Statistical Analysis for Hong Kong Financial Time Series Data

by
Jia Jingwen
(19250924)

A thesis submitted in the partial fulfillment of the requirements for the degree of

Bachelor of Science (Honors)
In Mathematics and Statistics

at

Hong Kong Baptist University

December 24th, 2023

ACKNOWLEDGEMENT

Part of the work presented in this thesis was done in collaboration with Ruey Tsay in the Analysis of Financial Time Series (3rd Version) Book. And the stocks prices about Vanke Hong Kong Equity (02202 HK), Vanke China (000002 SZ), Greenland Holdings Hong Kong (00337 HK), and Greenland Holdings China (600606 SH) from the Bloomberg Terminal. And the Figure 2.1 to Figure 2.4 were done in collaboration with Afshine Amidi and Shervine Amidi at Stanford CS230 Deep Learning Teaching materials about Recurrent Neural Networks cheat sheet.

[illegible]

Signature of Student

Student name
Jia Jingwen

Department of Mathematics
Hong Kong Baptist University

Date:
2023 December 24

Catalog

Statistical Analysis for Hong Kong Financial Time Series Data

Jia Jingwen
(19250924)

Department of Mathematics

Abstract

Evergrande Default and consistent China private developers default has attracted me and want to see how the China Real Estate Stock Market influences the Hong Kong Real Estate Market and the lag between them. In this project, I choose the Vanke Hong Kong (02202 HK) and Vanke China (000002 SZ) and Greenland Holdings Hong Kong (00337 HK) and Greenland Holdings China (600606 SH) as two major pairs, relying on the traditional Time Series model (ARIMA Model) and the Recurrent Neural Networks (LSTM) methods to train and predict the Hong Kong Stock Market in the Real Estate Sector. By comparing the different predictions from these two methods, we found the traditional time series model can't capture the nonlinear models and nearly deal with the noises and irregularities and lack feasibility only relying on the certain parameters. But the LSTM could reflect us a better estimation to capture a long-term time series data.

Chapter 1: Introduction

1.1 Hong Kong Financial Time Series Data

Hong Kong's financial landscape encompasses various markets, including the bond market (onshore and offshore), the foreign exchange (FX) market, and the stock market. When considering the sensitivity of these markets to changes over time, this project solely focuses on the equity market due to the following compelling reasons.

In terms of the bond market, issuers raise capital by borrowing funds from investors and subsequently commit to making periodic interest payments and repaying the principal at maturity. This predetermined nature of cash flows in bond interests and principal results in a lower level of sensitivity and fewer price fluctuations compared to the equity market. Furthermore, fluctuations in the bond market are primarily driven by changes in interest rates. Actually, interest rates are influenced not by the time changes and by a wide range of factors, including monetary policy decisions, inflation expectations, economic indicators, and geopolitical events. These factors may not always align with short-term fluctuations in the time series, leading to a less direct and immediate reflection of time series in the bond market. Additionally, credit ratings and default risks can significantly impact the bond market. However, the assessment of credit levels and default risk is based on longer-term factors such as the financial stability of issuers, business performance, and economic conditions, which may not be closely tied to short-term changes in time. Consequently, the bond market lacks a strong and immediate connection between time and price compared to the stock market.

In contrast, the stock market exhibits a stronger connection with time due to several key factors. Firstly, the stock market operates within a continuous flow of information. This includes the release of company earnings reports, economic indicators, and news events, which are disseminated periodically, often on a daily basis. The timely availability and incorporation of this information into stock prices establish a robust link between time and stock price changes. Investors swiftly react to new information, leading to price adjustments that reflect the prevailing market sentiment.

Furthermore, the stock market is inherently influenced by economic cycles. It experiences fluctuations that coincide with broader economic trends. Seasonality and calendar effects also come into play, with recurring patterns observed in the stock market. For instance, the "January Effect" refers to the historical tendency of the stock market to exhibit higher returns in January, while the "sell in May and go away" strategy suggests that stock market returns tend to be lower during the summer months. These observed patterns indicate that stock market behavior is influenced by time-related factors, and the timing of market cycles is essential for understanding stock price movements.

Moreover, the stock market has specific trading hours, opening and closing times, and trading sessions. These structural aspects create a temporal dimension to market activity. Trading sessions encompass pre-market and after-hours trading, which can impact price discovery and potentially result in gaps in stock prices between consecutive trading days. The trading structure and timing of market operations contribute to the stock market's strong connection with time. So based on the above, the equity market, unlike the bond market, demonstrates a stronger and more immediate relationship between time and price due to its continuous flow of information, its alignment with economic cycles, the presence of seasonality effects, and the structural elements governing trading activity. Just like the following graph (Figure 1.1: Hang Seng Index Open & Close Price (HKD) from 2023 January to December), it can reflect the Hang Seng Index Open Price and Closing Price changes with time. Overall, it reflects a downward tendency and there exists a usual lag of approximately 1 month, between January to October and in November, the discrepancy has been shortened and in December, the discrepancies reflected the expansion trend which will reflect on the January's Hang Seng Index. To detect the more accurate lag result and further prediction, we can apply the statistical methods to analyze and predict and the detailed procedure will be shown in the following section.

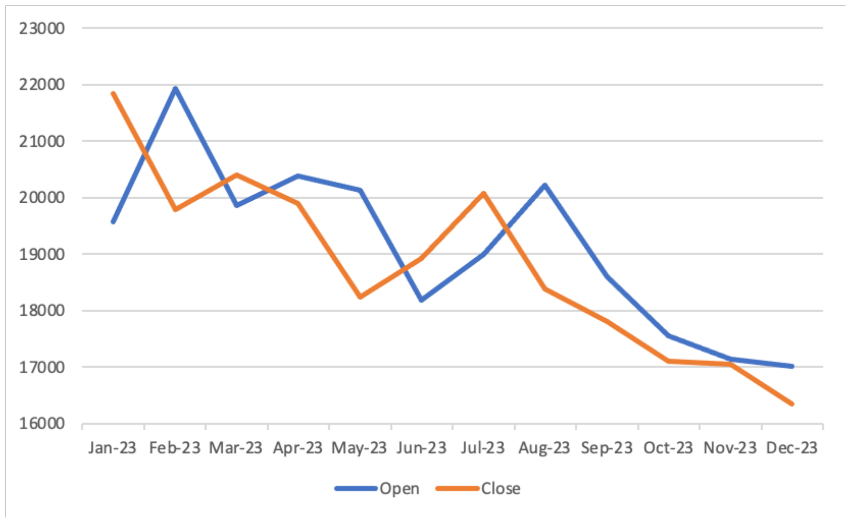


Figure 1.1: Hang Seng Index Open & Close Price (HKD) from 2023 January to December

1.2 Motivations & Aims

The bankruptcy declaration of Evergrande, a previously prestigious private real estate developer in mainland China, in December 2021 has had significant repercussions. This event has led to a downward trend in the mainland China Real Estate Industry Equity Market and has introduced a higher degree of market volatility. Given this context, there is a growing concern about the potential unpredictable negative influences that Evergrande's bankruptcy may have on the Hong Kong Equity Market, particularly within the Real Estate Sector.

The impact of Evergrande's bankruptcy extends beyond mainland China, as it has reverberated throughout the region's financial markets. The continuous defaults of both onshore and offshore bonds, combined with the suspension of stock trading, have further amplified the uncertainty and deteriorated investor sentiment. These developments have contributed to heightened volatility in stock prices within the equity market.

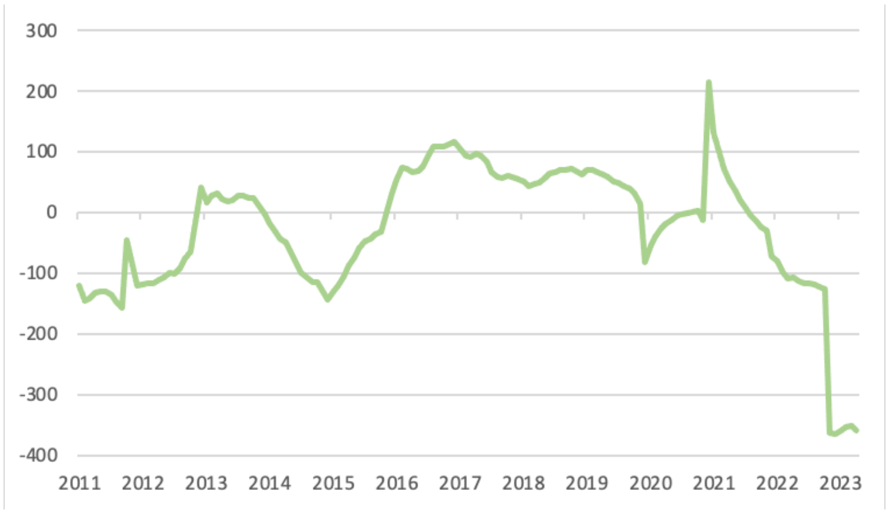


Figure 1.2: China Real Estate Funding Condition Trend, 2011-2023

Based on the Bloomberg (*Figure 1.2 China Real Estate Funding Condition Trend from 2011 to 2023*), I found there exists an index CHBGREM Index which can reflect the China Real Estate Funding Condition Trend from 2011 to 2023. The China Real Estate Funding Condition Index gives a snapshot of the funding condition in China's property market. A positive figure means the funding condition is better than its historical level and a rising reading of the index means increasing fund supply to the sector.

Given the interconnectedness of the global financial system, the implications of Evergrande's bankruptcy on the Hong Kong Equity Market, specifically within the Real Estate Sector, warrant careful analysis. The Hong Kong Real Estate Market, being closely linked to the mainland China Real Estate Market, may experience spillover effects from the turbulence caused by Evergrande's financial distress.

In this project, the objective is to examine how changes in the China Real Estate Equity Market will influence the tendency of the Hong Kong Real Estate Market. To achieve this, we will employ statistical methods such as Time Series Models and use the Recurrent Neural Networks (RNNs) to check if the LSTM method can gain better estimation and prediction for individual stock. The use of Time Series Models allows us to capture the temporal dynamics and patterns within the data, enabling us to analyze the historical behavior of the China Real Estate Equity Market. These models can help identify trends, seasonality, and other time-dependent patterns that may impact the Hong Kong Real Estate Market. RNNs are a type of neural network that can model sequential data, making them well-suited for analyzing time series data. By training an RNN model on historical data, we can capture the complex dependencies and non-linear relationships between the two markets.

However, it's important to note that while statistical methods and RNNs are powerful tools for analysis and prediction, they have limitations. Macroeconomic indicators such as investor sentiment/confidence, policy changes, and company-specific news play a significant role in shaping market behavior. These factors cannot be directly quantified as model parameters or biases. Therefore, it's crucial to consider and incorporate these qualitative factors alongside the quantitative analysis to gain a comprehensive understanding of the market dynamics.

By combining and comparing the results from both statistical methods and RNNs, we can gain valuable insights into the relationship between the China Real Estate Equity Market and the Hong Kong Real Estate Market. This analysis will contribute to a deeper understanding of the potential impacts of Evergrande's bankruptcy and help inform investment decisions and risk management strategies in the Hong Kong Real Estate Sector.

1.3 Literature Review

One essential method for examining and projecting changes in stock prices is time series analysis. It entails looking at past price data to spot trends, patterns, and seasonality. For stock price forecasting, a number of conventional time series models have been used, including exponential smoothing techniques and autoregressive integrated moving averages (ARIMA). These models give a reliable foundation for comparison with more sophisticated methods and capture the temporal dependencies in the data. The capacity of RNNs, a kind of neural network design, to simulate sequential data has made them useful in time series research. Long-term dependencies are well captured by them, and their stock price prediction has yielded encouraging results thus far. Learning longer-term dependencies is made easier with the help of LSTM, an RNN variation that tackles the vanishing gradient issue. LSTM networks are useful for capturing complicated temporal patterns because of their memory cell's ability to hold information over extended durations.

Stock price prediction using LSTM networks has been the subject of numerous studies. For instance, Zhang et al. (2017) suggested an LSTM-based model for stock price prediction that combined news sentiment and technical data. Their tests proved that LSTM is better than conventional time series models. Fischer et al. (2018) outperformed conventional linear models in a different investigation where they predicted stock returns using an LSTM architecture. These experiments demonstrate how LSTM may effectively capture the nonlinear dynamics and linkages found in stock price data.

Differently with the previous projects, I only focus on the statistical methods application into the Hong Kong Real Estate Market instead of the Hong Kong Hang Seng Index such benchmark. My project could provide a physical example in the Hong Kong Real Estate Market and China Real Estate Market connection and all the data comes from the same company but in different stock market just like Vanke Hong Kong (02202 HK) and Vanke China (000002 SZ); Greenland Hong Kong (0337 HK) and Greenland China (600606 SH), the different stock ticker but from the same parent companies could avoid other indicators' influence such as the company-specific event because the event or news from the same company would influence their stocks including in the Hong Kong Stock Market and in the China Stock Market. And in this project, we could see the limitations from the traditional time series model and the shortages for both time series methods and the Recurrent Neural Networks methods, reflecting that for the stock market, only relying on the quantitative model is not enough and the qualitative indicator would exist like white noises process which can't be ignored.

Chapter 2: Methodology & Implementations

2.1 Theorem

To effectively analyze Hong Kong Financial Time Series data, two notable approaches can be considered: Time Series Models and Recurrent Neural Networks (RNNs). Each approach offers distinct usefulness and limitations in dealing with financial data. This section aims to introduce the key theorems and algorithms utilized in this project, while elucidating their practical implementation.

2.1.1 Time Series

Time series analysis is a widely recognized statistical method employed for handling Hong Kong Financial Time Series Data, which is mathematically represented as a sequential arrangement of data points indexed by time. We can understand the definition in an easier approach: the previous events will influence the current event to some different extent. And the different influential extent can be considered as different weights parameters and they can be trained into a linear relationship then we can train Time Series Model under this circumstance. To effectively work with time series data, it is imperative to adhere to the following rigorous procedures:

1) Data Preprocess
    - Check the original data to be accurate, consistent, and complete
    - Identify the outliers, missing values, and anomalies
    - Ensure the stable variance against heteroscedasticity and ensure the linear relationship
2) Time Series Data Visualization
    - Draw the time series data to detect its tendency, seasonality, and patterns
    - Plot the ACF and PACF graphs to identify the potential lagged relationships
3) Stationary Test
    - Use the Augmented Dickey-Fuller (ADF) Test to see the p-value and compare it with 0.05 to check if we have enough proof to reject the null hypothesis
    - If not, use the differentiation method or log transformation to ensure the stationary
4) Model Selection and Estimation:

- Apply the Autoregressive Integrated Moving Average (ARIMA), Generalized Autoregressive Conditional Heteroskedasticity (GARCH), Exponential Smoothing (ETS), or Seasonal Decomposition of Time Series (STL)
- Check the AIC & BIC and variations as well as the Maximum Likelihood Estimation, Least Squares Estimations, Residuals' normality, independence, and constant variance

5) Forecasts
    - Predict for the further time period forecasts and assess the uncertainty of the forecasts through the confidence intervals or prediction intervals
    - Evaluate the forecast accuracy to check just like the Mean Absolute Error (MAE), Root Mean Squared Error (RMSE)

Autocorrelation Function (ACF)

For a weak stationary return series r*t*, if we want to know the linear dependence of the r*t* and the past values r *t-i*, we can use the correlation between them, called the lag-*l* autocorrelation of r*t* and under the weak stationary return assumption: Var (r*t*) = Var (r*t-l*).

$$\rho_\ell = \frac{\mathrm{Cov}(r_t, r_{t-\ell})}{\sqrt{\mathrm{Var}(r_t)\mathrm{Var}(r_{t-\ell})}} = \frac{\mathrm{Cov}(r_t, r_{t-\ell})}{\mathrm{Var}(r_t)} = \frac{\gamma_\ell}{\gamma_0}$$

The lag-l autocorrelation is a crucial measure to identify the parameters for Autoregressive Model (AR), Moving Averages Model (MA), and Autoregressive Moving Averages Model (ARMA). Different choices for lag l will make different autocorrelation and then formulate an ACF plot for every weak stationary series. And the ACF Plot reflects how the current value can be explained using the past values.

