# **Introductions and Problem Definitions**

Nowadays, the increasing popularity of online retails in UK attracts the attention of many large investors. However, they are concerned about the wider economy and the consumer spending power of the UK. As according to Office for National Statistics, the retail (both online and in-store) industry as a whole is used as an indicator of how the wider economy is performing and the strength of consumer spending. Besides, they might also be concerned of whether or not the dot.com crash during 2000 to 2002 would happen again, or if there is any uncertainties in the highly complex e-retailing.

The researches on this topic has been quite intense in the past years. The phenomenon was discussed that in early years, the internet only served as a communication tool, it is rarely adopted by retailers. However, many researchers had identified the potential of internet retailing.

Early in 1999, Doherty, et al (1999) generated a model to explain why level of retailing on the internet was quite low in UK at that time, while specifying the potentials of internet retailing and providing guidance for those business who wish to increase their internet involvement in retailing. Hart and Doherty (2000) conducted a survey and found out the majority of the UK retail organisations surveyed have not yet registered a Web site address. Moreover, of those retail organisations that have developed a Web site, the vast majority are using it primarily as a communication tool to promote corporate or product information to Internet users, rather than to support direct sales. Despite the fact of few adoptions, they identified that internet would become a major new retail format, replacing the traditional dominance of fixed location stores.

During the dot.com crash period from 2000 to 2002, internet retailing was characterized as a huge challenge for retailing. As Reynolds (2000) stated: ‘Rarely has the retail and consumer services sector been faced with a strategic challenge of such signi­ficance complexity and uncertainty which has grown in terms of that significance so rapidly’

Nowadays, as internet retailing is widely accepted, it is still needed to be aware that e-retailing has not been less complex and uncertain (Pantano et al, 2004). Thus, it would not be an easy decision to make big investment in the online retailing market despite its current popularity. This topic of e-retailing is still significance and follow-up studies are essential.

Briefly state, the problem is about an investment decision of a large multinational in-store retailer. Specifically, they want to decide whether or not investing in the UK internet retail market. This investment decision is a big strategic action while also a huge challenge for the in-store retailing giant. Thus, to avoid taking too much risks, they are considering to make a small initial investment as a trial and keep close monitoring on the sales performance over the next year to further decide the necessity of a larger investment afterwards.

The purpose of our study is to assess the feasibility of the initial investment and give recommendations to the investor in two ways:

1. Providing forecasts for the UK monthly internet retail sales over the following year
2. Providing monthly updating forecasts (12 forecasts in total) for the next month over the following year

The first way aims to provide 12 forecasts immediately at present. So that the company can have an overall idea of the sales over the following entire year, which is exactly the horizon of their potential initial investment.

The second way aims to provide only 1 forecast for next month’s sales figure at present, however, as the next month arrives, we update our forecast for the month afterwards based on all the information we would have at the time we make our forecast. This process continues throughout the following year, by the end of which we would have provided 12 monthly forecasts for the company. In this way, we tend to generate more reliable forecasts and can help the company keep monitoring the sales performance by keep updating the forecasts in the short-term.

# **Forecasting Steps**

According to Hyndman and Athanasopoulos (2014), a forecasting task usually involves five basic steps, which are the steps we employed in this report.

* Step 1: Problem definition
* Step 2: Gathering information
* Step 3: Exploratory analysis
* Step 4: Choosing and fitting models
* Step 5: Using and evaluating a forecasting model

# **Gathering Information**

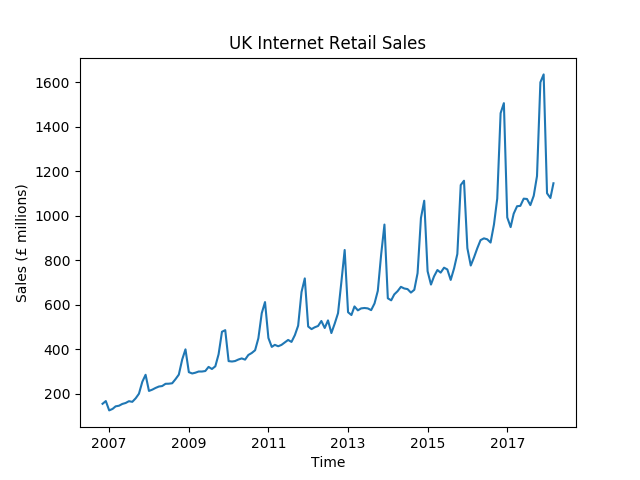
After defining our problem, the second step of our forecasting is gathering information. The dataset was collected from Office for National Statistics website. Based on the descriptions, the time series are sales by retailers in Great Britain directly to end consumers, including spending on goods online.

The data ranges from November 2006 to March 2018, with monthly interval, thus 137 observations in total. We employ this monthly data to build models and generate forecast for the following reasons: daily data is indeed more detailed, however, its neither efficient nor easily-available, the lots of noises in it make it unfavourable. Yearly data’s trend information is very obvious, but it is too general to spot any seasonal patterns from the data, especially for the UK internet retail sales dataset we used contains obvious seasonal patterns. In conclusion, monthly frequency is a perfect trade-off.

However, due to limited availability, the data before November 2006 is not in the dataset, which leads to the potential limitation of our analysis of not covering a long-enough economic cycle. We split the 137 time series dataset as in-sample set and out-of-sample set at a proportion of roughly 90% : 10% (125 observations as in-sample, and 12 observations as out-of-sample)

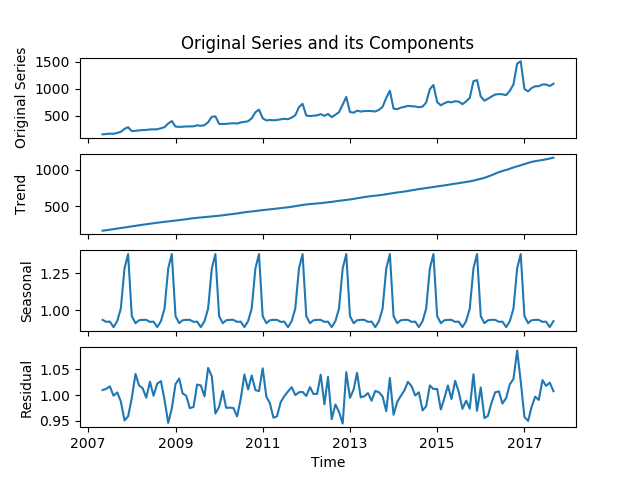
# **Exploratory Data Analysis**

## **The original series – plots**



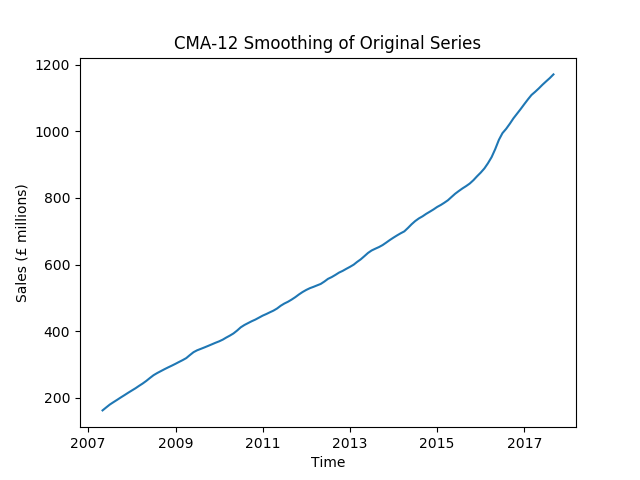
**Figure 4.1**

**Figure 4.1** shows the original series of UK Internet Retail Monthly Sales during the period from November 2006 to March 2018.



**Figure 4.2**

After discovering the series has a multiplicative seasonality with seasonal period of 12 months (more on this later), a multiplicative decomposition was conducted to separate trend, seasonal, cycle (insignificant thus can be ignored, more on this later), and residual components. It is presented as **Figure 4.2**.



**Figure 4.3**

**Figure 4.3** shows the smoothed series after conducting CMA-12 smoothing on the original series. The smoothed series can also be regarded as the initial trend estimate using in our decomposition above

## **Description of the series**

As from **Figure 4.1 and 4.2**, during this period of approximately 12 years, the UK internet retail sales shows an upward trend, with the trend being more significantly upward in the recent two to three years. The monthly internet retail sales in UK was only around £100m in 2007 and increased over time to almost £1,100m in 2018 without considering seasonal fluctuations. According to Bowsher (2018), the reasons for the continuing soar of online retail sales are mainly convenience and competitive prices. Bowsher (2018) also suggests that since 2016, the in-store retail sales has declined, which is because ‘online is now growing at the expense of outline’. This might be able to explain the reason why the upward trend in the UK Internet Retail Sales time series has become even more sloped in recent 2 years.

Besides, it presents obvious seasonal patterns with season period of 12 months. The seasonality and seasonal period of 12 months is obvious as we can observe 12 peaks during the 12 years, which were all in Decembers; while lowest sales were all in Januarys and Februarys, sales in other months were relatively smooth. To further prove this hypothesis of seasonal period of 12 months, we conducted a CMA-12 smoothing and found out that the seasonal patterns are all removed and left with a smoothed initial trend estimate (**Figure 4.3**). This experiment affirmed seasonal period of 12 months.

In addition, the seasonal peaks increase in magnitude when the trends magnitude increases, and the seasonal variation is proportional to the trend, which indicates a multiplicative rather than additive relationship between the trend and seasonal (and other) components. As we can discover from **Figure 4.1**, the difference between the highest and lowest sales in 2007 was about £40m; while grew all the way to around £640m in 2017. The variation of seasonality has increased 16 times over the past 10 years!

As it is found out that the series contains multiplicative seasonality, we conducted a multiplicative decomposition and generated **Figure 4.2**. After extracting the seasonal component and the trend component from the original series, only cycle and residual components are left. The residual plot indicates that there is no cycle component by showing almost same variation and no clear pattern throughout the period. Residuals fluctuate around 1 evenly seems reasonable since multiplicative model is used here.

## **Initial model selection inferred from EDA**

According to the above exploratory data analysis, we can summarise that the UK Monthly Internet Retail Sales series contains obvious upward trend component and multiplicative seasonal component. Therefore, when selecting models initially at this stage, the models that can deal with trend and multiplicative seasonal components are more favourable. Thus, the following models are employed in this report: Holt-Winters smoothing – multiplicative seasonality, SARIMA, neural network, and a forecast combination of them. Seasonal Naïve and drift method are also employed as benchmarks because Seasonal Naïve can characterize the seasonal component instead of trend component in the series, while Drift method can deal with the trend component but not seasonal component.

# **Methodology and Analysis**

## **Introduction**

After the initial model selection according to EDA, Naïve and drift method are employed as benchmarks; and Holt-Winters smoothing – multiplicative seasonality, SARIMA and neural network are used as models. Forecast combinations of these models using both equal weight and determined weight that can minimise the variance are also employed.

For each model, we fit the model using the in-sample set that contains 125 observations, then use the fitted model to generate forecasts in 2 ways:

1. Dynamic forecasting

Forecasts are generated based on the fixed forecast origin (using all the information up to the last time point of the in-sample set), with forecast horizon of 12 months.

i.e. Forecast based on , where denotes all the information up to time t.

1. One-step ahead forecasting

Forecasts are generated 12 times, one at a time (1-month forecast horizon) based on moving forecast origin (using all available information up to the time point when making each forecast).

i.e. Forecast based on , forecast based on , …, forecast based on .

After generating 2 types of forecasts for each model, we can then evaluate the model and the model combinations’ performances using several performance assessment criteria (ME, MAD, RMSE, MAPE) based on the differences between our forecasts and the out-of-sample validation set.

## **Seasonal Naïve Method**

### Model descriptions and explanations

While Naïve method which is the most basic forecast model uses the last observation to predict future, the seasonal naïve forecast future by using the last observed value of the last same season. The formula can be written as follow:

Where is the h-step ahead forecast, is the observation M period before time point t+h.

For example, when frequency M is equal to 12, prediction of the next January is equal to observation in the last January.

### Motivations

Seasonal Naïve model serves as a benchmark model, with which all other more advances models can compare. As from the EDA part, the UK Internet Retail Sales series shows obvious seasonal patterns. Thus the purpose of using this model as a benchmark is that compared to Naïve method, seasonal Naïve method is able to capture the seasonal patterns.

### Pre-processing steps and experiments

Step 1: Split out the in-sample set for model training

As discussed in the Gathering Information part, before the model-fitting process, we split the time series into in-sample set (125 observations) and out-of-sample set (12 observations), the model-training process as below only employs the in-sample series.

Step 2: Determining the seasonal period

We can observe from the original series plot (**Figure 4.1**) and the decomposed series plot (**Figure 4.2**) that the seasonal period should be 12, because 12 peaks occurred in 12 December and there were also similar patterns for the other months.

To be more rigorous, as from the EDA part, we conducted an experiment testing the period of seasonality by doing a CMA-12 Smoothing on the original series, and found out that all seasonal patterns are removed (**Figure 4.3**). Therefore, the seasonal period is assured to be 12.

### Model fitting and forecasting

There are no parameters to be estimated, so we can do forecasting directly for the period from April 2017 to March 2018. As seasonal period equals 12, the forecast for April 2017 to March 2018 are the observed values from April 2016 to March 2017 respectively. Here, the dynamic forecasting and the one-step ahead forecasting yield the same forecast as the 12 forecasts are all determined by existing information at March 2017. So, the added information for one-step ahead forecasting does not change the forecast results of the dynamic way.

*图片包含 屏幕截图

已生成极高可信度的说明*

**Figure 5.1**

**Figure 5.1** shows the forecasts of seasonal Naïve method, it appears with no doubt that the model only captures the seasonal information from the last seasonal period, without capturing any trend information at all. The resulting deviation from the out-of-sample set indicates the model’s negative performance when there is an apparent trend.

### Advantages and disadvantages

Like Naïve method, seasonal Naïve model is very straightforward and can be implemented easily with low cost. Although seasonal Naïve model is very simple, it is useful to capture seasonality and has acceptable accuracy. However, Naïve models are bad at forecasting when time series have apparent trend or time series contain much information, and obviously, simplicity can also be a disadvantage. Our forecasting results shown **Figure 5.1** demonstrate this shortcoming clearly. Also, it assumes constant seasonality, which fails to capture the multiplicative seasonality that increase its horizon as the series trends up in our case.

## **Drift Method**

### Model descriptions and explanations

Another benchmark model we chose is Drift method, which is suitable for capturing time series with trend. Unlike naïve method, drift method allows forecasts to change over time and the amount changed which is called drift, is the average change in historical data. Formula of drift method is shown as follow (linear form):

Where is h-step ahead forecast; and are the last and the first observation of in-sample data and T is the number of observations in in-sample set. The formula is equivalent to the extension line to the right of a line between the first and last observation.

### Motivations

The Drift method also serves as a benchmark model, however, different from Seasonal Naïve method, it can capture the upward trend component rather than the seasonal component of the original series.

### Pre-processing steps and experiments

Step 1: Split out the in-sample set for model training

Same as discussed in the Naïve method part, we split the time series into in-sample set (125 observations) and out-of-sample set (12 observations), the model-training process as below only employs the in-sample series.

### Model fitting and forecasting

Drift method only depends on past observations and there is no parameters needing to be estimated like Naïve method.

Dynamic forecasting

For dynamic forecasting, the intercept and slope of linear form formula are computed based on information we have at time T which is March 2017 in our case and the information would not be update when new sales value is observed. Thus, the intercept and slope would not change. As the forecasting horizon h increases, the forecast simple increases by the increased number of steps times the fixed slope.

One-step ahead forecasting

As for one-step ahead forecast, the information set is continuously updated, which means and T would change to the latest observed value, and the latest time point, indicating that the intercept and slope of the formula are also changing continuously correspondingly.

Forecasting outcomes

*图片包含 屏幕截图

已生成极高可信度的说明*

**Figure 5.2**

As expected, it can be seen from **Figure 5.2** that dynamic forecast only captures the upward trend shown as a straight line in orange. This forecast performs badly because the significant seasonality of original time series introduces a large gap between true values and forecasts.

As shown by green line, for one-step ahead forecast, updating new information into drift formula improves forecast accuracy dramatically as the updated data contains seasonal information.

### Advantages and disadvantages

Drift method is also a common choice of benchmark model for its simplicity. As **Figure 5.2** shows, drift method is good at capturing linear trend while it cannot capture any seasonality when doing dynamic forecast. This is exactly the opposite of seasonal Naïve method. Drift method is easy to understand and implement but it is unable to discover complex information in historical data and even to reflect these information on forecasting.

## **Holt-Winters smoothing – multiplicative seasonality**

### Model descriptions and explanations

Holt-Winters seasonal method is a subclass of exponential smoothing. As the formula below, forecasts produced using exponential smoothing methods are weighted averages of past observations, with the weights decaying exponentially as the observations get older (i.e. the more recent the observation the higher the associated weight) (Hyndman & Athanasopoulos, 2014).

Exponential smoothing provides a generalisation for the class of weighted average smoothing that is based on the forecasting rationale of exponentially decaying importance of past information, with acting as the smoothing parameter. The larger is, the more importance is given to the most current observation, the smaller is, the more smoothed result the model generates. Two common special cases are: when , it becomes Naïve forecasting; when , it approaches the equally weighted smoothing.

As the formula below, which is a deviation of the above formula, the idea can be further understood as a forecast can be made based on the weighted average of the observation and forecast for the latest time point .

All exponential smoothing methods are based on the rationale above. To introduce the sub-classes of exponential smoothing, there are Simple Exponential Smoothing (assume no trend, no seasonality), Holt’s linear trend method (assume trend, but no seasonality), and Holt-Winters seasonal method – additive and multiplicative (assume both trend and seasonality). As can be shown as the forecasting formula below, the difference between these methods is dominantly their assumptions of the time series, i.e. whether the series contains trend and seasonality components.

* For Simple Exponential Smoothing, it assumes
* For Holt’s linear method, it assumes
* For Holt-Winters seasonal method – multiplicative, it assumes

As will be discussed in the next section (Motivations), the exponential smoothing model we used is the Holt-Winters seasonal method – multiplicative, which assumes the time series contains trend and multiplicative seasonality component and generate forecasts based on this assumption. The formula are shown below:

From the above formula, are the level, trend, and seasonality information at time t. They are predicted using the existing observations, level, trend, and seasonality information. Using the estimated and pre-estimated , we can then generate the forecast for time . When forecasting dynamically, the forecast for the next period is used to generate forecast for later periods, and finally gives the equation to get - the h-horizon forecast. Note in the formula, means the remainder after h dividing by m.

### Motivations

Based on the idea of decaying rather than same importance of historical information in terms of forecasting, we decided to employ exponential smoothing type method to generate forecasts.

According to Hyndman and Athanasopoulos (2014), the selection of the method is generally based on recognising key components of the time series (trend and seasonal) and how these enter the smoothing method (in an additive or multiplicative manner). As discussed in EDA part, our series shows obvious multiplicative seasonal patterns, thus Holt-Winters multiplicative seasonal smoothing is considered the best exponential smoothing method characterizing the UK Internet Retail Sales time series and generating forecasts.

### Pre-processing steps and experiments

Step 1: Split out the in-sample set for model training

Same as discussed in the previous parts, we split the time series into in-sample set (125 observations) and out-of-sample set (12 observations), the model-training process as below only employs the in-sample series.

Step 2: Determining the seasonal period

Same as discussed in the previous models, the seasonal period is determined to be 12.

Step 3: Determining the starting values

To generate forecasts for the series, the level, trend, and seasonal factor, starting values () need to be specified at the beginning. Winters (1960) originally recommended that the seasonal factors should be normalised at the start of the series but not thereafter, so that they would soon ‘contain some of the trend effect’. Therefore, the initial seasonal factor are specified to be normalised (mean equals to 1), which is exactly as what has been specified in the ‘multiplicative’ function of the Python module ‘holtwinters’ as shown below, the initial level factor is determined to be the mean of the first m values, while the initial trend, initial seasonal factors and initial estimate are also specified as below:

Step 4: Determining the smoothing parameters

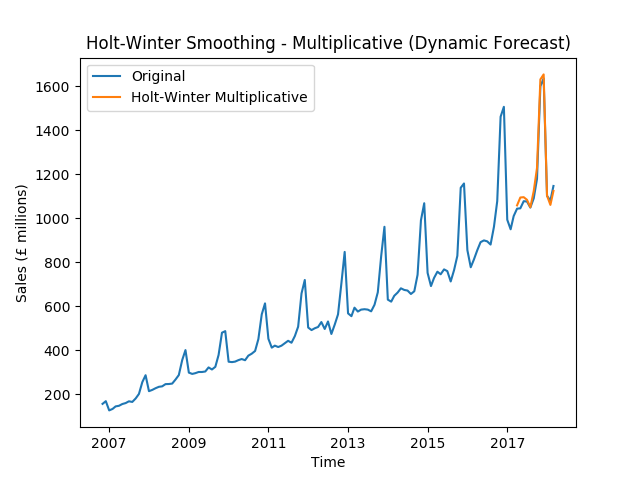
The smoothing parameters are determined based on the criteria of minimizing RMSE of the forecasts. This criteria is embedded in the ‘holtwinters’ package, thus makes it directly applicable when generating our own forecasts.

### Model fitting and forecasting

The model is fitted by importing the ‘holtwinters’ module and using its ‘multiplicative’ function, which is a function that takes the inputs of a series, seasonal period, and forecast horizon. The outputs of the function are generated by the function based on the formula and rationale presented in the ‘Model descriptions and explanations’ part. The outputs include the desired forecasting series based on the forecast horizon we specified.

Dynamic forecasting

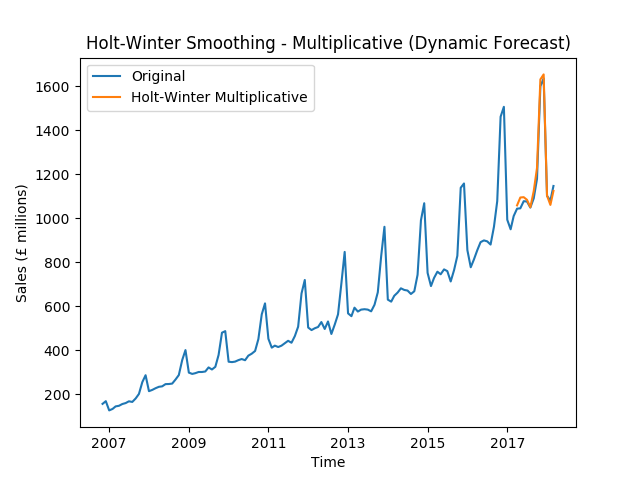
Since the series is the in-sample set of UK Internet Retail Sales time series, whose seasonal period is 12, to generate dynamic forecasts of horizon equals 12, we simply assign the last input of forecast horizon as 12. The forecasts are shown below as **Figure 5.3**:



**Figure 5.3**

One-step ahead forecasting

To generate one-step ahead forecast, we keep the seasonal period input as 12, and simply change the forecast horizon input to 1, and change the series input to be all available information at the time making forecasts. In other words, the series input is assigned to be the in-sample set when forecasting the first value of the out-of-sample set; it is then assigned to be the in-sample set plus the first observed value from the out-of-sample set when forecasting the second value of the out-of-sample set, etc. We repeat fitting this model 12 times to generate 12 forecasts, which is shown below as **Figure 5.4**.



**Figure 5.4**

To summarise, both forecasts using either dynamic or one-step ahead approach appear to be almost identical to the original out-of-sample set, which shows the ideal forecasting performance of Holt-Winter’s multiplicative smoothing method on our series.

### Advantages and disadvantages

The Holt-Winters method is easy-to-use and generally works quite well in practice (Chatfield and Yar, 1988). It can forecast a time series without the necessity of fitting a parametric model (Gelper et al, 2010).

One advantage of Holt-Winters method, or of all exponential smoothing methods is that as a weighted average smoothing method, its presents as the compromise of two extreme situations (Naïve and the average method). Also, the rationale behind it is highly reasonable, which is the decaying importance of historical information.

As it assumes both trend and seasonality, Holt–Winters smoothing is a widely used tool for forecasting business data that contain seasonality, changing trends and seasonal correlation. (Gelper et al, 2010) This framework generates reliable forecasts quickly and for a wide spectrum of time series which is a great advantage and of major importance to applications in industry (Hyndman & Athanasopoulos, 2014). As in our case, the dynamic forecasting using Holt-Winters multiplicative method demonstrates ideal performance, which is even better than the one-step ahead forecasting (will be discussed in Model Assessment part).

According to Gelper et al (2010), The exponential and Holt–Winters techniques are sensitive to unusual events or outliers and outliers can affect the forecasting methods in two ways. One is because the update equations involve current and past values of the series including the outliers. The other is by affecting the selection of the parameters used in the recursive updating scheme, which will significantly affect the forecasting results. Thus, Holt-Winters smoothing is non-robust and may be under-performed in the presence of outliers.

Another potential limitation of Holt-Winters seasonal method is that it assumes the trend keeps on forever, and the multiplicative seasonality component keep multiplying at the current scale forever, which might makes the model inappropriate in times of changes, such as recession in the economy. One possible solution to it would be using dampened method of Holt-Winters. However, in our case, we only need to forecast for the next 12 months, so this issue was not taken into consideration when generating our forecasts.

## **SARIMA**

### Model descriptions and explanations

ARIMA model examines autocorrelation in time series and in forecast error terms. It is a combination of autoregressive model (AR) and moving average model (MA), and “I” in ARIMA model represents order of differencing which is used to satisfy assumption of stationarity.

Seasonal ARIMA model is an extension of original ARIMA model combining a non-seasonal ARIMA with seasonal AR and MA in terms of seasonality (Hyndman & Athanasopoulos, 2014). Like original ARIMA model, It is assumed that time series should be weakly stationary—the first and the second moments are stationary, when implementing Seasonal ARIMA model. The model can be written as follows:

Within the model, is the non-seasonal term. Parameters are the orders of autoregressive model— AR(p), and moving average model—MA(q); and d is the order of differencing which is used to make time series stationary. Seasonal term in this model is. Capital letters P,D,Q have similar meaning with (p,d,q) but they are corresponding to seasonality. The subscript m shows seasonal period.

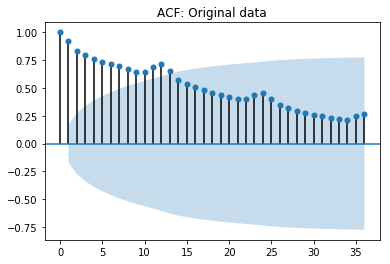
In order to make clear explanation, consider an example:

From subscript (m=12), we know that this model is used for monthly data with annual seasonality. To implement this model, we firstly do a first order seasonal differencing and a first order differencing. For observed time series :

Then we do AR(1) and seasonal MA(1) using differenced time series. In this example, the SARIMA model can be written as:

Where is the time series after differencing, is a constant, and are coefficients, and is the error term.

### Motivations



**Figure 5.5**

As can be seen from **Figure 4.1** and **Figure 5.5**, the patterns in the original series plot and the ACF plot’s slow drop to zero both indicate the existence of autocorrelations between the time series, which motivates the AR type model. Also, as discussed in the EDA part, the Internet Retail Sales time series also shows obvious seasonal and upward trend patterns. Therefore, a model that can characterize both autocorrelations and seasonality of a time series would be seasonal ARIMA.

### Pre-processing steps and experiments

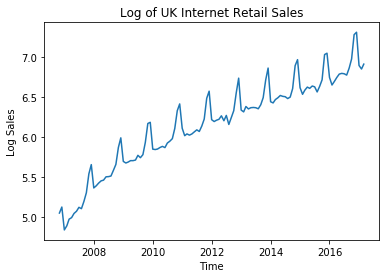
Step 1: Split out the in-sample set for model training

Same as discussed in the previous parts, we split the time series into in-sample set (125 observations) and out-of-sample set (12 observations), the model-training process as below only employs the in-sample series.

Step 2: Determining the seasonal period

Same as discussed in the previous models, the seasonal period is determined to be 12.

Step 3: Stabilizing the variance of the original series



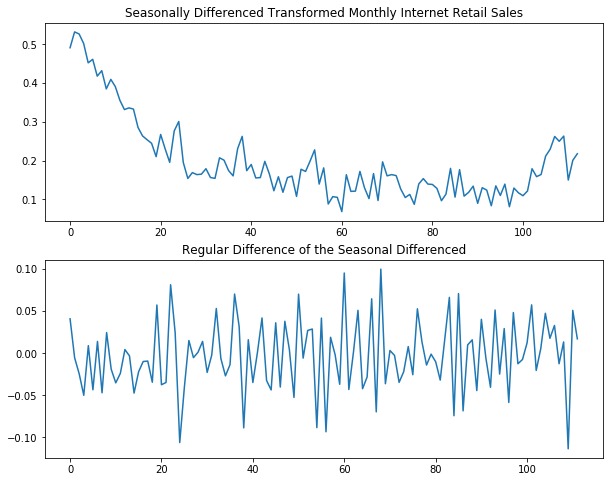
**Figure 5.6**

As discussed in the EDA part, the original series shows multiplicative seasonal period, which means its variance is unstable. Logarithmic transformation is thus taken on the original series to stabilize its variance, because this logarithmic transformation can turn a multiplicative series into an additive series. The transformed series is presented as **Figure 5.6**.

