The University of Southampton

Academic Year 2018/2019

Faculty of Social Sciences

Mathematical Sciences

MSc Dissertation

Comparing Housing Price Forecasting by Statistical and Machine Learning Approaches

Yujie Shen

A dissertation submitted in partial fulfilment of the MSc in Statistics

I am aware of the requirements of good academic practice and the potential penalties for any breaches. I confirm that this dissertation is all my own work.

Executive Summary

At this stage, machine learning (ML) is developing promptly as the mainstream method of solving artificial intelligence problems. The earliest machine learning algorithms date back to the early 20th century. In the previous 100 years, a great deal of excellent machine learning methods has been created. A time series is a series of data points indexed in the order of time. Time series forecasting is the utilisation of a model to forecast future values based on previously pragmatic values. In recent years, ML methods are gradually used in time series forecasting. However, there is intense controversy about the forecasting performance of the machine learning methods. Some studies aimed to compare the forecasting performance of ML methods and statistical methods (Fu et al., 2016; Makridakis et al., 2018). However, they only fitted models by univariate time series data.

In this thesis, we compare the forecasting performance of machine learning methods with that of statistical models which are also known as time series models. By fitting both univariate and multivariate data separately using seasonally adjusted monthly house price indexes (HPI) for nine Census Bureau divisions and the U.S., which is a multivariate data, and US, HPI is selected as univariate data, which was downloaded from the Federal Housing Finance Agency. Various data preprocessing will be applied and compared. We extend the previous researches by comparing the forecasting performance of ML and statistical methods for multivariate time series data. Besides, we hope to find out why researchers have diverse opinions on the forecasting performance of ML methods. We choose to build MLP, LSTM, SVR based on previous studies for ML methods while for statistical models, we choose to build ARIMA and VAR models. First, we build the ARIMA model by univariate data and normalised univariate data separately to obtain two 12- steps forecasting series and calculate their RMSE. Second, we utilise grid search to select the value of parameters of MLP, LSTM, and SVR which have minimum MSE and then train these MLP, LSTM, and SVR by univariante data, and normalised univariate data separately to obtain two 12-steps forecasting series and calculate the RMSE of the forecasting. Third, we build VAR model by nine Census Bureau divisions monthly HPIs and US HPI, ten-dimension multivariate data, and normalised multivariate data to obtain two 12-steps forecasting series and calculate the RMSE of the forecasting by order of time. Fourth, we train MLP, LSTM, and SVR in which the value of parameters are identical to the values that were selected in the second step, by raw, normalized, differenced

and joint differenced with normalised multivariate data separately to obtain two 12-steps forecasting series and calculate the RMSE of the forecasting by order of time. Finally, we evaluate the forecasting accuracy by comparing the RMSE of various methods with different data preprocessing.

For univariate forecasting, ARIMA offers the utmost robust forecasting performance because there is no significant difference in RMSE calculated by different preprocessed data. It implies that the data preprocessing does not affect the forecasting accuracy of ARIMA. Besides, there is no substantial decrease of forecasting accuracy with the increase of forecasting steps, followed by linear SVR and the MLP. LSTM indicates a reduced forecasting performance because its RMSE is much higher than the other three methods'. Besides, LSTM's forecasting accuracy drops fast as forecasting step increases.

For multivariate forecasting, VAR model also offers the most robust forecasting performance because there is no dramatic difference between RMSE calculated by different preprocessed data. This implies that data preprocessing does not affect the forecasting accuracy of VAR. Also, there is no significant decrease in forecasting accuracy with the increase of forecasting steps, followed by MLP and LSTM. However, SVR indicates a poor forecasting performance. First, this is because the RMSE of SVR is higher than that of other methods. Second, the RMSE of raw data and normalised data are much higher than RMSEs calculated by differenced data, and joint differenced and normalised data.

The basis of the argument for the ML method may be; first, some ML methods are not appropriate for the extended period forecasting. In other words, the forecasting accuracy decline with the increase of forecasting steps for some ML methods. Second, for some ML methods, particularly some neural network, the structure of the network has a significant impact on the forecasting performance. However, it would be challenging to figure out an appropriate structure for the network. Third, some ML models are hard to model selection. Fourth, a proper data preprocessing is vital for some ML methods.

The recommendations to improve the forecasting accuracy when fitting time-series data include, first when fitting time-series data, compared to ML methods, statistical methods are a better choice. Second, for time series analysis, MLP is the right choice in ML methods. Third, a complex structure is preferred when fitting time series data

with LSTM. Fourth, careful model selection and data preprocessing are required when fitting time series data with SVR. The recommendations for future work are first; we only compared VAR, MLP, LSTM, and SVR in the multivariate time series analysis, but volatility was not included in this thesis. In the future, research can compare more multivariate time series models with more ML methods in both forecasting performance and volatility. Second, more types of data preprocessing can be involved, along with the evaluation of computational cost in the future.

ACKNOWLEDGEMENT

First and foremost, I would like to show my deepest gratitude to my supervisor, Prof. Zudi Lu, a respectable, responsible and resourceful scholar, who has provided me with valuable guidance in every stage of the writing of this thesis. Without his enlightening instruction, impressive kindness and patience, I could not have completed my thesis. His keen and vigorous academic observation enlightens me not only in this thesis but also in my future study.

I would also like to thank all my teachers who have helped me to develop the fundamental and essential academic competence.

I would like to thank all my friends, especially my three lovely roommates, for their encouragement and support.

Finally, I must express my very profound gratitude to my parents and to my for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Author

Yujie Shen

Contents Page

Table of Figures

List of Tables

List of Figures

Summary

At this stage, machine learning (ML) is rapidly developing as the mainstream method of solving artificial intelligence problems. Recently, the use of ML methods has increased in time series forecasting. However, there is controversy regarding the forecasting performance of the ML methods. Some studies aimed to compare the forecasting performance of ML methods and statistical methods (Fu et al., 2016; Makridakis et al., 2018). However, they only compared the forecasting performance for univariate time series. In this thesis we compare the forecasting performance of machine learning methods with statistical models by fitting both univariate and multivariate data separately by using seasonally adjusted monthly house price indexes (HPI) for nine Census Bureau divisions and the U.S., which is a multivariate data and US, HPI is selected as univariate data. MLP, LSTM, SVR were chosen to represent ML methods while for statistical models, we choose to build ARIMA for univariate data and VAR models for multivariate data. A comparison of three different types of data preprocessing with RMSE of 12-steps forecasting was performed.

We found that, for univariate forecasting, ARIMA offers the utmost robust forecasting performance, followed by linear SVR and MLP, but LSTM's forecasting perform poorly. For multivariate forecasting, VAR model offers the strongest forecasting performance, followed by MLP and LSTM. However, SVR indicates poor forecasting performance. MLP's forecasting accuracy reduces with the increase of forecasting steps. LSTM's structure influences their forecasting accuracy. Rbf SVR does not fit the data well and data preprocessing impacts rbf SVR's forecasting performance dramatically.

In conclusion, statistical models have strong forecasting performance. Some ML methods are not useful for an extended period forecasting. For some ML methods, the structure of the network has a significant effect on forecasting performance. Proper data preprocessing and model selection are essential for some ML methods

1. Introduction:

Machine learning (ML) is seen as a subset of artificial intelligence. As the mainstream method to decipher artificial intelligence problems at this stage, machine learning is rapidly developing. The earliest ML algorithms dated back to the early 20th century about 100 years ago. In the last 100 years, several exceptional ML methods have been created. A time series is a series of data points indexed in the order of time. Time series forecasting refers to the use of a model to envisage future values based on previously observed values.

In recent years, ML methods are gradually used in time series forecasting. However, there is intense controversy about the forecasting performance of machine learning methods. Some studies suggest that the ML methods have excellent forecasting performance on time series data (Ahmed et al., 2010). However, some studies also suggest that they are not optimistic about the forecasting performance of machine learning methods (Makridakis et al., 2018). Some studies compared the forecasting performance of ML methods and that of statistical methods (Fu et al., 2016; Makridakis et al., 2018). However, they only compared the forecasting performance of the methods that fitted by univariate time series data. Therefore, in this thesis, we anticipate extending the previous researches by comparing the forecasting performance of ML and statistical methods for multivariate time series data. Besides, we hope to find out why researchers have diverse opinions on the forecasting performance of ML methods.

We compare the forecasting performance of machine learning methods with that of statistical models that are also known as time series models. By fitting both univariate and multivariate data separately using seasonally adjusted monthly house price indexes (HPI) for nine Census Bureau divisions and the U.S., which is a multivariate data and U.S., HPI is selected as univariate data downloaded from the Federal Housing Finance Agency, ("House Price Index Datasets | Federal Housing Finance Agency", 2014). Various data preprocessing will be applied and compared. To be more exact, Firstly, we build ARIMA and VAR for the time series models by using R (See Appendix 1 for the R code), we build SVR, LSTM and MLP for the ML methods by using python (See Appendix 2 for the Python code). We choose the models based on the literature review. Secondly, we calculate the 12-steps forecasts of different models with the differente types of data preprocessing. Finally, we measure the forecasting performance by comparing the root mean square error of each

forecastings.

This thesis is divided into seven chapters which are Introduction, Background, Literature review, Methodology, Results, Discussion, and Conclusion. In the Background section, we introduce the Federal Housing Finance Agency and house price index separatelydistinctly. The Literature review section is comprised of previous studies that include the current knowledge, including substantive findings, as well as theoretical and methodological contributions. It is divided into four parts. The first part presents the pieces of literature about classical time series models. These pieces of literature are used in this thesis to select the statistical models. The second part lists the pieces of literature that apply ML methods to analyse time series. These pieces of literature are used in this thesis to select ML methods. The third part corresponds to the literature that focuses on comparing classical time series models and ML methods, which designates the controversy about the forecasting performance of the machine learning methods by comparing it with statistical models. The final part presents the gaps that are found in this literature review. This thesis aims to solve these gaps. The Methodology section presents the models that are selected in the literature review section and outlines the reasons why these models are chosen first. Subsequent, the data and data preprocessing used in this thesis are presented. The last two parts correspond to some basic knowledge of statistical models and machine learning models distinctly.

The Results section commences with describing the data, followed by the presentation of the process of building the time series models and the machine learning models that are chosen in the Literature review section. Finally, we compare the forecasting performance between time series models and machine learning methods by different data preprocessing. The Discussion section concludes what we found in the Results section firstly. Secondly, we point out the results that are consistent with previous researches. The third part depicts the results that contradict with previous researches, and the final part proposes some possible reasons why there is a controversy on the forecasting accuracy of ML methods. In the conclusion and recommendation section, we summarise the results and discussion by previews chapters first, and then we offer some recommendation of how to achieve higher forecasting accuracy based on the conclusion and discussion sections in this thesis. Finally, we present some recommendation for further areas of research in the future.

There are three key contributions of this dissertation. First, we extend the previous researches by comparing the forecasting performance of ML methods and statistical

methods for multivariate time series data. Second, we provide possible reasons why researchers have diverse opinions on the forecasting performance of ML methods. Third, we offer various suggestions for improving the forecasting accuracy when forecasting time series data.

2. Background

2.1 Federal Housing Finance Agency (FHFA)

The Federal Housing Finance Agency (FHFA) was established by the Housing and Economic Recovery Act of 2008, and holds responsibility for the effective supervision, regulation, and housing mission oversight of Fannie Mae, Freddie Mac (the Enterprises) and the Federal Home Loan Bank System, which includes the Federal Home Loan Banks and the Office of Finance. FHFA is an independent federal agency, created to succeed the regulatory agency of the Federal Housing Finance Board, the office of Federal Housing Enterprise Oversight, and the U.S. Department of Housing and Urban Development government, ("Home | Federal Housing Finance Agency", 2008).

2.2 A house price index (HPI)

The Federal Housing Finance Agency (FHFA) house price indexes (HPIs) which extends back to January 1975, measure changes in single-family house prices, based on data that covers all 50 states, and spans over 400 USA cities. The HPI used in this thesis includes house price figures from the nine Census Bureau divisions for the District of Columbia, along with the 50 states, and for Metropolitan Statistical Areas (MSAs) and Divisions. HPIs applies a fully transparent methodology based on a weighted, repeat-sales statistical technique, to analyze data transactions from Fannie Mae and Freddie Mac. Therefore, HPIs measures the average price changes occuring in repeat sales, ("Home | Federal Housing Finance Agency", 2008). HPIs are a broad measure of movement involving single-family house prices, that commences from a certain specific start date, in the United States. HPI serves as a timely, accurate indicator of house price trends at various geographic levels. It also provides housing economists with an improved analytics tool for estimating changes in the rates of mortgage defaults, prepayments, and housing affordability in specific geographic areas, ("eAppraise | residential real estate appraisal valuation", 2000).

3. Literature review

This literature review is divided into four parts. The first part corresponds to pieces of literature that have utilized classical statistical models. The second part corresponds to the pieces of literature which have applied ML methods for time series forecasting. The third part corresponds to pieces of literature which focus on comparisons regarding forecasting performance, between classical statistical models and ML method, for time series problems. The final part presents the gaps which have been identified in this literature review.

3.1 Time series

A lot of researchers chose classical statistical methods to fit time series data. Volkan and Sertac (2007) used the ARIMA seasonal ARIMA (SARIMA) models to estimate the future, primary energy demand of Turkey from 2005 to 2020. Eddie and Shen (2006) fitted vector autoregressive and vector error-correction models to investigate the housing price bubble in Beijing and Shanghai. Rangan and Stephen (2010) also examined the time-series relationship between housing prices in eight Southern California, metropolitan statistical areas, using various vector autoregressive and vector error-correction models.

A number of studies focused on comparing the forecasting accuracy between different statistical models. Gordon and Michael (2003) compared the forecasting performance of three univariate time series models: ARIMA, GARCH, and regime-switching. They identified that simple ARIMA models generally perform better for out-of-sample forecasting. Stergiou et al. (1997) evaluated seven forecasting techniques; seasonal time-varying regression, multiple regression models, Winter's three-parameter exponential smoothing, ARIMA, harmonic regression, dynamic regression, and VAR. The results revealed that ARIMA performed best in the univariate time series model, while the multivariate DREG time series model performed best in the multivariate time series model.

3.2 Machine learning

Many researchers had chosen machine learning methods to fit both univariate and multivariate time series data. For the univariate time series data, Mu et al. (2014)

compared the prediction effects of SVM, least squares support vector machine (LSSVM), and partial least squares (PLS) methods, by applying the Boston suburb houses values. They found that SVR and LSSVM methods were superior to PLS for dealing with the problem of nonlinearity. The prediction effect of SVM was found to be superior to LSSVM. In addition, Ahmed et al. (2010) evaluated the performance of eight machine learning methods; MLP, SVR, Bayesian Neural Network (BNN), Radial Basis Functions (RBF), Generalized Regression Neural Networks (GRNN), K-Nearest Neighbor regression (KNN), CART regression trees (CART), and Gaussian Processes (GP). Of these, the best two methods were identified to be MLP and the GP. For multivariate data, Miiller et al. (1997) depicted that SVR showed better prediction performance of time series when compared to RBF networks.

3.3 Comparison

Fu et al. (2016) used Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) neural network (NN) methods to predict short-term traffic flow, and the experiments demonstrated that LSTM and GRU performed better than the usage of the ARIMA model. However, Makridakis et al. (2018) evaluated the performance of eight classical statistical methods and ten machine learning methods. The eight classical statistical methods were Naive 2, Simple Exponential Smoothing, Holt, Damped exponential smoothing, Average of SES, Holt, and Damped, Theta method, ARIMA, and ETS. The ten machine learning methods evaluated were Multi-Layer Perceptron (MLP), Bayesian Neural Network (BNN), Radial Basis Functions (RBF), Generalized Regression Neural Networks (GRNN), also called kernel regression, K-Nearest Neighbor regression (KNN), CART regression trees (CART), Support Vector Regression (SVR), Gaussian Processes (GP), Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM). The results indicated that ARIMA performed best in forecasting and was followed by BNN, while LSTM depicted poor performance. To be more precise, Makridakis et al. (2018) compared the forecasting performances of one-step-ahead forecasts and multi-step-ahead forecasting. Three approaches - Iterative forecasting, Direct forecasting, and Multi-neural network forecasting for multi-step-ahead forecasting were evaluated for machine learning methods. Iterative forecasting performed best amongst these three approaches. The results showed that, for statistics method, ETS and ARIMA out-performed machine learning methods for one-step-ahead forecasting. Theta and ARIMA out-performed machine learning methods for multi-step forecasting.

3.4 Conclusion of Literature Review

In conclusion, this part of the literature review offers at least two threads from which theory and insights might be woven. Here, five studies have been involved in classical statistical methods and three studies have been involved in machine learning methods. Volkan and Sertac (2007), Eddie and Shen (2006), and Rangan and Stephen (2010) provided case studies of classical time series models for both univariate data and multivariate data. Gordon and Michael (2003) and Stergiou et al. (1997) compared the forecasting performance of classical statistical models for univariate data and multivariate data, respectively. The ARIMA model was identified as the best performer in univariate data, and the multivariate DREG model was found to have the best performance in multivariate data. Mu et al. (2014), Ahmed et al. (2010) and Miiller et al. (1997) compared the forecasting performance of machine learning for univariate data and multivariate data, respectively. The MLP model was found to have excellent performance in univariate data, while the SVR model showed excellent performance in both univariate and multivariate data. Fu et al. (2016) and Makridakis et al. (2018) compared the forecasting performance between classical time series methods and machine learning methods by utilizing the univariate time series data. Fu et al. (2016) found that machine learning methods performed better than statistical time series methods, while Makridakis et al. (2018) argued that machine learning methods depicted the worst performance, when compared to statistical methods in time series forecasting.

The review of the literature revealed two gaps which have been addressed by this research. First, in recent years, machine learning methods have been increasingly applied in time series forecasting. However, the evaluation of machine learning's forecast performance is controversial. While certain studies suggest that the LM methods have excellent predictive performance on time series data (Ahmed et al., 2010; Fu et al., 2016), some studies do not have an optimistic view regarding the forecasting performance of machine learning methods (Makridakis et al., 2018). Second, a few studies intended to compare the forecasting performance of ML and statistical methods (Fu et al., 2016; Makridakis et al., 2018). However, only the forecasting performance for univariate time series data was compared. Furthermore, there is a lack of studies provided information regarding the comparison of the multivariate time series forecasting accuracy between statistical methods and machine learning methods. Therefore, this thesis has assumed two goals. First, it aims to compare the forecasting performance of statistical methods and machine

learning methods for both univariate time series data and multivariate data. Second, it focuses on identifying the reason behind the different opinions held by researchers on the forecasting performance of ML methods.

4. Methodology

4.1 Choosing Models

According to the reviewed results of the literature review for the machine learning methods, MLP model was found to have excellent performance in univariate data, and SVR model was found to perform well in both univariate and multivariate data. However, there is a controversy in LSTM's forecasting performance. For statistical methods, the ARIMA model depicts good forecasting accuracy in the univariate time series data. VAR model is primarily utilized in multivariate time series. Therefore we compare ARIMA with MLP, SVR, and LSTM on the univariate dataset and compare VECM-VAR with MLP, SVR, and LSTM on the multivariate dataset. An Iterative forecasting approach is used in this thesis. It implies that we utilize in-sample data to obtain one-step-ahead forecasts. The first value is used to predict the second forecasting. Moreover, we use the second forecast value to estimate the third prediction value, and so on, until predictions for all 12 horizons are established. It means that we obtain 12 forecasts using the first n-12 data points only. Zhang and Qi (2005) claimed that without appropriate preprocessing, Machine learning methods might become unstable, thus yielding suboptimal results. To find out the impact of data preprocessing on the predicted performance, we use different preprocessing to obtain data with diverse aspects, and then utilize these data to fit the model distinctly.

4.2 Data

This thesis compared the forecasting accuracy of statistical methods and ML methods of various types of data preprocessing. The data used in this thesis is seasonally adjusted monthly house price indexes (HPI) for nine Census Bureau divisions and the U.S., which is a multivariate data and U.S.; HPI is designated as univariate data, downloaded from the Federal Housing Finance Agency, ("House Price Index Datasets | Federal Housing Finance Agency", 2014). Different data preprocessing will be applied and compared. The data period is from February 1991 to May 2019 for 341 observations. However, this data has a limitation, that is: the data we use is

not-seasonised, and therefore we cannot analyze the effect of the season on the forecasting performance of the model. RMSE in this thesis is used to measure the forecasting accuracy. Three types of data preprocessing are utilized in this thesis, which includes normalization, difference, and the joint of normalization data preprocessing.

Root- Mean Square error (RMSE):

$$RMSE = \sqrt{MSE(\hat{Y})} = \sqrt{E((\hat{Y} - Y)^2)} = \sqrt{\frac{\sum_{t=1}^{T} (\hat{y}_t - y_t)^2}{T}} \quad , \tag{1}$$

where t is the forecasting horizon, y_t are the actual observations and $\hat{y}_t$ the forecasts produced by the model at point t.

Data preprocessing (Data Transformation):

Zhang and Qi (2005) claimed that ML methods might become unstable and yield suboptimal results without appropriate preprocessing. Preprocessing can be applied in three forms which are seasonal adjustments, log or power transformations, and removing the trend. In this thesis, the data used is not-seasonized. Therefore we do not deliberate seasoned adjustment here. We use the first difference to detrend and normalitye data to compare data from various places more easily. The following data preprocessing are used:

Original data (raw data): No preprocessing is applied.

Normalized data:

$$y_{new} = \frac{y - y_{\min}}{y_{\max} - y_{\min}} \quad , \tag{2}$$

Differenced data:

$$\Delta y_t = y_{t+1} - y_t \tag{3}$$

Normalised+differenced data: Combination of the above two transformation

Walk-Forward Validation:

Walk-forward validation is also called Rolling Window Analysis. The data is divided into a train set and a test set. Walk-forward validation does optimization on a training set; test on a period after the set and then rolls it all forward and replicate the process. We have multiple out-of-sample periods and look at these results combined ("Jon Baer: Archive", 2013). The process is as indicated in *Figure 1*.

