM')
```

```
best_model_lr,best_df_lr, best_X_train_lr, best_X_test_lr, best_test_data_lr, best_hyperparameters_lr, \
    best_test_set_rmse_lr, best_no_of_years_lr, best_no_of_days_lr = run_model('LR')


best_model_dt,best_df_dt, best_X_train_dt, best_X_test_dt, best_test_dt, best_hyperparameters_dt, \
    best_test_set_rmse_dt, best_no_of_years_dt, best_no_of_days_dt = run_model('DT')


best_model_knn,best_df_knn, best_X_train_knn, best_X_test_knn, best_test_knn,
best_hyperparameters_knn, \
    best_test_set_rmse_knn, best_no_of_years_knn, best_no_of_days_knn = run_model('KNN')


# Defining default values for model for comparision
default_hyperparameter_lstm = {'epochs': 1, 'batch_size': None}
default_hyperparameter_lr = {}
default_hyperparameter_dt = {'min_samples_split': 2}
default_hyperparameter_knn = {'n_neighbors': 5}

# Using pandas dataframe to show data in a nice output table
df = pd.DataFrame(
    {'Model':                       ['LSTM',                    'Linear Regression',        'Decision Tree
Regression',        'KNN Regression'],
     'Default Hyperparameters':          [default_hyperparameter_lstm,    default_hyperparameter_lr,
default_hyperparameter_dt,        default_hyperparameter_knn],
     'Best Hyperparameters':              [best_hyperparameters_lstm,      best_hyperparameters_lr,
best_hyperparameters_dt,          best_hyperparameters_knn],
     'Stock Price Data (No. of Years)':     [best_no_of_years_lstm,          best_no_of_years_lr,
best_no_of_years_dt,              best_no_of_years_knn],
     'Time Series Step (No. of Days)':      [best_no_of_days_lstm,           best_no_of_days_lr,
best_no_of_days_dt,               best_no_of_days_knn],
     'RMSE':                           [best_test_set_rmse_lstm,        best_test_set_rmse_lr,
best_test_set_rmse_dt,            best_test_set_rmse_knn]
    })
df


# Common function to plot graph to evaluate Model
def plot_graph(model_df, no_of_days, model, X_train, X_test):
    # Making prediction for train and test set for plotting
    train_predict = model.predict(X_train)
    test_predict = model.predict(X_test)

    look_back = no_of_days
    train_predict_plot = [ np.nan for i in range(len(model_df))]
    train_predict_plot[look_back:len(train_predict) + look_back] = np.array(train_predict)

    # Shift test predictions for plotting

    test_predict_plot = [ np.nan for i in range(len(model_df))]
    test_predict_plot[len(train_predict) + (look_back*2)+1:len(model_df)-1] = list(test_predict)

    # Plot baseline and predictions
    plt.plot(model_df)
```

```
    plt.plot(train_predict_plot)
    plt.plot(test_predict_plot)
    plt.show()


plot_graph(best_df_lr,best_no_of_days_lr, best_model_lr, best_X_train_lr, best_X_test_lr)

# Plotting Stock Market Trend for next 30 days using Linear Regression
lastest_data_index = len(best_test_data_lr) - best_no_of_days_lr
x_input = best_test_data_lr[lastest_data_index:]

temp_input = list(x_input)
temp_input = [temp_input[0]]

# demonstrate prediction for next n days
lst_output = []
last_n_days_data = best_test_data_lr[lastest_data_index : ]
next_day = None
for day in range(NUMBER_OF_DAYS_TO_PREDICT):
    if next_day is not None:
        last_n_days_data = last_n_days_data[1:]
        last_n_days_data.append(next_day)
    next_day = best_model_lr.predict([last_n_days_data])[0]
    lst_output.append(next_day)

day_new = np.arange(1,best_no_of_days_lr + 2)
day_pred = np.arange(best_no_of_days_lr + 1, best_no_of_days_lr +
NUMBER_OF_DAYS_TO_PREDICT + 1)

lastest_model_df_index = len(best_df_lr) - best_no_of_days_lr

plt.figure(figsize = (15,10))

last_n_days = list(best_df_lr[lastest_model_df_index:])
last_n_days.append(lst_output[0])

plt.title('Stock Market Movement Trend for next 30 days')
plt.plot(day_new,last_n_days)
plt.plot(day_pred,lst_output)


# Current Date of Model Run
print('Current Date:                ', dt.date.today())
print('Last Stock Closing Price:     ', best_test_data_lr[-1])

# Next day price using Linear Regression
last_n_days_data = best_test_data_lr[len(best_test_data_lr) - best_no_of_days_lr:]
next_day = best_model_lr.predict([last_n_days_data])[0]

print('Next Day Stock Closing Price: ', next_day)
```