Partial Autocorrelation Function (PACF)

The PACF of a stationary time series is a function of its ACF and considered to be a useful tool for determining the order p of an AR model. If we focus on the two random variables' correlation coefficient, we need to consider to get rid of the influence of other random variables related to them. Taking the AR(p) model as an example, assuming the stability:

$$z_t = \phi_{k1} z_{k-1} + \phi_{k2} z_{t-2} + \cdots + \phi_{kk} z_{t-k} + a_t \Rightarrow \boldsymbol{\phi} z_t = a_t$$

implies phi(B) has all the roots outside the unit circle. Therefore, |phi| < 1. We can consider it as the partial autocorrelation between zt and zt-k. And

AR(1): $z_t = \phi_{11} z_{t-1} + a_t \Rightarrow \phi_{11} = \rho_1$.

AR(2): $\rho_1 = \phi_{21} + \rho_1 \phi_{22},\ \rho_2 = \rho_1 \phi_{21} + \phi_{22} \Rightarrow \phi_{22} = \frac{\rho_2 - \rho_1^2}{1 - \rho_1^2}$

Using the Yule-walker Equations to detect the order p for AR(p) model

$$\begin{bmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_k \end{bmatrix} = \begin{bmatrix} 1 & \rho_1 & \cdots & \rho_{k-1} \\ \rho_1 & 1 & \cdots & \rho_{k-2} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{k-1} & \rho_{k-2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} \phi_{k1} \\ \phi_{k2} \\ \vdots \\ \phi_{kk} \end{bmatrix}$$

$$\phi_{kk} = \begin{vmatrix} 1 & \rho_1 & \cdots & \rho_{k-2} & \rho_1 \\ \rho_1 & 1 & \cdots & \rho_{k-3} & \rho_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \rho_{k-1} & \rho_{k-2} & \cdots & \rho_1 & \rho_k \end{vmatrix} \begin{vmatrix} 1 & \rho_1 & \cdots & \rho_{k-1} \\ \rho_1 & 1 & \cdots & \rho_{k-2} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{k-1} & \rho_{k-2} & \cdots & 1 \end{vmatrix}^{-1}$$

AR(p) Model:

- ACF: Combinations of damped exponentials / sinusoidal waves
- PACF: zero for lags larger than p

So if we want to know the order p from the AR(p) model, we need to mainly depend on the PACF Plot to see when the PACF Plot disappears. And the order p is the smallest value when the PACF becomes zero.

MA(q) Model:

$$z_t - \mu = (1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q) a_t = \theta(B) a_t$$

- ACF: Cut off after lag q (autocorrelation k is equal to 0 and k >q)
- PACF: Combinations of damped exponentials / sinusoidal waves

So if we want to know the order q from the MA(q) model, we need to concentrate on the ACF Plot to see when the ACF Plot becomes zero.

$$
\begin{aligned}
\gamma_0 &= (1+\theta_1^2+\cdots+\theta_q^2)\sigma^2 \\
\gamma_k &= (-\theta_k+\theta_1\theta_{k+1}+\cdots+\theta_{q-k}\theta_q)\sigma^2,\ k=1,2,\ldots,q \\
\gamma_k &= 0,\ k>q \\
\rho_k &= \frac{-\theta_k+\theta_1\theta_{k+1}+\cdots+\theta_{q-k}\theta_q}{1+\theta_1^2+\cdots+\theta_q^2},\ k=1,2,\ldots,q \\
\rho_k &= 0, k>q
\end{aligned}
$$

ARMA(p,q) Model:

$$(1-\phi_1 B-\cdots\phi_p B^p)(z_t-\mu)=(1-\theta_1 B-\cdots-\theta_q B^q)a_t$$

or

$$
\begin{aligned}
z_t-\mu &= \phi_1(z_{t-1}-\mu)+\cdots+\phi_p(z_{t-p}-\mu) \\
&+ a_t-\theta_1 a_{t-1}-\cdots-\theta_q a_{t-q}
\end{aligned}
$$

- $k>q-p$, ACF dominated by AR part
- $k>p-q$, PACF dominated by MA part

If the ACF plot exhibits substantial spikes at specific lags, it indicates that the time series has a significant connection with its lagged values at those delays. The number of notable spikes in the ACF plot may indicate the MA model's order. The PACF graphic depicts the partial correlation between the time series and its lagged values while accounting for intermediate delays. The lag is shown by the x-axis of the PACF plot, while the partial correlation coefficient between the time series and its lagged values is represented by the y-axis.

Aside from the naive measure shown in the ACF Plot and PACF Plot, we can also consider the Akaike Information Criterion (AIC) and Schwarz-Bayesian Information Criterion (BIC) as selection rules. And the formulas have been shown in the following:

$$\text{AIC}=\frac{-2}{T}\ln(\text{likelihood})+\frac{2}{T}\times(\text{number of parameters}),$$

Where the likelihood function is evaluated at the maximum-likelihood estimates and T is the sample size. For a Gaussian AR(l) model, AIC can be simplified into this:

$\mathrm{AIC}(\ell) = \ln(\tilde{\sigma}_\ell^2) + \frac{2\ell}{T}$ and the second term is called the penalty function of the criterion because it penalizes a candidate model by the model of parameters used. Different penalty functions result in different information criteria. Another formula to calculate the BIC has been shown here: $\mathrm{BIC}(\ell) = \ln(\tilde{\sigma}_\ell^2) + \frac{\ell \ln(T)}{T}$ So when selecting the order p for the AR model, we need to combine these two measures together.

Before applying traditional linear time series models, it is crucial to ensure that the underlying time series data is stationary. Stationarity is a fundamental assumption for many time series models as it allows for the application of statistical techniques that rely on the data exhibiting consistent properties over time. To determine stationarity, one commonly used test is the Augmented Dickey-Fuller (ADF) unit-root test. The null hypothesis of the ADF test is that a unit root exists in the sample time series, indicating non-stationarity. If the calculated p-value from the ADF test is smaller than the chosen significance level, we can reject the null hypothesis and consider the sample time series to be stationary.

However, if the p-value is above the significance level, indicating the presence of a unit root, we need to employ approaches to transform the non-stationary time series into a stationary one. Two common methods for achieving this are the Differencing Method and Log Transformation.

Differencing Method: Differencing involves computing the difference between consecutive observations in a time series. The goal of differencing is to eliminate or minimize the level variations, trends, and seasonality present in the data, thereby making the series stationary. By subtracting the previous observation from the current one, differencing can help stabilize the mean of the series. Actually, the differencing periods in this model have been taken into account by the ARIMA (p,d,q) model, where d denotes the number of differencing. Subtracting successive observations is a necessary step in each differencing step, often with a latency of 1. After every step, the differenced series is acquired, and its stationarity is examined.

Log Transformation: The Log Transformation is a method suitable for dealing with the change rate or relative differences in a time series. It involves taking the logarithm of the observations, which can help stabilize the variance of the data. This is particularly useful when the data exhibits exponential growth or large variations over time. The log transformation can compress large values and expand small values, resulting in a more symmetrical distribution and reducing the impact of extreme values.

By applying these methods, we can transform the non-stationary time series into a stationary one, allowing us to apply traditional time series models that assume linearity and stationary data. These models include autoregressive integrated moving average (ARIMA), autoregressive (AR), moving average (MA), and their combinations.
Time series analysis involves estimating these components and using them to make predictions or draw insights. Techniques like moving averages, exponential smoothing, and **autoregressive integrated moving average** (ARIMA) models are widely used for this purpose. Its analysis provides a mathematical framework to understand and analyze the patterns, trends, and relationships in stock price data. By using mathematical notation and visualizations like line plots, we can gain insights into the behavior of stock prices over time and make informed decisions.

Then, a popular time series method for analyzing stock price fluctuations is the Autoregressive Integrated Moving Average (ARIMA) model, which would be introduced, and is a powerful tool for forecasting time series data, including stock prices. It combines three components: autoregressive (AR), differencing (I), and moving average (MA).

$$\phi(B)(1-B)^d z_t = \theta_0 + \theta(B)a_t$$

Autoregressive (AR) component: The AR component represents the linear relationship between an observation in the time series and a certain number of lagged observations. It captures the dependency of the current stock price on its past values. The order of the AR component, denoted as , indicates the number of lagged observations used in the model. Differencing () component: The component is used to make the time series stationary, which means removing any trend or seasonality. Differencing involves taking the difference between consecutive observations. The order of differencing, denoted as , represents the number of times differencing is applied to achieve stationarity. Moving Average (MA) component: The MA component models the dependency between an observation and a residual error from a moving average of the lagged observations. It captures the short-term fluctuations in the stock price. The order of the MA component, denoted as , indicates the number of lagged residual errors used in the model.

To determine the appropriate values for , and , one can use techniques such as the autocorrelation function (ACF) and partial autocorrelation function (PACF) plots. These plots provide insights into the correlation between the current observation and its lagged values. Once the ARIMA model is fitted to the stock price data, it can be used to forecast future prices. The model considers the historical data and any underlying patterns or dependencies to make predictions. It's important to note that the ARIMA model assumes that the underlying data follows a stationary process, meaning that its statistical properties do not change over time. However, in practice, stock prices often exhibit non-stationarity and other complexities. In such cases, more advanced models like the autoregressive integrated moving average with exogenous variables (ARIMAX) or seasonal ARIMA (SARIMA) can be used.

2.1.2 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN) can be considered another approach to deal with the time series data, and a typical class of neural networks including the hidden states to transfer and track the previous outputs as the next layer's inputs. Recurrent neural networks (RNNs) are much more effective in predicting what will happen next in a sequence than feedforward neural networks because they can process input sequences using their internal state, or memory. Information can be transferred from one network phase to the next thanks to loops in the architecture of an RNN. A 'hidden state,' formed by this recurrence, records details about the calculations that have been made thus far. Essentially, the network's memory is the concealed state. The most distinctive aspect of an RNN is how its memory is used to process inputs and produce a hidden state that causes each echo to loop just shown in the following diagram (Figure 2. The Loop Structure in the RNN).

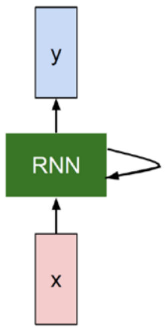


Figure 2.1 : The Loop Structure in the RNN

The fresh input and the most recent hidden state, which comprises information from earlier inputs, are contained within the loop and its output. RNN would function on a sequential dataset because, when combined, they provide the intermediary between anterior sequences of data and perspective data. And the approximate procedure and logic have been shown in the following diagram.

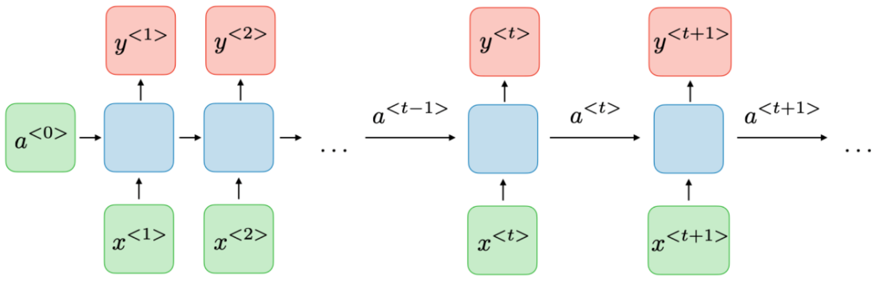
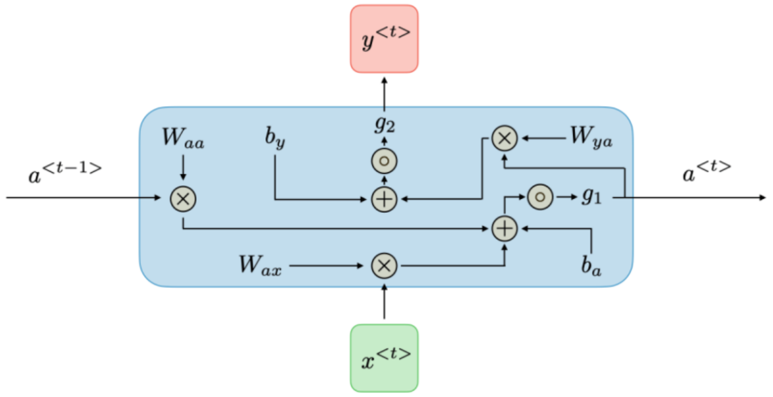


Figure 2.2: Multiple Structure RNN

And for the every time interval (time step t), the activation and the output as follows:

$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

Where W*ax*, W*aa*, W*ya*, b*a*, b*y* are coefficients that are shared temporally and g1, g2 activation functions. The following diagram shows the calculation logic between the a <*t-1*> and a <*t*>.



In our time series application in the RNN, the many-to-one is the most popular application in predicting the stock prices.

Time series prediction is the process of forecasting a time series' future values using its historical data. Applications in weather forecasting, finance forecasting, and other fields are closely connected. This is the one we would employ, with a many-to-one architecture. Future price projections are shown by the successive outputs just shown in the following diagram structure (Figure 2. : Many-to-one Application of Recurrent Neural Networks) In the following diagram, it means that we can get sequence data as input and generate one output. Under the time series application, it shows us the future output or forecast relies on the previous results with different weights.