Step 4: Satisfying stationarity assumption

* Step 4.1: Check the stationarity of the original series

As from the original plot (**Figure 4.1**) and the transformed series plot (**Figure 5.6**), it is obvious that both series are non-stationary. Thus, we implemented the following differencing steps.



**Figure 5.7**

* Step 4.2: Remove seasonal patterns by seasonal differencing

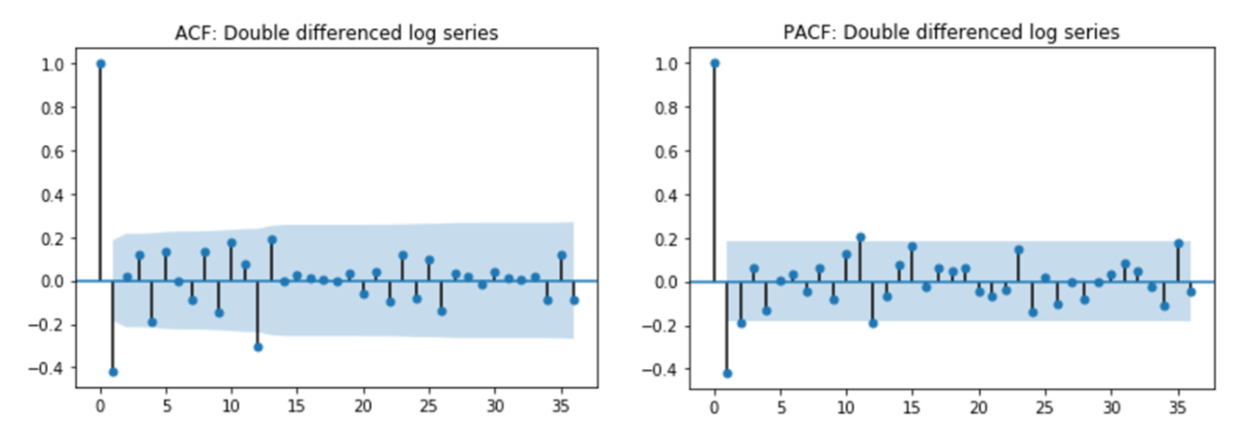
As the transformed data (**Figure 5.6**) shows strong seasonal pattern, a seasonal differencing is taken on the transformed series as shown in the upper part of **Figure 5.7**.

* Step 4.3: Remove the regular patterns by regular differencing

However, after doing seasonal differencing, the series is still not stationary. Differenced data in the first 20 months have a downward trend before they seem to be stationary around 0.15. Thus, a regular first order differencing is done on the seasonally differenced transformed series in order to further stabilize the series. The second panel of **Figure 5.7** shows that double differenced data are roughly mean stationary and we can move on to the next step.

Step 5: Initially determining the orders of SARIMA

To conduct the experiment of choosing the appropriate orders of the model, as a starting point, we plotted both ACF(Autocorrelation Function) and PACF (Partial Autocorrelation Function) plots (**Figure 5.8**).

**

**Figure 5.8**

* Step 5.1: Choosing the order of differencing: d and D

The ACF plot shows that the autocorrelation dies down to zero very quickly after conducting first-order seasonal and first-order differencing, which confirms that the series is now stationary. Thus, we can determine the order of differencing in seasonal ARIMA model to be d=1 and D=1.

* Step 5.2: Choosing the order of AR and MA: p, P, q, and Q

Parameters p, P, q and Q are determined based on the ACF and PACF plots above (**Figure 5.8**). At the seasonal level, autocorrelations of lag 12, 24 and 36 die down gradually in PACF while ACF plot has a clear cut-off at lag 12. As a result, we assign 1 to Q indicating a seasonal MA(1) model. At the non-seasonal level, the ACF plot displays a spike at lag 1 and the lag-1 autocorrelation is negative. As for PACF, it has significant partial autocorrelation at lag 1 and 2. Therefore, we consider adding a non-seasonal MA(1) term as well as a AR(2).

* Conclusion

To conclude, based on ACF and PACF plots and the above analysis, the seasonal is initially determined.

### Model fitting

The initially determined model: seasonal can be fitted easily using function from ‘statsmodels’ package provided by Python. However, the orders in non-seasonal and seasonal terms are chosen subjectively based on ACP and PACF plots as in the last part. We still need to find the optimal model.

One popular way to find the optimal orders of seasonal ARIMA model is through brute force grid search. In our case, “Brute force” method finds the global minimum of the BIC (Schwarz’s Bayesian Information Criterion) through computing values of BIC at multidimensional grid of points (scipy.optimize.brute, n.d.). The reason of using BIC as criterion is that BIC, similar to AIC (Akaike’s Information Criterion), can be used to select the optimal ARIMA model. Differently, BIC would consistently choose the best model with fewer parameters. All combinations of assigning p, P, q and Q from 0 to 4 and assigning order of differencing (including seasonal differencing) from 1 to 2 are tried. Since the number of grid points is reasonably small, despite the method’s theoretical limitations, negligible computation costs were taken implementing this method. The resulting model is . Function fitting Seasonal ARIMA model also provides calculated AIC and BIC. According to these two criteria, The best seasonal ARIMA model can be selected:

|  |  |  |
| --- | --- | --- |
|  | AIC | BIC |
| SARIMA | -416.847 | -402.705 |
| Optimal SARIMA | -416.469 | -405.156 |

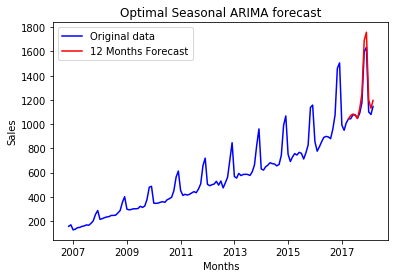
**Table 5.1: AIC and BIC for Seasonal ARIMA and the Optimal Seasonal ARIMA**

**Table 5.1** shows that the optimal seasonal ARIMA selected by Brute force method has similar AIC with model selected by ourselves and the lowest BIC. Thus, as a result, is considered better based on information in in-sample data. The p-value of Ljung-Box test is 0.69 indicating that the residuals follow white noise process and this result confirms is appropriate.

### Forecasting

Dynamic forecasting

The dynamic forecasting results for are shown below:

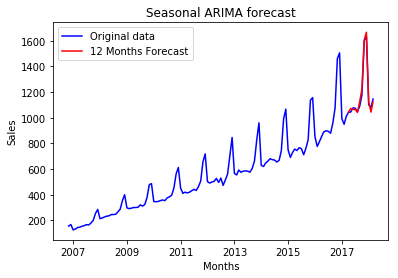
**

**Figure 5.9**

The forecasts appear to have a high degree of coincidence with the original out-of-sample set, except for the sales peak in November and December 2017 where the forecasts are about £150 million higher than the original data and the drop at the beginning of 2018 where the forecasts are around £100 million higher than raw data.

One-step ahead forecasting

The one-step forecasting results for are shown below:

**

**Figure 5.10**

**Figure 5.10** indicates that the one-step ahead forecast with rolling origin has higher accuracy than dynamic forecasting. The reason might be that by updating new observation to information set used to estimate model coefficients, the model learns more and the forecast consequently gets closer to true value. It still can be seen that forecast for December 2017 is slightly higher than true sales and forecasts for January and February in 2017 and 2018 are slightly lower than observed values.

### Advantages and disadvantages

As commonly used models, ARIMA or seasonal ARIMA have apparent advantages. First of all, ARIMA models only requires times series data in question, and this avoids some problems occurring with multivariate models (Meyler et al, 1998). In addition, the principle of ARIMA models is relatively simple to understand.

However, the disadvantages of ARIMA or seasonal ARIMA model are also obvious. Firstly, the model assumption requiring weak stationary which has to be achieved by differencing sometimes makes resulting model hard to be interpreted. Secondly, orders of ARIMA models are chosen by two ways. Observing significant spikes in ACF and PACF plots is too subjective while using searching algorithm such as Brute force method used above would exponentially increase computational cost when combinations of orders tried are too many. What’s more, ARIMA models are “backward looking” which leads to poor forecasting for time series with turning points and they are not suitable for long forecasting horizon (Meyler et al, 1998).

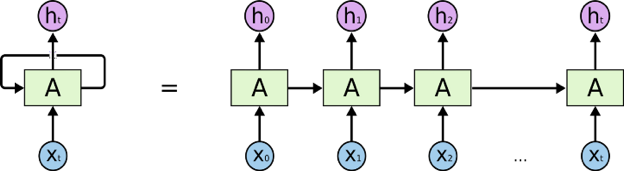
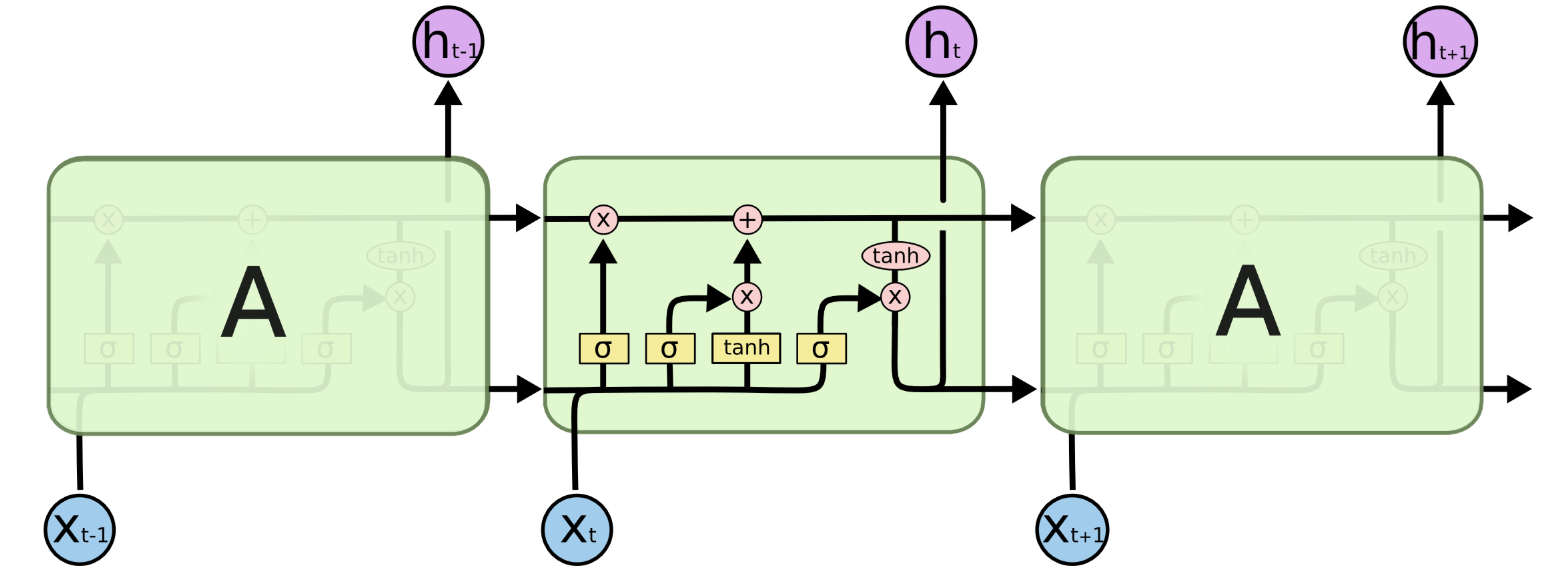
Combining with our topic, seasonal ARIMA model does better in one-step forecasting than in dynamic forecasting with several turning points at peaks and bottoms of sales during the last year. This is consistent with the last disadvantage discussed above.

## **Neural Network**

### Model descriptions and explanations

Neural network (NN) is a powerful approach for forecasting by simulating the way human brain solves problems. Information in the neural network flows from input layer, through an interconnected group of nodes to the output layer. At the output layer, a nonlinear activation function like sigmoid can be used to generate the forecasts, which introduces nonlinearity into the network. Generally speaking, NN is widely used for forecasting purpose.

However, NN is limited in terms of time series forecasting since it ignores the sequence dependence among the time series, the problem of which can be solved by recurrent neural network (RNN). Unlike NN, RNN allows information to be passed from one step of the network to the next like a loop. But training RNN incurs the problem of vanishing gradient due to the long-range dependences in the time series. This problem can be well addressed by the Long Short-Term Memory (LSTM) networks, which is also a type of neural networks.

**Figure 5.11** RNN has a loop **Figure 5.12** The Cell State of LSTM Networks

The LSTM networks consist of three parts:

* The “forget” part, which determines how much information should be dropped from the network:
* The “input” and “update” part, which determines what new information should be imported and updates the information stored in the network:
* The “output” part, which determines what information to be output:

Basically, the sigmoid function in LSTM networks controls the scale of the information stored in the network so that the vanishing gradient problem is addressed and the network is able to store information further away such as the long-term dependences among the time series.

### Motivations

Compared with traditional time series forecasting, neural networks have many advantages. According to Tang and Fishwick (1993), neural networks can outperform the Box-Jenkin models in terms of multiple-step-ahead forecasting, which is required in many real-world problems. Besides, unlike most conventional time series methods, which mainly focus on linear relationships, neural networks can learn both linear and nonlinear relationships. Additionally, assumptions do not need to be made when using neural networks, while this is necessary for traditional forecasting methods. For example, when doing regression forecasting, we need to make assumptions on the errors terms being uncorrelated with the predictors and so on. What’s more, since LSTM networks are able to pick up both long-term and short-term, a fixed size time window is not needed to be specified first, while for ARIMA method, we should specify the lagged terms in order to construct the model.

Since LSTM network has so many benefits stated above, we would like to apply this method to generate a competitive forecast against traditional time series forecasting methods.

### Pre-processing steps and experiments

Before fitting model, we split our data into training, validation and test sets, which contain 125, 12 and 12 data points separately and we will use cross validation in our training set during experiment. By assuming Gaussian distributing, our time sequence will be centered and scaled with standardization instead of normalization method. Due to the stochastic forecasting method implemented by LSTM, we will repeat forecast each model 10 times and use the aggregated results, average RMSE and standard deviation of RMSE, to measure each model’s performance. During the experiment we need to figure out four major questions: what kind of data should we input into the LSTM, what window size is suitable for training this model, how many neurons should we use and how many epochs will avoid both overfitting and underfitting issues.

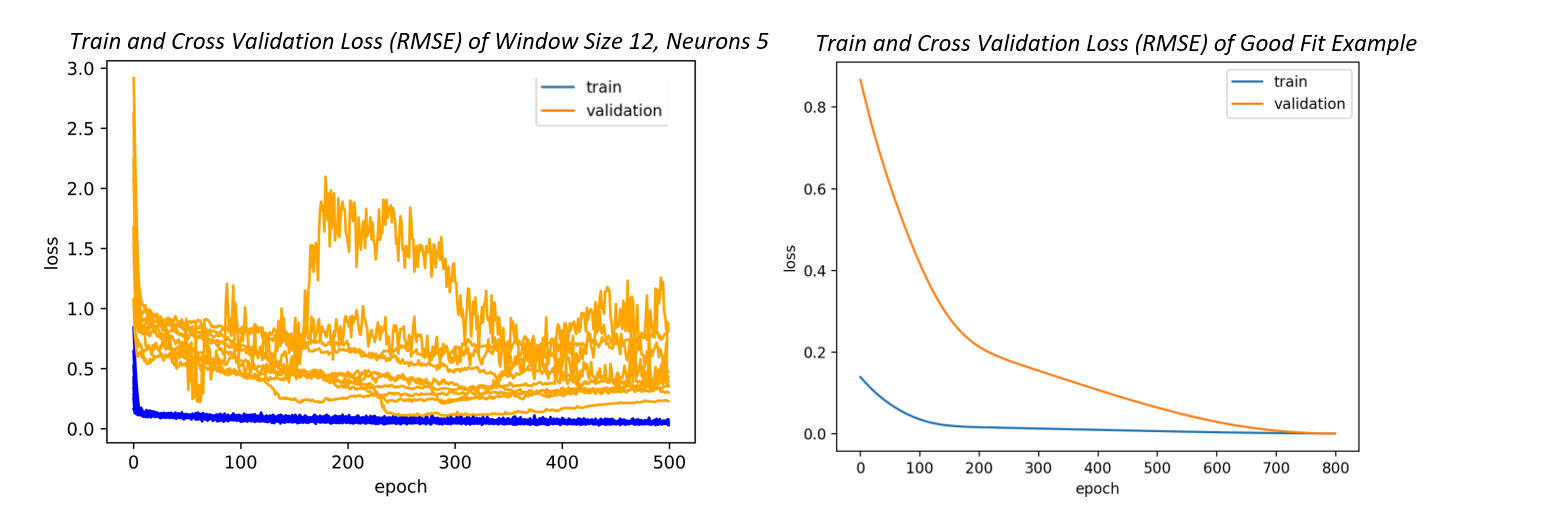
Step 1: Fitting model with unadjusted series

* Step 1.1: Varying input window size to optimal model performance

With the assumption that LSTM can learn the season, trend and cycle patterns of our data set altogether, we directly implement LSTM with unadjusted data series. Since changing window size can input different information volume to our model and thus may help us interpret the noise brought by the unadjusted data, we decide to focus on different window size first and fit our model with window of 6, 12, 15 and 20 and fix the number of neurons and epochs at 5 and 500 individually. However, the results show that our fitted models behave terribly with large one-step ahead validation RMSE at around 300 to 500 million pounds, and standard deviation of RMSE at 15 to 50 million pounds, which will be unacceptable in our forecast. Plus, the training and cross validation loss plots of those models, as shown in **Figure 5.13**, also significantly deviate that of a good fit model as shown in **Figure 5.14**.

* Step 2.1: Changing epoch and neuron numbers under different window size

We then change the number of neurons from 1 to 20 with step 5 at each window size but still cannot witness observable improvement of our validation RMSE and standard deviation of the validation RMSE.

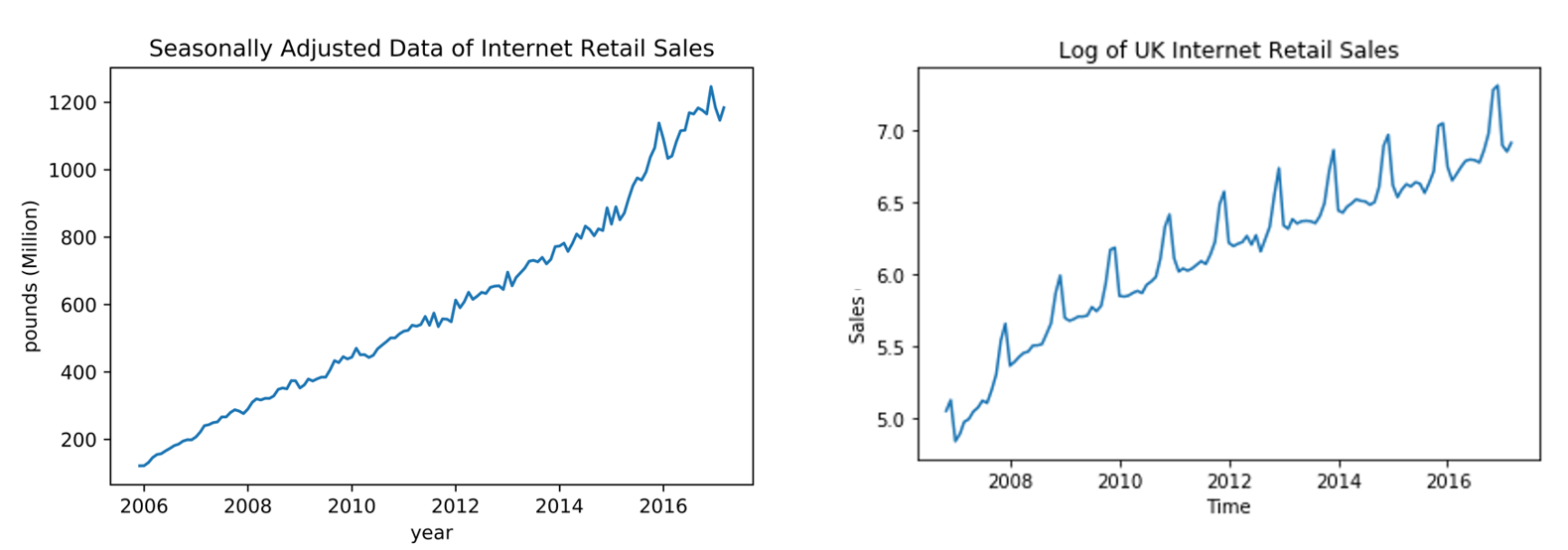


**Figure 5.13 Figure 5.14**

After these two procedures, we conclude that our data is not suitable for unadjusted input when training LSTM, and that may because LSTM cannot capture the noise in our sequence and we need to reduce variance of our data before using them.

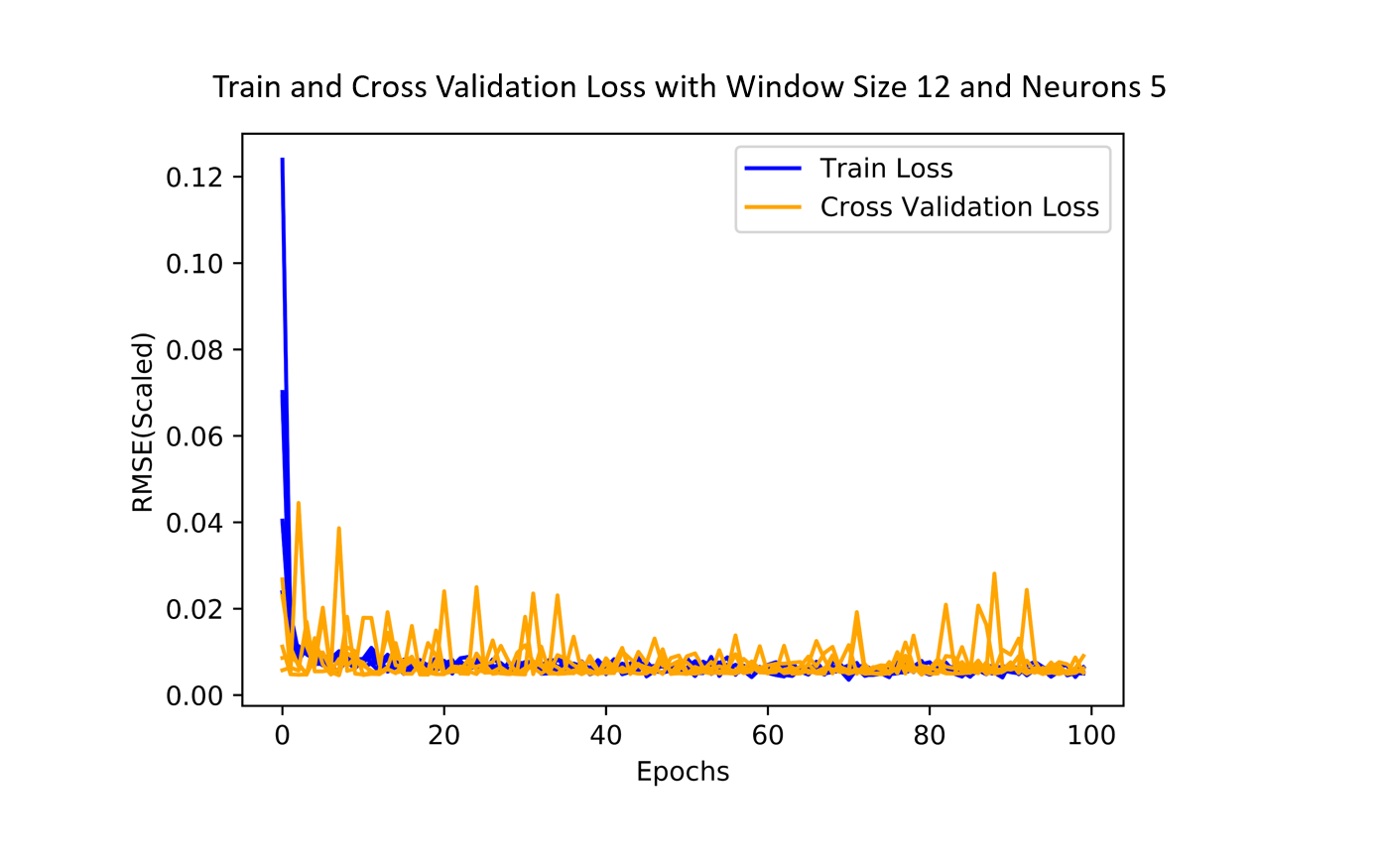
Step 2: Seasonally adjusting series to smooth variance

Due to the bad performance of our data in step 1, we decide to reduce the variance of our time series by removing seasonality and letting LSTM only learn the trend of our series, adjusted data can be seen in **Figure 5.15**. We use multiplicative decomposition method here since our data have constant seasonality and almost linear trend after log transform, as seen in **Figure 5.16**.



**Figure 5.15 Figure 5.16**

We still follow the logic of step 1, first fitting seasonally adjusted data in different window size and then varying the epoch and neuron number under specific time window. The RMSE and standard deviation of RMSE in our models is still quite large at around 200 to 300 million pounds and 12 to 20 million pounds separately. Moreover, from training and cross validation loss plots in **Figure 5.17**, we can see cross validation loss is even smaller than training loss in these models, which shows that our models behave abnormally. We infer that LSTM overinterpret the seasonally adjusted data and we need to input a more complicated time sequence than seasonally adjusted sequence during the learning process.



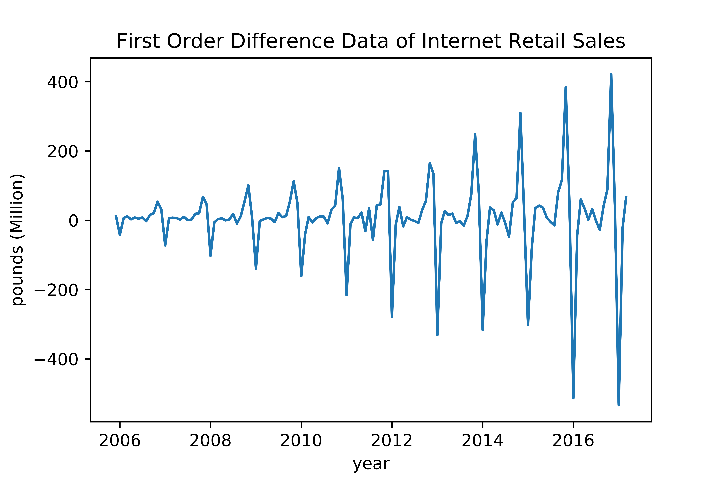
**Figure 5.17**

Step 3: Introducing Stationarity into data transformation

Since failing to fit our model successfully, we consider referring the stationarity assumption under traditional Box-Jenkins methods for time series forecasting and use stationary transformation for the sales data. We will try this step by step by first taking the first order difference to see if this works or not and then taking the first order and seasonal first order difference if the first step is not ideal.

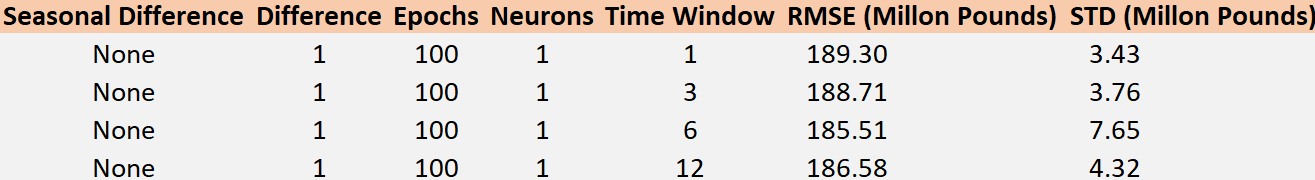
* Step 3.1: Using first order differenced sequence as input

The first trial in our stationary transformation is to take the first order difference, the sales data after transformation can be seen in **Figure 5.18**. We can see the data is still non-stationary and there is still seasonality remained in series.