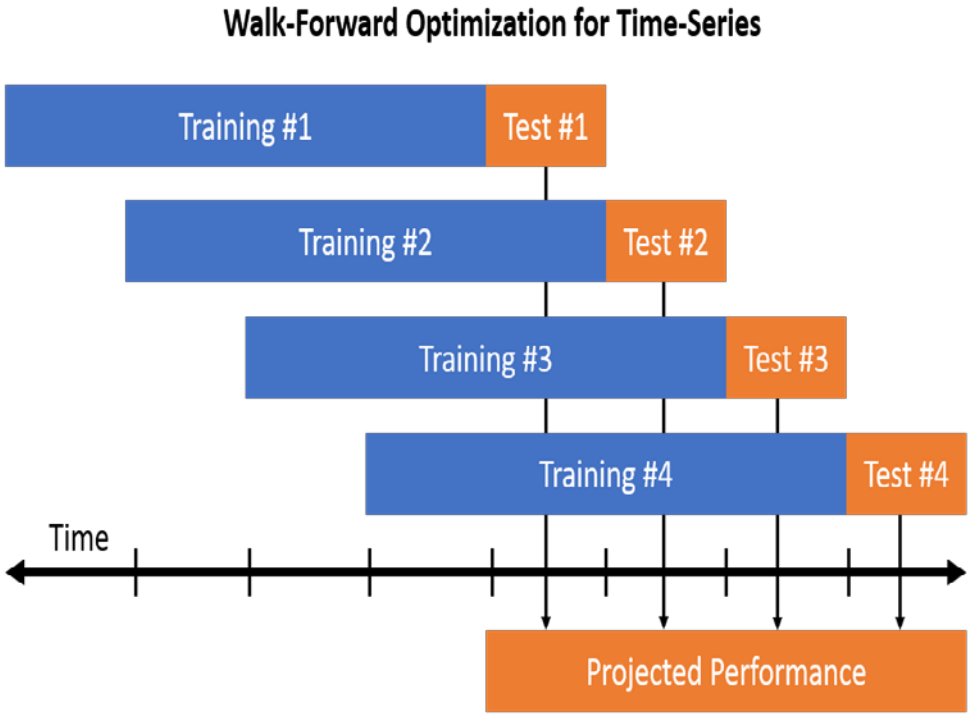

Figure 1 Walk-Forward Validation

In this thesis, Walk-Forward Validation is used to select the parameters with the minimum loss score in machine learning. The process is:

1. The data is divided into ten parts.

2. The first part is used to train a model.

3. The model predicts the next time step.

4. The prediction is evaluated against the second part.

5. The window is expanded to include the second part, and the process is repeated.

4.3 Introduction to Statistical Models

4.3.1 Model for Univariate

A univariate time series consists of value of a variable recorded over a long period of time. A time series $\{y_t : t = 1,2,...n\}$ is a set of observations of random variables $\{Y_t : t = 1,2,...n\}$. $Y(t)$ and $y(t)$ are continuous time series, while Y_t and y_t are discrete time series. The trend of the time series is $\mu(t) = E[Y(t)]$

White noise:

We have a sequence of uncorrelated random variables $\{X_t\}$ such that $E(\mathrm{X}_t) = 0$, $E(\mathrm{X}_t^2) = \sigma^2$ but no necessarily identically distributed. For white noise sequence, $\gamma(h) = 0$, for $h \neq 0$, $\gamma(0) = \sigma^2$. Written as $\{X_t\} \sim WN(0, \sigma^2)$

Correlogram:

Auto-covariance function:

$$\gamma(h) = \gamma(t+h, t) = E[\{Y(t+h) - \mu(t+h)\}\{Y(t) - \mu(t)\}], \tag{4}$$

Autocorrelation function (acf):

$$\rho(h) = \gamma(h)/\gamma(0), \tag{5}$$

since, $\gamma(0) = Var(Y(t+h)) = Var(Y(t))$, $\rho(h) = \rho(-h)$; $-1 \le \rho(h) \le 1$; If $y(t)$ and $y(t+h)$ are independent, then, $\rho(h) = 0$. For the discrete data, $\{Y_t : t = 1,2,...n\}$ is a stationary sequence, h is an integer. We often write Auto-covariance function and Auto-correlation function of $\{Y_t : t = 1,2,...n\}$ as γ_h, ρ_h.

The k-th sample Auto-covariance function:

$$g_k = \frac{1}{n}\sum_{t=k+1}^{n}(y_t - \overline{y})(y_{t-k} - \overline{y}), \tag{6}$$

where $\overline{y}$ is mean of $y_t, t = 1,2,...n$,

The k-th sample Autocorrelation function (sample acf):

$$r_k = g_k / g_0, \tag{7}$$

where, g_0 is the sample variance.

Partial Autocorrelation function (Pacf):

$$a(h) = \phi_{h,h}, \tag{8}$$

where $h \geq 1, a(0) = 0$, $\phi_{h,h}$ is the last component of $\phi_h = \Gamma_h^{-1}\gamma_h(-1)$ with $\hat{\Gamma}_h = [\gamma(\mathrm{i}-\mathrm{j})]_{i,j=1}^{h}$ and $\gamma_h(1) = (\gamma(1), \gamma(2), ...\gamma(h))'$, $\{X_t\}$ is stationary process.

Sample Partial Autocorrelation function (sample Pacf):

$$\hat{a}(h) = \phi_{h,h}, \tag{9}$$

where $h \geq 1$, $\hat{a}(0) = 0$, $\hat{\phi}_{h,h}$ is the last component of $\hat{\phi}_h = \Gamma_h^{-1}\hat{\gamma}_h(-1)$, $\hat{\phi}_h = \hat{\Gamma}_h^{-1}\hat{\gamma}_h(-1)$ with $\hat{\Gamma}_h = [\hat{\gamma}(\mathrm{i}-\mathrm{j})]_{i,j=1}^{h}$ and $\hat{\gamma}_h(1) = (\hat{\gamma}(1), \hat{\gamma}(2), ...\hat{\gamma}(h))'$.

Stationarity:

Stationarity implies that the probability of structure does not change over time. In practical, Second-order stationarity (weakly stationary) is often used.

Weakly stationary:

$$\mu(t) = t, \gamma(t,s) = \gamma(|t-s|). \tag{10}$$

Augmented Dickey–Fuller Test (ADF Test):

ADF Test (Li and McLeod, 1981) is the most commonly used test method for detecting the stationary of sequences. There are three main versions of the ADF Test, see equation(11). The test starts with model (c), followed by model (b), and finally model (a)

$$\Delta X_t = \delta X_{t-1} + \sum_{i=1}^{m} \beta_i \Delta X_{t-i} + \varepsilon_t \qquad \text{（a）}$$

$$\Delta X_t = a + \delta X_{t-1} + \sum_{i=1}^{m} \beta_i \Delta X_{t-i} + \varepsilon_t \qquad \text{（b）}, \tag{11}$$

$$\Delta X_t = a + \beta t + \delta X_{t-1} + \sum_{i=1}^{m} \beta_i \Delta X_{t-i} + \varepsilon_t \qquad \text{（c）}$$

where α is a constant,β is the coefficient on a time trend and m is the lag order of the autoregressive process.

$$H_0 : \delta = 0; \quad H_1: \quad \delta < 0$$

There is a unit root in the series if null hypothesis is accepted, which means the series is not stationary. If the null hypothesis is rejected, then the series is stationary and the

test is stopped. Otherwise, continue the test until the model a is verified.

$$H_0: \ \beta = 0, \ \ H_1: \ \beta \neq 0.$$

There is no trend in the series if the null hypothesis is accepted.

$$H_0: \ \alpha = 0, \ H_1: \ \alpha \neq 0$$

If the null hypothesis is accepted, then there is no drift.

Differencing:

First Difference:

$$\Delta y_t = y_t - y_{t-1}, \tag{12}$$

Second-order difference is:

$$\Delta^2 y_t = \Delta y_t - \Delta y_{t-1} = y_t - 2y_{t-1} + y_{t-2}. \tag{13}$$

Differencing is a method to remove the trend. Seasonal effects can also be removed by the use of appropriate differencing. For example, a monthly data, defining $B^k y_t = y_{t-k}$ to be backward shift operator of order k.

$$\Delta_{12} y_t = y_t - y_{t-12} = y_t - B^{12} y_t. \tag{14}$$

Portmanteau Test:

$$H_0 : Q_m = n(\mathrm{n}-2)\sum_{k=1}^{m} (\mathrm{n}-\mathrm{k})^{-1} \gamma_k^2 \sim \chi_m^2 . \tag{15}$$

The large value of Q_m suggests that sample Autocorrelations in data are too large to be sample from a white noise sequence. The sensitivity of Q_m to various types of departure from white noise depends on the choice of m. Portmanteau Test is used to test the white noise of the series. Ljung and Box (1978) proposed Ljung–Box test, which is a well-known version of a portmanteau test.

Diagnostic:

Akaike information criterion (AIC):

AIC is founded in information theory. AIC estimates the relative amount of information

lost by a given model, the less information a model loses, the higher the quality of that model.

Checking Residuals:

The residuals should have the appearance as white noise.

The correlogram of residuals should look like a white noise.

The null hypothesis of Portmanteau Test of residuals should be accepted.

Moving average: $Y_t \sim MA(q)$

$$Y_t = (1 + \sum_{j=1}^{q} \beta_j B^j) Z_t, \tag{16}$$

Autoregressive: $Y_t \sim AR(p)$

$$(1 - \sum_{i=1}^{p} \alpha_i B^i) Y_t = Z_t, \tag{17}$$

Autoregressive Moving average: $Y_t \sim ARMA(p,q)$

$$\phi(B) Y_t = \theta(B) Z_t, \tag{18}$$

Autoregressive Integrated Moving average: $Y_t \sim ARIMA(p,d,q)$

$$\phi(B)(1-B)^d Y_t = \theta(B) Z_t, \tag{19}$$

Seasonal ARIMA : $Y_t \sim SeasonalARIMA(p,d,q) \times (\mathrm{P},\mathrm{D},\mathrm{Q})_S$

$$\Phi_P(B^S)\phi(B)\nabla_S^D \nabla^d Y_t = \delta + \Theta_Q(B^S)\theta(B) Z_t, \tag{20}$$

where $\{Z_t\} \sim WN(0,\sigma^2)$, S denote the period of the seasonal effects (12 for monthly data, 4 for quarterly data) would seem to be appropriate. d and D are non-negative integers. B is backward shift operator, $B^k y_t = y_{t-k}$. The ordinary autoregressive and moving average components are represented by polynomial $\phi(B) = (1 - \sum_{i=1}^{p} \alpha_i B^i)$ and $\theta(B) = (1 + \sum_{j=1}^{q} \beta_j B^j)$ of orders p and q, respectively. The seasonal autoregressive and moving average components are represented by $\Phi_P(\mathrm{B}) = (1 - \sum_{i=1}^{P} \Phi_i B^i)$, $\Theta_Q(\mathrm{B}) = 1 + \sum_{j=1}^{Q} \Theta_j B^j$ of orders P and Q respectively. And the ordinary and seasonal difference components by $\nabla^d = (1-B)^d$ and

$$\nabla_S^D = (1 - B^S)^D$$

In this thesis, we build the Seasonal ARIMA model by following steps.

1. Checking the stationarity of the USA HPI by time series plot and correlogram plots (ACF, PACF). Differencing the series until it is stationary.
2. Fitting the ARIMA model with the minimum AIC.
3. Diagnosing the model by checking the white noise of the model residuals.
4. Forecasting by this model.

4.3.2 Model for multivariate

Considering a k dimensional time series $\mathbf{r}_t = (r_{1t}, r_{2t}, ...r_{kt})'$, $\mathbf{r}_t$ is weakly stationary if its first and second moments are time invariant.

$$\boldsymbol{\mu} = E(\mathbf{r}_t), \boldsymbol{\Gamma}_0 = E[(\mathbf{r}_t - \boldsymbol{\mu})(\mathbf{r}_t - \boldsymbol{\mu})'], \tag{21}$$

where the expectation is taken element by element over the joint distribution of $\mathbf{r}_t$. The mean $\boldsymbol{\mu} = (\mu_1, \mu_2, ...\mu_k)'$ is a k-dimensional vector consisting of the unconditional expectations of the components of $\mathbf{r}_t$. The covariance matrix $\boldsymbol{\Gamma}_0 = [\Gamma_{ij}(0)]$ is a $k \times k$ matrix. The i-th diagonal element of $\boldsymbol{\Gamma}_0$ is the variance of r_{it}, whereas the (i, j)th element of $\boldsymbol{\Gamma}_0$ is the covariance between r_{it} and r_{jt}, (Tsay, 2010).

Cross-Correlation Matrices:

$$\boldsymbol{\rho}_0 \equiv [\rho_{ij}(0)] = \mathbf{D}^{-1}\boldsymbol{\Gamma}_0\mathbf{D}^{-1}, \tag{22}$$

$$\mathbf{D} = diag\{\sqrt{\Gamma_{11}(0)}, \sqrt{\Gamma_{22}(0)}, ...\sqrt{\Gamma_{kk}(0)}\}, \tag{23}$$

$\boldsymbol{\rho}_0$ *is the correlation coefficient between* r_{it} *and* r_{jt} .

lag-cross-covariance matrix:

$$\mathbf{\Gamma}_l \equiv \Gamma_{ij}(l) = E[(\mathbf{r}_t - \boldsymbol{\mu})(\mathbf{r}_{t-l} - \boldsymbol{\mu})'], \tag{24}$$

The covariance matrix $\Gamma_l = [\Gamma_{ij}(l)]$ *is a* $k \times k$ *matrix. The* (i, j)th *element of* $\mathbf{\Gamma}_l$ *is the covariance between* r_{it} *and* $r_{j,t-l}$. For a weakly stationary series, the cross-covariance matrix $\mathbf{\Gamma}_l$ is a function of l, not the time index t.

The lag l cross-correlation matrix (CCM):

$$\boldsymbol{\rho}_l \equiv [\rho_{ij}(l)] = \mathbf{D}^{-1}\mathbf{\Gamma}_l\mathbf{D}^{-1} \quad l \geq 0, \tag{25}$$

Sample CCM:

$$\hat{\boldsymbol{\rho}}_l = \hat{\mathbf{D}}^{-1}\,\hat{\mathbf{\Gamma}}_l\,\hat{\mathbf{D}}^{-1}, \tag{26}$$

given the data $\{\mathbf{r}_t \big| t = 1,2,...T\}$, $\hat{\mathbf{\Gamma}}_l = \frac{1}{T}\sum_{t=l+1}^{T}(\mathbf{r}_t - \overline{\mathbf{r}})(\mathbf{r}_{\mathbf{t}-l} - \overline{\mathbf{r}})', l \geq 0$, $\overline{\mathbf{r}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{r}_t$,

$\hat{\mathbf{D}}$ is the *k* × *k* diagonal matrix of the sample standard deviations of the component series. (Tsay, 2010).

Multivariate Portmanteau Tests

The univariate Ljung–Box statistic Q_m has been generalized to the multivariate case by Hosking (1980, 1981) and Li and McLeod (1981). For a multivariate series, $H_0 : \boldsymbol{\rho}_1 = \boldsymbol{\rho}_2 = ... = \boldsymbol{\rho}_m = \mathbf{0}$; and $H_1 : \boldsymbol{\rho}_i \neq \mathbf{0}, i \in \{1,2,...m\}$. Under H_0, then, $Q_m \sim \chi^2_{k^2 m}$ approximately.

$$Q_k(m) = T^2\sum_{l=1}^{m}\frac{1}{t-l}tr(\hat{\mathbf{\Gamma}}_l'\,\hat{\mathbf{\Gamma}}_0^{-1}\,\hat{\mathbf{\Gamma}}_l\,\hat{\mathbf{\Gamma}}_0^{-1}), \tag{27}$$

where T is the sample size, k is the dimension of $\mathbf{r}_t$, and $tr(A)$ is the trace of the matrix A, which is the sum of the diagonal elements of A.

Integration:

A stochastic process X_t is called integrated of order d, I(d), d = 0, 1, 2, . . . if $\Delta^d(X_t - E(X_t))$ is I(0).

Cointegration:

The I(d) process X_t is called cointegrated CI (d, b) with cointegrating vector $\beta \neq 0$ if $\beta' X_t$ is I (d − b), b = 1, . . . , d, d is integer, and d >0

Johansen test:

The Johansen test is a procedure for testing cointegration of several $I(1)$ time series. This test permits more than one cointegrating relationship.

There are two types of Johansen test, either with trace or with eigenvalue. For the trace test, $H_0 : r = r^* < k$, r is the number of cointegration vectors. $H_1 : r = k$. And then, testing proceeds sequentially for $r^*=1,2$, etc. and the first acceptation of H_0 is taken as an estimate of r. For the "maximum eigenvalue", $H_0 : r = r^* < k$, $H_1 : r = r^* + 1$. And then, testing proceeds sequentially for $r^*=1,2$ etc., with the first non-rejection of H_0 is taken as an estimate of r.

VECM

Vector autoregressive (VAR) model is the commonly used in multivariate time series, while it can only describe the dynamic interrelationship among stationary variables. A vector error correction (VEC) model is a restricted VAR designed for use with non-stationary series that are known to be cointegrated. The multivariate time series Z_t follows a VAR model of order p, VAR(p), if

Consider the p-dimensional autoregressive process X_t defined by the equations.

$$\phi(B)Z_t = \phi_0 + a_t, \tag{28}$$

Subtracting Z_{t-1} on both sides of the equation and rearranging terms yields the VECM

$$\Delta Z_t = \Pi Z_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta Z_{t-i} + a_t, \tag{29}$$

where $\phi(B) = I_k - \sum_{i=1}^{p} \phi_i B^i$, ϕ_0 is a k-dimensional constant vector and ϕ_i are $k \times k$ matrices for i > 0, $\phi_p \neq 0$ and a_t is a sequence of independent and identically distributed random vectors with mean zero and covariance matrix Σ_a, which is positive-definite. Π=-$(I_k$-ϕ_1-ϕ_2-...-$\phi_p)$, $\Gamma_i = -(\phi_{i+1} + ... + \phi_p), \quad i = 1, 2, ... p-1.$

In this thesis, we build VECM by following steps.

1. Testing the stationary of each series by ADF Test separately.
2. Fitting the VAR model and choosing the lag p with the minimum AIC.
3. Testing the cointegration of the data by Johansen test.
4. Checking the residuals of the fitted models by Multivariate Portmanteau Tests and CCM
5. Forecasting.

4.4 Introduction of Machine Learning Methods

4.4.1 Neural Network (MLP and LSTM)

The process of Neural Network is shown in Figure 2. Neural Network aims to find a set of values for the weights of all layers with the minimum loss scores. Its weights parameterize the transformation implemented by a layer. The loss function takes the predictions and the true values and then computes a distance score (loss score). The loss score shows how well the network has done on this dataset. The optimizer is used in the Neural Network to adjust the values of weights to obtain a lower loss score. The loss score is likely to be very high at first because the weights of the network are assigned random values. With the network processes, the weights are adjusted a little in the correct direction and the loss score decreases. This process is called the training loop. After repeating the training loop a sufficient number of times, then weight values can minimize the loss function. A network with a minimal loss is one for which the outputs are as close as they can be to the targets: a trained network. In machine learning, the dataset should be split into three sets, and they are a train, validation, and test. A train set is used in the training loop to train the network. The validation set

is used to check overfitting. The model is overfitting to the training if the performance of the model on the validation set peaked after a few epochs and then began to degrade. The test set is used to calculate the forecasting accuracy of this model, (Chollet, 2018). In this thesis, MSE is chosen as the loss function, and Rectified Linear Unit (ReLU) is applied as the Optimizer. ReLU: $f(x) = \max(0, x)$.

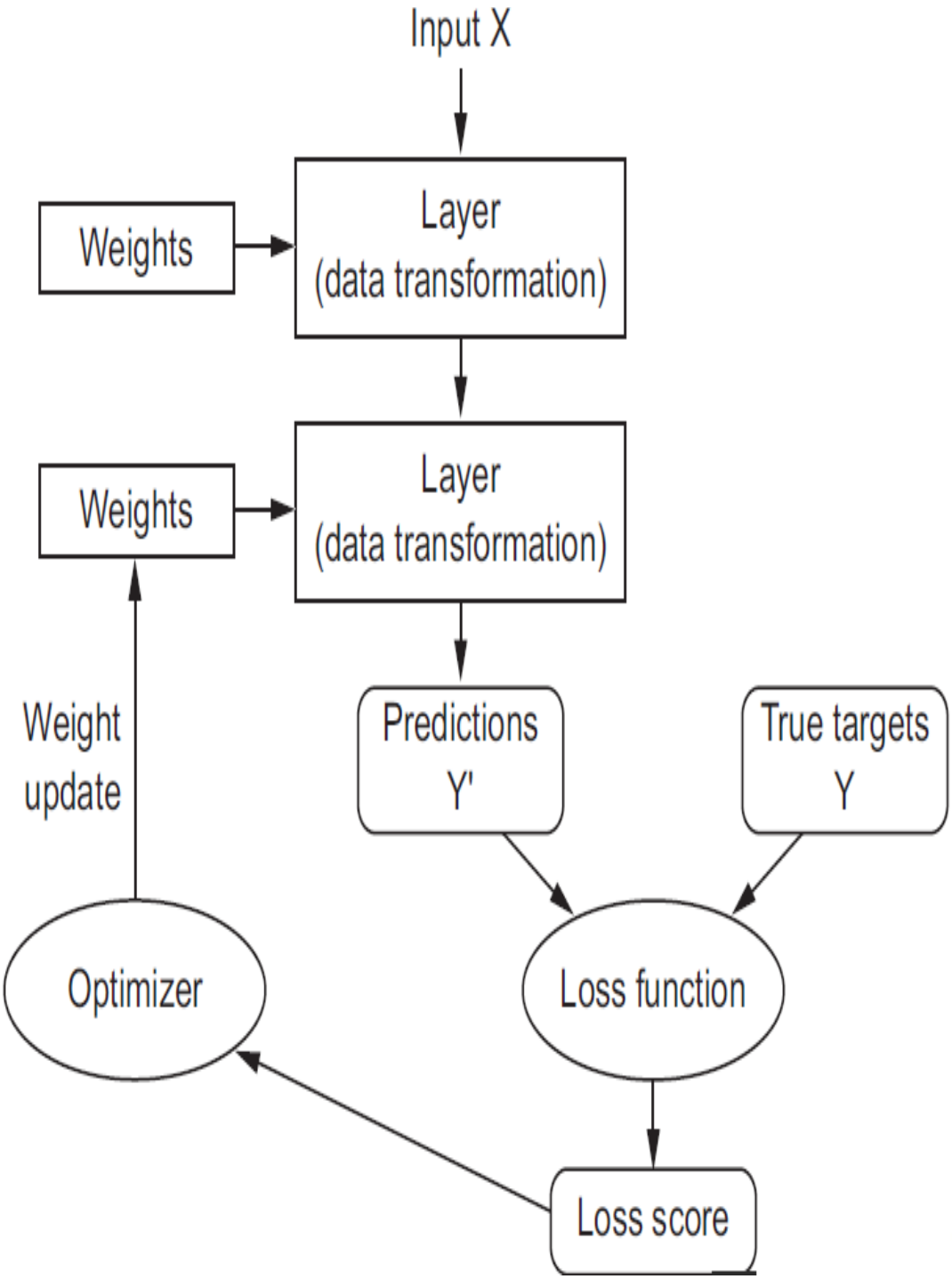

Figure 2 Introduction of Neural Network

Multilayer perceptron

A multilayer perceptron (MLP) is a class neural network. An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function.

Long short-term memory

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. The LSTM was proposed by (Hochreiter and Schmidhuber,1997) to avoid the long-term dependency problem present for the case of the latter.

4.4.2 Support Vector Regression:

SVM aims at solving classification problems by finding ethical decision boundaries (separation hyperplane) between two sets of points belonging to two different categories. Kernel trick is introduced to transform a nonlinear classification rule to a linear classification rule for the transformed data points by increasing the dimension of data.

SVMs proceed to find these boundaries in two steps:

1 The data is mapped to a new high-dimensional representation where the decision boundary can be expressed as a hyperplane.

2 A separation hyperplane is computed by trying to maximize the distance between the hyperplane and the closest data points from each class, a step called maximizing the margin (Chollet, 2018).