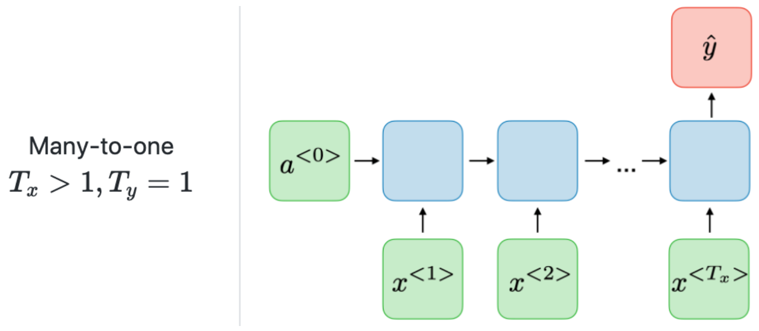


Figure 2.3 : Many-to-One Structure & Illustration

Long Short-Term Memory (LSTM) Networks are a class of sequential neural networks that leverage deep learning capabilities to effectively retain and utilize information over time. They address the vanishing gradient problem encountered by traditional recurrent neural networks (RNNs) through the utilization of the Backpropagation Method. LSTM networks are distinguished by their memory blocks, which are interconnected across layers instead of individual neurons. Each memory block within an LSTM network comprises specialized gates that govern the block's state and output. Unlike individual neurons, these blocks operate on input sequences, and each gate within a block employs sigmoid activation units to regulate its activation. This gate-based mechanism enables the LSTM network to conditionally control the flow of information through the blocks, facilitating the modification of the state and the incorporation of new information. Within an LSTM unit, three key gates are employed: the Forget Gate, the Input Gate, and the Output Gate. The Forget Gate plays a crucial role in determining which features or aspects of the input sequence should be discarded or disregarded. The Input Gate, on the other hand, determines the specific values from the input that need to be updated within the memory state. Lastly, the Output Gate is responsible for determining the information or data that should be conveyed as the output of the block, thus shaping the memory retention of the block. By leveraging these gates and their associated functionalities, LSTM networks can effectively capture and utilize long-term dependencies in sequential data, making them particularly well-suited for analyzing and modeling complex temporal patterns in domains such as finance, including Hong Kong Financial Time Series Data.

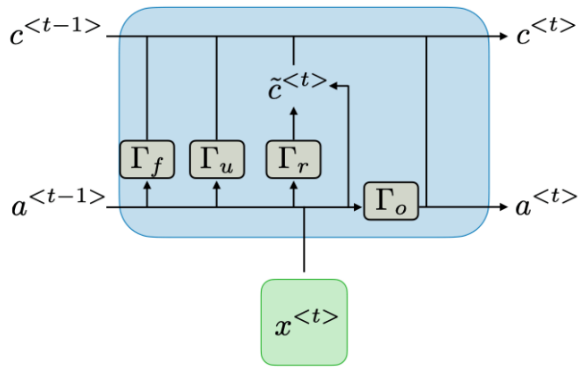
Characterization	**Long Short-Term Memory**
$c^{\wedge}<t>$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$
$c<t>$	$\Gamma_u \star c^{\wedge}<t> + \Gamma_f \star c<t-1>$
$a<t>$	$\Gamma_o \star c<t>$
Dependencies	 $c^{<t-1>}$ $c^{<t>}$ $\tilde{c}^{<t>}$ Γ_f Γ_u Γ_r Γ_o $a^{<t-1>}$ $a^{<t>}$ $x^{<t>}$

Figure 2.4 :LSTM Basic Parameters and Structure

Long Short-Term Memory (LSTM) networks are excellent at capturing long-term dependencies and are generally regarded as best practices for sequence prediction problems. They are often used in jobs that involve sequences and time series. The unique strength of LSTM is its capacity to capture and preserve order dependency, which is essential for resolving complex situations.

When models are trained over lengthy periods, disappearing and exploding gradients provide a problem for conventional Recurrent Neural Networks (RNNs). In order to solve this problem, LSTM networks include a gating mechanism that either selectively remembers or forgets information. Even in situations when there is a considerable lag between pertinent events in the sequence, these gates allow the LSTM model to recognize and retain the pertinent context. Because of this, LSTMs are especially useful for jobs like machine translation where context knowledge is crucial.

It is noteworthy, nonetheless, that LSTM networks are computationally more costly than more straightforward structures, such as feed-forward neural networks. This may restrict their capacity to scale in situations involving huge datasets or in limited computing conditions. since of their computational complexity, training LSTM networks can also take longer since they need more data and longer training periods to reach high performance.

Parallelizing the processing of sentences becomes difficult since LSTM processes input sequences word by word in a sequential fashion. The sequential structure of this work hinders the effective allocation of the workload among several processors or GPUs, potentially affecting the total processing speed and scalability of models based on LSTMs.

Notwithstanding these drawbacks, long-term dependencies and sequential context are two of the most important things that LSTM networks are good at, which makes them useful in a lot of different fields, such time series analysis, speech recognition, and natural language processing. It is important to carefully consider the trade-off between computational complexity and performance in light of the particular needs and limitations of the work at hand.

2.2 Implementation

2.2.1 Time Series Model

To put the traditional time series model into practice in my chosen four stocks, I will focus on the following procedures to conduct step by step:

1. Data Collection and Processing
    - Download the data from Bloomberg, ensure the complete and accuracy of these stocks prices
    - Remove non-trading days and account for limit up and limit down scenarios, ensuring data consistency

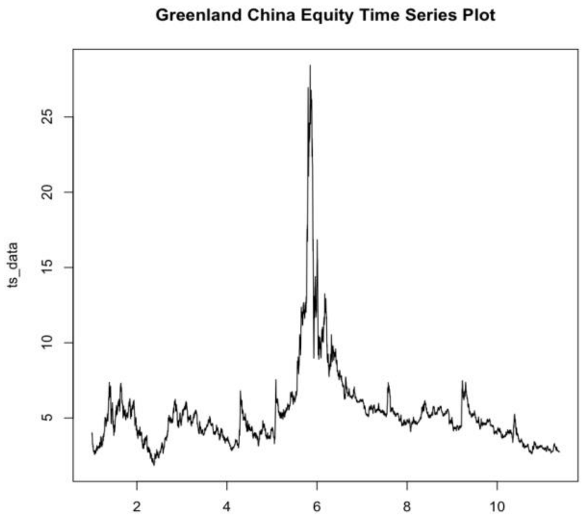
In this step, if I choose the stock prices as time series data, there is no additional procedure to deal with the outliers and missing values.

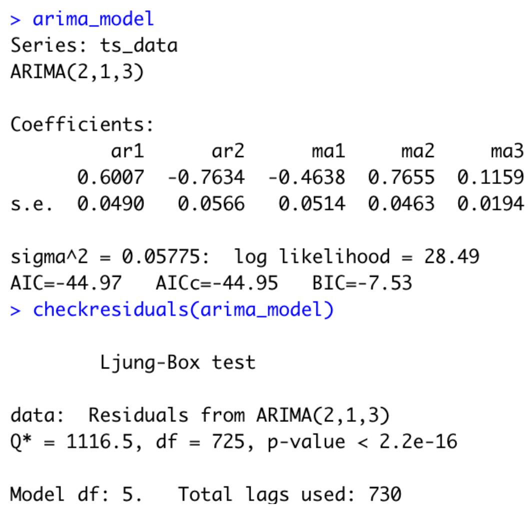
2. Initial Attempt using ARIMA model with stock prices:
    - Apply the auto.arima function in the RStudio, which can help us to differentiate the non-stationary series and automatically match the parameters p, d, q to fit the original data to the best ARIMA model
    - Draw the residuals of the ARIMA model and check the residuals by Ljung-Box Test to see if the residuals are not distinguishable with the white noise series

   - If not, try to change the time series data from the stock prices into the returns
3. Transfer the time series data into the daily returns
   - Proceed with transferring the time series data from stock prices to daily returns for each individual stock
   - Deal with the outliers, missing values for the daily returns
   - Repeat the same logic as in the initial attempt using ARIMA with the transformed daily return time series
   - Assess whether this approach yields better results in terms of forecasting accuracy and detecting lag and autocorrelation
4. Alternative approach: Section 2.2.2 Recurrent Neural Networks

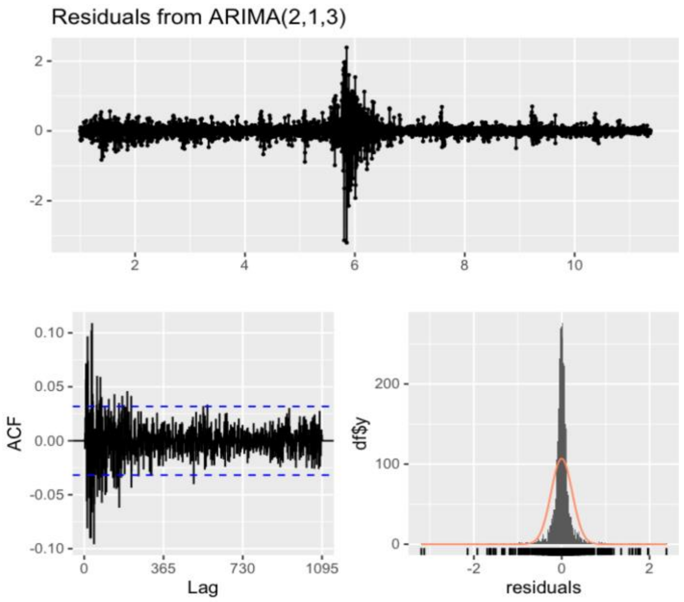
During this process, I also add the prediction using the Holt-Winters Model to check if there exist some seasonality or trend because this model estimates the current trend and modifies it over time using exponentially weighted moving averages. While trend may also be taken into account by other time series models, their methods may differ, for example, by using polynomial regression in some circumstances or autoregressive factors in ARIMA models. And this model estimates the seasonal component and makes adjustments to it over time using seasonal smoothing parameters. Some time series models, on the other hand, might not explicitly account for seasonality or might call for extra procedures to manage seasonal trends, including deseasonalizing the data before using the model.

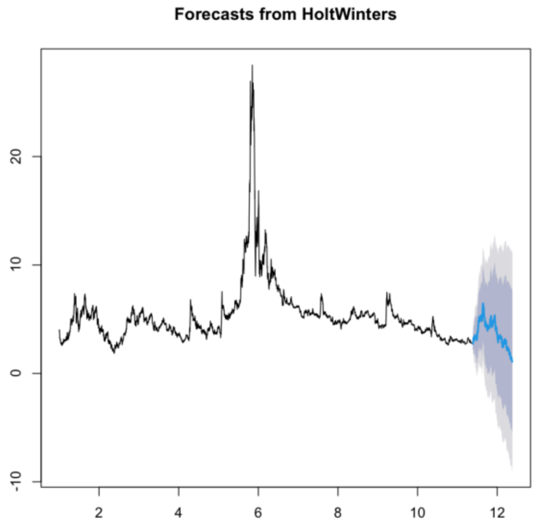
Initial Attempt: Traditional ARIMA Model for four individual stocks' outputs

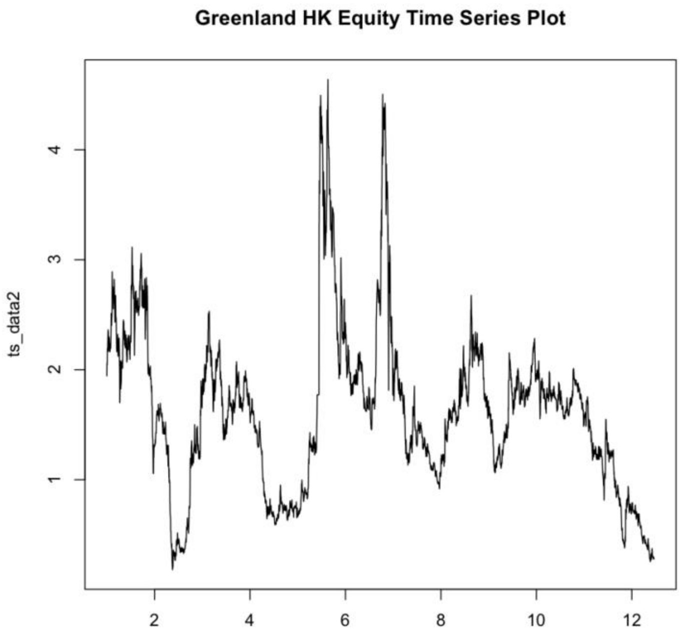




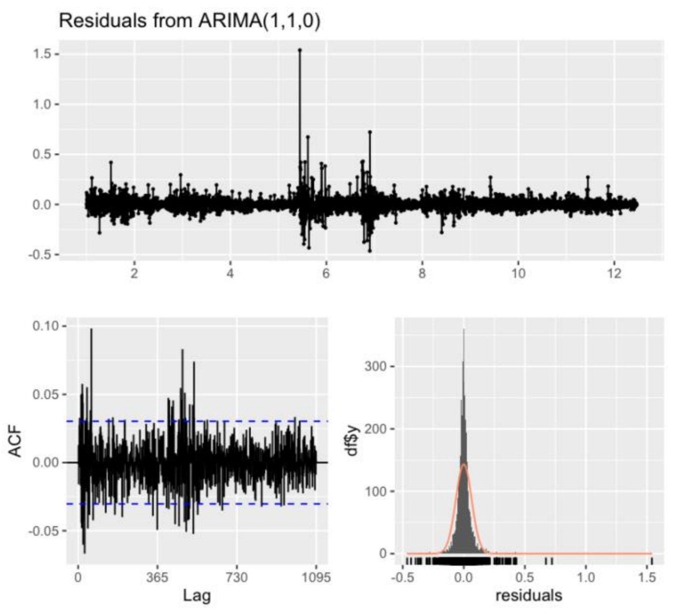
```
> arima_model
Series: ts_data
ARIMA(2,1,3)

Coefficients:
         ar1      ar2      ma1     ma2     ma3
      0.6007  -0.7634  -0.4638  0.7655  0.1159
s.e.  0.0490   0.0566   0.0514  0.0463  0.0194

sigma^2 = 0.05775:  log likelihood = 28.49
AIC=-44.97   AICc=-44.95   BIC=-7.53
> checkresiduals(arima_model)

        Ljung-Box test

data:  Residuals from ARIMA(2,1,3)
Q* = 1116.5, df = 725, p-value < 2.2e-16

Model df: 5.   Total lags used: 730
```

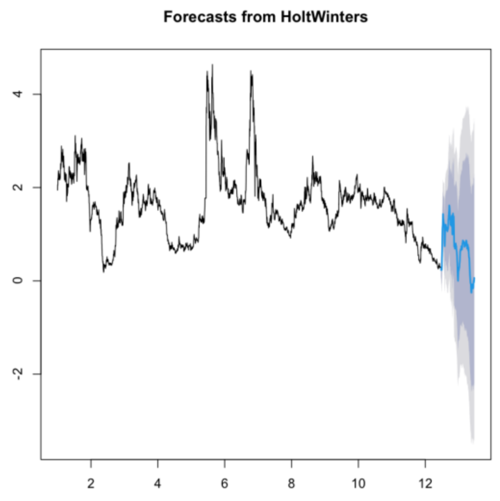






```
> arima_model2
Series: ts_data2
ARIMA(1,1,0)

Coefficients:
         ar1
      0.0877
s.e.  0.0154

sigma^2 = 0.004213:  log likelihood = 5508.87
AIC=-11013.74   AICc=-11013.74   BIC=-11001.07
> checkresiduals(arima_model2)

        Ljung-Box test

data:  Residuals from ARIMA(1,1,0)
Q* = 1135.5, df = 729, p-value < 2.2e-16

Model df: 1.   Total lags used: 730
```





By fitting ARIMA (2,1,3) and ARIMA (1,1,0) models to the Greenland Holdings Hong Kong and Greenland Holdings China Equity Pair, we can evaluate their goodness of fit using the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC). The selection criteria for these models prioritize the smallest possible AIC and BIC values, indicating a better fit to the original data. Additionally, a larger log-likelihood signifies that the ARIMA (2,1,3) and ARIMA (1,1,0) models have made significant efforts to capture the characteristics of the original data.