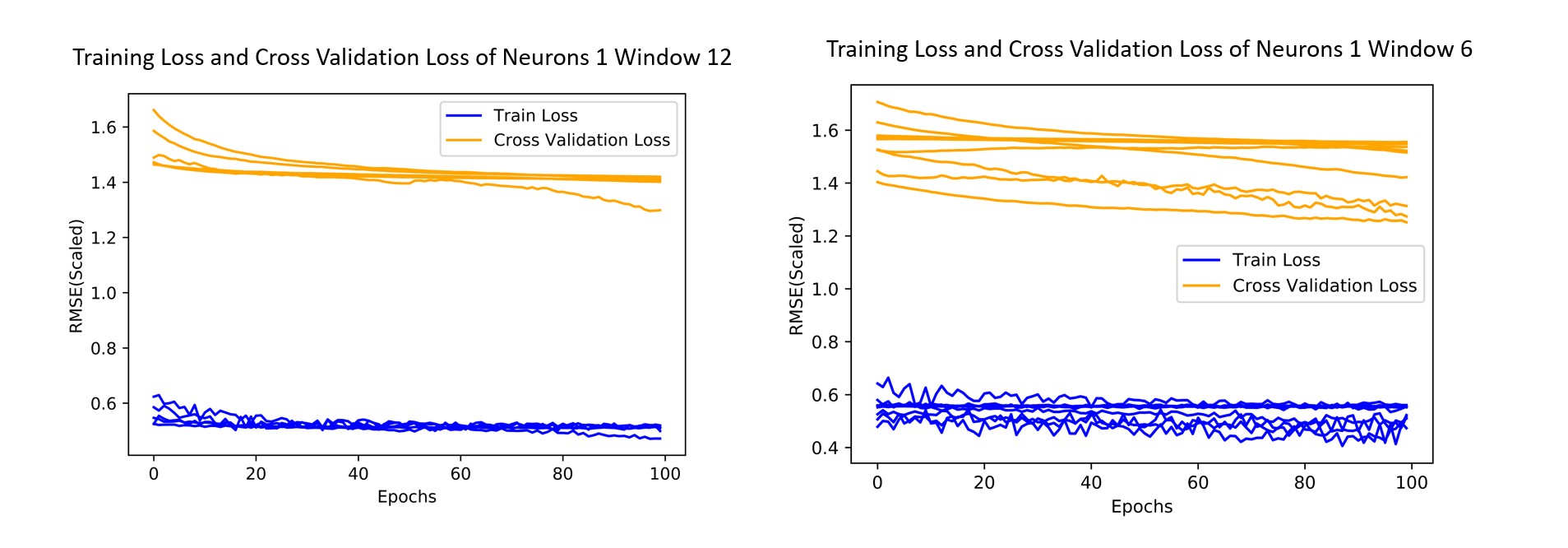
**Figure 5.18**

During tuning neurons and epochs in steps 1-2, we found that they have really mild effects on our model performance, thus, we will only focus on changing window size here and set neuron at 1 and epoch at 100. After fitting the following models in **Table 5.2**,we found the RMSE imporves a lot and the standard deviation of RMSE decreases significantly, which means that our models start to behave stable and we are on the right track.



**Table 5.2**

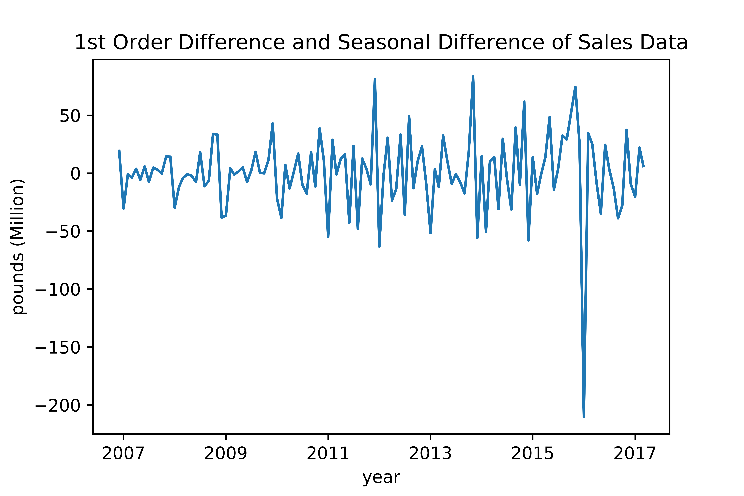
The training and cross validation loss plots, example in **Figure 5.19**, shows that there is less randomness in our loss lines, which can further interpret the reason why our standard deviation of RMSE improves a lot. However, after changing window size we cannot see any further improvement in our RMSE scores and the training and cross validation loss lines have no trend of convergence, seen in **Figure 5.20**. Plus, we further experiment by varying neuron and epoch number under different window size, and we also cannot find any noticeable improvement by doing that, which again confirms our conclusion of weak influence from neurons and epochs. Thus, although we are on the right way of finding a suitable model, this data transformation is still not enough.



**Figure 5.19 Figure 5.20**

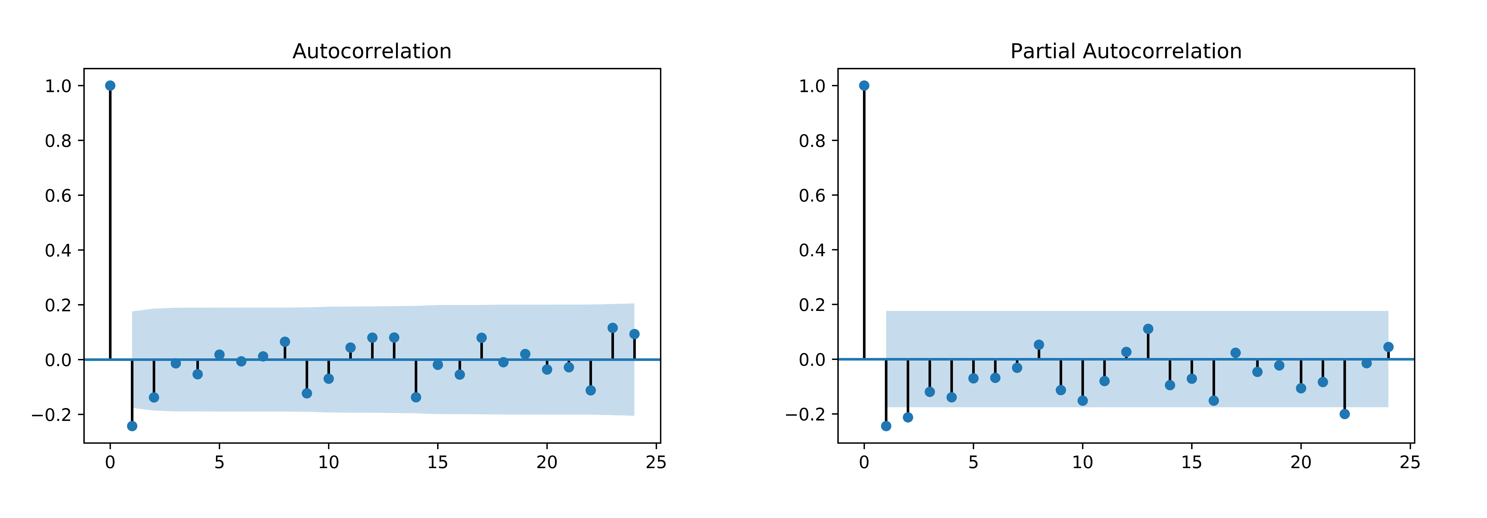
* Step 3.2: Improving model with first order and seasonally differenced data

Our second trail in stationary transformation is to take the first order and seasonal first order difference. After transformation, our data look like the plot in **Figure 5.21**. We can see although there is one significant outlier, the data is almost stationary.



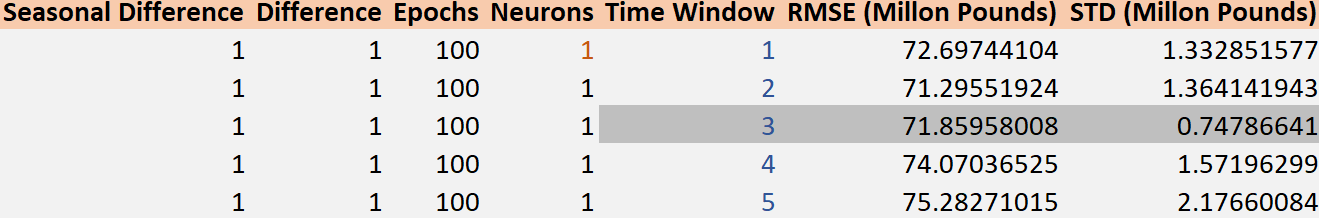
**Figure 5.21**

To further improve our efficiency, we also want to employ ACF and PACF plot to diagnose the remaining AR effect in the differenced series and use the lags larger than confidence interval in PACF plot as our reference for choosing window size. From the PACF plot in **Figure 5.23**, we can see there are two lags significantly different from 0 and thus the suggestion for window size given by PACF plot:



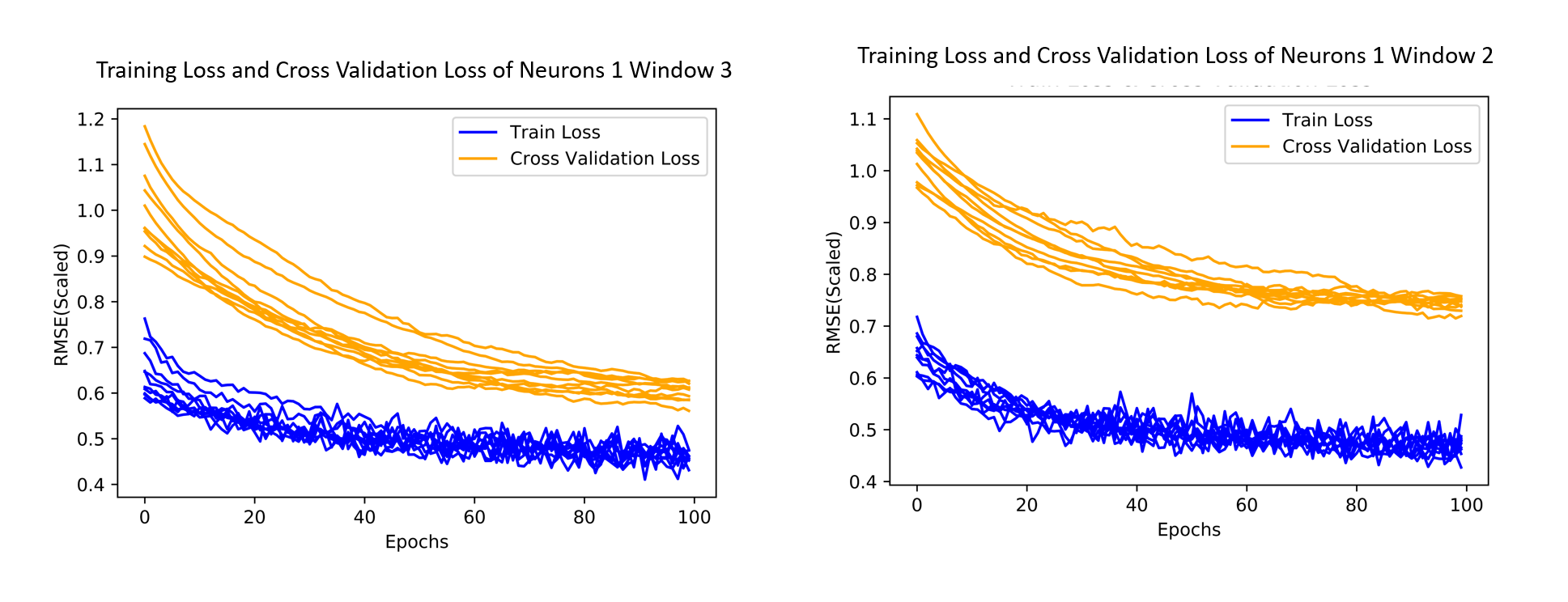
**Figure 5.22 Figure 5.23**

To confirm our assumption that window size can be inferred from ACF and PACF plot, we run models with fixed neuron of 1 and epochs of 100 and change the number of window size starting from 1. After running models with window size range from 1 to 5, we found that the RMSE score for window size 3 is the lowest, and this is the same with our assumption of choosing window size by PACF plot. The RMSE score for different window size are in **Table 5.3.**



**Table 5.3**

And we can also find that the RMSE score for this model is acceptable and the standard deviation of the RMSE score is again improved. From the training and cross validation loss plots, in **Figure 5.25**, we can also see that model with time window 3 significantly outperforms the one with time window 2. And the cross-validation loss lines in window 3 may converge to training loss line in the future if we can select optimal neurons and epochs number for slight adjustment purpose.



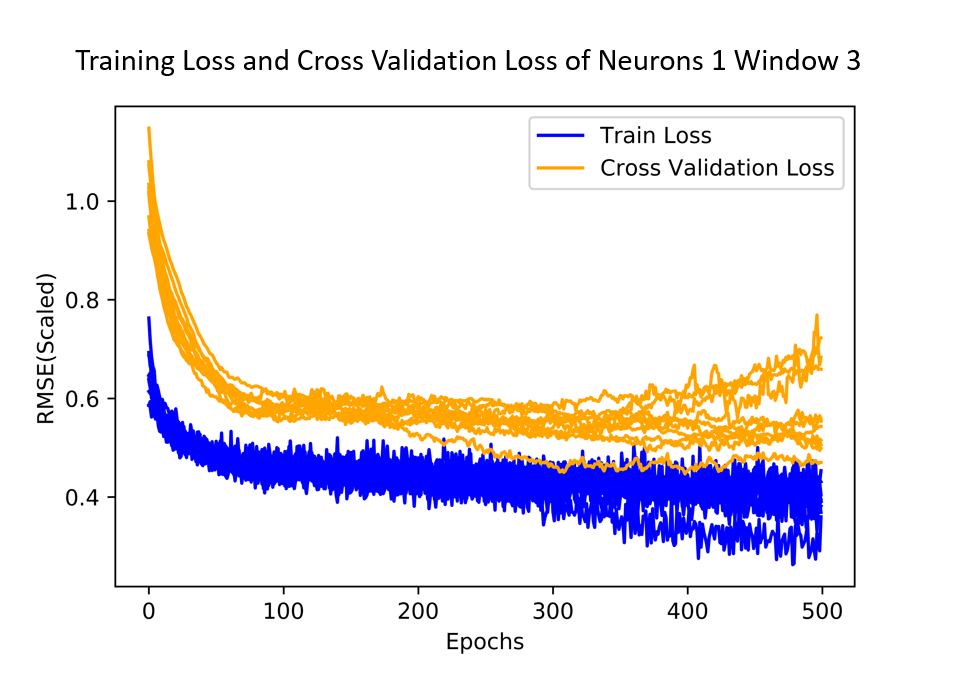
**Figure 5.25**

Therefore, we will use first order difference and first order seasonal difference as our data input and use time window 3 as one of hyper-parameters and we will tuning neuron and epoch numbers during model fitting process.

### Model fitting

Step 1: Tuning epoch number

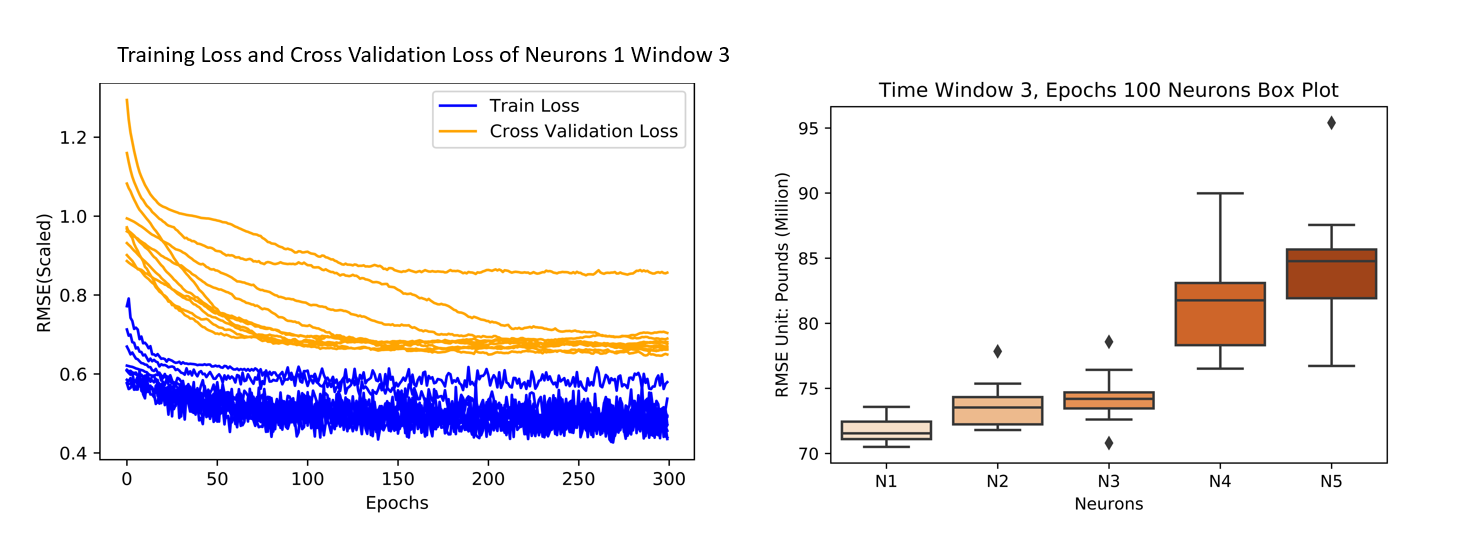
During our experiment, we find that epoch number has the smallest effect on model performance and optimal epoch numbers are almost the same when varying window sizes and neurons. And based on the above training and cross validation loss plots in our experiment setup process, we observe that epoch equalling to 100 has already been closed to the optimal range. Thus, to find a better epoch number, we will choose the optimal epoch by setting neuron at 1 and window size at 3 and find the number of epoch that can lead to the lowest cross validation loss in our training and cross validation loss plot. From the **Figure 5.26** below we can see that the optimal number of epoch is around 300. Therefore, we will use 300 as our optimal epoch in the following neurons tuning process and final forecast model.



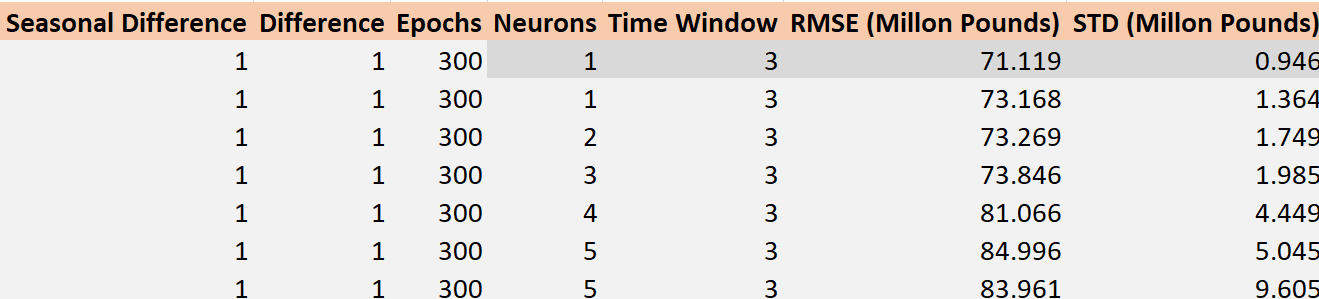
**Figure 5.26**

Step 2: Tuning neuron number

In our experiment set up process, we find that our data only need a small number of neurons and by adding more neurons will tend to induce overfitting issues. Thus, we will find the optimal number of neurons starting from neuron equal to 1 and use epoch equal to 300 and time window equal to 3 as other hyperparameters’ inputs. During our tuning process, we find that neuron equal to one can give us the lowest RMSE and standard deviation of RMSE, as shown in **Figure 5.28.** Plus, the training and cross validation loss plot of one neuron has good performance. Thus, we stop tuning the number of neurons after increasing them to 5 and decide to select one neuron as our hyperparameter for final forecast model. We can see our neurons tuning process in **Table 5.4** and **Figure 5.27** below.



**Figure 5.27 Figure 5.28**



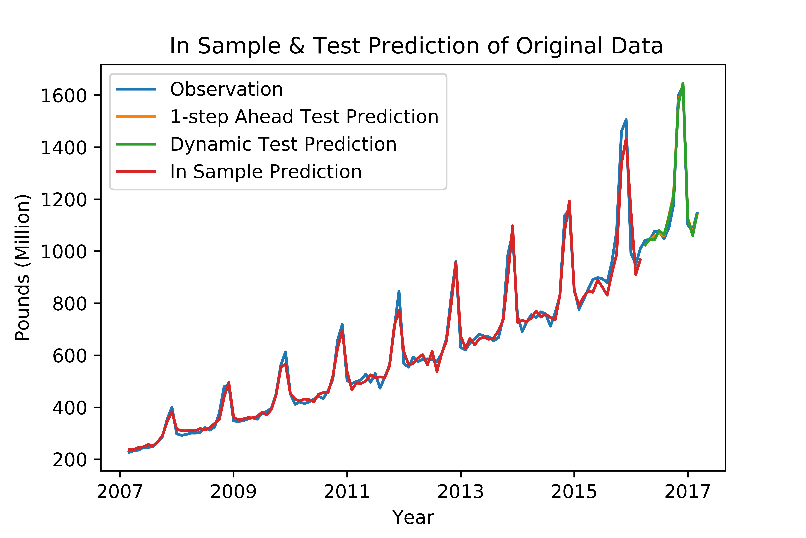
**Table 5.4**

Step 3: the number of repeats

Assuming stochastic process can generate normally distributed results, we will repeat 30 times for our final forecast model and use the average forecast results as our final forecast so that we can diversify some variance in our forecast model.

### Forecasting

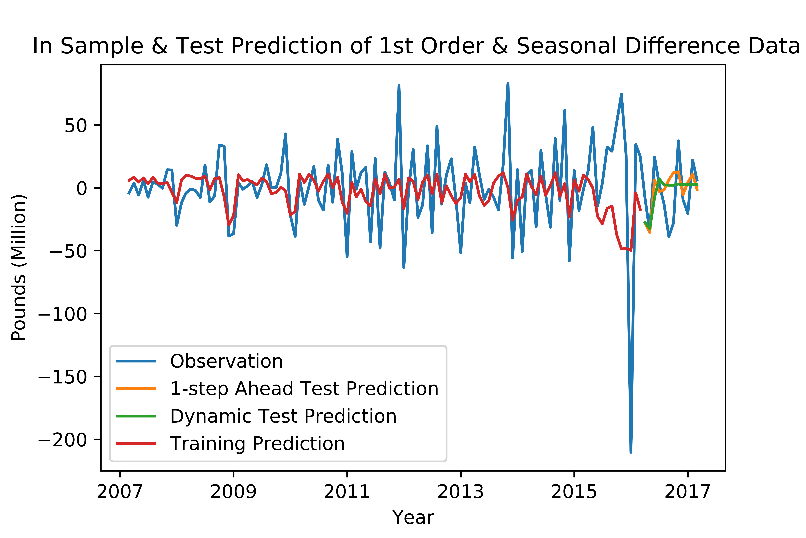
Setting neurons at 1, epochs at 500, time window at 3, the dynamic and one-step-ahead forecasts are generated and the result is shown below:



**Figure 5.29**

As is shown in **Figure 5.29**, the dynamic and one-step-ahead forecasts are almost overlapped, as the difference of the RMSE for the two forecasts is quite small, with the RMSE of dynamic forecasts slightly higher than that of one-step-ahead forecast. Since the RMSE of LSTM is lower than either that of the benchmark or the Box-Jenkin methods, we conclude that both dynamic and one-step-ahead forecasts perform well. It can be seen that the increase of the sales in 2017 is larger than 2016 and we use the sales in 2017 as a validation set. However, the forecasts of LSTM still perform well, especially for one-step-ahead forecasts, which indicates LSTM’s powerful learning ability in terms of the temporal dependences among the time series. Note that the RMSEs of the dynamic and one-step-ahead forecasts is much smaller than that of the training results. This is simply because for forecasts, only a year’s RMSE is calculated while for training, about 10 years’ RMSE is calculated.

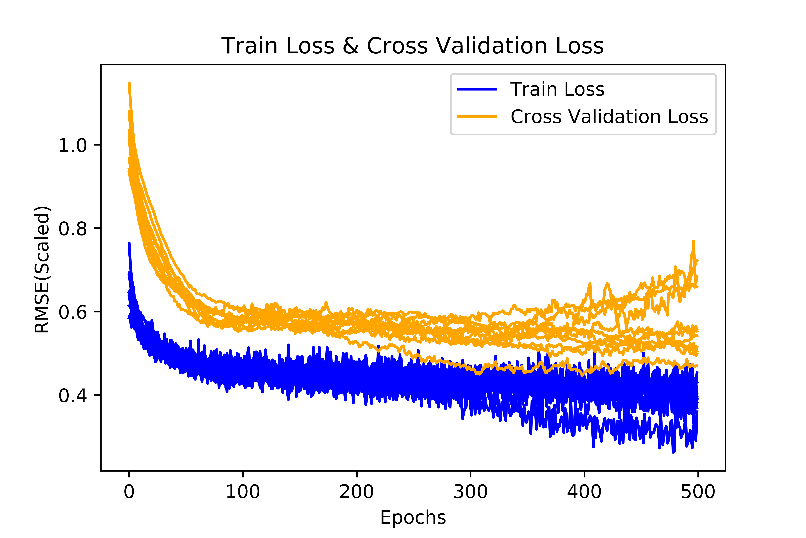
The forecast result of the 1st order and seasonal difference data are shown below:



**Figure 5.30**

From **Figure 5.30** we can see that LSTM network picks up the noise in the data with the training prediction following the trend of the noise in the observations. For dynamic forecast, we can see that after two months it almost stays unchanged, while for one-step-ahead forecast, it can still follow the general trend.

The training loss and cross-validation loss are plotted together in the graph below:



**Figure 5.31**

From **Figure 5.31**, we can see that the difference of the train loss and the cross-validation loss decreases as the number of epoch increases and when epoch is about 400, the train loss and cross-validation loss are quite close to each other, which indicates that the model is a good fit.

### Advantages and disadvantages

As is discussed in the motivations, LSTM has many benefits compared with conventional time series forecasting methods, such as no assumptions needed to be made, powerful learning ability of the linear and nonlinear relationships in the data, etc. This introduces a lot convenience and flexibility when fitting the LSTM model as well as high accuracy of forecasting results. Furthermore, neural network is robust to the noise in the data. That is why although with noise and even outliers in the data (which can be seen from **figure 5.30**) the forecasts of LSTM can still outperform traditional forecasting methods.

However, the disadvantages of LSTM networks are also obvious. Firstly, the neural networks are generally regarded as “black boxes”. It is difficult to truly understand how the data is processed in the network and how the forecasts are generated. This introduces difficulty to the process of tuning the hyperparameters since there is no golden rules for the optimal values of the parameters. Besides, the tuning process is also time-consuming. In our case, it took us a long time to tune the hyperparameters and fit the model. In addition, LSTM is a stochastic algorithm by its nature, which means that the same model trained on the same data can generate different results. This leads instability to the forecasts. The performance of the model can vary a lot. Therefore, for this set of sample data, we have to repeat fitting the model for 30 times to make the RMSE close to the true value, which is quite time-consuming.

## **Forecast Combinations**

### Purpose of combining forecasts

The performance of forecasts can be improved by forecast combination (Bates & Granger, 1969).

According to Bates and Granger (1969), the main reason of using forecast combination is that it combines information contained in one forecast that is not considered by the other forecasts, thus make the forecasting more robust. For example, as in previous discussions of benchmark models, seasonal Naïve model only captures seasonality while drift method only captures trend. Seasonal and trend information can be gathered into forecast through forecast combination.

Another benefit is the reduction of variance. It is shown by Bates and Granger (1969) that the variation of forecast error decreases when forecasts from different method are combined, and the reason for this is the benefit of diversification, which means the variation would decrease when combining different variables as long as they are not perfectly positively correlated.

### Assigning weights

There are many ways to assign weights when combining forecasts and we consider two methods that are most common and easy to apply—equal weights and forecasting errors’ variance-minimising weights.

Equally weighted forecast combination

Combining forecasts with equal weights simply takes the average forecasts from different models:

Where is the h-step ahead combined forecast; m is the number of forecasting models and is the h-step ahead forecast from model.

Combining forecasts by simple average is very easy to apply and often works surprisingly well comparing with complex techniques and equal weighting method allows to combine as many models as we want. However, this method might be oversimplified and it cannot assign higher weight to models having higher forecasting accuracy (Assigning weights to an averaged forecast, 2016). Equal weighted combination performs well when forecasting models perform roughly the same based on similar time series (Timmermann, A., 2013).

Variance-minimising forecast combination

Another method used is assigning weight that minimising mean squared errors of combined forecast. The underline principle is similar to portfolio optimisation minimizing risk (variance of return) through diversification. More specifically, the process of assigning weights is as follow:

Where is weight assigned to forecast of model, and it is equal to weights minimizing variance of combined forecast error (). When there is only two models to be combined, the variance of combined forecast error and weight formula can be written as:

Where , are variance of forecasting errors from model 1 and 2 respectively, and is the correlation coefficient of two series of forecasting errors.

As for combining 3 models, Lagrange Multiplier is used to calculate the optimal weights.

Where w is the vector of weights, and is the variance-covariance matrix of forecasting errors.

To solve this optimization problem, Lagrange Multiplier is used:

Where is the Lagrange Multiplier.

This weight assigning method has solid theoretical support which is minimising variance of combined forecast errors. In addition, this method is easy to understand as well and is used as basis for some of advanced combining techniques. However, some of forecasts combined might be changing over time resulting in changes of error variance of themselves. In this case, variance of combined forecasting errors would change accordingly and weights should be adjusted (Bates & Granger, 1969). In addition, the weights calculated through this method is not guaranteed positive. Thus, additional constraint is needed which increases computational difficulty especially when combining more than two models. This method is suitable for unbiased forecasts due to its assumptions (Bates & Granger, 1969).