Support Vector Regression (SVR) is the regression process performed by a Support Vector Machine. The cost function of the model that produced by support-vector classification ignores the training points that lie beyond the margin, while the cost function of the model produced by SVR ignores the training points close to the model prediction (see *Figure 3*). Training the original SVR means solving:

$$\text{minimize}\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i - \xi_i^*)$$
$$s.t.\begin{cases} y_i - (w'x_i + b) < \varepsilon + \xi_i \\ (w'x_i + b) - y_i < \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* > 0 \end{cases}, \quad (30)$$

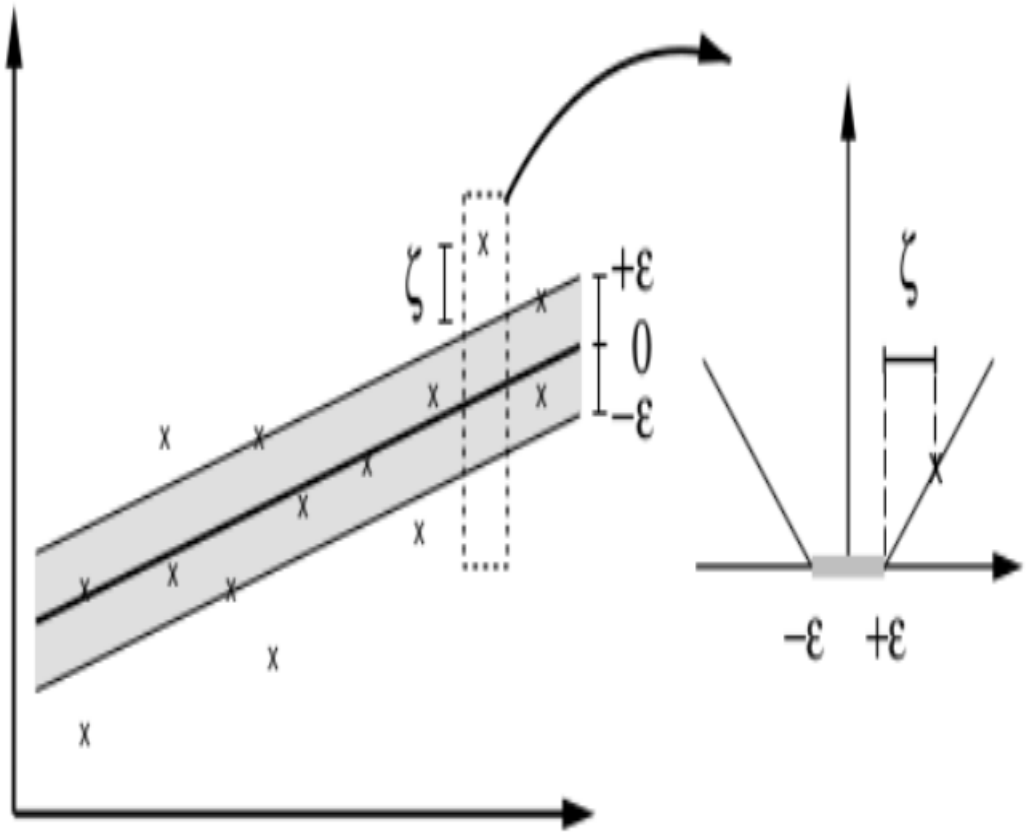

Figure 3 Introduction of SVR

5. Results

This Section is divided into four parts. The first part describes the data that was utilized in this thesis. The second part corresponds to the process of building the time series models. The third part corresponds to the process of building the ML Methods. The final part shows the comparison of forecasting between time series models and machine learning models.

5.1 Data Describing

The data we utilized in this thesis is seasonally adjusted monthly house price indexes (HPI) for nine Census Bureau divisions and the U.S., which is a multivariate data and U.S.; HPI is selected as univariate data, downloaded from the Federal Housing Finance Agency, ("House Price Index Datasets | Federal Housing Finance Agency", 2014). The data period is from February 1991 to May 2019 with 341 observations. *Table 1* depicts the mean, standard deviation (sd), median, min, max, range, kurtosis, standard error (se). From mean, median, The HPI of Census Bureau division Mountain is the highest, while the HPI of Census Bureau division East North Central is the lowest. From SE and sd, the variability of Census Bureau division Mountain's HPI is the maximum, while the variability of Census Bureau division East North Central's HPI is the lowest. *Figure 4* and *Figure 9* indicate the time plots of the ten series. All series indicate a clear upward trend. RMSE in this thesis is used to measure the forecasting accuracy.

In this chapter, we obtained three types of data, called differenced data, normalized data and differenced+normalized data, by applying data preprocessing difference, normalize, and the joint of difference and normalize. There are four sets of data that are utilized in this chapter to fit the models which are raw data (data without any data preprocessing), differenced data, normalized data, differenced+normalized data.

Table 1 Describe data

	mean	sd	median	min	max	kurtosis	se
East North Central	161.44	31.52	165.93	100.00	228.39	-0.71	1.71
East South Central	166.90	38.72	176.54	100.00	252.49	-0.88	2.10
Middle Atlantic	165.47	49.29	191.70	98.61	245.03	-1.61	2.67
Mountain	211.06	70.26	206.68	98.62	376.00	-0.77	3.81
New England	173.46	54.12	196.46	95.28	259.55	-1.49	2.93
Pacific	180.61	69.82	179.16	94.97	318.14	-1.24	3.78
South Atlantic	174.42	51.90	178.86	100.00	281.10	-1.20	2.81
West North Central	179.03	45.73	192.17	100.00	272.99	-0.96	2.48
West South Central	173.22	50.73	171.87	99.71	287.62	-0.74	2.75
USA	173.39	49.14	182.00	100.00	275.03	-1.13	2.66

5.2 Statistical Methods

5.2.1 ARIMA model

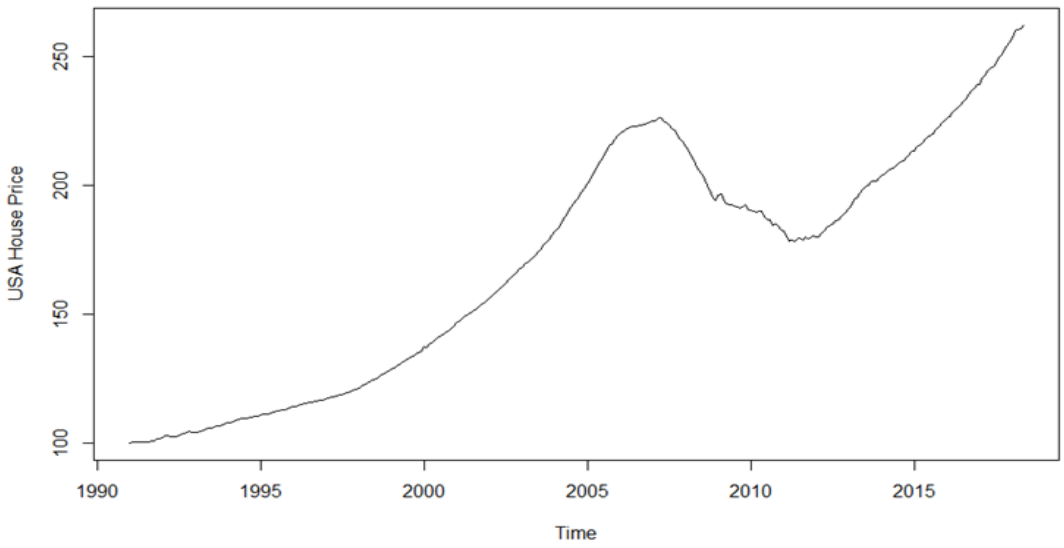

Figure 4 HPI for the US

Figure 4 shows U.S. monthly house price indexes for the U.S. There are 341 months measured from January of 1991 to May of 2019. Modelling such series commences by observing the primary patterns in the time history. In this case, note the gradually increasing underlying trend over the period. The first differenced series of U.S. HPI are indicated in the first subplot of *Figure 5*. The second subplot of *Figure 5* indicates the sample ACF of the first order differenced monthly HPI for the U.S. The third subplot in *Figure 5* depicts the sample PACF of the first differenced seasonally adjusted HPI of U.S. The dashed lines shown on the ACF or PACF plots denote the standard error limits. The ACF and PACF plots show that the sequence of ACF and PACF are significant at first and decrease very gradually. It implies that first differenced seasonally adjusted HPI of U.S is not stationary. The first subplot of *Figure 6* shows the time plot of second-order differenced monthly HPI. The second and third subplot depicts sample ACF and PACF of the second-order difference series. The ACF of the second-order differenced monthly HPI is continuously small. It denotes that the second-order differenced series is stationary.

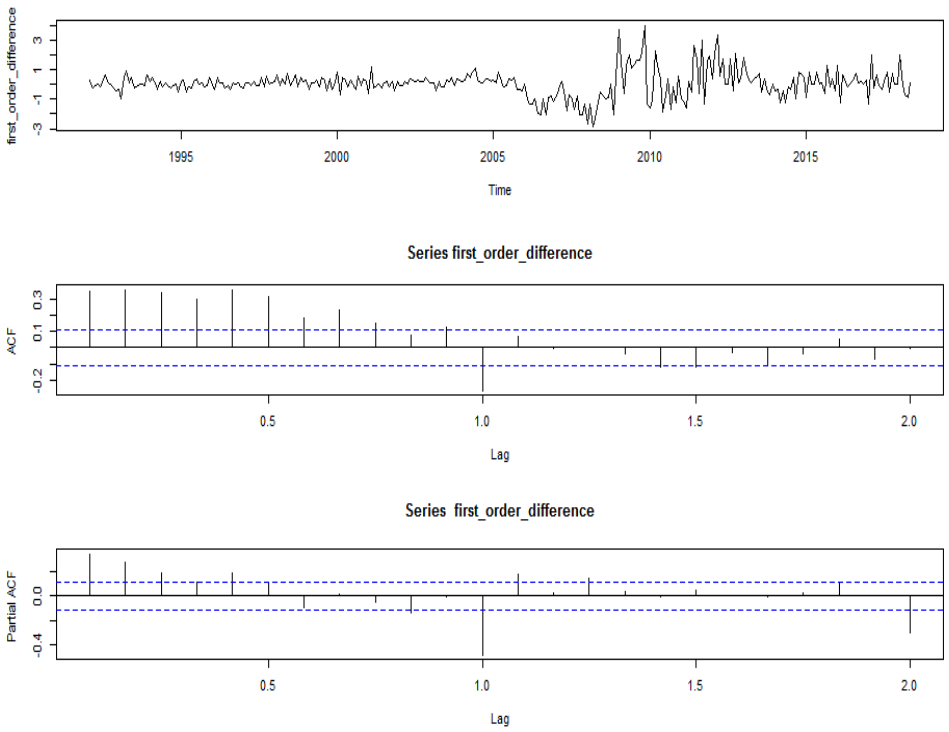

Figure 5 Time Plot and Correlogram Plot for the First Order Differenced Series

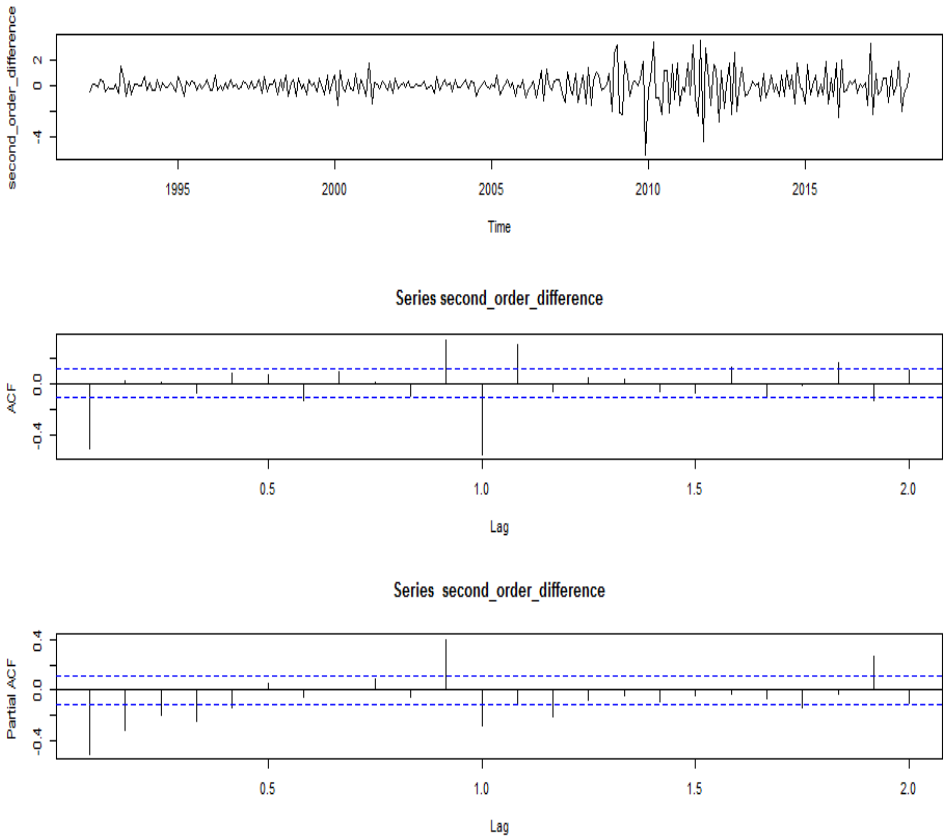

Figure 6 Time Plot and Correlogram Plot for the Second Order Differenced Series

Seasonal ARIMA model with the minimum AIC was fitted. The model is:

$SeasonalARIMA(0,2,1)\times(0,0,2)_{12}$.

Figure 7 shows a diagnose plot of the standardized residuals, the ACF of the residuals, and the p-values linked to the Ljung-Box-statistic. Inspection of the time plot of the standardized residuals in the first subplot of *Figure 7* depicts that there might be an ARCH effect for the series; however, volatility for series in this thesis is not considered. Therefore, we do not need to continue building the volatility model. The ACF of the standardized residuals shows no apparent departure from the model assumptions, (see the second subplot of *Figure 7*). The Ljung-Box-statistic of residuals is not significant at the lags as shown (see the second subplot of *Figure 7*). The forecast of US HPI is as shown in *Figure 8*.

We also fit another model by normalized data. The fitting process is identical, and therefore we do not show the exact fitting process of normalized data here.

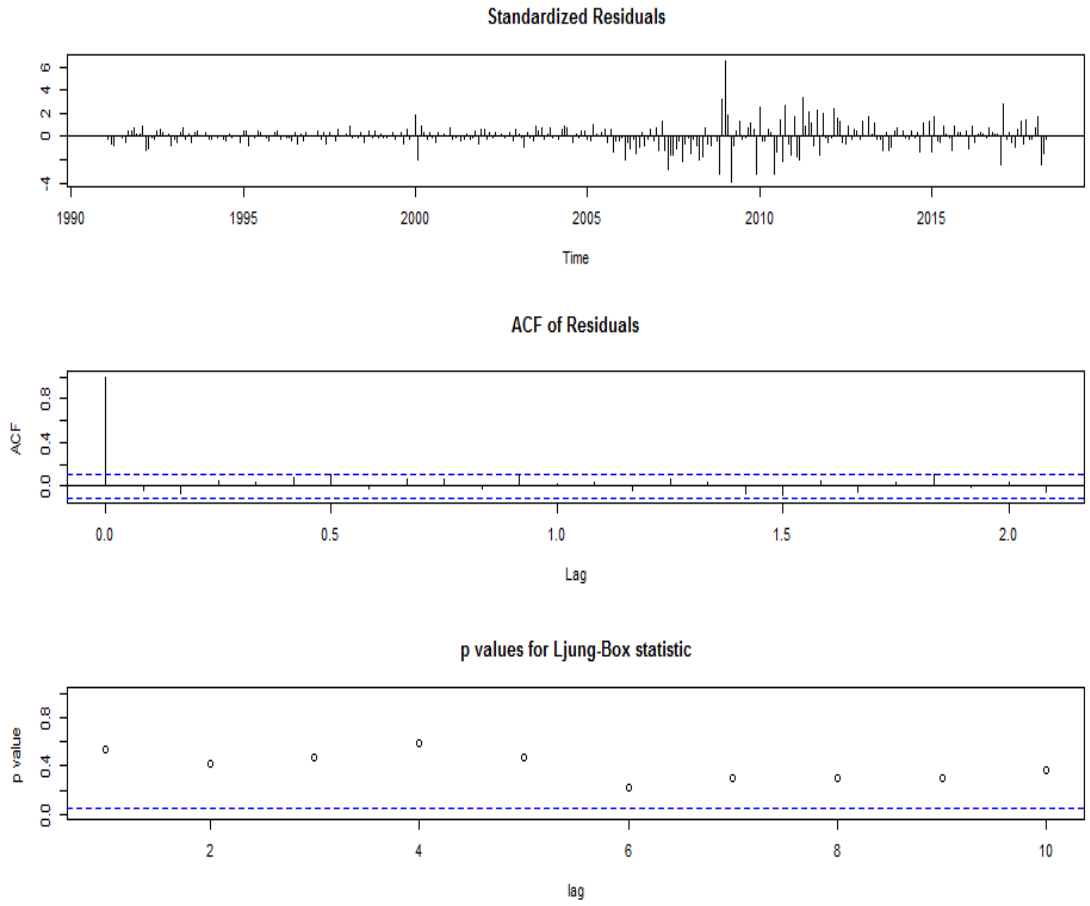

Figure 7 Diagnose Plot of the Standardized Residuals

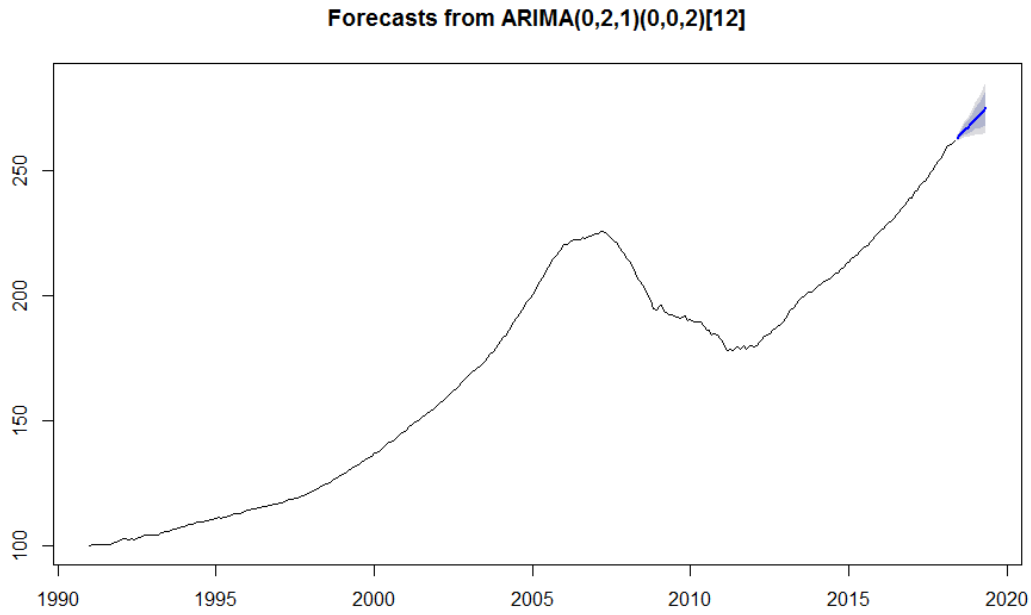

Figure 8 The forecast of US HPI

5.2.2 VAR

Figure 9 depicts the seasonally adjusted monthly HPI for nine Census Bureau divisions. There are 341 months measured from January of 1991 to May of 2019. Therefore, for the multivariate analysis, the model is built by a ten dimension time series (HPI for nine Census Bureau divisions and HPI for the U.S.). *Table 2* shows the ADF Test's results of ten HPIs. *Table 3* display the ADF Test's outcomes of ten first-order differenced HPIs. *Table 4* shows the ADF Test's results of ten second-order differenced HPIs. Equation (11) depicts three main versions of the ADF Test, tau3 refers to δ in model (c), phi3 refers to β in the model (c), phi2 refers to α in model (c), tau2 refers to δ in model (b), phi1 refers to α in model (b), and tau1 refers to δ in model (a).

For East North Central, East South Central, Middle Atlantic, Mountain, New England, South Atlantic, West North Central, USA variables, the null hypothesizes of tau3, phi3, tau2, and tau1 accept the critical values of 5% (see *Table 5*), but reject the null hypothesis of phi2 and phi1. It means that there is a unit root and drift but without trend in these series. All parameters' null hypothesis is accepted at critical values of 5% (see *Table 5*). It implies that there is a unit root in the 12 lag differenced of these series without trend and drift. All parameters' null hypothesis is rejected under critical values of 5%. It means that there is no unit root in the second-order differenced of these

series. For the Pacific variable, the null hypothesis of all parameters is accepted at critical values of 5%. It implies that there is a unit root but without drift and trend in the Pacific series and its 12 lag first-order-differenced series. All parameters' null hypothesis was rejected at critical values of 5%. It means that there is no unit root in the second-order differenced of Pacific series. For West South Central, the null hypothesis of tau3, tau2, and tau1 are accepted at critical values of 5% but reject the null hypothesis of phi3, phi2, and phi1. It implies that there is a unit root, drift, and trend in the West South Central series. All parameters' null hypothesis is accepted at critical values of 5%. It means that there is a unit root in the 12 lag differenced of West South Central series without trend and drift. All parameters' null hypothesis is rejected at critical values of 5%. It implies that there is no unit root in the second-order differenced of West South Central series.

In summary, for the period from January 1991 to May 2019, the series of East North Central, East South Central, Middle Atlantic, Mountain, New England, South Atlantic, West North Central, and USA variables, encompass unit roots and positive drifts, but there is no firm evidence of a time trend. The series of Pacific contains a unit root, but there is no firm evidence of a time trend and drift. The series of West South Central encompasses a unit root and a time trend.

