# Stock prediction model based on LSTM and ARIMA

**20 May 2022**

**Group 3**

Zheng Wugeng p930026173@mail.uic.edu.cn
Qu Weiting p930031140@mail.uic.edu.cn
Chen Ziqi p930026017@mail.uic.edu.cn

***Abstract* —** The prediction of stock prices has always been a hot research topic. However, the commonly used autoregressive integrated moving average (ARIMA) model still has its own advantages and disadvantages. The use of the long short-term memory (LSTM) network model for prediction also shows interesting possibilities. This article compares two models specifically through the analysis of the principles of the two models and the prediction results. The combination of time series and external factors may be a worthy research direction.

*Keywords: Stock prediction; time series; deep learning, neural networks; auto regressive integrated moving average (ARIMA); long short-term memory (LSTM)*

## Introduction and Background

In the financial domain, stock prediction is a very crucial task. Based on this prediction, future transactions influence a lot. With the continuous application and development of artificial intelligence technology and big data technology, along with the further improvement of the financial market and the strong demand of the financial service industry, the stock market prediction has attracted extensive attention from the industry and academia.

This paper firstly establishes the ARIMA model, which is used to study the trend of the stock, and obtain the estimated value of the stock forecast, and check the fit and adaptability of the model. Combining the stock forecast model needs to deal with nonlinear problems and the stock has the characteristics of time series, so the work also uses the recurrent neural network to forecast the stock. While Recurrent Neural Networks (RNNs) allow persistence of information. However, the general RNN model has a weak ability to describe time series data with long memory. When the time series is too long, there are gradient dissipation and gradient explosion phenomena, which make RNN training very difficult. The Long Short-Term Memory (LSTM) model proposed is modified on the basis of the RNN structure, thus solving the problem that the RNN model cannot describe the long memory of time series.

To sum up, the ARIMA and LSTM model in deep learning can well describe the long memory of time series and get results for stock price prediction.

## Related Works

The prediction of financial stocks has always been a research hotspot in the financial field. In terms of methods, it can be roughly divided into linear prediction models and nonlinear prediction

models. Among them, linear prediction models mainly include the Autoregressive Integrated Moving Average model (ARIMA), GARCH, EGARCH and IGARCH. As an early stock forecasting model, the above-mentioned linear forecasting model has played a pivotal role in promoting the forecasting development of the entire financial stock.

However, given the high noise and nonlinear characteristics of financial time series, it is still very difficult to accurately predict financial stock prices through linear prediction models. With the rapid development of computer technology and the advancement of deep learning research, neural networks in the field of machine learning are increasingly widely used in stock forecasting and have achieved more efficient and accurate forecasting results than linear forecasting models. The predicted accuracy using BP neural network and grey GARCH-BP model is significantly better than GARCH model.

Since the neural network prediction model has significant nonlinearity, we classify the neural network model as a nonlinear prediction model. Neural networks are divided into two categories: the first category is artificial neural network (ANN). However, due to the single structure of the ANN model, the generalization ability of the model is greatly weakened due to over-fitting, and the problem of local extreme values leads to a greatly weakened model prediction ability. And in the optimization process, it is easy to cause the gradient disappearance or gradient explosion problem due to too many neuron weights, and finally make the neural network model prediction invalid.

Deep neural network models (DNN), such as convolutional neural network (CNN), recurrent neural network (RNN) and long short-term memory neural network (LSTM), are the most efficient and cutting-edge forecasting models in the field of financial forecasting. Advantages: There is no restriction on the form of input variables, and the information that may be relevant to the forecasting problem can be used as the model input, considering the characteristics of the stock market being easily affected by various kinds of information. And it can effectively fit the nonlinear complex relationship between input variables, improve the degree of sample fitting, and at the same time, through the principle of neuron weight cycle, the number of neuron weights is greatly reduced, and the phenomenon of over-fitting is effectively prevented. Through the tanh activation function in DNN, the problems of gradient explosion and gradient disappearance in ANN can be significantly solved.

## Methodology

**ARIMA Model Architecture:**

Autoregressive integrated moving average (ARIMA) models can be used to predict time series data based on the history data.

An ARIMA model is characterized by 3 terms: $p, d, q$ where

$p$: the order of the Auto Regressive (AR) term

$q$: the order of the Moving Average (MA) term

$d$: the number of nonseasonal differences needed for stationarity

The ARIMA process is defined as:

$$X_t = c + \epsilon_t + \sum_{i=1}^{p} \phi_i X_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j}$$

$\phi_i$ *is AR parameters;*
$\theta_j$ *is MA parameters*

**ARIMA model process:**

a) Based on the scatter plot, autocorrelation function and partial autocorrelation function plots of the time series, the variance and trend of the series are identified with an ADF unit root test. Generally speaking, the time series of stock price data is not a stationary series.

b) Smoothing of non-stationary series. If the data series is non-stationary and has a certain increasing or decreasing trend, the data needs to be differenced.

c) According to the identification rules of the time series model, build the corresponding model If the bias correlation function and autocorrelation function of the smooth series are both trailing, the series is suitable for the ARIMA model.

d) Perform parameter estimation and test for statistical significance.

e) Perform a hypothesis test to diagnose whether the residual series is white noise.

f) Perform predictive analysis using the tested model.

**LSTM Model Architecture:**

A long short-term memory (LSTM) is a type of Recurrent Neural Network (RNN) specially designed to prevent the neural network output for a given input from either decaying or exploding as it cycles through the feedback loops. Memory of past input is critical for solving sequence learning tasks and Long short-term memory networks provide better performance than other Neural Networks.

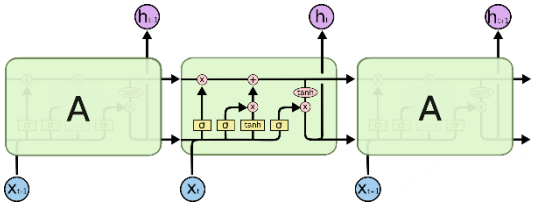



*Figure 1 Each layer in LSTM*

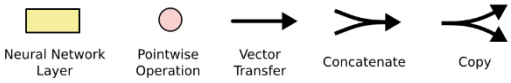



*Figure 2 Notations in LSTM*

LSTM Architecture consists of linear units and its neural network layers interact in a special way. In each layer, 4 different neural network layers make up on cell and in each cell, it has three inputs: $C_{t-1}, h_{t-1}$ and $x_t$ those are the states of last cell and the input $x$ respectively. Then each cell output 2 values, $C_t$ and $h_t$. They are both the state of this cell. When receiving $C_{t-1}$, $h_{t-1}$ and $x_t$, it passes through three gates (forget gate, input gate and output gate).

a) For F gate, which is the forget gate. In this gate, we decide what information we're going to throw away from the cell state.

Determine how much the unit status of the previous moment $C_{t-1}$ is retained to the current moment $C_t$.

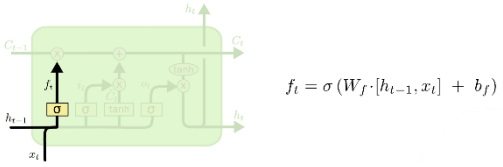
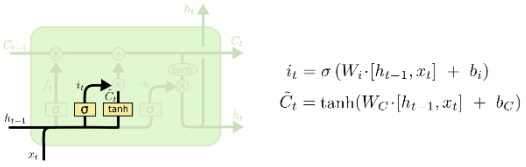



*Figure 3 Structure of Forget gate*

When input $x_t$ and $h_{t-1}$, the model merges the two values and process through the sigmoid function. The calculated value is then combined with another input $C_{t-1}$. Through this process, the cell can choose to forget unimportant information in the state. Selectively forgetting the information

from the previous cell.

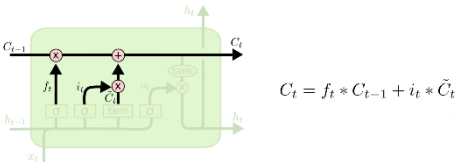
b) For input gate, which decides which values are updated. Next, a tanh layer and sigmoid layer create a vectors of new candidate values. In the next step, combine these two to create an update to the state.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

*Figure 4 Structure of Input gate*

Then, drop the information which need to forget and add the new information. By this way, update the state of this cell

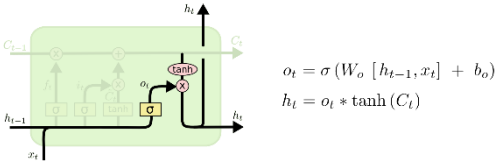


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

*Figure 5 After update the cell*

Inputs are $C_{t-1}, h_{t-1}$ and $x_t$ and layers are sigmoid, tanh. For now, update the state of this cell.

c) For output gate, need to decide what we're going to output and it depend on the state of this cell. First, we process a sigmoid function, then we put the state through tanh and combine together. Finally, we output the value.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

*Figure 6 Structure of Input gate*

**Our LSTM model structure:**

For this model, which contains 5 LSTM layers, 5 dropout layers and 2 dense layers. For LSTM layers, the first layer is to deal with input, the rest is in order to extract the information. For dropout layers, to prevent overfitting, we add dropout layers between each 2 of other layers. At last, add dense layers to output values.

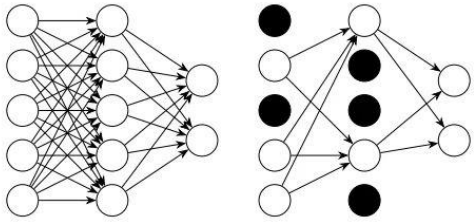


*Figure 7 Dropout Layers*

**Parameters:**

In the model, set parameters and hyper parameters.

| Parameter | Explain |
|---|---|
| start_time | The start date of the training set |
| window | Window time for each row of data |
| timestamp | Training set Test Set Division Time (last n days) |
| epochs | Training set Number of training sessions |
| batch_size | the number of data samples captured in a training |

## Data Description and Preprocessing

**Data Description:**

When searching for stock data and the news from the National Association of Securities Dealers Automated Quotations (NASDAQ), we noticed that the current value of NVIDIA's stock is similar to the value of the stock around 2017-2018, when the stock went up due to the Cryptocurrency mining by GPUs. And in recent years, cryptocurrency becomes popular as well, which makes us pay more focused to NVIDIA's stock price.

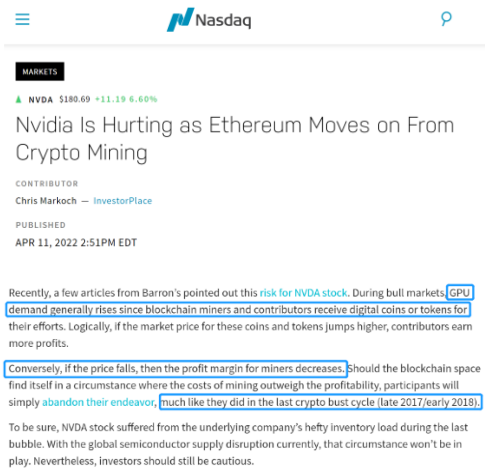



*Figure 8 NVIDIA's news on Nasdaq*

Hence, we select Nvidia stock data from (https://www.nasdaq.com/) NASDAQ official website, which is the world's largest stock market.

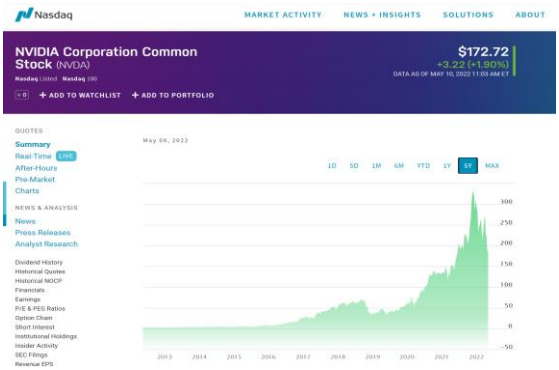



*Figure 9 NVIDIA's stock data on Nasdaq*

Select Nvidia's historical quotes. Download and process stock data from 1999-01-22 to 2021-11-12.

**Data preprocessing:**

After imported dataset, we got data that was started in 1999-01-22.

| Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|
| 1999-01-22 | 0.401941 | 0.448595 | 0.356484 | 0.376820 | 271468800.0 |
| 1999-01-25 | 0.406726 | 0.421081 | 0.376820 | 0.416296 | 51048000.0 |
| 1999-01-26 | 0.421081 | 0.429455 | 0.378016 | 0.383998 | 34320000.0 |
| 1999-01-27 | 0.385194 | 0.394764 | 0.363661 | 0.382801 | 24436800.0 |
| 1999-01-28 | 0.382801 | 0.385194 | 0.379212 | 0.381605 | 22752000.0 |
| ... | ... | ... | ... | ... | ... |
| 2021-11-08 | 301.489990 | 311.000000 | 299.070007 | 308.040009 | 50310100.0 |
| 2021-11-09 | 322.820007 | 323.100006 | 299.640015 | 306.570007 | 64674600.0 |
| 2021-11-10 | 293.559998 | 308.500000 | 287.779999 | 294.589996 | 63620600.0 |
| 2021-11-11 | 304.679993 | 305.899994 | 297.769989 | 303.899994 | 33217200.0 |
| 2021-11-12 | 300.100006 | 306.799988 | 296.299988 | 303.899994 | 41215100.0 |

*Figure 10 original dataset*

Hence, select the close price to do the data training since the close price can represent a more truly price of a company.