### Analysing results of forecast combinations

Firstly, we have to assign weights to each model in different combinations. In **Table 5.5** below, w1, w2, w3 represent weights assigned to forecasts from Holt-Winters smoothing model (HW), seasonal ARIMA model (SARIMA) and Neural Network (LSTM) respectively.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | HW&SARIMA | HW&LSTM | SARIMA&LSTM | HW&SARIMA&LSTM |
| equal weight | w1 | 1/2 | 1/2 |  | 1/3 |
| w2 | 1/2 |  | 1/2 | 1/3 |
| w3 |  | 1/2 | 1/2 | 1/3 |
| weight minimizing Variance  (Dynamic) | w1 | 0.7899 | 0.5235 |  | 0.4692 |
| w2 | 0.2101 |  | 0.1939 | -0.0242 |
| w3 |  | 0.4765 | 0.8061 | 0.5549 |
| weight minimizing Variance  (One-step) | w1 | 0.5095 | 0.4579 |  | 0.4789 |
| w2 | 0.4905 |  | 0.4066 | 0.1131 |
| w3 |  | 0.5421 | 0.5934 | 0.4080 |

**Table 5.5**

**Table 5.5** shows that in dynamic forecasting, Holt-Winters model and Neural Network model play more important roles than seasonal ARIMA model because weights assigned to HW and LSTM are higher than weight assigned to SARIMA. However, in one-step forecasting, similar weights are assigned to each model in two model combination, while SARIMA still has smallest weight in three model combination.

As illustrated before, it is possible to have negative weight when doing error variance minimizing and there is a negative weight assigned to dynamic SARIMA forecasts when combing three models. Assessment of forecast combinations will be discussed in assessment part.

## **Model Assessments**

### Descriptions of the performance measures

In order to find the optimal forecasting model, assessment criteria are crucial. There are four commonly used criteria—Mean Error (ME), Mean Absolute Deviation (MAD), Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE).

1. ME is the simplest one calculated by averaging forecast errors which are differences between true observations and corresponding forecasts. The formula of ME is:

Where is the true observation and is the forecast at time t.

Mean error being equal to zero implies that forecast model is unbiased. However, ME might not be a good criteria assessing forecast accuracy because when it is equal to zero, there might still be large variation in errors and positive and negative errors cancel each other out by computing average. Thus, this criteria is more suitable when forecasts bias is considered more important than typical size of errors. Including ME in our assessment helps to check whether there is forecast bias or not.

1. MAD makes a modification based on ME:

MAD calculates average of absolute errors instead of errors and this modification does not allow positive and negative errors to cancel each other out. MAD is easy to calculate, understand and interpret and minimizing MAD leads to forecasts of median (Hyndman & Athanasopoulos, 2014).

1. MSE squares errors rather than taking absolute values in MAD:

Since Mean Squared Error has different scale with other criteria and is hard to interpret, we use RMSE which is the square root of MSE as the third criterion instead of using MSE.

Squaring errors makes MSE penalize more on large errors no matter they are positive or negative. As a result, RMSE is usually larger than MAD. While forecast of median is led by minimizing MAD, minimizing RMSE leads to forecasts of the mean (Hyndman & Athanasopoulos, 2014).

1. MAPE is a very popular assessment criterion in business forecasting.

As shown in formula above, MAPE is a scale-free criterion presented in percentage form which is suitable when comparing forecast performance between different data sets. However, drawbacks of MAPE are obvious. The observation at time t is used as denominator in the formula indicating that this criterion cannot be used if any . Also, MAPE will have extreme value when any is close to zero. As a result, MAPE cannot be used when assessing forecast accuracy of time series like stock returns.

### Comparisons between models

First of all, the best forecast combination is selected and then we compare it with benchmark models as well as individual models.

For dynamic forecasting, values of four criteria discussed above are listed:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dynamic |  | ME | MAD | RMSE | MAPE |
| Equal weight | HW&SARIMA | -31.1855 | 31.1855 | 39.3947 | 2.5040 |
| HW&LSTM | -8.3778 | 15.5756 | 19.7580 | 1.3728 |
| SARIMA&LSTM | -24.4043 | 28.0743 | 35.1482 | 2.3022 |
| 3 MODELS | -21.3225 | 21.9273 | 28.9595 | 1.7851 |
| Min Variance | HW&SARIMA | -21.8937 | 23.8889 | 29.8254 | 1.9680 |
| HW&LSTM | -8.6963 | 15.3833 | 19.8842 | 1.3581 |
| SARIMA&LSTM | -10.4394 | 19.66 | 23.9961 | 1.6809 |
| 3 MODELS | -13.2518 | 16.9632 | 21.7558 | 1.4423 |

**Table 5.6** Assessment of dynamic forecast combination

(Equal weighting and weights minimizing error variance)

***Table 5.6*** shows that the combination of Holt-Winters model and Long Short Term Memory model performs best having lowest values of MAD, RMSE and MAPE compared with other combinations. Although combination weighting by minimizing variance has lower MAD and MAPE, equal weighted combination of HW and LSTM is chosen since size of forecasting error is considered important.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| One-step |  | ME | MAD | RMSE | MAPE |
| Equal weight | HW&SARIMA | -31.1855 | 31.1855 | 39.3947 | 2.5041 |
| HW&LSTM | 2.3758 | 16.4038 | 19.5199 | 1.4102 |
| SARIMA&LSTM | -4.5539 | 19.162 | 22.2576 | 1.655 |
| 3 MODELS | -0.9199 | 16.8312 | 20.1044 | 1.4811 |
| Min Variance | HW&SARIMA | -0.4506 | 18.2896 | 22.2353 | 1.6536 |
| HW&LSTM | 2.0417 | 16.2376 | 19.4469 | 1.3973 |
| SARIMA&LSTM | -4.0013 | 19.1269 | 22.7035 | 1.649 |
| 3 MODELS | 2.2741 | 16.2649 | 19.4693 | 1.3957 |

**Table 5.7** Assessment of one-step forecast combination

As shown by **Table 5.7**, the conclusion of best one-step forecast combination is the same with dynamic forecasting. However, combination of Holt-Winters model and LSTM model weighting by minimizing error variance, has lower values of all three criteria—MAD, RMSE and MAPE than equal weighted combination. Thus combination of HW and LSTM weighted by the second method is chosen as the best model.

After the best combination is chosen, we can assess all models considered.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Dynamic* | *ME* | *MAD* | *RMSE* | *MAPE* |
| *Seasonal Naïve* | 145.3000 | 145.3000 | 147.8572 | 12.7622 |
| *Drift Method* | 129.7804 | 129.7804 | 232.9199 | 9.0676 |
| *Holt-Winters - multiplicative* | *-15.1590* | *22.1956* | *26.9980* | *1.9036* |
| *SARIMA* | -47.2120 | 47.4065 | 62.4124 | 3.7549 |
| *Long Short Term Memory* | *-1.5965* | *19.6698* | *23.2109* | *1.6873* |
| *Forecast Combination – equal weight* | -8.3778 | 15.5756 | 19.7580 | 1.3728 |
| *Forecast Combination – min variance* | -8.6963 | 15.3833 | 19.8842 | 1.3581 |

**Table 5.8** Assessment of dynamic forecasting

Obviously, the forecast combination performs the best with smallest MAD, RMSE and MAPE, and equal weighted combination is the optimal model having the lowest RMSE (19.758) penalizing more on large errors. For three individual models, LSTM has the lowest absolute values for all four criteria while seasonal ARIMA model has the highest. This result attributes a lot in assigning weights discussed in forecast combination part, in which forecasts from HW model and LSTM model have greater weight than SARIMA. Also, LSTM is the model having lowest bias among all models considered here with ME being equal to only -1.5965. Apparently, as expected, individual models (HW, SARIMA, LSTM) and forecast combinations outperform two benchmark models.

One-step forecasting evaluation is as following:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *One-step* | *ME* | *MAD* | *RMSE* | *MAPE* |
| *Seasonal Naïve* | 145.3000 | 145.3000 | 147.8572 | 12.7622 |
| *Drift Method* | 3.7092 | 107.3149 | 200.6018 | 8.6937 |
| *Holt-Winters - multiplicative* | *6.8694* | *23.2373* | *24.4882* | *2.0145* |
| *SARIMA* | -7.5113 | 21.7582 | 25.8082 | 1.8772 |
| *Long Short Term Memory* | *-3.1619* | *17.6984* | *22.2246* | *1.5398* |
| *Forecast Combination – equal weight* | 2.3758 | 16.4038 | 19.5199 | 1.4102 |
| *Forecast Combination – min variance* | 2.0417 | 16.2376 | 19.4469 | 1.3973 |

**Table 5.9** Assessment of one-step forecasting

Same with dynamic forecasting, forecast combination has the highest forecast accuracy with lowest values of all criteria calculated and has lowest bias, but combination minimizing variance is considered optimal. Long Short Term Memory model again outperforms Holt-Winters and seasonal ARIMA model. However, the absolute values of four criteria of seasonal ARIMA model decrease dramatically compared with dynamic forecasts and seasonal ARIMA even has higher forecast accuracy than Holt-Winters when considering MAD (21.7582) and MAPE (1.8772). Benchmark models have worst performance having criteria values being more than five times larger than other models

As a result, equal weighted forecast combination is considered optimal in dynamic forecasting while forecast combination weighting by minimizing variance is the best in one-step forecasting.

# **Prediction**

According to the best model selected by model assessment, the dynamic forecasts and the one-step ahead forecasts are shown as below:

# 

**Figure 6.1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Dynamic** | HW | LSTM | 1/2HW+1/2LSTM |
| Apr-18 | 1166.647 | 1023.72 | 1095.183539 |
| May-18 | 1180.11 | 1046.58 | 1113.345113 |
| Jun-18 | 1216.962 | 1044.681 | 1130.821538 |
| Jul-18 | 1211.219 | 1079.476 | 1145.347204 |
| Aug-18 | 1174.466 | 1062.992 | 1118.72897 |
| Sep-18 | 1225.744 | 1130.638 | 1178.19107 |
| Oct-18 | 1333.571 | 1207.95 | 1270.760528 |
| Nov-18 | 1809.048 | 1564.644 | 1686.84584 |
| Dec-18 | 1843.825 | 1646.581 | 1745.203197 |
| Jan-19 | 1237.206 | 1123.88 | 1180.542887 |
| Feb-19 | 1201.909 | 1060.057 | 1130.98309 |
| Mar-19 | 1265.224 | 1142.259 | 1203.741494 |

**Table 6.1: Dynamic Forecasting**

|  |  |  |  |
| --- | --- | --- | --- |
| **one-step** | HW | LSTM | Min var weighted |
| Apr-18 | 1166.647 | 1023.72 | 1089.065886 |

**Table 6.2: One-Step Ahead Forecasting**

The weights corresponding to Figure 6.2 are: HW – 0.4579, LSTM – 0.5421.

# **Conclusions and Recommendations**

In conclusion, this report aims at facilitating a large multinational in-store retailer on an investment decision regarding to the UK internet retail market. To give well-grounded recommendations, we generated two types of forecasts (dynamic forecasting, and one-step ahead forecasting) for the UK monthly internet retail sales based on the time series of UK internet retail sales obtained from the Office for National Statistics website. The dynamic forecasts aims at giving the investor an overall idea of the monthly sales for next year, while the one-step ahead forecast serves as the main source of generating our recommendations, since it provides more reliable forecasts in terms of the highly dynamic UK e-retailing market. Using one-step ahead forecast, we can help the investor keep monitoring the sales performance by monthly updating our forecast.

Seasonal Naïve method and drift method are used as benchmark model as they are simple and can characterise seasonal and trend pattern respectively. Three advance models: Holt-Winters methods – multiplicative seasonality, SARIMA, and Neural Network are also employed after conducting comprehensive experiments on them. The experiment results show the seasonal period of 12, helped determine the starting values for Holt-Winter method’s level, trend, seasonality factors, and initial smoothed value; the Holt-Winter’s smoothing parameters. Before fitting SARIMA, the experiments also informed us to stabilise the original series by taking logarithmic transformation on the series; to take seasonal and regular differencing to satisfy the stationary assumption; to decide the orders for SARIMA and finally determine the use of seasonal . The results generated from the Long Term Short Memory experiment leads to four important conclusions. First, data transformation is crucial in LSTM model fitting. we need to use first order difference and seasonal first order difference to conduct stationarity transformation when processing our data. Second, time window plays more roles in model performance than epoch and neuron numbers. Our optimal hyperparameters for final forecasting model here are time window 3, epoch 300 and neurons 1. Third, during experiment we found that we can use PACF and ACF plot as a reference for selecting time. Fourth, LSTM is suitable for learning temporal dependence and adapt really fast to the change of data pattern. In addition, forecast combinations are also used, and weights are assigned in 2 ways: equal weights, and weights that minimise variances of forecasting errors. We experimented on every possible combinations of the 3 models (3 combinations in pairs and a combination of the 3 models) according to both dynamic forecasting and one-step ahead forecasting.

After fitting the models, their forecasting performances are assessed using criteria of ME, MAD, RMSE, and MAPE. The result shows that best performance is achieved by combining the pair of HW and LSTM. Further, based on the criteria of minimizing RMSE, the combination performs better if equally weighted when dynamic forecasting; whilst the same pair performs better if choosing weights that can minimise variance when performing 1-step ahead forecasting. Based on this best method, we then generate monthly forecasts for the next month (using variance-minimizing weighted combination) and next year (using equally weighted combination). As from our thorough analysis, all the forecasting models including the best method (combination of HW & LSTM) indicate an obvious increase of the UK internet retail sales over the next year. Thus, based on the approximately last 12 years' historical data, we can conclude and recommend the investor to realise the idea of investing in the UK internet retail industry.

However, there are several limitations within our analysis: firstly, the dataset we used (137 observations, only 125 in-sample series) are not considered to be a large sample size, which affects the quality of fitting our models. Secondly, there are limitations within our models, since all models we used are quantitative models that are vulnerable when intrinsic assumptions are not satisfied. Thirdly, as mentioned, the forecasting results yield by dynamic forecasting are not that reliable as the one-step forecast, because the UK online retail market is changing rapidly with uncertainties. Some possible remedies are sourcing for more data, combining our forecasts with qualitative methods that can incorporate economic, political, regulatory, ethical issues; however, this might requires domain expertise regarding to online retailing industry.

**References:**

*Assigning weights to an averaged forecast*. (2016). Retrieved May 20, 2018, from <https://stats.stackexchange.com/questions/163074/assigning-weights-to-an-averaged-forecast>

Bowsher, E. (2018). *Online retail sales continue to soar*. Retrieved from:

<https://www.ft.com/content/a8f5c780-f46d-11e7-a4c9-bbdefa4f210b>

Chatfield, C., & Yar, M. (1988). *Holt-Winters Forecasting: Some Practical Issues. Journal of the Royal Statistical Society*. Series D (The Statistician), 37(2), 129-140. doi:10.2307/2348687

Doherty, N. F., Chadwick, F. E., Hart, C. A. (1999). *Cyber retailing in the UK: the potential of the Internet as a retail channel*. International Journal of Retail & Distribution Management, 27(1), 22-36.

Doherty, N. F., Chadwick, F. E., Hart, C. A. (2000). *Retailer adoption of the Internet: Implications for retail marketing*. European Journal of Marketing, 34(8), 954-974.

Gelper, S. , Fried, R. & Croux, C. (2010). *Robust forecasting with exponential and Holt–Winters smoothing*. J. Forecast., 29, 285-300.

Hyndman, R., & Athanasopoulos, G. (2014). *Forecasting: principles and practice.* Retrieved from: <https://www.otexts.org/fpp>

Meyler, A., Kenny, G., Quinn, T. (1998). *Forecasting irish inflation using ARIMA models*. Retrieved from: <https://mpra.ub.uni-muenchen.de/11359/>

Office for National Statistics (2018). *Internet retail sales, £ millions. All retailing.* Retrieved from:

<https://www.ons.gov.uk/businessindustryandtrade/retailindustry/timeseries/je2j/drsi>

Pantano, E., Nguyen, B., Dennis, C., Merrilees, B., & Gerlach, S. (2004). *Internet Retailing and Future Perspectives* (2nd ed.). New York, NY: Routledge.

Reynolds, J. (2002). *Charting the multi-channel future:Retail choices and constraints*, International Journal of Retail, and Distribution Management, 30 (11), 530–535.

*scipy.optimize.brute*. (n.d.). Retrieved May 16, 2018, from

<https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.brute.html>

Timmermann, A. (2013, July-August). *Forecasting Combinations* [PowerPoint presentation]. Retrieved from: <http://www.oxford-man.ox.ac.uk/sites/default/files/events/combination_Sofie.pdf>

Winters, P. R. (1960). *Forecasting sales by exponentially weighted moving averages*, Management Science, 6, 324-342.

Zaiyong Tang, Paul A. Fishwick. (1993). *Feedforward Neural Nets as Models for Time Series Forecasting*. INFORMS Journal on Computing, 374-385

# **Appendices**

Appendix – Code



**TEAM TASK MEETING AGENDA**

|  |
| --- |
| **TEAM MEETING AGENDA**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Project Group 110\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  Meeting to be held \_\_\_\_\_\_\_\_\_Law Library Group Study Room M108\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_May 7, 2018\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Date)  \_\_\_\_\_\_\_\_\_\_\_\_\_14.00pm \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Time)  Chairperson: \_\_\_\_\_\_\_\_Di Xu\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Minute-Taker: \_\_\_\_\_\_\_Chen Chen\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 1. Attendance: All attended 2. Discussion Point and Actions Decided   (1) Go through the requirements of the assignment  (2) Discuss the expectation from the professor  (3) Set up a plan for completing each part  (4) Allocate tasks to each individual  (5) Discuss any remaining issues  (6) Determine the next meeting time and date   1. Any other business – No 2. Next meeting – May 11, 2018 |



**TEAM TASK MEETING AGENDA**

|  |
| --- |
| **TEAM MEETING AGENDA**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Project Group 110\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  Meeting to be held \_\_\_\_\_\_\_\_\_Law Library Group Study Room M104\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_May 11, 2018\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Date)  \_\_\_\_\_\_\_\_\_\_\_\_\_14.00pm \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Time)  Chairperson: \_\_\_\_\_\_\_\_\_Manjing Fang \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Minute-Taker: \_\_\_\_\_\_\_\_\_Chen Chen\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 1. Attendance: All attended 2. Discussion Point   (1) Everyone presents his / her work  (2) Discussion  (3) Decide on the dataset  (4) Update new thoughts of the assignment requirements  (5) Update our plan and work allocations  (6) Determine the next meeting time and date   1. Any other business – No 2. Next meeting – May 18, 2018 |



**TEAM TASK MEETING AGENDA**

|  |
| --- |
| **TEAM MEETING AGENDA**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Project Group 110\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  Meeting to be held \_\_\_\_\_\_\_\_\_Law Library Group Study Room M108\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_May 18, 2018\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Date)  \_\_\_\_\_\_\_\_\_\_\_\_\_14.00pm \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Time)  Chairperson: \_\_\_\_\_\_\_\_\_ Danlu Liang \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Minute-Taker: \_\_\_\_\_\_\_\_\_Chen Chen\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 1. Attendance: All attended 2. Discussion Point   (1) Everyone presents his / her work  (2) Discussion  (3) Decide on specific issues (how to set parameters, etc.)  (4) Determine the next meeting time and date   1. Any other business – No 2. Next meeting – May 25, 2018 |

**Appendix**

**Code**

**EDA, Seasonal ARIMA model and Benchmark models**

external libraries

'''

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import statsmodels.api as smapi

import statsmodels.tsa.api as smt

from statsmodels.tsa.arima\_model import ARIMA

'''

read data

'''

#%%

data = pd.read\_excel('uk\_internet\_retail\_sales.xlsx', dayfirst = True, parse\_data = [0])

sales = data['Sales']

date = data['Date']

data.set\_index("Date", inplace=True)

'''

plot data

'''

fig = plt.figure()

plt.plot(date, sales)

plt.title('UK Internet Retail Sales')

plt.xlabel('Time')

plt.ylabel('Sales (£ millions)')

#fig.savefig('original\_data.pdf')

'''

split training data & test data

'''

train = sales.iloc[:-12]

test = sales.iloc[-12:]

#%%

# ACF of original data

smt.graphics.plot\_acf(sales, lags=36)

plt.title("ACF: Original data")

#%%

# EDA seasonal decomposition

decomp\_obj=smapi.tsa.seasonal\_decompose(data.iloc[:-12],model="multiplicative")

decomp\_obj.plot()

# log transformation

train\_log= np.log(train.values)

sales\_log= np.log(sales.values)

fig = plt.figure()

plt.plot(date.iloc[:-12], train\_log)

plt.title('Log of UK Internet Retail Sales')

plt.xlabel('Time')

plt.ylabel('Log Sales')

# seasonal differencing

# referrencing lecture example

train\_ds=train\_log[12:]-train\_log[:-12]

train\_d=train\_ds[1:]-train\_ds[:-1]

plt.figure(figsize=(10,8))

plt.subplot(211)

plt.plot(train\_ds)

plt.title('Seasonally Differenced Transformed Monthly Internet Retail Sales')

plt.subplot(212)

plt.plot(train\_d)

plt.title('Regular Difference of the Seasonal Differenced')

# Plot ACF and PACF for double differenced data

smt.graphics.plot\_acf(train\_d, lags=36)

plt.title("ACF: Double differenced log series")

smt.graphics.plot\_pacf(train\_d, lags=36)

plt.title("PACF: Double differenced log series")

#%%

# Seasonal naive, drift and combination

# seasonal naive

sn=train[-12:]

# Drift method

d=list()

train\_v=train.values

for h in range(len(test)):

d.append(train\_v[-1]+h\*((train\_v[-1]-train\_v[1])/(len(train\_v)-1)))

# one-step ahead drift

d\_o=list()

train\_vo=train.tolist()

test\_v=test.tolist()

# first forecast is based on yT-y1

# second forecast is based on y\_T+1, y1

for h in range(len(test)):

train\_vo.append(test\_v[h])

d\_o.append(train\_vo[-2]+1\*((train\_vo[-2]-train\_vo[1])/(len(train\_vo)-2)))

plt.figure(figsize=(12,8))

plt.plot(date,sales,label="Original data")

plt.plot(date[-12:],sn,label="12 Months Forecast--Seasonal Naive")

plt.legend()

plt.title('Benchmark forecast--Seasonal Naive')

plt.xlabel('Months')

plt.ylabel('Sales')

plt.figure(figsize=(12,8))

plt.plot(date,sales,label="Original data")

plt.plot(date[-12:],d,label="12 Months Forecast--Drift")

plt.plot(date[-12:],d\_o,label="12 Months Forecast--Drift(one-step)")

plt.legend()

plt.title('Benchmark forecast--Drift')

plt.xlabel('Months')

plt.ylabel('Sales')

#%%

def criteria(targets, predictions):

me=(targets - predictions).mean()

mad=np.absolute(targets - predictions).mean()

rmse=np.sqrt(((targets - predictions)\*\*2).mean())

mape=100\*(np.absolute(targets-predictions)/targets).mean()

results=[me,mad,rmse,mape]

return results

result\_sn=criteria(test.values,sn.values)

result\_drift=criteria(test.values,d)

result\_drift\_one=criteria(test.values,d\_o)

#%%

# fit seasonal arima model

seasonal\_arima = smapi.tsa.statespace.SARIMAX(train\_log, trend='n', order=[2,1,1], seasonal\_order=[0,1,1,12])

results = seasonal\_arima.fit()

print(results.summary())

seasonal\_forecast = results.predict(start=train\_log.shape[0],end=train\_log.shape[0]+11,dynamic=True)

# transform back

forecast=np.exp(seasonal\_forecast)

# Plot forecasts and original data

plt.figure()

plt.plot(date,sales,label="Original data",color='b')

plt.plot(date[-12:],forecast,label="12 Months Forecast",color='r')

plt.legend()

plt.title('Seasonal ARIMA forecast')

plt.xlabel('Months')

plt.ylabel('Sales')

#%%

# Brute force to fit the model

from scipy.optimize import brute

def optfunc(order\_full, endog):

converted\_order = [int(x) for x in order\_full]

try:

fit = smapi.tsa.statespace.SARIMAX(endog, trend='n', order=order\_full[:3], seasonal\_order=converted\_order[3:]).fit()

return fit.bic

except:

return np.inf

grid = (slice(0, 4, 1), slice(1,2,1), slice(0, 4, 1), slice(0, 4, 1), slice(1,2,1), slice(0, 4, 1), slice(12, 13, 1))

optimal = brute(optfunc, grid, args=(train\_log,), finish=None)

converted\_order = [int(x) for x in optimal]

optimal\_seasonal\_model = smapi.tsa.statespace.SARIMAX(train\_log, trend='n', order=converted\_order[:3], seasonal\_order=converted\_order[3:],)

results = optimal\_seasonal\_model.fit()

print(results.summary())

seasonal\_forecast\_optimal = results.predict(start=train\_log.shape[0],end=train\_log.shape[0]+11,dynamic=True)

# transform back

forecast\_optimal=np.exp(seasonal\_forecast\_optimal)

# Plot forecasts and original data

plt.figure()

plt.plot(date,sales,label="Original data",color='b')

plt.plot(date[-12:],forecast\_optimal,label="12 Months Forecast",color='r')

plt.legend()

plt.title('Optimal Seasonal ARIMA forecast')

plt.xlabel('Months')

plt.ylabel('Sales')

#%%

# calculate evaluation criteria

result\_sarima=criteria(test.values,forecast)

result\_opsarima=criteria(test.values,forecast\_optimal)

#%%

# one-step ahead forecasting with updated information

# fit seasonal arima model

seasonal\_f=list()

test\_log=np.log(test.values)

for i in range(len(test)):

seasonal\_arima = smapi.tsa.statespace.SARIMAX(sales\_log[0:125+i], trend='n', order=[2,1,1], seasonal\_order=[0,1,1,12])

results = seasonal\_arima.fit()

start\_index=len(train)+i

end\_index=len(train)+i

r\_one\_step = results.predict(start=start\_index,end=end\_index, dynamic=False)

seasonal\_f.extend(r\_one\_step.tolist())

# transform back

forecast\_one\_step=np.exp(seasonal\_f)

#%%

plt.figure()

plt.plot(date,sales,label="Original data",color='b')

plt.plot(date[-12:],forecast\_one\_step,label="12 Months Forecast",color='r')

plt.legend()

plt.title('Seasonal ARIMA forecast')

plt.xlabel('Months')

plt.ylabel('Sales')

#%%

#one-step for optimal SARIMA(1,1,1)(0,1,1)12

seasonal\_f\_o=list()

test\_log=np.log(test.values)

for i in range(len(test)):

seasonal\_arima\_o = smapi.tsa.statespace.SARIMAX(sales\_log[0:125+i], trend='n', order=[1,1,1], seasonal\_order=[0,1,1,12])

results = seasonal\_arima\_o.fit()

start\_index\_o=len(train)+i

end\_index\_o=len(train)+i

r\_one\_step\_o = results.predict(start=start\_index\_o,end=end\_index\_o, dynamic=False)

seasonal\_f\_o.extend(r\_one\_step\_o.tolist())

# transform back

forecast\_one\_step\_o=np.exp(seasonal\_f\_o)

plt.figure()

plt.plot(date,sales,label="Original data",color='b')

plt.plot(date[-12:],forecast\_one\_step\_o,label="12 Months Forecast",color='r')

plt.legend()

plt.title('Seasonal ARIMA forecast')

plt.xlabel('Months')

plt.ylabel('Sales')

#%%

# calculate evaluation criteria

result\_sarima\_one\_step=criteria(test.values,forecast\_one\_step)

result\_opsarima\_one\_step=criteria(test.values,forecast\_one\_step\_o)

error\_dynamic=test.values-forecast\_optimal

error\_one\_step=test.values-forecast\_one\_step\_o

# Export errors to excel

columns=['Dynamic','One-step']

forecast\_error=np.transpose(np.vstack((error\_dynamic,error\_one\_step)))

forecast\_error\_sarima=pd.DataFrame(forecast\_error, columns=columns)

forecast\_error\_sarima.to\_excel('error\_sarima.xlsx')

forecast\_sarima=np.transpose(np.vstack((forecast\_optimal,forecast\_one\_step\_o)))

forecast\_sarima=pd.DataFrame(forecast\_sarima, columns=columns)

forecast\_sarima.to\_excel('forecast\_sarima.xlsx')

**Holt-Winters:**

'''

external libraries

'''

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import holtwinters as hw

'''

read data

'''

data = pd.read\_excel('uk\_internet\_retail\_sales.xlsx', dayfirst = True, parse\_data = [0])

sales = data['Sales']

date = data['Date']

'''

EDA - 1: plot original series

'''

fig = plt.figure()

plt.plot(date, sales)

plt.title('UK Internet Retail Sales')

plt.xlabel('Time')

plt.ylabel('Sales (£ millions)')

#fig.savefig('original\_data.pdf')

'''

split training data & test data

'''

train = sales.iloc[:-12]

test = sales.iloc[-12:]

train\_date = date.iloc[:-12]

test\_date = date.iloc[-12:]

'''

EDA - 2: calculate initial trend-cycle estimation by moving average

'''