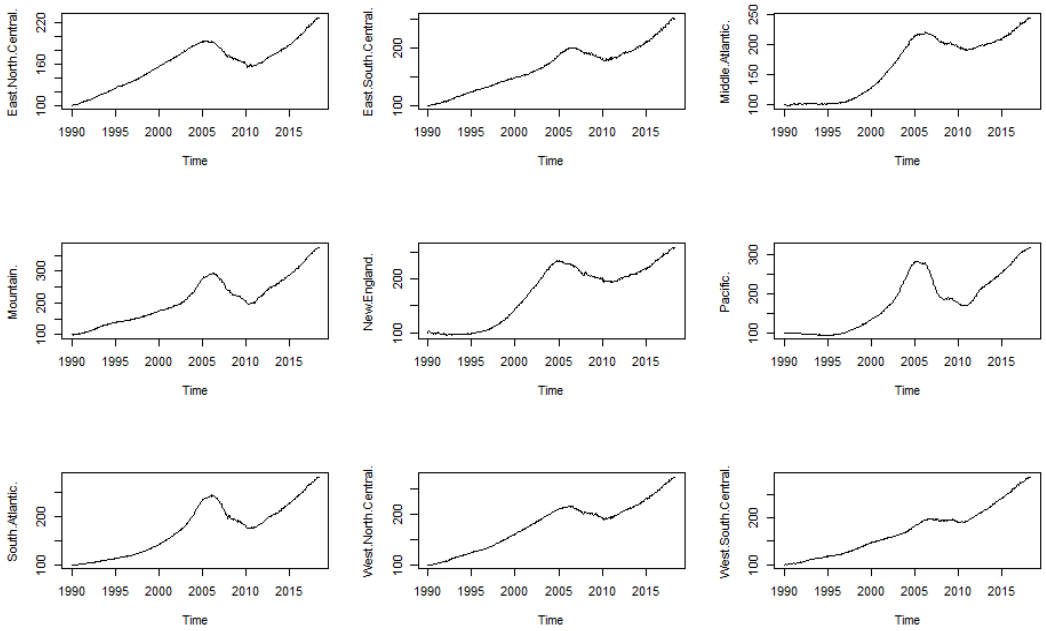

Figure 9 Seasonally Adjusted monthly HPI for nine Census Bureau division

Table 2 ADF test for HPI of nine Census Bureau divisions and US

	tau3	phi2	phi3	tau2	phi1	tau1
East North Central	-0.65	11.34	0.22	-14.23	101.25	5.59
East South Central	-0.18	23.17	0.68	-19.23	184.88	8.29
Middle Atlantic	-0.75	11.82	0.54	-14.14	99.97	5.70
Mountain	-0.23	7.06	0.80	-9.76	47.65	4.52
New England	-0.67	10.01	0.45	-15.13	114.42	5.22
Pacific	-1.09	2.43	1.14	-5.79	16.74	2.18
South Atlantic	-0.59	6.24	0.85	-9.74	47.45	4.17
West North Central	-0.29	24.53	0.42	-15.58	121.42	8.42
West South Central	1.90	45.49	12.09	-17.95	161.01	11.54
USA	-0.82	4.88	0.89	-7.65	29.25	3.62

Table 3 ADF test for 12 lag First-Order-Differenced HPI of nine Census Bureau divisions and US

	tau3	phi2	phi3	tau2	phi1	tau1
East North Central.	-1.02	0.60	0.77	-1.01	0.64	0.51
East South Central	-1.64	1.07	1.49	-1.54	1.30	-0.76
Middle Atlantic	-1.19	0.54	0.71	-1.17	0.80	-0.72
Mountain	-1.19	0.60	0.82	-1.10	0.68	-0.74
New England	-1.25	0.64	0.80	-1.26	0.96	-0.77
Pacific	-1.46	0.73	1.07	-1.45	1.08	-1.27
South Atlantic	-1.15	0.54	0.71	-1.10	0.71	-0.76
West North Central	-1.54	0.91	1.22	-1.50	1.28	-0.65
West South Central	-1.90	1.27	1.80	-1.62	1.42	-0.61
USA	-1.27	0.61	0.83	-1.22	0.84	-0.77

Table 4 ADF test for Second-order-differenced HPI of nine Census Bureau divisions and US

	tau3	phi2	phi3	tau2	phi1	tau1
East North Central.	-14.59	70.92	106.38	-14.57	106.20	-14.58
East South Central	-19.66	128.90	193.33	-19.66	193.37	-19.67
Middle Atlantic	-14.57	70.73	106.10	-14.59	106.42	-14.60
Mountain	-10.24	34.94	52.41	-10.25	52.50	-10.25
New England	-15.64	81.59	122.38	-15.67	122.73	-15.67
Pacific	-6.07	12.29	18.44	-6.08	18.49	-6.09
South Atlantic	-9.95	33.00	49.50	-9.96	49.64	-9.97
West North Central	-16.02	85.60	128.39	-16.04	128.65	-16.05
West South Central	-18.90	119.16	178.73	-18.94	179.29	-18.94
USA	-7.82	20.41	30.61	-7.83	30.70	-7.84

Table 5 Critical for ADF Test

	tau3	phi2	phi3	tau2	phi1	tau1
1pct	-3.98	6.15	8.34	-3.44	6.47	-2.58
5pct	-3.42	4.71	6.30	-2.87	4.60	-1.95
10pct	-3.13	4.05	5.36	-2.57	3.79	-1.62

The ADF test confirm that the ten series are unit-root nonstationary. All are I(2) series. We still utilize Johansen's cointegration tests in the I(2) series. To apply cointegration tests, we employ VAR models and select the value of lag based on the minimum AIC, and then select a VAR(12) model. Therefore, we use a VAR(11) model in cointegration tests and a VAR(12) specification in estimation. With K = 11, the cointegration test indicate that the eigenvalues are 0.32, 0.24, 0.22 0.16, 0.15, 0.12, 0.11, 0.07, 0.02, 0.003, respectively. The test statistics and some critical values are provided inTable 6. Table 6 depicts that we cannot reject the null hypothesis of m = 8. Eight cointegrating vectors are y_1(1, 0.03, 1.11, 0.49, 0.06, 0.69, 0.66, 0.23, 0.85, -5.20)', y_2 (1, 40.32, -83.02, -103.30, -26.11, -154.99, -105.68, -305.33, -67.94, 896.13)', y_3(1,-0.15, 0.60, 0.48, 0.19, 0.45, -0.25, -0.24, 0.76, -2.90), y_4(1, 0.38, 0.45, 0.48, 0.80, 0.93, 0.68, 0.32, 0.91, 0.91)', y_5(1,-0.36, 0.46, 0.15, 0.17, 0.15, 0.86, -0.41, 0.71, -2.64)', y_6(1, -1.25, 1.31, -0.74, -1.74, 0.26, 0.79, 1.46, 0.09, -0.64)', y_7(1, 9.57, 9.15, -10.79, -26.83, -5.32, -55.01, -41.78, -15.72, 146.84)', y_8(1, -0.15, -0.60, -0.48, 1.81, 2.42, 9.78, 5.35, 3.29, -23.06)', respectively. These eigenvectors are normalized

so that the first element of each vector is 1. Let $\beta = [\mathbf{y_1}, \mathbf{y_2}, \mathbf{y_3}, \mathbf{y_4}, \mathbf{y_5}, \mathbf{y_6}, \mathbf{y_7}, \mathbf{y_8}]$ and $\mathbf{w}_t = \left(w_{1t},\ w_{2t}, w_{3t}, w_{4t}, w_{5t}, w_{6t}, w_{7t}, w_{8t}\right)' = \beta z_t$. We estimate the ECM-VAR(12) model for the ten-dimensional monthly series. *Figure 10* plots the p-values of Ljung–Box test statistics for the residuals of the fitted ECM model. The Ljung-Box-statistic of residuals are not significant at the lags presented. *Figure 11* and *Figure 12* depicts the time plots of the corresponding residuals. There are no apparent patterns of standardised residuals. The model fits the data well. *Figure 13* shows the forecasting of HPI of nine Census Bureau divisions and the US.

We also fit another model by normalized data. The fitting process is identical, and therefore we do not present the precise fitting process of normalized data here.

Table 6 Cointegration Tests for a Ten-Dimensional seasonally adjusted monthly HPI for nine Census Bureau divisions and U.S. Based on a VAR(11)Model

	test	10%	5%	1%
r <= 9	0.80	6.50	8.18	11.65
r <= 8	8.60	15.66	17.95	23.52
r <= 7	32.45	28.71	31.52	37.22
r <= 6	68.91	45.23	48.28	55.43
r <= 5	108.48	66.49	70.60	78.87
r <= 4	161.87	85.18	90.39	104.20
r <= 3	218.93	118.99	124.25	136.06
r <= 2	297.87	151.38	157.11	168.92
r <= 1	386.15	186.54	192.84	204.79
r = 0	507.31	226.34	232.49	246.27

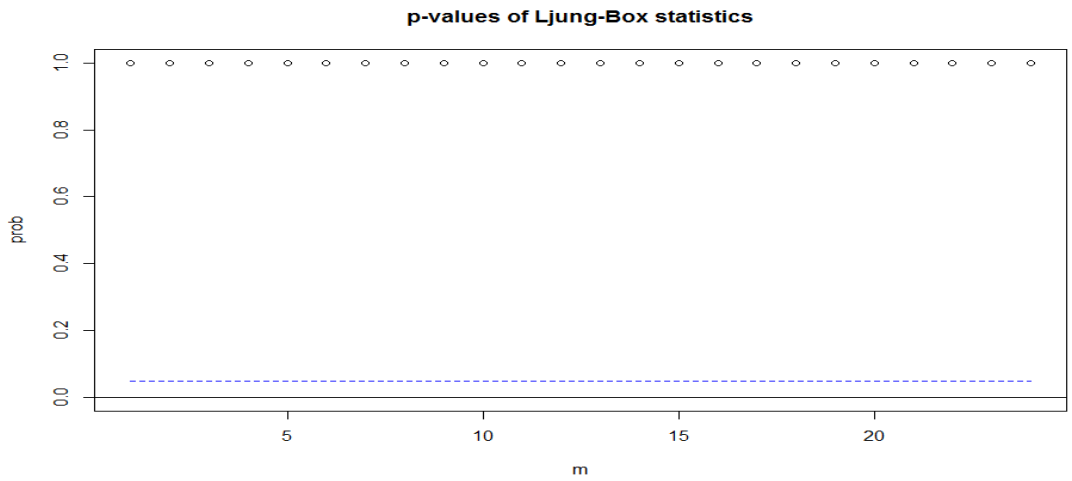

Figure 10 Plot of p-values of the Ljung–Box statistics for the residuals of ECM-VAR(12)

model for ten-dimensional U.S. Monthly HPIs

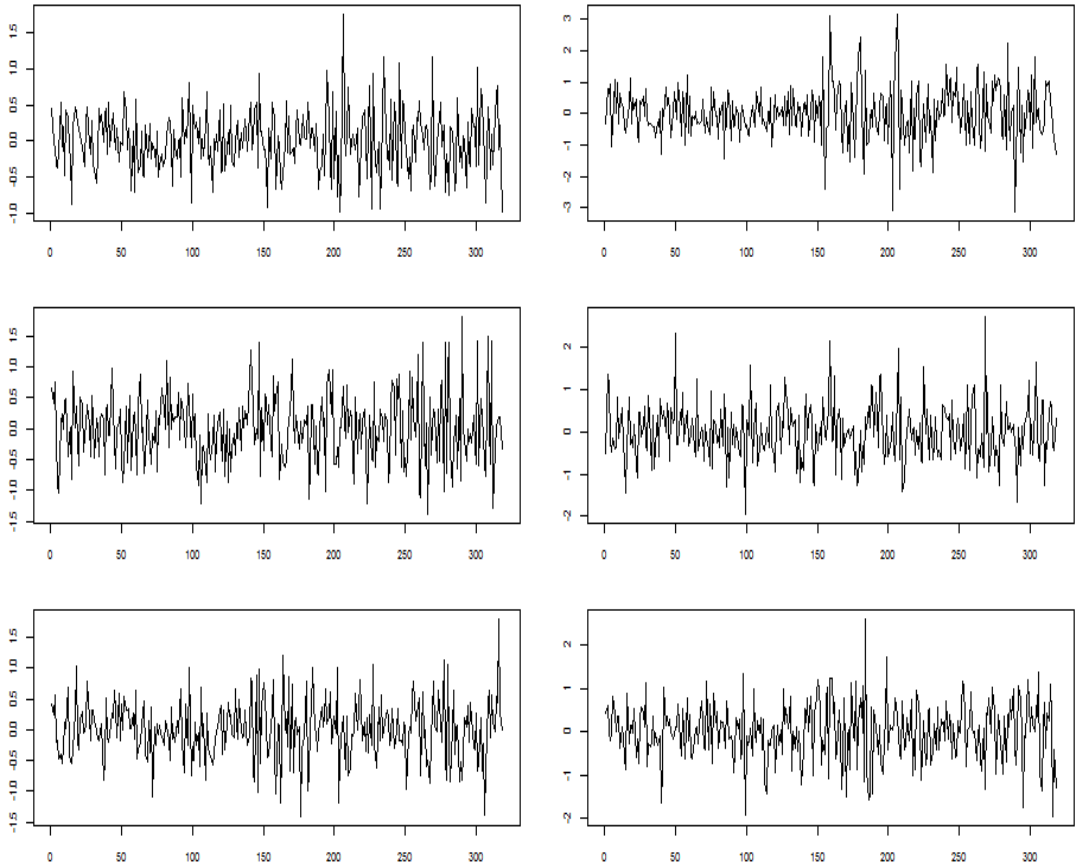

Figure 11 Time Plots of the Corresponding Residuals(1).

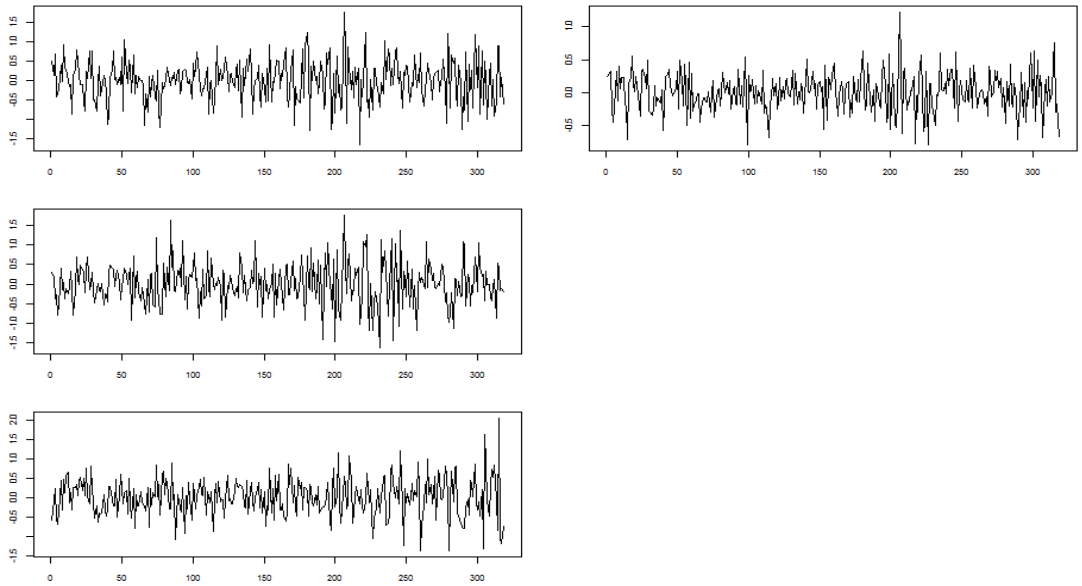

Figure 12 Time Plots of the Corresponding Residuals(2).

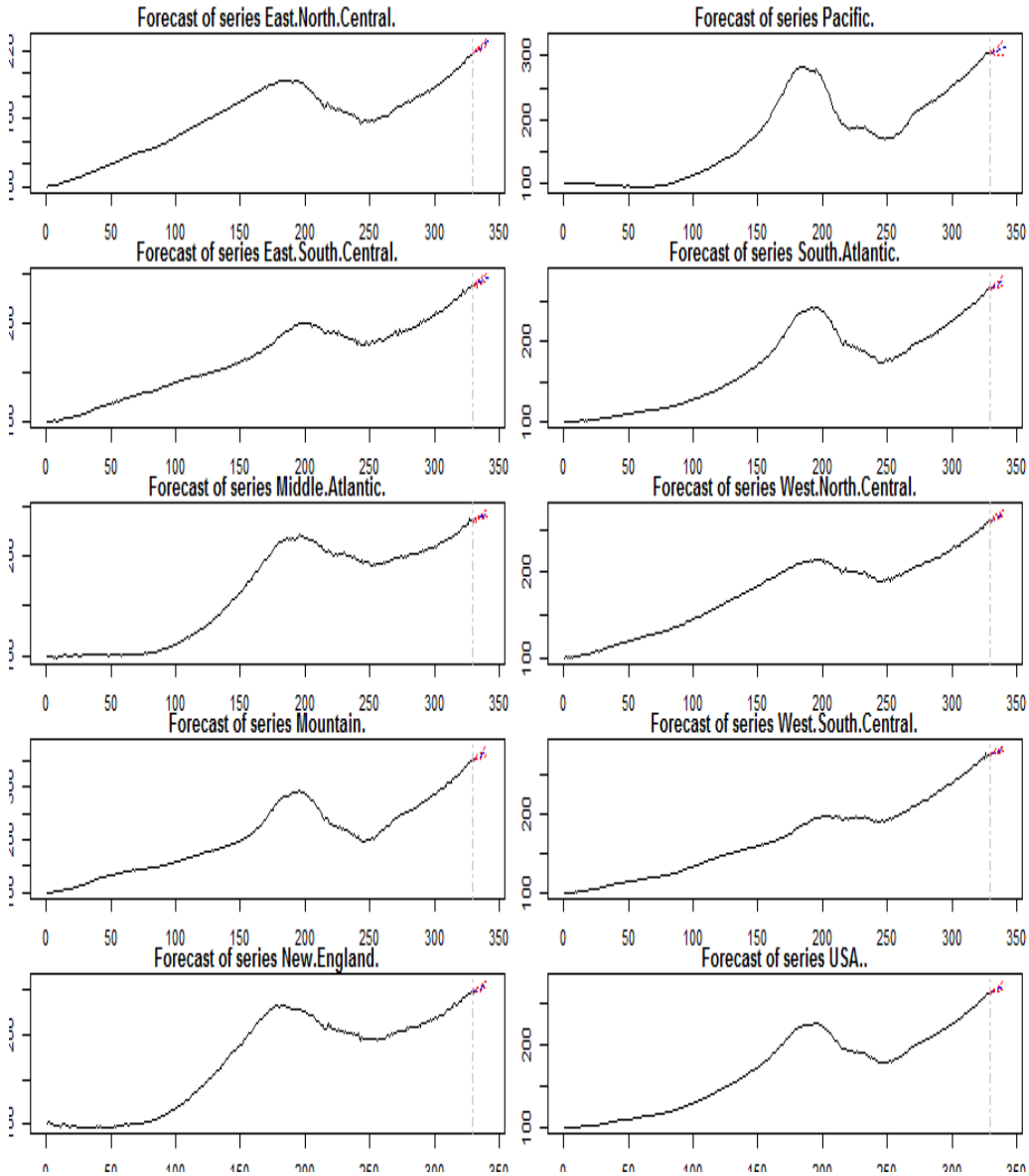

Figure 13 Forecasting of HPI of nine Census Bureau Divisions and the US

5.3 Machine Learning Methods

5.3.1 Univariate

MLP

For this thesis, an MLP with a single hidden layer was constructed. For the univariate time series dataset, the best number of input nodes (N=6), outputs nodes (N=1), hidden nodes (N=8), epochs (N=200), and batches (N=1) were chosen by grid search using a walk-forward validation. Following this, overfitting of this network was checked by applying the loss plot. The loss of MLP over the training and validation data during

training did not indicate overfitting. For the MLP model, both train and validation loss reached its minimum after about five epochs (see *Figure 14*). Finally, 12 forecasting values were predicted using this network. Initial tests helped determine four sets of forecasting values by integrating this network with raw data, differenced data, normalized data, and differenced+normalized data.

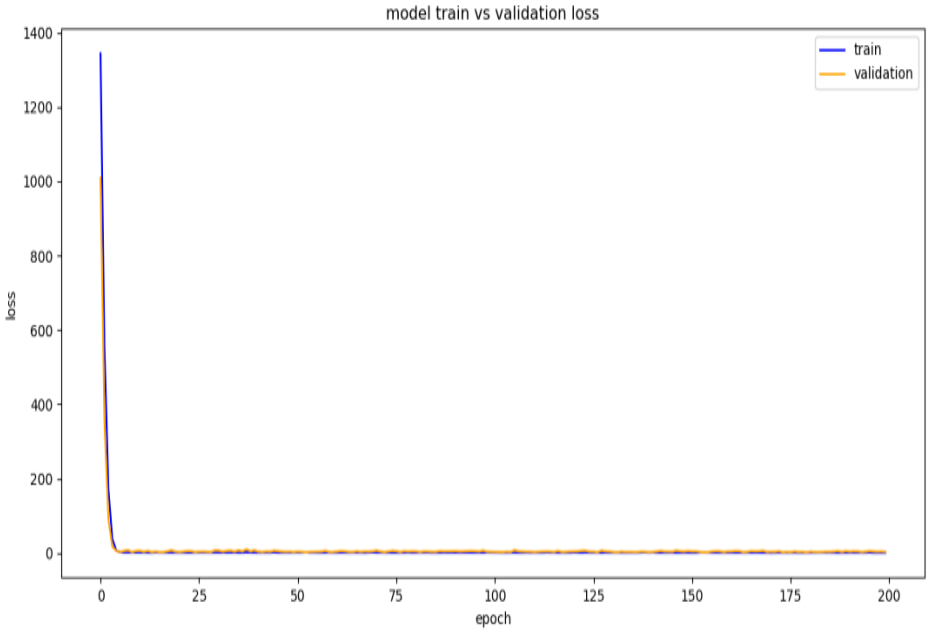

Figure 14 The loss of the MLP over the training and validation data for univariate

LSTM

For the univariate, a Vanilla LSTM model was used, which is an LSTM model with a single hidden layer of LSTM units and an output layer. For Vanilla LSTM on univariate dataset, the best number of input nodes (N=6), outputs nodes (N=1), hidden nodes (N=50), epochs (N=100), and batches (N=1) were defined by grid search using a 10-fold walk forward validation process. Subsequently, the overfitting of this network was verified by the loss plot. The loss of the Vanilla LSTM over the training and validation data during training did not indicate overfitting. For the Vanilla LSTM model, both train and validation loss reached its minimum after about ten epochs (see Figure 15). Finally, 12 forecasting values were predicted using this network. 12 forecasting values were predicted by this network. We obtained four sets of forecasting values by integrating this network with raw data, differenced data, normalized data, and differenced+normalized data.

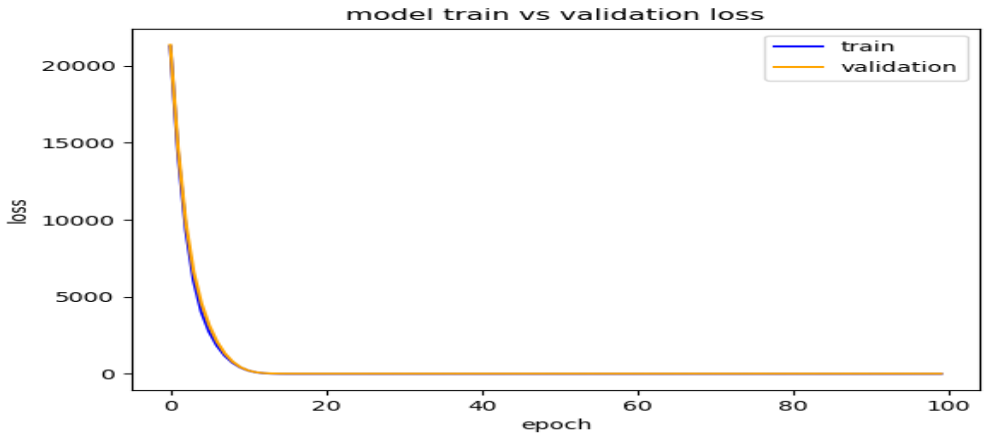

Figure 15 The loss of Vanilla LSTM over the training and validation data

SVR

In this thesis, ε was equal to 0.01. For the univariate dataset, linear kernel was used in training and predicting (linear SVR), mainly due to its efficient general performance and the smaller number of parameters required. We applied the walk-forward validation to choose the value of parameter c in linear SVR, by the principle of the minimum mean squared error (MSE), with the number of inputs equalling to 6. *Figure 16* indicates that the linear SVR model had minimum MSE when c was equal to 5. In this case, 12 forecasting values were predicted using this network. Further tests enabled us to obtain four sets of forecasting values by integrating this network with raw data, differenced data, normalized data, and differenced+normalized data.

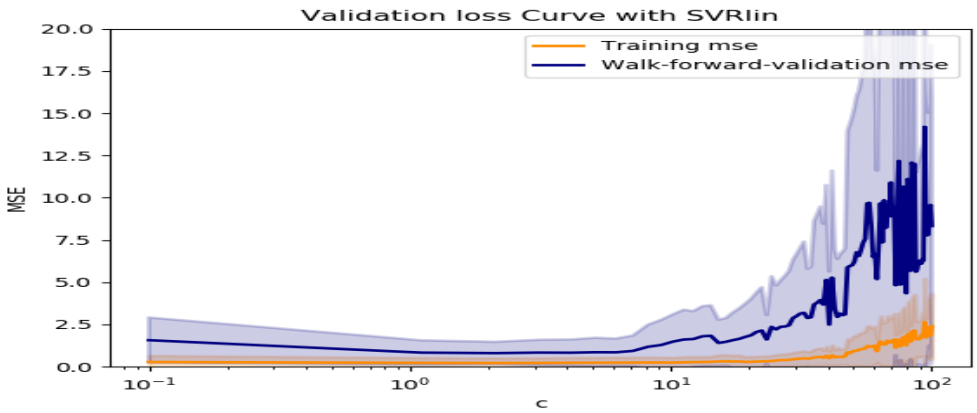

Figure 16 The loss of Linear SVR over the training and validation data

5.3.2 Multivariate

MLP

Firstly, for the multivariate time series dataset, we extended the network of a

univariate dataset to the multivariate dataset. Then, the overfitting of this network was verified using the loss plot. The loss of the MLP over the training and validation data during training did not indicate overfitting. For the MLP model, both the train and validation loss reached its minimum after about 25 epochs (See *Figure 17*). Finally, 12-horizon forecasting values were predicted using this network. 12 forecasting values are predicted by this network. During these tests, four sets of forecasting values were obtained by fitting this network with raw data, differenced data, normalized data and differenced+normalized data.