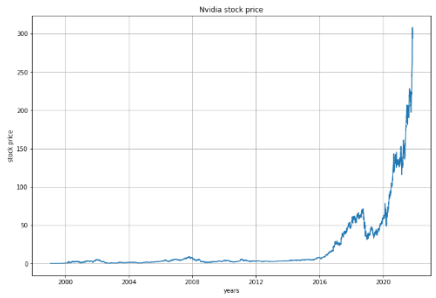



*Figure 11 Plot for NVIDIA price*

Then, we do the min-max normalization for data. The function of min-max normalization is

$$x' = \frac{x - min}{max - min}$$

So that got regularized data

```
Date
1999-01-22    0.000206
1999-01-25    0.000334
1999-01-26    0.000229
1999-01-27    0.000225
1999-01-28    0.000222
                ...
2021-11-08    1.000000
2021-11-09    0.995223
2021-11-10    0.956292
2021-11-11    0.986546
2021-11-12    0.986546
Name: value, Length: 5743, dtype: float64
```

*Figure 12 after min-max normalization*

Found that stock price of NVIDIA before 2015 can extract little information. So that we selected data that is from 2015-01-01.

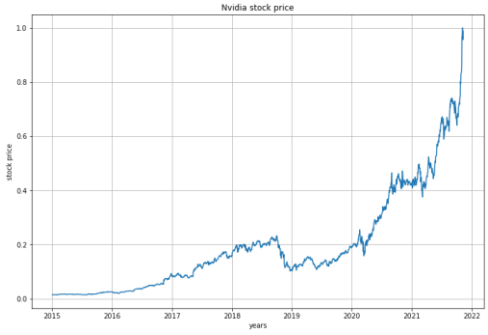



*Figure 13 NVIDIA price since 2015-01-01*

We set a window of 5 days, which means that for each piece of data, we train based on the stock price of the previous 5 days.

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | [0.014709955… | [0.01444429… | [0.013975492… | [0.01393642… | [0.01449… |
| 1 | [0.01444429… | [0.01397549… | [0.013936425… | [0.01449899… | [0.01456… |
| 2 | [0.013975492… | [0.01393642… | [0.014498995… | [0.01456149… | [0.01436… |
| 3 | [0.013936425… | [0.01449899… | [0.014561499… | [0.01436616… | [0.01434… |
| 4 | [0.01449899… | [0.01456149… | [0.014366163… | [0.01434272… | [0.01440… |
| 5 | [0.014561499… | [0.01436616… | [0.014342723… | [0.01440522… | [0.01429… |
| 6 | [0.014366163… | [0.01434272… | [0.014405225… | [0.01429584… | [0.01457… |
| 7 | [0.014342723… | [0.01440522… | [0.014295843… | [0.01457712… | [0.01462… |
| 8 | [0.014405225… | [0.01429584… | [0.014577126… | [0.01462400… | [0.01484… |
| 9 | [0.014295843… | [0.01457712… | [0.014624006… | [0.01484278… | [0.01511… |
| 10 | [0.014577126… | [0.01462400… | [0.014842784… | [0.01511625… | [0.01516… |
| 11 | [0.014624006… | [0.01484278… | [0.015116251… | [0.01516313… | [0.01509… |
| 12 | [0.014842784… | [0.01511625… | [0.015163136… | [0.01509281… | [0.01431… |
| 13 | [0.015116251… | [0.01516313… | [0.015092816… | [0.01431928… | [0.01406… |
| 14 | [0.015163136… | [0.01509281… | [0.014319283… | [0.01406925… | [0.01443… |
| 15 | [0.015092816… | [0.01431928… | [0.014069257… | [0.01443648… | [0.01398… |

*Figure 14 Training set*

After data preprocessing, we got the 1615 training data and 100 test data.

```
train_x (1624, 5, 1)
train_y (1624,)
test_x  (100, 5, 1)
test_y  (100,)
```

*Figure 15 Quantity of data*

## Model Evaluation and Prediction

**For ARIMA:**

**Model construction**

Autoregressive process assumes that the observation at previous several time steps are useful to predict the next step. It depends on autocorrelation. We plotted the observation at the previous time step (t) with the observation at the next time step (t+1) as a scatter plot in the following figure, which clearly shows a relationship or some correlation.

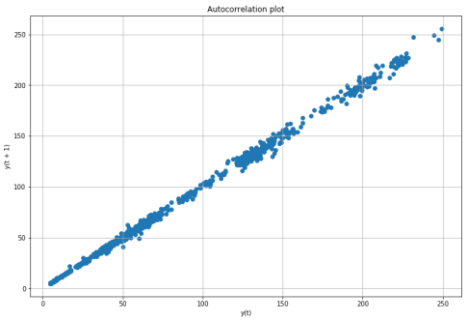



*Figure 16 autocorrelation plot*

From the data plot, the price has the trend of increasing, so this time series data is not stationary. We take a series of one order difference to make sure the data become stationary.

Do Dickey-Fuller Test on the differenced data and it showed that in 90% confidence level. The data after first order difference is stationary, and average change, show in figure below, is not large, so the data can be considered stable.

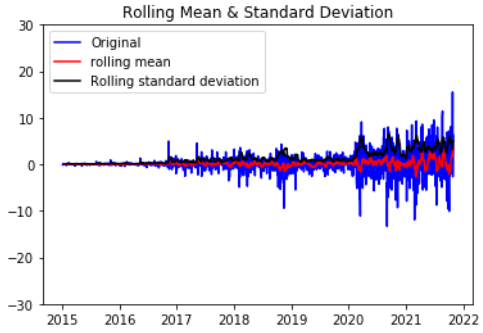



*Figure 17 rolling mean & standard deviation*

In ARIMA process, autocorrelation function and partial autocorrelation function could be used to determine parameter p and q of the model.

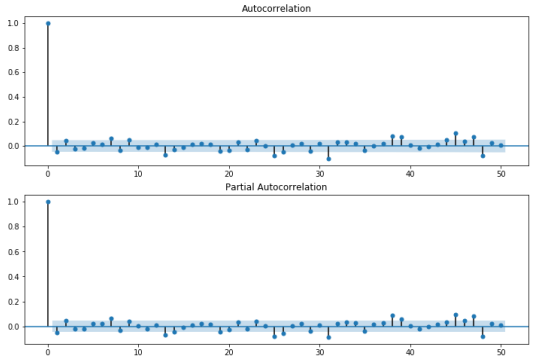



*Figure 18 autocorrelation & partial autocorrelation*

Based on Bayesian Information Criterion (BIC), use the tool in statsmodels package, BIC min order of (p, q) is (4, 2). We choose to use the Mean Square Error (MSE), Root Mean Square Error (RMSE), Mean Absolute Error (MAE) to evaluate our models.

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

$$RMSE(X,h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(x_i) - y_i)^2}$$

$$\text{MAE}\ (X,h) = \frac{1}{m}\sum_{i=1}^{m}|h(x_i) - y_i|$$

The model performance on test set shown as follow:

| **ARIMA（4, 1 ,2 ）** | |
|---|---|
| MSE | 22.388 |
| MAE | 3.647 |
| RMSE | 4.731 |

*Table 1 model evaluation*

**Residual white noise test:**

A white noise test is performed on the residual series of the ARIMA model. If the residuals are white noise series, it means that there is no more information that can be mined. The results of visualizing the residuals of the model are as follows.

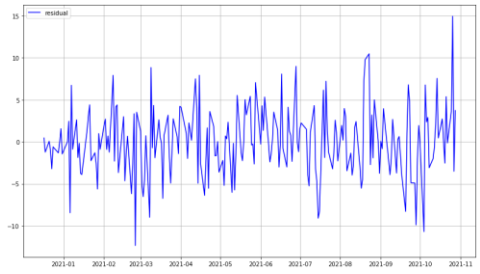



*Figure 19 model residual*

Here we use the ljung-box test. all p-values are greater than the significance level (e.g. 0.05) and the original hypothesis (that the series is a white noise series) cannot be rejected and the series can be considered as a white noise series. Therefore, there is no need to continue adding models.

```
[0.93754653 0.99677313 0.99954248 0.99996496 0.99999801 0.90740554
 0.93463142 0.96591488 0.98272858 0.9895207  0.99388265 0.98942624
 0.98311467 0.97617098 0.97419876 0.95521277 0.96300452 0.97545711
 0.83655811 0.81633406 0.84914263 0.88443476 0.80500401 0.837157
 0.53784668 0.45231719 0.4135847  0.44910952 0.48283134 0.50080892
 0.23252594 0.26557014 0.29764269 0.30195056 0.33892936 0.378934
 0.42368698 0.38006644 0.39592104 0.43699012]
```

*Table 2 ljung-box test, p-value*

**ARIMA Evaluation and Prediction:**

Finally, we use ARIMA (4, 1, 2) model to predict the stock price in last 90 days. The predicted price result as follow:

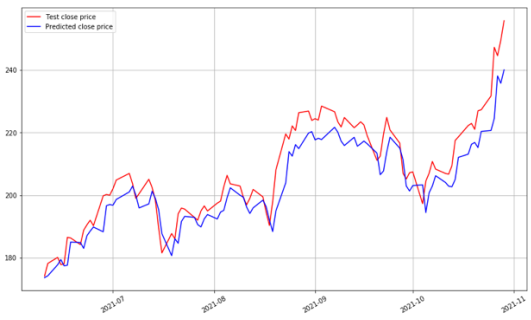



*Figure 20 ARIMA (4,1, 2) price prediction*

| **ARIMA（4, 1 ,2 ）** | |
|---|---|
| MSE | 27.832 |
| MAE | 6.654 |
| RMSE | 9.471 |

*Table 3 ARIMA model evaluation on test set*

**For LSTM:**

We train the model with epoch is 100, batch size is 512, optimizer is “Adam” and loss function is MSE.

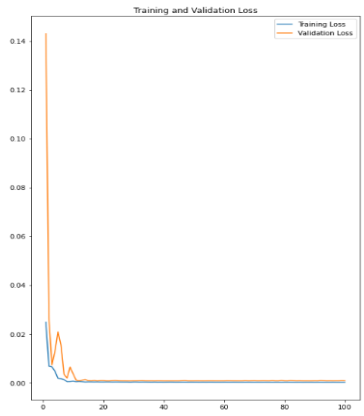



*Figure 21 Loss during training*

After training, we restore the values to that before min-max normalization. Then, compared with training set and test set and got the RMSE function.

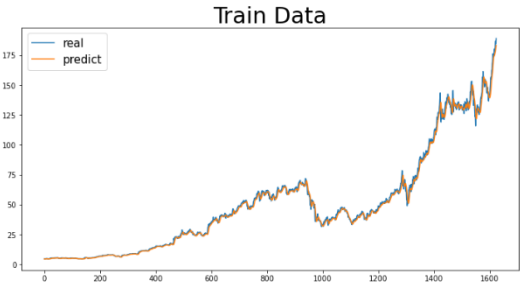



*Figure 22 Compared with training data*

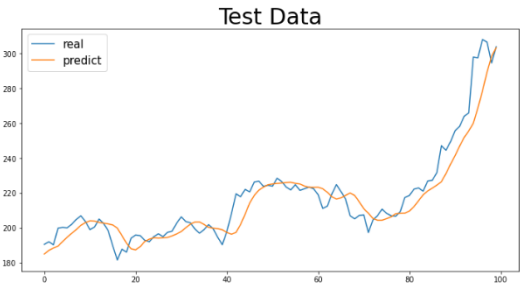



*Figure 23 Loss during predicting*

We choose to use the MSE, RMSE, MAE to evaluate our models. Here is our LSTM model results for different windows and eopch.

| Window | Epoch | MSE | RMSE | MAE |
|---|---|---|---|---|
| 5 | 50 | 92.791 | 9.633 | 7.48 |
| 5 | 100 | 84.992 | 9.219 | 5.519 |
| 10 | 50 | 114.404 | 10.696 | 7.663 |
| 10 | 100 | 133.182 | 11.540 | 8.206 |
| 20 | 50 | 115.178 | 10.732 | 7.717 |
| 20 | 100 | 117.007 | 10.817 | 7.96 |

*Table 4 LSTM model evaluation on test set*

## Conclusion

Through the analysis of the establishment process and results of these two models, conclusions can be made. The stock prediction of the LSTM model is better than that of the ARIMA model. And ARIMA can further improve the accuracy of the ARIMA model if the residual series of the white noise sequence exists.

The disadvantage is that, as we all know, the fluctuation of stock prices is not only related to changes in time, but also related to economic factors, socio-political factors, and the listing of other stocks. These two models are essentially deduced by using possible relationships in the time series without considering other external factors. This is also the direction in which future research can be further in-depth. Of course, the further development and use of LSTM model in stock price prediction is also a subject of research value.

## Group Work on GitHub Website:

https://github.com/ZhengWugeng/Financial_Computing_Group

## References


[1] Long Short-Term Memory (LSTM). (2020, February 21). NVIDIA Developer. https://developer.nvidia.com/discover/lstm#:%7E:text=A%20Long%20shortterm%20memory%20%28LSTM%29%20is%20a%20type,better%20at%20pattern%20recognition%20than%20other%20neural%20networks.

[2] Olah, C. (27–08-15). U*nderstanding LSTM Networks* -- colah's blog. Understanding LSTM Networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[3] Team, K. (2022, February 3). Keras documentation: LSTM layer. LSTM-Keras. https://keras.io/api/layers/recurrent_layers/lstm/

[4] Mehtab, S., Sen, J., & Dutta, A. (2020, October). Stock price prediction using machine learning and LSTM-based deep learning models. In *Symposium on Machine Learning and Metaheuristics Algorithms, and Applications* (pp. 88-106). Springer, Singapore.

[5] Ariyo, A. A., Adewumi, A. O., & Ayo, C. K. (2014, March). Stock price prediction using the ARIMA model. In *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation* (pp. 106-112). IEEE.