#do a 2 \* 12 MA (by chaining a 2-MA and a 12-MA)

T = sales.rolling(2, center = True).mean().rolling(12, center = True).mean()

T = T.shift(-1) #shift to make it balance

#plot the initial trend estimate

plt.figure()

plt.plot(date[6:-6], T[6:-6])

plt.title('CMA-12 Smoothing of Original Series')

plt.xlabel('Time')

plt.ylabel('Sales (£ millions)')

'''

EDA - 3: calculate seasonal components

'''

#subtract the trend from the actual data, which will leave us with the seasonal components

S = sales / T

#plot to verify

plt.figure()

plt.plot(date[6:-6], S[6:-6])

plt.title('Seasonal multiplicative - UK Internet Retail Sales')

plt.xlabel('Time')

plt.ylabel('Sales (£ millions)')

'''

EDA - 4: calculate seasonally adjusted data (did no do normalisation, can be improved)

'''

#calculate the average for each month

monthly\_S = np.reshape(S[:-5], (11,12)) #start from November, end in October

monthly\_avg = np.nanmean(monthly\_S, axis = 0)

#repeat the average for 5 times(years)

tiled\_avg = np.tile(monthly\_avg, 12)[:-7]

#subtract the seasonal average from the original data

seasonally\_adjusted = sales / tiled\_avg

#plot the seasonally adjusted data

plt.figure()

plt.plot(date, seasonally\_adjusted)

plt.title('Seasonally adjusted - UK Internet Retail Sales')

plt.xlabel('Time')

plt.ylabel('Sales (£ millions)')

'''

EDA - 5: plot adjusted seasonal components

'''

plt.figure()

plt.plot(date, tiled\_avg)

plt.title('Adjusted Seasonal Component - UK Internet Retail Sales')

plt.xlabel('Time')

plt.ylabel('Sales (£ millions)')

'''

EDA - 6: calculate final trend estimation

'''

T\_final = seasonally\_adjusted.rolling(2, center = True).mean().rolling(12, center = True).mean()

T\_final = T\_final.shift(-1)

plt.figure()

plt.plot(date[6:-6], T\_final[6:-6])

plt.title('Trend Component - UK Internet Retail Sales')

plt.xlabel('Time')

plt.ylabel('Sales (£ millions)')

'''

EDA - 7: residual plot

'''

residual = sales / (tiled\_avg \* T\_final)

plt.figure()

plt.plot(date[6:-6], residual[6:-6])

plt.title('Residuals - UK Internet Retail Sales')

plt.xlabel('Time')

plt.ylabel('Sales (£ millions)')

'''

EDA - 8: plot'em together

'''

plt\_1 = plt.subplot(4,1,1)

plt.plot(date[6:-6], sales[6:-6])

plt.ylabel('Original Series')

plt\_1.xaxis.set\_major\_formatter(plt.NullFormatter())

plt.title('Original Series and its Components')

plt\_2 = plt.subplot(4,1,2)

plt.plot(date[6:-6], T\_final[6:-6])

plt.ylabel('Trend')

plt\_2.xaxis.set\_major\_formatter(plt.NullFormatter())

plt\_3 = plt.subplot(4,1,3)

plt.plot(date[6:-6], tiled\_avg[6:-6])

plt.ylabel('Seasonal')

plt\_3.xaxis.set\_major\_formatter(plt.NullFormatter())

plt.subplot(4,1,4)

plt.plot(date[6:-6], residual[6:-6])

plt.ylabel('Residual')

plt.xlabel('Time')

'''

3) Produce a forecast using the Holt-Winters Library

'''

'''

3.1) Forecast horizon = 12, fixed origin (dynamic forecast)

'''

#use holtwinters package to get forecasts

smoothed\_1, forecast\_1, alpha\_1, beta\_1, gamma\_1, rmse\_1 = hw.multiplicative(train.tolist(), fc=12, m=12)

#plot the original traing and test set data (called: Original)

#compare it with only the forecasts (called: Holt-Winter Multiplicative)

#note the 12-month validation period is plotted twice to compare and see the errors

plt.figure()

plt.plot(date, sales, label="Original")

plt.plot(test\_date, forecast\_1, label="Holt-Winter Multiplicative")

plt.legend()

plt.title('Holt-Winter Smoothing - Multiplicative (Dynamic Forecast)')

plt.xlabel('Time')

plt.ylabel('Sales (£ millions)')

#compute the forecasting measures (RMSE, MAD, MAPE, ME)

validation = np.asarray(test)

forecast = np.asarray(forecast\_1)

error\_hw1 = validation - forecast

RMSE\_hw1 = np.sqrt(np.mean(np.power(error\_hw1, 2)))

MAD\_hw1 = np.mean(np.abs(error\_hw1))

MAPE\_hw1 = np.mean(((np.abs(error\_hw1))/validation)\*100)

ME\_hw1 = np.mean(error\_hw1)

'''

3.2) Forecast horizon = 1, moving origin (1-step ahead forecast \* conducted 12 times)

'''

#iterate 12 times, each time using holtwinters package to get one-step ahead forecast

forecast\_list = []

for j in range(12):

train = sales.iloc[:-12+j] #expanding window size

smoothed\_2, forecast\_2, alpha\_2, beta\_2, gamma\_2, rmse\_2 = hw.multiplicative(train.tolist(), fc=1, m=12)

forecast\_list.append(forecast\_2)

#plot the original traing and test set data (called: Origianl)

#compare it with only the forecasts (called: Holt-Winter Multiplicative)

#note the 12-month validation period is plotted twice to compare and see the errors

plt.figure()

plt.plot(date, sales, label="Original")

plt.plot(test\_date, forecast\_list, label="Holt-Winter Multiplicative")

plt.legend()

plt.title('Holt-Winter Smoothing - Multiplicative (One-Step Ahead Forecast)')

plt.xlabel('Time')

plt.ylabel('Sales (£ millions)')

#compute the forecasting measures (RMSE, MAD, MAPE, ME)

validation\_new = validation

forecast\_series = pd.Series((v[0] for v in forecast\_list))

forecast\_new = np.asarray(forecast\_series)

error\_hw2 = validation\_new - forecast\_new

RMSE\_hw2 = np.sqrt(np.mean(np.power(error\_hw2, 2)))

MAD\_hw2 = np.mean(np.abs(error\_hw2))

MAPE\_hw2 = np.mean(((np.abs(error\_hw2))/validation\_new)\*100)

ME\_hw2 = np.mean(error\_hw2)

#%%

# forecast the next one month

smoothed\_3, forecast\_3, alpha\_3, beta\_3, gamma\_3, rmse\_3 = hw.multiplicative(sales.tolist(), fc=1, m=12)

forecast\_one\_month=forecast\_3

# forecast the next twelve month

smoothed\_4, forecast\_4, alpha\_4, beta\_4, gamma\_4, rmse\_4 = hw.multiplicative(sales.tolist(), fc=12, m=12)

forecast\_12month=forecast\_4

#%%

# Export errors to excel

columns=['Dynamic','One-step']

forecast\_error=np.transpose(np.vstack((error\_hw1,error\_hw2)))

forecast\_error\_hw=pd.DataFrame(forecast\_error, columns=columns)

forecast\_error\_hw.to\_excel('error\_hw.xlsx')

forecast\_hw=np.transpose(np.vstack((forecast,forecast\_new)))

forecast\_hw=pd.DataFrame(forecast\_hw,columns=columns)

forecast\_hw.to\_excel('forecast\_hw.xlsx')

**Long Short Term Memory (from jupyter notebook)**

**Experiment**

# coding: utf-8

# # <center>Assignment 02 Neural Network

# In[32]:

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

import time

import math

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error

from keras.layers.core import Dense

from keras.models import Sequential

# In[20]:

data=pd.read\_csv('internet\_retail\_sales.csv',parse\_dates=[0], index\_col=0)

sales=data.values

# In[21]:

np.random.seed(1)

# ## Data Preparation

# ### Standardisation

# In[22]:

# Standardisation

scaler = StandardScaler()

sales\_scaled=scaler.fit\_transform(sales)

# ### Fixed Data Window

# In[60]:

# Creating Training and Test feature

time\_window=20

Xall, yall=[],[]

for i in range(time\_window, len(sales\_scaled)):

Xall.append(sales\_scaled[i-time\_window:i,0])

yall.append(sales\_scaled[i,0])

Xall=np.array(Xall)

yall=np.array(yall)

# ### Train Test Split

# In[61]:

# Train Test Split

test\_size=12

Xtrain, Xtest=Xall[0:-test\_size], Xall[-test\_size:]

ytrain, ytest=yall[0:-test\_size], yall[-test\_size:]

# In[62]:

y\_train\_raw=sales[0:-test\_size]

y\_test\_raw=sales[-test\_size:]

# ## Feed Forward Neural Network Model Fitting

# In[74]:

neurons=100

epochs=1000

#%% NN Define the Feed Forward NN model

model=Sequential()

model.add(Dense(neurons,input\_dim=time\_window,activation='tanh'))

model.add(Dense(1))

#%% NN Compile

model.compile(loss='mean\_squared\_error',optimizer='adam')

#model.fit(Xtrain,Ytrain,epochs=500,batch\_size=10,verbose=2,validation\_split=0.05)

history=model.fit(Xtrain,ytrain,batch\_size=10,nb\_epoch=epochs,validation\_split=0.1)

# In[75]:

# Predict

train\_predict = model.predict(Xtrain,batch\_size=1)

test\_predict = model.predict(Xtest,batch\_size=1)

# Inverse Prediction

train\_predict=scaler.inverse\_transform(train\_predict)

test\_predict=scaler.inverse\_transform(test\_predict)

# ### Training Score

# In[76]:

# epochs\_summary=[]

# neurons\_summary=[]

# rmse\_summary=[]

# window\_summary=[]

# In[77]:

rmse=math.sqrt(mean\_squared\_error(y\_test\_raw,test\_predict))

# In[78]:

epochs\_summary.append(epochs)

neurons\_summary.append(neurons)

rmse\_summary.append(rmse)

window\_summary.append(time\_window)

print(epochs\_summary)

print(neurons\_summary)

print(window\_summary)

print(rmse\_summary)

# ## Diagnostic Plot

# In[79]:

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model train vs validation loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'validation'], loc='upper right')

plt.show()

**Experiment 01**

# coding: utf-8

# # <center>Assignment 2 LSTM - Fixed Window Size Loop with Adjusted-Seasonality Data

# In[77]:

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

# In[78]:

# Basic Time Series Setting

import statsmodels.api as smapi

import statsmodels.tsa.api as smt

# In[79]:

# RNN Package Setting

import time

import math

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

from keras.layers.core import Dense, Activation, Dropout

from keras.layers.recurrent import LSTM

from keras.models import Sequential

from keras import optimizers

np.random.seed(1)

# ## Data Preparation

# In[80]:

data=pd.read\_csv('internet\_retail\_sales.csv',parse\_dates=[0], index\_col=0)

sales=data.values

# ### First Order Difference

# In[81]:

sales\_diff=pd.Series.diff(data)

sales\_diff=sales\_diff.dropna().values

# sales\_diff

# ### Seosonal Decomposition

# In[93]:

fig=plt.figure()

date=pd.date\_range(start='11/10/2005',end='03/01/2017',freq='MS')

plt.plot(date,sales\_trend[:-1])

plt.xlabel('year')

plt.ylabel('pounds (Million)')

plt.title('Seasonally Adjusted Data of Internet Retail Sales')

plt.show()

fig.savefig('seasonallyadj.png',dpi=1000)

# In[91]:

fig=plt.figure()

date=pd.date\_range(start='11/10/2005',end='03/01/2017',freq='MS')

plt.plot(date,sales\_diff)

plt.xlabel('year')

plt.ylabel('pounds (Million)')

plt.title('First Order Difference Data of Internet Retail Sales')

plt.show()

fig.savefig('1stord.png',dpi=1000)

# In[85]:

decomp\_obj = smapi.tsa.seasonal\_decompose(data['Sales'],model='multiplicative')

# decomp\_obj.plot()

sales\_sea=decomp\_obj.seasonal

# sales\_trend=sales/sales\_sea

sales\_trend=np.divide(np.array(sales),pd.DataFrame(sales\_sea).values)

sales\_season=pd.DataFrame(sales\_sea).values

seasonality=sales\_season[-12:]

# In[8]:

len(seasonality)

# In[9]:

plt.figure()

plt.plot(seasonality)

plt.show()

# ### Standardisation - Trend Standardisation

# In[17]:

# Standardisation

scaler = StandardScaler()

trend\_scaled=scaler.fit\_transform(sales\_trend)

# Std & Mean Calculation

trend\_std=np.std(sales\_trend)

trend\_mean=np.mean(sales\_trend)

print(trend\_std)

print(trend\_mean)

# ## RNN-Fixed Window Size Model Fitting

# ### Trend Model Fitting

# In[34]:

# Model Fitting

epochs=50

repeats=10

neurons\_min=1

neurons\_max=2

window\_min=1

window\_max=2

step=1

# train\_loss=np.zeros(shape=(window\_max-window\_min,int((neurons\_max-neurons\_min)/5)))

# cv\_loss=np.zeros(shape=(window\_max-window\_min,int((neurons\_max-neurons\_min)/5)))

# test\_loss=[np.zeros(shape=(window\_max-window\_min,int((neurons\_max-neurons\_min)/5)))]

train\_loss=[]

train\_loss\_res=[]

cv\_loss=[]

cv\_loss\_res=[]

test\_loss\_trend=[]

test\_loss\_multi=[]

for w in range(window\_min,window\_max):

time\_window=w

Xall, yall=[],[]

for i in range(time\_window, len(trend\_scaled)):

Xall.append(trend\_scaled[i-time\_window:i,0])

yall.append(trend\_scaled[i,0])

Xall=np.array(Xall)

yall=np.array(yall)

# Real Train Test Split

test\_size=12

Xtrain, Xtest=Xall[0:-test\_size], Xall[-test\_size:]

ytrain, ytest=yall[0:-test\_size], yall[-test\_size:]

# Unscaled Data Preparation for Evaluation

# Trend

y\_train\_raw=sales\_trend[0:-test\_size]

y\_test\_raw=sales\_trend[-test\_size:]

# Trend+Seasonality

y\_train\_raw\_multi=sales[0:-test\_size]

y\_test\_raw\_multi=sales[-test\_size:]

# Validation Train Test Split

Xtrain\_pse, Xtest\_pse=Xtrain[0:-test\_size], Xtrain[-test\_size:]

ytrain\_pse, ytest\_pse=ytrain[0:-test\_size], ytrain[-test\_size:]

# Unscaled Data Preparation for Evaluation

# Trend

y\_train\_pse\_raw=y\_train\_raw[0:-test\_size]

y\_test\_pse\_raw=y\_train\_raw[-test\_size:]

# Trend+Seasonality

y\_train\_pse\_raw\_multi=y\_train\_raw\_multi[0:-test\_size]

y\_test\_pse\_raw\_multi=y\_train\_raw\_multi[-test\_size:]

# Real Train Test Split Reshape

Xtrain, Xtest=Xtrain.reshape(Xtrain.shape[0], time\_window, 1), Xtest.reshape(Xtest.shape[0], time\_window, 1)

# Validation Train Test Split Reshape

Xtrain\_pse, Xtest\_pse=Xtrain\_pse.reshape(Xtrain\_pse.shape[0], time\_window, 1), Xtest\_pse.reshape(Xtest\_pse.shape[0], time\_window, 1)

for n in range(neurons\_min,neurons\_max,step):

# fig=plt.figure()

rmse\_t\_trend=[]

rmse\_t\_multi=[]

for j in range(repeats):

model=Sequential()

model.add(LSTM(n,input\_shape = (time\_window,1), batch\_size=1,stateful=True))

model.add(Dropout(0.2))

model.add(Dense(1,activation='linear'))

# model.add(Dense(1))

# compile model for use

start=time.time()

model.compile(loss="mse", optimizer="rmsprop")

print("Compilation Time : ", time.time() - start)

train\_loss\_t=[]

cv\_loss\_t=[]

for i in range(epochs):

history=model.fit(Xtrain\_pse,ytrain\_pse, epochs=1, batch\_size=1, verbose=2,validation\_split=0.1,shuffle=False)

train\_loss\_t.append(history.history['loss'])

cv\_loss\_t.append(history.history['val\_loss'])

model.reset\_states()

# Loss Calculation

# Test RMSE - For my own happiness

test\_predict\_pse = model.predict(Xtest\_pse,batch\_size=1)

test\_predict\_pse=scaler.inverse\_transform(test\_predict\_pse)

test\_predict\_pse\_multi=np.multiply(np.array(test\_predict\_pse),seasonality)

# Trend RMSE

rmse\_t\_trend.append(math.sqrt(mean\_squared\_error(y\_test\_pse\_raw,test\_predict\_pse)))

# Trend+Seasonality RMSE

rmse\_t\_multi.append(math.sqrt(mean\_squared\_error(y\_test\_pse\_raw\_multi,test\_predict\_pse\_multi)))

# cross validation loss

cv\_loss.append(np.array(cv\_loss\_t))

cv\_loss\_res.append(np.array(cv\_loss\_t)\*trend\_std)

# training loss

train\_loss.append(np.array(train\_loss\_t))

train\_loss\_res.append(np.array(train\_loss\_t)\*trend\_std)

# TREND

rmse\_t\_trend.append(np.mean(rmse\_t\_trend)) # calculate mean and std of Trend RMSE

rmse\_t\_trend.append(np.std(rmse\_t\_trend))

test\_loss\_trend.append(rmse\_t\_trend) # append mean and std of rmse

# TREND+SEASONALITY RMSE

rmse\_t\_multi.append(np.mean(rmse\_t\_multi)) # calculate mean and std of Trend RMSE

rmse\_t\_multi.append(np.std(rmse\_t\_multi))

test\_loss\_multi.append(rmse\_t\_multi) # append mean and std of rmse

# plt.plot(train\_loss,color='blue')

# plt.plot(cv\_loss,color='orange')

# plt.title('model train vs validation loss')

# plt.ylabel('loss')

# plt.xlabel('epoch')

# # # plt.legend(['train', 'validation'], loc='upper right')

# plt.show()

# fig.savefig('epochs\_tuning.png',dpi=1000)

# In[35]:

print(test\_loss\_trend)

print(test\_loss\_multi)

# In[27]:

# test\_loss\_trend\_sum=[]

# test\_loss\_multi\_sum=[]

test\_loss\_trend\_sum.append(np.array(test\_loss\_trend))

test\_loss\_multi\_sum.append(np.array(test\_loss\_multi))

# In[28]:

print(test\_loss\_multi\_sum)

# In[46]:

cv\_loss

# In[47]:

cv\_loss\_res

# In[48]:

train\_loss\_res

# In[36]:

# Cross Validation Visualization

fig=plt.figure()

line1=plt.plot(np.array(train\_loss)[0,:],color='blue',label="Train Loss")

plt.plot(np.array(train\_loss)[1,:],color='blue')

plt.plot(np.array(train\_loss)[2,:],color='blue')

plt.plot(np.array(train\_loss)[3,:],color='blue')

plt.plot(np.array(train\_loss)[4,:],color='blue')

plt.plot(np.array(train\_loss)[5,:],color='blue')

plt.plot(np.array(train\_loss)[6,:],color='blue')

plt.plot(np.array(train\_loss)[7,:],color='blue')

plt.plot(np.array(train\_loss)[8,:],color='blue')

plt.plot(np.array(train\_loss)[9,:],color='blue')

line2=plt.plot(np.array(cv\_loss)[0,:],color='orange',label= "Cross Validation Loss")

plt.plot(np.array(cv\_loss)[1,:],color='orange')

plt.plot(np.array(cv\_loss)[2,:],color='orange')

plt.plot(np.array(cv\_loss)[3,:],color='orange')

plt.plot(np.array(cv\_loss)[4,:],color='orange')

plt.plot(np.array(cv\_loss)[5,:],color='orange')

plt.plot(np.array(cv\_loss)[6,:],color='orange')

plt.plot(np.array(cv\_loss)[7,:],color='orange')

plt.plot(np.array(cv\_loss)[8,:],color='orange')

plt.plot(np.array(cv\_loss)[9,:],color='orange')

plt.legend()

plt.xlabel('Epochs')

plt.ylabel('RMSE(Scaled)')

plt.title('Train Loss & Cross Validation Loss')

plt.show()

fig.savefig('loss\_output.png',dpi=1000)

# In[268]:

fig=plt.figure()

plt.plot(np.arange(40,100,5),test\_loss.iloc[0,:],label='time\_window=10,linear activation',color='blue')

plt.plot(np.arange(40,100,5),test\_loss.iloc[1,:],label='time\_window=11,linear activation',color='red')

plt.plot(np.arange(40,100,5),test\_loss.iloc[2,:],label='time\_window=12,linear activation',color='yellow')

plt.plot(np.arange(40,100,5),test\_loss.iloc[3,:],label='time\_window=13,linear activation',color='orange')

plt.plot(np.arange(40,100,5),test\_loss.iloc[4,:],label='time\_window=14,linear activation',color='green')

plt.legend()

plt.xlabel('neurons')

plt.ylabel('loss')

plt.title('One Step Ahead Loss')

plt.show()

fig.savefig('loss\_output.png',dpi=1000)

# In[269]:

test\_loss.to\_excel('Loss\_T10-14\_N40-100.xlsx')

# In[200]:

# Predict

train\_predict = model.predict(Xtrain,batch\_size=1)

test\_predict = model.predict(Xtest,batch\_size=1)

# Inverse Prediction

train\_predict=scaler.inverse\_transform(train\_predict)

test\_predict=scaler.inverse\_transform(test\_predict)

# ## Training Score

# In[201]:

# epochs\_summary=[]

# neurons\_summary=[]

# rmse\_summary=[]

# mae\_summary=[]

# In[202]:

rmse=math.sqrt(mean\_squared\_error(y\_test\_raw,test\_predict))

mae=mean\_absolute\_error(y\_test\_raw,test\_predict)

# In[203]:

epochs\_summary.append(epochs)

neurons\_summary.append(neurons)

rmse\_summary.append(rmse)

mae\_summary.append(mae)

print(epochs\_summary)

print(neurons\_summary)

print(rmse\_summary)

print(mae\_summary)

# ## Diagnostic Plot

# In[101]:

fig=plt.figure()

plt.plot(train\_loss\_summary)

plt.plot(cv\_loss\_summary)

plt.title('model train vs validation loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'validation'], loc='upper right')

plt.show()

fig.savefig('epochs\_tuning.png',dpi=1000)

# In[102]:

train\_loss\_summary

# In[60]:

t3\_data=pd.read\_csv('test\_loss.csv')

t3\_data.head(10)

# In[75]:

ax=sns.boxplot(data=t3\_data, palette='Oranges')

ax.set\_title('Time Window 3, Epochs 100 Neurons Box Plot')

ax.set\_ylabel('RMSE Unit: Pounds (Million)')

ax.set\_xlabel('Neurons')

ax.figure.savefig('T3NeuronsBoxPlot.png',dpi=1000)

**Experiment 02:**

# coding: utf-8

# # <center>Assignment 2 LSTM - Fixed Window Size Loop with First Order Difference

# In[1]:

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

# In[2]:

# Basic Time Series Setting

import statsmodels.api as smapi

import statsmodels.tsa.api as smt

# In[3]:

# RNN Package Setting

import time

import math

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

from keras.layers.core import Dense, Activation, Dropout

from keras.layers.recurrent import LSTM

from keras.models import Sequential

from keras import optimizers

np.random.seed(1)

# ## Data Preparation

# In[4]:

data=pd.read\_csv('internet\_retail\_sales.csv',parse\_dates=[0], index\_col=0)

sales=data.values

# ### First Order Difference

# In[5]:

sales\_diff=pd.Series.diff(data)

sales\_diff=sales\_diff.dropna().values

# sales\_diff

# In[95]:

fig=plt.figure()

date=pd.date\_range(start='11/10/2005',end='03/01/2017',freq='MS')

plt.figure()

plt.plot(date,sales\_diff)

plt.show()

fig.savefig('1stord.png',dpi=1000)

# In[93]:

fig=smt.graphics.plot\_acf(sales\_diff, lags=24)

plt.show()

fig.savefig('ACF.png',dpi=1000)

# In[92]:

fig=smt.graphics.plot\_pacf(sales\_diff, lags=24)

plt.show()

fig.savefig('PACF.png',dpi=1000)

# ### Standardisation - Trend Standardisation

# In[6]:

# Standardisation

scaler = StandardScaler()

diff\_scaled=scaler.fit\_transform(sales\_diff)

# Std & Mean Calculation

diff\_std=np.std(sales\_diff)

diff\_mean=np.mean(sales\_diff)

print(diff\_std)

print(diff\_mean)

# ## RNN-Fixed Window Size Model Fitting

# ### Trend Model Fitting

# In[85]:

# Model Fitting

epochs=100

repeats=5

neurons\_min=30

neurons\_max=31

window\_min=2

window\_max=3

step=1

# train\_loss=np.zeros(shape=(window\_max-window\_min,int((neurons\_max-neurons\_min)/5)))

# cv\_loss=np.zeros(shape=(window\_max-window\_min,int((neurons\_max-neurons\_min)/5)))

# test\_loss=[np.zeros(shape=(window\_max-window\_min,int((neurons\_max-neurons\_min)/5)))]

train\_loss=[]

train\_loss\_res=[]

cv\_loss=[]

cv\_loss\_res=[]

test\_loss=[]

for w in range(window\_min,window\_max):

time\_window=w

Xall, yall=[],[]

for i in range(time\_window, len(diff\_scaled)):

Xall.append(diff\_scaled[i-time\_window:i,0])

yall.append(diff\_scaled[i,0])

Xall=np.array(Xall)

yall=np.array(yall)

# Real Train Test Split

test\_size=12

Xtrain, Xtest=Xall[0:-test\_size], Xall[-test\_size:]

ytrain, ytest=yall[0:-test\_size], yall[-test\_size:]

# Unscaled Data Preparation for Evaluation

y\_train\_raw=sales\_diff[0:-test\_size]

y\_test\_raw=sales\_diff[-test\_size:]

# Validation Train Test Split

Xtrain\_pse, Xtest\_pse=Xtrain[0:-test\_size], Xtrain[-test\_size:]

ytrain\_pse, ytest\_pse=ytrain[0:-test\_size], ytrain[-test\_size:]

# Unscaled Data Preparation for Evaluation

# Trend

y\_train\_pse\_raw=y\_train\_raw[0:-test\_size]

y\_test\_pse\_raw=y\_train\_raw[-test\_size:]

# Real Train Test Split Reshape

Xtrain, Xtest=Xtrain.reshape(Xtrain.shape[0], time\_window, 1), Xtest.reshape(Xtest.shape[0], time\_window, 1)

# Validation Train Test Split Reshape

Xtrain\_pse, Xtest\_pse=Xtrain\_pse.reshape(Xtrain\_pse.shape[0], time\_window, 1), Xtest\_pse.reshape(Xtest\_pse.shape[0], time\_window, 1)

for n in range(neurons\_min,neurons\_max,step):

# fig=plt.figure()

rmse\_t=[]

for j in range(repeats):

model=Sequential()

model.add(LSTM(n,input\_shape = (time\_window,1), batch\_size=1,stateful=True))

model.add(Dropout(0.2))

model.add(Dense(1,activation='linear'))

# model.add(Dense(1))

# compile model for use

start=time.time()

model.compile(loss="mse", optimizer="rmsprop")

print("Compilation Time : ", time.time() - start)

train\_loss\_t=[]

cv\_loss\_t=[]

for i in range(epochs):

history=model.fit(Xtrain\_pse,ytrain\_pse, epochs=1, batch\_size=1, verbose=2,validation\_split=0.1,shuffle=False)

train\_loss\_t.append(history.history['loss'])

cv\_loss\_t.append(history.history['val\_loss'])

model.reset\_states()

# Loss Calculation

# Test RMSE - For my own happiness

test\_predict\_pse = model.predict(Xtest\_pse,batch\_size=1)

test\_predict\_pse=scaler.inverse\_transform(test\_predict\_pse)

# Trend RMSE

rmse\_t.append(math.sqrt(mean\_squared\_error(y\_test\_pse\_raw,test\_predict\_pse)))

# cross validation loss

cv\_loss.append(np.array(cv\_loss\_t))

cv\_loss\_res.append(np.array(cv\_loss\_t)\*diff\_std)

# training loss

train\_loss.append(np.array(train\_loss\_t))

train\_loss\_res.append(np.array(train\_loss\_t)\*diff\_std)

# TREND

rmse\_t.append(np.mean(rmse\_t)) # calculate mean and std of Trend RMSE

rmse\_t.append(np.std(rmse\_t))

test\_loss.append(rmse\_t) # append mean and std of rmse

# plt.plot(train\_loss,color='blue')

# plt.plot(cv\_loss,color='orange')

# plt.title('model train vs validation loss')

# plt.ylabel('loss')

# plt.xlabel('epoch')

# # # plt.legend(['train', 'validation'], loc='upper right')

# plt.show()

# fig.savefig('epochs\_tuning.png',dpi=1000)

# In[87]:

test\_loss

# In[88]:

# test\_loss\_sum=[]

# test\_loss\_multi\_sum=[]

test\_loss\_sum.append(np.array(test\_loss))