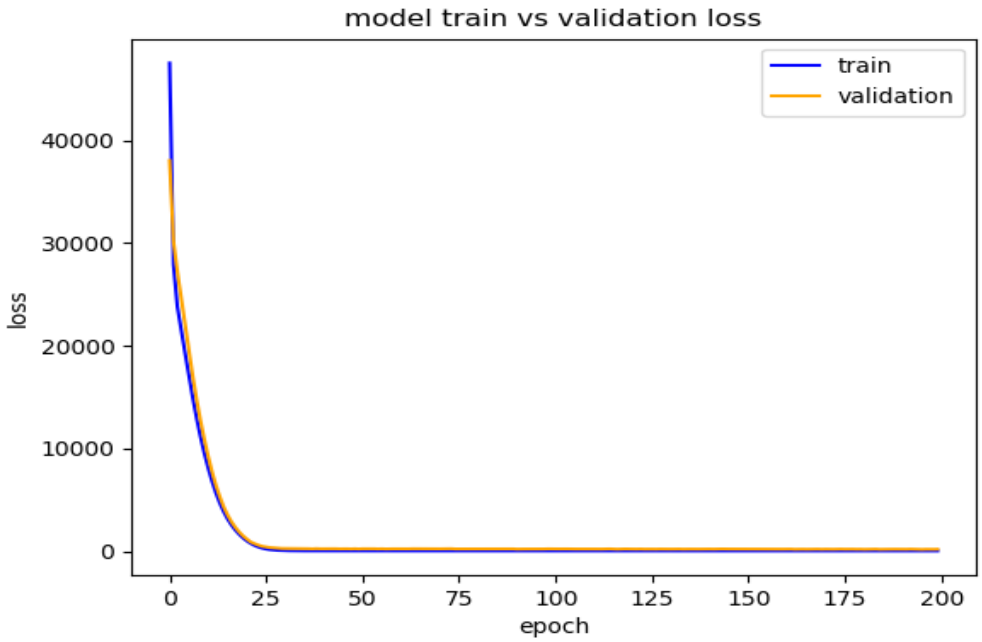

Figure 17 The loss of the MLP over the training and validation data for multivariate

LSTM

We applied an Encoder-Decoder LSTM in a multivariate dataset, which was a model specifically developed for forecasting variable length output sequences. The same values of parameters were used in the Encoder-Decoder LSTM on the multivariate dataset: Input nodes (N=6), outputs nodes (N=1), hidden nodes (N=50), epochs (N=100), and batches (N=1). The loss of the Encoder-Decoder LSTM over the training and validation data during training did not suggest overfitting. For the Encoder-Decoder LSTM model, both the train and validation loss reached its minimum after about five epochs (see Figure 18).Furthermore, 12 forecasting values were predicted using this network. This series of tests helped us obtain four sets of forecasting values by incorporating this network with raw data, differenced data, normalized data, and differenced+normalized data.

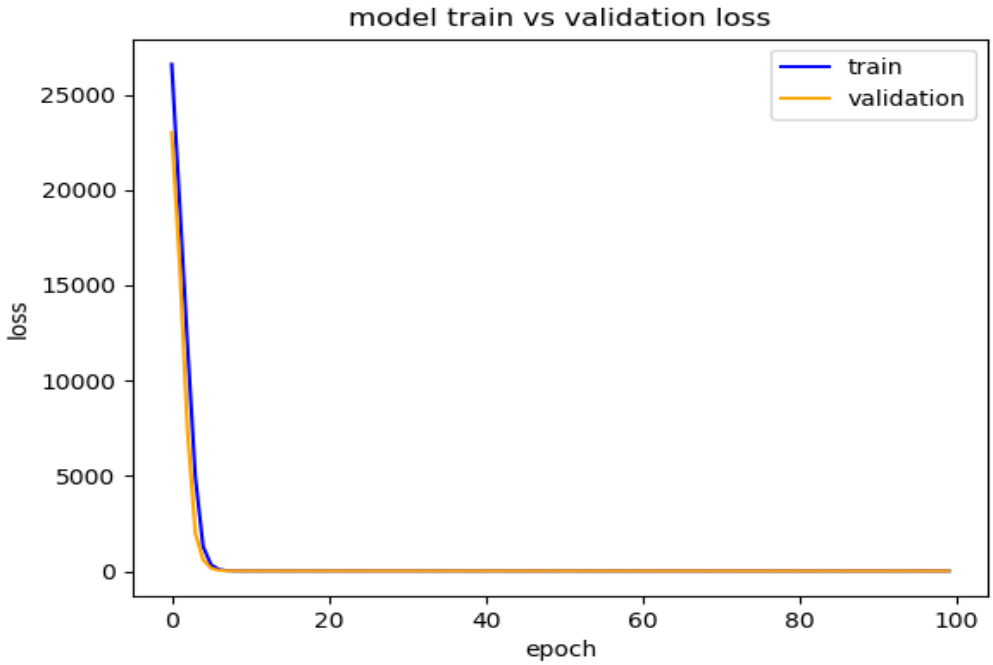

Figure 18 The loss of the Encoder-Decoder LSTM over the training and validation data

SVR

For the multivariate dataset, a radial basic kernel was used in training and predicting (rbf SVR). The linear kernel performed poorly on the multivariate dataset, while the forecasting accuracy of rbf SVRs was found to be comparatively better than that of linear SVRs. The linear SVR did not fit the multivariate data, owing to the accuracy score of linear SVR being comparatively lower. (see *Figure 19*). Therefore, we attempted to apply rbf SVR in the multivariate data. Firstly, we chose the value of parameter r with parameter c equals to 5, to obtain the highest accuracy score by walk forward validation. *Figure 20* illustrates that the scores of both train and validation sets increased and achieved the maximum value when r was equal to 0.1. Then, the value of c was chosen with the parameter r equivalent to 0.1, to obtain the highest accuracy score by walk forward validation. *Figure 21* shows that the MSE of validation sets decreased and reached the minimum value when c was equal to 20. Therefore, the rbf SVR, with parameters c equivalent to 20 and r equivalent to 0.1 was chosen. In other words, multivariate data was matched by applying the rbf SVR model with parameter c equivalent to 20 and r equivalent to 0.1. Finally, 12-horizon forecasting were predicted using this rbf SVR in multivariate analysis. In the end, we obtained four sets of forecasting values by matching this network with raw data, differenced data, normalized data, and differenced+normalized data.

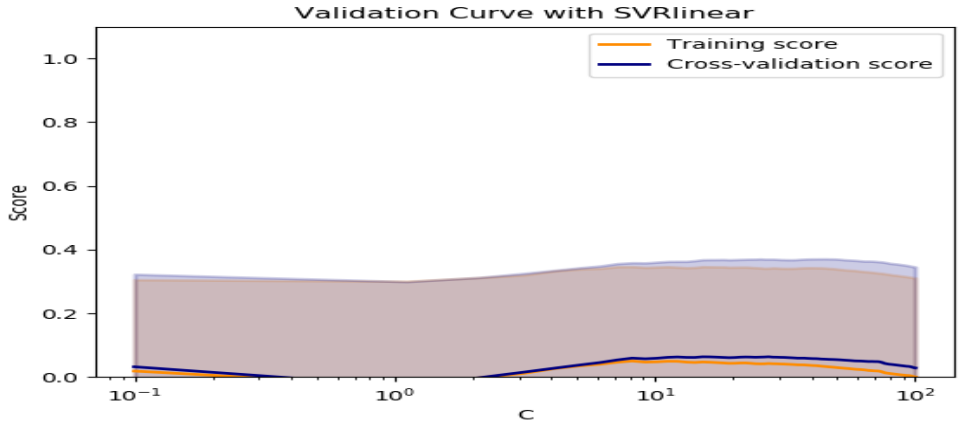

Figure 19 The accuracy score of linear SVR

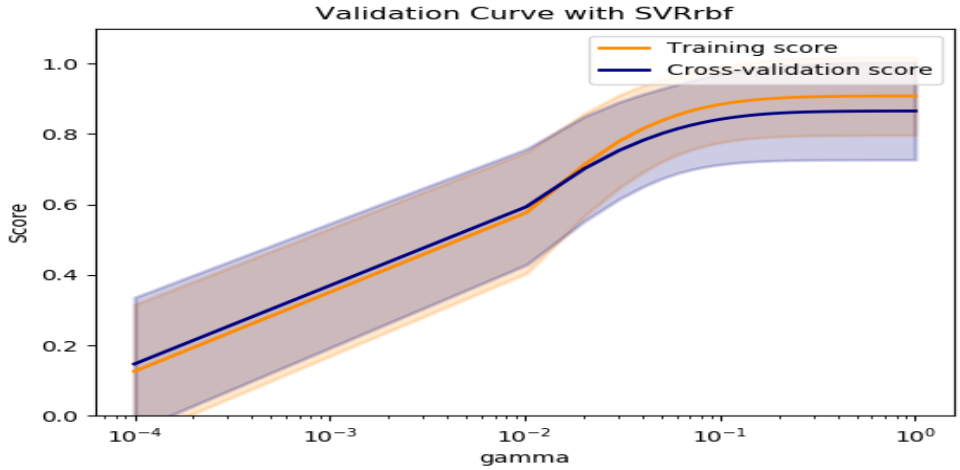

Figure 20 The accuracy score of rbf SVR c=5

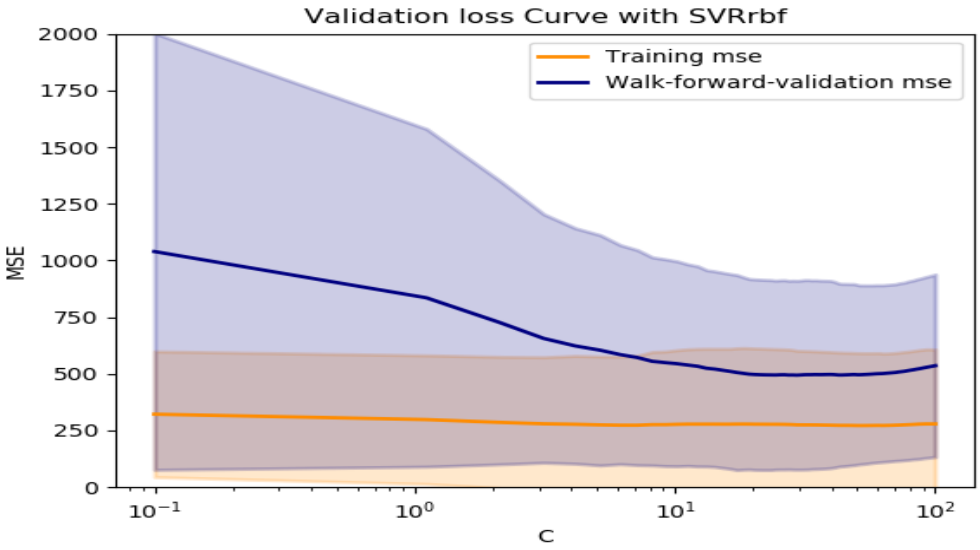

Figure 21 The MSE of rbf SVR, γ=0.1

5.4 Comparison

5.4.1 Univariate

Figure 22 compares the forecasts of different models with similar types of data preprocessing (transformation) from May 2018 to May 2019. It illustrated how forecasting accuracy varied with the increase of prediction steps. The first subplot compared the 12-step-prediction of 4 methods calculated from raw data. It showed that ARIMA, MLP, and Linear-SVR indicated good forecasting performance through all 12 predictions. However, LSTM showed signs of bad forecasting performance. To make matters worse, performance decayed with increase in forecasting steps. The second subplot compared the 12-step-prediction of 3 machine learning methods calculated from differenced data. It showed that Linear-SVR had the best forecasting performance, followed by MLP, while the performance of MLP deteriorated with the increase in forecasting steps. LSTM had a repeat bad forecasting performance and the performance showed a degradation with the increase in forecasting steps. The third subplot compared the 12-step-prediction of 3 machine learning methods calculated from normalized+differenced data. It showed that Linear-SVR and MLP indicated excellent forecasting performance through all 12 predictions. The forecasting performance of LSTM did not indicate entirely poor results. However, the performance of LSTM still decayed with the increase in forecasting steps. The last subplot compared the 12-step-prediction of 4 methods calculated from normalized data. It showed that ARIMA displayed the best forecasting performance through all 12 predictions, followed by Linear-SVR. The forecasting performance of MLP also diminished through forecasting steps. However, LSTM still depicted bad forecasting performance, which led to performance reduction with increase in forecasting steps.

From Figure 22, we can deduce that the LSTM model had the worst forecasting performance, while ARIMA had the best. For machine learning methods, the forecasts of Linear-SVR were more robust than that of MLP.

In *Figure 23*, the RMSE of the forecast was compared to different models. The MLP model meshed well with the differenced data (RMSE=0.66), while it indicated an imbalance with the normalized data (RMSE=4.94). The LSTM model showed substantially good alignment with the normalized data (RMSE=0.96), while it skewed with raw data (RMSE=11.39). Linear SVR model meshed well with both the differenced data and Normalized+differenced data (RMSE=0.38), while it depicted a

misalignment with normalized data (RMSE=4.27). ARIMA model was suitable for both the raw data and the normalized data (RMSE=0.67 and 0.62).

In summary, ARIMA model was considered the 'magnum opus' for all kinds of data preprocessing, followed by the linear SVR model, and the MLP model. Vanilla LSTM showed inadequate performance.

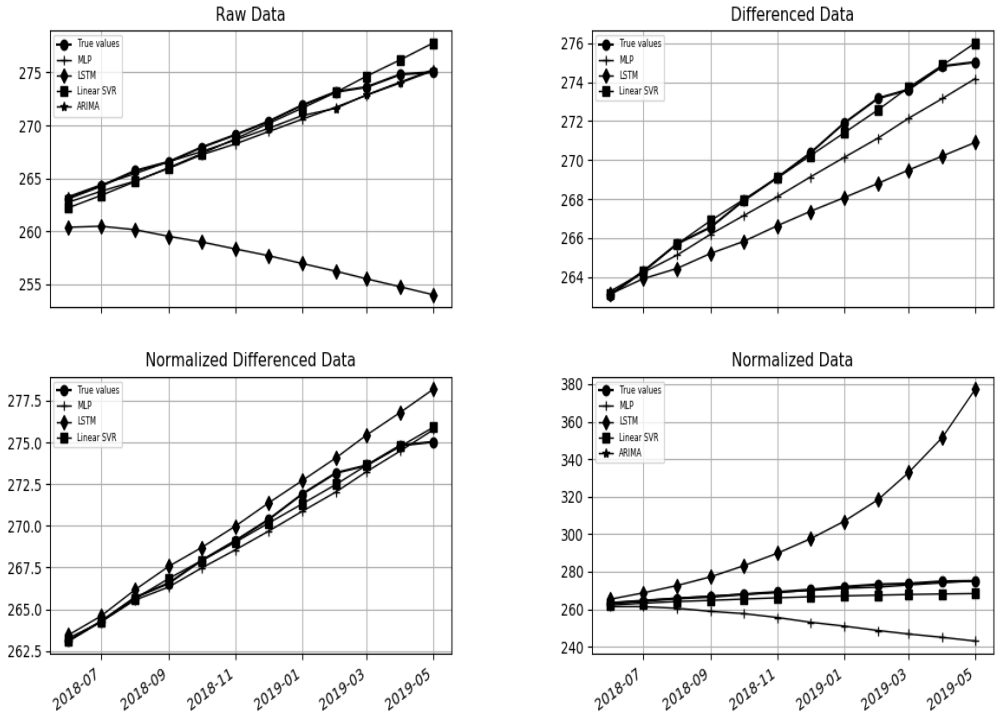

Figure 22 Comparison of prediction by different methods with the same data preprocessing

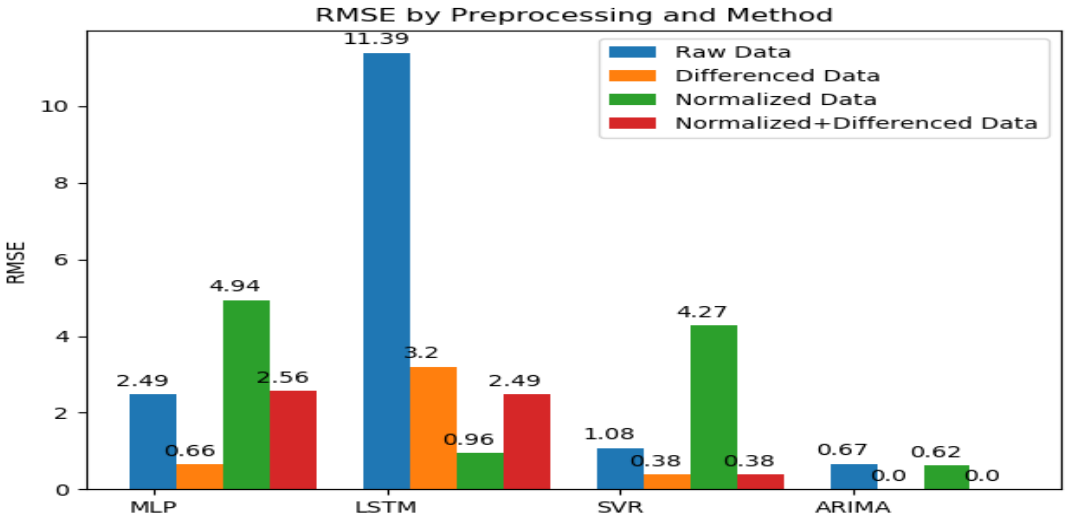

Figure 23 RMSE of forecast compared by different models

5.4.2 Multivariate

Figure 24 compares RMSE that was calculated from different data preprocessing methods separately. The first subplot compared the RMSE of MLP by different data preprocesses from May 2018 to May 2019. The Differenced+normalized data provided the smallest RMSE, followed by Differenced data and normalized data. Their RMSE increased rapidly with increase in forecasting steps. The RMSE of raw data was the highest. The second subplot indicates the RMSE of LSTM by different data preprocesses from May 2018 to May 2019. Here, the Differenced+normalized data provided the smallest RMSE, followed by Differenced data and normalized data. The RMSE of raw data was the highest and its RMSE rose rapidly with increase in forecasting steps. The third subplot showed the RMSE of rbf SVR by different data preprocessing from May 2018 to May 2019. The RMSE of differenced data and normalized+differenced data was negligible. However, the RMSE of raw data and normalized data was quite significant. It meant that forecasting accuracy of rbf SVR calculated from raw data and normalized data showed poor results. The last subplot showed the RMSE of the VEC-VAR model by different data preprocesses from May 2018 to May 2019. The RMSE of all types of data was negligible.

Then, *Figure 25* compares the RMSE of different methods with four types of data preprocesses separately. The first subplot compared the RMSE of MLP, VEC, LSTM, and rbf SVR from May 2018 to May 2019, calculated by row data. Here, the VEC and LSTMs RMSE were the smallest, followed by MLP method, while the RMSE of rbf SVR was large. The second subplot compared the RMSE of MLP, LSTM, and rbf SVR from May 2018 to May 2019, which were calculated by differenced data. This subplot illustrated that the RMSE of three machine learning methods rose with increase in forecasting steps. The RMSE of MLP was the smallest, followed by LSTM method, while the RMSE of rbf SVR was the largest again. The third subplot compared the RMSE of MLP, LSTM, and rbf SVR from May 2018 to May 2019, calculated by differenced+normalized data. This subplot illustrated that the RMSE of three machine learning methods scaled upwards with increase in forecasting steps. The RMSE of MLP and LSTM were quite small. The RMSE of rbf SVR increased rapidly through forecasting steps. The last subplot compares the RMSE of MLP, LSTM, rbf SVR and VEC from May 2018 to May 2019, calculated by normalized data. RMSE of VEC was the smallest, followed by MLP and LSTM methods, while the RMSE of rbf SVR was large.

Figure 26 compares the mean RMSE through 12 forecasting steps by different methods. To be precise, MLP model fitted the normalized+differenced data well (RMSE=2.72), followed by normalized data (RMSE=7.56) and differenced data (RMSE=8.85), while it fited the raw data worst (RMSE=20.24). LSTM model meshed with differenced+normalized data well (RMSE=4.62), followed by differenced data (RMSE=6.95) and normalized data (RMSE=8.26), while it was inept with raw data (RMSE=14.53). rbf SVR model meshed with differenced data well, with RMSE equivalent to 9.45, followed by normalized +differenced data (RMSE=19.9), while it fitted the raw data and normalized data poorly (RMSE=177.72, and 175.85). VEC model showed proper compatibility with both the raw and normalized data (RMSE=3.0).

To summarize, MLP was best suited with Normalized+differenced data, with the smallest mean of RMSE, while it showed an upward trend through forecasting steps. It meant that the forecasting accuracy of MLP calculated from Normalized+differenced data would decrease with an increase in forecasting steps. The VEC model performed well in all data preprocesses. In addition, the forecasting accuracy did not indicate a downward trend by increasing forecasting steps. In other words, the VEC model showed robust performance in forecasts. Contrary to the lousy forecasting performance of Vanilla LSTM for univariate analysis, Encoder-Decoder LSTM performed well in multivariate time series forecasting. Although linear SVR performed well in forecasting univariate time series data, rbf SVR still showed poor forecasting performance in multivariate time series.