Analyzing the residuals of these models, we observe distinct characteristics in their distributions. For Greenland China, the residuals exhibit a nearly zero mean, but the left tail appears thicker than the right tail, indicating left skewness. The distribution of these residuals approximates a normal distribution. Conversely, for Greenland Hong Kong, the residuals also exhibit a mean close to zero, but the right tail appears thicker than the left tail, indicating right skewness. This suggests the presence of outliers on the right side of the distribution.

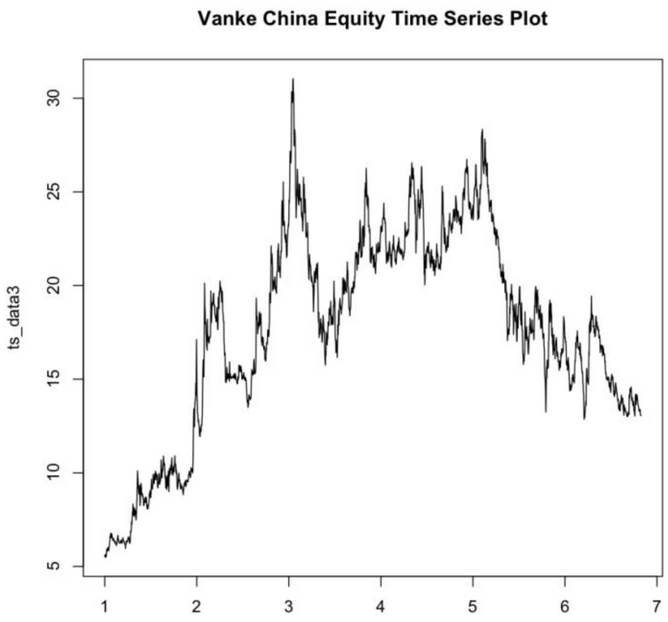
Conducting the Ljung-Box test, we assess the autocorrelation of the residuals. The obtained p-values are significantly smaller than the chosen significance level (0.05), providing sufficient evidence to reject the null hypothesis that the residuals follow a white noise process. This implies that the ARIMA models are unable to fully capture all the nonlinear features present in the original datasets, resulting in residual dependencies.

In summary, the ARIMA models, despite their efforts to fit the data, exhibit limitations in capturing all the nonlinear characteristics of the original datasets. The residuals show indications of correlation, which suggests the presence of unexplained patterns within the data. These findings emphasize the need for alternative modeling approaches that can better capture the complexities and nonlinearity present in the relationship between the Greenland Holdings Hong Kong and Greenland Holdings China Equity Pair.

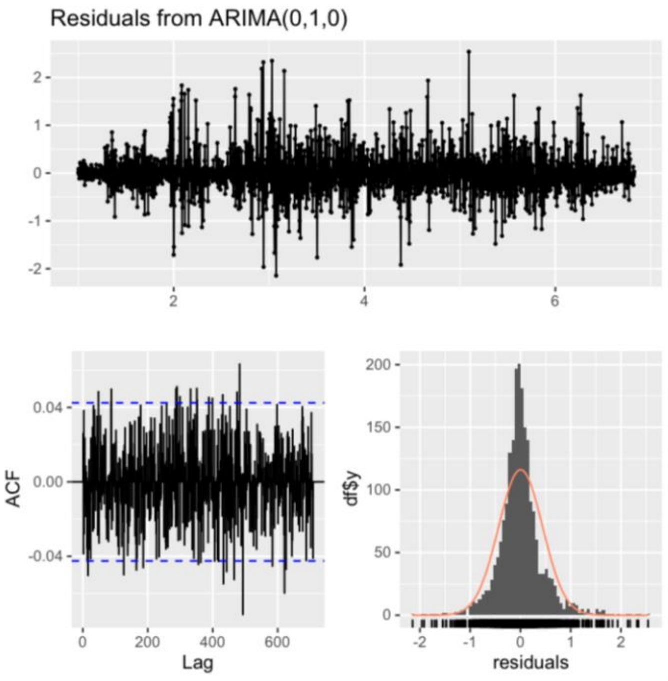
In the next two pages, we can see the ARIMA(0,1,0) model for Vanke Hong Kong and ARIMA(2,1,2) model for Vanke China exhibit poor performance based on their AIC and BIC values. These selection criteria aim to find the best-fitting models, but in this case, they indicate that the ARIMA models are not suitable for capturing the characteristics of the original data. The extremely large AIC and BIC values, coupled with negative log-likelihood, suggest that the ARIMA models fail to adequately fit the data.

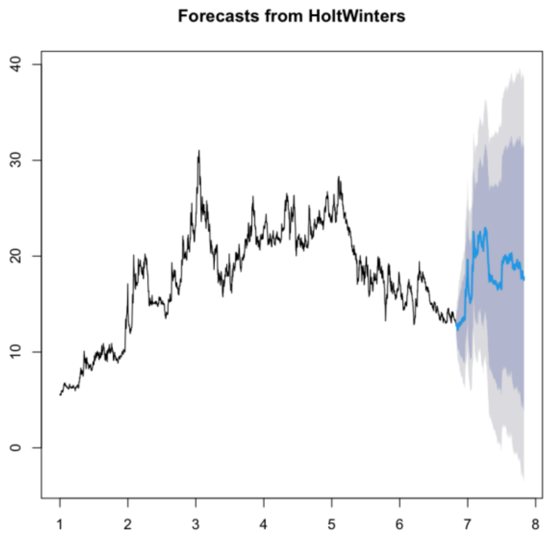
Furthermore, the Ljung-Box Test reveals the presence of residuals in the relationship that cannot be captured by the ARIMA models. The p-values obtained from the test indicate that the residuals exhibit significant autocorrelation, implying that the linear relationship assumed by the ARIMA models is insufficient to capture the underlying dynamics.

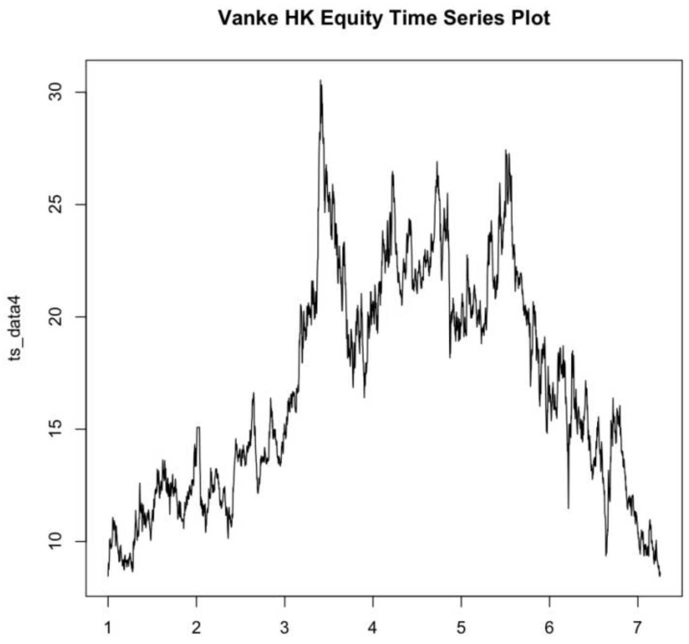
Given these limitations, it may be necessary to explore alternative approaches and abandon the use of stock prices as the time series. The ARIMA models heavily rely on the assumption of a linear relationship, which may not hold in the case of the Vanke Hong Kong and Vanke China data. Therefore, it is important to consider other modeling techniques that can capture the nonlinear nature and potentially uncover more complex relationships between the variables.



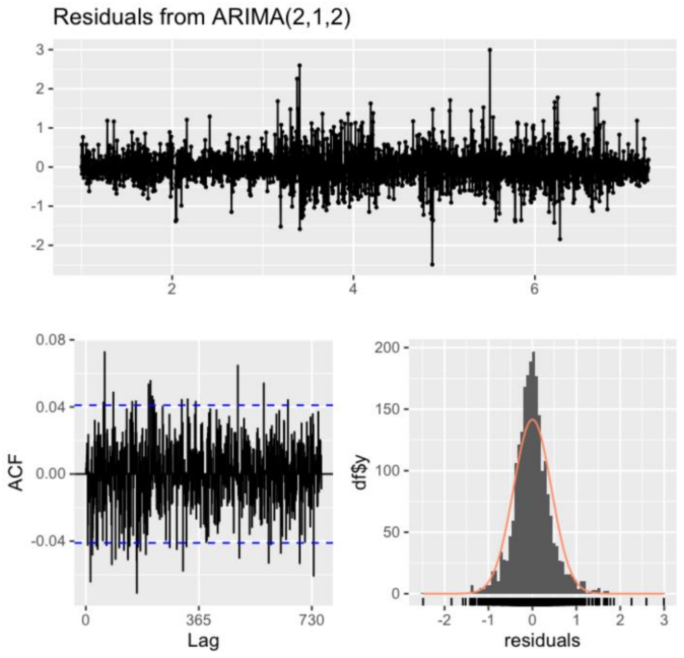
```
> arima_model3
Series: ts_data3
ARIMA(0,1,0)

sigma^2 = 0.1926:  log likelihood = -1266.44
AIC=2534.88   AICc=2534.89   BIC=2540.55
> checkresiduals(arima_model3)

        Ljung-Box test

data:  Residuals from ARIMA(0,1,0)
Q* = 481.39, df = 426, p-value = 0.03261

Model df: 0.   Total lags used: 426
```

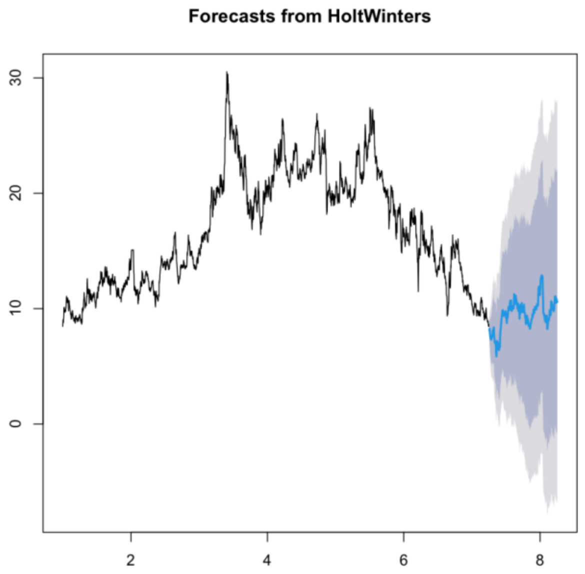




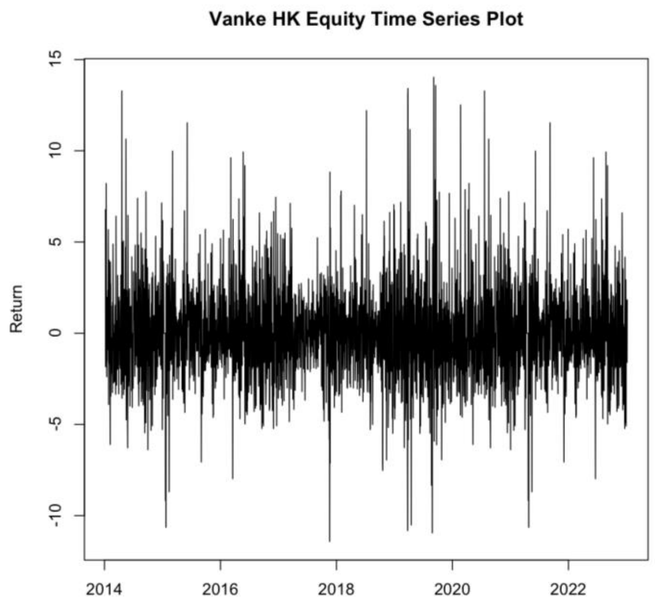


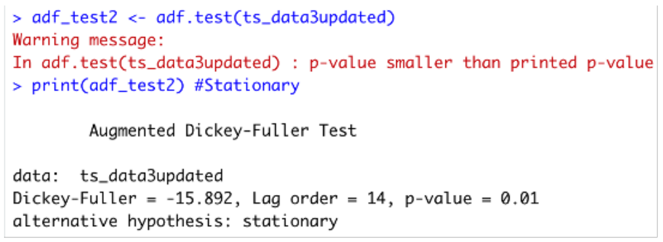
```
> arima_model4
Series: ts_data4
ARIMA(2,1,2)

Coefficients:
          ar1      ar2     ma1     ma2
      -0.8539  -0.2207  0.8732  0.1853
s.e.   2.1019   1.5391  2.1005  1.6210

sigma^2 = 0.1855:  log likelihood = -1313.68
AIC=2637.36   AICc=2637.38   BIC=2666.02
> checkresiduals(arima_model4)

        Ljung-Box test

data:  Residuals from ARIMA(2,1,2)
Q* = 573.37, df = 453, p-value = 0.0001025

Model df: 4.   Total lags used: 457
```

