PROJECT REPORT

STOCK MARKET PREDICTION ANALYSIS

Submitted by

BILAL ASHRAF 2018-310-032

in partial fulfilment for the award of the degree of
BACHELORS OF TECHNOLOGY

Under the supervision of
Dr. JAWED

Department of Computer Science & EngineeringSchool of Engineering Sciences & Technology

JAMIA HAMDARD

New Delhi - 110062

(2022)

JAMIA HAMDARD

Phone : 011-2605 9588 (12Lines)
Fax : 00-91-11-26059663
Email:inquiry@jamiahamdard.edu
Website: www.jamiahamdard.edu

(Hamdard University)

(*Declared as Deemed-to-be University under Section 3 of the UGC Act, 1956 vide Notification No. F.9-18/85-U.3 dated 10.5.1989 of the Government of India)*

HAMDARD NAGAR
NEW DELHI-110062

Accredited by NAAC in 'A' Category

CERTIFICATE

On the basis of the declaration submitted by **Mr. Bilal Ashraf (Enrolment No: 2018-310-032)** a student of **Bachelor of Technology (Computer Science & Engineering),** I hereby certify that the dissertation entitled **"Stock Market Prediction Analysis"** being submitted to the Department of Computer Science & Engineering, Jamia Hamdard, New Delhi in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology (Computer Science & Engineering),** is carried out by him under my supervision.

Dr. Jawed
(Supervisor)

Dr. Farheen Siddiqui
Head, Department of CSE

PLAGIARISM REPORT

Curiginal

Document Information

Analyzed document	plag2.pdf (D138371331)
Submitted	2022-05-29T19:36:00.0000000
Submitted by	Tabish
Submitter email	tabishmufti@jamiahamdard.ac.in
Similarity	1%
Analysis address	tabishmufti.jahau@analysis.urkund.com

Sources included in the report

W	URL: https://link.springer.com/article/10.1007/s11276-022-02994-y Fetched: 2022-05-29T19:36:45.4200000	1
W	URL: https://www.pluralsight.com/guides/introduction-to-lstm-units-in-rnn Fetched: 2020-11-11T13:03:14.2770000	1

DECLARATION

I, Mr. Bilal Ashraf, a student of Bachelors of Technology (B.Tech), (Enrolment No: 2018-310-032) hereby declare that the Project/Dissertation entitled "Stock Market Prediction Analysis" which is being submitted by me to the Department of Computer Science, Jamia Hamdard, New Delhi in partial fulfilment of the requirement for the award of the degree of Bachelors of Technology (B.Tech), is my original work and has not been submitted anywhere else for the award of any Degree, Diploma, Associateship, Fellowship or other similar title or recognition

(Signature and Name of the Applicant)

Date: 31 MAY 2022

Place: JAMIA HAMDARD, DELHI

ACKNOWLEDGEMENTS

I am very thankful to my supervisor, Dr. Jawed, for his invaluable assistance throughout the semester; his direction on this research-based capstone was highly welcomed and appreciated. I am also thankful to all the other instructors I had during my bachelor's degree in computer science; they imparted vital skills that aided me throughout this project and more. In addition, I would like to thank everyone who provided me with research papers, tutorials, figures, and pre-built models in order to complete this study. In addition, I would like to thank my entourage, particularly my family and friends, for the emotional support they provided me through both good and difficult times. Finally, I would want to convey my heartfelt appreciation to everyone who has taken the time to read this report, which I hope you will find fascinating.

CONTENTS

SYSTEM REQUIREMENTS

Recommended System Requirements

- **Processors:**
    - Intel® Core™ i5 processor 4300M at 2.60 GHz or 2.59 GHz (1 socket, 2 cores, 2 threads per core), 8 GB of DRAM
- **Disk space:** 2 to 3 GB
- **Graphics card:** Nvidia 960
- **Operating systems:** Windows® 10, macOS®, and Linux®

Minimum System Requirements

- **Processors:** Intel Atom® processor or Intel® Core™ i3 processor
- **Graphics card:** Intel HD 640
- **Disk space:** 1.5 GB
- **Operating systems:** Windows® 7 or later, macOS®, and Linux®
- **Python versions:** 3.X
- **Included development tools:** Conda, Conda-Env, Jupyter Notebook (IPython)
- **Compatible tools:** Microsoft Visual Studio®, PyCharm
- **Included Python packages:** NumPy, SciPy, scikit-learn, pandas, Matplotlib, Numba, Intel® Threading Building Blocks, pyDAAL, Jupyter, mpi4py, PIP, and others.

MODULES

- ***Pandas:*** - Pandas is an open source library in Python. It provides ready to use high-performance data structures and data analysis tools. Pandas module runs on top of NumPy and it is popularly used for data science and data analytics.
- ***NumPy***: - NumPy is a Python package which stands for 'Numerical Python'. It is the core library for scientific computing, which contains a powerful n-dimensional array object, provide tools for integrating C, C++ etc. It is also useful in linear algebra, random number capability etc
- ***Matplotlib***: - Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.
- ***Scikit-learn:*** - Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy.
- ***Keras: -***Keras is a minimalist Python library for deep learning that can run on top of Theano or TensorFlow. It was developed to make implementing deep learning models as fast and easy as possible for research and development
- ***TensorFlow:*** - It is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow.
- ***Pandas-datareader:*** - Up to date remote data access for **pandas**, works for multiple versions of **pandas**. Warning. v0. 8.0 is the last version which officially supports Python 2.7.
- ***Math:*** - **Python math module** is defined as the most popular mathematical functions, which includes trigonometric functions, representation functions, logarithmic functions, etc. Furthermore, it also defines two mathematical constants, i.e., Pie and Euler number, etc.

ABSTRACT

It is not an easy task to predict any shares value, even more so using machine learning. There are a lot of variables involved in the same. In this project report I have tried to accomplish different methods to find out how a stock will behave. Portraying a stock price is done usually using time series data, where neural networks look for existing data movement to find out the inclination of what the next prices could be. Using BMCE BANK previous data of its prices, we analyse the results of a basic single LSTM model. A hybrid model that combines sentiment analysis and stacking auto-encoders has also been looked at.

INTRODUCTION

Anybody who is only starting out in the world of stock market can employ these techniques and benefit largely in making better decisions. To help someone who is new, the neural networks that have been built before can be employed for transfer learning. After then, whatever projections our data represent, we will overlap it with what the actual stock prices behaved like. We can thus make safer investment knowing our money isn't at risk. Because of the quality of bending easily without breaking and disburse pre-built models and open-source modules, python happens to be the best choice . Also, the best-fitting model is the LSTM (Long Short Term Memory) model. The memory component can be very beneficial to further improve the quality of LSTM model which will further help for the time series forecasting. This model even has the potential to perform even better than the earlier ones to conjecture the possible outcomes.

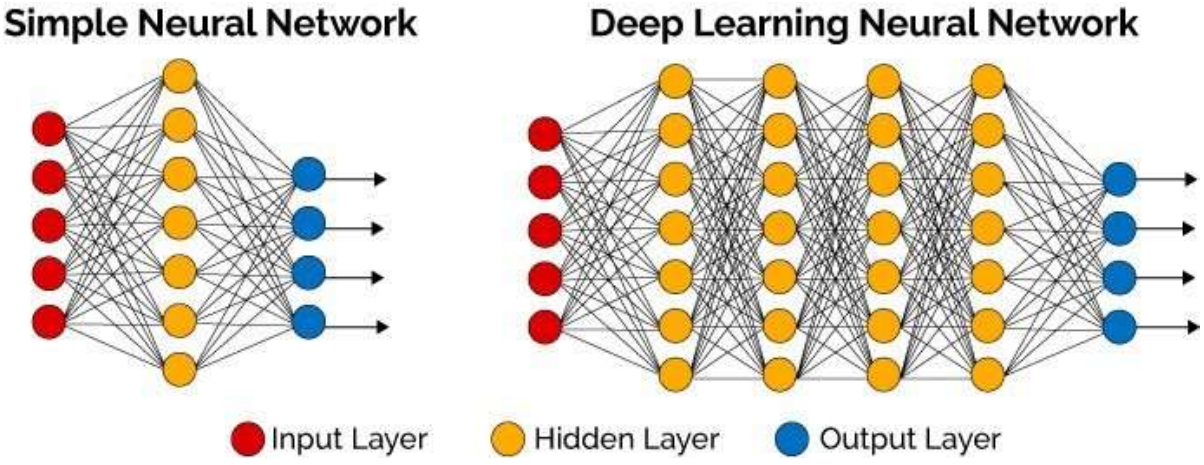
CHAPTER 1

THEORITICAL BACKGROUND

1.1 Deep Learning

The way humans' brain perform, deep learning has been an approach that uses artificial intelligence, and lately been used to give lessons to robot. Many applications have come forward for this like self-driving cars to understand traffic signs, recognise pedestrians, and even evaluate whether or not a driver is attentive in order to park the vehicle safely.

A Deep Learning allows a computer model to learn to categorise images, text, or voice. Deep learning models have the potential to attain cutting-edge accuracy, sometimes surpassing that of humans.



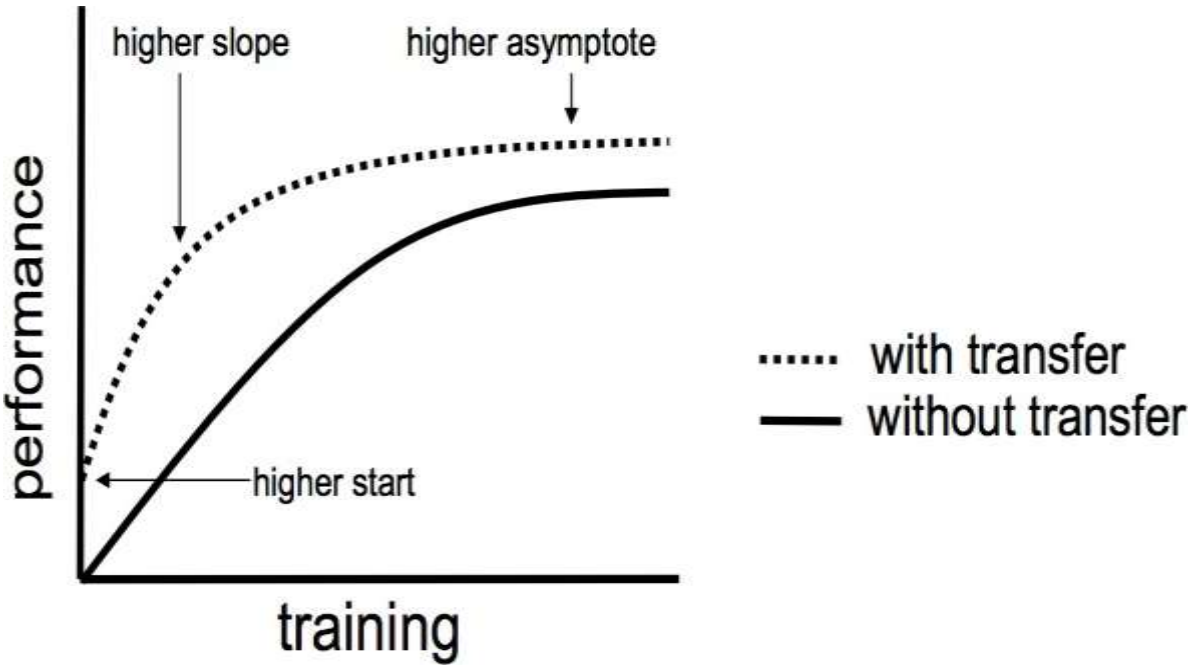
(Image Source: https://thedatascientist.com/what-deep-learning-is-and-isnt/)

1.2 Transfer Learning

This uses earlier data points to find out newer ones. What this does is hugely improve the process of accomplishing an action. The approach we'll try to look into is -

Model Development Approach:

To begin, we pick all the stocks prices of a stock ever since it was created only to increase the data points. Different target points are taken to eventually calculate the upcoming targets.



Performance with Transfer Learning & without

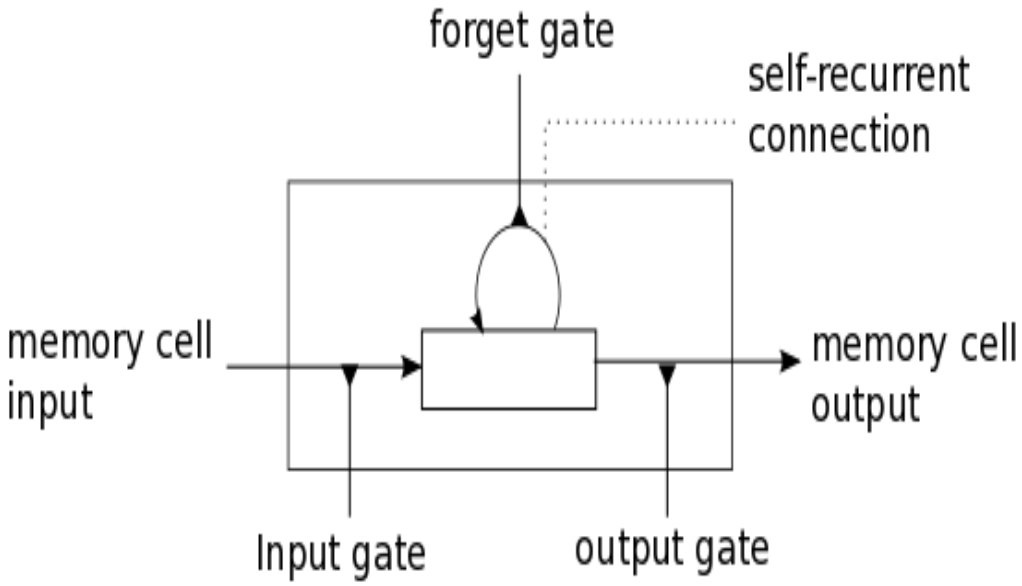
(Image Source: https://www.researchgate.net/figure/Performance-graph-with-and-without-Transfer-Learning_fig2_345904103)

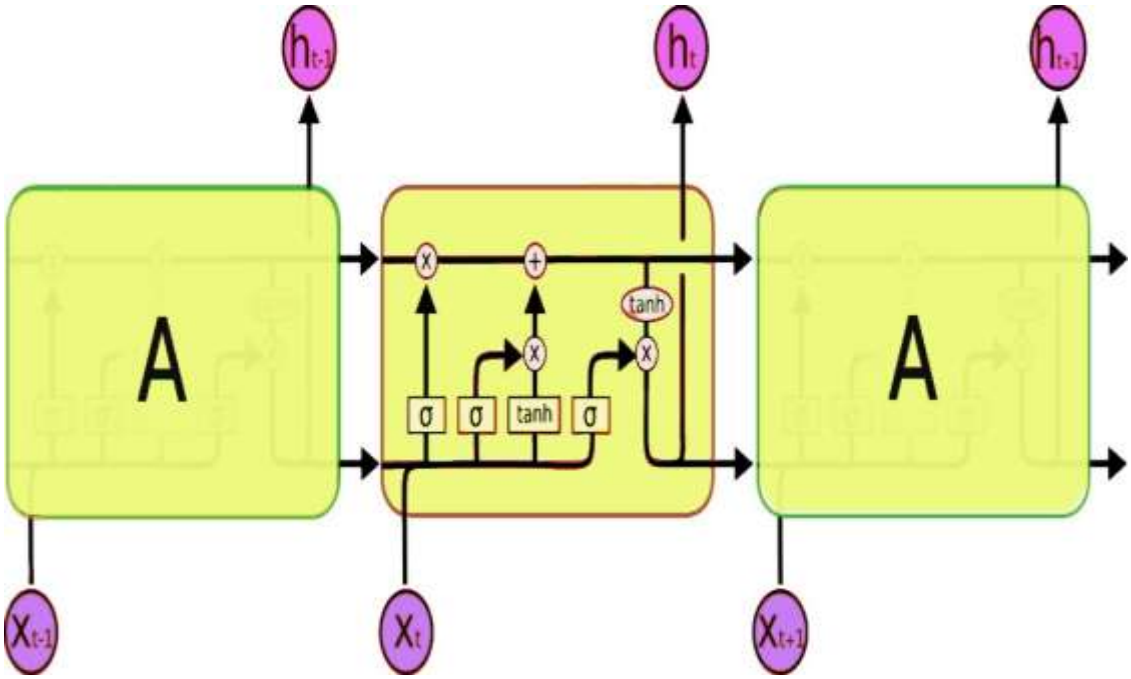
1.3 Time Series Analysis

A time time series is a set of data points or values that have been organized, catalogued, or visualised. Time-series analysis' major aims are to understand what is really going on and anticipate what will occur subsequently. Among these objectives are the finding and systematic elucidation of empirical results in time series. Procedures for gathering feedback from a variety of data points over duration are referred to as time series analysis. A series of statistical mining approaches may be used, each with varying degrees of efficacy. Following that, we'll look at some very intriguing methods for estimating time series data, and therefore how LSTM neural networks have swept over the domain.

1.4 Long Short Term Memory model (LSTM)

A time series assessing neural network is described as an LSTM. This is due to their ability to store trends deliberately throughout period. LSTMs replace the conventional feed-forward neural networks and recurrent neural networks in a lot of formats. LSTMs may arbitrarily recall or discard details. They really do not divide between 'critical' and 'less valuable' data.



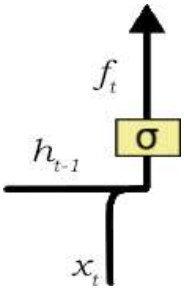


LSTM Cell Architecture

(Image Source: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/)

The cells which are like blocks of different memory are what comprise a LSTM. Mainly hidden and cell state get carried over to the cell which is next in line. Memory blocks are in charge of knowing information, and they are altered by three basic procedures known as gates which we discuss in the upcoming pages.

Forget Gate:



(Image Source: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/)

Forget gate suffers from amnesia and any unnecessary data, as the name suggests itself, is forgotten. For the forward passing of an LSTM cell with a forget gate, the succinct formulations of the calculations are:

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

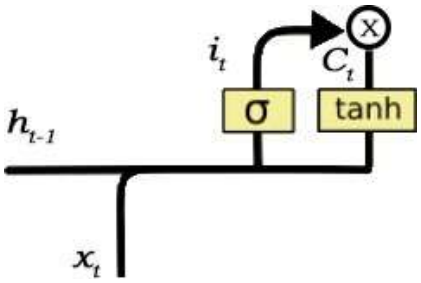
(Source: https://en.wikipedia.org/wiki/Long_short-term_memory)

Variables :

- $x_t \in \mathbb{R}^d$: input vector to the LSTM unit
- $f_t \in (0,1)^h$: forget gate's activation vector
- $i_t \in (0,1)^h$: input/update gate's activation vector
- $o_t \in (0,1)^h$: output gate's activation vector
- $h_t \in (-1,1)^h$: hidden state vector also known as output vector of the LSTM unit
- $\tilde{c}_t \in (-1,1)^h$: cell input activation vector
- $c_t \in \mathbb{R}^h$: cell state vector
- $W \in \mathbb{R}^{h \times d}$, $U \in \mathbb{R}^{h \times h}$ and $b \in \mathbb{R}^h$: weight matrices and bias vector parameters which need to be learned during training

where the superscripts d and h refer to the number of input features and number of hidden units, respectively.

Input Gate:



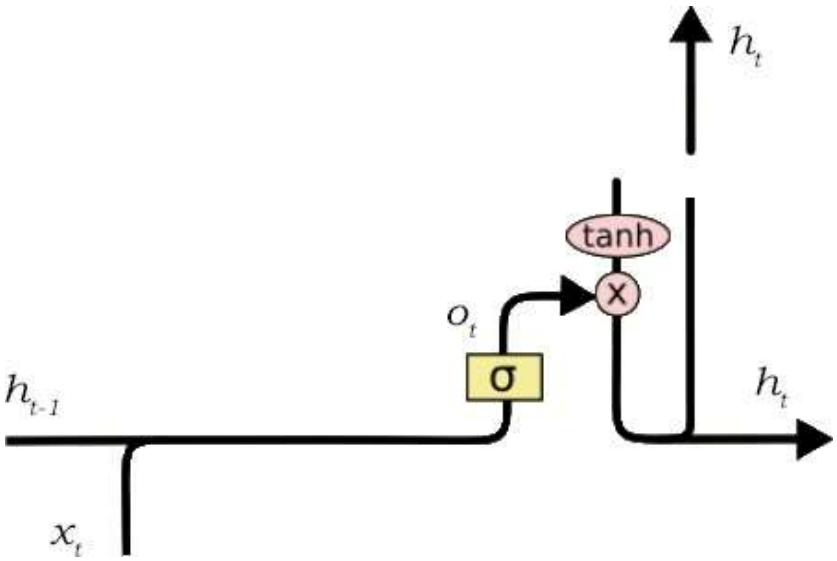
(Image Source: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/)

The data has to be offered towards the cell state, for which input gate is made in charge of and to patch the cell state, the input gate conducts the three processes -

- The subsequent sigmoid function receives the existing operational X(t) and the formerly concealed data h(t-1). The data is adjusted from 0 (critical) to 1 (less vital).
- The tanh function will be used to provide the identical information from the concealed state and contemporary state. The tanh operator will produce a vector (C(t)) with all the attribute outcomes within -1 and 1 to monitor the infrastructure.
- The activation functions provide model parameters that are suitable for stage multiplication.

We ensure that only pertinent information is submitted to the repository after these three requirements have been met.

Output Gate:



(Image Source: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/)

The output gate is in function of extracting and presenting critical data from an existing cell state. An output gate's action may be composed of three main components once more:

1. Using the tanh function on the cell state and adjusting the numbers to -1 to +1 to create a vector.
2. Implementing a strainer utilizing parameters of h t-1 and x t to govern the results from either the vector produced before. Yet another, a sigmoid function can be used in any strainer.
3. Spreading the outcome of this prudential strainer through the vector created in and passing it as an outlet also to the subsequent cell's hidden state.

1.5 Stacked Auto-Encoders

An input layer, a convolution layer, and a rebuilding layer make up a unique layer autoencoder, and that is a three-layer neural net. These autoencoders are used to produce convolution layers features that are greater depth or more conceptual. During the development of these autoencoders, we try to minimise the discrepancy between the intake and rebuilding layers.

For financial time series, the piled autoencoder is a variety of deep learning model. It has four autoencoders and five layers in this context, and its goal is to provide enhanced qualities. A multilayered auto encoder's initial layer learns first-order information (such as picture edges), while the lower part learns second-order data (i.e., contours or corners).

CHAPTER 2

RELATED WORK AND RESULTS

2.1 Single LSTM Model example

Our algorithm is formed by two tiered Hidden layer and a fully - connected layers, featuring linear regression executed at the final layer level. MSE (mean squared error) has been used for loss evaluations, while RMSProp is seen as an evolutionary algorithm. The implications of this paradigm are presented in the graphic here:

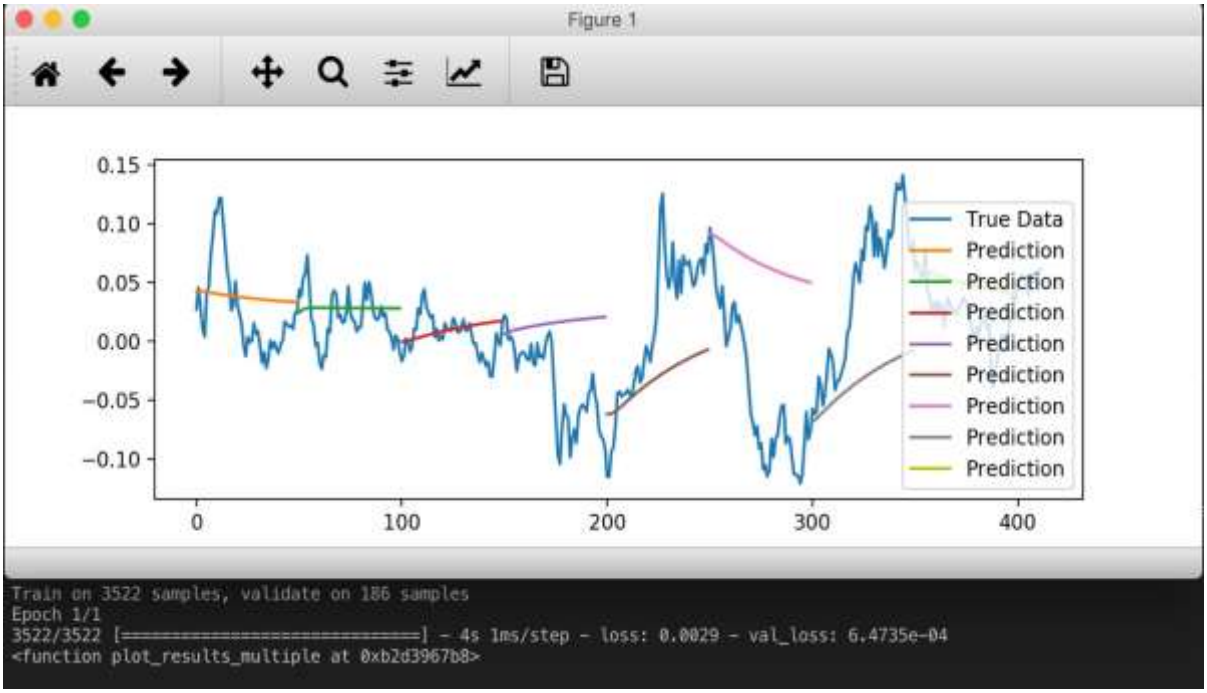
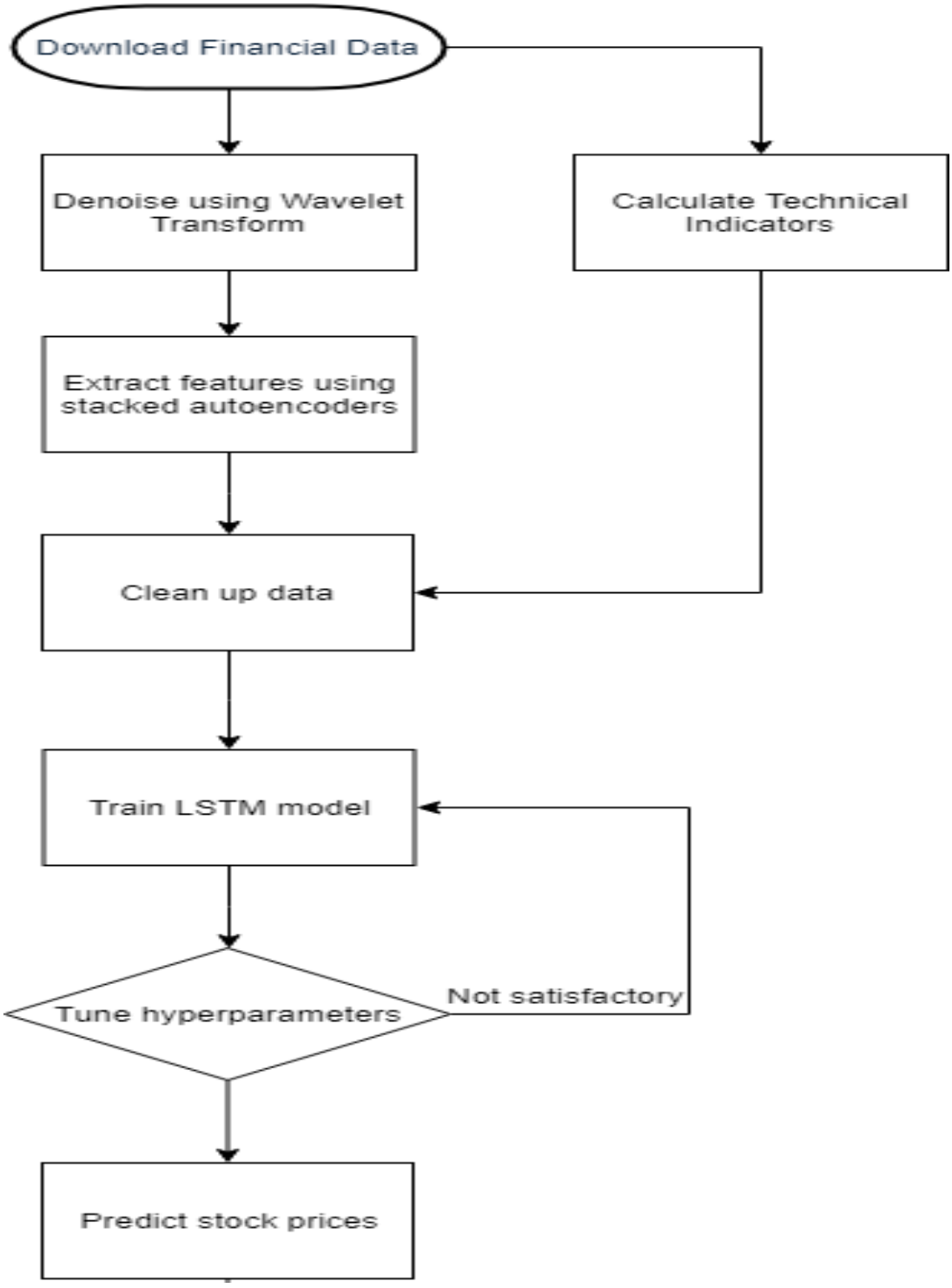


Figure: Results of Single LSTM

It's a relatively straightforward concept to include an LSTM deep neural network to foresee value of equities. The plot here displays that the model's recommendations aren't as comprehensive as we'd like. It's probably the easiest way, as it is built for educating and exploring with time series data.

2.2 Hybrid LSTM Model Flowchart



It really is a deep learning method that incorporates layered autoencoders to blend news scraping for sentiment analysis. For scalability and improvement, this program utilizes the Keras toolbox to make multiple closely packed tiers with 20 as the incoming variable, a tanh activation unit, and an interaction rejigger to severely punish component disturbance. It leverages SGD (Stochastic gradient descent) for effectiveness and reliability, which are noteworthy aspects for a pretrained model throughout 2000 iterations.

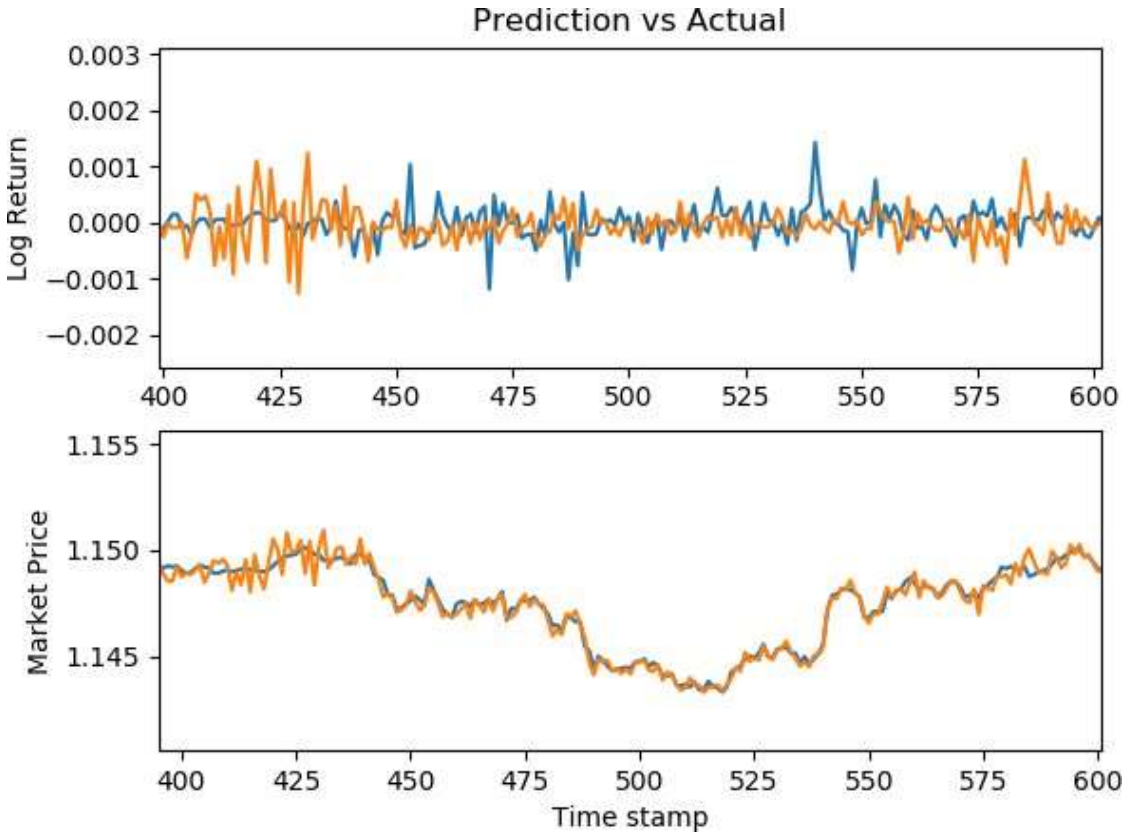


Figure: Results of Hybrid LSTM

CHAPTER 3

STOCK PREDICTION FOR BMCE BANK

The statistics below comes from a three-year-old Moroccan information gathering for BMCE shares. The following was the initial set of results generated from the Casablanca-bourse official site:

Session	Security	Reference price	Last price	+Intraday high	+ Intraday low	Number of shares traded	Capitalisation
19/04/2019	BMCE BANK	190,05	190,00	190,05	187,00	16044	34 098 044 100,00
18/04/2019	BMCE BANK	192,10	190,00	193,00	190,00	18096	34 098 044 100,00
17/04/2019	BMCE BANK	192,00	193,10	193,20	192,00	675	34 654 380 609,00
16/04/2019	BMCE BANK	194,00	193,00	194,00	191,60	6124	34 636 434 270,00
15/04/2019	BMCE BANK	192,00	193,00	193,00	191,05	4965	34 636 434 270,00
12/04/2019	BMCE BANK	191,05	193,00	193,00	191,05	665	34 636 434 270,00
11/04/2019	BMCE BANK	193,00	193,00	193,00	193,00	225	34 636 434 270,00
10/04/2019	BMCE BANK	193,00	194,95	194,95	191,00	23159	34 986 387 880,50
09/04/2019	BMCE BANK	190,00	193,00	193,00	189,00	683	34 636 434 270,00
08/04/2019	BMCE BANK	186,50	193,00	193,00	186,50	466	34 636 434 270,00
05/04/2019	BMCE BANK	187,00	190,00	190,00	187,00	45084	34 098 044 100,00
04/04/2019	BMCE BANK	185,00	188,00	188,00	185,00	60	33 739 117 320,00
03/04/2019	BMCE BANK	190,00	185,00	190,00	185,00	19210	33 200 727 150,00
02/04/2019	BMCE BANK	182,00	185,00	185,00	181,95	8308	33 200 727 150,00

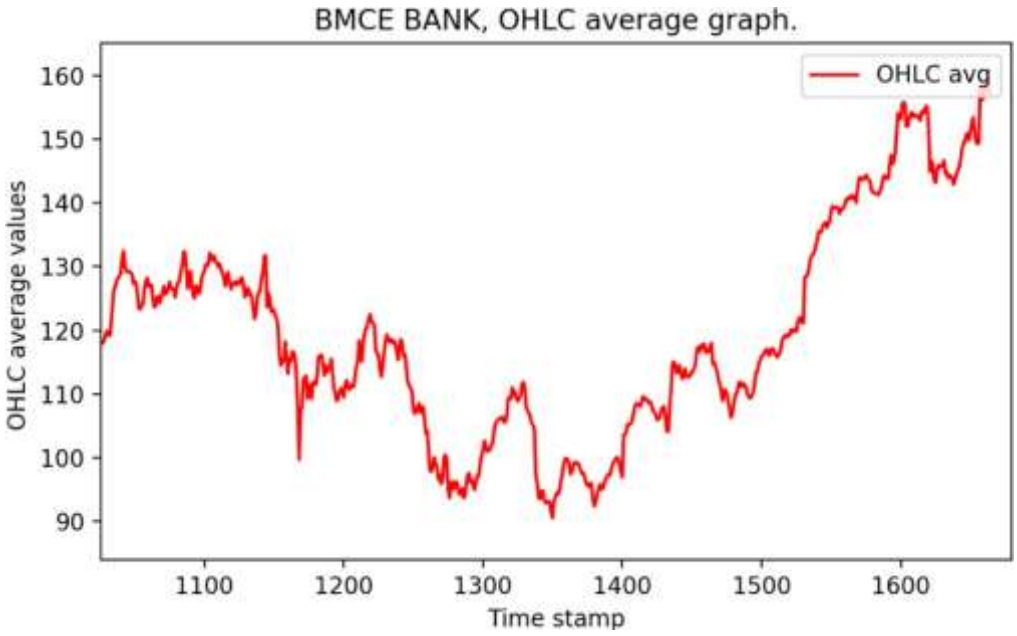
Original BMCE BANK Dataset

The information was processed, scraped, and transformed to.csv format. Our conditions now looks such as this:

Date	Open	High	Low	Close	Volume
19-Apr-19	190.05	190.05	187	190	34 098 044 100
18-Apr-19	192.1	193	190	190	34 098 044 100
17-Apr-19	192	193.2	192	193.1	34 654 380 609
16-Apr-19	194	194	191.6	193	34 636 434 270
15-Apr-19	192	193	191.05	193	34 636 434 270
12-Apr-19	191.05	193	191.05	193	34 636 434 270
11-Apr-19	193	193	193	193	34 636 434 270
10-Apr-19	193	194.95	191	194.95	34 986 387 880.50
9-Apr-19	190	193	189	193	34 636 434 270
8-Apr-19	186.5	193	186.5	193	34 636 434 270
5-Apr-19	187	190	187	190	34 098 044 100
4-Apr-19	185	188	185	188	33 739 117 320
3-Apr-19	190	190	185	185	33 200 727 150
2-Apr-19	182	185	181.95	185	33 200 727 150
1-Apr-19	178	178.05	177	177	31 765 020 030
29-Mar-19	182	182	178	178	31 944 483 420
28-Mar-19	178	182	178	182	32 662 336 980
27-Mar-19	179	180	178.1	180	32 303 410 200
26-Mar-19	179.9	179.9	177.1	179.9	32 285 463 861
25-Mar-19	184.85	184.85	180	180	32 303 410 200
22-Mar-19	185	185	178	178	31 944 483 420
21-Mar-19	179.55	180	179.05	180	32 303 410 200
20-Mar-19	180	180	179	180	32 303 410 200

To make projections, market participants often employ two indices:

1. OHLC (average of Open, High, Low and Closing Prices)
2. HLC (average of High, Low and Closing Prices



- The time series is normalized using the OHLC method. There is now only solitary field of information in the data source.

- The time series was therefore broken into 2 categories for analysis..

    1. The stock price at period interval is reflected in the first column.
    2. The financial position during time t+1 is presented in the second column.

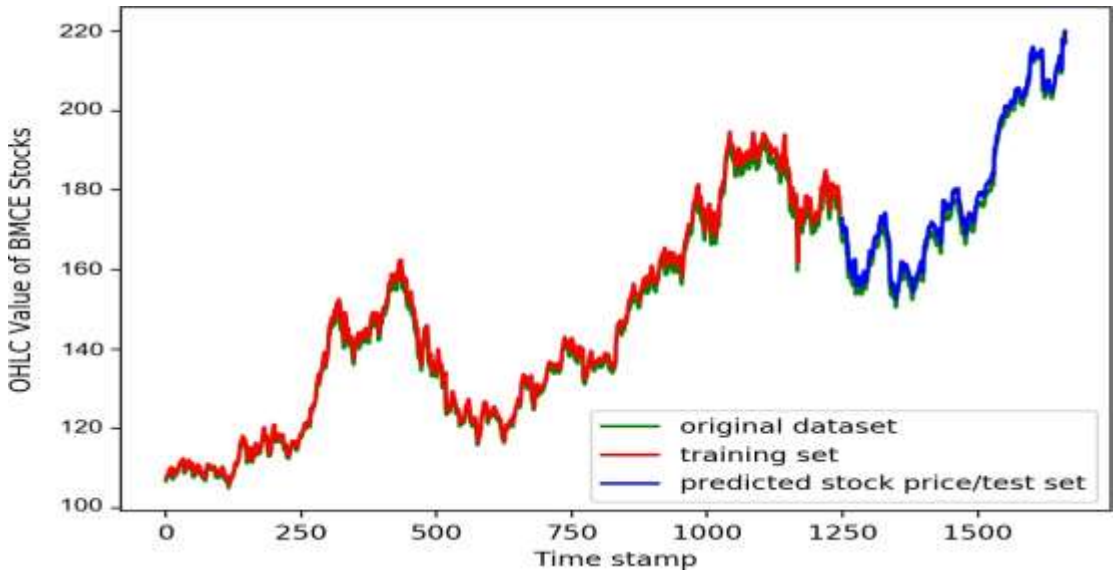
.

- After that, all levels are related within 1 and 0.

Divided into training and testing data with appropriate proportions of 75% and 25%:

```
# TRAIN-TEST SPLIT
train_OHLC = int(len(OHLC_avg) * 0.75)
test_OHLC = len(OHLC_avg) - train_OHLC
train_OHLC, test_OHLC = OHLC_avg[0:train_OHLC,:], OHLC_avg[train_OHLC:len(OHLC_avg),:]
```

This system contains two consecutive LSTM stages put out using the Keras deep learning toolbox. For the last tier, linear regression is applied considering the nature of the problem. I opted to be using the Adam optimizer after altering the script, especially for the data pre-processing module, and playing with other metaheuristics.



```
Epoch 1/10
 - 8s - loss: 0.0101
Epoch 2/10
 - 6s - loss: 2.5335e-04
Epoch 3/10
 - 6s - loss: 1.6282e-04
Epoch 4/10
 - 6s - loss: 1.6146e-04
Epoch 5/10
 - 6s - loss: 1.6072e-04
Epoch 6/10
 - 6s - loss: 1.8197e-04
Epoch 7/10
 - 6s - loss: 1.6378e-04
Epoch 8/10
 - 6s - loss: 1.7064e-04
Epoch 9/10
 - 6s - loss: 1.8801e-04
Epoch 10/10
 - 6s - loss: 1.7277e-04
Train RMSE: 2.04
Test RMSE: 2.40
Last Day Value: 189.82058715820312
Next Day Value: 190.32862854003906
```

Training for 10 Epochs

Here, one can see how much he might be losing out by looking at the root mean squared error.

CHAPTER 4

CONCLUSION AND FUTURE WORKS

After doing so, it's clear that the integration of LSTM networks, stacked autoencoders, and sentiment analysis yields reliable findings for realtime investing. New efforts on this venture will necessitate this in investigation commitment to tailor the combined approach, particularly yields the desired outcomes, to the BMCE market. In this case, the enormous volume of information data, which is unsuitable for an effective and marketable classifier, is perhaps a considerable impediment. It may be taken into account by applying data augmentation techniques to existing data sets in order to enhance their capacity and render them relevant for profound project - based learning..

Appendix A –

SINGLE LSTM MODEL CODE SNIPPETS

Building the model:

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import newaxis
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential


def load_data(filename, seq_len, normalise_window):
    f = open(filename, 'rb').read()
    data = f.decode().split('\n')

    sequence_length = seq_len + 1
    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index: index + sequence_length])

    if normalise_window:
        result = normalise_windows(result)

    result = np.array(result)

    row = round(0.9 * result.shape[0])
    train = result[:int(row), :]
    np.random.shuffle(train)
    x_train = train[:, :-1]
    y_train = train[:, -1]
    x_test = result[int(row):, :-1]
    y_test = result[int(row):, -1]

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

    return [x_train, y_train, x_test, y_test]

def normalise_windows(window_data):
    normalised_data = []
    for window in window_data:
        normalised_window = [((float(p) / float(window[0])) - 1) for p in window]
        normalised_data.append(normalised_window)
    return normalised_data

```

```
def build_model(layers):
    model = Sequential()

    model.add(LSTM(
        input_shape=(layers[1], layers[0]),
        output_dim=layers[1],
        return_sequences=True))
    model.add(Dropout(0.2))

    model.add(LSTM(
        layers[2],
        return_sequences=False))
    model.add(Dropout(0.2))

    model.add(Dense(
        output_dim=layers[3]))
    model.add(Activation("linear"))

    start = time.time()
    model.compile(loss="mse", optimizer="rmsprop")
    print("> Compilation Time : ", time.time() - start)
    return model
```

```
def predict_point_by_point(model, data):
    #Predict each timestep given the last sequence of true data, in effect only predicting 1 step ahead each time
    predicted = model.predict(data)
    predicted = np.reshape(predicted, (predicted.size,))
    return predicted

def predict_sequence_full(model, data, window_size):
    #Shift the window by 1 new prediction each time, re-run predictions on new window
    curr_frame = data[0]
    predicted = []
    for i in range(len(data)):
        predicted.append(model.predict(curr_frame[newaxis,:,:])[0,0])
        curr_frame = curr_frame[1:]
        curr_frame = np.insert(curr_frame, [window_size-1], predicted[-1], axis=0)
    return predicted

def predict_sequences_multiple(model, data, window_size, prediction_len):
    #Predict sequence of 50 steps before shifting prediction run forward by 50 steps
    prediction_seqs = []
    for i in range(int(len(data)/prediction_len)):
        curr_frame = data[i*prediction_len]
        predicted = []
        for j in range(prediction_len):
            predicted.append(model.predict(curr_frame[newaxis,:,:])[0,0])
            curr_frame = curr_frame[1:]
            curr_frame = np.insert(curr_frame, [window_size-1], predicted[-1], axis=0)
        prediction_seqs.append(predicted)
    return prediction_seqs

def plot_results_multiple(predicted_data, true_data, prediction_len):
    fig = plt.figure(facecolor='white')
    ax = fig.add_subplot(111)
    ax.plot(true_data, label='True Data')
    #Pad the list of predictions to shift it in the graph to it's correct start
    for i, data in enumerate(predicted_data):
        padding = [None for p in range(i * prediction_len)]
        plt.plot(padding + data, label='Prediction')
        plt.legend()
    plt.show()
```

Running the model:

```
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential
import lstm, time


#loading the Data.
X_train, y_train, X_test, y_test = lstm.load_data('data/sp500.csv', 50, True)

#building Model
model = Sequential()

model.add(LSTM(input_dim=1,
               output_dim=50,
               return_sequences=True))

model.add(Dropout(0.2))

model.add(LSTM(100, return_sequences=False))
model.add(Dropout(0.2))

model.add(Dense(
           output_dim=1))
model.add(Activation('linear'))

start = time.time()
model.compile(loss='mse', optimizer='rmsprop')

print('compilation time:', time.time() - start)

#Train the model
model.fit(
    X_train,
    y_train,
    batch_size=512,
    nb_epoch=1,
    validation_split=0.05)


print(lstm.plot_results_multiple)


#plot results
predictions = lstm.predict_sequences_multiple(model, X_test, 50, 50)
lstm.plot_results_multiple(predictions, y_test, 50)

```