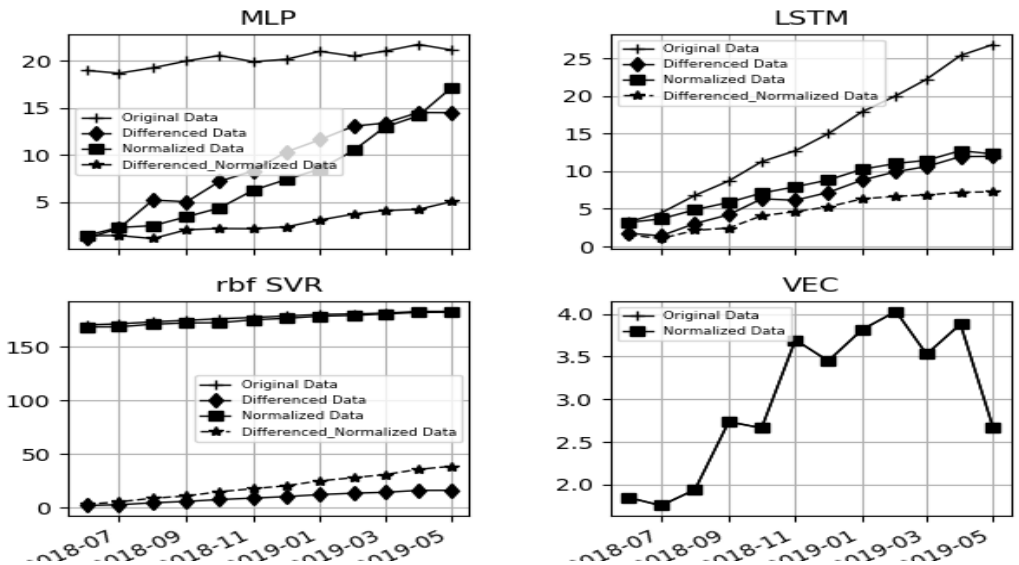

Figure 24 Comparison of RMSE by different data (preprocessing) transformations

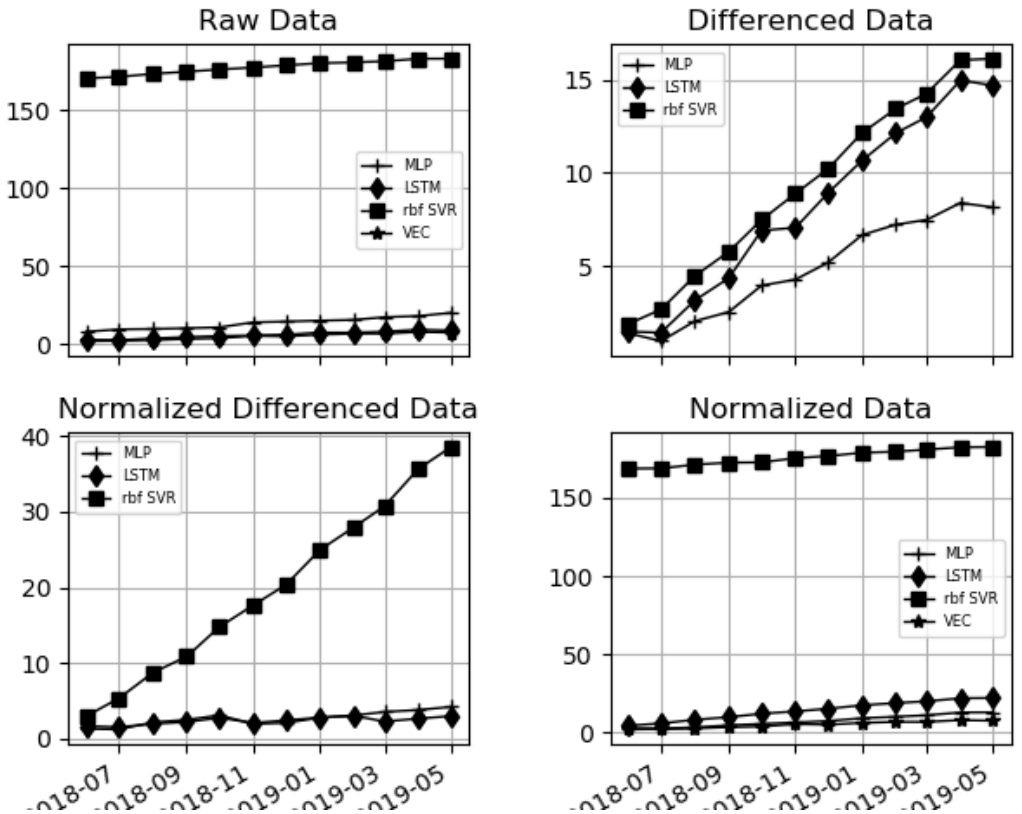

Figure 25 comparison of the RMSE by different methods

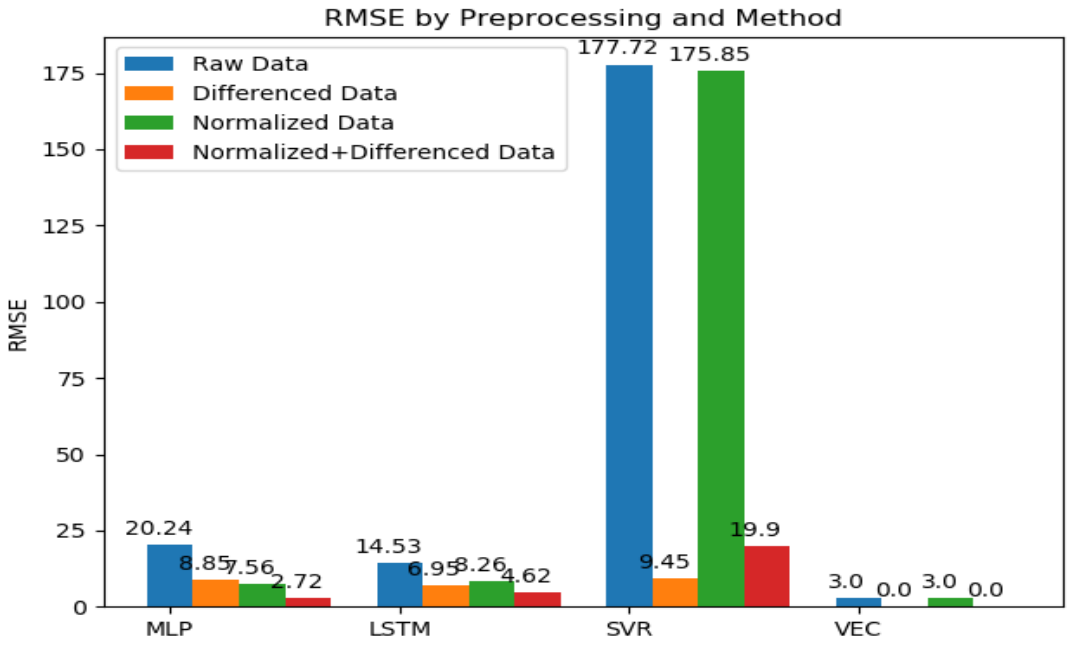

Figure 26 Comparison of mean RMSE by different methods

6. Discussion

This Discussion is divided into four parts. The first part called “What was found in this thesis” shall conclude the results of the observations. The second part called “Results as Expected” shall point out the results that have been found to be consistent with previous research. The third part called “Unexpected Findings” shall point out the results which contradict previous research. The final part called “Explanations for Unexpected Findings” shall proposes possible reasons corresponding to the unexpected findings listed in the third part.

6.1 What was found in this thesis

Statistical models provide the most robust forecasting performance for both univariate and multivariate time series analysis. The forecasting accuracy of statistical models did not show significant change on applying different types of data preprocessing. Besides, there was no significant decrease in forecasting accuracy as a result of an increase in the forecasting steps.

MLP models also performed well in both univariate and multivariate time series forecasting. However, there was a significant decrease of forecasting accuracy due to the increase in forecasting steps, especially for the multivariate analysis. To be more precise, for univariate forecasting, the accuracy was lower than that of ARIMA. For the multivariate forecasting, the MLP that was trained by the normalized+differenced data depicted better forecasting performance than the VAR model. However, the forecasting accuracy showed a steady decrease with the increase in the forecasting steps. Therefore, MLP was found to be unsuitable for long period forecasting.

The forecasting accuracy of SVR changes dramatically due to the different types of data preprocessing. In short, we used Linear SVR to train the univariate data and rbf SVR to train multivariate data. Linear SVR performed well in univariate time series forecasting. However, rbf SVR indicated a poor forecasting performance during multivariate analysis. The forecasting accuracy can be improved dramatically by the application of differencing data. Therefore, we need to experiment using different data preprocessing methods to achieve better forecasting accuracy when fitting SVR.

In this thesis, Vanilla LSTM had been applied in univariate time series analysis, while Encoder-Decoder LSTM had been applied in multivariate time series forecasting.

Vanilla LSTM has only one single hidden layer, but Encoder-Decoder LSTM has a more complex structure. Vanilla LSTM showed poor forecasting performance. Also, the forecasting accuracy dipped quickly when the forecasting steps were increased. Encoder-Decoder LSTM indicated a much better performance in multivariate forecasting. Therefore, the forecasting accuracy of LSTM would be impacted by their structures. However, it would be challenging to figure out a suitable structure.

6.2 Results as Expected

ARIMA performed better than machine learning methods in univariate time series forecasting, which is consistent with the conclusions of Makridakis et al. (2018). Likewise, MLP performed best in univariate time series forecasting through three machine learning methods，which was consistent with the conclusions of Makridakis et al. (2018) and Ahmed et al. (2010).

6.3 Unexpected Findings

For LSTM methods, the forecasting accuracy of LSTM is quite low in the univariate time series, while LSTM performed well in multivariate time series forecasting, which indicates a self-contradiction. The existence of a contradiction was also identified between previous researches. Fu et al. (2016) discovered that LSTM performed better than statistical time series method, while Makridakis et al. (2018) argued that machine learning methods showed a deplorable performance when compared to statistical methods in time series forecasting. The forecasting performance of LSTM was found to be worse than the forecasting performance of other machine learning methods.

For SVR methods, the forecasting performance of Linear SVR was quite excellent in analyzing the univariate time series, while rbf SVR performed poorly in multivariate time series forecasting, which also indicated a self-contradiction. Having reviewed previous research, Miiller et al. (1997), Mu et al. (2014) and Makridakis et al. (2018) agreed with the view that SVR performed well in forecasting time series data. However, Ahmed et al. (2010) discovered a terrible forecasting performance by the SVR method.

6.4 Explanations for Unexpected Findings

For SVR methods, in the Results section, we identified that, firstly, the MSE of Linear SVR is much lower than the MSE for the rbf SVR. It means the rbf SVR model is not compatible with the data. We could identify a more suitable SVR model which is better suited to the data, and the suitable rbf SVR would have a good forecasting performance. This indicated that a proper process of model selection is essential for rbf SVR methods. Furthermore, rbf SVR's forecasting performance improved dramatically by differencing data in both univariate forecasting and the multivariate forecasting. Zhang and Qi (2005) claimed that without appropriate preprocessing, Machine learning methods might become unstable and yield suboptimal results. The result shows that data preprocessing influenced rbf SVR to a great extent. Therefore, two reasons have been found to be the cause for the rbf SVR's poor forecasting accuracy in multivariate analysis. First, the model selection of rbf SVR that was employed for this thesis was not accurate. Second, the data preprocessing used in this thesis was not suitable for the rbf SVR.

For the LSTM methods in this thesis, the forecasting performance of univariate time series was found to be worse than that of MLP, SVR, and ARIMA, which is consistent with the conclusion from a previous study (Makridakis et al., 2018). However, it is worth noting that, in this thesis, a Vanilla LSTM was applied in univariate time series analysis, while an Encoder-Decoder LSTM was applied in multivariate time series forecasting. The vanilla LSTM is the simplest LSTM, with only one single hidden layer. Besides, the LSTM that Makridakis et al. (2018) trained also contained only a single hidden layer. However, Encoder-Decoder LSTM has a more complex structure. Therefore, a different structure of LSTM may lead to different forecasting accuracy. To be more exact, LSTM with complex structure may provide a higher forecasting accuracy.

In conclusion, firstly, a proper process of model selection is essential for rbf SVR. Secondly, proper data preprocessing is essential when fitting rbf SVR. Finally, LSTM with complex structure may provide a higher forecasting accuracy.

7. Conclusions and Recommendations

7.1 Conclusions

This thesis focuses on comparing the forecasting performance of classical statistical methods and machine learning methods. For the univariate time series, we compared MLP, LSTM, and linear SVR with ARIMA. For the multivariate time series, we compared MLP, LSTM, and rbf SVR with VAR. Following the review process, two objectives have been listed in the Introduction. First, we hope to extend the previous research by comparing the forecasting performance of ML and statistical methods for multivariate time series data. Futhermore, we also hope to identify the reason behind the different opinions held by researchers on the forecasting performance of ML methods. Therefore, this chapter will present the conclusion regarding Univariate comparison, Multivariate comparison, and possible reasons for causing controversial ML methods' forecasting performance separately. Following this, we shall provide some recommendations for improving the forecasting accuracy. Finally, we shall provide a few recommendations for further research.

7.1.1 Univariate comparison

The ARIMA model provided a robust forecasting performance due to the lack of a dramatic difference of RMSE calculated from different sets of data. In other words, the data preprocessing did not impact the forecasting accuracy of ARIMA. Besides, no significant decrease in forecasting accuracy was identified due to the increase of forecasting steps, followed by linear SVR and the MLP. LSTM showed a poor forecasting performance since it possessed a higher RMSE than the other three methods. Also, the forecasting accuracy dropped quickly with the increase in forecasting steps.

7.1.2 Multivariate comparison

The VAR model provided the most robust forecasting performance owing to the lack of a dramatic difference of RMSE, which was calculated using different sets of data. In other words, the data preprocessing did not impact the forecasting accuracy of VAR. Besides, there was no significant decrease in forecasting accuracy by the increase of forecasting steps, followed by MLP and LSTM. However, SVR showed poor

forecasting performance since the RMSE of SVR is higher than other methods, and also, because the RMSE of raw data and normalized data are much higher than RMSEs calculated using differenced data and differenced+normalized data.

7.1.3 Reasons for causing controversy ML methods' forecasting performance.

Some machine learning methods are not suitable for a long period forecasting. In other words, the forecasting accuracy decreases with the increase in forecasting steps for certain machine learning methods. For example, as seen in the Results, the forecasting accuracy of MLP trained by differenced+normalized data showed a significant decrease with the increase in forecasting steps.

For certain machine learning methods, especially neural networks, the structure of the network impacts the forecasting performance to a large extent. However, it would be challenging to figure out a suitable structure. For example, as mentioned in the Discussion chapter, different structures of LSTM may lead to different forecasting accuracy. To be more exact, LSTM with complex structures may provide higher forecasting accuracy.

For certain machine learning methods, the model selection can be a difficult process. For example, as mentioned in the Discussion chapter, the rbf SVR did not fit the data adequately, in this thesis. Therefore, the forecasting performance was found to be reduced.

A proper data preprocessing is essential for some machine learning methods. For example, from the Results section, we identified that data preprocessing greatly influenced the rbf SVR's forecasting performance. However, choosing a proper data preprocessing is not easy in some cases.

7.2 Recommendations

According to the results, discussions, conclusions, we provided recommendations that might improve the forecasting accuracy when forecasting time series data. First, for fitting time series, statistical methods are a better choice than machine learning methods. Second, for time series analysis, MLP is an adept choice for machine learning methods. Third, when fitting time series data with LSTM, a complex structure

tends to provide better results. Fourth, careful model selection and data preprocessing are required when fitting time series data with SVR.

7.3 Further areas of researches

This thesis does not involve complex time series problems. For example, we provided comparisons only for VAR, MLP, LSTM, and SVR in the multivariate time series analysis. There are still a large number of multivariate time series models and machine learning models. Future research can focus on comparing more multivariate time series models with machine learning methods. Secondly, an increased number of types of data preprocessing can be involved in the future. Besides, this thesis does not look into the volatility of the data. Finally, for machine learning methods, it is necessary to focus on model selection. Besides, the computational costs could also be evaluated in the future.

Reference

Ahmed, N., Atiya, A., Gayar, N. and El-Shishiny, H. (2010). An Empirical Comparison of Machine Learning Models for Time Series Forecasting. *Econometric Reviews*, 29(5-6), pp.594-621.

Bzdok, D., Altman, N. and Krzywinski, M. (2018). Statistics versus machine learning. *Nature Methods*, 15(4), pp.233-234.

Chollet, F. (2018). *Deep learning with Python*. Shelter Islands: Manning, pp.9-11,15-16.

Crawford, G. and Fratantoni, M. (2003). Assessing the Forecasting Performance of Regime-Switching, ARIMA and GARCH Models of House Prices. *Real Estate Economics*, 31(2), pp.223-243.

EAPPRAISE. (2000). *eAppraise | residential real estate appraisal valuation*. [online] Available at: http://e-appraise.com/ [Accessed 29 Feb. 2000].

Ediger, V. and Akar, S. (2007). ARIMA forecasting of primary energy demand by fuel in Turkey. *Energy Policy*, 35(3), pp.1701-1708.

Fhfa.gov. (2008). *Home | Federal Housing Finance Agency*. [online] Available at: https://www.fhfa.gov/ [Accessed 13 Sep. 2019].

Fhfa.gov. (2014). *House Price Index | Federal Housing Finance Agency*. [online] Available at: https://www.fhfa.gov/DataTools/Downloads/Pages/House-Price-Index.aspx [Accessed 25 Apr. 2014].

Fhfa.gov. (2014). *House Price Index Datasets | Federal Housing Finance Agency*. [online] Available at: https://www.fhfa.gov/DataTools/Downloads/Pages/House-Price-Index-Datasets.aspx#mpo [Accessed 27 Jun. 2019].

Fu, R., Zhang, Z. and Li, L. (2016). Using LSTM and GRU neural network methods for traffic flow prediction. *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*.

Gupta, R. and Miller, S. (2010). The Time-Series Properties of House Prices: A Case Study of the Southern California Market. *The Journal of Real Estate Finance and Economics*, 44(3), pp.339-361.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), pp.1735-1780.

Hosking, J. (1981). Equivalent Forms of the Multivariate Portmanteau Statistic. *Journal of the Royal Statistical Society: Series B (Methodological)*, 43(2), pp.261-262.

Hui, E. and Yue, S. (2006). Housing Price Bubbles in Hong Kong, Beijing and Shanghai: A Comparative Study. *The Journal of Real Estate Finance and Economics*, 33(4), pp.299-327.

Jonbaer.com. (2013). *Jon Baer: Archive*. [online] Available at: http://jonbaer.com/archive/2013/5?cv=1&before_time=1364372989 [Accessed 9 Aug. 2019].

Li, W. and McLeod, A. (1981). Distribution of the Residual Autocorrelations in Multivariate Arma Time Series Models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 43(2), pp.231-239.

Ljung, G. and Box, G. (1978). On a Measure of Lack of Fit in Time Series Models. *Biometrika*, 65(2), p.297.

Makridakis, S., Spiliotis, E. and Assimakopoulos, V. (2018). Statistical and Machine Learning forecasting methods: Concerns and ways forward. *PLOS ONE*, 13(3), p.e0194889.

Mu, J., Wu, F. and Zhang, A. (2014). Housing Value Forecasting Based on Machine Learning Methods. *Abstract and Applied Analysis*, 2014, pp.1-7.

Müller, K., Smola, A., Rätsch, G., Schölkopf, B., Kohlmorgen, J. and Vapnik, V. (1997). Predicting time series with support vector machines. *Lecture Notes in Computer Science*, pp.999-1004.

Stergiou, K., Christou, E. and Petrakis, G. (1997). Modelling and forecasting monthly fisheries catches: comparison of regression, univariate and multivariate time series methods. *Fisheries Research*, 29(1), pp.55-95.

Tsay, R. (2010). *Analysis of financial time series*. 3rd ed. Hoboken: John Wiley & Sons, pp.390-393.

Zhang, G. and Qi, M. (2005). Neural network forecasting for seasonal and trend time series. *European Journal of Operational Research*, 160(2), pp.501-514.