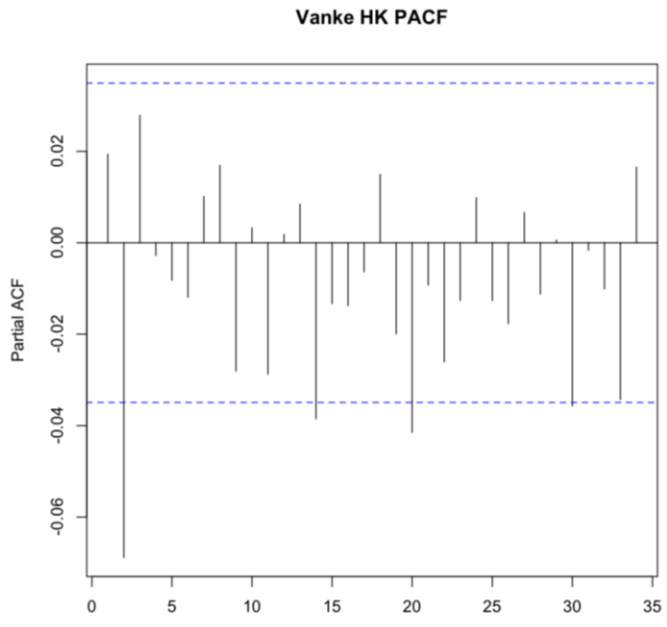


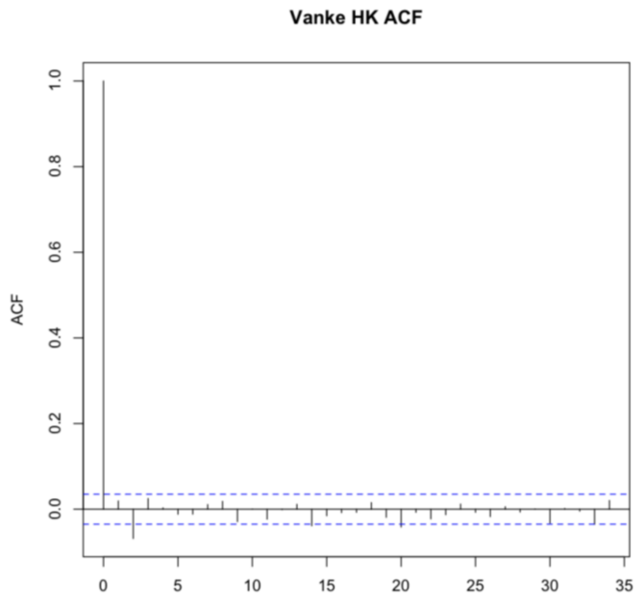


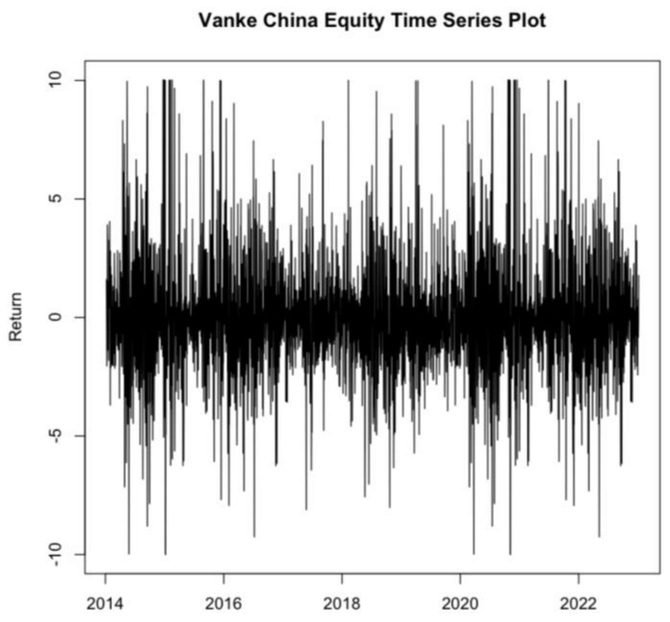
Second Attempt using the daily return as time series data output:

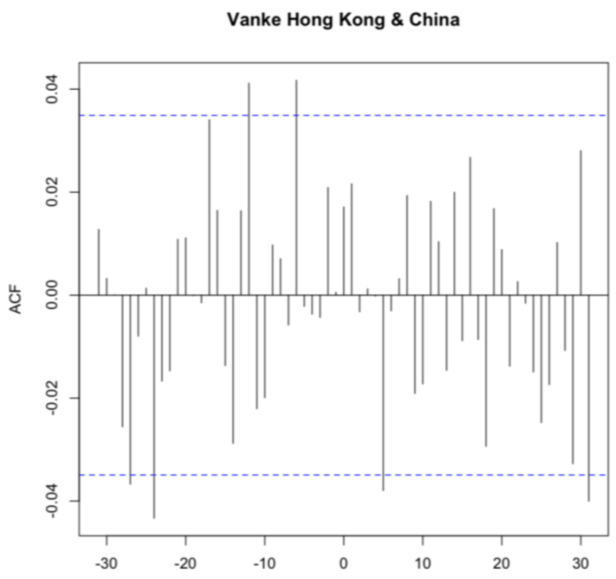


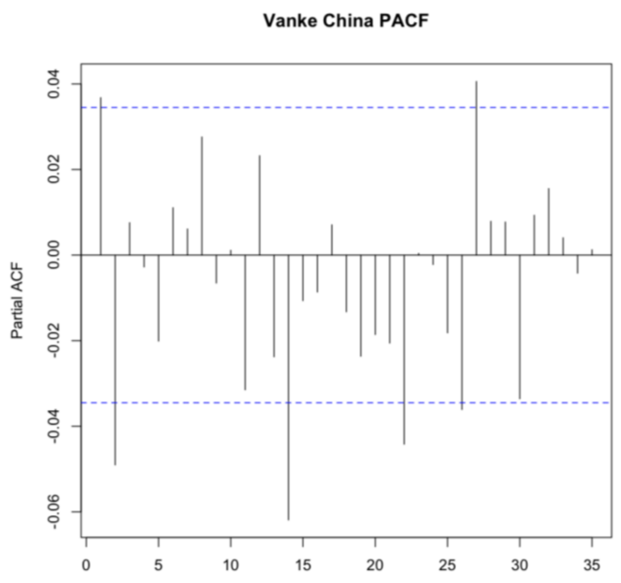

```
> adf_test2 <- adf.test(ts_data3updated)
Warning message:
In adf.test(ts_data3updated) : p-value smaller than printed p-value
> print(adf_test2) #Stationary

        Augmented Dickey-Fuller Test

data:  ts_data3updated
Dickey-Fuller = -15.892, Lag order = 14, p-value = 0.01
alternative hypothesis: stationary
```

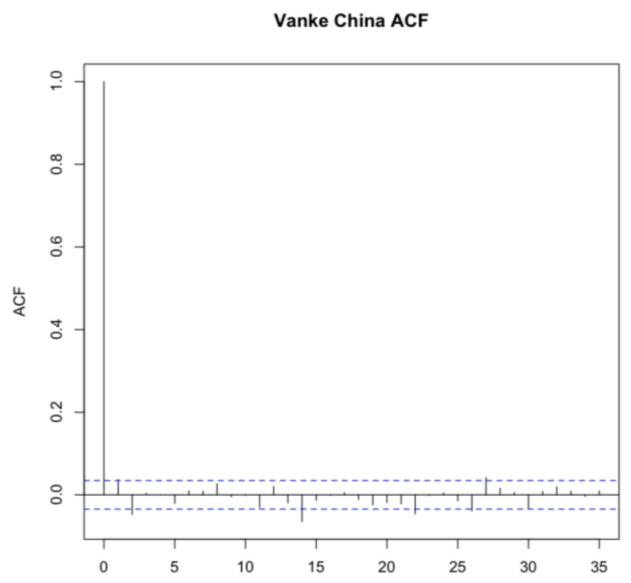












The negative lags of -20 for Greenland China and Greenland Hong Kong, and -24 for the Vanke pair, provide valuable insights into the relationship and potential dynamics between the two markets. These negative lags signify those changes in the daily returns of China's real estate sector precede or lead changes in Hong Kong's daily returns. This suggests a temporal relationship where developments in the Chinese market have an impact on the Hong Kong market, with a certain time delay.

The negative lags imply that it takes more than 20 trading days for the Hong Kong real estate sector to react to changes in the China real estate sector. This delay can be attributed to various factors, such as the differences in market structure, trading hours, and regulatory environments between the two markets. It highlights that the Hong Kong real estate market is influenced by and responds to trends and events in the Chinese real estate market, albeit with a time lag.

However, it is important to note that these lags provide insights into the relationship between the markets from a linear perspective. While the negative lags help us understand the temporal aspects of their relationship, it's crucial to recognize that the linear correlations between China's and Hong Kong's stocks are weak. This indicates that their relationship is not solely driven by linear associations. Other non-linear factors, such as market sentiment, investor behavior, economic conditions, and geopolitical influences, may play significant roles in shaping the dynamics between the two markets.

By acknowledging the limitations of linear correlations and considering additional factors, we can gain a more comprehensive understanding of the complex and multifaceted relationship between China's and Hong Kong's real estate sectors. This broader perspective allows for a more nuanced analysis and interpretation of the interdependencies and influences between these markets.

2.2.2 Recurrent Neural Networks

From the above theorem section 2.1.2, it has shown the LSTM can be considered the best practices for sequence prediction tasks and excels in capturing long-term dependencies. So in this implementation stage, I choose the LSTM to train the model for the four individual stocks and compare the actual prices and predicted prices together to see whether the LSTM can return us a better practice in prediction compared to the traditional time series model.

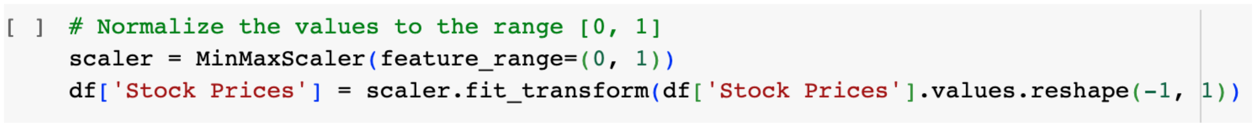
Here, I list the procedure using the RNN and the LSTM methods to deal with the four individual stocks: Because I downloaded the data from Bloomberg Terminal, which helped me to delete the non-trading days and missing values, owing to the stocks' limit down and limit up, this kind of feature helps us to waive the procedure to delete the outliers.

Data preparation is a crucial step in the development of robust stock price prediction models. To ensure an effective approach, it is common practice to employ an 80-20 split ratio, allocating 64% of the dataset as the training set, 16% as the validation set, and the remaining 20% as the test set. However, prior to this division, it is necessary to normalize the original dataset, mapping it into the range of [0, 1].

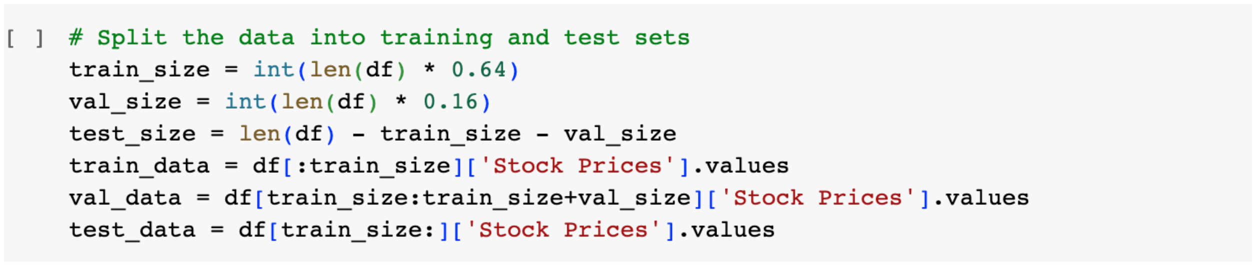
The normalization process brings forth several advantages that significantly enhance the overall performance and efficacy of stock price prediction projects. One prominent benefit is the elimination of scale dependency among features. By normalizing the data, we mitigate the potential dominance of any single feature within the prediction process, thus fostering a more balanced and accurate outcome.

Furthermore, data normalization plays a pivotal role in facilitating the optimization process, particularly when employing gradient descent algorithms. By transforming the data into the [0, 1] range, we promote a better convergence of these algorithms, consequently expediting the training of our recurrent neural network. This accelerated training empowers the model to effectively capture intricate temporal patterns, ultimately leading to improved accuracy in stock price predictions. By adhering to these practices and employing data normalization techniques, we can ensure a more robust and reliable foundation for the subsequent stages of model development and evaluation.

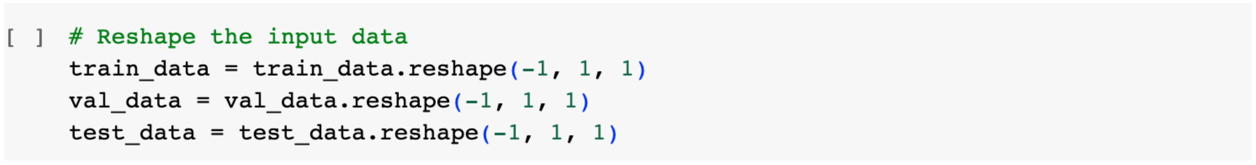
Based on the above, I apply the following code at the data preparation stage at the Google Colab:



```
# Normalize the values to the range [0, 1]
scaler = MinMaxScaler(feature_range=(0, 1))
df['Stock Prices'] = scaler.fit_transform(df['Stock Prices'].values.reshape(-1, 1))
```



```
# Split the data into training and test sets
train_size = int(len(df) * 0.64)
val_size = int(len(df) * 0.16)
test_size = len(df) - train_size - val_size
train_data = df[:train_size]['Stock Prices'].values
val_data = df[train_size:train_size+val_size]['Stock Prices'].values
test_data = df[train_size:]['Stock Prices'].values
```



```
# Reshape the input data
train_data = train_data.reshape(-1, 1, 1)
val_data = val_data.reshape(-1, 1, 1)
test_data = test_data.reshape(-1, 1, 1)
```

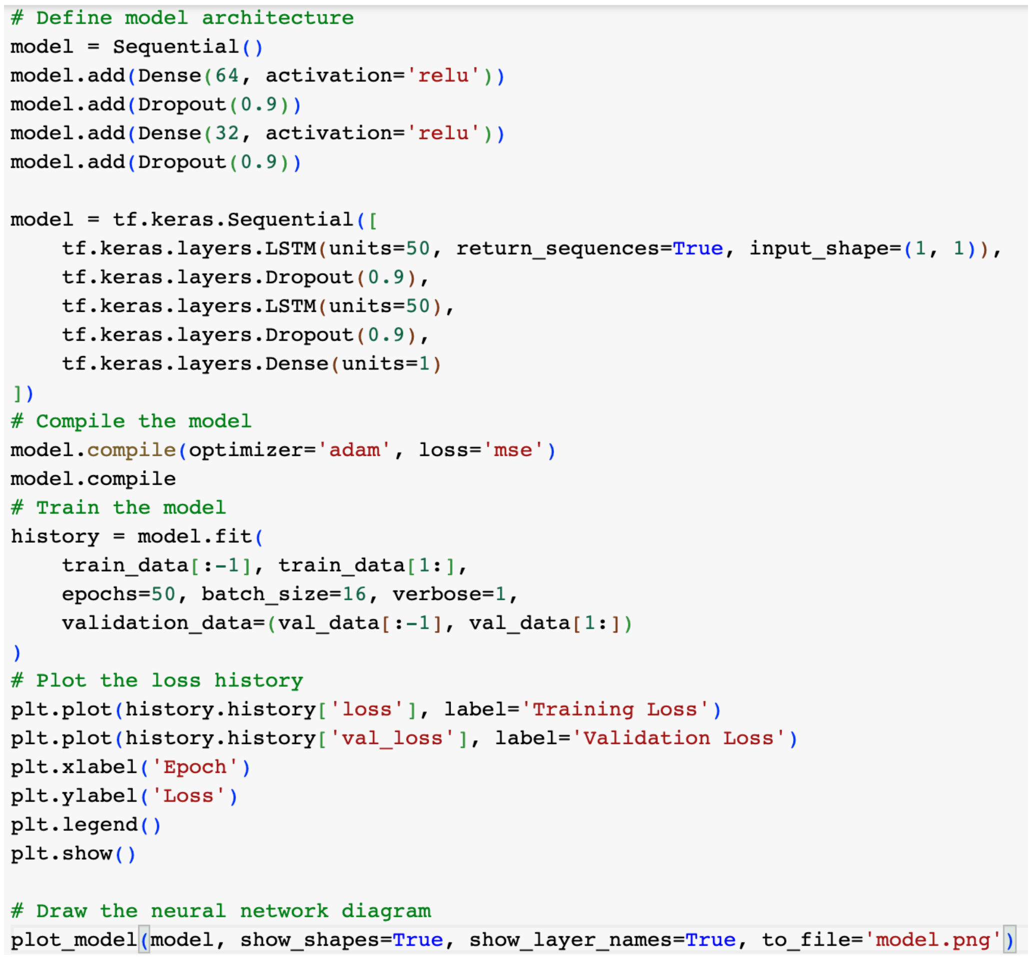
Modeling: Owing to the features of the Recurrent Neural Networks and Sequential Data, stock price data is inherently sequential just like the time series data, reshaping enables the model to capture the temporal dependencies effectively. The LSTM layers in the architecture contain memory cells that can retain information over long periods, helping the model capture long-term dependencies in the stock price data.