# test\_loss\_multi\_sum.append(np.array(test\_loss\_multi))

# In[89]:

print(test\_loss\_sum)

# In[62]:

# cv\_loss

# In[63]:

# cv\_loss\_res

# In[65]:

# train\_loss\_res

# In[90]:

# Cross Validation Visualization

fig=plt.figure()

line1=plt.plot(np.array(train\_loss)[0,:],color='blue',label="Train Loss")

plt.plot(np.array(train\_loss)[1,:],color='blue')

plt.plot(np.array(train\_loss)[2,:],color='blue')

plt.plot(np.array(train\_loss)[3,:],color='blue')

plt.plot(np.array(train\_loss)[4,:],color='blue')

# plt.plot(np.array(train\_loss)[5,:],color='blue')

# plt.plot(np.array(train\_loss)[6,:],color='blue')

# plt.plot(np.array(train\_loss)[7,:],color='blue')

# plt.plot(np.array(train\_loss)[8,:],color='blue')

# plt.plot(np.array(train\_loss)[9,:],color='blue')

line2=plt.plot(np.array(cv\_loss)[0,:],color='orange',label= "Cross Validation Loss")

plt.plot(np.array(cv\_loss)[1,:],color='orange')

plt.plot(np.array(cv\_loss)[2,:],color='orange')

plt.plot(np.array(cv\_loss)[3,:],color='orange')

plt.plot(np.array(cv\_loss)[4,:],color='orange')

# plt.plot(np.array(cv\_loss)[5,:],color='orange')

# plt.plot(np.array(cv\_loss)[6,:],color='orange')

# plt.plot(np.array(cv\_loss)[7,:],color='orange')

# plt.plot(np.array(cv\_loss)[8,:],color='orange')

# plt.plot(np.array(cv\_loss)[9,:],color='orange')

plt.legend()

plt.xlabel('Epochs')

plt.ylabel('RMSE(Scaled)')

plt.title('Train Loss & Cross Validation Loss')

plt.show()

fig.savefig('loss\_output.png',dpi=1000)

# In[268]:

fig=plt.figure()

plt.plot(np.arange(40,100,5),test\_loss.iloc[0,:],label='time\_window=10,linear activation',color='blue')

plt.plot(np.arange(40,100,5),test\_loss.iloc[1,:],label='time\_window=11,linear activation',color='red')

plt.plot(np.arange(40,100,5),test\_loss.iloc[2,:],label='time\_window=12,linear activation',color='yellow')

plt.plot(np.arange(40,100,5),test\_loss.iloc[3,:],label='time\_window=13,linear activation',color='orange')

plt.plot(np.arange(40,100,5),test\_loss.iloc[4,:],label='time\_window=14,linear activation',color='green')

plt.legend()

plt.xlabel('neurons')

plt.ylabel('loss')

plt.title('One Step Ahead Loss')

plt.show()

fig.savefig('loss\_output.png',dpi=1000)

# In[269]:

test\_loss.to\_excel('Loss\_T10-14\_N40-100.xlsx')

# In[200]:

# Predict

train\_predict = model.predict(Xtrain,batch\_size=1)

test\_predict = model.predict(Xtest,batch\_size=1)

# Inverse Prediction

train\_predict=scaler.inverse\_transform(train\_predict)

test\_predict=scaler.inverse\_transform(test\_predict)

# ## Training Score

# In[201]:

# epochs\_summary=[]

# neurons\_summary=[]

# rmse\_summary=[]

# mae\_summary=[]

# In[202]:

rmse=math.sqrt(mean\_squared\_error(y\_test\_raw,test\_predict))

mae=mean\_absolute\_error(y\_test\_raw,test\_predict)

# In[203]:

epochs\_summary.append(epochs)

neurons\_summary.append(neurons)

rmse\_summary.append(rmse)

mae\_summary.append(mae)

print(epochs\_summary)

print(neurons\_summary)

print(rmse\_summary)

print(mae\_summary)

# ## Diagnostic Plot

# In[101]:

fig=plt.figure()

plt.plot(train\_loss\_summary)

plt.plot(cv\_loss\_summary)

plt.title('model train vs validation loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'validation'], loc='upper right')

plt.show()

fig.savefig('epochs\_tuning.png',dpi=1000)

# In[102]:

train\_loss\_summary

**Experiment 03:**

# coding: utf-8

# # <center>Assignment 2 LSTM - Fixed Window Size Loop with Seasonal Difference

#

# In[2]:

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

# In[3]:

# Basic Time Series Setting

import statsmodels.api as smapi

import statsmodels.tsa.api as smt

# In[4]:

# RNN Package Setting

import time

import math

import statsmodels.api as smapi

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

from keras.layers.core import Dense, Activation, Dropout

from keras.layers.recurrent import LSTM

from keras.models import Sequential

from keras import optimizers

np.random.seed(1)

# ## Data Preparation

# In[5]:

data=pd.read\_csv('internet\_retail\_sales.csv',parse\_dates=[0], index\_col=0)

sales=data.values

len(sales)

# ### First Order ACF & PACF Plot

# In[6]:

sales\_diff=pd.Series.diff(data)

sales\_diff=sales\_diff.dropna().values

# sales\_diff

# In[7]:

fig=smt.graphics.plot\_acf(sales\_diff, lags=24)

plt.show()

fig.savefig('ACF.png',dpi=1000)

# In[8]:

fig=smt.graphics.plot\_pacf(sales\_diff, lags=24)

plt.show()

fig.savefig('PACF.png',dpi=1000)

# ### Seasonal Difference

# In[10]:

# First Order Difference

# sales\_diff=pd.Series.diff(data)

# sales\_diff=sales\_diff.dropna().values

# Seasonally First Order Difference

season\_diff=pd.Series.diff(data,periods=12)

season\_diff=season\_diff.dropna().values

# In[11]:

plt.figure()

plt.plot(season\_diff)

plt.show()

# In[12]:

fig=smt.graphics.plot\_acf(season\_diff, lags=24)

plt.show()

fig.savefig('SACF.png',dpi=1000)

# In[13]:

fig=smt.graphics.plot\_pacf(season\_diff, lags=24)

plt.show()

fig.savefig('SPACF.png',dpi=1000)

# ### Standardisation - Trend Standardisation

# In[14]:

# Standardisation

scaler = StandardScaler()

diff\_scaled=scaler.fit\_transform(season\_diff)

# Std & Mean Calculation

diff\_std=np.std(season\_diff)

diff\_mean=np.mean(season\_diff)

print(diff\_std)

print(diff\_mean)

# ## RNN- Model Training

# ### Trend Model Fitting

# In[21]:

# Model Fitting

epochs=200

repeats=5

neurons\_min=1

neurons\_max=2

window\_min=2

window\_max=3

step=1

# train\_loss=np.zeros(shape=(window\_max-window\_min,int((neurons\_max-neurons\_min)/5)))

# cv\_loss=np.zeros(shape=(window\_max-window\_min,int((neurons\_max-neurons\_min)/5)))

# test\_loss=[np.zeros(shape=(window\_max-window\_min,int((neurons\_max-neurons\_min)/5)))]

train\_loss=[]

train\_loss\_res=[]

cv\_loss=[]

cv\_loss\_res=[]

test\_loss=[]

for w in range(window\_min,window\_max):

time\_window=w

Xall, yall=[],[]

for i in range(time\_window, len(diff\_scaled)):

Xall.append(diff\_scaled[i-time\_window:i,0])

yall.append(diff\_scaled[i,0])

Xall=np.array(Xall)

yall=np.array(yall)

# Real Train Test Split

test\_size=12

Xtrain, Xtest=Xall[0:-test\_size], Xall[-test\_size:]

ytrain, ytest=yall[0:-test\_size], yall[-test\_size:]

# Unscaled Data Preparation for Evaluation

y\_train\_raw=season\_diff[0:-test\_size]

y\_test\_raw=season\_diff[-test\_size:]

# Validation Train Test Split

Xtrain\_pse, Xtest\_pse=Xtrain[0:-test\_size], Xtrain[-test\_size:]

ytrain\_pse, ytest\_pse=ytrain[0:-test\_size], ytrain[-test\_size:]

# Unscaled Data Preparation for Evaluation

# Trend

y\_train\_pse\_raw=y\_train\_raw[0:-test\_size]

y\_test\_pse\_raw=y\_train\_raw[-test\_size:]

# Real Train Test Split Reshape

Xtrain, Xtest=Xtrain.reshape(Xtrain.shape[0], time\_window, 1), Xtest.reshape(Xtest.shape[0], time\_window, 1)

# Validation Train Test Split Reshape

Xtrain\_pse, Xtest\_pse=Xtrain\_pse.reshape(Xtrain\_pse.shape[0], time\_window, 1), Xtest\_pse.reshape(Xtest\_pse.shape[0], time\_window, 1)

for n in range(neurons\_min,neurons\_max,step):

# fig=plt.figure()

rmse\_t=[]

for j in range(repeats):

model=Sequential()

model.add(LSTM(n,input\_shape = (time\_window,1), batch\_size=1,stateful=True))

model.add(Dropout(0.2))

model.add(Dense(1,activation='linear'))

# model.add(Dense(1))

# compile model for use

start=time.time()

model.compile(loss="mse", optimizer="rmsprop")

print("Compilation Time : ", time.time() - start)

train\_loss\_t=[]

cv\_loss\_t=[]

for i in range(epochs):

history=model.fit(Xtrain\_pse,ytrain\_pse, epochs=1, batch\_size=1, verbose=2,validation\_split=0.1,shuffle=False)

train\_loss\_t.append(history.history['loss'])

cv\_loss\_t.append(history.history['val\_loss'])

model.reset\_states()

# Loss Calculation

# Test RMSE - For my own happiness

test\_predict\_pse = model.predict(Xtest\_pse,batch\_size=1)

test\_predict\_pse=scaler.inverse\_transform(test\_predict\_pse)

# Trend RMSE

rmse\_t.append(math.sqrt(mean\_squared\_error(y\_test\_pse\_raw,test\_predict\_pse)))

# cross validation loss

cv\_loss.append(np.array(cv\_loss\_t))

cv\_loss\_res.append(np.array(cv\_loss\_t)\*diff\_std)

# training loss

train\_loss.append(np.array(train\_loss\_t))

train\_loss\_res.append(np.array(train\_loss\_t)\*diff\_std)

# TREND

rmse\_t.append(np.mean(rmse\_t)) # calculate mean and std of Trend RMSE

rmse\_t.append(np.std(rmse\_t))

test\_loss.append(rmse\_t) # append mean and std of rmse

# plt.plot(train\_loss,color='blue')

# plt.plot(cv\_loss,color='orange')

# plt.title('model train vs validation loss')

# plt.ylabel('loss')

# plt.xlabel('epoch')

# # # plt.legend(['train', 'validation'], loc='upper right')

# plt.show()

# fig.savefig('epochs\_tuning.png',dpi=1000)

# In[22]:

test\_loss

# In[23]:

# test\_loss\_sum=[]

# test\_loss\_multi\_sum=[]

test\_loss\_sum.append(np.array(test\_loss))

# test\_loss\_multi\_sum.append(np.array(test\_loss\_multi))

# In[24]:

print(test\_loss\_sum)

# In[25]:

# cv\_loss

# In[26]:

# cv\_loss\_res

# In[27]:

# train\_loss\_res

# In[28]:

# Cross Validation Visualization

fig=plt.figure()

line1=plt.plot(np.array(train\_loss)[0,:],color='blue',label="Train Loss")

plt.plot(np.array(train\_loss)[1,:],color='blue')

plt.plot(np.array(train\_loss)[2,:],color='blue')

plt.plot(np.array(train\_loss)[3,:],color='blue')

plt.plot(np.array(train\_loss)[4,:],color='blue')

# plt.plot(np.array(train\_loss)[5,:],color='blue')

# plt.plot(np.array(train\_loss)[6,:],color='blue')

# plt.plot(np.array(train\_loss)[7,:],color='blue')

# plt.plot(np.array(train\_loss)[8,:],color='blue')

# plt.plot(np.array(train\_loss)[9,:],color='blue')

line2=plt.plot(np.array(cv\_loss)[0,:],color='orange',label= "Cross Validation Loss")

plt.plot(np.array(cv\_loss)[1,:],color='orange')

plt.plot(np.array(cv\_loss)[2,:],color='orange')

plt.plot(np.array(cv\_loss)[3,:],color='orange')

plt.plot(np.array(cv\_loss)[4,:],color='orange')

# plt.plot(np.array(cv\_loss)[5,:],color='orange')

# plt.plot(np.array(cv\_loss)[6,:],color='orange')

# plt.plot(np.array(cv\_loss)[7,:],color='orange')

# plt.plot(np.array(cv\_loss)[8,:],color='orange')

# plt.plot(np.array(cv\_loss)[9,:],color='orange')

plt.legend()

plt.xlabel('Epochs')

plt.ylabel('RMSE(Scaled)')

plt.title('Train Loss & Cross Validation Loss')

plt.show()

fig.savefig('loss\_output.png',dpi=1000)

# In[268]:

fig=plt.figure()

plt.plot(np.arange(40,100,5),test\_loss.iloc[0,:],label='time\_window=10,linear activation',color='blue')

plt.plot(np.arange(40,100,5),test\_loss.iloc[1,:],label='time\_window=11,linear activation',color='red')

plt.plot(np.arange(40,100,5),test\_loss.iloc[2,:],label='time\_window=12,linear activation',color='yellow')

plt.plot(np.arange(40,100,5),test\_loss.iloc[3,:],label='time\_window=13,linear activation',color='orange')

plt.plot(np.arange(40,100,5),test\_loss.iloc[4,:],label='time\_window=14,linear activation',color='green')

plt.legend()

plt.xlabel('neurons')

plt.ylabel('loss')

plt.title('One Step Ahead Loss')

plt.show()

fig.savefig('loss\_output.png',dpi=1000)

# In[269]:

test\_loss.to\_excel('Loss\_T10-14\_N40-100.xlsx')

# In[200]:

# Predict

train\_predict = model.predict(Xtrain,batch\_size=1)

test\_predict = model.predict(Xtest,batch\_size=1)

# Inverse Prediction

train\_predict=scaler.inverse\_transform(train\_predict)

test\_predict=scaler.inverse\_transform(test\_predict)

# ## Training Score

# In[201]:

# epochs\_summary=[]

# neurons\_summary=[]

# rmse\_summary=[]

# mae\_summary=[]

# In[202]:

rmse=math.sqrt(mean\_squared\_error(y\_test\_raw,test\_predict))

mae=mean\_absolute\_error(y\_test\_raw,test\_predict)

# In[203]:

epochs\_summary.append(epochs)

neurons\_summary.append(neurons)

rmse\_summary.append(rmse)

mae\_summary.append(mae)

print(epochs\_summary)

print(neurons\_summary)

print(rmse\_summary)

print(mae\_summary)

# ## Diagnostic Plot

# In[101]:

fig=plt.figure()

plt.plot(train\_loss\_summary)

plt.plot(cv\_loss\_summary)

plt.title('model train vs validation loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'validation'], loc='upper right')

plt.show()

fig.savefig('epochs\_tuning.png',dpi=1000)

# In[102]:

train\_loss\_summary

# ## Final Model Fitting

# In[ ]:

# Model Fitting

epochs=500

repeats=5

neurons\_min=1

neurons\_max=2

window\_min=2

window\_max=3

step=1

# train\_loss=np.zeros(shape=(window\_max-window\_min,int((neurons\_max-neurons\_min)/5)))

# cv\_loss=np.zeros(shape=(window\_max-window\_min,int((neurons\_max-neurons\_min)/5)))

# test\_loss=[np.zeros(shape=(window\_max-window\_min,int((neurons\_max-neurons\_min)/5)))]

train\_loss=[]

train\_loss\_res=[]

cv\_loss=[]

cv\_loss\_res=[]

test\_loss=[]

for w in range(window\_min,window\_max):

time\_window=w

Xall, yall=[],[]

for i in range(time\_window, len(diff\_scaled)):

Xall.append(diff\_scaled[i-time\_window:i,0])

yall.append(diff\_scaled[i,0])

Xall=np.array(Xall)

yall=np.array(yall)

# Real Train Test Split

test\_size=12

Xtrain, Xtest=Xall[0:-test\_size], Xall[-test\_size:]

ytrain, ytest=yall[0:-test\_size], yall[-test\_size:]

# Unscaled Data Preparation for Evaluation

y\_train\_raw=season\_diff[0:-test\_size]

y\_test\_raw=season\_diff[-test\_size:]

# # Validation Train Test Split

# Xtrain\_pse, Xtest\_pse=Xtrain[0:-test\_size], Xtrain[-test\_size:]

# ytrain\_pse, ytest\_pse=ytrain[0:-test\_size], ytrain[-test\_size:]

# # Unscaled Data Preparation for Evaluation

# # Trend

# y\_train\_pse\_raw=y\_train\_raw[0:-test\_size]

# y\_test\_pse\_raw=y\_train\_raw[-test\_size:]

# Real Train Test Split Reshape

Xtrain, Xtest=Xtrain.reshape(Xtrain.shape[0], time\_window, 1), Xtest.reshape(Xtest.shape[0], time\_window, 1)

# Validation Train Test Split Reshape

# Xtrain\_pse, Xtest\_pse=Xtrain\_pse.reshape(Xtrain\_pse.shape[0], time\_window, 1), Xtest\_pse.reshape(Xtest\_pse.shape[0], time\_window, 1)

for n in range(neurons\_min,neurons\_max,step):

# fig=plt.figure()

rmse\_t=[]

for j in range(repeats):

model=Sequential()

model.add(LSTM(n,input\_shape = (time\_window,1), batch\_size=1,stateful=True))

model.add(Dropout(0.2))

model.add(Dense(1,activation='linear'))

# model.add(Dense(1))

# compile model for use

start=time.time()

model.compile(loss="mse", optimizer="rmsprop")

print("Compilation Time : ", time.time() - start)

train\_loss\_t=[]

cv\_loss\_t=[]

for i in range(epochs):

history=model.fit(Xtrain,ytrain, epochs=1, batch\_size=1, verbose=2,validation\_split=0.1,shuffle=False)

train\_loss\_t.append(history.history['loss'])

cv\_loss\_t.append(history.history['val\_loss'])

model.reset\_states()

# Loss Calculation

# Test RMSE

# One Step-ahead Forecast

test\_predict = model.predict(Xtest,batch\_size=1)

test\_predict\_pse=scaler.inverse\_transform(test\_predict)

# Trend RMSE

rmse\_t.append(math.sqrt(mean\_squared\_error(y\_test\_raw,test\_predict)))

# cross validation loss

cv\_loss.append(np.array(cv\_loss\_t))

cv\_loss\_res.append(np.array(cv\_loss\_t)\*diff\_std)

# training loss

train\_loss.append(np.array(train\_loss\_t))

train\_loss\_res.append(np.array(train\_loss\_t)\*diff\_std)

# TREND

rmse\_t.append(np.mean(rmse\_t)) # calculate mean and std of Trend RMSE

rmse\_t.append(np.std(rmse\_t))

test\_loss.append(rmse\_t) # append mean and std of rmse

# plt.plot(train\_loss,color='blue')

# plt.plot(cv\_loss,color='orange')

# plt.title('model train vs validation loss')

# plt.ylabel('loss')

# plt.xlabel('epoch')

# # # plt.legend(['train', 'validation'], loc='upper right')

# plt.show()

# fig.savefig('epochs\_tuning.png',dpi=1000)

**Experiment 04:**

# coding: utf-8

# # <center>Assignment 2 LSTM - Fixed Window Size Loop With Unadjusted-Seasonality Data

# In[321]:

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

# In[322]:

# Basic Time Series Setting

import statsmodels.api as smapi

import statsmodels.tsa.api as smt

# In[323]:

# RNN Package Setting

import time

import math

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

from keras.layers.core import Dense, Activation, Dropout

from keras.layers.recurrent import LSTM

from keras.models import Sequential

from keras import optimizers

np.random.seed(1)

# ## Data Preparation

# In[324]:

data=pd.read\_csv('internet\_retail\_sales.csv',parse\_dates=[0], index\_col=0)

sales=data.values

# ### Standardisation

# In[325]:

# Standardisation

scaler = StandardScaler()

sales\_scaled=scaler.fit\_transform(sales)

# Std & Mean Calculation

data\_std=np.std(sales)

data\_mean=np.mean(sales)

print(data\_std)

print(data\_mean)

# ### Fixed Data Window

# In[326]:

# Creating Training and Test feature

time\_window=12

Xall, yall=[],[]

for i in range(time\_window, len(sales\_scaled)):

Xall.append(sales\_scaled[i-time\_window:i,0])

yall.append(sales\_scaled[i,0])

Xall=np.array(Xall)

yall=np.array(yall)

# ### Train Test Split

# #### Real Test Split for Hold out

# In[313]:

test\_size=12

Xtrain, Xtest=Xall[0:-test\_size], Xall[-test\_size:]

ytrain, ytest=yall[0:-test\_size], yall[-test\_size:]

# Unscaled Data Preparation for Evaluation

y\_train\_raw=sales[0:-test\_size]

y\_test\_raw=sales[-test\_size:]

# #### Pseudo Test Split for Validation

# In[314]:

Xtrain\_pse, Xtest\_pse=Xtrain[0:-test\_size], Xtrain[-test\_size:]

ytrain\_pse, ytest\_pse=ytrain[0:-test\_size], ytrain[-test\_size:]

# Unscaled Data Preparation for Evaluation

y\_train\_pse\_raw=y\_train\_raw[0:-test\_size]

y\_test\_pse\_raw=y\_train\_raw[-test\_size:]

# ### Reshape Data Window into 3D

# In[315]:

# Real Train Test Split

Xtrain, Xtest=Xtrain.reshape(Xtrain.shape[0], time\_window, 1), Xtest.reshape(Xtest.shape[0], time\_window, 1)

# Pseudo Train Test Split

Xtrain\_pse, Xtest\_pse=Xtrain\_pse.reshape(Xtrain\_pse.shape[0], time\_window, 1), Xtest\_pse.reshape(Xtest\_pse.shape[0], time\_window, 1)

print(Xtrain.shape)

print(Xtest.shape)

print(Xtrain\_pse.shape)

print(Xtest\_pse.shape)

# ## RNN-Fixed Window Size Model Fitting

# In[328]:

# Model Fitting

epochs=1

repeats=2

neurons\_min=1

neurons\_max=2

window\_min=12

window\_max=13

step=1

# train\_loss=np.zeros(shape=(window\_max-window\_min,int((neurons\_max-neurons\_min)/5)))

# cv\_loss=np.zeros(shape=(window\_max-window\_min,int((neurons\_max-neurons\_min)/5)))

# test\_loss=[np.zeros(shape=(window\_max-window\_min,int((neurons\_max-neurons\_min)/5)))]

train\_loss=[]

train\_loss\_res=[]

cv\_loss=[]

cv\_loss\_res=[]

test\_loss=[]

for w in range(window\_min,window\_max):

time\_window=w

Xall, yall=[],[]

for i in range(time\_window, len(sales\_scaled)):

Xall.append(sales\_scaled[i-time\_window:i,0])

yall.append(sales\_scaled[i,0])

Xall=np.array(Xall)

yall=np.array(yall)

# Real Train Test Split

test\_size=12

Xtrain, Xtest=Xall[0:-test\_size], Xall[-test\_size:]

ytrain, ytest=yall[0:-test\_size], yall[-test\_size:]

# Unscaled Data Preparation for Evaluation

y\_train\_raw=sales[0:-test\_size]

y\_test\_raw=sales[-test\_size:]

# Validation Train Test Split

Xtrain\_pse, Xtest\_pse=Xtrain[0:-test\_size], Xtrain[-test\_size:]

ytrain\_pse, ytest\_pse=ytrain[0:-test\_size], ytrain[-test\_size:]

# Unscaled Data Preparation for Evaluation

y\_train\_pse\_raw=y\_train\_raw[0:-test\_size]

y\_test\_pse\_raw=y\_train\_raw[-test\_size:]

# Real Train Test Split Reshape

Xtrain, Xtest=Xtrain.reshape(Xtrain.shape[0], time\_window, 1), Xtest.reshape(Xtest.shape[0], time\_window, 1)

# Validation Train Test Split Reshape

Xtrain\_pse, Xtest\_pse=Xtrain\_pse.reshape(Xtrain\_pse.shape[0], time\_window, 1), Xtest\_pse.reshape(Xtest\_pse.shape[0], time\_window, 1)

for n in range(neurons\_min,neurons\_max,step):

# fig=plt.figure()

rmse\_t=[]

for j in range(repeats):

model=Sequential()

model.add(LSTM(n,input\_shape = (time\_window,1), batch\_size=1,stateful=True))

model.add(Dropout(0.2))

model.add(Dense(1,activation='linear'))

# model.add(Dense(1))

# compile model for use

start=time.time()

model.compile(loss="mse", optimizer="rmsprop")

print("Compilation Time : ", time.time() - start)

train\_loss\_t=[]

cv\_loss\_t=[]

for i in range(epochs):

history=model.fit(Xtrain\_pse,ytrain\_pse, epochs=1, batch\_size=1, verbose=2,validation\_split=0.1,shuffle=False)

train\_loss\_t.append(history.history['loss'])

cv\_loss\_t.append(history.history['val\_loss'])

model.reset\_states()

# Loss Calculation

# Test RMSE - For my own happiness

test\_predict\_pse = model.predict(Xtest\_pse,batch\_size=1)

test\_predict\_pse=scaler.inverse\_transform(test\_predict\_pse)

rmse\_t.append(math.sqrt(mean\_squared\_error(y\_test\_pse\_raw,test\_predict\_pse)))

# cross validation loss

cv\_loss.append(np.array(cv\_loss\_t))

cv\_loss\_res.append(np.array(cv\_loss\_t)\*data\_std)

# training loss

train\_loss.append(np.array(train\_loss\_t))

train\_loss\_res.append(np.array(train\_loss\_t)\*data\_std)

# calculate mean and std of muti-rmse

rmse\_t.append(np.mean(rmse\_t))

rmse\_t.append(np.std(rmse\_t))

test\_loss.append(rmse\_t)

# plt.plot(train\_loss,color='blue')

# plt.plot(cv\_loss,color='orange')

# plt.title('model train vs validation loss')

# plt.ylabel('loss')

# plt.xlabel('epoch')

# # # plt.legend(['train', 'validation'], loc='upper right')

# plt.show()

# fig.savefig('epochs\_tuning.png',dpi=1000)

# In[329]:

test\_loss

# In[330]:

train\_loss

# In[331]:

cv\_loss

# In[332]:

cv\_loss\_res

# In[333]:

train\_loss\_res

# In[303]:

# Cross Validation Visualization

fig=plt.figure()

plt.plot(np.array(train\_loss)[0,:],color='blue')

plt.plot(np.array(train\_loss)[1,:],color='blue')

plt.plot(np.array(train\_loss)[2,:],color='blue')

plt.plot(np.array(train\_loss)[3,:],color='blue')

plt.plot(np.array(train\_loss)[4,:],color='blue')

plt.plot(np.array(cv\_loss)[0,:],color='orange')

plt.plot(np.array(cv\_loss)[1,:],color='orange')

plt.plot(np.array(cv\_loss)[2,:],color='orange')

plt.plot(np.array(cv\_loss)[3,:],color='orange')

plt.plot(np.array(cv\_loss)[4,:],color='orange')

# plt.legend()

plt.xlabel('epochs')

plt.ylabel('loss')

plt.title('train Loss & Cross Validation Loss')

plt.show()

fig.savefig('loss\_output.png',dpi=1000)

# In[268]:

fig=plt.figure()

plt.plot(np.arange(40,100,5),test\_loss.iloc[0,:],label='time\_window=10,linear activation',color='blue')

plt.plot(np.arange(40,100,5),test\_loss.iloc[1,:],label='time\_window=11,linear activation',color='red')

plt.plot(np.arange(40,100,5),test\_loss.iloc[2,:],label='time\_window=12,linear activation',color='yellow')

plt.plot(np.arange(40,100,5),test\_loss.iloc[3,:],label='time\_window=13,linear activation',color='orange')

plt.plot(np.arange(40,100,5),test\_loss.iloc[4,:],label='time\_window=14,linear activation',color='green')

plt.legend()

plt.xlabel('neurons')

plt.ylabel('loss')

plt.title('One Step Ahead Loss')

plt.show()

fig.savefig('loss\_output.png',dpi=1000)

# In[269]:

test\_loss.to\_excel('Loss\_T10-14\_N40-100.xlsx')

# In[200]:

# Predict

train\_predict = model.predict(Xtrain,batch\_size=1)

test\_predict = model.predict(Xtest,batch\_size=1)

# Inverse Prediction

train\_predict=scaler.inverse\_transform(train\_predict)

test\_predict=scaler.inverse\_transform(test\_predict)

# ## Training Score

# In[201]:

# epochs\_summary=[]

# neurons\_summary=[]

# rmse\_summary=[]

# mae\_summary=[]

# In[202]:

rmse=math.sqrt(mean\_squared\_error(y\_test\_raw,test\_predict))

mae=mean\_absolute\_error(y\_test\_raw,test\_predict)