Appendices

I R Code

```
#######univariate
library(hydroGOF)
library(urca)
library(tseries)
library(fGarch)
library(rugarch)#garch
library(TSA)#
library(ggplot2)#
#library(ccgarch)#JB
library(quantmod)
library(zoo)
library(forecast)
library(zoo)
library(forecast)
```

```
data<-read.table("C:/Users/Administrator/Desktop/dissertation/HPI_PO_monthly_hist.csv", sep=",", header=T)
ts.data <- ts(as.matrix(data[,c(2:11)]), start=c(1990,1), frequency=12)
datafit<-data.frame(data[1:329,])
ts.datafit <- ts(as.matrix(datafit[,c(2:11)]), start=c(1990,1), frequency=12)
ussafit<-ts(datafit$USA..,frequency = 12,start=c(1991,1))
par(mfrow=c(1,1))
plot(ussafit,ylab='USA House Price')

first_order_difference<-diff(diff(ussafit),12)
par(mfrow=c(3,1))
plot(first_order_difference)
acf(first_order_difference)
pacf(first_order_difference)# 不通过，长期在零轴一边，具有单调趋势。
main

second_order_difference<-diff(first_order_difference)
summary(ur.df(ussafit,type = 'trend',selectlags = 'AIC'))#
summary(ur.df(ussafit,type ='drift' ,selectlags = 'AIC'))#
summary(ur.df(ussafit,type = 'none',selectlags = 'AIC'))#
par(mfrow=c(3,1))
plot(second_order_difference)
acf(second_order_difference)#
pacf(second_order_difference)
#Q TEST
for(i in 1:6) print(Box.test(ussafit.dif2 ,type="Ljung-Box",lag=2*i))#

#autoarima
arimasafit<-auto.arima(ussafit)
par(mfrow=c(1,1))
title(main = "LM-Test")
lmresult=McLeod.Li.test(y=residuals(arimasafit))

#fit
arimasa.fit<-arima(ussafit,order=c(0,2,1),seasonal=list(order=c(0,0,2),period=12))
#diagnose
acf(arimasa.fit$residual)
tsdiag(arimasa.fit)
cpgram(arimasa.fit$residuals)
```

```
#forecast
par(mfrow=c(1,1))
arimasa.fitfore <-forecast(arimasa.fit ,h=12)
plot(arimasa.fitfore)
pre<-arimasa.fitfore$mean
test<-ts.data[330:341]
sqrt(mse(sim = pre, obs = test, weights = 1, na.rm = FALSE))

#normalized
myNormalize  <- function (target) {
   (target - min(target))/(max(target) - min(target))}
nomorized<-myNormalize(ussafit)
arimasafitn<-auto.arima(nomorized)
arimasa.fitforen <-forecast(arimasafitn ,h=12)
plot(arimasa.fitforen)
arimasa.fitforen$mean
#######multivariate
library(ggplot2)
library(reshape2)
library(psych)
library(vars)
library(xts)
library(MTS)
library(hydroGOF)
#insert data
data<-read.table("C:/Users/Administrator/Desktop/dissertation/HPI_PO_mont
hly_hist.csv",  sep=",", header=T)

#describe data
describe(data[,2:11])
data<-data.frame(data[,-1])
ts.data <- ts(as.matrix(data), start=c(1990,1), frequency=12)
#difference
dif1<-diff(ts.data,12)
dif2<-diff(dif1)
dif1fit<-dif1[1:317,]
dif2fit<-diff(dif1fit)
dif1test<-dif1[318:329,]
dif2test<-diff(dif1test)
```

```
datafit<-data.frame(data[1:329,])
datatest<-data.frame(data[330:341,])
ts.datafit <- ts(as.matrix(datafit), start=c(1990,1), frequency=12)
ts.datatest <- ts(as.matrix(datatest), start=c(2018,6), frequency=12)

par(mfrow=c(3,3))
for(i in 1:9){ts.plot(ts.data[,i],col=1,xlab="Time",ylab=names(datafit)[i+1])}
par(mfrow=c(3,3))
for(i in 1:9){ts.plot(dif1[,i],col=1,xlab="Time",ylab=names(datafit)[i+1])}
par(mfrow=c(3,3))
for(i in 1:9){ts.plot(dif2[,i],col=1,xlab="Time",ylab=names(datafit)[i+1])}

####ADF TEST
summary(ur.df(datafit[,1],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(datafit[,2],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(datafit[,3],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(datafit[,4],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(datafit[,5],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(datafit[,6],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(datafit[,7],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(datafit[,8],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(datafit[,9],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(datafit[,10],type = 'trend',selectlags = 'AIC'))#

summary(ur.df(datafit[,1],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(datafit[,2],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(datafit[,3],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(datafit[,4],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(datafit[,5],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(datafit[,6],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(datafit[,7],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(datafit[,8],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(datafit[,9],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(datafit[,10],type ='drift',selectlags = 'AIC'))#

summary(ur.df(datafit[,1],type = 'none',selectlags = 'AIC'))#
summary(ur.df(datafit[,2],type = 'none',selectlags = 'AIC'))#
summary(ur.df(datafit[,3],type = 'none',selectlags = 'AIC'))#
summary(ur.df(datafit[,4],type = 'none',selectlags = 'AIC'))#
```

```
summary(ur.df(datafit[,5],type = 'none',selectlags = 'AIC'))#
summary(ur.df(datafit[,6],type = 'none',selectlags = 'AIC'))#
summary(ur.df(datafit[,7],type = 'none',selectlags = 'AIC'))#
summary(ur.df(datafit[,8],type = 'none',selectlags = 'AIC'))#
summary(ur.df(datafit[,9],type = 'none',selectlags = 'AIC'))#
summary(ur.df(datafit[,10],type ='none',selectlags = 'AIC'))#

summary(ur.df(dif1[,1],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(dif1[,2],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(dif1[,3],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(dif1[,4],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(dif1[,5],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(dif1[,6],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(dif1[,7],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(dif1[,8],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(dif1[,9],type = 'trend',selectlags = 'AIC'))#
summary(ur.df(dif1[,10],type = 'trend',selectlags = 'AIC'))#

summary(ur.df(dif1[,1],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(dif1[,2],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(dif1[,3],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(dif1[,4],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(dif1[,5],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(dif1[,6],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(dif1[,7],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(dif1[,8],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(dif1[,9],type = 'drift',selectlags = 'AIC'))#
summary(ur.df(dif1[,10],type ='drift',selectlags = 'AIC'))#

summary(ur.df(dif1[,1],type = 'none',selectlags = 'AIC'))#
summary(ur.df(dif1[,2],type = 'none',selectlags = 'AIC'))#
summary(ur.df(dif1[,3],type = 'none',selectlags = 'AIC'))#
summary(ur.df(dif1[,4],type = 'none',selectlags = 'AIC'))#
summary(ur.df(dif1[,5],type = 'none',selectlags = 'AIC'))#
summary(ur.df(dif1[,6],type = 'none',selectlags = 'AIC'))#
summary(ur.df(dif1[,7],type = 'none',selectlags = 'AIC'))#
summary(ur.df(dif1[,8],type = 'none',selectlags = 'AIC'))#
summary(ur.df(dif1[,9],type = 'none',selectlags = 'AIC'))#
summary(ur.df(dif1[,10],type ='none',selectlags = 'AIC'))#
```

```
summary(ur.df(dif2[,1],type = 'trend',selectlags = 'AIC'))
summary(ur.df(dif2[,2],type = 'trend',selectlags = 'AIC'))
summary(ur.df(dif2[,3],type = 'trend',selectlags = 'AIC'))
summary(ur.df(dif2[,4],type = 'trend',selectlags = 'AIC'))
summary(ur.df(dif2[,5],type = 'trend',selectlags = 'AIC'))
summary(ur.df(dif2[,6],type = 'trend',selectlags = 'AIC'))
summary(ur.df(dif2[,7],type = 'trend',selectlags = 'AIC'))
summary(ur.df(dif2[,8],type = 'trend',selectlags = 'AIC'))
summary(ur.df(dif2[,9],type = 'trend',selectlags = 'AIC'))
summary(ur.df(dif2[,10],type = 'trend',selectlags = 'AIC'))

summary(ur.df(dif2[,1],type = 'drift',selectlags = 'AIC'))
summary(ur.df(dif2[,2],type = 'drift',selectlags = 'AIC'))
summary(ur.df(dif2[,3],type = 'drift',selectlags = 'AIC'))
summary(ur.df(dif2[,4],type = 'drift',selectlags = 'AIC'))
summary(ur.df(dif2[,5],type = 'drift',selectlags = 'AIC'))
summary(ur.df(dif2[,6],type = 'drift',selectlags = 'AIC'))
summary(ur.df(dif2[,7],type = 'drift',selectlags = 'AIC'))
summary(ur.df(dif2[,8],type = 'drift',selectlags = 'AIC'))
summary(ur.df(dif2[,9],type = 'drift',selectlags = 'AIC'))
summary(ur.df(dif2[,10],type = 'drift',selectlags = 'AIC'))

summary(ur.df(dif2[,1],type = 'none',selectlags = 'AIC'))
summary(ur.df(dif2[,2],type = 'none',selectlags = 'AIC'))
summary(ur.df(dif2[,3],type = 'none',selectlags = 'AIC'))
summary(ur.df(dif2[,4],type = 'none',selectlags = 'AIC'))
summary(ur.df(dif2[,5],type = 'none',selectlags = 'AIC'))
summary(ur.df(dif2[,6],type = 'none',selectlags = 'AIC'))
summary(ur.df(dif2[,7],type = 'none',selectlags = 'AIC'))
summary(ur.df(dif2[,8],type = 'none',selectlags = 'AIC'))
summary(ur.df(dif2[,9],type = 'none',selectlags = 'AIC'))
summary(ur.df(dif2[,10],type = 'none',selectlags = 'AIC'))#CI(2)
#JOHANSON TEST
m1<-VARorder(datafit,12)#13，2，2
m2<-ca.jo(datafit,type='eigen',K=11,season = 12)#8
m3<-ca.jo(datafit,type='trace',K=11,season= 12)#8
summary(m2)###r=8
w1<-c(1,0.02769659, 1.10610909, 0.48763449, 0.05872420, 0.68682660,
0.66192377, 0.22956632, 0.85202272, -5.20082171)
w2<-c(1,40.31703, -83.02178, -103.30168, -26.11436, -154.99007,
```

```
-105.67596, -305.32835, -67.94345, 896.12987)
w3<-c(1, -0.1537248, 0.6018490, 0.4830768, 0.1880556, 0.4520209,
-0.2480363, -0.2416094, 0.7604907, -2.9015840)
w4<-c(1, 0.3750629, 0.4488994, 0.4843448, 0.7974734, 0.9271606,
0.6787607, 0.3223989, 0.9061719, -6.0832076)
w5<-c(1, -0.3566378, 0.4596188, 0.1549881, 0.1713201, 0.1506104,
0.8611400, -0.4117347, 0.7105385, -2.6419417)
w6<-c(1,-1.24872025, 1.30639682, -0.74142197, -1.73800569, 0.26391902,
0.79025312, 1.46050205, 0.08911421, -0.64236208)
w7<-c(1, 9.573192, 9.146137, -10.789400, -26.829922, -5.315670,
-55.005456, -41.783355, -15.718067, 146.836593)
w8<-c(1,-0.1491611, -0.5986173, -0.4758242, 1.8074373, 2.4166762,
9.7773742, 5.3474278, 3.2945527, -23.0553487)
x1 <- as.matrix(datafit) %*%w1
x2 <- as.matrix(datafit) %*%w2
x3 <- as.matrix(datafit) %*%w3
x4 <- as.matrix(datafit) %*%w4
x5 <- as.matrix(datafit) %*%w5
x6 <- as.matrix(datafit) %*%w6
x7 <- as.matrix(datafit) %*%w7
x8 <- as.matrix(datafit) %*%w8

summary(ur.df(x1,type = 'trend',selectlags = 'AIC'))
summary(ur.df(x2,type = 'trend',selectlags = 'AIC'))
summary(ur.df(x3,type = 'trend',selectlags = 'AIC'))
summary(ur.df(x4,type = 'trend',selectlags = 'AIC'))
summary(ur.df(x5,type = 'trend',selectlags = 'AIC'))#
summary(ur.df(x6,type = 'trend',selectlags = 'AIC'))#
summary(ur.df(x7,type = 'trend',selectlags = 'AIC'))
summary(ur.df(x8,type = 'trend',selectlags = 'AIC'))#

summary(ur.df(x1,type = 'drift',selectlags = 'AIC'))
summary(ur.df(x2,type = 'drift',selectlags = 'AIC'))
summary(ur.df(x3,type = 'drift',selectlags = 'AIC'))
summary(ur.df(x4,type = 'drift',selectlags = 'AIC'))
summary(ur.df(x5,type = 'drift',selectlags = 'AIC'))#
summary(ur.df(x6,type = 'drift',selectlags = 'AIC'))#
summary(ur.df(x7,type = 'drift',selectlags = 'AIC'))
summary(ur.df(x8,type = 'drift',selectlags = 'AIC'))#
```

```
summary(m3)

y1<-c(1, 0.02769659, 1.10610909, 0.48763449, 0.05872420, 0.68682660,
0.66192377, 0.22956632, 0.85202272, -5.20082171)
y2<-c(1,40.31703, -83.02178, -103.30168, -26.11436, -154.99007, -105.67596,
-305.32835, -67.94345, 896.12987)
y3<-c(1,-0.1537248, 0.6018490, 0.4830768, 0.1880556, 0.4520209,
-0.2480363, -0.2416094, 0.7604907, -2.9015840)
y4<-c(1,0.3750629, 0.4488994, 0.4843448, 0.7974734, 0.9271606,
0.6787607, 0.3223989, 0.9061719, 0.9061719)
y5<-c(1,-0.3566378, 0.4596188, 0.1549881, 0.1713201, 0.1506104,
0.8611400, -0.4117347, 0.7105385, -2.6419417)
y6<-c(1, -1.24872025, 1.30639682, -0.74142197, -1.73800569, 0.26391902,
0.79025312, 1.46050205, 0.08911421, -0.64236208)
y7<-c(1, 9.573192, 9.146137, -10.789400, -26.829922, -5.315670, -55.005456,
-41.783355, -15.718067, 146.836593)
y8<-c(1,-0.1491611,-0.5986173,-0.4758242,1.8074373,2.4166762,9.7773742,
5.3474278,3.2945527,-23.0553487)
y9<-c(1,-15.428660, 2.042358, -1.862235, -3.093001, -2.369651, 15.046969,
11.085810, 4.756740, -12.379940)
y10<-c(1,-0.05390805, 0.50811435, 0.15356026, -0.11026241, -0.04989142,
-0.22742874, -1.20870610, 0.43740748, -0.12658843)
z1 <- as.matrix(datafit) %*%y1
z2 <- as.matrix(datafit) %*%y2
z3 <- as.matrix(datafit) %*%y3
z4 <- as.matrix(datafit) %*%y4
z5 <- as.matrix(datafit) %*%y5
z6 <- as.matrix(datafit) %*%y6
z7 <- as.matrix(datafit) %*%y7
z8 <- as.matrix(datafit) %*%y8
z9 <- as.matrix(datafit) %*%y9
z10 <- as.matrix(datafit) %*%y10

summary(ur.df(z1,type = 'trend',selectlags = 'AIC'))
summary(ur.df(z2,type = 'trend',selectlags = 'AIC'))
summary(ur.df(z3,type = 'trend',selectlags = 'AIC'))
summary(ur.df(z4,type = 'trend',selectlags = 'AIC'))
summary(ur.df(z5,type = 'trend',selectlags = 'AIC'))#
summary(ur.df(z6,type = 'trend',selectlags = 'AIC'))#
summary(ur.df(z7,type = 'trend',selectlags = 'AIC'))
```

```
summary(ur.df(z8,type = 'trend',selectlags = 'AIC'))#
summary(ur.df(z9,type = 'trend',selectlags = 'AIC'))#
summary(ur.df(z10,type = 'trend',selectlags = 'AIC'))

summary(ur.df(z1,type = 'drift',selectlags = 'AIC'))
summary(ur.df(z2,type = 'drift',selectlags = 'AIC'))
summary(ur.df(z3,type = 'drift',selectlags = 'AIC'))
summary(ur.df(z4,type = 'drift',selectlags = 'AIC'))
summary(ur.df(z5,type = 'drift',selectlags = 'AIC'))#
summary(ur.df(z6,type = 'drift',selectlags = 'AIC'))#
summary(ur.df(z7,type = 'drift',selectlags = 'AIC'))
summary(ur.df(z8,type = 'drift',selectlags = 'AIC'))#
summary(ur.df(z9,type = 'drift',selectlags = 'AIC'))#
summary(ur.df(z10,type = 'drift',selectlags = 'AIC'))

summary(ur.df(z1,type = 'none',selectlags = 'AIC'))
summary(ur.df(z2,type = 'none',selectlags = 'AIC'))#
summary(ur.df(z3,type = 'none',selectlags = 'AIC'))
summary(ur.df(z4,type = 'none',selectlags = 'AIC'))
summary(ur.df(z5,type = 'none',selectlags = 'AIC'))
summary(ur.df(z6,type = 'none',selectlags = 'AIC'))
summary(ur.df(z7,type = 'none',selectlags = 'AIC'))
summary(ur.df(z8,type = 'none',selectlags = 'AIC'))#
summary(ur.df(z9,type = 'none',selectlags = 'AIC'))#
summary(ur.df(z10,type = 'none',selectlags = 'AIC'))
Z=cbind(z1,z2,z3,z4,z5,z6,z7,z8)
par(mfrow=c(4,2))
for(i in 1:8){ts.plot(Z[,i],col=1)}
mm1<-ECMvar1(datafit,12,Z,include.const = TRUE)
MTSdiag(mm1)
m4=vec2var(m3,r=8)
prediction2<-predict(m4,n.ahead = 12)
plot(prediction2,xaxt="n")

write.csv(prediction2$fcst,file="C:\\Users\\Administrator\\Desktop\\dissertation\
\RPRE2.csv",quote=F,row.names = F)
pred2<-read.table("C:\\Users\\Administrator\\Desktop\\dissertation\\RPRE2.cs
v",sep=",",header=T)
rmse<-sqrt(mse(pred2, datatest,na.rm = TRUE))
```

```
#Normalized
myNormalize  <- function  (target) {
   (target - min(target))/(max(target) - min(target))}
datascaled<-myNormalize(datafit)

m4<-VARorder(datascaled)#13，2，2
m5<-ca.jo(datascaled,type='eigen',K=12,season = 12)#8
m6<-ca.jo(datascaled,type='trace',K=12,season = 12)#8
summary(m5)###r=8
summary(m6)#8
vecm<-cajorls(m2,r=8)
model.var5<-vec2var(m5,r=8)
model.var6<-vec2var(m6,r=8)
prediction2<-predict(model.var5,n.ahead  = 12)
plot(prediction2)

write.csv(prediction$fcst,file="C:\\Users\\Administrator\\Desktop\\dissertation\\
SPRE.csv",quote=F,row.names = F)
pred<-read.table("C:\\Users\\Administrator\\Desktop\\dissertation\\SPRE.csv",
sep=",",header=T)
mse(pred, datatest,na.rm = TRUE)
```