To mitigate the risk of overfitting, which can compromise the model's generalization ability, dropout layers are incorporated into both the feedforward neural network and LSTM-based architectures. Dropout randomly deactivated neurons during the training process, thus compelling the network to rely on different combinations of neurons and preventing an excessive reliance on specific features. This regularization technique improves the model's ability to generalize well to unseen data and reduces the likelihood of overfitting to the training data.

By thoughtfully integrating these components into the model's architecture, we can effectively leverage the strengths of RNNs and sequential data, enhancing the model's capacity to capture and learn intricate patterns within the stock price data.

The tf.keras.Sequential function provides a straightforward and intuitive way to define sequential models in TensorFlow. It allows stacking layers sequentially, specifying the desired architecture in a clear and concise manner. This simplicity makes it easier to experiment with different layer configurations and architectures, enabling us to iterate and refine model design efficiently. TensorFlow and Keras provide a powerful and widely used deep learning framework. By utilizing the tf.keras module, we can leverage the extensive functionality and optimization capabilities offered by TensorFlow while benefiting from the user-friendly and high-level API provided by Keras. This integration simplifies the development process and facilitates efficient model training and evaluation.

Considering the above factors, the approximate procedure about modeling has been attached here:



```python
# Define model architecture
model = Sequential()
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.9))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.9))

model = tf.keras.Sequential([
    tf.keras.layers.LSTM(units=50, return_sequences=True, input_shape=(1, 1)),
    tf.keras.layers.Dropout(0.9),
    tf.keras.layers.LSTM(units=50),
    tf.keras.layers.Dropout(0.9),
    tf.keras.layers.Dense(units=1)
])
# Compile the model
model.compile(optimizer='adam', loss='mse')
model.compile
# Train the model
history = model.fit(
    train_data[:-1], train_data[1:],
    epochs=50, batch_size=16, verbose=1,
    validation_data=(val_data[:-1], val_data[1:])
)
# Plot the loss history
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Draw the neural network diagram
plot_model(model, show_shapes=True, show_layer_names=True, to_file='model.png')
```

Considering the prediction result from the traditional time series model, here, we consider the activation function, ReLU at here, adding the nonlinearity into this training model, as well as two dense layers with 64 and 32 units. Dropout layers with a dropout rate are also added after each dense layer. Dropout is a regularization technique that randomly deactivated neurons during training to prevent overfitting.

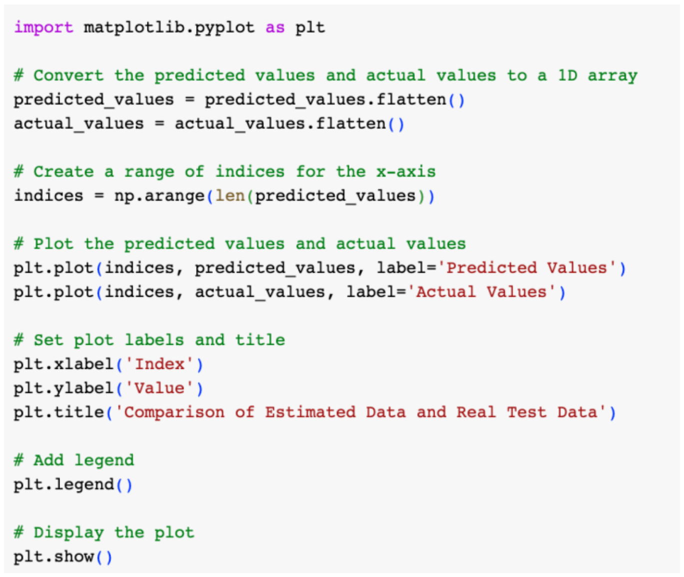
To ensure the accuracy of our model, we can utilize the training error and validation error to evaluate its performance. By visualizing these errors, we can gain insights into the model's behavior. Actually, we expect that the validation error is comparable to or slightly lower than the training error. This is a positive sign as it indicates that our model is not overfitting and can generalize well to unseen data.

When the validation error is significantly higher than the training error, it suggests that the model may be overfitting. Overfitting occurs when the model becomes too specialized in the training data and performs poorly on new, unseen data. However, since we have added the Dropout Layers, it has fewer possibilities for this scenario. After evaluating the model's performance, we proceed to draw the neural network diagram and print the model summary. The output shows the architecture of our model, which consists of two LSTM layers followed by two dropout layers.

```
# Print the model summary
model.summary()
```

```
# Perform prediction on the test data
predictions = model.predict(test_data)

# Convert the predicted values to their original scale using the scaler
predicted_values = scaler.inverse_transform(predictions.reshape(-1, 1))

# Convert the actual test data to their original scale using the scaler
actual_values = scaler.inverse_transform(test_data.reshape(-1, 1))

# Calculate accuracy (e.g., using mean squared error)
mse = np.mean((actual_values - predicted_values) ** 2)
rmse = np.sqrt(mse)

print(f"Root Mean Squared Error (RMSE) on Test Data: {rmse}")
```

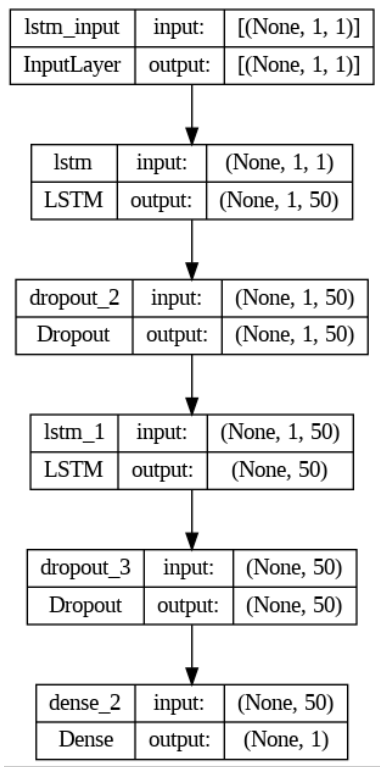
By examining the model summary and understanding the input and output shapes, we gain a better understanding of how our model is structured and the dimensions of the data it expects. This information is crucial for ensuring that the input data is properly aligned with the model's architecture and for interpreting the output predictions correctly. Transforming it into a more visualized version, we could gain the model structure. By comparing the actual values and our prediction values, we can check if there exist some factors that can't be quantified into our training models.

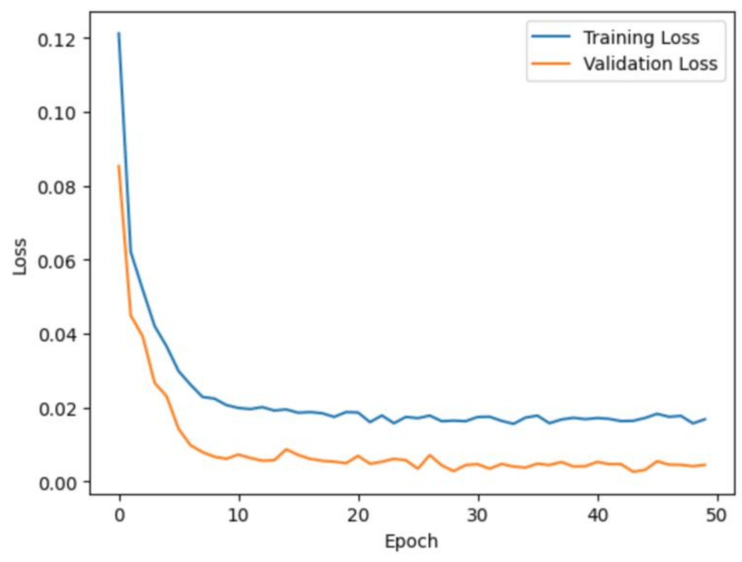


```python
import matplotlib.pyplot as plt

# Convert the predicted values and actual values to a 1D array
predicted_values = predicted_values.flatten()
actual_values = actual_values.flatten()

# Create a range of indices for the x-axis
indices = np.arange(len(predicted_values))

# Plot the predicted values and actual values
plt.plot(indices, predicted_values, label='Predicted Values')
plt.plot(indices, actual_values, label='Actual Values')

# Set plot labels and title
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Comparison of Estimated Data and Real Test Data')

# Add legend
plt.legend()

# Display the plot
plt.show()
```

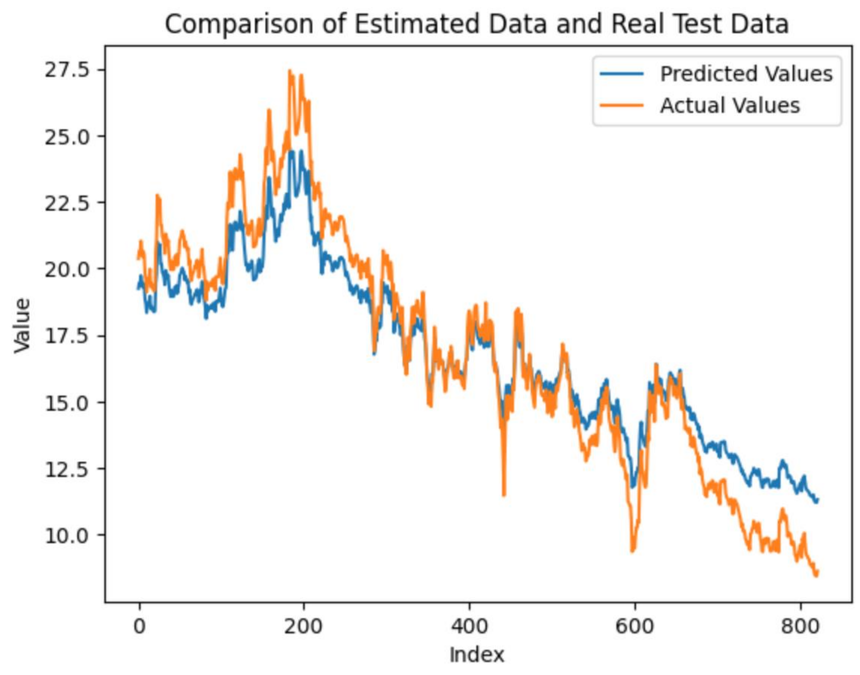
Vanke HK Equity (02202 HK) Output:

For Vanke HK Equity, I only focus on the range from 25th July 2014 to 29th September 2023 as our total datasets with the amount of 2283 pairs. So due to my 80-20 Split Ratios, there exists a training set with an amount of 1461, a validation set with an amount of 365, as well as a testing amount of 457. After the normalization of the original dataset, I add these two dense layers using the activation function ReLu with 64 and 32 units, respectively. As the model only takes one feature at a time, the input shape is given as (1, 1). The return_sequences=True option is used to guarantee that the output of the first LSTM layer is provided as input to the second LSTM layer. Each LSTM layer has 50 units. To avoid overfitting, dropout layers are added after each LSTM layer. The last layer is a thick layer with one unit, which stands for the anticipated price of the stock. And the training loss and validation loss from the training model are both relatively small, reflecting the model's accuracy. The model shows two LSTM models and two dropout layers inside. For the "(None,1,1)" in the input layer, "None" means the model can accept a variable number of input samples. And the (1,1) means every input one for date and one for the stock prices.

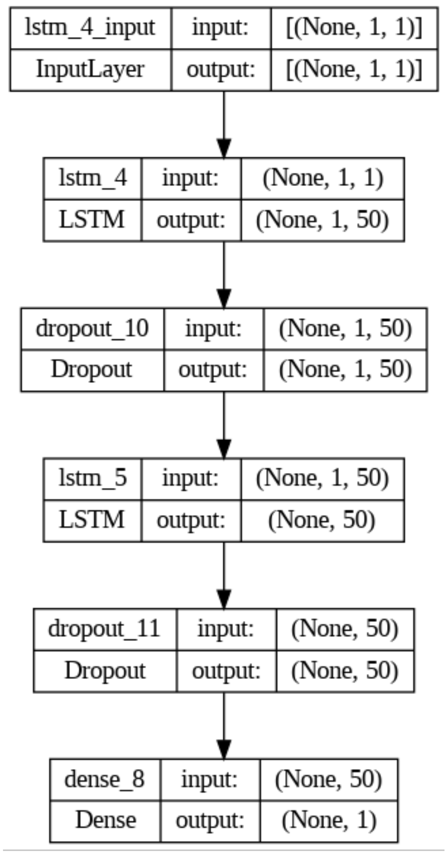

lstm_input | InputLayer | input: [(None, 1, 1)] | output: [(None, 1, 1)]
lstm | LSTM | input: (None, 1, 1) | output: (None, 1, 50)
dropout_2 | Dropout | input: (None, 1, 50) | output: (None, 1, 50)
lstm_1 | LSTM | input: (None, 1, 50) | output: (None, 50)
dropout_3 | Dropout | input: (None, 50) | output: (None, 50)
dense_2 | Dense | input: (None, 50) | output: (None, 1)