# In[203]:

epochs\_summary.append(epochs)

neurons\_summary.append(neurons)

rmse\_summary.append(rmse)

mae\_summary.append(mae)

print(epochs\_summary)

print(neurons\_summary)

print(rmse\_summary)

print(mae\_summary)

# ## Diagnostic Plot

# In[101]:

fig=plt.figure()

plt.plot(train\_loss\_summary)

plt.plot(cv\_loss\_summary)

plt.title('model train vs validation loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'validation'], loc='upper right')

plt.show()

fig.savefig('epochs\_tuning.png',dpi=1000)

# In[102]:

train\_loss\_summary

**Experiment 05:**

# coding: utf-8

# # <center>Assignment 2 LSTM - No Window Size

# In[1]:

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

# In[12]:

# Basic Time Series Setting

import statsmodels.api as smapi

import statsmodels.tsa.api as smt

# In[104]:

# RNN Package Setting

import time

import math

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error

from keras.layers.core import Dense, Activation, Dropout

from keras.layers.recurrent import LSTM

from keras.models import Sequential

# ## 1. Exploratory Data Analysis

# In[35]:

data=pd.read\_csv('internet\_retail\_sales.csv',parse\_dates=[0], index\_col=0)

sales=data.values

# In[3]:

data.head(10)

# In[5]:

sns.distplot(sales,hist=True,label='sales')

sns.despine()

plt.show()

# In[286]:

plt.plot(np.log(data),label='Internet Retail Sales')

plt.legend()

sns.despine()

plt.show()

# In[287]:

plt.plot(data,label='Internet Retail Sales')

plt.legend()

sns.despine()

plt.show()

# In[16]:

decomp\_obj = smapi.tsa.seasonal\_decompose(data['Sales'])

decomp\_obj.plot()

# ## 2. Data Preparation

# In[7]:

np.random.seed(1)

# ### Prepare X and Y

# In[99]:

# Prepare X and Y

def timeseries\_to\_supervised(data, lag=1):

df = pd.DataFrame(data)

columns = [df.shift(i) for i in range(1, lag+1)]

columns.append(df)

df = pd.concat(columns, axis=1)

df = df.drop(0)

return df

data\_supervised=timeseries\_to\_supervised(sales, 1)

print(data\_supervised.head(10))

data\_supervised\_values=data\_supervised.values

# data\_supervised\_values

# ### Train Test Split

# In[145]:

# train test split

test\_size=12

train, test=data\_supervised\_values[0:-test\_size], data\_supervised\_values[-test\_size:]

# ### Standardisation

# In[146]:

# fit scaler

scaler = StandardScaler()

scaler = scaler.fit(train)

# transform train

train = train.reshape(train.shape[0], train.shape[1])

train\_scaled = scaler.transform(train)

# transform test

test = test.reshape(test.shape[0], test.shape[1])

test\_scaled = scaler.transform(test)

print(train\_scaled[1:10])

print('\n')

print(test\_scaled[1:10])

# In[217]:

y\_train\_raw=train[:,-1]

y\_test\_raw=test[:,-1]

y\_train\_raw=y\_train\_raw.reshape(y\_train\_raw.shape[0],1)

y\_test\_raw=y\_test\_raw.reshape(y\_test\_raw.shape[0],1)

# In[180]:

scaler\_y=StandardScaler()

scaler\_y.fit(y\_train\_raw)

# ### Reshpe Data Window into 3D

# In[148]:

X\_train, y\_train=train\_scaled[:,0:-1], train\_scaled[:,-1]

X\_test,y\_test=test\_scaled[:,0:-1], test\_scaled[:,-1]

X\_train=X\_train.reshape(X\_train.shape[0], 1, X\_train.shape[1])

X\_test=X\_test.reshape(X\_test.shape[0], 1, X\_test.shape[1])

print(X\_train.shape)

print(X\_test.shape)

# ## 3. RNN-LSTM Model Fitting

# In[277]:

# build LSTM Model

epochs=10

neurons=30

model=Sequential()

model.add(LSTM(100,batch\_input\_shape=(1, X\_train.shape[1], X\_train.shape[2]),stateful=True))

model.add(Dropout(0.2))

model.add(Dense(1))

# compile model for use

start=time.time()

model.compile(loss="mse", optimizer="rmsprop")

print("Compilation Time : ", time.time() - start)

# In[278]:

max\_len = 1

from keras.callbacks import Callback

class ResetStatesCallback(Callback):

def \_\_init\_\_(self):

self.counter = 0

def on\_batch\_begin(self, batch, logs={}):

if self.counter % max\_len == 0:

self.model.reset\_states()

self.counter += 1

# In[279]:

# Training Model

history=model.fit(X\_train,y\_train,callbacks=[ResetStatesCallback()],batch\_size=1,nb\_epoch=10,shuffle=False)

# validation\_split=0.1,

# model.reset\_states()

# In[280]:

# One-step Predict

train\_predict=model.predict(X\_train,batch\_size=1)

test\_predict=model.predict(X\_test,batch\_size=1)

# Inverse Train Test Prediction

train\_predict=scaler\_y.inverse\_transform(train\_predict)

test\_predict=scaler\_y.inverse\_transform(test\_predict)

# ### Train Score

# In[238]:

# epochs\_summary=[]

# rmse=[]

# neurons\_summary=[]

# In[281]:

train\_score=math.sqrt(mean\_squared\_error(y\_test\_raw,test\_predict))

train\_score

rmse.append(train\_score)

epochs\_summary.append(epochs)

neurons\_summary.append(neurons)

# In[282]:

print(rmse)

print(epochs\_summary)

print(neurons\_summary)

# In[284]:

fig=plt.figure()

plt.plot(history.history['loss'])

# plt.plot(history.history['val\_loss'])

plt.title('model train vs validation loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'validation'], loc='upper right')

plt.show()

fig.savefig('fig1.png')

# In[240]:

# epochs\_summary.append(10)

# neurons\_summary.append(10)

# In[269]:

print(epochs\_summary)

print(neurons\_summary)

print(rmse)

# ### Visualization

# In[258]:

fig=plt.figure()

plt.plot(data.values[1:],label='Sales')

plt.plot(np.arange(1,len(train\_predict)+1),train\_predict,label='Train Prediction')

plt.plot(np.arange(len(train\_predict)+1,len(train\_predict)+test\_size+1),test\_predict,label='Test Prediction')

plt.legend()

plt.show()

fig.savefig('fig1.png')

# In[216]:

len(test\_predict)

# In[206]:

len(train\_predict)

# In[207]:

len(data)

# In[203]:

np.arange(1,len(train\_predict)+1)

# In[210]:

np.arange(len(train\_predict)+1,len(train\_predict)+test\_size+1)

**Experiment 06:**

# coding: utf-8

# # <center>Assignment 2 LSTM - Fixed Window Size

# In[1]:

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

# In[2]:

# Basic Time Series Setting

import statsmodels.api as smapi

import statsmodels.tsa.api as smt

# In[3]:

# RNN Package Setting

import time

import math

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error

from keras.layers.core import Dense, Activation, Dropout

from keras.layers.recurrent import LSTM

from keras.models import Sequential

from keras import optimizers

# In[4]:

data=pd.read\_csv('internet\_retail\_sales.csv',parse\_dates=[0], index\_col=0)

sales=data.values

# In[5]:

np.random.seed(1)

# ## Data Preparation

# ### Seasonality Adjust

# ### Standardisation

# In[6]:

# Standardisation

scaler = StandardScaler()

sales\_scaled=scaler.fit\_transform(sales)

# ### Fixed Data Window

# In[7]:

# Creating Training and Test feature

time\_window=12

Xall, yall=[],[]

for i in range(time\_window, len(sales\_scaled)):

Xall.append(sales\_scaled[i-time\_window:i,0])

yall.append(sales\_scaled[i,0])

Xall=np.array(Xall)

yall=np.array(yall)

# ### Train Test Split

# In[8]:

# Train Test Split

test\_size=12

Xtrain, Xtest=Xall[0:-test\_size], Xall[-test\_size:]

ytrain, ytest=yall[0:-test\_size], yall[-test\_size:]

# In[9]:

y\_train\_raw=sales[0:-test\_size]

y\_test\_raw=sales[-test\_size:]

# ### Reshape Data Window into 3D

# In[10]:

Xtrain, Xtest=Xtrain.reshape(Xtrain.shape[0], time\_window, 1), Xtest.reshape(Xtest.shape[0], time\_window, 1)

print(Xtrain.shape)

print(Xtest.shape)

# ## RNN-Fixed Window Size Model Fitting

# In[11]:

# build LSTM Model

adam=optimizers.Adam(lr=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

neurons=1

model=Sequential()

model.add(LSTM(neurons,input\_shape = (time\_window,1), batch\_size=1,stateful=True))

# model.add(Dropout(0.2))

model.add(Dense(1))

# compile model for use

start=time.time()

model.compile(loss="mse", optimizer="adam")

print("Compilation Time : ", time.time() - start)

# In[12]:

max\_len = 12

from keras.callbacks import Callback

class ResetStatesCallback(Callback):

def \_\_init\_\_(self):

self.counter = 0

def on\_batch\_begin(self, batch, logs={}):

if self.counter % max\_len == 0:

self.model.reset\_states()

self.counter += 1

# In[13]:

# Training Model

epochs=1

history=model.fit(Xtrain,ytrain,batch\_size=1,nb\_epoch=epochs,validation\_split=0.1,shuffle=False)

# callbacks=[ResetStatesCallback()],

#

# model.reset\_states()

# In[14]:

# Predict

train\_predict = model.predict(Xtrain,batch\_size=1)

test\_predict = model.predict(Xtest,batch\_size=1)

# Inverse Prediction

train\_predict=scaler.inverse\_transform(train\_predict)

test\_predict=scaler.inverse\_transform(test\_predict)

# ## Training Score

# In[15]:

epochs\_summary=[]

neurons\_summary=[]

rmse\_summary=[]

# In[16]:

rmse=math.sqrt(mean\_squared\_error(y\_test\_raw,test\_predict))

# In[17]:

epochs\_summary.append(epochs)

neurons\_summary.append(neurons)

rmse\_summary.append(rmse)

print(epochs\_summary)

print(neurons\_summary)

print(rmse\_summary)

# ## Diagnostic Plot

# In[18]:

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model train vs validation loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'validation'], loc='upper right')

plt.show()

# In[19]:

history.history

**Final Model:**

# coding: utf-8

# # <center>Assignment 2 LSTM - Fixed Window Size Loop with First Order Difference & Seasonal Difference (Final Model)

#

# In[1]:

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

# In[2]:

# Basic Time Series Setting

import statsmodels.api as smapi

import statsmodels.tsa.api as smt

# In[3]:

# RNN Package Setting

import time

import math

import statsmodels.api as smapi

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

from keras.layers.core import Dense, Activation, Dropout

from keras.layers.recurrent import LSTM

from keras.models import Sequential

from keras import optimizers

np.random.seed(1)

# In[79]:

def mean\_absolute\_percentage\_error(y\_true, y\_pred):

y\_true, y\_pred = np.array(y\_true), np.array(y\_pred)

return np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100

# In[80]:

def mean\_error(y\_true, y\_pred):

y\_true, y\_pred = np.array(y\_true), np.array(y\_pred)

return np.mean(y\_true - y\_pred)

# ## Data Preparation & Exploratory Data Analysis

# In[4]:

data=pd.read\_csv('internet\_retail\_sales.csv',parse\_dates=[0], index\_col=0)

sales=data.values

# ### First Order Difference & Seasonal Difference

# #### First Order Difference

# In[5]:

sales\_diff\_trial=pd.Series.diff(data)

sales\_diff\_trial=sales\_diff\_trial.dropna()

# #### First Order Difference ACF Plot

# In[6]:

fig=smt.graphics.plot\_acf(sales\_diff\_trial, lags=24)

plt.show()

fig.savefig('ACF.png',dpi=1000)

# #### First Order Difference PACF Plot

# In[7]:

fig=smt.graphics.plot\_pacf(sales\_diff\_trial, lags=24)

plt.show()

fig.savefig('PACF.png',dpi=1000)

# In[188]:

# First Order Difference

# plt.plot(date,sales\_trend[:-1])

# sales\_diff=pd.Series.diff(data)

# sales\_diff=sales\_diff.dropna().values

# Seasonally First Order Difference

season\_diff=pd.Series.diff(sales\_diff,periods=12)

season\_diff=season\_diff.dropna().values

fig=plt.figure()

date=pd.date\_range(start='11/10/2005',end='03/01/2017',freq='MS')

plt.plot(date[-len(season\_diff):],season\_diff)

plt.xlabel('year')

plt.ylabel('pounds (Million)')

plt.title('1st Order Difference and Seasonal Difference of Sales Data')

plt.show()

fig.savefig('2ndord.png',dpi=1000)

# #### First Order & Seasonal Difference ACF Plot

# In[10]:

fig=smt.graphics.plot\_acf(season\_diff, lags=24)

plt.show()

fig.savefig('SACF.png',dpi=1000)

# #### First Order & Seasonal Difference PACF Plot

# In[11]:

fig=smt.graphics.plot\_pacf(season\_diff, lags=24)

plt.show()

fig.savefig('SPACF.png',dpi=1000)

# ### Standardisation

# In[13]:

# Standardisation

scaler = StandardScaler()

diff\_scaled=scaler.fit\_transform(season\_diff)

# Std & Mean Calculation

diff\_std=np.std(season\_diff)

diff\_mean=np.mean(season\_diff)

print(diff\_std)

print(diff\_mean)

# ## LSTM - Model Training

# \*\*Data\*\*: First Order Difference + Seasonal Difference

# <br>

# \*\*Model:\*\*

# - LSTM Fixed Window size=2

# - Neurons=1

# - Combine 30 Repeat and use the average forecast result as our final prediction

# <br>

# We tuning our hyperparameters here, including \*\*neurons, window size and epochs\*\*

#

# In[81]:

# Memory Variables preparation

test\_loss\_sum=[]

training\_final\_loss\_sum=[]

hyper\_para=[]

hyper\_para\_sum=[]

# In[82]:

# Model Fitting

epochs=100

repeats=10

neurons\_min=6

neurons\_max=7

window\_min=3

window\_max=4

step=1

train\_loss=[]

train\_loss\_res=[]

cv\_loss=[]

cv\_loss\_res=[]

test\_loss=[]

training\_final\_loss=[]

test\_predict\_pse\_sum=[]

train\_predict\_pse\_sum=[]

len\_window=[]

for w in range(window\_min,window\_max):

time\_window=w

Xall, yall=[],[]

for i in range(time\_window, len(diff\_scaled)):

Xall.append(diff\_scaled[i-time\_window:i,0])

yall.append(diff\_scaled[i,0])

Xall=np.array(Xall)

yall=np.array(yall)

len\_window=len(Xall)

# Real Train Test Split

test\_size=12

Xtrain, Xtest=Xall[0:-test\_size], Xall[-test\_size:]

ytrain, ytest=yall[0:-test\_size], yall[-test\_size:]

# Unscaled Data Preparation for Evaluation

y\_total=season\_diff[-len(Xall):]

y\_train\_raw=y\_total[0:-test\_size]

y\_test\_raw=y\_total[-test\_size:]

# Validation Train Test Split

Xtrain\_pse, Xtest\_pse=Xtrain[0:-test\_size], Xtrain[-test\_size:]

ytrain\_pse, ytest\_pse=ytrain[0:-test\_size], ytrain[-test\_size:]

# Unscaled Data Preparation for Evaluation

# Trend

y\_train\_pse\_raw=y\_train\_raw[0:-test\_size]

y\_test\_pse\_raw=y\_train\_raw[-test\_size:]

# Real Train Test Split Reshape

Xtrain, Xtest=Xtrain.reshape(Xtrain.shape[0], time\_window, 1), Xtest.reshape(Xtest.shape[0], time\_window, 1)

# Validation Train Test Split Reshape

Xtrain\_pse, Xtest\_pse=Xtrain\_pse.reshape(Xtrain\_pse.shape[0], time\_window, 1), Xtest\_pse.reshape(Xtest\_pse.shape[0], time\_window, 1)

for n in range(neurons\_min,neurons\_max,step):

rmse\_t\_test=[]

rmse\_t\_train=[]

test\_predict\_pse\_sum\_t=[]

train\_predict\_pse\_sum\_t=[]

for j in range(repeats):

model=Sequential()

model.add(LSTM(n,input\_shape = (time\_window,1), batch\_size=1,stateful=True))

model.add(Dropout(0.2))

model.add(Dense(1,activation='linear'))

# model.add(Dense(1))

# compile model for use

start=time.time()

model.compile(loss="mse", optimizer="rmsprop")

print("Compilation Time : ", time.time() - start)

train\_loss\_t=[]

cv\_loss\_t=[]

for i in range(epochs):

history=model.fit(Xtrain\_pse,ytrain\_pse, epochs=1, batch\_size=1, verbose=2,validation\_split=0.1,shuffle=False)

train\_loss\_t.append(history.history['loss'])

cv\_loss\_t.append(history.history['val\_loss'])

model.reset\_states()

# Loss Calculation

# Validation Prediction

test\_predict\_pse = model.predict(Xtest\_pse,batch\_size=1)

test\_predict\_pse=scaler.inverse\_transform(test\_predict\_pse)

test\_predict\_pse\_sum\_t.append(np.array(test\_predict\_pse))

# Training Prediction

train\_predict\_pse=model.predict(Xtrain\_pse,batch\_size=1)

train\_predict\_pse=scaler.inverse\_transform(train\_predict\_pse)

train\_predict\_pse\_sum\_t.append(np.array(train\_predict\_pse))

# Test RMSE

rmse\_t\_test.append(math.sqrt(mean\_squared\_error(y\_test\_pse\_raw,test\_predict\_pse)))

# Training RMSE

rmse\_t\_train.append(math.sqrt(mean\_squared\_error(y\_train\_pse\_raw,train\_predict\_pse)))

# cross validation loss

cv\_loss.append(np.array(cv\_loss\_t))

cv\_loss\_res.append(np.array(cv\_loss\_t)\*diff\_std)

# training loss

train\_loss.append(np.array(train\_loss\_t))

train\_loss\_res.append(np.array(train\_loss\_t)\*diff\_std)

# summary statistics

# Model Combination Predicting

test\_predict\_pse\_sum.append(np.mean(np.array(test\_predict\_pse\_sum\_t),axis=0)) # validation

train\_predict\_pse\_sum.append(np.mean(np.array(train\_predict\_pse\_sum\_t),axis=0)) # training

# Validation RMSE

rmse\_t\_test.append(np.std(rmse\_t\_test))

rmse\_t\_test.append(math.sqrt(mean\_squared\_error(y\_test\_pse\_raw,np.mean(np.array(test\_predict\_pse\_sum\_t),axis=0)))) # calculate mean and std of Trend RMSE

test\_loss.append(rmse\_t\_test) # append Combination RMSE

# Training RMSE

rmse\_t\_train.append(np.std(rmse\_t\_train))

rmse\_t\_train.append(math.sqrt(mean\_squared\_error(y\_train\_pse\_raw,np.mean(np.array(train\_predict\_pse\_sum\_t),axis=0)))) # calculate mean and std of Trend RMSE

training\_final\_loss.append(rmse\_t\_train) # append Combination RMSE

# ### Summary Statistics

# #### Validation Loss

# In[83]:

pd.DataFrame(test\_loss)

# In[84]:

# Meomory

test\_loss\_sum.append(test\_loss)

hyper\_para.append(neurons\_min)

hyper\_para.append(epochs)

hyper\_para.append(window\_min)

hyper\_para\_sum.append(np.array(hyper\_para))

# In[85]:

test\_loss\_sum

# In[86]:

pd.DataFrame(test\_loss\_sum).to\_excel('test\_loss\_sum\_T2.xlsx')

# In[87]:

pd.DataFrame(hyper\_para\_sum)

# #### Training Loss

# In[88]:

pd.DataFrame(training\_final\_loss)

# In[89]:

training\_final\_loss\_sum.append(training\_final\_loss)

training\_final\_loss\_sum

# In[90]:

pd.DataFrame(training\_final\_loss\_sum).to\_excel('train\_loss\_sum\_T2.xlsx')

# #### Prediction

# In[91]:

test\_predict\_pse\_sum=np.array(test\_predict\_pse\_sum)

# test\_predict\_pse\_sum.shape

test\_predict\_pse\_sum=test\_predict\_pse\_sum.reshape(12,1)

train\_predict\_pse\_sum=np.array(train\_predict\_pse\_sum)

# test\_predict\_pse\_sum.shape

train\_predict\_pse\_sum=train\_predict\_pse\_sum.reshape(len\_window-2\*test\_size,1)

# In[92]:

test\_predict\_pse\_sum

# In[93]:

# Prediction Visualization

fig=plt.figure()

date=pd.date\_range(start='11/10/2006',end='03/01/2017',freq='MS')

date=date[-len\_window:-test\_size]

# date\_train=date[0:]

# date\_train\_actual=date[-len\_window:-test\_size]

plt.plot(date,season\_diff[-len\_window:-test\_size],label='Observation')

plt.plot(date[-test\_size:],test\_predict\_pse\_sum,label='Validation Prediction')

plt.plot(date[0:-test\_size],train\_predict\_pse\_sum,label='Training Prediction')

plt.legend()

plt.title('In Sample Prediction and Validation Prediction of 1st Order & Seasonal Difference Data')

plt.show()

fig.savefig('S1D1ForecaastV.png',dpi=1000)

# #### Forecast Recovery

# In[94]:

# Reshift Back to First Order Difference

season\_shift=sales\_diff.shift(12)[-len\_window:].values

season\_is\_forecast\_rec=train\_predict\_pse\_sum+season\_shift[0:-2\*test\_size]

season\_v\_forecast\_rec=test\_predict\_pse\_sum+season\_shift[-2\*test\_size:-test\_size]

# Reshift to Orignal Data

fst\_shift=data.shift(1)[-len\_window:].values

is\_forecast\_rec=season\_is\_forecast\_rec+fst\_shift[0:-2\*test\_size]

v\_forecast\_rec=season\_v\_forecast\_rec+fst\_shift[-2\*test\_size:-test\_size]

# In[95]:

# Original Data Prediction Visualization

fig=plt.figure()

date=pd.date\_range(start='11/10/2006',end='03/01/2017',freq='MS')

date=date[-len\_window:-test\_size]

# date\_train=date[0:]

# date\_train\_actual=date[-len\_window:-test\_size]

plt.plot(date,sales[-len\_window:-test\_size],label='Observation')

plt.plot(date[-test\_size:],v\_forecast\_rec,label='Validation Prediction')

plt.plot(date[0:-test\_size],is\_forecast\_rec,label='Training Prediction')

plt.legend()

plt.title('In Sample Prediction and Validation Prediction of Original Data')

plt.show()

fig.savefig('ForecaastV.png',dpi=1000)

# #### Training and Cross Validation Loss

# In[96]:

# Cross Validation Visualization

fig=plt.figure()

line1=plt.plot(np.array(train\_loss)[0,:],color='blue',label="Train Loss")

plt.plot(np.array(train\_loss)[1,:],color='blue')

plt.plot(np.array(train\_loss)[2,:],color='blue')

plt.plot(np.array(train\_loss)[3,:],color='blue')

plt.plot(np.array(train\_loss)[4,:],color='blue')

plt.plot(np.array(train\_loss)[5,:],color='blue')

plt.plot(np.array(train\_loss)[6,:],color='blue')

plt.plot(np.array(train\_loss)[7,:],color='blue')

plt.plot(np.array(train\_loss)[8,:],color='blue')

plt.plot(np.array(train\_loss)[9,:],color='blue')

# plt.plot(np.array(train\_loss)[10,:],color='blue')

# plt.plot(np.array(train\_loss)[11,:],color='blue')

# plt.plot(np.array(train\_loss)[12,:],color='blue')

# plt.plot(np.array(train\_loss)[13,:],color='blue')

# plt.plot(np.array(train\_loss)[14,:],color='blue')

# plt.plot(np.array(train\_loss)[15,:],color='blue')

# plt.plot(np.array(train\_loss)[16,:],color='blue')

# plt.plot(np.array(train\_loss)[17,:],color='blue')

# plt.plot(np.array(train\_loss)[18,:],color='blue')

# plt.plot(np.array(train\_loss)[19,:],color='blue')

# plt.plot(np.array(train\_loss)[20,:],color='blue')

# plt.plot(np.array(train\_loss)[21,:],color='blue')

# plt.plot(np.array(train\_loss)[22,:],color='blue')

# plt.plot(np.array(train\_loss)[23,:],color='blue')

# plt.plot(np.array(train\_loss)[24,:],color='blue')

# plt.plot(np.array(train\_loss)[25,:],color='blue')

# plt.plot(np.array(train\_loss)[26,:],color='blue')

# plt.plot(np.array(train\_loss)[27,:],color='blue')

# plt.plot(np.array(train\_loss)[28,:],color='blue')

# plt.plot(np.array(train\_loss)[29,:],color='blue')

line2=plt.plot(np.array(cv\_loss)[0,:],color='orange',label= "Cross Validation Loss")

plt.plot(np.array(cv\_loss)[1,:],color='orange')

plt.plot(np.array(cv\_loss)[2,:],color='orange')

plt.plot(np.array(cv\_loss)[3,:],color='orange')

plt.plot(np.array(cv\_loss)[4,:],color='orange')

plt.plot(np.array(cv\_loss)[5,:],color='orange')

plt.plot(np.array(cv\_loss)[6,:],color='orange')

plt.plot(np.array(cv\_loss)[7,:],color='orange')

plt.plot(np.array(cv\_loss)[8,:],color='orange')

plt.plot(np.array(cv\_loss)[9,:],color='orange')

# plt.plot(np.array(cv\_loss)[10,:],color='orange')

# plt.plot(np.array(cv\_loss)[11,:],color='orange')

# plt.plot(np.array(cv\_loss)[12,:],color='orange')

# plt.plot(np.array(cv\_loss)[13,:],color='orange')

# plt.plot(np.array(cv\_loss)[14,:],color='orange')

# plt.plot(np.array(cv\_loss)[15,:],color='orange')

# plt.plot(np.array(cv\_loss)[16,:],color='orange')

# plt.plot(np.array(cv\_loss)[17,:],color='orange')

# plt.plot(np.array(cv\_loss)[18,:],color='orange')

# plt.plot(np.array(cv\_loss)[19,:],color='orange')

# plt.plot(np.array(cv\_loss)[20,:],color='orange')

# plt.plot(np.array(cv\_loss)[21,:],color='orange')

# plt.plot(np.array(cv\_loss)[22,:],color='orange')

# plt.plot(np.array(cv\_loss)[23,:],color='orange')

# plt.plot(np.array(cv\_loss)[24,:],color='orange')

# plt.plot(np.array(cv\_loss)[25,:],color='orange')

# plt.plot(np.array(cv\_loss)[26,:],color='orange')

# plt.plot(np.array(cv\_loss)[27,:],color='orange')

# plt.plot(np.array(cv\_loss)[28,:],color='orange')

# plt.plot(np.array(cv\_loss)[29,:],color='orange')

plt.legend()

plt.xlabel('Epochs')

plt.ylabel('RMSE(Scaled)')

plt.title('Train Loss & Cross Validation Loss')

plt.show()

fig.savefig('loss\_output.png',dpi=1000)

# ## Final Optimal Model Fitting

# We train our optimal selected model here with the following hyperparameters:

# - Neurons=1

# - Window=3

# - Epochs=300

# and we will calculate the \*\*1-step ahead forecast and dynamic forecast RMSE MAD MAPE and ME\*\* here for our real test set

# In[156]:

# Model Fitting

###################### Variable Preparation ########################

epochs=300

repeats=30

neurons\_min=1

neurons\_max=2

window\_min=3

window\_max=4

step=1

train\_loss=[]

train\_loss\_res=[]

cv\_loss=[]

cv\_loss\_res=[]

# RMSE

test\_loss\_rmse\_1stp=[]

test\_loss\_rmse\_dyn=[]

training\_final\_loss\_rmse=[]

# MAD

test\_loss\_mad\_1stp=[]

test\_loss\_mad\_dyn=[]

training\_final\_loss\_mad=[]

#MAPE

test\_loss\_mape\_1stp=[]

test\_loss\_mape\_dyn=[]

training\_final\_loss\_mape=[]

#ME

test\_loss\_me\_1stp=[]

test\_loss\_me\_dyn=[]

training\_final\_loss\_me=[]

test\_predict\_sum\_1stp=[]

test\_predict\_sum\_dyn=[]

train\_predict\_sum=[]

len\_window=[]

for w in range(window\_min,window\_max): # Time Window Iteration

################################## Model Fitting ###############################################

time\_window=w

Xall, yall=[],[]

for i in range(time\_window, len(diff\_scaled)):

Xall.append(diff\_scaled[i-time\_window:i,0])

yall.append(diff\_scaled[i,0])

Xall=np.array(Xall)

yall=np.array(yall)

len\_window=len(Xall)

# Real Train Test Split

test\_size=12

Xtrain, Xtest=Xall[0:-test\_size], Xall[-test\_size:]

ytrain, ytest=yall[0:-test\_size], yall[-test\_size:]