[6] Munish Kumar, Surbhi Gupta, Krishan Kumar, and Monika Sachdeva. 2020. *SPREADING OF COVID-19 IN INDIA, ITALY, JAPAN, SPAIN, UK, US: A Prediction Using ARIMA and LSTM Model.* Digit. Gov.: Res. Pract. 1, 4, Article 24 (October 2020), 9 pages. https://doi.org/10.1145/3411760

[7] Newbold, P. (1983). *ARIMA Model Building and the Time Series Analysis Approach to Forecasting. Journal of Forecasting,* 2, 23-35. https://link.gale.com/apps/doc/A2582223/AONE?u=uicbnu&sid=bookmark-AONE&xid=12750cc3

# Appendix 1: LSTM Model

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [ ]:
```
data = pd.read_csv("NVidia_stock_history.csv", index_col=0)
data = data.iloc[:, :5]
data.index = pd.to_datetime(data.index, format="%Y/%m/%d")
data
```

Out[ ]:

| Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|
| **1999-01-22** | 0.401941 | 0.448595 | 0.356484 | 0.376820 | 271468800.0 |
| **1999-01-25** | 0.406726 | 0.421081 | 0.376820 | 0.416296 | 51048000.0 |
| **1999-01-26** | 0.421081 | 0.429455 | 0.378016 | 0.383998 | 34320000.0 |
| **1999-01-27** | 0.385194 | 0.394764 | 0.363661 | 0.382801 | 24436800.0 |
| **1999-01-28** | 0.382801 | 0.385194 | 0.379212 | 0.381605 | 22752000.0 |
| **...** | ... | ... | ... | ... | ... |
| **2021-11-08** | 301.489990 | 311.000000 | 299.070007 | 308.040009 | 50310100.0 |
| **2021-11-09** | 322.820007 | 323.100006 | 299.640015 | 306.570007 | 64674600.0 |
| **2021-11-10** | 293.559998 | 308.500000 | 287.779999 | 294.589996 | 63620600.0 |
| **2021-11-11** | 304.679993 | 305.899994 | 297.769989 | 303.899994 | 33217200.0 |
| **2021-11-12** | 300.100006 | 306.799988 | 296.299988 | 303.899994 | 41215100.0 |

5743 rows × 5 columns

In [ ]:
```
data_model = data['Close']
print(data_model)
```

```
Date
1999-01-22      0.376820
1999-01-25      0.416296
1999-01-26      0.383998
1999-01-27      0.382801
1999-01-28      0.381605
                  ...
2021-11-08    308.040009
2021-11-09    306.570007
2021-11-10    294.589996
2021-11-11    303.899994
2021-11-12    303.899994
Name: Close, Length: 5743, dtype: float64
```

In [ ]:
```
plt.figure(figsize=(12, 8))
plt.grid()
plt.plot(data_model)
plt.xlabel('years')
```

```
plt.ylabel('stock price')
plt.title("Nvidia stock price")
plt.show()
```

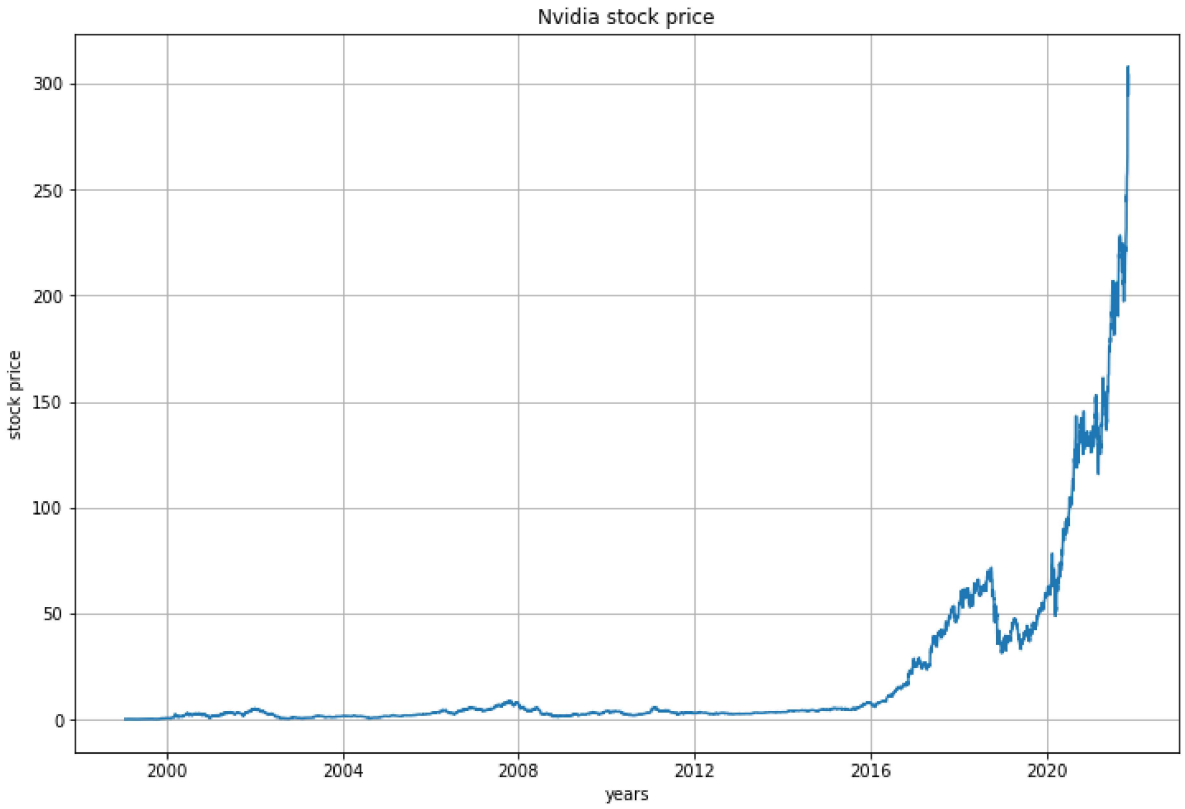
Out[ ]: <Figure size 864x576 with 0 Axes>

Out[ ]: [<matplotlib.lines.Line2D at 0x149c672ffa0>]

Out[ ]: Text(0.5, 0, 'years')

Out[ ]: Text(0, 0.5, 'stock price')

Out[ ]: Text(0.5, 1.0, 'Nvidia stock price')




In [ ]:
```
# Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range=(0, 1))
data_set_scaled = sc.fit_transform(pd.DataFrame(data_model))
data_set_df = pd.DataFrame(
    data_set_scaled, index=data_model.index, columns=["value"])
data_set_df['value']
```

Out[ ]:
```
Date
1999-01-22    0.000206
1999-01-25    0.000334
1999-01-26    0.000229
1999-01-27    0.000225
1999-01-28    0.000222
                ...
2021-11-08    1.000000
2021-11-09    0.995223
2021-11-10    0.956292
2021-11-11    0.986546
2021-11-12    0.986546
Name: value, Length: 5743, dtype: float64
```

In [ ]:
```
plt.figure(figsize=(12, 8))
plt.grid()
plt.xlabel('years')
```

```
plt.ylabel('stock price')
plt.title("Nvidia stock price")
plt.plot(data_set_df['value'])
```

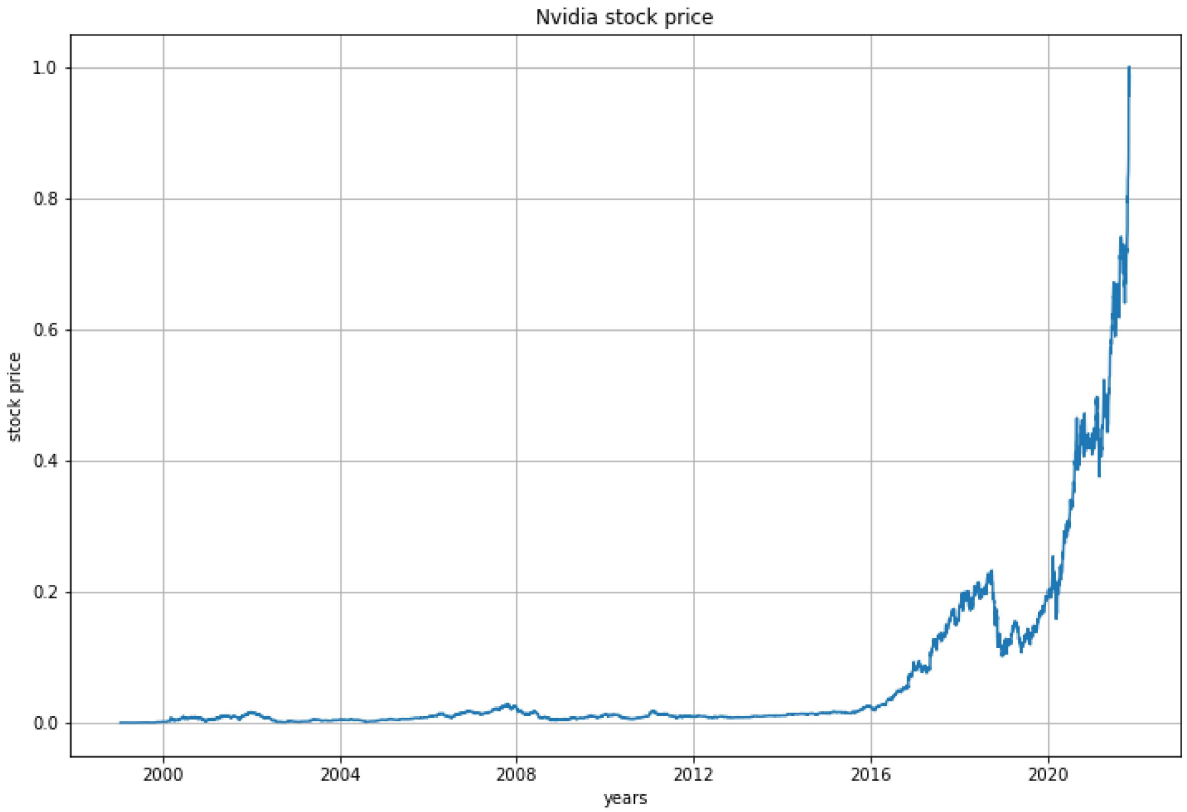
Out[ ]: <Figure size 864x576 with 0 Axes>

Out[ ]: Text(0.5, 0, 'years')

Out[ ]: Text(0, 0.5, 'stock price')

Out[ ]: Text(0.5, 1.0, 'Nvidia stock price')

Out[ ]: [<matplotlib.lines.Line2D at 0x149c8fb81c0>]




In [ ]:
```
start_time = pd.to_datetime("2015/01/01", format="%Y/%m/%d")
data_set_df = data_set_df.loc[start_time:]
data_set_df.shape
```

Out[ ]: (1730, 1)

In [ ]:
```
plt.figure(figsize=(12, 8))
plt.grid()
plt.plot(data_set_df)
plt.xlabel('years')
plt.ylabel('stock price')
plt.title("Nvidia stock price")
plt.show()
```

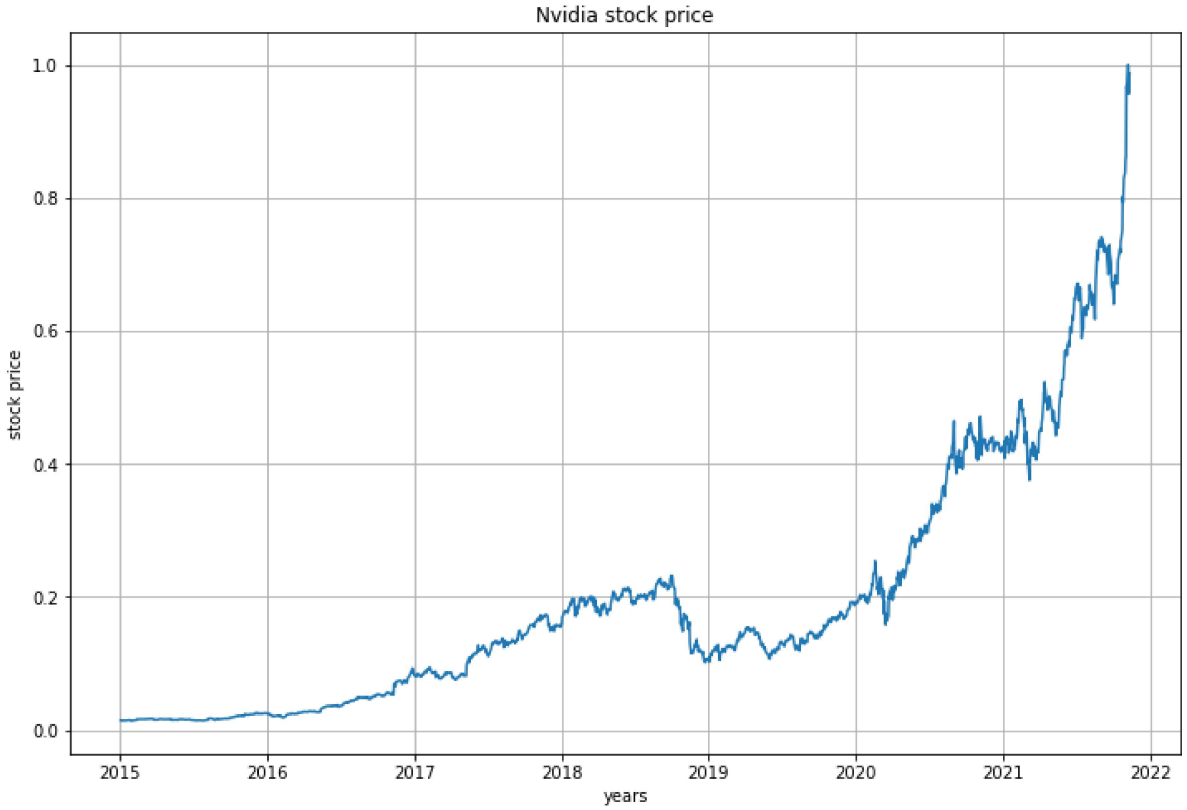
Out[ ]: <Figure size 864x576 with 0 Axes>

Out[ ]: [<matplotlib.lines.Line2D at 0x149c90423d0>]

Out[ ]: Text(0.5, 0, 'years')