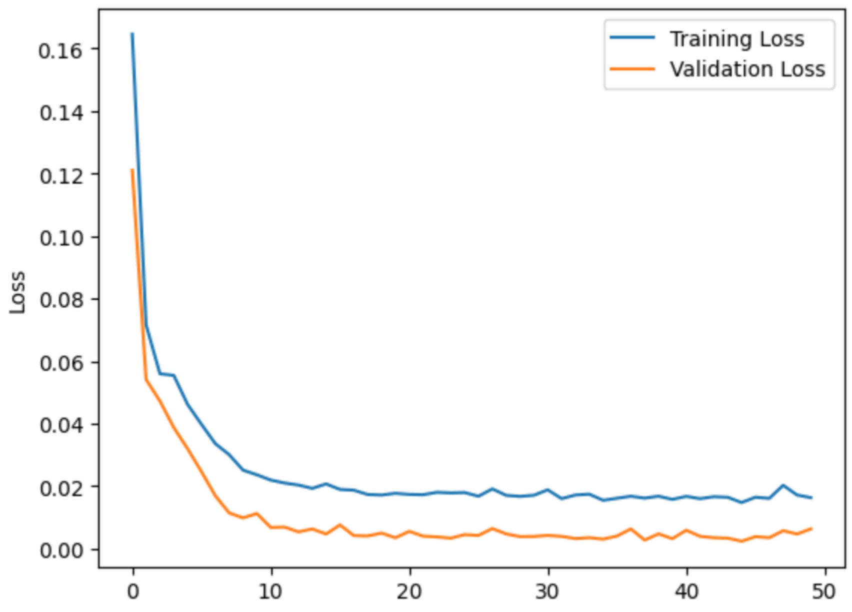


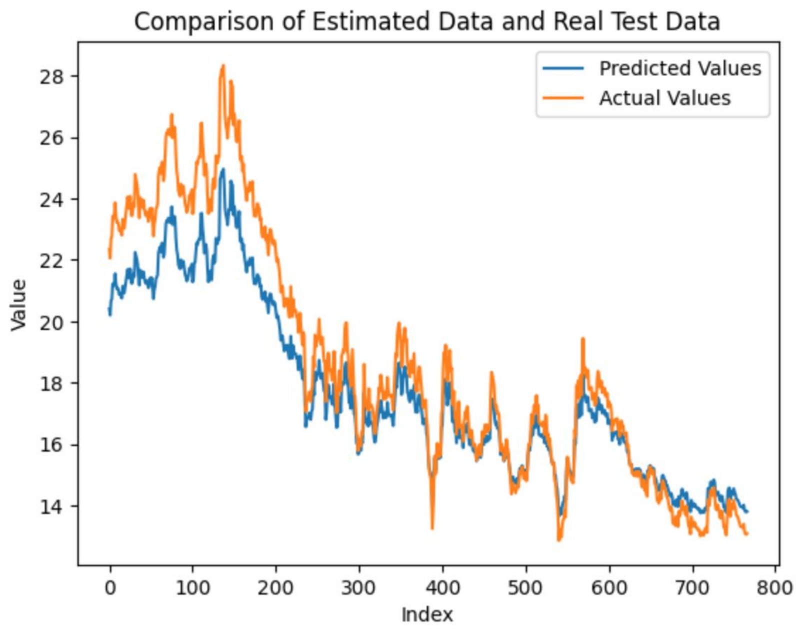
And the Root Mean Squared Error (RMSE) on the Test Data is 1.3685109077488955.



Vanke China Output:

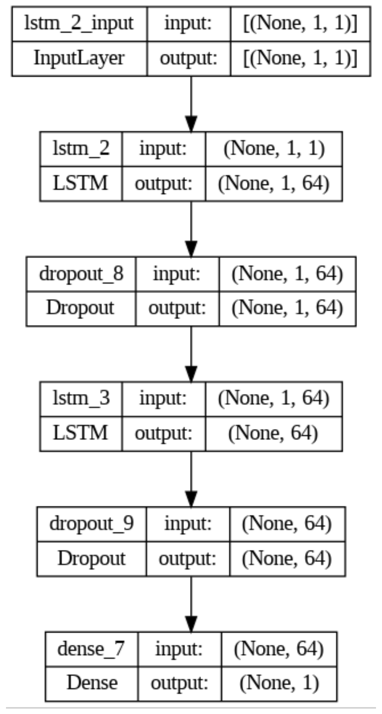

lstm_4_input | input: | [(None, 1, 1)]
InputLayer | output: | [(None, 1, 1)]
lstm_4 | input: | (None, 1, 1)
LSTM | output: | (None, 1, 50)
dropout_10 | input: | (None, 1, 50)
Dropout | output: | (None, 1, 50)
lstm_5 | input: | (None, 1, 50)
LSTM | output: | (None, 50)
dropout_11 | input: | (None, 50)
Dropout | output: | (None, 50)
dense_8 | input: | (None, 50)
Dense | output: | (None, 1)

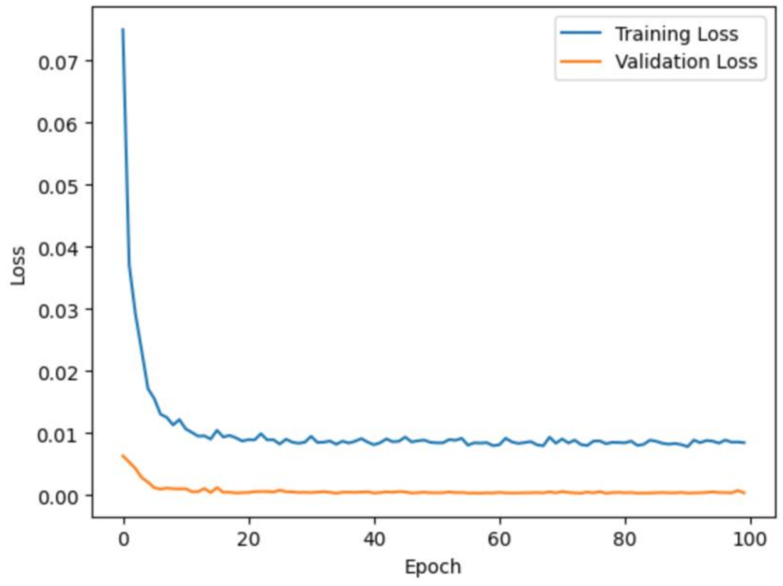




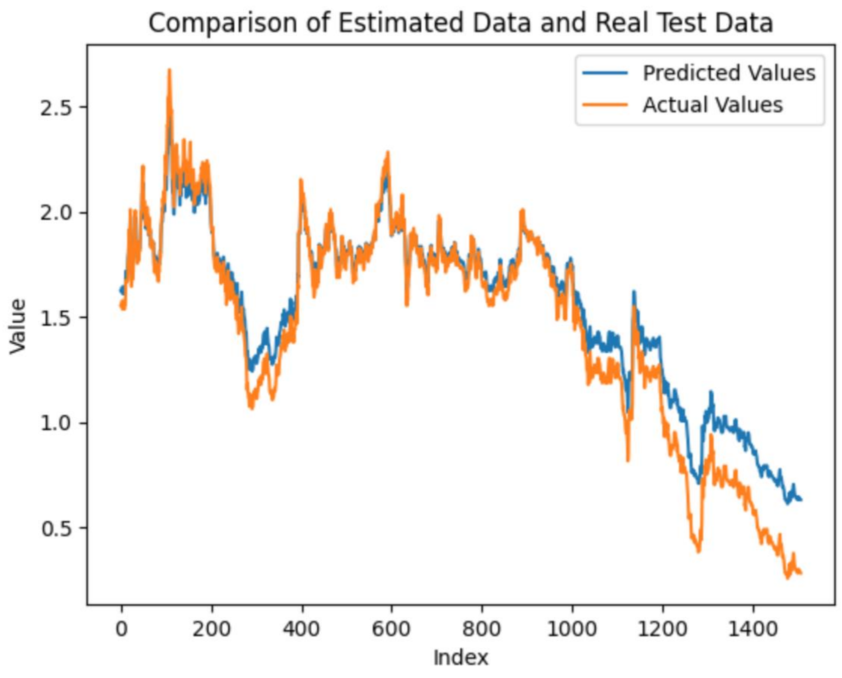
And the Root Mean Squared Error (RMSE) on the Testing Data is 1.3956633213405678.

Greenland China Equity:

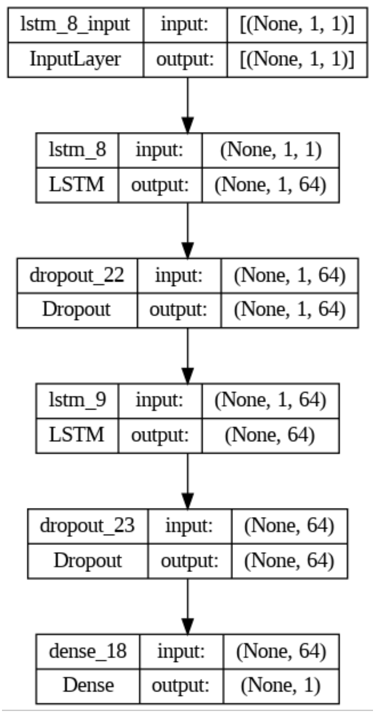

lstm_2_input | input: | [(None, 1, 1)]
InputLayer | output: | [(None, 1, 1)]
lstm_2 | input: | (None, 1, 1)
LSTM | output: | (None, 1, 64)
dropout_8 | input: | (None, 1, 64)
Dropout | output: | (None, 1, 64)
lstm_3 | input: | (None, 1, 64)
LSTM | output: | (None, 64)
dropout_9 | input: | (None, 64)
Dropout | output: | (None, 64)
dense_7 | input: | (None, 64)
Dense | output: | (None, 1)

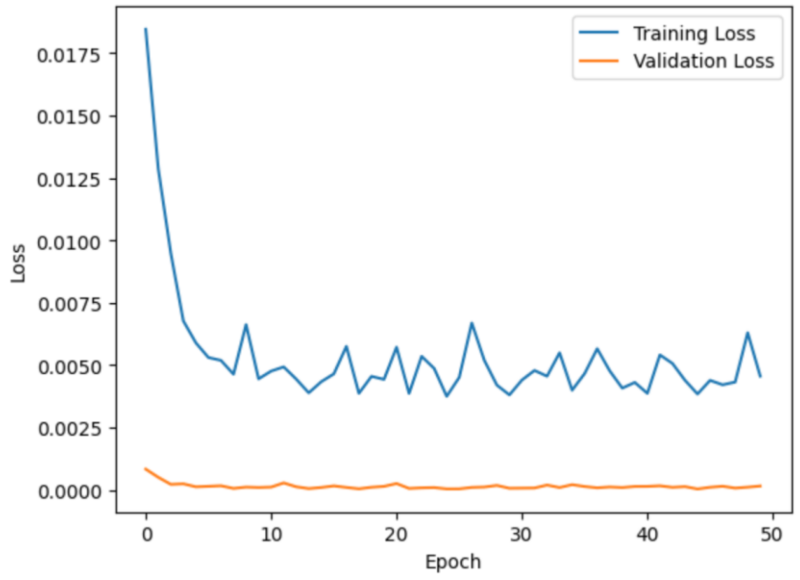


And the Root Mean Squared Error (RMSE) on the Test Data is 0.14599423510876286.

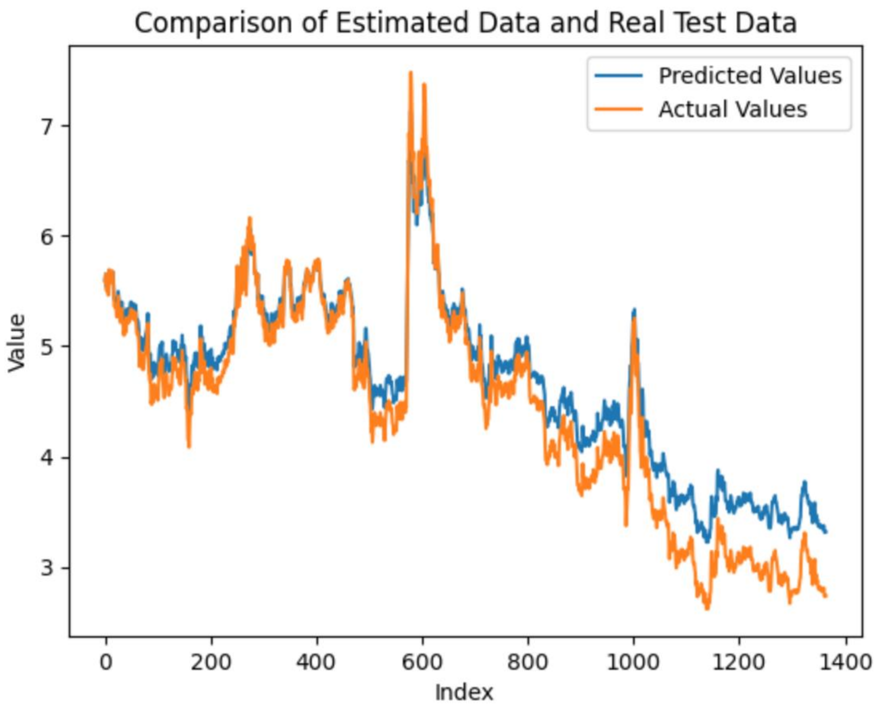


Greenland HK Equity:


lstm_8_input | input: | [(None, 1, 1)]
InputLayer | output: | [(None, 1, 1)]
lstm_8 | input: | (None, 1, 1)
LSTM | output: | (None, 1, 64)
dropout_22 | input: | (None, 1, 64)
Dropout | output: | (None, 1, 64)
lstm_9 | input: | (None, 1, 64)
LSTM | output: | (None, 64)
dropout_23 | input: | (None, 64)
Dropout | output: | (None, 64)
dense_18 | input: | (None, 64)
Dense | output: | (None, 1)



And the Root Mean Squared Error (RMSE) on the testing data is 0.31079234905232367 .

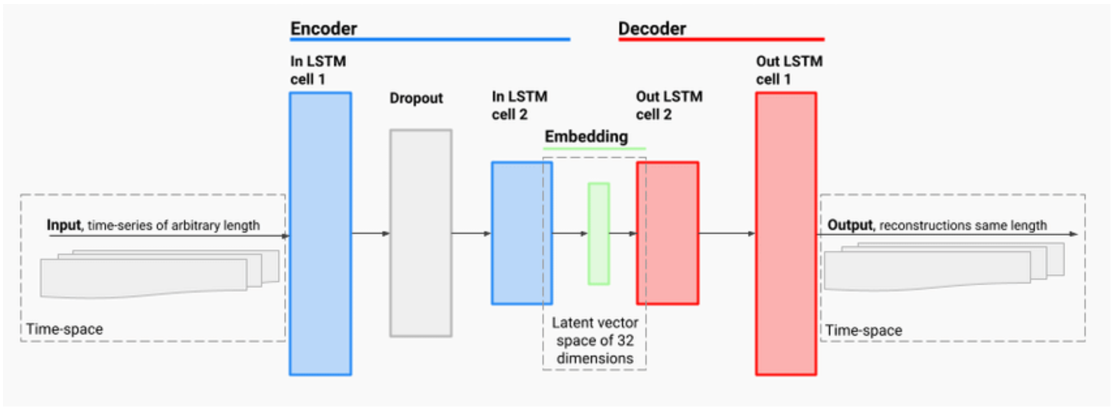


Based on the above, we can simplify the LSTM and the Dropout Model into the following diagram. And all of the four LSTM training models own small loss / errors from their loss plots we can see.

The final step we want to engage in is to apply the test set to see the difference between the predictions and actual data. So, we use the rest 20% test data as actual data and the discrepancies mean the difference that can't be covered in our prediction model. The x-axis of the plot represents different time points or data instances, while the y-axis represents the corresponding values.