# Unscaled Data Preparation for Evaluation

y\_total=season\_diff[-len(Xall):]

y\_train\_raw=y\_total[0:-test\_size]

y\_test\_raw=y\_total[-test\_size:]

# Real Train Test Split Reshape

Xtrain, Xtest=Xtrain.reshape(Xtrain.shape[0], time\_window, 1), Xtest.reshape(Xtest.shape[0], time\_window, 1)

for n in range(neurons\_min,neurons\_max,step): # Neurons Iteration

rmse\_t\_test\_1stp=[]

rmse\_t\_test\_dyn=[]

rmse\_t\_train=[]

mad\_t\_test\_1stp=[]

mad\_t\_test\_dyn=[]

mad\_t\_train=[]

mape\_t\_test\_1stp=[]

mape\_t\_test\_dyn=[]

mape\_t\_train=[]

me\_t\_test\_1stp=[]

me\_t\_test\_dyn=[]

me\_t\_train=[]

test\_predict\_sum\_t\_1stp=[]

test\_predict\_sum\_t\_dyn=[]

train\_predict\_sum\_t=[]

for j in range(repeats): # Repeat Times

model=Sequential()

model.add(LSTM(n,input\_shape = (time\_window,1), batch\_size=1,stateful=True))

model.add(Dropout(0.2))

model.add(Dense(1,activation='linear'))

# compile model for use

start=time.time()

model.compile(loss="mse", optimizer="rmsprop")

print("Compilation Time : ", time.time() - start)

train\_loss\_t=[]

cv\_loss\_t=[]

for i in range(epochs):

history=model.fit(Xtrain,ytrain, epochs=1, batch\_size=1, verbose=2,validation\_split=0.1,shuffle=False)

train\_loss\_t.append(history.history['loss'])

cv\_loss\_t.append(history.history['val\_loss'])

model.reset\_states()

################################# Predict and Loss Calculation #############################

# 1 Step Ahead Prediction -Out of Sample

test\_predict\_1stp = model.predict(Xtest,batch\_size=1)

test\_predict\_1stp=scaler.inverse\_transform(test\_predict\_1stp)

test\_predict\_sum\_t\_1stp.append(np.array(test\_predict\_1stp))

# Dynamic Prediction - Out of Sample

dynamic\_prediction=season\_diff[-len\_window:-test\_size]

for i in range(len\_window-test\_size,len\_window):

last\_feature=dynamic\_prediction[i-time\_window:i].reshape(1,time\_window, 1)

# last\_feature.reshape(1,time\_window, 1)

next\_pred=model.predict(last\_feature)

dynamic\_prediction=np.append(dynamic\_prediction,next\_pred)

dynamic\_prediction=dynamic\_prediction.reshape(-1,1)

dynamic\_prediction=scaler.inverse\_transform(dynamic\_prediction)

dynamic\_prediction=dynamic\_prediction[-test\_size:]

test\_predict\_sum\_t\_dyn.append(np.array(dynamic\_prediction))

# Training Prediction - In Sample

train\_predict=model.predict(Xtrain,batch\_size=1)

train\_predict=scaler.inverse\_transform(train\_predict)

train\_predict\_sum\_t.append(np.array(train\_predict))

############### Test Err Calc

# 1 Step Ahead Error Calc

rmse\_t\_test\_1stp.append(math.sqrt(mean\_squared\_error(y\_test\_raw,test\_predict\_1stp))) # RMSE

mad\_t\_test\_1stp.append(mean\_absolute\_error(y\_test\_raw,test\_predict\_1stp)) # MAD

mape\_t\_test\_1stp.append(mean\_absolute\_percentage\_error(y\_test\_raw,test\_predict\_1stp)) # MAPE

me\_t\_test\_1stp.append(mean\_error(y\_test\_raw,test\_predict\_1stp)) # ME

# Dynamic Error Calc

rmse\_t\_test\_dyn.append(math.sqrt(mean\_squared\_error(y\_test\_raw,dynamic\_prediction))) # RMSE

mad\_t\_test\_dyn.append(mean\_absolute\_error(y\_test\_raw,dynamic\_prediction)) # MAD

mape\_t\_test\_dyn.append(mean\_absolute\_percentage\_error(y\_test\_raw,dynamic\_prediction)) # MAPE

me\_t\_test\_dyn.append(mean\_error(y\_test\_raw,dynamic\_prediction)) # ME

############## Training Err Calc

rmse\_t\_train.append(math.sqrt(mean\_squared\_error(y\_train\_raw,train\_predict))) # RMSE

mad\_t\_train.append(mean\_absolute\_error(y\_train\_raw,train\_predict)) # MAD

mape\_t\_train.append(mean\_absolute\_percentage\_error(y\_train\_raw,train\_predict)) # MAPE

me\_t\_train.append(mean\_error(y\_train\_raw,train\_predict)) #ME

################ Cross validation Loss Generated by Model

cv\_loss.append(np.array(cv\_loss\_t))

cv\_loss\_res.append(np.array(cv\_loss\_t)\*diff\_std)

############### Training loss Loss Generated by Model

train\_loss.append(np.array(train\_loss\_t))

train\_loss\_res.append(np.array(train\_loss\_t)\*diff\_std)

######### Model Combination Predicting - Average the Repeat results as our final model

test\_predict\_sum\_1stp.append(np.mean(np.array(test\_predict\_sum\_t\_1stp),axis=0)) # 1 Step Ahead Forecst

test\_predict\_sum\_dyn.append(np.mean(np.array(test\_predict\_sum\_t\_dyn),axis=0)) # Dynamic Forecast

train\_predict\_sum.append(np.mean(np.array(train\_predict\_sum\_t),axis=0)) # Training

####################### Test Error Summary ##############################

# 1 Step Ahead Forecast

# RMSE

rmse\_t\_test\_1stp.append(np.std(rmse\_t\_test\_1stp))# std calculation

rmse\_t\_test\_1stp.append(math.sqrt(mean\_squared\_error(y\_test\_raw,np.mean(np.array(test\_predict\_sum\_t\_1stp),axis=0)))) # append Combination RMSE

test\_loss\_rmse\_1stp.append(rmse\_t\_test\_1stp)

# MAD

mad\_t\_test\_1stp.append(np.std(mad\_t\_test\_1stp)) # std calculation

mad\_t\_test\_1stp.append(mean\_absolute\_error(y\_test\_raw,np.mean(np.array(test\_predict\_sum\_t\_1stp),axis=0)))# append Combination MAD

test\_loss\_mad\_1stp.append(mad\_t\_test\_1stp)

# MAPE

mape\_t\_test\_1stp.append(np.std(mape\_t\_test\_1stp))

mape\_t\_test\_1stp.append(mean\_absolute\_percentage\_error(y\_test\_raw,np.mean(np.array(test\_predict\_sum\_t\_1stp),axis=0))) # append Combination MAPE

test\_loss\_mape\_1stp.append(mape\_t\_test\_1stp)

# ME

me\_t\_test\_1stp.append(np.std(me\_t\_test\_1stp))

me\_t\_test\_1stp.append(mean\_error(y\_test\_raw,np.mean(np.array(test\_predict\_sum\_t\_1stp),axis=0))) # append Combination MAPE

test\_loss\_me\_1stp.append(me\_t\_test\_1stp)

# Dynamic Forecast

# RMSE

rmse\_t\_test\_dyn.append(np.std(rmse\_t\_test\_dyn))# std calculation

rmse\_t\_test\_dyn.append(math.sqrt(mean\_squared\_error(y\_test\_raw,np.mean(np.array(test\_predict\_sum\_t\_dyn),axis=0)))) # append Combination RMSE

test\_loss\_rmse\_dyn.append(rmse\_t\_test\_dyn)

# MAD

mad\_t\_test\_dyn.append(np.std(mad\_t\_test\_dyn)) # std calculation

mad\_t\_test\_dyn.append(mean\_absolute\_error(y\_test\_raw,np.mean(np.array(test\_predict\_sum\_t\_dyn),axis=0)))# append Combination MAD

test\_loss\_mad\_dyn.append(mad\_t\_test\_dyn)

# MAPE

mape\_t\_test\_dyn.append(np.std(mape\_t\_test\_dyn))

mape\_t\_test\_dyn.append(mean\_absolute\_percentage\_error(y\_test\_raw,np.mean(np.array(test\_predict\_sum\_t\_dyn),axis=0))) # append Combination MAPE

test\_loss\_mape\_dyn.append(mape\_t\_test\_dyn)

# ME

me\_t\_test\_dyn.append(np.std(me\_t\_test\_dyn))

me\_t\_test\_dyn.append(mean\_error(y\_test\_raw,np.mean(np.array(test\_predict\_sum\_t\_dyn),axis=0))) # append Combination MAPE

test\_loss\_me\_dyn.append(me\_t\_test\_dyn)

###################### Training Error Summary #############################

# RMSE

rmse\_t\_train.append(np.std(rmse\_t\_train)) # std calculation

rmse\_t\_train.append(math.sqrt(mean\_squared\_error(y\_train\_raw,np.mean(np.array(train\_predict\_sum\_t),axis=0))))# append Combination RMSE

training\_final\_loss\_rmse.append(rmse\_t\_train)

# MAD

mad\_t\_train.append(np.std(mad\_t\_train)) # std calculation

mad\_t\_train.append(mean\_absolute\_error(y\_train\_raw,np.mean(np.array(train\_predict\_sum\_t),axis=0)))# append Combination MAD

training\_final\_loss\_mad.append(mad\_t\_train)

# MAPE

mape\_t\_train.append(np.std(mape\_t\_train)) # std calculation

mape\_t\_train.append(mean\_absolute\_percentage\_error(y\_train\_raw,np.mean(np.array(train\_predict\_sum\_t),axis=0)))# append Combination MAD

training\_final\_loss\_mape.append(mape\_t\_train)

# ME

me\_t\_train.append(np.std(me\_t\_train)) # std calculation

me\_t\_train.append(mean\_error(y\_train\_raw,np.mean(np.array(train\_predict\_sum\_t),axis=0)))# append Combination MAD

training\_final\_loss\_me.append(me\_t\_train)

# ### Result Summary

# #### Training Result

# ##### RMSE

# In[158]:

pd.DataFrame(training\_final\_loss\_rmse).round(3)

# ##### MAD

# In[159]:

pd.DataFrame(training\_final\_loss\_mad).round(3)

# ##### MAPE

# In[160]:

pd.DataFrame(training\_final\_loss\_mape).round(3)

# ##### ME

# In[161]:

pd.DataFrame(training\_final\_loss\_me).round(3)

# ###### Training In Sample Forecast

# In[162]:

train\_predict\_sum=np.array(train\_predict\_sum).reshape(-1,1)

train\_predict\_sum

# #### Test Result

# ##### RMSE

# ###### 1 Step Ahead

# In[163]:

pd.DataFrame(test\_loss\_rmse\_1stp)

# ###### Dynamic

# In[164]:

pd.DataFrame(test\_loss\_rmse\_dyn)

# ##### MAD

# ###### 1 Step Ahead

# In[165]:

pd.DataFrame(test\_loss\_mad\_1stp)

# ###### Dynamic

# In[166]:

pd.DataFrame(test\_loss\_mad\_dyn)

# ##### MAPE

# ###### 1 Step Ahead

# In[167]:

pd.DataFrame(test\_loss\_mape\_1stp)

# ###### Dynamic

# In[168]:

pd.DataFrame(test\_loss\_mape\_dyn)

# ##### ME

# ##### 1 Step Ahead

# In[169]:

pd.DataFrame(test\_loss\_me\_1stp)

# ###### Dynamic

# In[170]:

pd.DataFrame(test\_loss\_me\_dyn)

# ##### Test Out-of Sample Forecast

# ###### 1 Step Ahead

# In[171]:

test\_predict\_sum\_1stp=np.array(test\_predict\_sum\_1stp).reshape(-1,1)

test\_predict\_sum\_1stp

# In[181]:

pd.DataFrame(test\_predict\_sum\_1stp).to\_excel('1stp.xlsx')

# ###### Dynamic

# In[172]:

test\_predict\_sum\_dyn=np.array(test\_predict\_sum\_dyn).reshape(-1,1)

test\_predict\_sum\_dyn

# In[180]:

pd.DataFrame(test\_predict\_sum\_dyn).to\_excel('dyn.xlsx')

# ##### Data Visualization

# In[173]:

# Difference Data Prediction Visualization

fig=plt.figure()

date=pd.date\_range(start='11/10/2006',end='03/01/2017',freq='MS')

date=date[-len\_window:]

plt.plot(date,season\_diff[-len\_window:],label='Observation')

plt.plot(date[-test\_size:],test\_predict\_sum\_1stp,label='1-step Ahead Test Prediction')

plt.plot(date[-test\_size:],test\_predict\_sum\_dyn,label='Dynamic Test Prediction')

plt.plot(date[-len\_window:-test\_size],train\_predict\_sum,label='Training Prediction')

plt.ylabel('Pounds (Million)')

plt.xlabel('Year')

plt.legend()

plt.title(' In Sample & Test Prediction of 1st Order & Seasonal Difference Data')

plt.show()

fig.savefig('S1D1ForecaastFinal.png',dpi=1000)

# ##### Forecast Recovery and Visualization

# In[174]:

# Reshift Back to First Order Difference

season\_shift=sales\_diff.shift(12)[-len\_window:].values

# training

season\_fis\_forecast\_rec=train\_predict\_sum+season\_shift[-len\_window:-test\_size]

# 1 step ahead

season\_test\_1stp\_forecast\_rec=test\_predict\_sum\_1stp+season\_shift[-test\_size:]

# dynamic

season\_test\_dyn\_forecast\_rec=test\_predict\_sum\_dyn+season\_shift[-test\_size:]

# Reshift to Orignal Data

fst\_shift=data.shift(1)[-len\_window:].values

# training

fis\_forecast\_rec=season\_fis\_forecast\_rec+fst\_shift[-len\_window:-test\_size]

# 1 step ahead

test\_1stp\_forecast\_rec=season\_test\_1stp\_forecast\_rec+fst\_shift[-test\_size:]

# dynamic

test\_dyn\_forecast\_rec=season\_test\_dyn\_forecast\_rec+fst\_shift[-test\_size:]

# In[189]:

pd.DataFrame(test\_1stp\_forecast\_rec).to\_excel('1stp.xlsx')

pd.DataFrame(test\_dyn\_forecast\_rec).to\_excel('dyn.xlsx')

# In[175]:

# Difference Data Prediction Visualization

fig=plt.figure()

date=pd.date\_range(start='11/10/2006',end='03/01/2017',freq='MS')

date=date[-len\_window:]

plt.plot(date,sales[-len\_window:],label='Observation')

plt.plot(date[-test\_size:],test\_1stp\_forecast\_rec,label='1-step Ahead Test Prediction')

plt.plot(date[-test\_size:],test\_dyn\_forecast\_rec,label='Dynamic Test Prediction')

plt.plot(date[-len\_window:-test\_size],fis\_forecast\_rec,label='In Sample Prediction')

plt.ylabel('Pounds (Million)')

plt.xlabel('Year')

plt.legend()

plt.title(' In Sample & Test Prediction of Original Data')

plt.show()

fig.savefig('ForecaastFinal.png',dpi=1000)

# #### Training and Cross Validation Loss

# In[176]:

# Cross Validation Visualization

fig=plt.figure()

line1=plt.plot(np.array(train\_loss)[0,:],color='blue',label="Train Loss")

plt.plot(np.array(train\_loss)[1,:],color='blue')

plt.plot(np.array(train\_loss)[2,:],color='blue')

plt.plot(np.array(train\_loss)[3,:],color='blue')

plt.plot(np.array(train\_loss)[4,:],color='blue')

plt.plot(np.array(train\_loss)[5,:],color='blue')

plt.plot(np.array(train\_loss)[6,:],color='blue')

plt.plot(np.array(train\_loss)[7,:],color='blue')

plt.plot(np.array(train\_loss)[8,:],color='blue')

plt.plot(np.array(train\_loss)[9,:],color='blue')

plt.plot(np.array(train\_loss)[10,:],color='blue')

plt.plot(np.array(train\_loss)[11,:],color='blue')

plt.plot(np.array(train\_loss)[12,:],color='blue')

plt.plot(np.array(train\_loss)[13,:],color='blue')

plt.plot(np.array(train\_loss)[14,:],color='blue')

plt.plot(np.array(train\_loss)[15,:],color='blue')

plt.plot(np.array(train\_loss)[16,:],color='blue')

plt.plot(np.array(train\_loss)[17,:],color='blue')

plt.plot(np.array(train\_loss)[18,:],color='blue')

plt.plot(np.array(train\_loss)[19,:],color='blue')

plt.plot(np.array(train\_loss)[20,:],color='blue')

plt.plot(np.array(train\_loss)[21,:],color='blue')

plt.plot(np.array(train\_loss)[22,:],color='blue')

plt.plot(np.array(train\_loss)[23,:],color='blue')

plt.plot(np.array(train\_loss)[24,:],color='blue')

plt.plot(np.array(train\_loss)[25,:],color='blue')

plt.plot(np.array(train\_loss)[26,:],color='blue')

plt.plot(np.array(train\_loss)[27,:],color='blue')

plt.plot(np.array(train\_loss)[28,:],color='blue')

plt.plot(np.array(train\_loss)[29,:],color='blue')

line2=plt.plot(np.array(cv\_loss)[0,:],color='orange',label= "Cross Validation Loss")

plt.plot(np.array(cv\_loss)[1,:],color='orange')

plt.plot(np.array(cv\_loss)[2,:],color='orange')

plt.plot(np.array(cv\_loss)[3,:],color='orange')

plt.plot(np.array(cv\_loss)[4,:],color='orange')

plt.plot(np.array(cv\_loss)[5,:],color='orange')

plt.plot(np.array(cv\_loss)[6,:],color='orange')

plt.plot(np.array(cv\_loss)[7,:],color='orange')

plt.plot(np.array(cv\_loss)[8,:],color='orange')

plt.plot(np.array(cv\_loss)[9,:],color='orange')

plt.plot(np.array(cv\_loss)[10,:],color='orange')

plt.plot(np.array(cv\_loss)[11,:],color='orange')

plt.plot(np.array(cv\_loss)[12,:],color='orange')

plt.plot(np.array(cv\_loss)[13,:],color='orange')

plt.plot(np.array(cv\_loss)[14,:],color='orange')

plt.plot(np.array(cv\_loss)[15,:],color='orange')

plt.plot(np.array(cv\_loss)[16,:],color='orange')

plt.plot(np.array(cv\_loss)[17,:],color='orange')

plt.plot(np.array(cv\_loss)[18,:],color='orange')

plt.plot(np.array(cv\_loss)[19,:],color='orange')

plt.plot(np.array(cv\_loss)[20,:],color='orange')

plt.plot(np.array(cv\_loss)[21,:],color='orange')

plt.plot(np.array(cv\_loss)[22,:],color='orange')

plt.plot(np.array(cv\_loss)[23,:],color='orange')

plt.plot(np.array(cv\_loss)[24,:],color='orange')

plt.plot(np.array(cv\_loss)[25,:],color='orange')

plt.plot(np.array(cv\_loss)[26,:],color='orange')

plt.plot(np.array(cv\_loss)[27,:],color='orange')

plt.plot(np.array(cv\_loss)[28,:],color='orange')

plt.plot(np.array(cv\_loss)[29,:],color='orange')

plt.legend()

plt.xlabel('Epochs')

plt.ylabel('RMSE(Scaled)')

plt.title('Train Loss & Cross Validation Loss')

plt.show()

fig.savefig('loss\_output.png',dpi=1000)

# ### Result Interpret Visualization Plot

# #### Box Plot of Model Tuning Process

# In[178]:

t3\_data=pd.read\_csv('test\_loss.csv')

t3\_data.head(10)

# In[179]:

ax=sns.boxplot(data=t3\_data, palette='Oranges')

ax.set\_title('Time Window 3, Epochs 100 Neurons Box Plot')

ax.set\_ylabel('RMSE Unit: Pounds (Million)')

ax.set\_xlabel('Neurons')

ax.figure.savefig('T3NeuronsBoxPlot.png',dpi=1000)

**Model Combination:**

# -\*- coding: utf-8 -\*-

"""

"""

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

#%%

# import data

error\_hw=pd.read\_excel('error\_hw.xlsx')

error\_sarima=pd.read\_excel('error\_sarima.xlsx')

error\_nn=pd.read\_excel('err.xlsx')

forecast\_hw=pd.read\_excel('forecast\_hw.xlsx')

forecast\_sarima=pd.read\_excel('forecast\_sarima.xlsx')

forecast\_nn=pd.read\_excel('forecast.xlsx')

columns=['HW','SARIMA','NN']

error\_dynamic=pd.concat([error\_hw['Dynamic'],error\_sarima['Dynamic'],error\_nn['Dynamic']],axis=1)

error\_one\_step=pd.concat([error\_hw['One-step'],error\_sarima['One-step'],error\_nn['Dynamic']],axis=1)

error\_dynamic.columns=columns

error\_one\_step.columns=columns

forecast\_dynamic=pd.concat([forecast\_hw['Dynamic'],forecast\_sarima['Dynamic'],forecast\_nn['Dynamic']],axis=1)

forecast\_one\_step=pd.concat([forecast\_hw['One-step'],forecast\_sarima['One-step'],forecast\_nn['Dynamic']],axis=1)

forecast\_dynamic.columns=columns

forecast\_one\_step.columns=columns

#%%

# define functions to find weights

# combining two models

def weights(error1,error2):

sigma1=error1.std()

sigma2=error2.std()

comatrix=np.corrcoef(error1.values,error2.values)

rho=comatrix[0,1]

w1=(sigma2\*\*2-rho\*sigma1\*sigma2)/(sigma1\*\*2+sigma2\*\*2-2\*rho\*sigma1\*sigma2)

w2=1-w1

weight=[w1,w2]

return weight

# combining three models

def weights\_three(error1, error2, error3):

error\_df=pd.concat([error1,error2,error3],axis=1)

coef=error\_df.cov()

coef\_matrix=np.mat(coef.values)

i=np.mat(np.ones((3,1)))

w=(coef\_matrix.I)\*i/(i.T\*(coef\_matrix.I)\*i)

return w

#%%

#%% weights of HW and SARIMA

weight\_HW\_SARIMA\_dynamic=weights(error\_dynamic['HW'],error\_dynamic['SARIMA'])

weight\_HW\_SARIMA\_OneStep=weights(error\_one\_step['HW'],error\_one\_step['SARIMA'])

#%% weights of HW and NN

weight\_HW\_NN\_dynamic=weights(error\_dynamic['HW'],error\_dynamic['NN'])

weight\_HW\_NN\_OneStep=weights(error\_one\_step['HW'],error\_one\_step['NN'])

#%% weights of SARIMA and NN

weight\_SARIMA\_NN\_dynamic=weights(error\_dynamic['SARIMA'],error\_dynamic['NN'])

weight\_SARIMA\_NN\_OneStep=weights(error\_one\_step['SARIMA'],error\_one\_step['NN'])

#%% weights of three models

weight\_three\_dynamic=weights\_three(error\_dynamic['HW'],error\_dynamic['SARIMA'],error\_dynamic['NN'])

weight\_three\_OneStep=weights\_three(error\_one\_step['HW'],error\_one\_step['SARIMA'],error\_one\_step['NN'])

#%% four criteria assessing combined forecast

# import test set

data = pd.read\_excel('uk\_internet\_retail\_sales.xlsx', dayfirst = True, parse\_data = [0])

sales = data['Sales']

test = sales.iloc[-12:]

# define 4 criteria

def criteria(targets, predictions):

me=(targets - predictions).mean()

mad=np.absolute(targets - predictions).mean()

rmse=np.sqrt(((targets - predictions)\*\*2).mean())

mape=100\*(np.absolute(targets-predictions)/targets).mean()

results=[me,mad,rmse,mape]

return results

# calculate combined forecast

# Dynamic

# HW&SARIMA

equal\_HW\_SARIMA\_dynamic=forecast\_dynamic['HW']\*0.5 + forecast\_dynamic['SARIMA']\*0.5

varmin\_HW\_SARIMA\_dynamic=forecast\_dynamic['HW']\*weight\_HW\_SARIMA\_dynamic[0] + forecast\_dynamic['SARIMA']\*weight\_HW\_SARIMA\_dynamic[1]

criteria\_HW\_SARIMA\_dynamic\_equal=criteria(test.values, equal\_HW\_SARIMA\_dynamic.values)

criteria\_HW\_SARIMA\_dynamic=criteria(test.values, varmin\_HW\_SARIMA\_dynamic.values)

# HW&NN

equal\_HW\_NN\_dynamic=forecast\_dynamic['HW']\*0.5 + forecast\_dynamic['NN']\*0.5

varmin\_HW\_NN\_dynamic=forecast\_dynamic['HW']\*weight\_HW\_NN\_dynamic[0] + forecast\_dynamic['NN']\*weight\_HW\_NN\_dynamic[1]

criteria\_HW\_NN\_dynamic\_equal=criteria(test.values, equal\_HW\_NN\_dynamic.values)

criteria\_HW\_NN\_dynamic=criteria(test.values, varmin\_HW\_NN\_dynamic.values)

# SARIMA&NN

equal\_SARIMA\_NN\_dynamic=forecast\_dynamic['SARIMA']\*0.5 + forecast\_dynamic['NN']\*0.5

varmin\_SARIMA\_NN\_dynamic=forecast\_dynamic['SARIMA']\*weight\_SARIMA\_NN\_dynamic[0] + forecast\_dynamic['NN']\*weight\_SARIMA\_NN\_dynamic[1]

criteria\_SARIMA\_NN\_dynamic\_equal=criteria(test.values, equal\_SARIMA\_NN\_dynamic.values)

criteria\_SARIMA\_NN\_dynamic=criteria(test.values, varmin\_SARIMA\_NN\_dynamic.values)

# 3 MODELS

equal\_3models\_dynamic=forecast\_dynamic['HW']\*(1/3)+forecast\_dynamic['SARIMA']\*(1/3) + forecast\_dynamic['NN']\*(1/3)

varmin\_3models\_dynamic=forecast\_dynamic['HW']\*weight\_three\_dynamic[0,0]+forecast\_dynamic['SARIMA']\*weight\_three\_dynamic[1,0]+forecast\_dynamic['NN']\*weight\_three\_dynamic[2,0]

criteria\_3models\_dynamic\_equal=criteria(test.values, equal\_3models\_dynamic)

criteria\_3models\_dynamic=criteria(test.values, varmin\_3models\_dynamic)

# One-step

# HW&SARIMA

equal\_HW\_SARIMA\_one\_step=forecast\_one\_step['HW']\*0.5 + forecast\_one\_step['SARIMA']\*0.5

varmin\_HW\_SARIMA\_one\_step=forecast\_one\_step['HW']\*weight\_HW\_SARIMA\_OneStep[0] + forecast\_one\_step['SARIMA']\*weight\_HW\_SARIMA\_OneStep[1]

criteria\_HW\_SARIMA\_OneStep\_equal=criteria(test.values, equal\_HW\_SARIMA\_dynamic.values)

criteria\_HW\_SARIMA\_OneStep=criteria(test.values, varmin\_HW\_SARIMA\_one\_step.values)

# HW&NN

equal\_HW\_NN\_one\_step=forecast\_one\_step['HW']\*0.5 + forecast\_one\_step['NN']\*0.5

varmin\_HW\_NN\_one\_step=forecast\_one\_step['HW']\*weight\_HW\_NN\_OneStep[0] + forecast\_one\_step['NN']\*weight\_HW\_NN\_OneStep[1]

criteria\_HW\_NN\_OneStep\_equal=criteria(test.values, equal\_HW\_NN\_one\_step.values)

criteria\_HW\_NN\_OneStep=criteria(test.values, varmin\_HW\_NN\_one\_step.values)

# SARIMA&NN

equal\_SARIMA\_NN\_one\_step=forecast\_one\_step['SARIMA']\*0.5 + forecast\_one\_step['NN']\*0.5

varmin\_SARIMA\_NN\_one\_step=forecast\_one\_step['SARIMA']\*weight\_SARIMA\_NN\_OneStep[0] + forecast\_one\_step['NN']\*weight\_SARIMA\_NN\_OneStep[1]

criteria\_SARIMA\_NN\_OneStep\_equal=criteria(test.values, equal\_SARIMA\_NN\_one\_step.values)

criteria\_SARIMA\_NN\_OneStep=criteria(test.values, varmin\_SARIMA\_NN\_one\_step.values)

# 3 MODELS

equal\_3models\_one\_step=forecast\_one\_step['HW']\*(1/3)+forecast\_one\_step['SARIMA']\*(1/3) + forecast\_one\_step['NN']\*(1/3)

varmin\_3models\_one\_step=forecast\_one\_step['HW']\*weight\_three\_OneStep[0,0]+forecast\_one\_step['SARIMA']\*weight\_three\_OneStep[1,0]+forecast\_one\_step['NN']\*weight\_three\_OneStep[2,0]

criteria\_3models\_OneStep\_equal=criteria(test.values, equal\_3models\_one\_step)

criteria\_3models\_OneStep=criteria(test.values, varmin\_3models\_one\_step)