Out[ ]: Text(0, 0.5, 'stock price')

Out[ ]: Text(0.5, 1.0, 'Nvidia stock price')




In [ ]:
```
# initial training set
window = 5
timestamp = 100
amount_of_features = len(data_set_df.columns)

data = data_set_df.values
sequence_length = window + 1
sample = []

for index in range(len(data) - sequence_length):
    sample.append(data[index: index + sequence_length])
result = np.array(sample)

x = result[:-timestamp, :]
train_x = x[:, :-1]
train_y = x[:, -1][:, -1]
test_x = result[-timestamp:, :-1]
test_y = result[-timestamp:, -1][:, -1]
train_x = np.reshape(
    train_x, (train_x.shape[0], train_x.shape[1], amount_of_features))
test_x = np.reshape(
    test_x, (test_x.shape[0], test_x.shape[1], amount_of_features))
```

In [ ]:
```
print("train_x", train_x.shape)
print("train_y", train_y.shape)
print("test_x ", test_x.shape)
print("test_y ", test_y.shape)
```

```
train_x (1624, 5, 1)
train_y (1624,)
test_x  (100, 5, 1)
test_y  (100,)
```

In [ ]:
```
# Building a function Recurrent Neural Network with Keras
# Dropout is being used to prevent overfitting

from keras.models import Sequential
```

```
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
```

In [ ]:
```
# parameters setting
epochs = 100
batch_size = 512
```

In [ ]:
```
# Initialisation
# Adding layers and definig the model
# this model has 5 hidden layers

# Initialisation
reg = Sequential()

# Layer 1
reg.add(LSTM(units=200, return_sequences=True,
        input_shape=(train_x.shape[1], 1)))
reg.add(Dropout(0.2))
# Layer 2
reg = Sequential()
reg.add(LSTM(units=200, return_sequences=True))
reg.add(Dropout(0.2))
# Layer 3
reg = Sequential()
reg.add(LSTM(units=100, return_sequences=True))
reg.add(Dropout(0.2))
# Layer 4
reg = Sequential()
reg.add(LSTM(units=100, return_sequences=True))
reg.add(Dropout(0.2))
# Layer 5
reg = Sequential()
reg.add(LSTM(units=100))
reg.add(Dropout(0.2))
reg.add(Dense(4))
# Final Output layer
reg.add(Dense(units=1, activation='relu'))
```

In [ ]:
```
# Compiling our neural network by choosing our loss functions and optimizer
# Adam optimizer
reg.compile(optimizer='adam', loss='mse')
```

In [ ]:
```
# Training the model on the training data
history = reg.fit(x=train_x, y=train_y, epochs=epochs,
        batch_size=batch_size, validation_data=(test_x, test_y))
```

```
Epoch 1/100
4/4 [==============================] - 4s 105ms/step - loss: 0.0377 - val_loss: 0.29
29
Epoch 2/100
4/4 [==============================] - 0s 10ms/step - loss: 0.0151 - val_loss: 0.108
5
Epoch 3/100
4/4 [==============================] - 0s 11ms/step - loss: 0.0073 - val_loss: 0.027
8
Epoch 4/100
4/4 [==============================] - 0s 10ms/step - loss: 0.0080 - val_loss: 0.016
7
Epoch 5/100
4/4 [==============================] - 0s 11ms/step - loss: 0.0049 - val_loss: 0.024
2
Epoch 6/100
4/4 [==============================] - 0s 11ms/step - loss: 0.0022 - val_loss: 0.028
2
Epoch 7/100
4/4 [==============================] - 0s 11ms/step - loss: 0.0019 - val_loss: 0.018
6
Epoch 8/100
4/4 [==============================] - 0s 11ms/step - loss: 0.0014 - val_loss: 0.004
6
Epoch 9/100
4/4 [==============================] - 0s 10ms/step - loss: 4.8951e-04 - val_loss:
0.0015
Epoch 10/100
4/4 [==============================] - 0s 11ms/step - loss: 5.1768e-04 - val_loss:
0.0061
Epoch 11/100
4/4 [==============================] - 0s 11ms/step - loss: 5.5565e-04 - val_loss:
0.0047
Epoch 12/100
4/4 [==============================] - 0s 12ms/step - loss: 4.0632e-04 - val_loss:
0.0023
Epoch 13/100
4/4 [==============================] - 0s 11ms/step - loss: 4.6031e-04 - val_loss:
0.0018
Epoch 14/100
4/4 [==============================] - 0s 11ms/step - loss: 4.4458e-04 - val_loss:
0.0022
Epoch 15/100
4/4 [==============================] - 0s 12ms/step - loss: 3.6524e-04 - val_loss:
0.0021
Epoch 16/100
4/4 [==============================] - 0s 11ms/step - loss: 3.4597e-04 - val_loss:
0.0011
Epoch 17/100
4/4 [==============================] - 0s 11ms/step - loss: 3.4610e-04 - val_loss:
8.8059e-04
Epoch 18/100
4/4 [==============================] - 0s 12ms/step - loss: 3.6608e-04 - val_loss:
9.0343e-04
Epoch 19/100
4/4 [==============================] - 0s 12ms/step - loss: 3.1835e-04 - val_loss:
8.8997e-04
Epoch 20/100
4/4 [==============================] - 0s 11ms/step - loss: 3.2745e-04 - val_loss:
9.3575e-04
Epoch 21/100
4/4 [==============================] - 0s 11ms/step - loss: 3.2492e-04 - val_loss:
8.7274e-04
Epoch 22/100
```

```
4/4 [==============================] - 0s 11ms/step - loss: 2.9823e-04 - val_loss:
8.8658e-04
Epoch 23/100
4/4 [==============================] - 0s 10ms/step - loss: 3.5274e-04 - val_loss:
0.0011
Epoch 24/100
4/4 [==============================] - 0s 11ms/step - loss: 2.8232e-04 - val_loss:
0.0011
Epoch 25/100
4/4 [==============================] - 0s 10ms/step - loss: 3.0209e-04 - val_loss:
0.0010
Epoch 26/100
4/4 [==============================] - 0s 12ms/step - loss: 3.3700e-04 - val_loss:
0.0010
Epoch 27/100
4/4 [==============================] - 0s 10ms/step - loss: 3.2379e-04 - val_loss:
0.0011
Epoch 28/100
4/4 [==============================] - 0s 11ms/step - loss: 3.3678e-04 - val_loss:
0.0011
Epoch 29/100
4/4 [==============================] - 0s 11ms/step - loss: 2.8618e-04 - val_loss:
9.2046e-04
Epoch 30/100
4/4 [==============================] - 0s 10ms/step - loss: 3.0624e-04 - val_loss:
8.9755e-04
Epoch 31/100
4/4 [==============================] - 0s 11ms/step - loss: 3.0974e-04 - val_loss:
9.1552e-04
Epoch 32/100
4/4 [==============================] - 0s 10ms/step - loss: 3.0951e-04 - val_loss:
9.8311e-04
Epoch 33/100
4/4 [==============================] - 0s 10ms/step - loss: 2.9292e-04 - val_loss:
0.0011
Epoch 34/100
4/4 [==============================] - 0s 11ms/step - loss: 3.0766e-04 - val_loss:
0.0010
Epoch 35/100
4/4 [==============================] - 0s 10ms/step - loss: 3.1330e-04 - val_loss:
9.4362e-04
Epoch 36/100
4/4 [==============================] - 0s 11ms/step - loss: 2.6832e-04 - val_loss:
8.9616e-04
Epoch 37/100
4/4 [==============================] - 0s 10ms/step - loss: 2.6567e-04 - val_loss:
9.3265e-04
Epoch 38/100
4/4 [==============================] - 0s 10ms/step - loss: 2.7495e-04 - val_loss:
9.3432e-04
Epoch 39/100
4/4 [==============================] - 0s 10ms/step - loss: 2.4705e-04 - val_loss:
8.8020e-04
Epoch 40/100
4/4 [==============================] - 0s 10ms/step - loss: 3.2694e-04 - val_loss:
8.7285e-04
Epoch 41/100
4/4 [==============================] - 0s 10ms/step - loss: 2.5916e-04 - val_loss:
9.1437e-04
Epoch 42/100
4/4 [==============================] - 0s 10ms/step - loss: 2.9702e-04 - val_loss:
9.5992e-04
Epoch 43/100
4/4 [==============================] - 0s 10ms/step - loss: 2.5958e-04 - val_loss:
```

9.4579e-04
Epoch 44/100
4/4 [==============================] - 0s 10ms/step - loss: 2.8560e-04 - val_loss:
9.9252e-04
Epoch 45/100
4/4 [==============================] - 0s 11ms/step - loss: 2.6327e-04 - val_loss:
9.6681e-04
Epoch 46/100
4/4 [==============================] - 0s 10ms/step - loss: 2.6064e-04 - val_loss:
8.9181e-04
Epoch 47/100
4/4 [==============================] - 0s 10ms/step - loss: 2.6608e-04 - val_loss:
8.7661e-04
Epoch 48/100
4/4 [==============================] - 0s 9ms/step - loss: 2.7611e-04 - val_loss: 8.
9319e-04
Epoch 49/100
4/4 [==============================] - 0s 10ms/step - loss: 2.8630e-04 - val_loss:
8.8783e-04
Epoch 50/100
4/4 [==============================] - 0s 9ms/step - loss: 2.5993e-04 - val_loss: 8.
7596e-04
Epoch 51/100
4/4 [==============================] - 0s 11ms/step - loss: 2.3367e-04 - val_loss:
8.8317e-04
Epoch 52/100
4/4 [==============================] - 0s 11ms/step - loss: 2.6625e-04 - val_loss:
9.1584e-04
Epoch 53/100
4/4 [==============================] - 0s 10ms/step - loss: 2.6847e-04 - val_loss:
8.9646e-04
Epoch 54/100
4/4 [==============================] - 0s 11ms/step - loss: 2.4650e-04 - val_loss:
9.4243e-04
Epoch 55/100
4/4 [==============================] - 0s 10ms/step - loss: 3.0667e-04 - val_loss:
8.8197e-04
Epoch 56/100
4/4 [==============================] - 0s 10ms/step - loss: 2.2519e-04 - val_loss:
9.2543e-04
Epoch 57/100
4/4 [==============================] - 0s 11ms/step - loss: 2.6708e-04 - val_loss:
8.7939e-04
Epoch 58/100
4/4 [==============================] - 0s 9ms/step - loss: 2.3833e-04 - val_loss: 8.
7884e-04
Epoch 59/100
4/4 [==============================] - 0s 10ms/step - loss: 2.3373e-04 - val_loss:
9.4191e-04
Epoch 60/100
4/4 [==============================] - 0s 10ms/step - loss: 2.4981e-04 - val_loss:
9.2360e-04
Epoch 61/100
4/4 [==============================] - 0s 10ms/step - loss: 2.6134e-04 - val_loss:
8.9170e-04
Epoch 62/100
4/4 [==============================] - 0s 10ms/step - loss: 2.3122e-04 - val_loss:
8.8182e-04
Epoch 63/100
4/4 [==============================] - 0s 11ms/step - loss: 2.3460e-04 - val_loss:
9.0881e-04
Epoch 64/100
4/4 [==============================] - 0s 10ms/step - loss: 2.4182e-04 - val_loss:
8.8567e-04

```
Epoch 65/100
4/4 [==============================] - 0s 10ms/step - loss: 2.4892e-04 - val_loss:
8.9761e-04
Epoch 66/100
4/4 [==============================] - 0s 10ms/step - loss: 2.1171e-04 - val_loss:
9.1620e-04
Epoch 67/100
4/4 [==============================] - 0s 10ms/step - loss: 2.8253e-04 - val_loss:
8.8918e-04
Epoch 68/100
4/4 [==============================] - 0s 12ms/step - loss: 2.6419e-04 - val_loss:
8.8307e-04
Epoch 69/100
4/4 [==============================] - 0s 10ms/step - loss: 2.5053e-04 - val_loss:
9.1073e-04
Epoch 70/100
4/4 [==============================] - 0s 10ms/step - loss: 2.3481e-04 - val_loss:
9.0088e-04
Epoch 71/100
4/4 [==============================] - 0s 10ms/step - loss: 2.4118e-04 - val_loss:
8.9221e-04
Epoch 72/100
4/4 [==============================] - 0s 11ms/step - loss: 2.1644e-04 - val_loss:
8.9081e-04
Epoch 73/100
4/4 [==============================] - 0s 9ms/step - loss: 2.2624e-04 - val_loss: 8.
8910e-04
Epoch 74/100
4/4 [==============================] - 0s 10ms/step - loss: 2.2436e-04 - val_loss:
8.9477e-04
Epoch 75/100
4/4 [==============================] - 0s 11ms/step - loss: 2.6315e-04 - val_loss:
8.8779e-04
Epoch 76/100
4/4 [==============================] - 0s 13ms/step - loss: 2.1088e-04 - val_loss:
8.8625e-04
Epoch 77/100
4/4 [==============================] - 0s 9ms/step - loss: 2.3041e-04 - val_loss: 8.
8213e-04
Epoch 78/100
4/4 [==============================] - 0s 9ms/step - loss: 2.1473e-04 - val_loss: 8.
9180e-04
Epoch 79/100
4/4 [==============================] - 0s 10ms/step - loss: 2.1884e-04 - val_loss:
9.0536e-04
Epoch 80/100
4/4 [==============================] - 0s 9ms/step - loss: 2.0041e-04 - val_loss: 8.
8676e-04
Epoch 81/100
4/4 [==============================] - 0s 11ms/step - loss: 2.3277e-04 - val_loss:
8.9473e-04
Epoch 82/100
4/4 [==============================] - 0s 11ms/step - loss: 2.2147e-04 - val_loss:
8.9979e-04
Epoch 83/100
4/4 [==============================] - 0s 10ms/step - loss: 2.0699e-04 - val_loss:
9.4844e-04
Epoch 84/100
4/4 [==============================] - 0s 10ms/step - loss: 2.3508e-04 - val_loss:
8.8928e-04
Epoch 85/100
4/4 [==============================] - 0s 10ms/step - loss: 2.0400e-04 - val_loss:
9.2299e-04
Epoch 86/100
```

```
4/4 [==============================] - 0s 10ms/step - loss: 1.8994e-04 - val_loss:
0.0010
Epoch 87/100
4/4 [==============================] - 0s 10ms/step - loss: 2.0243e-04 - val_loss:
8.8857e-04
Epoch 88/100
4/4 [==============================] - 0s 10ms/step - loss: 1.9965e-04 - val_loss:
8.8989e-04
Epoch 89/100
4/4 [==============================] - 0s 10ms/step - loss: 2.0940e-04 - val_loss:
9.3547e-04
Epoch 90/100
4/4 [==============================] - 0s 10ms/step - loss: 2.1488e-04 - val_loss:
8.9169e-04
Epoch 91/100
4/4 [==============================] - 0s 11ms/step - loss: 2.0815e-04 - val_loss:
8.8609e-04
Epoch 92/100
4/4 [==============================] - 0s 10ms/step - loss: 1.8297e-04 - val_loss:
9.0593e-04
Epoch 93/100
4/4 [==============================] - 0s 10ms/step - loss: 2.0259e-04 - val_loss:
9.3407e-04
Epoch 94/100
4/4 [==============================] - 0s 10ms/step - loss: 2.1148e-04 - val_loss:
8.8963e-04
Epoch 95/100
4/4 [==============================] - 0s 10ms/step - loss: 2.1482e-04 - val_loss:
9.1810e-04
Epoch 96/100
4/4 [==============================] - 0s 9ms/step - loss: 1.8893e-04 - val_loss: 9.
3610e-04
Epoch 97/100
4/4 [==============================] - 0s 10ms/step - loss: 2.0584e-04 - val_loss:
9.3994e-04
Epoch 98/100
4/4 [==============================] - 0s 10ms/step - loss: 1.9573e-04 - val_loss:
8.8846e-04
Epoch 99/100
4/4 [==============================] - 0s 11ms/step - loss: 2.2011e-04 - val_loss:
9.1778e-04
Epoch 100/100
4/4 [==============================] - 0s 10ms/step - loss: 2.1501e-04 - val_loss:
9.2366e-04
```