However, when we put the predictions and actual values together, we could conclude from these four comparison plots. The prediction values look more moderate than the actual values. And all of the four prediction values reflect the downward tendency and the discrepancies between the predicted values and actual values except the Vanke China Equity, the rest three have the expansion trend for the future outlooking. It has been observed that there are significant differences between the predicted values and the actual values, particularly for the last 200 data points. These discrepancies indicate that our prediction model struggles to accurately capture the dynamics during those periods. The rest of the dataset reflects a moderate accuracy. We can see the prediction model looks more moderate than the actual model. These discrepancies can potentially be attributed to factors such as macroeconomic conditions, policy changes, or investor sentiment, which are difficult to quantify within our prediction model. For instance, the COVID-19 pandemic and events like Evergrande's bankruptcy might have influenced investor sentiment, leading to negative attitudes towards investing in the Hong Kong and China's Real Estate Market. Such external factors and investor sentiment can significantly impact stock prices, resulting in deviations between the predicted values and the actual data.

The overall performance of our prediction model appears to be moderate, as it demonstrates reasonable accuracy for the majority of the dataset. However, it is important to acknowledge the limitations of our model and the challenges associated with capturing the effects of complex macroeconomic events or investor sentiment accurately. Just like the Vanke China, why have the discrepancies between the actual values and the predicted values been shortened? Owing to the recent news, it has shown that Vanke China Holdings Company's liquidity risk has decreased and gained another expansion term for their debts for the local banks, beneficial for the investors to rebuild their confidence and interest in this stock price prediction. Although the LSTM can capture the non-linearity model, for such macroeconomic events or investors' sentiment or local government's support, these indicators still can't be added into the models. So we can consider it as the shortages whether for the LSTM models or traditional time series models.


Encoder
Decoder
In LSTM cell 1
Dropout
In LSTM cell 2
Embedding
Out LSTM cell 2
Out LSTM cell 1
Input, time-series of arbitrary length
Time-space
Latent vector space of 32 dimensions
Output, reconstructions same length
Time-space

Chapter 3: Conclusion

By applying traditional time series models and recurrent neural networks (RNNs) to real stock price prediction, it becomes apparent that the assumptions made by traditional time series models may not adequately capture the complex dynamics and non-linear relationships present in stock prices. Traditional time series models assume linearity, constant variance, normality, and efficient market behavior, which may not hold in practice. Stock price movements are influenced by a multitude of factors, including sudden macroeconomic events, unpredictable disasters, and investor sentiment. These features cannot be easily quantified or incorporated into linear models or treated as biases. The assumption of linearity in traditional time series models limits their ability to capture the inherent non-linearities and complex relationships observed in stock prices.

Furthermore, the assumption of stationarity, which serves as the foundation for applying traditional time series models, can pose challenges when dealing with non-stationary data. Techniques such as log transformation and differencing may overlook important features present in the original data. Traditional models also assume constant variance (homoscedasticity), leading to inadequate modeling of volatility clustering, where periods of high volatility are followed by periods of low volatility. This can result in underestimation of prediction intervals. Stock price returns often exhibit heavy-tailed distributions, with occasional extreme values or outliers. However, traditional models assume normality or relatively light-tailed distributions, which may not effectively capture the fat tails and extreme events commonly observed in financial data.

In contrast, recurrent neural networks, such as the Long Short-Term Memory (LSTM) method, have the advantage of capturing non-linear relationships and incorporating long-term dependencies from the data. However, even though LSTM models can capture a wide range of features, their predictions still tend to be more moderate than the actual values. This is because local financial market dynamics, local benchmarks, and investor sentiment are often difficult to incorporate into neural networks.

In practice, it is crucial to consider both quantitative indicators and qualitative indicators together when predicting stock prices. Combining the strengths of traditional time series models, machine learning techniques, and domain knowledge can lead to more robust and accurate predictions. By incorporating quantitative indicators from time series analysis and qualitative indicators such as local market conditions and investor sentiment, a more comprehensive understanding of the complex dynamics of stock prices can be gained.

Chapter 4: References and Appendices

References:

Tsay, R. S. (2016). *Analysis of Financial Time Series Chapter 1 – Chapter 3*. Wiley India.

Zhang et al. (2017) proposed an LSTM-based model that incorporated technical indicators and news sentiment to predict stock prices

Fischer et al. (2018) utilized an LSTM architecture to predict stock returns and outperformed traditional linear models

Chan, W., & Fong, W. M. (2016). Forecasting stock indices using ARIMA-SVR hybrid model. Expert Systems with Applications, 56, 182-189

Zhang, Y., Li, X., & Wei, Y. (2015). Forecasting stock indices with ARIMA-EGARCH hybrid model. International Journal of Economics, Commerce and Management, 3(8), 63-74

Wong, W. K., & Li, W. K. (2000). On a class of tests for normality: with applications to Hong Kong stock prices. Journal of Applied Econometrics, 15(6), 705-722

Code Template: Time Series Forecasting using variants of Long Short-Term Memories (LSTM) Recurrent Neural Networks

https://www.alibabacloud.com/blog/time-series-forecasting-using-variants-of-long-short-term-memory-lstm-recurrent-neural-networks_598410

Appendix:

Appendix 1: RStudio Code about the initial attempts for the four stocks prices forecasts

```
install.packages("forecast")
library(forecast)
## Greenland China Equity (600606 CH Equity)
install.packages("readxl")
library(readxl)
GC <-read_excel("Property Market Data.xlsx", sheet = 2)
names(GC)
GC$Time <- as.Date(GC$Time)
ts_data <- ts(GC$`Stock Prices`, frequency = 365)
plot(ts_data, main="Greenland China Equity Time Series Plot")
arima_model <- auto.arima(ts_data)
arima_model
hw_model <- HoltWinters(ts_data) # Fit a Holt_Winters Model
checkresiduals(arima_model)
forecast_result <- forecast(arima_model, h=365)
accuracy(forecast_result, main = "Forecasted Values")
plot(forecast_result) # Generate 365 forecasts using on ARIMA model
## Greenland Hong Kong Equity (0337 Hk Equity)
GH <-read_excel("Property Market Data.xlsx", sheet =1)
names(GH)
GH$date <- as.Date(GH$date)
ts_data2 <- ts(GH$`value`, frequency = 365)
plot(ts_data2, main="Greenland HK Equity Time Series Plot")
arima_model2 <- auto.arima(ts_data2)
arima_model2
hw_model2 <- HoltWinters(ts_data2) # Fit a Holt_Winters Model
hw_model2
checkresiduals(arima_model2)
forecast_result2 <- forecast(arima_model2, h=365)
accuracy(forecast_result2, main = "Forecasted Values")
plot(forecast_result2) # Generate 365 forecasts using on ARIMA model
ccf(ts_data,ts_data2, lag.max = 200)
## Vanke China Equity (00002 Equity)
VC <-read_excel("Property Market Data.xlsx", sheet = 3)
names(VC)
VC$Time <- as.Date(VC$Time)
ts_data3 <- ts(VC$`Stock Prices`, frequency = 365)
plot(ts_data3, main="Vanke China Equity Time Series Plot")
arima_model3 <- auto.arima(ts_data3)
arima_model3
hw_model3 <- HoltWinters(ts_data3) # Fit a Holt_Winters Model
hw_model3
checkresiduals(arima_model3)
```

```r
forecast_result3 <- forecast(arima_model3, h=365)
accuracy(forecast_result3, main = "Forecasted Values")
plot(forecast_result3) # Generate 365 forecasts using on ARIMA model
## Vanke Hong Kong Equity (2202 Hk Equity)
VH <-read_excel("Property Market Data.xlsx", sheet = 4)
names(VH)
VH$Time <- as.Date(VH$Time)
ts_data4 <- ts(VH$`Stock Prices`, frequency = 365)
plot(ts_data4, main="Vanke HK Equity Time Series Plot")
arima_model4 <- auto.arima(ts_data4)
arima_model4
hw_model4 <- HoltWinters(ts_data4) # Fit a Holt_Winters Model
hw_model4
checkresiduals(arima_model4)
forecast_result4 <- forecast(arima_model4, h=365)
accuracy(forecast_result4, main = "Forecasted Values")
plot(forecast_result4) # Generate 365 forecasts using on ARIMA model
```

Appendix 2: RStudio Code for the Daily Return of these four stocks

```r
library(forecast)
library(tseries)
library(readxl)
library(stats)
library(tsoutliers)
#Vanke HK
VH <-read_excel("FYP.xlsx", sheet = 4)
names(VH)
VH$Time <- as.Date(VH$Time)
ts_data4 <- ts(VH$`Return`, start = c(2014, 6, 25), end = c(2023, 6, 25), frequency = 365)
ts_data4updated<- na.omit(ts_data4[ts_data4 != 0])
plot(ts_data4, main="Vanke HK Equity Time Series Plot",xlab = "Year", ylab = "Return")
adf_test <- adf.test(ts_data4updated)
print(adf_test) #Stationary
acf_result <- acf(ts_data4updated)
plot(acf_result, main = "Vanke HK ACF")
pacf_result <- pacf(ts_data4updated)
plot(pacf_result, main = "Vanke HK PACF")
#Vanke China
VC <-read_excel("FYP.xlsx", sheet = 3)
names(VC)
VC$Time <- as.Date(VC$Time)
ts_data3 <- ts(VC$`Return`, start = c(2014, 6, 25), end = c(2023, 6, 25), frequency = 365)
ts_data3updated<- na.omit(ts_data3[ts_data3 != 0])
plot(ts_data3, main="Vanke China Equity Time Series Plot",xlab = "Year", ylab = "Return")
adf_test2 <- adf.test(ts_data3updated)
print(adf_test2) #Stationary
acf_result1 <- acf(ts_data3updated)
plot(acf_result1, main = "Vanke China ACF")
pacf_result1 <- pacf(ts_data3updated)
plot(pacf_result1, main = "Vanke China PACF")
cross_corr <- ccf(ts_data3updated,ts_data4updated)
plot(cross_corr, main = "Vanke Hong Kong & China")
lag <- cross_corr$lag[which.max(abs(cross_corr$acf))]
print(paste("The lag between the series is:", lag))
## lag = -24 means the ts_data3updated ahead of ts_data4updated about 24 trading days
#Greenland HK
```

```r
GH <-read_excel("FYP.xlsx", sheet = 2)
names(GH)
GH$Time <- as.Date(GH$Time)
ts_data2 <- ts(GH$`Return`, start = c(2006, 6, 25), end = c(2023, 6, 25), frequency = 365)
ts_data2updated<- na.omit(ts_data2[ts_data2 != 0])
plot(ts_data2, main="Greenland HK Equity Time Series Plot",xlab = "Year", ylab = "Return")
adf_test3 <- adf.test(ts_data2updated)
print(adf_test3) #Stationary
acf_result2 <- acf(ts_data2updated)
plot(acf_result2, main = "Greenland HK ACF")
pacf_result2 <- pacf(ts_data2updated)
plot(pacf_result2, main = "Greenland HK PACF")
#Greenland China
GC <-read_excel("FYP.xlsx", sheet = 1)
names(GC)
GC$Time <- as.Date(GC$Time)
ts_data <- ts(GC$`Return`, start = c(2006, 10, 9), end = c(2023, 6, 25), frequency = 365)
ts_dataupdated<- na.omit(ts_data[ts_data != 0])
plot(ts_data, main="Greenland China Equity Time Series Plot",xlab = "Year", ylab = "Return")
adf_test4 <- adf.test(ts_dataupdated)
print(adf_test4) #Stationary
acf_result3 <- acf(ts_dataupdated)
plot(acf_result3, main = "Greenland China ACF")
pacf_result3 <- pacf(ts_dataupdated)
plot(pacf_result3, main = "Greenland China PACF")
cross_corr1 <- ccf(ts_dataupdated,ts_data2updated)
cross_corr1
lag <- cross_corr1$lag[which.max(abs(cross_corr1$acf))]
print(paste("The lag between the series is:", lag))
## lag = -20 means the ts_dataupdated ahead of ts_data2updated about 20 trading days
```

Appendix 3: Google Colab using the RNN to predict the four individual stocks

```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from google.colab import files
uploaded = files.upload()
# Specify the filename of the uploaded Excel file
excel_filename = 'Greenland HK Equity.xlsx'

# Read the Excel file into a Pandas DataFrame
df = pd.read_excel(excel_filename)
# Check column names
print(df.columns)
# Normalize the values to the range [0, 1]
scaler = MinMaxScaler(feature_range=(0, 1))
df['Stock Prices'] = scaler.fit_transform(df['Stock Prices'].values.reshape(-1, 1))
# Split the data into training and test sets
train_size = int(len(df) * 0.64)
val_size = int(len(df) * 0.16)
test_size = len(df) - train_size - val_size
train_data = df[:train_size]['Stock Prices'].values
val_data = df[train_size:train_size+val_size]['Stock Prices'].values
```

```
test_data = df[train_size:]['Stock Prices'].values
# Reshape the input data
train_data = train_data.reshape(-1, 1, 1)
val_data = val_data.reshape(-1, 1, 1)
test_data = test_data.reshape(-1, 1, 1)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.utils import plot_model
# Define model architecture
model = Sequential()
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.9))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.9))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.9))

model = tf.keras.Sequential([
  tf.keras.layers.LSTM(units=64, return_sequences=True, input_shape=(1, 1)),
  tf.keras.layers.Dropout(0.9),
  tf.keras.layers.LSTM(units=64),
  tf.keras.layers.Dropout(0.9),
  tf.keras.layers.Dense(units=1)
])
# Compile the model
model.compile(optimizer='adam', loss='mse')
model.compile
# Train the model
history = model.fit(
  train_data[:-1], train_data[1:],
  epochs=50, batch_size=16, verbose=1,
  validation_data=(val_data[:-1], val_data[1:])
)
# Plot the loss history
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Draw the neural network diagram
plot_model(model, show_shapes=True, show_layer_names=True, to_file='model.png')
model.fit
#Print the model summary
model.summary()
# Perform prediction on the test data
```

```
predictions = model.predict(test_data)

# Convert the predicted values to their original scale using the scaler
predicted_values = scaler.inverse_transform(predictions.reshape(-1, 1))

# Convert the actual test data to their original scale using the scaler
actual_values = scaler.inverse_transform(test_data.reshape(-1, 1))

# Calculate accuracy (e.g., using mean squared error)
mse = np.mean((actual_values - predicted_values) ** 2)
rmse = np.sqrt(mse)

print(f"Root Mean Squared Error (RMSE) on Test Data: {rmse}")
import matplotlib.pyplot as plt

# Convert the predicted values and actual values to a 1D array
predicted_values = predicted_values.flatten()
actual_values = actual_values.flatten()

# Create a range of indices for the x-axis
indices = np.arange(len(predicted_values))

# Plot the predicted values and actual values
plt.plot(indices, predicted_values, label='Predicted Values')
plt.plot(indices, actual_values, label='Actual Values')

# Set plot labels and title
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Comparison of Estimated Data and Real Test Data')

# Add legend
plt.legend()

# Display the plot
plt.show()
```