II Python Code

```
###Univariant

from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import plot_model
import os,arrow
import numpy as np
import pandas as pd
import csv
import matplotlib.pyplot as plt
import tensorflow as tf
import sklearn
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor
```

```
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
import random
import keras
import math
import sys
import test
from pandas import Series
from sklearn.preprocessing import MinMaxScaler
import math
from math import sqrt
from numpy import mean
from sklearn.svm import SVR
from pandas import Series
from sklearn.metrics import mean_squared_error
from matplotlib.ticker import NullFormatter
from matplotlib.ticker import MaxNLocator
from collections import namedtuple
import datetime
import matplotlib.dates as mdates
import matplotlib.dates as mdate
import matplotlib.cbook as cbook
import ztools as zt
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
pd.set_option('display.width',450)
#pd.set_option('display.float_format')
rlog='/ailib/log_tmp'
#if os.path.exists(rlog):tf.gfile.DeleteRecursively(rlog)
#import    matplotlib.font_manager.FontProperties
from matplotlib import font_manager
#from font_manager import FontProperties
# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# fit a model
def mlp1(dataset,n_steps):
    X, y = split_sequence(dataset, n_steps)
    #X = X.reshape((X.shape[0], n_input))
    # define model
    model = Sequential()
```

```
    model.add(Dense(8, activation='relu', input_dim=n_steps))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse',metrics=['acc'])
    # fit model
    model.summary()
    SVG(model_to_dot(model).create(prog='dot', format='svg'))
    plot_model(model, to_file='mlp1.png',show_shapes=True)
    tbCallBack = keras.callbacks.TensorBoard(log_dir=rlog,write_graph=True,
write_images=True)
    history=model.fit(X, y, epochs=200, batch_size=1, verbose=0,callbacks=[tbCallBack])
    predictionsmlp=[]
    #x_input = x_input.reshape((1, n_input))
    yhat = model.predict(X[-1].reshape((1, n_steps)))
    Xpre=np.append(X[-1,1-n_steps:],yhat)
    for i in range(12):
        predictionsmlp.append(yhat)
        yhat = model.predict( Xpre.reshape(1, n_steps))
        Xpre=np.append(Xpre[1-n_steps:],yhat)
    print(predictionsmlp)
    return model,predictionsmlp

def lstm1(dataset,n_steps_in):
    # covert into input/output
    X, y = split_sequence(dataset, n_steps_in)
    # the dataset knows the number of features, e.g. 2
    X = X.reshape(X.shape[0], 1, X.shape[1])
    # define model
    model = Sequential()
    model.add(LSTM(50, activation='relu', batch_input_shape=(1, X.shape[1], X.shape[2])))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse',metrics=['acc'])
    model.summary()
    SVG(model_to_dot(model).create(prog='dot', format='svg'))
    plot_model (model,to_file='lstm1.png',show_shapes=True)
    tbCallBack = keras.callbacks.TensorBoard(log_dir=rlog,write_graph=True,
write_images=True)
    # fit model
    model.fit(X, y, epochs=100, verbose=0, batch_size=1,callbacks=[tbCallBack])
    predictionslstm=[]
    yhat = model.predict(X[-1].reshape((1, 1, n_steps_in)))
    Xpre=np.append(X[-1],yhat)[-n_steps_in:]
    for i in range(12):
        predictionslstm.append(yhat)
        yhat = model.predict( Xpre.reshape(1, 1, n_steps_in))
        Xpre=np.append(Xpre,yhat)[-n_steps_in:]
    print(predictionslstm)
    return model,predictionslstm

def svrlin(dataset,n_steps_in):
    X, y = split_sequence(dataset, n_steps_in)
    model=SVR(kernel='linear',C=5,epsilon=0.01,cache_size=1000)
    model.fit(X,y)
    #plot_model (model,to_file='tmp/svr1.png')
    predictionssvr=[]
```

```
    yhat = model.predict(X[-1].reshape(1,-1))
    Xpre=np.append(X[-1,1-n_steps_in:],yhat)
    for i in range(12):
        predictionssvr.append(yhat)
        yhat = model.predict( Xpre.reshape(1,-1))
        Xpre=np.append(Xpre[1-n_steps_in:],yhat)
    print(predictionssvr)
    return model ,predictionssvr

######################3data#################################
DATA=pd.read_csv("C:/Users/Administrator/Desktop/dissertation/HPI_PO_monthly_hist.csv",ind
ex_col='Month')
DATA.index = pd.to_datetime(DATA.index)
SA = DATA['USA\n\n']
#标准化
SAtest=SA[-12:]
SAtrain=SA[0:-12]
data = SAtrain.values
TEST=SAtest.values

#################invert#########################################

##########invert difference###############
def invert_diff(data,series):
    predictions=[]
    oldseries=data[-1]
    for i in range(len(series)):
        oldseries=series[i]+oldseries
        predictions.append(oldseries)
    return predictions
###################invert scale#################

def inverse_scale(data,series):
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(data.reshape(-1,1))
    predictions=[]
    for i in range(len(series)):
        yhat=scaler.inverse_transform(series[i].reshape(1,-1))
        predictions.append(yhat)
    return predictions


#########################raw test#################################
premlpr=mlp1(data,6)[1]
presvrr=svrlin(data,6)[1]
prelstmr=lstm1(data,6)[1]
prearimar=[263.2600, 264.3790, 265.4306, 266.5721, 267.5218,268.6163, 269.7053,270.9221,
271.6135, 272.8852, 274.0953, 275.2780]
prearimar=np.array(prearimar)
#################################difference#####################
###################differeice#############################
diff_values = SA[0:-12].diff().fillna(0)
diff_values=diff_values.values
```

```
premlpd=mlp1(diff_values,6)[1]
presvrd=svrlin(diff_values,6)[1]
prelstmd=lstm1(diff_values,6)[1]

premlpd=invert_diff(data,premlpd)
presvrd=invert_diff(data,presvrd)
prelstmd=invert_diff(data,prelstmd)
######################scale#################################
      # fit scaler
scaler = MinMaxScaler(feature_range=(-1, 1))
scaler = scaler.fit(data.reshape(-1,1))
scaled_values = scaler.transform(data.reshape(-1,1))
scaled_values=scaled_values.reshape(len(scaled_values),)

premlps=mlp1(scaled_values,6)[1]
presvrs=svrlin(scaled_values,6)[1]
prelstms=lstm1(scaled_values,6)[1]

premlps=inverse_scale(data,premlps)
presvrs=inverse_scale(data,presvrs)
prelstms=inverse_scale(data,prelstms)
prearimas=[263.4538,  264.7044,  265.8124,  266.9799,  267.9015,268.9548,  269.9854,271.1067,
271.7039, 272.9004, 274.0299, 275.1171]
prearimas=np.array(prearimas)
####################difference+scale#############################
################diff+scale#####################################
scaler1 = MinMaxScaler(feature_range=(-1, 1))
scaler1 = scaler.fit(diff_values.reshape(-1,1))
diff_scaled_values = scaler1.transform(diff_values.reshape(-1,1))
diff_scaled_values=diff_scaled_values.reshape(len(diff_scaled_values),)

premlpds=mlp1(diff_scaled_values,6)[1]
premlpds=inverse_scale(diff_values,premlpds)
premlpds=invert_diff(data,premlpds)

presvrds=svrlin(diff_scaled_values,6)[1]
presvrds=inverse_scale(diff_values,presvrds)
presvrds=invert_diff(data,presvrds)

prelstmds=lstm1(diff_scaled_values,6)[1]
prelstmds=inverse_scale(diff_values,prelstmds)
prelstmds=invert_diff(data,prelstmds)

premlpr=np.array(premlpr).reshape(len(premlpr),)
prelstmr=np.array(prelstmr).reshape(len(prelstmr),)
presvrr=np.array(presvrr).reshape(len(presvrr),)
premlpd=np.array(premlpd).reshape(len(premlpd),)
prelstmd=np.array(prelstmd).reshape(len(prelstmd),)
presvrd=np.array(presvrd).reshape(len(presvrd),)
premlps=np.array(premlps).reshape(len(premlps),)
prelstms=np.array(prelstms).reshape(len(prelstms),)
presvrs=np.array(presvrs).reshape(len(presvrs),)
premlpds=np.array(premlpds).reshape(len(premlpds),)
prelstmds=np.array(prelstmds).reshape(len(prelstmds),)
```

```
presvrds=np.array(presvrds).reshape(len(presvrds),)

###################################rmse#####################
rmsemlpr=[]
rmselstmr=[]
rmsesvrr=[]
rmsemlpd=[]
rmselstmd=[]
rmsesvrd=[]
rmsemlps=[]
rmselstms=[]
rmsesvrs=[]
rmsemlpds=[]
rmselstmds=[]
rmsesvrds=[]
rmsearimas=[]
rmsearimar=[]
rmsemlpr.append(sqrt(mean_squared_error(TEST, premlpr)))
rmselstmr.append(sqrt(mean_squared_error(TEST, prelstmr)))
rmsesvrr.append(sqrt(mean_squared_error(TEST, presvrr)))
rmsemlpd.append(sqrt(mean_squared_error(TEST, premlpd)))
rmselstmd.append(sqrt(mean_squared_error(TEST, prelstmd)))
rmsesvrd.append(sqrt(mean_squared_error(TEST, presvrd)))
rmsemlps.append(sqrt(mean_squared_error(TEST, premlps)))
rmselstms.append(sqrt(mean_squared_error(TEST, prelstms)))
rmsesvrs.append(sqrt(mean_squared_error(TEST, presvrs)))
rmsemlpds.append(sqrt(mean_squared_error(TEST, premlpds)))
rmselstmds.append(sqrt(mean_squared_error(TEST, prelstmds)))
rmsesvrds.append(sqrt(mean_squared_error(TEST, presvrds)))
rmsearimas.append(sqrt(mean_squared_error(TEST, prearimas)))
rmsearimar.append(sqrt(mean_squared_error(TEST, prearimar)))
###################################plot#####################
x=SAtest.index
#x=x.strftime('%b' )

#ax = plt.gca()
fig, ax= plt.subplots() # an empty figure with no axes
##fig.suptitle('12_steps Univarience Prediction Comparation')    # Add a title so we know which it
is
##fig, ax_lst = plt.subplots(2, 2)

months = mdates.MonthLocator()    # every month
monthFmt = mdates.DateFormatter('%b')

plt.subplot(221)
plt.plot(x, SAtest, color="K", linestyle="-", marker="o", linewidth=1.5,label='True values')
plt.plot(x, premlpr, color="K", linestyle="-", marker="+", linewidth=1.0,label='MLP')
plt.plot(x, prelstmr, color="K", linestyle="-", marker="d", linewidth=1.0,label='LSTM')
plt.plot(x, presvrr, color="K", linestyle="-", marker="s", linewidth=1.0,label='Linear SVR')
plt.plot(x, prearimar, color="K", linestyle="-", marker="*", linewidth=1.0,label='ARIMA')
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(monthFmt)
fig.autofmt_xdate()
plt.title('Raw Data')
```

```
plt.legend(prop={'size':6})
plt.grid(True)

plt.subplot(222)
plt.plot(x, SAtest, color="k", linestyle="-", marker="o", linewidth=1.5,label='True values')
plt.plot(x, premlpd, color="K", linestyle="-", marker="+", linewidth=1.0,label='MLP')
plt.plot(x, prelstmd, color="K", linestyle="-", marker="d", linewidth=1.0,label='LSTM')
plt.plot(x, presvrd, color="K", linestyle="-", marker="s", linewidth=1.0,label='Linear SVR')
#plt.plot(x, prearimar, color="K", linestyle="--", marker="*", linewidth=1.0)
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(monthFmt)
fig.autofmt_xdate()
plt.title('Differenced Data')
plt.legend(prop={'size':6})
plt.grid(True)

plt.subplot(223)
plt.plot(x, SAtest, color="k", linestyle="-", marker="o", linewidth=1.5,label='True values')
plt.plot(x, premlpds, color="K", linestyle="-", marker="+", linewidth=1.0,label='MLP')
plt.plot(x, prelstmds, color="K", linestyle="-", marker="d", linewidth=1.0,label='LSTM')
plt.plot(x, presvrds, color="K", linestyle="-", marker="s", linewidth=1.0,label='Linear SVR')
#plt.plot(x, prearimar, color="K", linestyle="--", marker="*", linewidth=1.0)
plt.title('Normalized Differenced Data')
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(monthFmt)
fig.autofmt_xdate()
plt.legend(prop={'size':6})
plt.grid(True)

plt.subplot(224)
plt.plot(x, SAtest, color="k", linestyle="-", marker="o", linewidth=1.5,label='True values')
plt.plot(x, premlps, color="K", linestyle="-", marker="+", linewidth=1.0,label='MLP')
plt.plot(x, prelstms, color="K", linestyle="-", marker="d", linewidth=1.0,label='LSTM')
plt.plot(x, presvrs, color="K", linestyle="-", marker="s", linewidth=1.0,label='Linear SVR')
plt.plot(x, prearimas, color="K", linestyle="-", marker="*", linewidth=1.0,label='ARIMA')
plt.title('Normalized Data')
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(monthFmt)
fig.autofmt_xdate()
plt.legend(prop={'size':6})
plt.grid(True)

plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25, wspace=0.35)
plt.show()

name_list = ['MLP','LSTM','SVR','ARIMA']
raw=[rmsemlpr, rmselstmr, rmsesvrr, rmsearimar]
difference=[rmsemlpd, rmselstmd, rmsesvrd, 0]
normalized=[rmsemlps, rmselstms, rmsesvrs, rmsearimas]
difference_scale_data=[rmsemlpds, rmselstmds, rmsesvrds, 0]

func = lambda x: [y for l in x for y in func(l)] if type(x) is list else [x]

raw=func(raw)
```

```
difference=func(difference)
normalized=func(normalized)
difference_scale_data=func(difference_scale_data)

raw=[round (i,2) for i in raw]
difference=[round (i,2) for i in difference]
normalized=[round (i,2) for i in normalized]
difference_scale_data=[round (i,2) for i in difference_scale_data]

x = np.arange(len(name_list))
total_width, n = 0.8, 4
width = total_width / n
x = x - (total_width - width) / 2

fig, ax = plt.subplots()
x = np.arange(len(name_list))
rects1 = ax.bar(x, raw,    width=width, label='Raw Data')
rects2 = ax.bar(x + width, difference, width=width, label='Differenced Data')
rects3 = ax.bar(x + 2 * width, normalized, width=width, label='Normalized Data')
rects4 = ax.bar(x + 3 * width, difference_scale_data, width=width, label='Normalized+Differenced
Data')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('RMSE')
ax.set_title('RMSE by Preprocessing and Method')
ax.set_xticks(x)
ax.set_xticklabels(name_list)
ax.legend()

def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 4, height),
                    xytext=(0, 3),    # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')
autolabel(rects1)
autolabel(rects2)
autolabel(rects3)
autolabel(rects4)
fig.tight_layout()
plt.show()

#######Multivariate
# multivariate multi-step encoder-decoder mlp example
from numpy import array
from numpy import hstack
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed
```

```
# grid search lstm
from math import sqrt
from numpy import array
from numpy import mean
from pandas import DataFrame
from pandas import concat
from pandas import read_csv
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
import os,arrow
import numpy as np
import pandas as pd
import csv
import matplotlib.pyplot as plt
import tensorflow as tf
import sklearn
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
import random
import keras
import math
import sys
import test
from pandas import Series
from sklearn.preprocessing import MinMaxScaler
import math
from math import sqrt
from numpy import mean
from sklearn.svm import SVR
from pandas import Series
from sklearn.metrics import mean_squared_error
from matplotlib.ticker import NullFormatter
from matplotlib.ticker import MaxNLocator
from collections import namedtuple
import datetime
import matplotlib.dates as mdates
import matplotlib.dates as mdate
import matplotlib.cbook as cbook
#import    matplotlib.font_manager.FontProperties
from matplotlib import font_manager

# split a multivariate sequence into samples
def split_sequences(sequences, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out
        # check if we are beyond the dataset
        if out_end_ix > len(sequences):
```

```
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix:out_end_ix, :]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# fit a model
def mlp1(dataset,n_steps_in, n_steps_out):
    X, y = split_sequences(dataset, n_steps_in, n_steps_out)
    # define model
    n_features = X[0].shape[1]
    n_input = X[1].shape[0] * X[1].shape[1]
    X = X.reshape((X.shape[0], n_input))
    # flatten output
    n_output = y[1].shape[0] * y[1].shape[1]
    y = y.reshape((y.shape[0], n_output))
    # define model
    model = Sequential()
    model.add(Dense(8, activation='relu', input_dim=n_input))
    model.add(Dense(n_output))
    model.compile(optimizer='adam', loss='mse',metrics=['acc'])
    # fit model
    history=model.fit(X, y, epochs=200, batch_size=1, verbose=0)
    predictionsmlp=[]
    #x_input = x_input.reshape((1, n_input))
    yhat = model.predict(X[-1].reshape((1, n_input)))
    Xpre=np.append(X[-1,n_output-n_input:],yhat)
    for i in range(12):
        predictionsmlp.append(yhat)
        yhat = model.predict( Xpre.reshape(1, n_input))
        Xpre=np.append(Xpre[n_output-n_input:],yhat)
    print(predictionsmlp)
    return model,predictionsmlp

def lstm1(dataset,n_steps_in, n_steps_out):
    # covert into input/output
    X, y = split_sequences(dataset, n_steps_in, n_steps_out)
    # the dataset knows the number of features, e.g. 2
    n_features = X[0].shape[1]
    Z=(1-n_steps_in)*n_features
    # define model
    model = Sequential()
    model.add(LSTM(50, activation='relu', input_shape=(n_steps_in, n_features)))
    model.add(RepeatVector(n_steps_out))
    model.add(LSTM(50, activation='relu', return_sequences=True))
    model.add(TimeDistributed(Dense(n_features)))
    model.compile(optimizer='adam', loss='mse',metrics=['acc'])
    # fit model
    model.fit(X, y, epochs=100, verbose=0, batch_size=1)
    predictionslstm=[]
    yhat = model.predict(X[-1].reshape((1, n_steps_in, n_features)))
    Xpre=np.append(X[-1,n_steps_out-n_steps_in:],yhat)
    for i in range(12):
```

```
            predictionslstm.append(yhat)
            yhat = model.predict( Xpre.reshape(1, n_steps_in, n_features))
            Xpre=np.append(Xpre[Z:],yhat)
        print(predictionslstm)
        return model,predictionslstm

def svrrbf(dataset,n_steps_in, n_steps_out):
        X, y = split_sequences(dataset, n_steps_in, n_steps_out)
        # define model
        model=SVR(kernel='rbf',C=20,epsilon=0.01,gamma=0.2, cache_size=1000)
        model.fit(X[1].reshape(X[1].shape[1],X[1].shape[0]), y[1].reshape(X[1].shape[1],))
        predictionssvr=[]
        yhat = model.predict(X[-1].reshape(X[-1].shape[1],X[-1].shape[0]))
        Xpre=np.append(X[-1,n_steps_out-n_steps_in:],yhat)
        for i in range(12):
            predictionssvr.append(yhat)
            yhat = model.predict( Xpre.reshape(X[-1].shape[1],X[-1].shape[0]))
            Xpre=np.append(Xpre[((1-n_steps_in)*X[-1].shape[1]):],yhat)
        print(predictionssvr)
        return model ,predictionssvr
##########invert difference###############
def invert_diff(data,series):
        predictions=[]
        oldseries=data[-1]
        for i in range(len(series)):
            oldseries=series[i]+oldseries
            predictions.append(oldseries)
        return predictions
###################invert difference#################
def inverse_scale(data,series):
        scaler = MinMaxScaler(feature_range=(-1, 1))
        scaler = scaler.fit(data)
        predictions=[]
        for i in range(len(series)):
            yhat=scaler.inverse_transform(series[i].reshape(1,-1))
            predictions.append(yhat)
        return predictions

#data
DATA=pd.read_csv("C:/Users/Administrator/Desktop/dissertation/HPI_PO_monthly_hist.csv",ind
ex_col='Month')
DATA.index = pd.to_datetime(DATA.index)
VECR=pd.read_csv("C:/Users/Administrator/Desktop/dissertation/RPRE.csv")
VECS=pd.read_csv("C:/Users/Administrator/Desktop/dissertation/RPRE.csv")
#split train test
raw_values = DATA.values
TRAIN=raw_values[0:-12]
TEST=raw_values[-12:]
VECR=np.array(VECR)
VECS=np.array(VECS)
premlp=mlp1(TRAIN,6,1)
presvr=svrrbf(TRAIN,6,1)
prelstm=lstm1(TRAIN,6,1)
```

```
############row train test#########################
premlpr=mlp1(TRAIN,6,1)[1]
presvrr=svrrbf(TRAIN,6,1)[1]
prelstmr=lstm1(TRAIN,6,1)[1]

###################differeice#########################
diff_values = DATA[0:-12].diff().fillna(0)
diff_values=diff_values.values

premlpd=mlp1(diff_values,6,1)[1]
presvrd=svrrbf(diff_values,6,1)[1]
prelstmd=lstm1(diff_values,6,1)[1]

premlpd=invert_diff(TRAIN,premlpd)
presvrd=invert_diff(TRAIN,presvrd)
prelstmd=invert_diff(TRAIN,prelstmd)

######################scaled#################################
      # fit scaler
scaler = MinMaxScaler(feature_range=(-1, 1))
scaler = scaler.fit(TRAIN)
scaled_values = scaler.transform(TRAIN)

premlps=mlp1(scaled_values,6,1)[1]
presvrs=svrrbf(scaled_values,6,1)[1]
prelstms=lstm1(scaled_values,6,1)[1]

premlps=inverse_scale(TRAIN,premlps)
presvrs=inverse_scale(TRAIN,presvrs)
prelstms=inverse_scale(TRAIN,prelstms)

###############diff+scale#################################
scaler1 = MinMaxScaler(feature_range=(-1, 1))
scaler1 = scaler.fit(diff_values)
diff_scaled_values = scaler1.transform(diff_values)

premlpds=mlp1(diff_scaled_values,6,1)[1]
premlpds=inverse_scale(diff_values,premlpds)
premlpds=invert_diff(TRAIN,premlpds)

presvrds=svrrbf(diff_scaled_values,6,1)[1]
presvrds=inverse_scale(diff_values,presvrds)
presvrds=invert_diff(TRAIN,presvrds)

prelstmds=lstm1(diff_scaled_values,6,1)[1]
prelstmds=inverse_scale(diff_values,prelstmds)
prelstmds=invert_diff(TRAIN,prelstmds)

########################
rmsemlpr=[]
rmselstmr=[]
rmsesvrr=[]
rmsemlpd=[]
rmselstmd=[]
```

```
rmsesvrd=[]
rmsemlps=[]
rmselstms=[]
rmsesvrs=[]
rmsemlpds=[]
rmselstmds=[]
rmsesvrds=[]
rmsevecr=[]
rmsevecs=[]
for i in range(12):
    rmsemlpr.append(sqrt(mean_squared_error(TEST[i], premlpr[i].reshape(TEST.shape[1],))))
    rmselstmr.append(sqrt(mean_squared_error(TEST[i], prelstmr[i].reshape(TEST.shape[1],))))
    rmsesvrr.append(sqrt(mean_squared_error(TEST[i], presvrr[i].reshape(TEST.shape[1],))))
    rmsemlpd.append(sqrt(mean_squared_error(TEST[i], premlpd[i].reshape(TEST.shape[1],))))
    rmselstmd.append(sqrt(mean_squared_error(TEST[i], prelstmd[i].reshape(TEST.shape[1],))))
    rmsesvrd.append(sqrt(mean_squared_error(TEST[i], presvrd[i].reshape(TEST.shape[1],))))
    rmsemlps.append(sqrt(mean_squared_error(TEST[i], premlps[i].reshape(TEST.shape[1],))))
    rmselstms.append(sqrt(mean_squared_error(TEST[i], prelstms[i].reshape(TEST.shape[1],))))
    rmsesvrs.append(sqrt(mean_squared_error(TEST[i], presvrs[i].reshape(TEST.shape[1],))))
    rmsemlpds.append(sqrt(mean_squared_error(TEST[i],
premlpds[i].reshape(TEST.shape[1],))))
    rmselstmds.append(sqrt(mean_squared_error(TEST[i],
prelstmds[i].reshape(TEST.shape[1],))))
    rmsesvrds.append(sqrt(mean_squared_error(TEST[i], presvrds[i].reshape(TEST.shape[1],))))
    rmsevecr.append(sqrt(mean_squared_error(TEST[i], VECR[i].reshape(TEST.shape[1],))))
    rmsevecs.append(sqrt(mean_squared_error(TEST[i], VECS[i].reshape(TEST.shape[1],))))
##################################uni#############################################
#####################################

x=DATA[-12:].index
#x=x.strftime('%b' )

#ax = plt.gca()
fig, ax= plt.subplots() # an empty figure with no axes
##fig.suptitle('12_steps Univarience Prediction Comparation')    # Add a title so we know which it
is
##fig, ax_lst = plt.subplots(2, 2)

months = mdates.MonthLocator()    # every month
monthFmt = mdates.DateFormatter('%b')

plt.subplot(221)
#plt.plot(x, SAtest, color="K", linestyle="-", marker="o", linewidth=1.5,label='True values')
plt.plot(x, rmsemlpr, color="K", linestyle="-", marker="+", linewidth=1.0,label='MLP')
plt.plot(x, rmselstmr, color="K", linestyle="-", marker="d", linewidth=1.0,label='LSTM')
plt.plot(x, rmsesvrr, color="K", linestyle="-", marker="s", linewidth=1.0,label='rbf SVR')
plt.plot(x, rmsevecr, color="K", linestyle="-", marker="*", linewidth=1.0,label='VEC')
#plt.plot(x, rmsearimar, color="K", linestyle="-", marker="*", linewidth=1.0,label='ARIMA')
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(monthFmt)
fig.autofmt_xdate()
plt.title('Raw Data')
plt.legend(prop={'size':6})
plt.grid(True)
```

```
plt.subplot(222)
#plt.plot(x, SAtest, color="k", linestyle="-", marker="o", linewidth=1.5,label='True values')
plt.plot(x, rmsemlpd, color="K", linestyle="-", marker="+", linewidth=1.0,label='MLP')
plt.plot(x, rmselstmd, color="K", linestyle="-", marker="d", linewidth=1.0,label='LSTM')
plt.plot(x, rmsesvrd, color="K", linestyle="-", marker="s", linewidth=1.0,label='rbf SVR')
#plt.plot(x, prearimar, color="K", linestyle="--", marker="*", linewidth=1.0)
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(monthFmt)
fig.autofmt_xdate()
plt.title('Differenced Data')
plt.legend(prop={'size':6})
plt.grid(True)

plt.subplot(223)
#plt.plot(x, SAtest, color="k", linestyle="-", marker="o", linewidth=1.5,label='True values')
plt.plot(x, rmsemlpds, color="K", linestyle="-", marker="+", linewidth=1.0,label='MLP')
plt.plot(x, rmselstmds, color="K", linestyle="-", marker="d", linewidth=1.0,label='LSTM')
plt.plot(x, rmsesvrds, color="K", linestyle="-", marker="s", linewidth=1.0,label='rbf SVR')
#plt.plot(x, prearimar, color="K", linestyle="--", marker="*", linewidth=1.0)
plt.title('Normalized Differenced Data')
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(monthFmt)
fig.autofmt_xdate()
plt.legend(prop={'size':6})
plt.grid(True)

plt.subplot(224)
#plt.plot(x, SAtest, color="k", linestyle="-", marker="o", linewidth=1.5,label='True values')
plt.plot(x, rmsemlps, color="K", linestyle="-", marker="+", linewidth=1.0,label='MLP')
plt.plot(x, rmselstms, color="K", linestyle="-", marker="d", linewidth=1.0,label='LSTM')
plt.plot(x, rmsesvrs, color="K", linestyle="-", marker="s", linewidth=1.0,label='rbf SVR')
plt.plot(x, rmsevecs, color="K", linestyle="-", marker="*", linewidth=1.0,label='VEC')
#plt.plot(x, prearimas, color="K", linestyle="-", marker="*", linewidth=1.0,label='ARIMA')
plt.title('Normalized Data')
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(monthFmt)
fig.autofmt_xdate()
plt.legend(prop={'size':6})
plt.grid(True)

plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25, wspace=0.35)
plt.show()

x=DATA[-12:].index
#x=x.strftime('%b' )

#ax = plt.gca()
fig, ax= plt.subplots() # an empty figure with no axes
##fig.suptitle('12_steps Univarience Prediction Comparation')    # Add a title so we know which it
is
##fig, ax_lst = plt.subplots(2, 2)

months = mdates.MonthLocator()    # every month
```

```
monthFmt = mdates.DateFormatter('%b')

plt.subplot(221)
#plt.plot(x, SAtest, color="K", linestyle="-", marker="o", linewidth=1.5,label='True values')
plt.plot(x, rmsemlpr, color="K", linestyle="-", marker="+", linewidth=1.0,label='Original Data')
plt.plot(x, rmsemlpd, color="K", linestyle="-", marker="d", linewidth=1.0,label='Differenced Data')
plt.plot(x, rmsemlps, color="K", linestyle="-", marker="s", linewidth=1.0,label='Normalized Data')
plt.plot(x, rmsemlpds, color="K", linestyle="-", marker="*", linewidth=1.0,label='Differenced_Normalized Data')
#plt.plot(x, rmsearimar, color="K", linestyle="-", marker="*", linewidth=1.0,label='ARIMA')
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(monthFmt)
fig.autofmt_xdate()
plt.title('MLP')
plt.legend(prop={'size':6})
plt.grid(True)

plt.subplot(222)
#plt.plot(x, SAtest, color="k", linestyle="-", marker="o", linewidth=1.5,label='True values')
plt.plot(x, rmselstmr, color="K", linestyle="-", marker="+", linewidth=1.0,label='Original Data')
plt.plot(x, rmselstmd, color="K", linestyle="-", marker="d", linewidth=1.0,label='Differenced Data')
plt.plot(x, rmselstms, color="K", linestyle="-", marker="s", linewidth=1.0,label='Normalized Data')
plt.plot(x, rmselstmds, color="K", linestyle="--", marker="*", linewidth=1.0, label='Differenced_Normalized Data')
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(monthFmt)
fig.autofmt_xdate()
plt.title('LSTM')
plt.legend(prop={'size':6})
plt.grid(True)

plt.subplot(223)
#plt.plot(x, SAtest, color="k", linestyle="-", marker="o", linewidth=1.5,label='True values')
plt.plot(x, rmsesvrr, color="K", linestyle="-", marker="+", linewidth=1.0,label='Original Data')
plt.plot(x, rmsesvrd, color="K", linestyle="-", marker="d", linewidth=1.0,label='Differenced Data')
plt.plot(x, rmsesvrs, color="K", linestyle="-", marker="s", linewidth=1.0,label='Normalized Data')
plt.plot(x, rmsesvrds, color="K", linestyle="--", marker="*", linewidth=1.0,label='Differenced_Normalized Data')
plt.title('rbf SVR')
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(monthFmt)
fig.autofmt_xdate()
plt.legend(prop={'size':6})
plt.grid(True)

plt.subplot(224)
#plt.plot(x, SAtest, color="k", linestyle="-", marker="o", linewidth=1.5,label='True values')
plt.plot(x, rmsevecr, color="K", linestyle="-", marker="+", linewidth=1.0,label='Original Data')
#plt.plot(x, rmselstms, color="K", linestyle="-", marker="d", linewidth=1.0,label='LSTM')
plt.plot(x, rmsevecs, color="K", linestyle="-", marker="s", linewidth=1.0,label='Normalized Data')
#plt.plot(x, rmsevecs, color="K", linestyle="-", marker="*", linewidth=1.0,label='VECM')
#plt.plot(x, prearimas, color="K", linestyle="-", marker="*", linewidth=1.0,label='ARIMA')
```

```
plt.title('VEC')
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(monthFmt)
fig.autofmt_xdate()
plt.legend(prop={'size':6})
plt.grid(True)

plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25, wspace=0.35)
plt.show()

############bar plot#################
rmsemlpr=np.mean(rmsemlpr)
rmselstmr=np.mean(rmselstmr)
rmsesvrr=np.mean(rmsesvrr)
rmsearimar=np.mean(rmsevecr)

rmsemlpd=np.mean(rmsemlpd)
rmselstmd=np.mean(rmselstmd)
rmsesvrd=np.mean(rmsesvrd)

rmsemlps=np.mean(rmsemlps)
rmselstms=np.mean(rmselstms)
rmsesvrs=np.mean(rmsesvrs)
rmsearimas=np.mean(rmsevecs)

rmsemlpds=np.mean(rmsemlpds)
rmselstmds=np.mean(rmselstmds)
rmsesvrds=np.mean(rmsesvrds)

name_list = ['MLP','LSTM','SVR','VEC']
raw=[rmsemlpr, rmselstmr, rmsesvrr, rmsearimar]
difference=[rmsemlpd, rmselstmd, rmsesvrd, 0]
normalized=[rmsemlps, rmselstms, rmsesvrs, rmsearimas]
difference_scale_data=[rmsemlpds, rmselstmds, rmsesvrds, 0]

func = lambda x: [y for l in x for y in func(l)] if type(x) is list else [x]

raw=func(raw)
difference=func(difference)
normalized=func(normalized)
difference_scale_data=func(difference_scale_data)

raw=[round (i,2) for i in raw]
difference=[round (i,2) for i in difference]
normalized=[round (i,2) for i in normalized]
difference_scale_data=[round (i,2) for i in difference_scale_data]

x = np.arange(len(name_list))
total_width, n = 0.8, 4
width = total_width / n
x = x - (total_width - width) / 2

fig, ax = plt.subplots()
x = np.arange(len(name_list))
```

```
rects1 = ax.bar(x, raw,    width=width, label='Raw Data')
rects2 = ax.bar(x + width, difference, width=width, label='Differenced Data')
rects3 = ax.bar(x + 2 * width, normalized, width=width, label='Normalized Data')
rects4 = ax.bar(x + 3 * width, difference_scale_data, width=width, label='Normalized+Differenced
Data')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('RMSE')
ax.set_title('RMSE by Preprocessing and Method')
ax.set_xticks(x)
ax.set_xticklabels(name_list)
ax.legend()

def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 4, height),
                    xytext=(0, 3),    # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')
autolabel(rects1)
autolabel(rects2)
autolabel(rects3)
autolabel(rects4)
fig.tight_layout()
plt.show()
```