In [ ]:
```
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(1, len(loss) + 1)

plt.figure(figsize=(8,12))
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

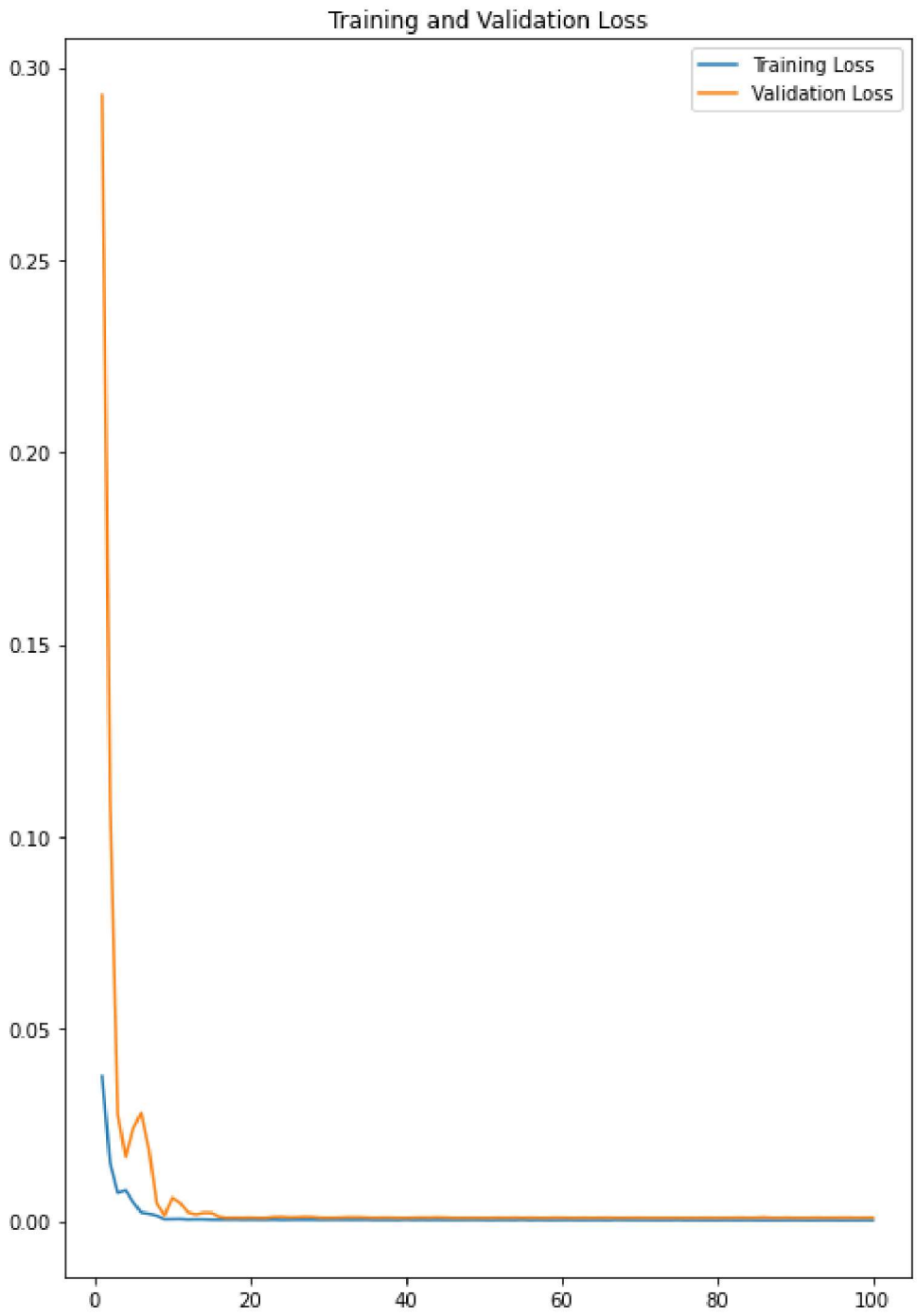
Out[ ]:
```
<Figure size 576x864 with 0 Axes>
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x149bd776610>]
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x149bd7768b0>]
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x149d13e0fd0>
```

Out[ ]: Text(0.5, 1.0, 'Training and Validation Loss')




In [ ]:
```
train_y_predict = reg.predict(train_x)[:, 0]
```

In [ ]:
```
train_y_ori = sc.inverse_transform(
    train_y.reshape(-1, 1)).reshape(train_y.shape)
train_y_predict_ori = sc.inverse_transform(
    train_y_predict.reshape(-1, 1)).reshape(train_y_predict.shape)
```

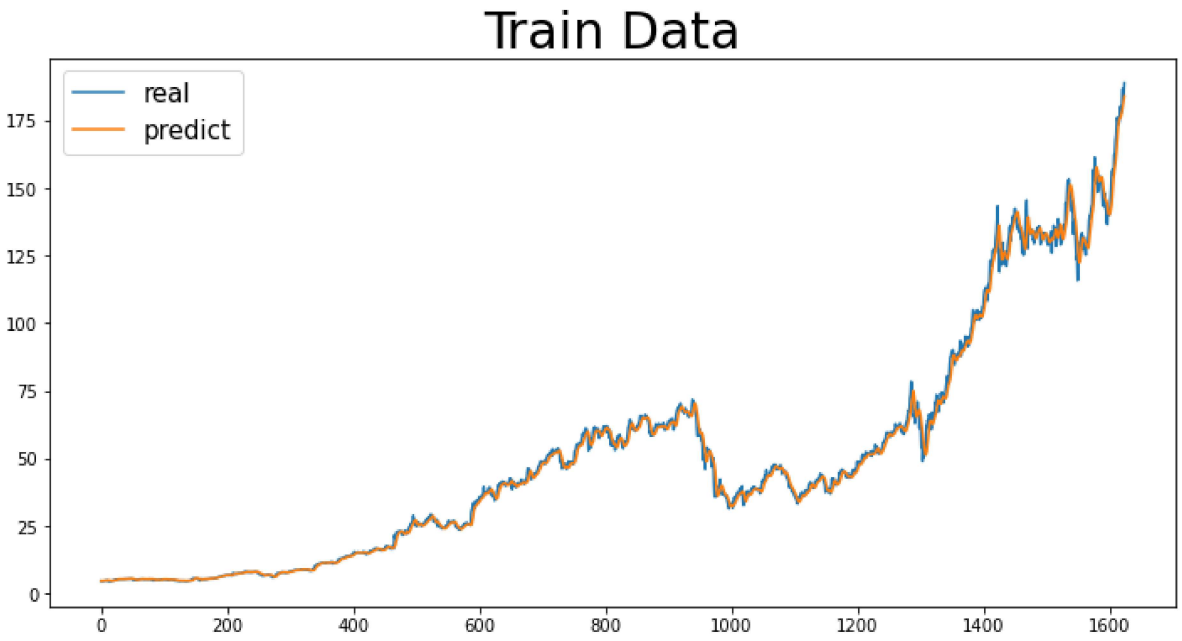
In [ ]:
```
draw = pd.concat([pd.DataFrame(train_y_ori),
                  pd.DataFrame(train_y_predict_ori)], axis=1)
draw.iloc[:, 0].plot(figsize=(12, 6))
draw.iloc[:, 1].plot(figsize=(12, 6))
plt.legend(('real', 'predict'), loc='upper left', fontsize='15')
plt.title("Train Data", fontsize='30')
```

Out[ ]: <AxesSubplot:>

Out[ ]: <AxesSubplot:>

Out[ ]: <matplotlib.legend.Legend at 0x149bdd1fe20>

Out[ ]: Text(0.5, 1.0, 'Train Data')




In [ ]:
```
test_y_predict = reg.predict(test_x)[:, 0]
```

In [ ]:
```
test_y_ori = sc.inverse_transform(test_y.reshape(-1, 1)).reshape(test_y.shape)
test_y_predict_ori = sc.inverse_transform(
    test_y_predict.reshape(-1, 1)).reshape(test_y_predict.shape)
```

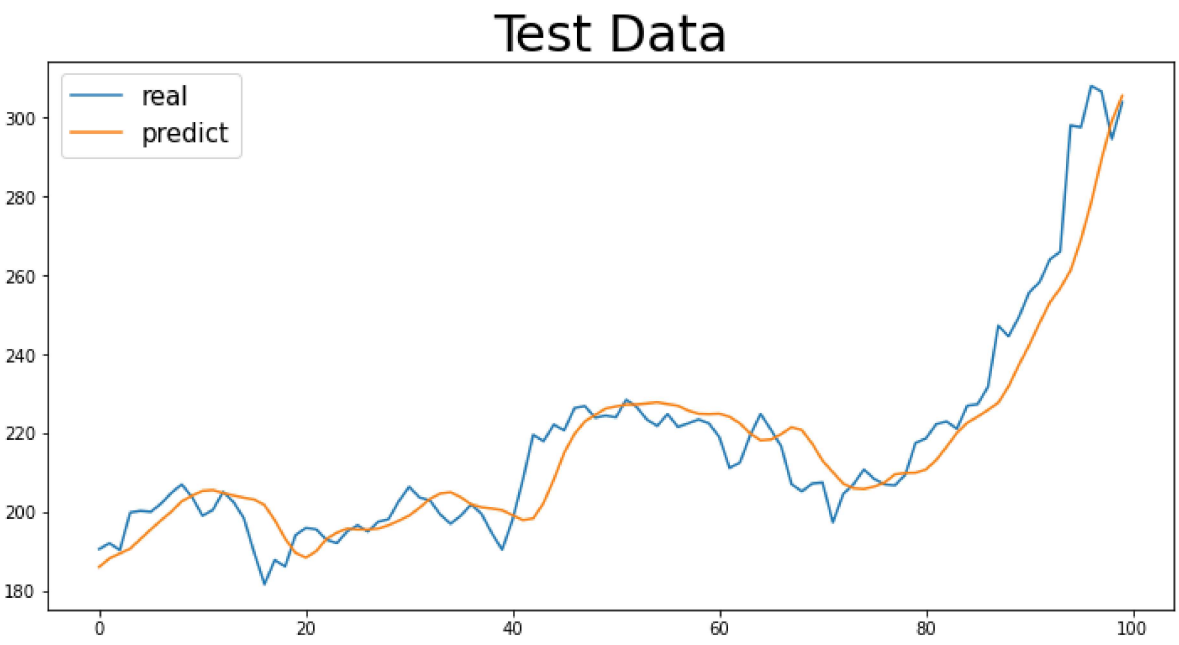
In [ ]:
```
draw = pd.concat(
    [pd.DataFrame(test_y_ori), pd.DataFrame(test_y_predict_ori)], axis=1)
draw.iloc[:, 0].plot(figsize=(12, 6))
draw.iloc[:, 1].plot(figsize=(12, 6))
plt.legend(('real', 'predict'), loc='upper left', fontsize='15')
plt.title("Test Data", fontsize='30')
```

Out[ ]: <AxesSubplot:>

Out[ ]: <AxesSubplot:>

Out[ ]: <matplotlib.legend.Legend at 0x149bd908520>

Out[ ]: Text(0.5, 1.0, 'Test Data')




In [ ]:
```
# Root mean squared error
import math
from sklearn.metrics import mean_squared_error,mean_absolute_error
rmse = math.sqrt(mean_squared_error(test_y_ori, test_y_predict_ori))
mae = mean_absolute_error(test_y_ori, test_y_predict_ori)
print(mean_squared_error(test_y_ori, test_y_predict_ori))
print(rmse)
print(mae)
```

```
87.46662711801939
9.352359441232966
6.668003540039063
```

In [ ]:
```
print(window)
print(len(loss))
print(rmse)
df = pd.DataFrame([loss,val_loss],index=["loss","val_loss"]).T
filename = "window"+ str(window) + "epochs" + str(len(loss)) + ".csv"
# df.to_csv("result/"+filename+"", index=False, sep=',')
```

```
5
100
9.352359441232966
```

# Appendix 2: ARIMA Model

# ARIMA model in stock price prediction

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
%matplotlib inline

from math import sqrt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
from pandas.plotting import lag_plot
from pylab import rcParams

import statsmodels as sm
from statsmodels.tsa.seasonal import seasonal_decompose
from pandas import DataFrame
from pandas import concat
from statsmodels.tsa.ar_model import AR
from statsmodels.tsa.arima_model import ARMA
from statsmodels.tsa.arima_model import ARIMA

from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import acf, pacf
```

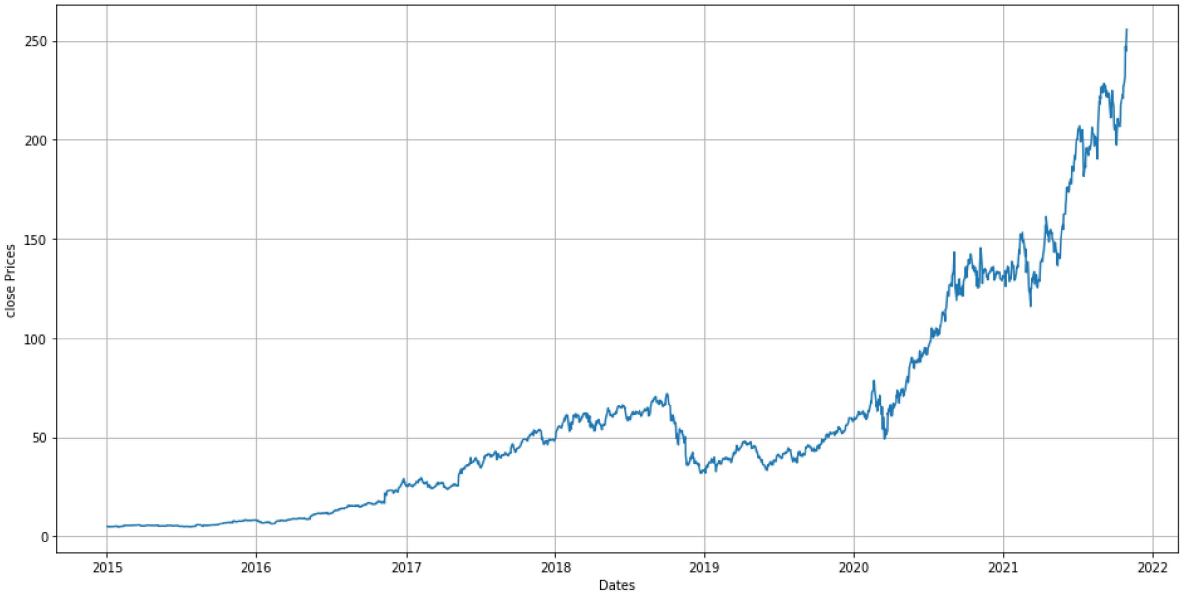
```
/anaconda3/lib/python3.7/site-packages/pandas/compat/_optional.py:138: UserWarning:
Pandas requires version '2.7.0' or newer of 'numexpr' (version '2.6.8' currently ins
talled).
  warnings.warn(msg, UserWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py:37: DeprecationWa
rning: distutils Version classes are deprecated. Use packaging.version instead.
  LARGE_SPARSE_SUPPORTED = LooseVersion(scipy_version) >= '0.14.0'
```

In [ ]:
```
import warnings
warnings.filterwarnings('ignore')
```

## Data Preparation

In [ ]:
```
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
data = pd.read_csv('NVidia_stock_history.csv',
                   parse_dates=['Date'], index_col='Date',date_parser=dateparse)
data=data.iloc[4013:-10,0:11]  #start at 2015
plt.figure(figsize=(16,8))
plt.grid(True)
plt.xlabel('Dates')
plt.ylabel('close Prices')
plt.plot(data['Close'])
```

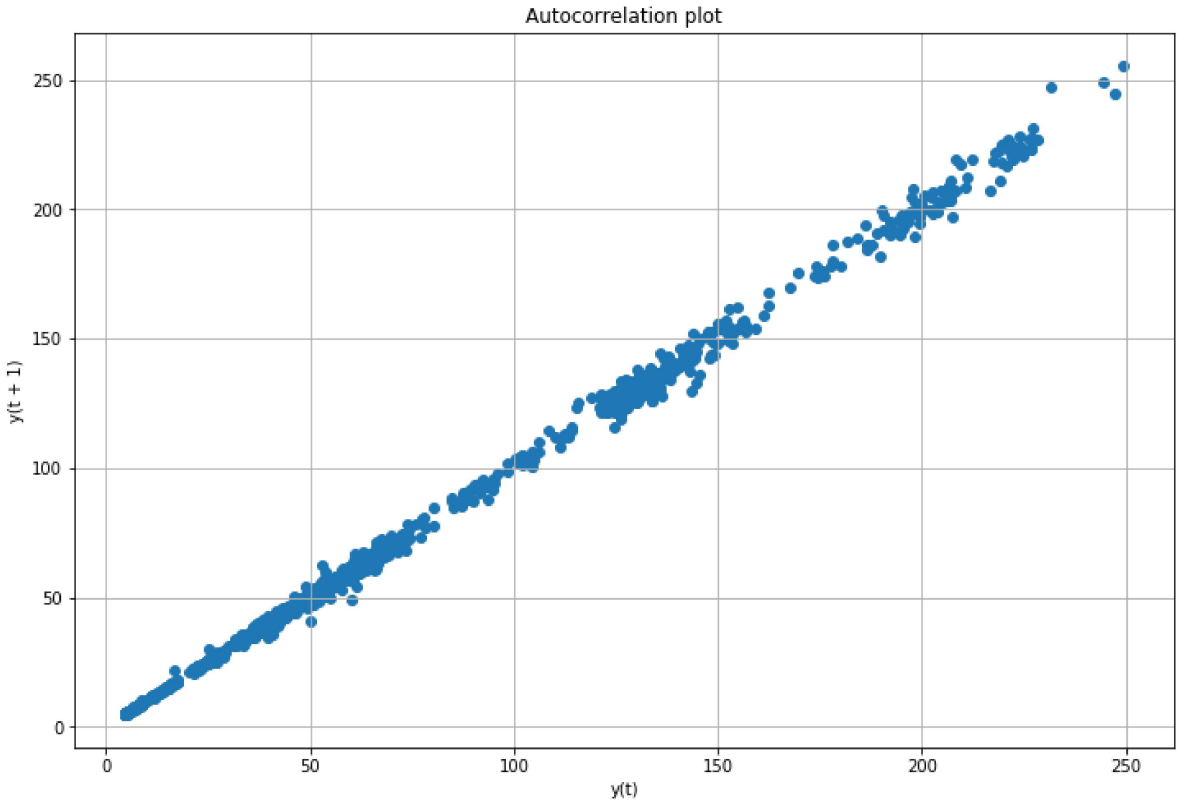
Out[ ]:
```
[<matplotlib.lines.Line2D at 0x14252f9e8>]
```




In [ ]:
```
#test_data
```

## The charts above shows a high correlation between y(t) and y(t+1).

In [ ]:
```
plt.figure(figsize=(12,8))
lag_plot(data['Close'], lag=1)
plt.title('Autocorrelation plot')
plt.grid(True)
#此图说明数据有自相关性
```




## Checking correlation by calculating covariance:

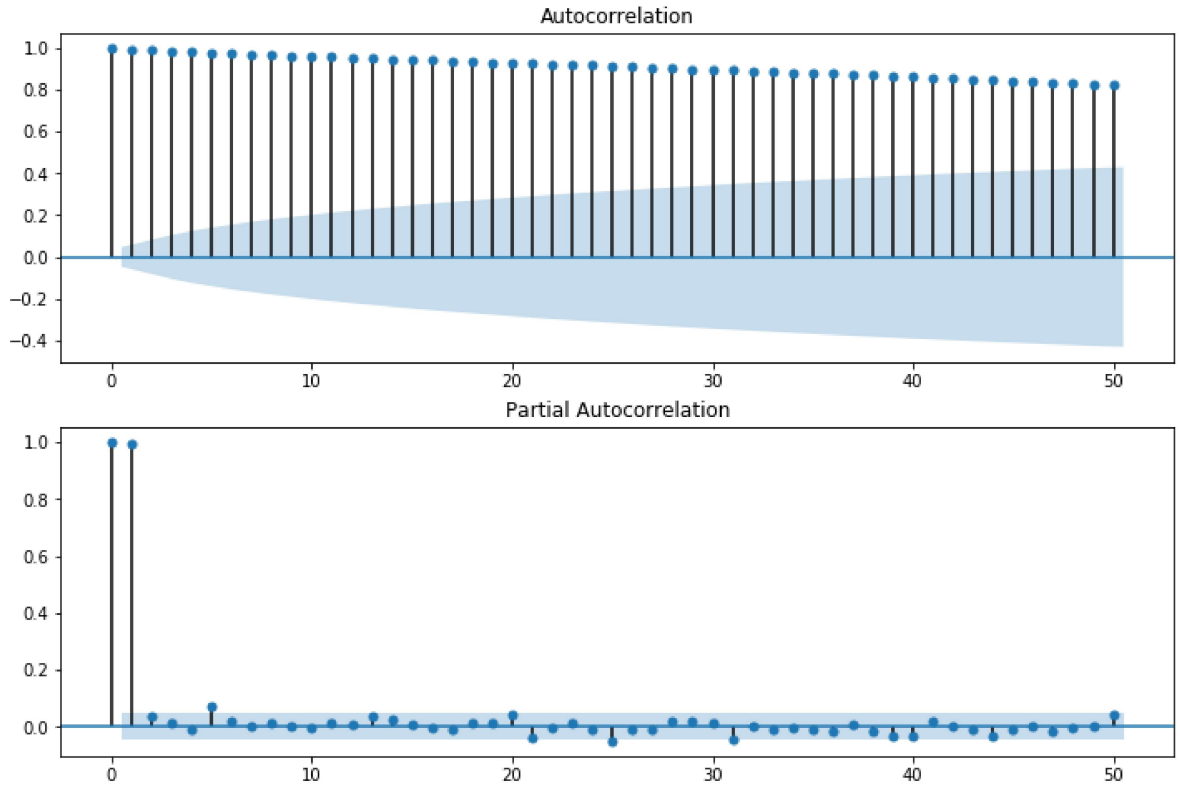
In [ ]:
```
#查看自相关性
values = DataFrame(data['Close'].values)
```

```
dataframe = concat([values.shift(1),values.shift(5),values.shift(10),values.shift(30
dataframe.columns = ['t', 't+1', 't+5', 't+10', 't+30']
result = dataframe.corr()
print(result)
```

| | t | t+1 | t+5 | t+10 | t+30 |
|---|---|---|---|---|---|
| t | 1.000000 | 0.997297 | 0.994183 | 0.985304 | 0.999274 |
| t+1 | 0.997297 | 1.000000 | 0.996771 | 0.986275 | 0.996683 |
| t+5 | 0.994183 | 0.996771 | 1.000000 | 0.988337 | 0.993443 |
| t+10 | 0.985304 | 0.986275 | 0.988337 | 1.000000 | 0.985114 |
| t+30 | 0.999274 | 0.996683 | 0.993443 | 0.985114 | 1.000000 |

In [ ]:
```
from statsmodels.graphics import tsaplots
```

In [ ]:
```
fig = plt.figure(figsize=(12, 8))
ax1 = fig.add_subplot(211)
fig = tsaplots.plot_acf(data['Close'], lags=50, alpha=0.05, ax=ax1)
ax2 = fig.add_subplot(212)
fig = tsaplots.plot_pacf(data['Close'], lags=50, alpha=0.05, ax=ax2)
#下图说明愿数据不平稳，需要进行处理
```




In [ ]:
```
train_data, test_data = data[0:-100], data[-100:]
```

# AR

In [ ]:
```
#可以不平稳
train_ar = train_data['Close'].dropna() #删掉空值 选择close value作为训练数据
test_ar = test_data['Close']

model = AR(train_ar)
model_fit = model.fit()
#训练模型后得到的参数： AR的之后项个数p，和自回归函数的各个系数
coef = model_fit.params
p = model_fit.k_ar  # 即时间序列模型中常见的p，即AR(p)，ARMA(p,q)，ARIMA(p,d,q)中的p
# p的实际含义，此处得到p=24，意味着当天由最近24天预测。
window = 1 #built ar 1 model
```

```
history = train_ar[len(train_ar)-window:]
history = [history[i] for i in range(len(history))]
predictions = list()

for t in range(len(test_ar)):
    length = len(history)
    lag = [history[i] for i in range(length-window,length)]
    #print(window)
    yhat = coef[0]
    for d in range(window):
        yhat += coef[d+1] * lag[window-d-1]
    obs = test_ar[t]
    predictions.append(yhat+22)
    history.append(obs)

print('MSE: '+str(mean_squared_error(test_data['Close'], predictions)))
print('MAE: '+str(mean_absolute_error(test_data['Close'], predictions)))
print('RMSE: '+str(sqrt(mean_squared_error(test_data['Close'], predictions))))
```

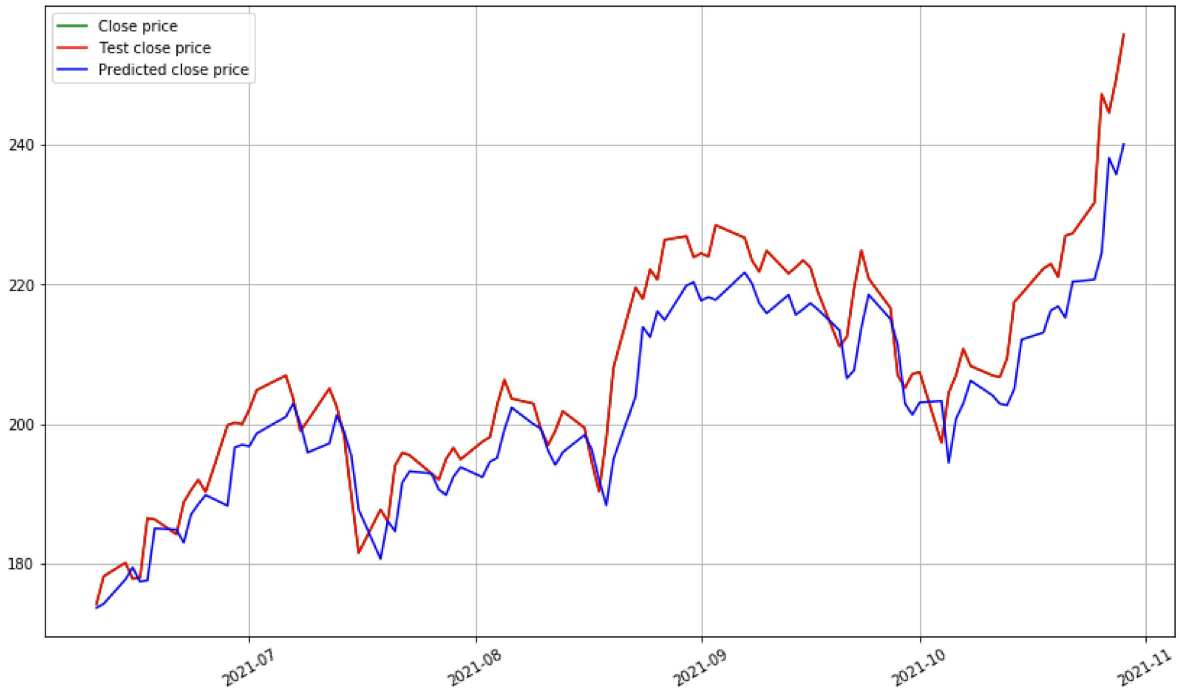
```
MSE: 46.34421792513785
MAE: 5.472446949780711
RMSE: 6.807658769734118
```

In [ ]:

In [ ]:
```
plt.figure(figsize=(14,8))
plt.plot(data.index[-600:], data['Close'].tail(600), color='green', label='Close pr
plt.plot(test_data.index, test_data['Close'], color='red', label='Test close price')
plt.plot(test_data.index, predictions, color='blue', label='Predicted close price')
#plt.plot(pred.index[-100:], pred[-100:], color='blue', label='Predicted close price
plt.xticks(rotation=30)
plt.grid(True)
plt.legend()
plt.figure(figsize=(14,8))
print('Lag: %s' % model_fit.k_ar)
plt.plot(data.index[-100:], data['Close'].tail(100), color='green', label='Close pr
plt.plot(test_data.index, test_data['Close'], color='red', label='Test close price')
plt.plot(test_data.index, predictions, color='blue', label='Predicted close price')
#plt.plot(pred.index[-100:], pred[-100:], color='blue', label='Predicted close price
plt.xticks(rotation=30)
plt.grid(True)
plt.legend()
```

```
Lag: 24
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x14276ceb8>
```


# MA

In [ ]:
```
train_ma = train_data['Close'].dropna()
test_ma = test_data['Close']
history = [x for x in train_ma]
y = test_ma
predictions = list()


model = ARMA(history, order=(0,1))
model_fit = model.fit(disp= -1)
q = model_fit.k_ar
print(q)
yhat = model_fit.forecast()[0]
predictions.append(yhat)
history.append(y[0])
```

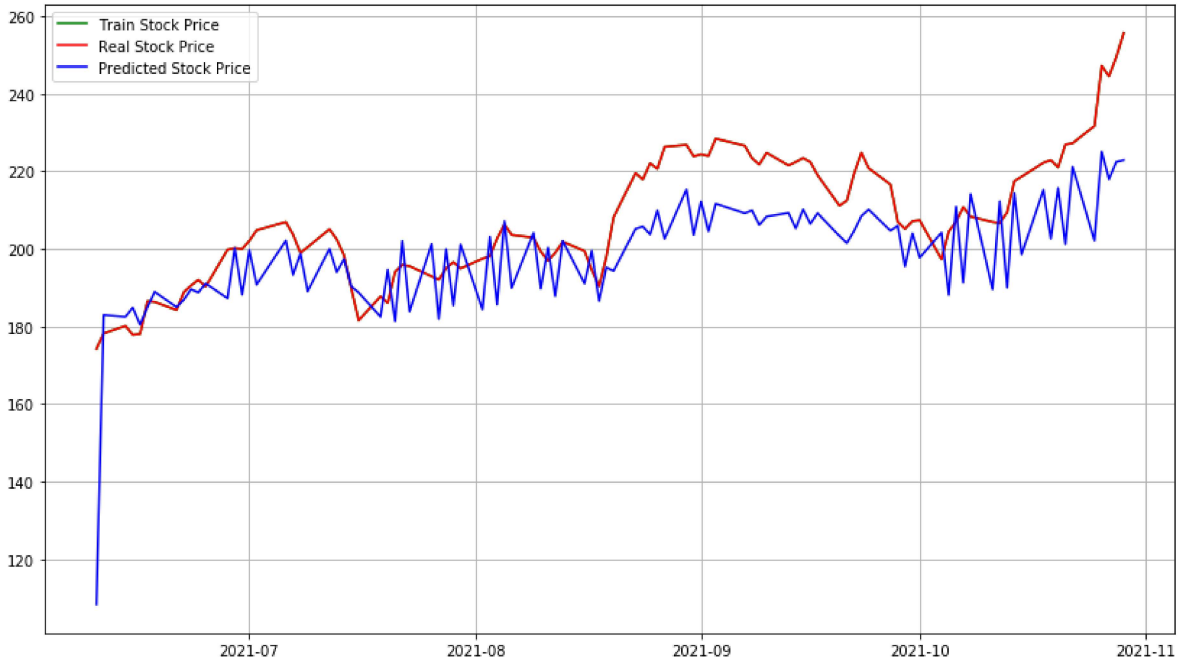
```
for i in range(1, len(y)):
    model = ARMA(history, order=(0,1))
    model_fit = model.fit(disp=0)
    yhat = model_fit.forecast()[0]
    predictions.append(yhat+70)
    obs = y[i]
    history.append(obs)



print('MSE: '+str(mean_squared_error(y, predictions)))
print('MAE: '+str(mean_absolute_error(y, predictions)))
print('RMSE: '+str(sqrt(mean_squared_error(y, predictions))))
```

```
0
MSE: 200.88513323074633
MAE: 10.852043450964263
RMSE: 14.17339526121904
```

In [ ]:
```
yhat = model_fit.forecast()[0]
print(yhat)
```

```
[152.92129464]
```

In [ ]:
```
plt.figure(figsize=(14,8))
plt.plot(data.index[-600:], data['Close'].tail(600), color='green', label = 'Train S
plt.plot(test_data.index, y, color = 'red', label = 'Real Stock Price')
plt.plot(test_data.index, predictions, color = 'blue', label = 'Predicted Stock Pric
plt.legend()
plt.grid(True)
plt.show()
plt.figure(figsize=(14,8))
plt.plot(data.index[-100:], data['Close'].tail(100), color='green', label = 'Train S
plt.plot(test_data.index, y, color = 'red', label = 'Real Stock Price')
plt.plot(test_data.index, predictions, color = 'blue', label = 'Predicted Stock Pric
plt.legend()
plt.grid(True)
plt.show()
```

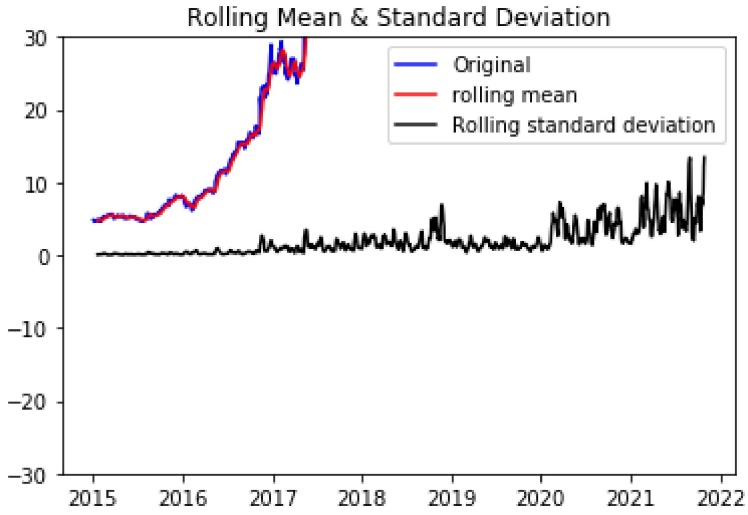



# ARMA

Difference.

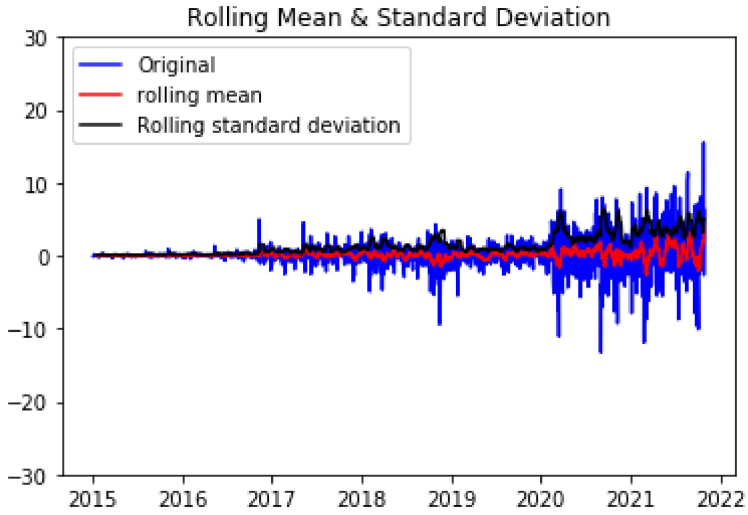
In [ ]:
```
def test_stationarity(timeseries):
    rolmean = pd.Series.rolling(timeseries,window=12).mean()
    rolstd = pd.Series.rolling(timeseries, window=12).std()
    fig = plt.figure()
    fig.add_subplot()
    orig = plt.plot(timeseries, color = 'blue',label='Original')
    mean = plt.plot(rolmean , color = 'red',label = 'rolling mean')
    std = plt.plot(rolstd, color = 'black', label= 'Rolling standard deviation')
    plt.ylim([-30,30])

    plt.legend(loc = 'best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)
    print( 'Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries,autolag = 'AIC')
    dfoutput = pd.Series(dftest[0:4],index = ['Test Statistic','p-value','#Lags Used
    for key,value in dftest[4].items():
        dfoutput['Critical value (%s)' %key] = value
    print(dfoutput)

#ts_log = data['Close']
#ts_log_diff = ts_log - ts_log.shift()
#ts_log_diff.dropna(inplace=True)
```

In [ ]:
```
test_stationarity(data['Close'])
```




```
Results of Dickey-Fuller Test:
Test Statistic                    3.787452
p-value                           1.000000
#Lags Used                       25.000000
Number of Observations Used    1694.000000
Critical value (1%)              -3.434216
Critical value (5%)              -2.863248
Critical value (10%)             -2.567679
dtype: float64
```

In [ ]:
```
datadif = data['Close'].diff()
datadif.dropna(inplace = True)
test_stationarity(datadif)
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -8.539803e+00
p-value                          9.853139e-14
#Lags Used                       2.500000e+01
Number of Observations Used      1.693000e+03
Critical value (1%)             -3.434218e+00
Critical value (5%)             -2.863249e+00
Critical value (10%)            -2.567680e+00
dtype: float64
```

In [ ]:
```
#通过测试，是平稳的
#测试是否为白噪声序列，若是则停止分析
from statsmodels.stats.diagnostic import acorr_ljungbox
```

In [ ]:
```
lbvalue,pvalue = acorr_ljungbox(datadif,lags=5)
pvalue
```

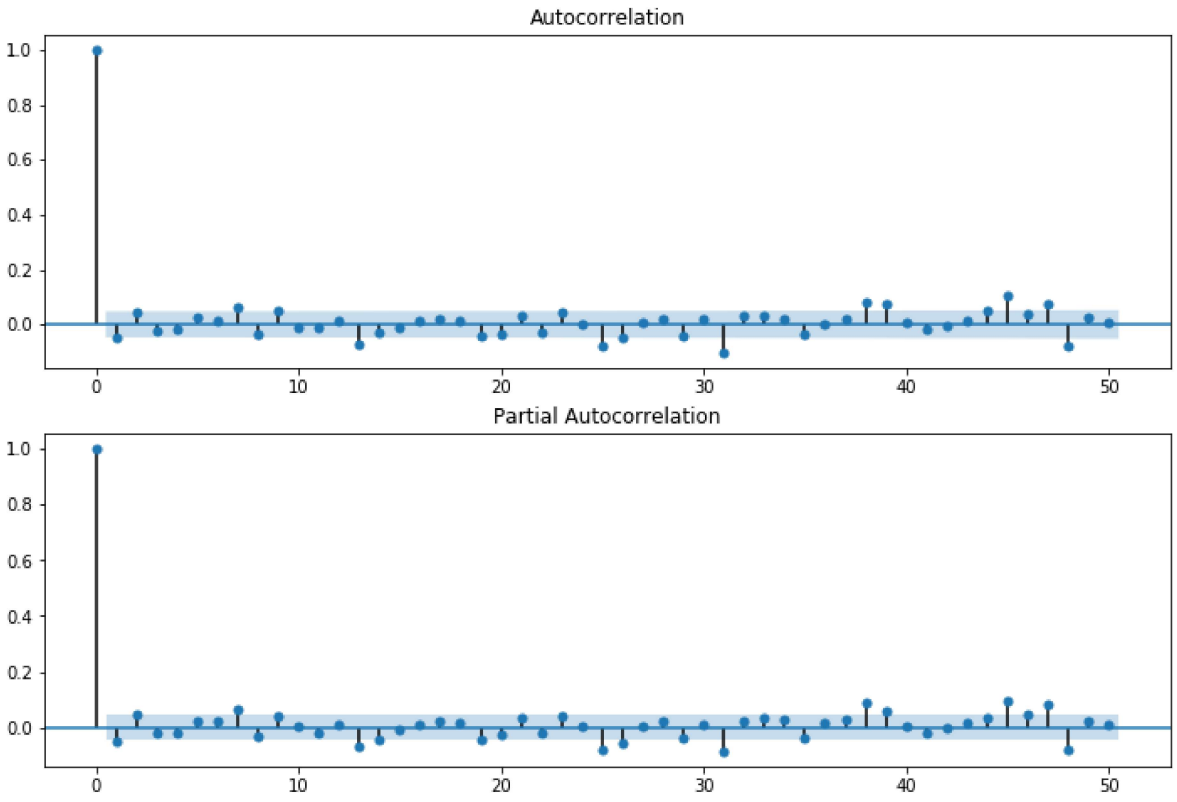
Out[ ]:
```
array([0.03483478, 0.01757889, 0.02981796, 0.04838994, 0.05984103])
```

In [ ]:
```
#变化不大，换一个方法
#平滑法  对size个数据进行加权移动平均
def draw_trend(timeSeries, size):
    f = plt.figure(facecolor='white')

    # 对size个数据进行移动平均
    rol_mean = timeSeries.rolling(window=size).mean()


    # 对size个数据进行加权移动平均
    rol_weighted_mean = pd.DataFrame.ewm(timeSeries, span=size).mean()

    timeSeries.plot(color='blue', label='Original')
    rol_mean.plot(color='red', label='Rolling Mean')
    rol_weighted_mean.plot(color='black', label='Weighted Rolling Mean')
    plt.legend(loc='best')
    plt.title('Rolling Mean')
    plt.show()
    return rol_weighted_mean


#ddfff = draw_trend(ts_log_diff,12)
#test_stationarity(ddfff)  #是所有的open data做平滑，后面test也可以用这个
```

怎么确定pq值?

In [ ]:
```
from statsmodels.graphics import tsaplots
```

In [ ]:
```
fig = plt.figure(figsize=(12, 8))
ax1 = fig.add_subplot(211)
fig = tsaplots.plot_acf(datadif, lags=50, alpha=0.05, ax=ax1)
ax2 = fig.add_subplot(212)
fig = tsaplots.plot_pacf(datadif, lags=50, alpha=0.05, ax=ax2)
```



## ARMA process with data after difference:

In [ ]:
```
datadif = datadif[1400:]
print(len(datadif))
```

```
319
```

In [ ]:
```
#两个模型的RMSE能不能直接比较 跟数据预处理是不是有影响？
#使用滚动窗口
testdata = data[-90:]
train_arma, test_arma = datadif[0:-100], datadif[-100:]
```

In [ ]:
```
import  statsmodels.tsa.stattools as sts
resid = sts.arma_order_select_ic(train_arma , max_ar=5, max_ma=5,ic=['aic','bic','h
print ('AIC-order :{}'.format(resid.aic_min_order))
print ('BIC-order :{}'.format(resid.bic_min_order))
print ('HQIC-order :{}'.format(resid.hqic_min_order))
```

```
AIC-order :(5, 3)
BIC-order :(5, 3)
HQIC-order :(5, 3)
```

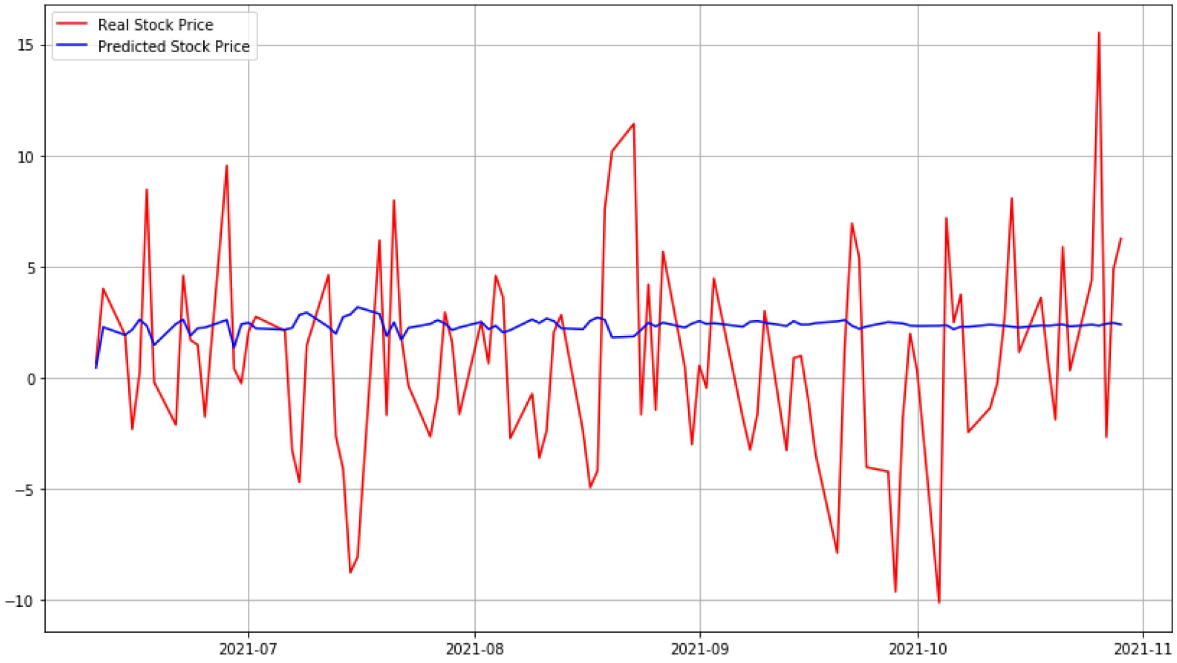
In [ ]:
```

```

In [ ]:
```
from statsmodels.tsa.arima.model import ARIMA
```

In [ ]:
```
history = [x for x in train_arma]
#y=draw_trend(test_arma, 12)
y=test_arma
predictions = list()
model = ARIMA(history, order=(1,0,0))  #(p,1,q)
model_fit = model.fit()
yhat = model_fit.forecast()[0]
predictions.append(yhat)
history.append(y[0])

for i in range(1, len(y)):
    model = ARIMA(history, order=(1,0,0))
    model_fit = model.fit()
    yhat = model_fit.forecast()[0]
    predictions.append(yhat+2)
    obs = y[i]
    history.append(obs)

print('MSE: '+str(mean_squared_error(y, predictions)))
print('MAE: '+str(mean_absolute_error(y, predictions)))
print('RMSE: '+str(sqrt(mean_squared_error(y, predictions))))
```

```
MSE: 22.773437147984318
MAE: 3.7363416338492677
RMSE: 4.772152255322992
```

In [ ]:
```
plt.figure(figsize=(14,8))
plt.plot(test_arma.index, y, color = 'red', label = 'Real Stock Price')
plt.plot(test_arma.index, predictions, color = 'blue', label = 'Predicted Stock Pric
plt.legend()
plt.grid(True)
plt.show()
```




In [ ]:
```
model_fit.summary()
```

Out[ ]:

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | y | **No. Observations:** | 318 |
| **Model:** | ARIMA(1, 0, 0) | **Log Likelihood** | -890.228 |
| **Date:** | Mon, 16 May 2022 | **AIC** | 1786.456 |
| **Time:** | 11:00:37 | **BIC** | 1797.742 |
| **Sample:** | 0 | **HQIC** | 1790.964 |
| | - 318 | | |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | 0.4570 | 0.221 | 2.070 | 0.038 | 0.024 | 0.890 |
| **ar.L1** | -0.0133 | 0.049 | -0.273 | 0.785 | -0.109 | 0.082 |
| **sigma2** | 15.8159 | 1.036 | 15.272 | 0.000 | 13.786 | 17.846 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 0.00 | **Jarque-Bera (JB):** | 12.36 |
| **Prob(Q):** | 1.00 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 1.47 | **Skew:** | -0.03 |
| **Prob(H) (two-sided):** | 0.05 | **Kurtosis:** | 3.96 |

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

# 把prediction还原成差分前的数据

ye = pd.Series(predictions) prevy= test_data['Close']

# print(ye)

yre=ye.cumsum()+prevy.values[-1]

# print(yre)

# 还原y的预测值

y_restored = pd.Series([prevy[0]], index=[prevy.index[0]]).append(ye).cumsum() y_restored = y_restored[0:100] print(type(y_restored))

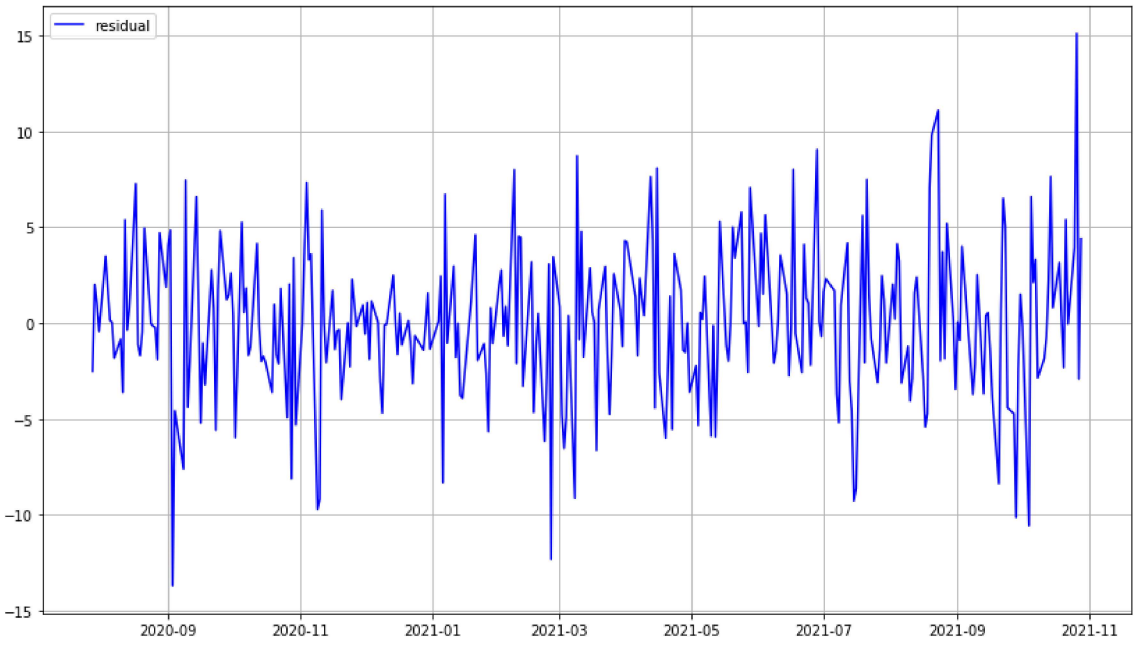
# print(len(prevy),len(y_restored[0:100]))

plt.figure(figsize=(14,8)) fig = plt.figure() fig = matplotlib.pyplot.gcf() fig.set_size_inches(18.5, 10.5) ax = fig.add_subplot(111) ax.plot(test_data.index, prevy, color='green', label = 'Real Stock Price') ax2 = ax.twinx() ax2.plot(test_data.index,y_restored, color='red',label = 'Predicted Retrun') ax.legend(loc = (0.02, 0.95)) ax.grid() ax.set_xlabel("Time") ax.set_ylabel("Return") ax2.set_yticks([]) ax2.legend(loc = (0.02, 0.91))

In [ ]:

In [ ]:
```
armares = model_fit.resid
lbvalue,pvalue = acorr_ljungbox(armares,lags=20)
print(pvalue)
plt.figure(figsize=(14,8))
#plt.plot(test_arma.index, y, color = 'red', label = 'Real Stock Price')
plt.plot(datadif.index[:-1], armares, color = 'blue', label = 'residual')
plt.legend()
plt.grid(True)
plt.show()

#pvalue 序列位白噪声
```

```
[0.99701658 0.95502609 0.60323546 0.71863092 0.80519968 0.71221757
 0.76045538 0.8312504  0.88273925 0.92362742 0.95227659 0.96188353
 0.92252106 0.76258303 0.81761712 0.85097308 0.8888     0.91958492
 0.89028716 0.90256699]
```




## autocorrelation test on the residuals

In [ ]:
```
from statsmodels.stats import diagnostic
resid = model_fit.resid
_, pvalue, _, bppvalue = diagnostic.acorr_ljungbox(resid, lags=None, boxpierce=True
print (pvalue)
```

```
[0.99701658 0.95502609 0.60323546 0.71863092 0.80519968 0.71221757
 0.76045538 0.8312504  0.88273925 0.92362742 0.95227659 0.96188353
 0.92252106 0.76258303 0.81761712 0.85097308 0.8888     0.91958492
 0.89028716 0.90256699 0.92876708 0.9469323  0.94529734 0.96068215
 0.86077665 0.75877403 0.79284089 0.8301664  0.84347763 0.8720875
 0.65845229 0.70257758 0.73925887 0.73433346 0.71996637 0.75283428
 0.78953218 0.65961949 0.62082724 0.65027961]
```

In [ ]: