Online Sales Forecasting

Vasileios Plessas

08/03/2022

Contents

1 Introduction

Forecasting is required in many situations: deciding whether to build another power generation plant in the next five years requires forecasts of future demand; scheduling staff in a call centre next week requires forecasts of call volumes; stocking an inventory requires forecasts of stock requirements (Hyndman 2021).

The predictability of an event or a quantity depends on several factors including:

1) How well we understand the factors that contribute to it;
2) How much data is available;
3) How similar the future is to the past;
4) Whether the forecasts can affect the thing we are trying to forecast.

In this analysis, we have assumed the role of a data scientist, assigned with the task of developing a time series forecasting model of $h = 28$ forecasting horizon. The dataset analysed was an extract from a UK online store's sales database located in the UCI Machine Learning Repository . It contained a year's worth of sales recorded for over three thousand items on a daily granularity level. Our time-series dataset was of a multi-variate nature which we then reduced into a univariate time-series in order to focus our analysis on a specific subset of the data.

The dataset contained a number of inconsistencies that needed to be normalised before we could commence our analysis, for example missing values, outliers, non-standardised product codes etc. For that reason, a significant part of this report has been devoted in exploratory data analysis and in time-series features analysis. Due to the large number of independent time series in the dataset (over three thousand product codes), we have decided to group our data in higher granularity product categories based on a two dimensional 80/20 rule. The new product hierarchy consisted of:

A = Products which make up the 80% of the company's revenue across the year. B = Products making the rest 20%

H = Products which drive 80% of the orders (Invoices) L = Products making up the rest 20%

We have combined above categories into a two by two table to review all combinations and isolated the products belonging in the "AH" intersection i.e., products of high revenue and large volume of orders which have the majority of impact on both the company's top and bottom lines. A time-series analysis has helped us identify the trend and weekly seasonality pattern in our data. We have used those features to fine tune the algorithm's used during the development of our model. Specifically, we've identified that there an upward additive trend in the series and that sales tend to peak around the middle of each week.

Our methodology commenced by splitting our data into a 28 days long dataset named online_store_validate which we'll be reserved until the very end of our analysis to test our model's accuracy, and a fit dataset named online_store_fit which contains all data prior to the validation set. We have further proceeded in splitting the online_store_fit data into train and test sets which will be used to develop our model. The algorithm with the best performance we'll be then recalculated over the entire online_store_fit dataset and produce 28 days of multi-step forecasts to compare with the actual values in the validation set.

We then proceeded with a mathematical overview and application of the forecasting methods and their corresponding algorithms as well as examining the score results of each algorithm's performance over the test dataset. We have provided a visual representation of the forecasts generated from the fitted values on the train set as well as their 80% and 95% confidence intervals. The methods used were a) The naive method (SNAIVE) with a trend and seasonal component applied (Hyndman 2021) , b) Linear Regression (TSLM) with trend and seasonality components (Hyndman 2021), and c) the Exponential Smoothing algorithm in its multiple versions for trend and seasonality adjustments (Hyndman 2021). Exponential Smoothing and specifically the Damped Trend with multiplicative error and seasonality had outperformed all other methods by producing a smaller MAPE = 25% when tested and validated against hold-out part of our dataset reserved for that final estimation.

To ensure we haven't over fitted our model and that is generalised enough to produce unbiased forecasts based on new data we have used timeseries bootstrapping to simulate 100 time series and produced individual forecasts using the Exponential Smoothing algorithm (Bergmeir, Hyndman, and Benítez 2016). We then averaged the individual forecasts ("forecast bagging") and measured the accuracy vs the validation dataset. We have seen a marginal accuracy improvement with a MAPE = 24%.

This analysis relies heavily on the work of and tools developed by Professor Rob J Hyndmand and his team at the Department of Econometrics & Business Statistics, Monash University, Australia. A lot of the code and theory in this analysis has been adapted and inspired from the The Forecasting Principles and Practice 3rd ed, online textbook produced by the same team.

2 Exploratory Analysis

In this early stage of our exploratory analysis, we have used the skimr package as way to quickly review different statistics about our dataset. Skimr provides a frictionless approach to summary statistics which conforms to the "principle of least surprise," displaying summary statistics the user can skim quickly to understand their data. It handles different data types and returns a skim_df object which can be included in a pipeline or displayed nicely for the human reader.

Based on Skimr's output we can see that our dataset consists of 541,909 rows and 8 columns in totat and of types character, numeric and timestamp. More interestingly , Skimr informs us that a couple of fields are incomplete. Specifically the item description and CustomerID. The latter is missing a significant percentage of it's values, so at this stage we are flagging it as a candidate to be ognored in our further analysis.

An important observation is regarding the Quantity and Price fields. We can observe variances ranging from large negative values to equally large positive ones. These are potential ouliers that we'll later identify and either remove or replace to avoid skewing the results of our analysis. Finally, the timestamp range is approximately a year with the earliest sale recorded in December 2010 and the latest one in December 2011.

2.1 First pass

Table 1: Data summary

Name	online_store
Number of rows	541909
Number of columns	8
Column type frequency:	
character	4
numeric	3
POSIXct	1
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
InvoiceNo	0	1	6	7	0	25900	0
StockCode	0	1	1	12	0	4070	0
Description	1454	1	1	35	0	4211	0
Country	0	1	3	20	0	38	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Quantity	0	1.00	9.55	218.08	-80995.00	1.00	3.00	10.00	80995	
UnitPrice	0	1.00	4.61	96.76	-11062.06	1.25	2.08	4.13	38970	
CustomerID	135080	0.75	15287.69	1713.60	12346.00	13953.00	15152.00	16791.00	18287	

Variable type: POSIXct

skim_variable	n_missing	complete_rate	min	max	median	n_unique
InvoiceDate	0	1	2010-12-01 08:26:00	2011-12-09 12:50:00	2011-07-19 17:17:00	23260

We have examined the bottom 10 prices in the dataset. As expected the large negative values observed earlier are present here and they are not related to products, rather financial adjustments. There's also a number of products with zero prices, meaning they have contrbuted to the company's revenue.

Similarly, for the top 10 prices , we can see that they correspond to customer fees and instruction manuals which are not considered as products.

Another important observation histogram that product codes (StockCode) appear to differentiate a lot, both in their formats and character length. Is there potentially a pattern that the majority of product codes follow? Figure 1, shows that the vast majority of codes follow a five character format, with a few codes being shorter and a few more longer than that.

We have first looked at the shorter length code and examine what products do they refer to. As expected they are not referring to actual products that can be traded.

However, product codes of length >5 are actual products whose code has a letter suffix. This could potentially indicated that they are part of an assortment or they are variations of a "parent" project.

Similarly, if we filter on the top 10 products based on Quantity. We can identify the large value observed earlier. The first and second observations, although they refer to actual products and have contributed to the company's revenue (price >0), they appear to have been one-offs. We can confirm this since the next invoice of a priced product (573008), has a much lower quantity.

At this point we can start cleaning up some of these inconsistencies and review again what our data look like. Specifically we have filtered out any entries with prices less than zero. Entries with negative quantities have also been removed as they in general refer to product returns, stock adjustments or damaged stock. We are interested in forecasting the volume of stock that the business will need to deliver to customers so negative quantities need to be factored out. Additionally, we have only kept entries whose product codes are of character length 5 or 6. That way our dataset will consists of actual products.

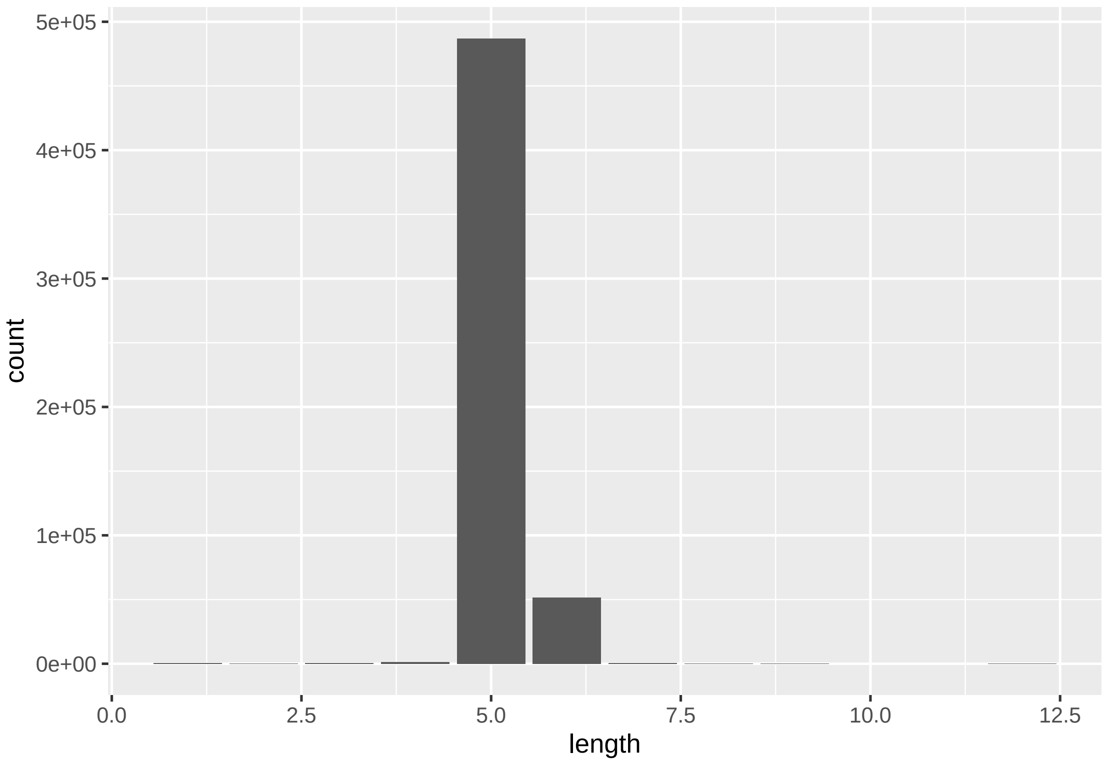

Figure 1: Product Code Character lengths

2.2 Second Pass

The new skimr output is a lot cleaner and something we can move into the next phase of our analysis.

Table 5: Data summary

Name	online_store
Number of rows	527340
Number of columns	8
Column type frequency:	
character	4
numeric	3
POSIXct	1
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
InvoiceNo	0	1	6	6	0	19769	0
StockCode	0	1	5	6	0	3897	0
Description	0	1	8	35	0	3992	0
Country	0	1	3	20	0	38	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Quantity	0	1.00	10.28	37.75	1.00	1.00	3.00	11.00	4800.0	
UnitPrice	0	1.00	3.26	4.38	0.04	1.25	2.08	4.13	649.5	
CustomerID	131296	0.75	15302.05	1710.00	12347.00	13975.00	15159.00	16804.00	18287.0	

Variable type: POSIXct

skim_variable	n_missing	complete_rate	min	max	median	n_unique
InvoiceDate	0	1	2010-12-01 08:26:00	2011-12-09 12:50:00	2011-07-20 13:30:00	18328

A final check we need to do with regards to product codes is to check if we'd left in any with a character length 5 but of non-numeric type as the actual product codes are. We have used the str_sub() function from the Stringr package to isolate the first 5 characters (since 6 letter products still have a letter suffix to them), then temporarily coerced the output into a numeric class. The logic is that is non numeric product codes exist, the coercion will introduce NAs which will then isolate and review. This check has returned zero entries so our product base is clean.

2.3 Performance by Country

Another interesting distribution to examine is that of revenue by country (see Figure 2). UK sales dominate our market spread, which is natural since our company is a UK online store. To ensure we reduce potentially variability in the sales data used in our analysis due to different buying patterns across countries, we will focus only on UK sales. Furthermore, we will remove the suffix letter from the six character length products identified earlier and aggregate all variations against their parent 5 letter code. Assortment mix planning is beyond the scope of this analysis and it is usually handled much later in the supply chain planning process.

2.4 Overall trend

Figure 3 shows the time series distribution of our final dataset. We fit a linear regression line to indicate that our sales data exhibit an identifiable upwards trend. We can also observe a few outliers that we hadn't detected in the previous steps. For these will, use some more sophisticated techniques later that would allows us to normalise our dataset better.

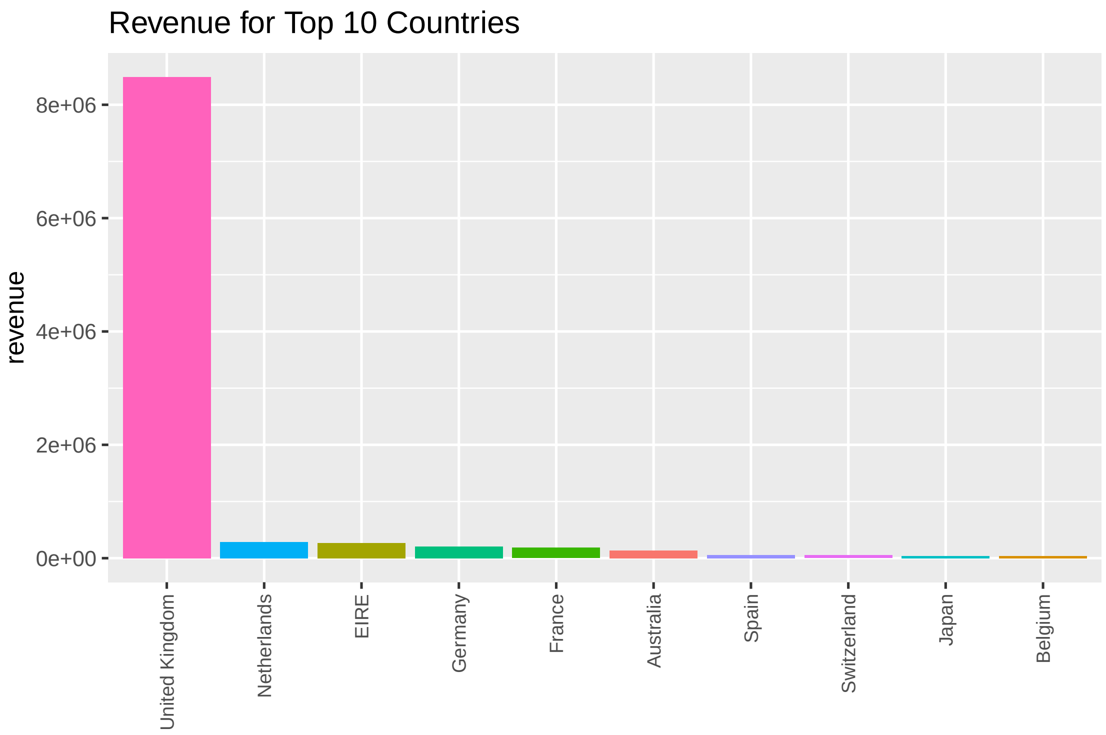

Figure 2: Revenue by Country

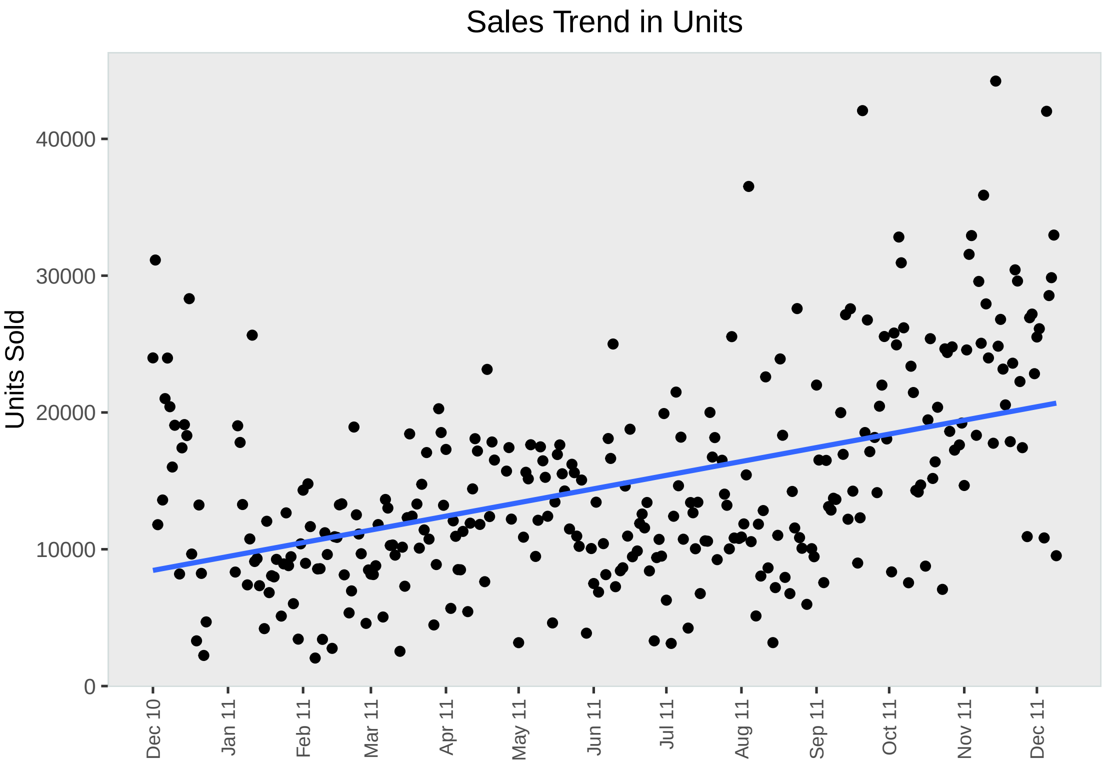

Figure 3: Time Series Distribution

2.5 Product Grouping

Our dataset is currently made of 3278 unique product codes. This is a huge number of individual time series to be able to forecast for. Moreover, there's significant variability across these series which makes it even more difficult to develop a generalised model that performs well overall.

Grouping products in higher granularities is a fundamental step to identify important trends in the data and unique features shared across product families. Each business has their own logic on how to group similar products together which is usually referred to as a product hierarchy. For a toy company for example this could be a case where all plush toys are grouped under the same family to differentiate them from products of different customer target groups such as action figures.

In our case and since we don't have any prior knowledge of these products we've decided to follow a two dimensional 80/20 rule and create an A,B - H,L product split based on Revenue and Order Volume.

A = Products which make up the 80% of the company's revenue across the year. B = Products making the rest 20%

H = Products which drive 80% of the orders (Invoices) L = Products making up the rest 20%

After assigning the those categories we have used the table() function to create a 2x2 table showing the number of products in each quadrant. Out 3278 , 716 (AH) of them are driving 80% of the company's revenue and orders. These are items that have an immediate impact both on the top line but also on the bottom line as a high order volume could incur large supply chain costs in order to successfully meet that customer demand.

There is a small number of products (AL) that also contribute to the 80% revenue baseline but they don't have a large numebr of orders agaisnt them. We can assume that these are high price point items or potentially infrequent bulk orders of items.

Perharps from a business perspective the items that will need to reviewed by management are the ones belonging to the BH category. That is products that drive a large number of orders but they only have minimal contribution to the revenue baseline. These products could significantly increase supply chain costs and can potentially have an impact on the company's bottom-line.

Finally the majority of the company's product codes fall under the low revenue/low orders bucket (BL). These products could potentially be old and discontinued products or simply not popular ones that have been marked down and the business is simply trying to offload from their inventories.

	H	L
A	716	69
B	487	2006

Figure 4 shows a faceted time-series plot which visualises each category's sales performance in units across the year. For the rest of our analysis we have focused on the AH category as these product are of the greatest strategic importance due to their contribution in the business's top and bottom lines.

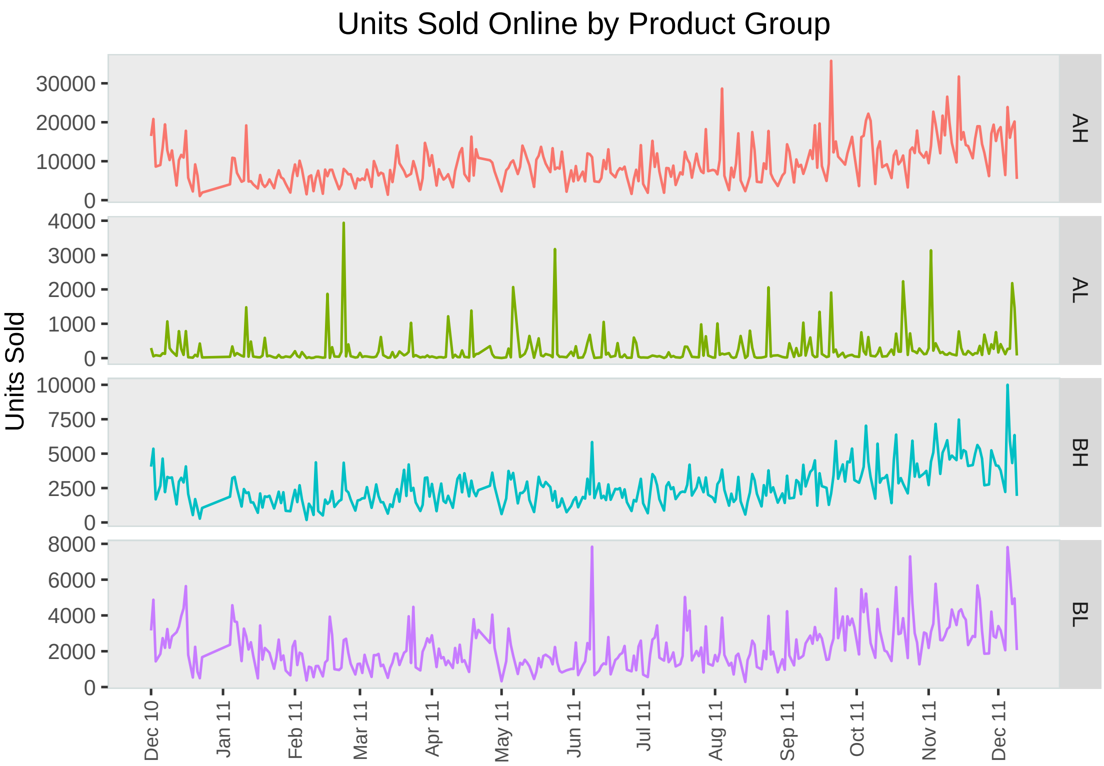

Figure 4: Sales Performance by Product Category

3 Time Series Analysis

For the following part of out this report we will be using the tidyverts packages which consist of :

tsibble : a data infrastructure for tidy temporal data with wrangling tools. feasts: a collection of tools for the analysis of time series data. fable : a collection of commonly used univariate and multivariate time series forecasting models including exponential smoothing via state space models.

A lot of the code has been inspired or adapted from Rob J Hyndman's the excellent online textbook on forecasting Forecasting Principles and Practice 3rd ed..

3.1 Convert into a tsibble object

To coerce a data frame to tsibble, we need to declare key and index. In our case the ab_hl column containing the categorical variables can be considered as key and the index is the InvoiceDate column containing temporal values. The Quantity column or any other numerical fields can be considered as measured variables. Printing out the top 6 tsibble rows shows the new class "A tsibble 6x3

$$1d$$

" indicating the number of rows and columns visible as well as the detected temporal frequency (daily). The printout also indicates what the key of this tsibble object.

The feasts package contains a very useful plotting function autoplot() which allows us to easily produce time-series plots from a tibble with minimum code. We have used this function across the rest of our analysis.

Figure 5 shows the time series for AH product sales in 2011. We have fitted a linear regression line to indicate the upward sales trend which we've detected earlier. At this point it's worth revisiting the previously observed outliers which are visible in the second half of the year.

We have proceed in applying common techniques to identify and smooth out those outliers in the following steps.

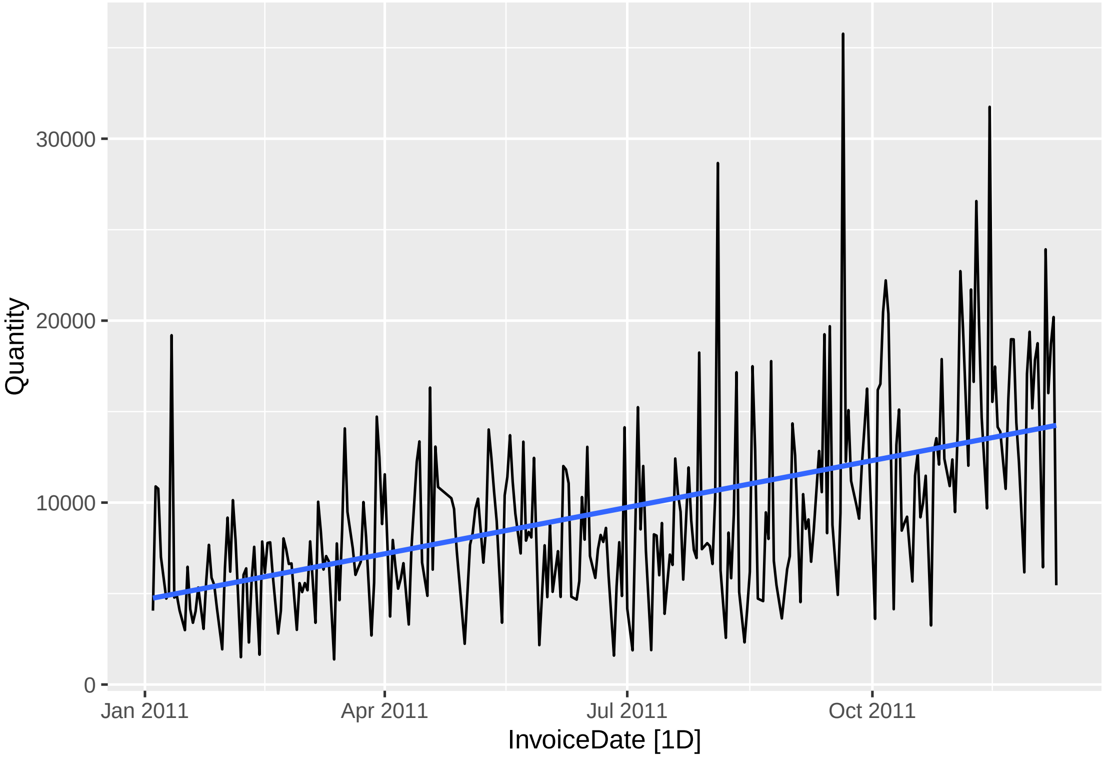

Figure 5: AH Product Sales Trend

3.2 Missing Values

Before proceeding any further we need to inspect our tsibble for any implicit gaps. This is an impotant step as a number of the models we'll try to fit in our dataset cannot process missing values. The has_gaps() function from the tsibble package has identified the existence of gaps in our set.

Figure 6 shows the distribution of gaps in our tsibble. It appears that we don't have a lot of consecutive gaps in our timeseries, rather intermitent gaps of one and a few of two and 3 consecutive ones. The count_gaps() function from the tsibble package has help in ths regard.

It is common practice in time series analysis to replace missing values with an approximate value of mean or median , fill with the previous or next non-blank observation or use a more advanced method for interpolation (Hyndman 2021). Here we have filled any implicit gaps with the last

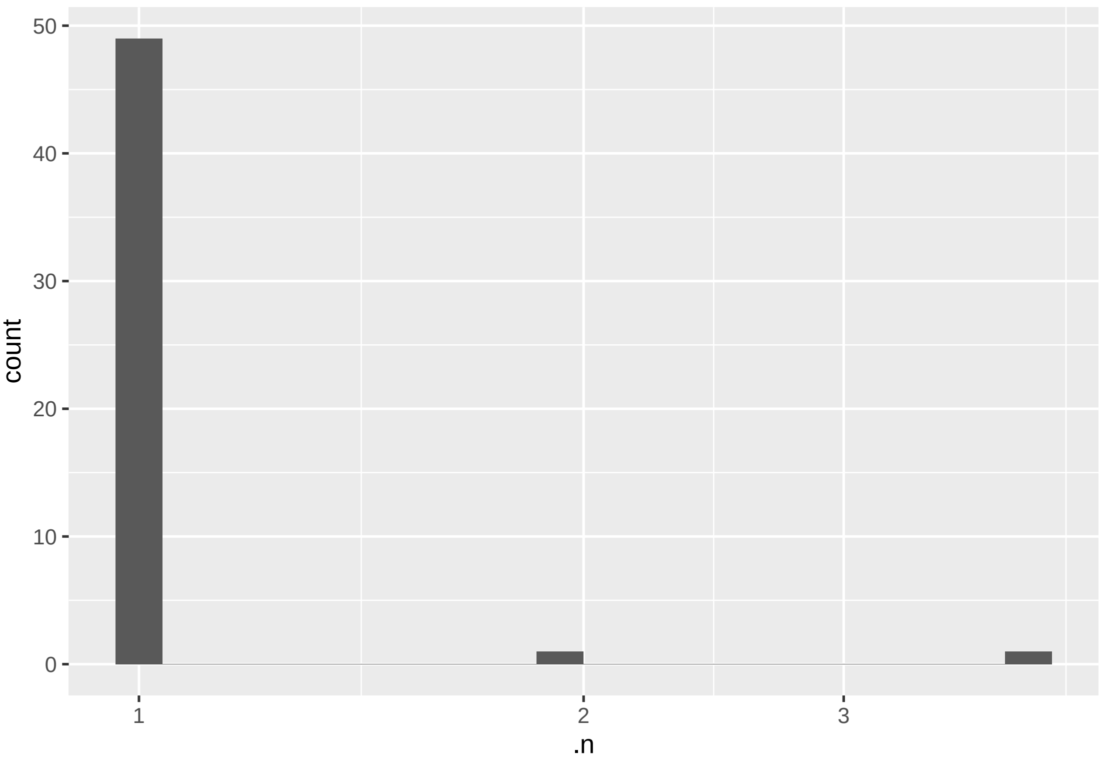

Figure 6: Gaps in the data

non-blank observation in the time series. Running the has_gaps() function again, verifies that we don't have any further gaps in our data.

3.3 Outliers

Now that our dataset is gap free, we can try and smooth out any outliers still present in our time series before we proceed in the modelling and forecasting phase. We will fit an STL Decomposition model in our data.Then any outliers should show up in the remainder series which will indicate any values that could not be explained by the model (Hyndman 2021). Figure 7 shows the output of our decomposed time series with the 3 previously suspected outliers clearly visible in the remainder part of the plot.

We have used a rule to define outliers as those that are greater than 3 interquartile ranges (IQRs) from the central 50% of the data, which would make only 1 in 500,000 normally distributed observations to be outliers. Below table show the output of this:

As a final step, we have removed those outliers from the dataset and have used an Auto Regressive Integrated Moving Average (ARIMA) model from the fable package to replace them with interpolation.

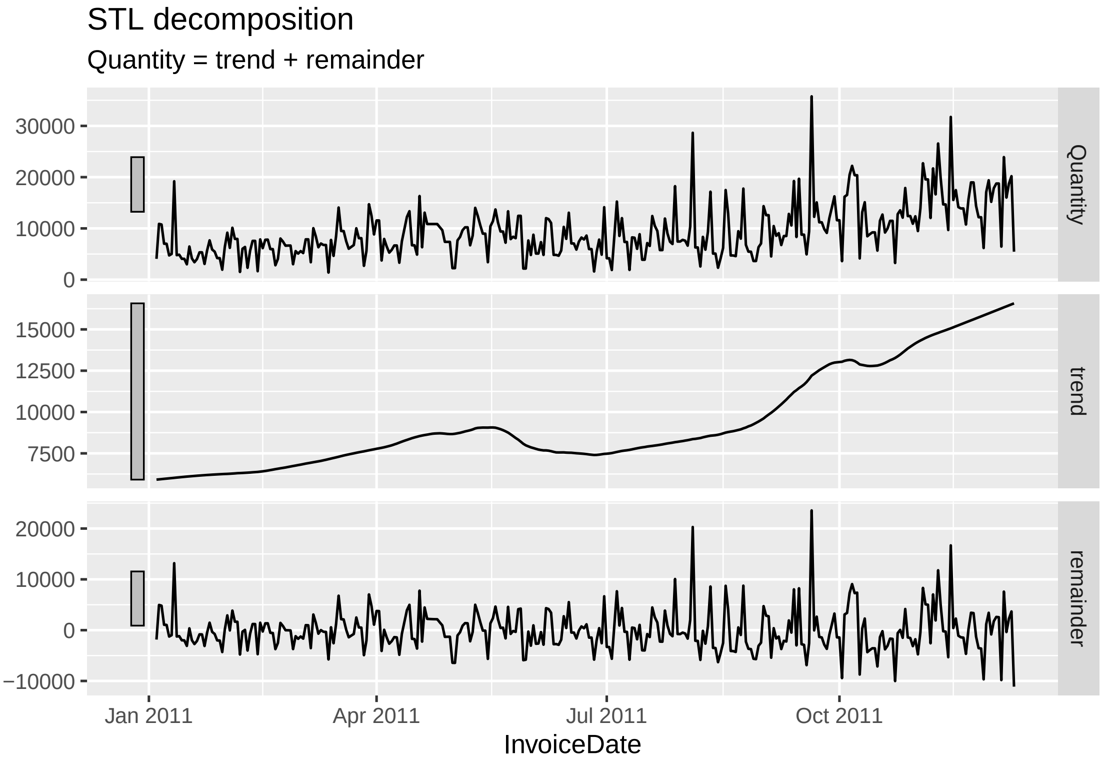

Figure 7: Time Series Decomposition

4 Time Series Features

Figure 8 shows the now normalised time series. Observe how the outliers are no longer skewing the distribution of the series.

4.1 Lag Plots

Our time series is univariate, meaning there's only one indipendent variance (time) interacting with the response variable (Units Sold). In such time series every time point can be thought as a feature by creating different time lags and examine the relationship to the response variable (Unwin 2015). Figure 9 shows lagged values of the time-series. Each graph shows y_t plotted against y_{t-k} for different values of k . The colours here indicate the days of the week. There seems to be a strong positive relationship for lag 7 which indicates the existence of a weekly seasonality.

4.2 Autocorrelation

Autocorrelation measures the extent of a linear relationship between lagged values of a time series. Lagged autocorrelation coefficients, for example r_1 measures the relationship between y_t and y_{t-1} , r_2 measures the relationship between y_t and y_{t-2} and so on.

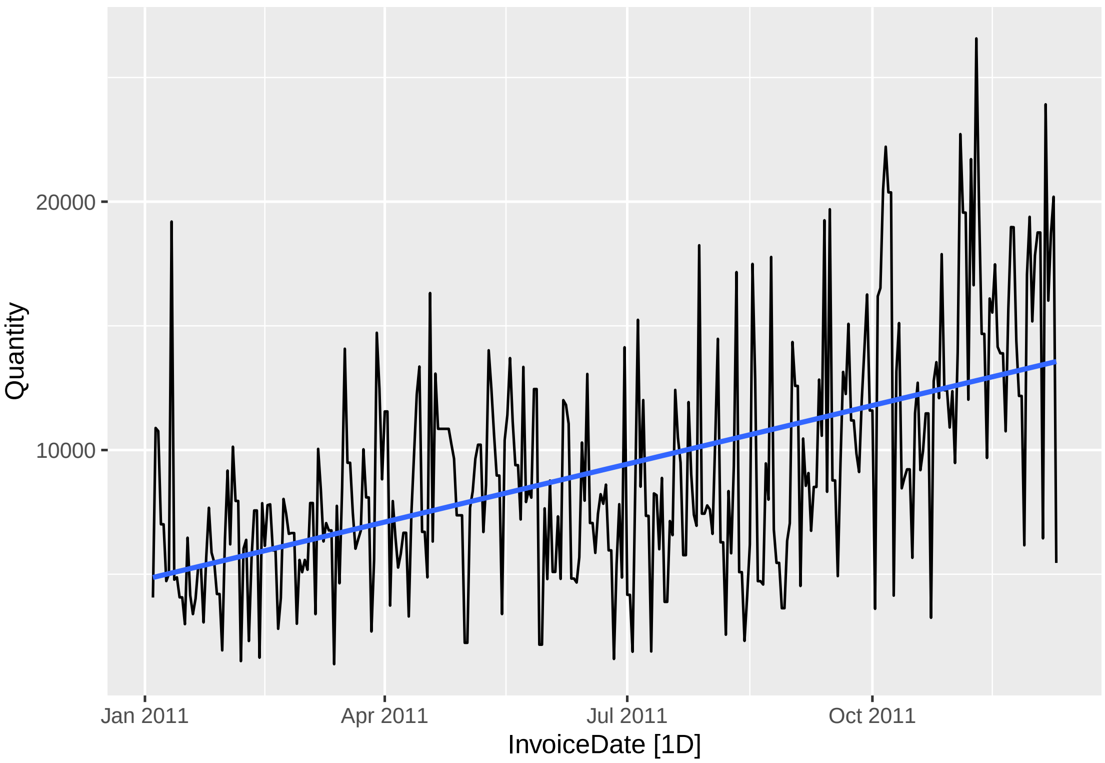

Figure 8: Normalised Time Series

The autocorrelation function can be written as

$$r_k = \frac{\sum\limits_{t=k+1}^{T} (y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum\limits_{t=1}^{T} (y_t - \bar{y})^2}$$

when T is the length of the time series (Hyndman 2021).

Figure 10 shows the autocorrelation plot (correlogram) for k lag $= 28$. r_7 , r_{14}, r_{21} and r_{28} are higher than the others indicating a visible weekly seasonal pattern in the data. The dashed blue line indicates where the correlations are significantly different from zero.

For data with trend like ours it's useful to increase the number of estimated lags (see Figure 11). When data have a trend, the autocorrelations for small lags tend to be large and positive because observations nearby in time are also nearby in value. So the ACF of a trended time series tends to have positive values that slowly decrease as the lags increase.

4.3 STL Decomposition

A time series decomposition can be used to measure the strength of trend and seasonality in a time series (Cleveland 1990). The decomposition is written as

$$y_t = T_t + S_t + R_t$$

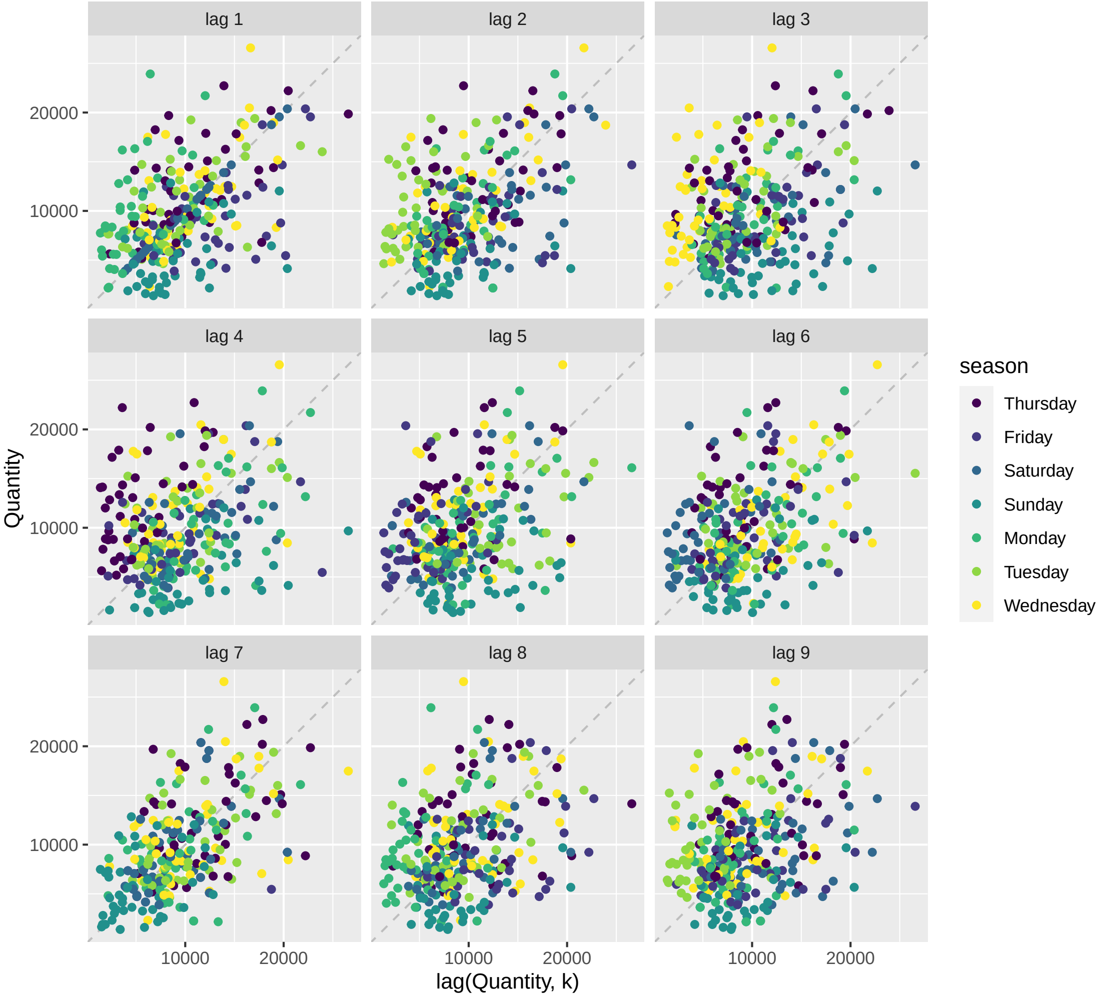

Figure 9: Lagged Plots

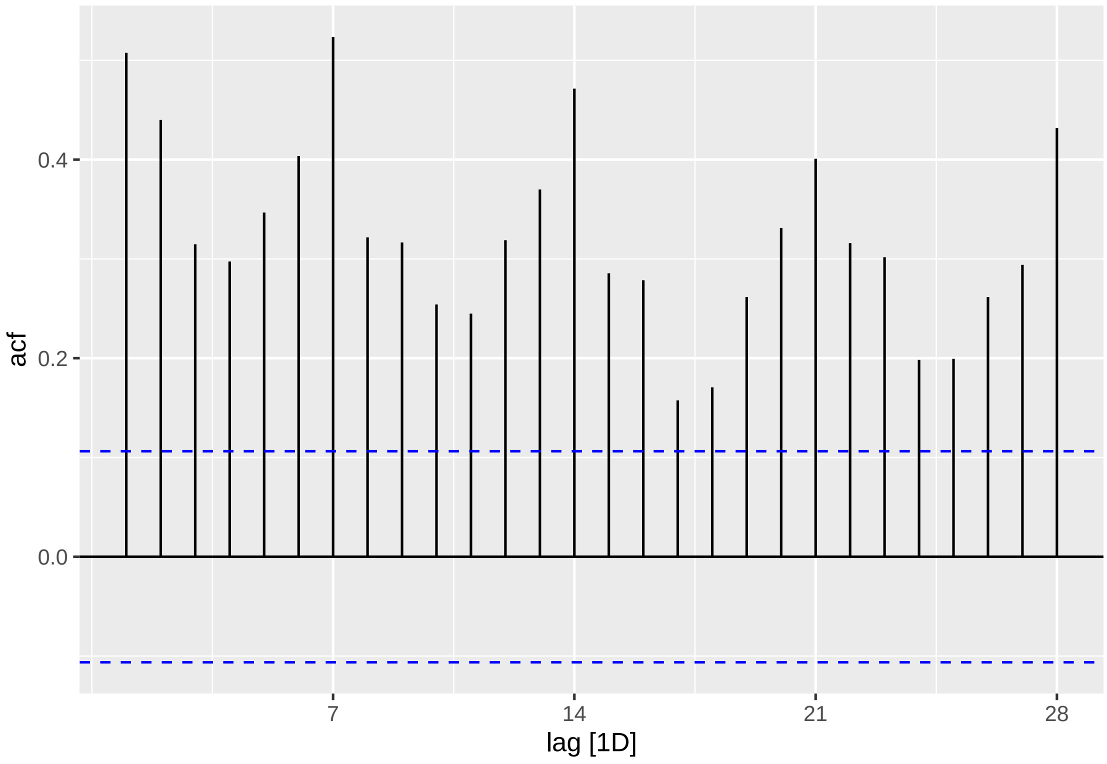

Figure 10: Auto Correlation Plot , lags=28

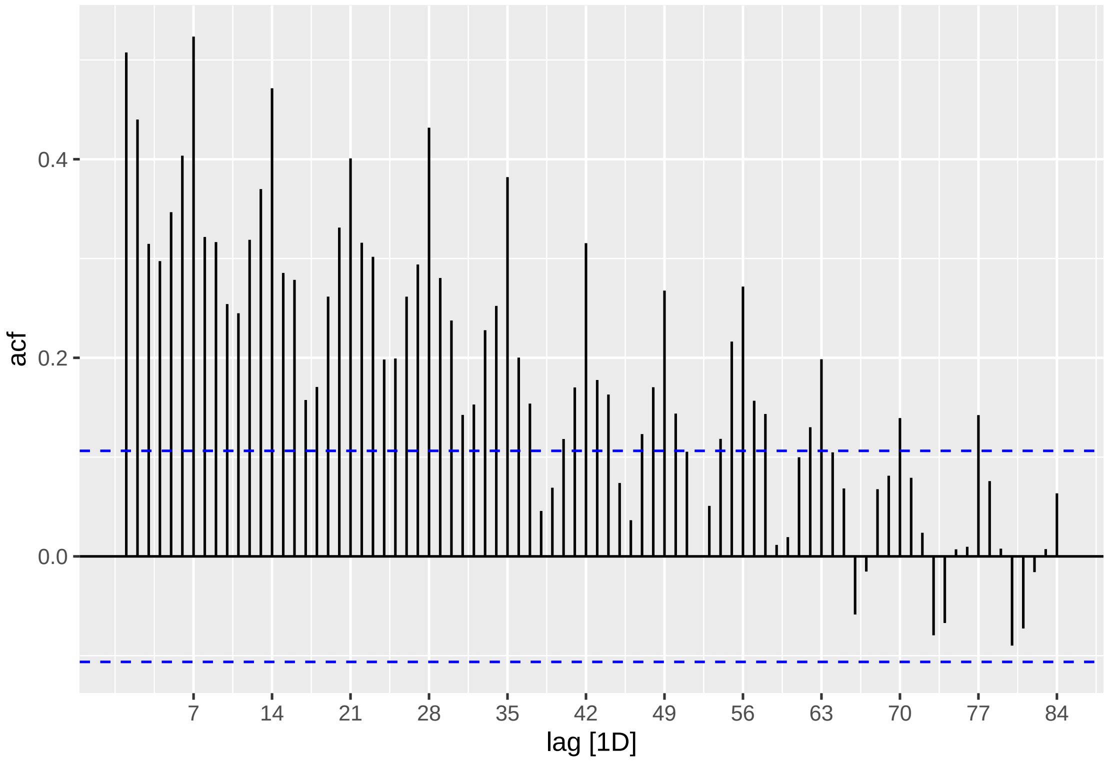

Figure 11: Autocorrelation Plot, lags = 84

where T_t is the smoothed trend component, $S_t t$ is the seasonal component and R_t is a remainder component. For strongly trended data, the seasonally adjusted data should have much more variation than the remainder component. But for data with little or no trend, the two variances should be approximately the same. So we define the strength of trend as:

$$F_T = \max\left(0, 1 - \frac{\text{Var}(R_t)}{\text{Var}(T_t + R_t)}\right)$$

This will give a measure of the strength of the trend between 0 and 1

The strength of the seasonality is defined similarly but with respect to the detrended data tather than the seasonally adjusted data:

$$F_S = \max\left(0, 1 - \frac{\text{Var}(R_t)}{\text{Var}(S_t + R_t)}\right)$$

A series with seasonal strength FS close to 0 exhibits almost no seasonality, while a series with strong seasonality will have F_S close to 1.

We can use the feat_stl() function from the feasts function to extract those features for one or multiple time series in our tsibble (Hyndman 2021), (Wang 2006) . Below we see the values for the trend and seasonal and trend strength of our time series as well as the peak and trough values in the season. We can see that within a week our series tends to peak on the 3rd day (Wednesday) and falls on the 6th (Saturday.)

We've previously fitted an STL Decomposition model, suppressing the seasonality in order to identify any potential outliers in our data. Figure 12 shows a full decomposition model with the smoothed upwards trend and the weekly seasonal pattern clearly visible. The remainder component shown in the bottom panel is what is left over when the seasonal and trend-cycle components have been subtracted from the data. These components can be added together to reconstruct the data shown in the top panel.

Correctly identifying the components tha make our time series is an important step to produce good forecast by selecting the appropriate model which fits our data best.

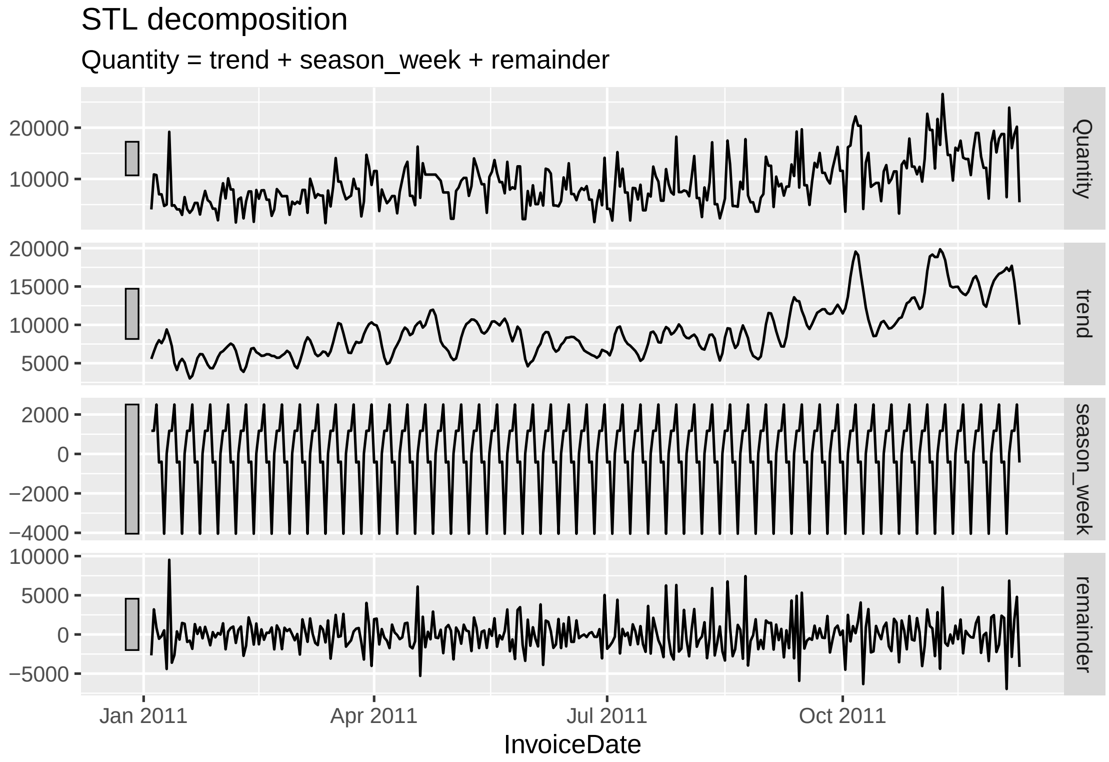

Figure 12: Time Series Components Decomposition

5 Forecasting Methodology

5.1 Train and Test sets

It is best practice to split our time series between train and test data sets. As we are testing out models to select the most appropriate one for our data, we train those models in the train data and we evaluate the performance on the test set which contains data that have not been used in the training process. That way we can avoid over-fitting and ensure our model generalises good enough to predict outcomes on new data.

For our methodology, we have split our data between the online_store_fit and online_store_validate (28 days) datasets. We have further split the online_store_fit set between the online_store_train and online_store_test (28 days) datasets.

We have then fitted multiple models on the online_store_train set and have measured their accuracy on the online_store_test dataset by producing one-step forecasts using the refit(). It is worth noting that each model tested is not re-estimated in this case. Instead, the model obtained previously (and stored as method_fit) is applied to the test data. Because the model was not re-estimated, the "residuals" obtained here are actually one-step forecast errors.

Finally we refitted all the trained models on the entire online_store_fit dataset and produced one-step forecast on teh online_store_validate dataset which until now had not been used. The model that yielded the smaller error was then selected to produce multiple step forecasts (or true forecasts) using the entire dataset and measure it's performance on the last 28 days of that set.

In order to ensure we have not over-fitted our model, we have used time series bootstrapping to generate 100 time series of bootstrap block size 28. We've then created simulated forecasts from those series using the winning model from our previous step and measured our accuracy again over the online_store_validate datasets and compared with our previous result.

5.2 Forecasting Methods

5.2.1 NAIVE

One of the simplest forecasting techniques available is the Naive method. For naïve forecasts, we simply set all forecasts to be the value of the last observation. This method is usually a good benchmark for other methods, or simply put if a more advanced method cannot beat the Naive's accuracy score with we might as well use the Naive one as our best estimation for demand.

The simple Naive can be written as :

$$\hat{y}_{T+h|T} = y_T$$

In our case because our data have a seasonality, we will use a variation fo the Naive method called Seasonal Naive (Hyndman 2021). In this case, we set each forecast to be equal to the last observed value from the same season. It can be written as :

$$\hat{y}_{T+h|T} = y_{T+h-m(k+1)},$$

where $m =$ the seasonal period and k is the integer part of $(h-1)/m$ the number of completed points in the period prior to time $(h-1)/m$.

We've fitted the SNAIVE algorithm on the training set and tested 28 one-step forecast on the test one. Figure 12 visualises the generated forecasts as well as their 80% and 95% confident intervals. The evaluation on the trained model yielded a MAPE of 28.9% which is will be our benchmark.

5.2.2 Linear Regression

A Linear Regression model in its simplest case, allows for a linear relationship between the forecast variable y and a single predictor variable x (Sheather 2009) :

$$y_t = \beta_0 + \beta_1 x_t + \varepsilon_t$$

The coefficients b_0 and b_1 denote the intercept and the slope of the line respectively. The intercept b_0 represents the predicted value of y when x=0. The slope b_1 represents the average predicted change in y resulting from a one unit increase in x. We can think of each observation yt as consisting of the systematic or explained part of the model,$\beta_0 + \beta_1 x_t$, and the random "error," ε_t. The linear regression equation is estimated using the TSLM() function from the fable package.

We will use the trend and seasonality variables we've identified earlier as predictor variables in the linear regression function (Hyndman 2021).

Figure 13 shows the plotted one step forecasts derived from fitting the linear regression algorithm on the train dataset. A MAPE of 28.2 has been observed which is only a marginal improvement over the SNAIVE one. However, unlike SNAIVE, the seasonal pattern looks more consistent and the prediction intervals are narrower.

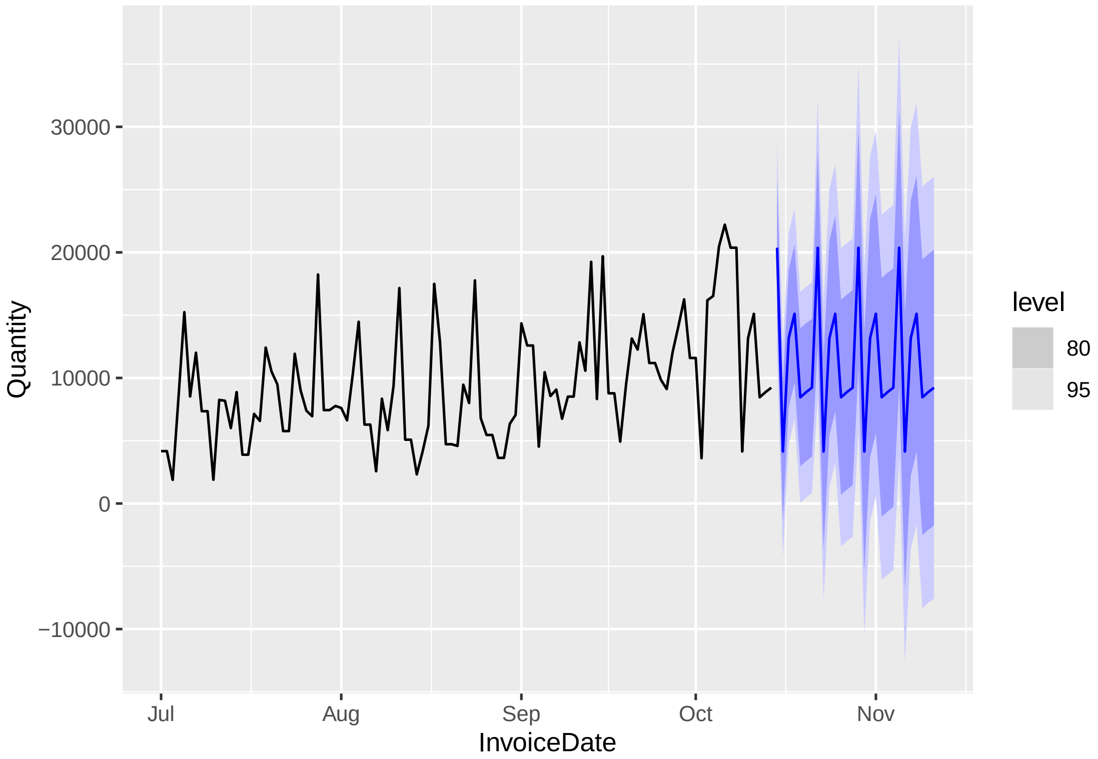

Figure 13: Seasonal Naive Forecasts

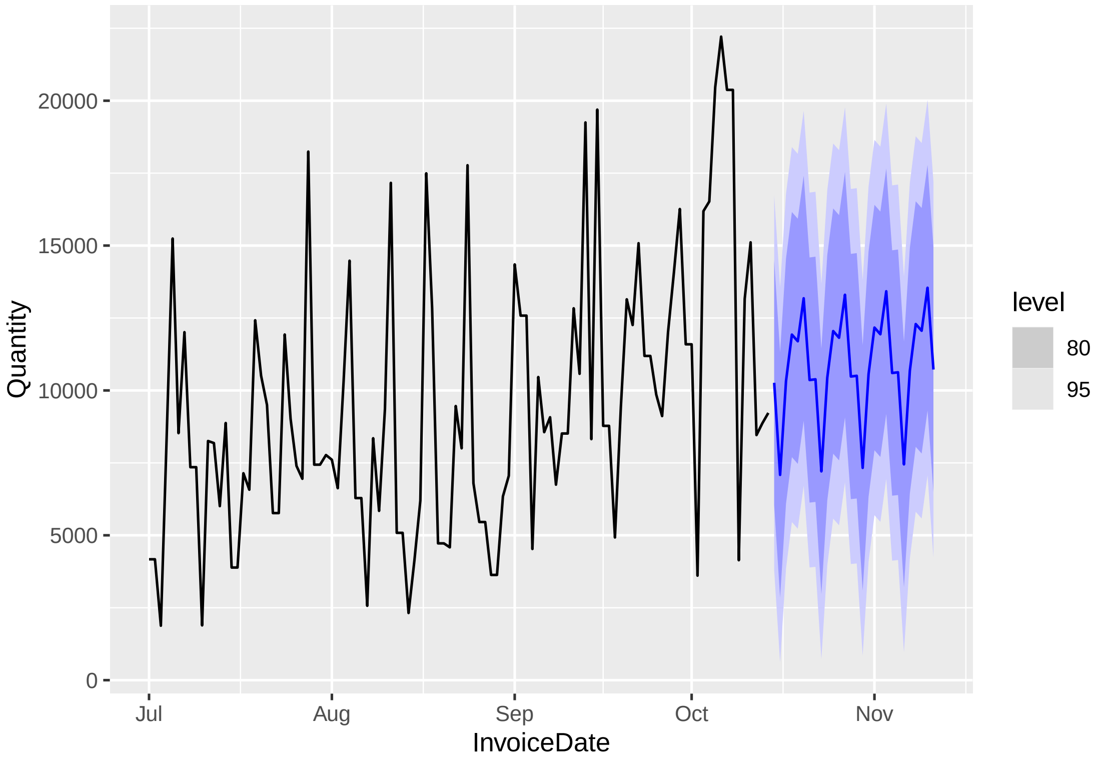

Figure 14: Linear Regression Forecasts with Seasonal and Trend Components

5.2.3 Exponential Smoothing

Exponential smoothing was proposed in the late 1950s (Brown 1959), (Holt 2004), (Winters 1960), and has motivated some of the most successful forecasting methods. Forecasts produced using exponential smoothing methods are weighted averages of past observations, with the weights decaying exponentially as the observations get older. In other words, the more recent the observation the higher the associated weight. This framework generates reliable forecasts quickly and for a wide range of time series, which is a great advantage and of major importance to applications in industry.

5.2.3.1 Simple Exponential Smoothing This method is suitable for forecasting data with no clear trend or seasonal pattern. Forecasts are calculated using weighted averages, where the weights decrease exponentially as observations come from further in the past — the smallest weights are associated with the oldest observations:

$$\hat{y}_{T+1|T} = \alpha y_T + \alpha(1-\alpha)y_{T-1} + \alpha(1-\alpha)^2 y_{T-2} + \cdots$$

where $0 <= a <= 1$ is the smoothing parameter. The one-step-ahead forecast for time $T+1$ is a weighted average of all of the observations in the series $y1, ..., yT$. The rate at which the weights decrease is controlled by the parameter a. For any a between 0 and 1, the weights attached to the observations decrease exponentially as we go back in time, hence the name “exponential smoothing.”

The weighted average form of the equation can be written as:

$$\hat{y}_{T+1|T} = \alpha y_T + (1-\alpha)\hat{y}_{T|T-1},$$

where $0 \leq a \leq 1$ is the smoothing parameter.

We fit a Simple Exponential Smoothing to our train data using the ETL function from fable package and by passing the parameters error(“A”) + trend(“N”) + season(“N”). “A” stands for additive, meaning the residual errors get added as the algorithm progresses over the time series, and “N” simply cancels out the algorithms function to search for trend and seasonality in the data (Hyndman 2021).

Single exponential smoothing produces flat forecasts as no trend or seasonality components are being accounted. Figure 14, visualises the produces forecasts for the next 28 days along with the 80% and 95% confidence intervals

Given both the trend and seasonal patterns in our data, it comes as no surprise that we don’t have a good MAPE result in this case.

5.2.3.2 Holt’s Exponential Smoothing Holt (1957) extended simple exponential smoothing to allow the forecasting of data with a trend. This method involves a forecast equation and two smoothing equations (one for the level and one for the trend):

where lt denotes an estimate of the level of the series at time t, bt denotes an estimate of the trend (slope) of the series at time t, a is the smoothing parameter for the level, $0 <= a <= 1$, and b∗ is the smoothing parameter for the trend, $0 <= b* <= 1$.

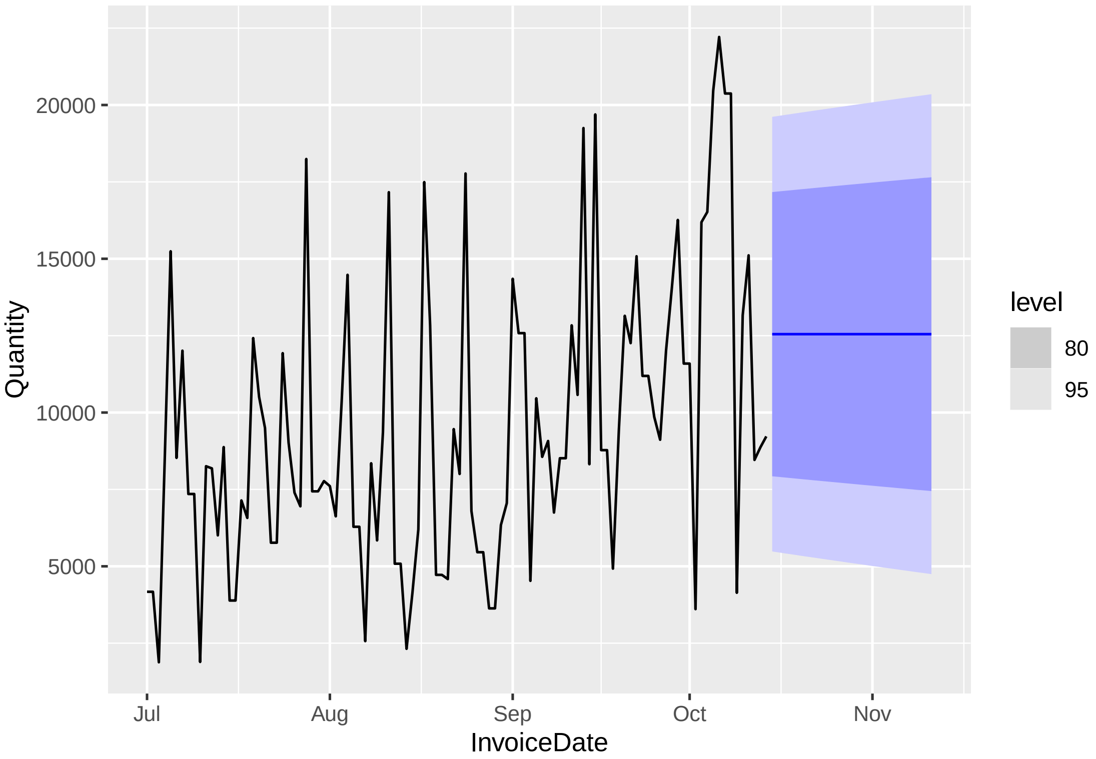

Figure 15: Simple Exponential Smoothing Flat Forecasts

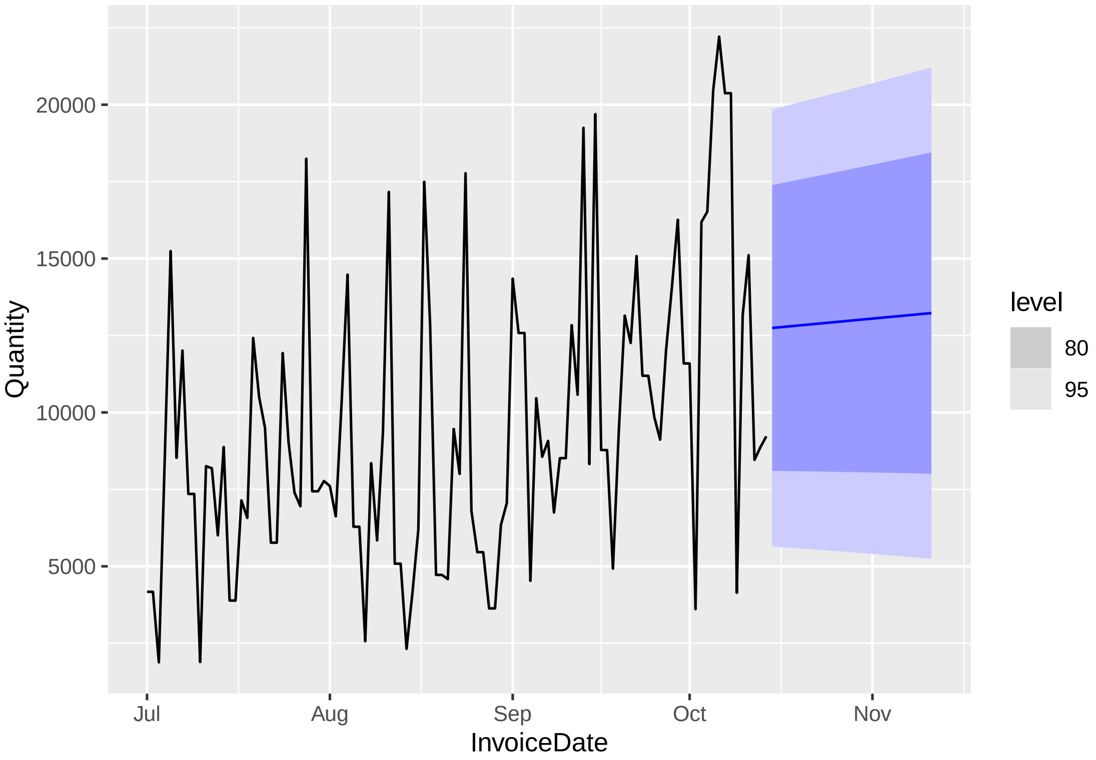

Figure 16: Trend Adjusted Exponential Smoothing Forecasts

The forecast function is no longer flat but trending (See Figure 15). The h-step-ahead forecast is equal to the last estimated level plus h times the last estimated trend value. Hence the forecasts are a linear function of h.

Also, there's been a noticeable improvement in the MAPE score which indicates that our algorithm has began to fit our training data better.

5.2.3.3 Holt Winter's Exponential Smoothing Holt (1957) and Winters (1960) extended Holt's method to capture seasonality. The Holt-Winters seasonal method comprises the forecast equation and three smoothing equations — one for the level *lt*, one for the trend *bt*, and one for the seasonal component *st*, with corresponding smoothing parameters *a*, *b*∗ and γ. We use m to denote the period of the seasonality, i.e., the number of seasons in a year.

There are two variations to this method that differ in the nature of the seasonal component. The additive method is preferred when the seasonal variations are roughly constant through the series, while the multiplicative method is preferred when the seasonal variations are changing proportional to the level of the series.

The component form for the additive method is:

$$\begin{aligned}\hat{y}_{t+h|t} &= \ell_t + hb_t + s_{t+h-m(k+1)} \\ \ell_t &= \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\ b_t &= \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1} \\ s_t &= \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m},\end{aligned} \tag{1}$$

where *k* is the integer part of (h—1)/m, which ensures that the estimates of the seasonal indices used for forecasting come from the final year of the sample. The level equation shows a weighted average between the seasonally adjusted observation (yt—st—m) and the non-seasonal forecast (*lt*—1+bt—1) for time t.

The component form for the multiplicative method is:

$$\begin{aligned}\hat{y}_{t+h|t} &= (\ell_t + hb_t)s_{t+h-m(k+1)} \\ \ell_t &= \alpha\frac{y_t}{s_{t-m}} + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\ b_t &= \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1} \\ s_t &= \gamma\frac{y_t}{(\ell_{t-1} + b_{t-1})} + (1-\gamma)s_{t-m}.\end{aligned} \tag{2}$$

A method that often provides accurate and robust forecasts for seasonal data is the Holt-Winters method with a damped trend and multiplicative seasonality:

$$\begin{aligned}\hat{y}_{t+h|t} &= \left[\ell_t + (\phi + \phi^2 + \cdots + \phi^h)b_t\right] s_{t+h-m(k+1)} \\ \ell_t &= \alpha(y_t/s_{t-m}) + (1-\alpha)(\ell_{t-1} + \phi b_{t-1}) \\ b_t &= \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)\phi b_{t-1} \\ s_t &= \gamma\frac{y_t}{(\ell_{t-1} + \phi b_{t-1})} + (1-\gamma)s_{t-m}.\end{aligned} \tag{3}$$

Using the model function in fable we can fit all three versions of Exponential Smoothing on our train data simultaneously. We differentiate between the method by passing different arguments within the corresponding functions. For example for the multiplicative method we can define error("M") + trend("A") + season("M").

Three sets of forecasts are plotted in figure 16 which correspond to each of the methods used to fit our train data. We can see that the algorithm has correctly picked up on both the trend and seasonal components of our time series, however the differences appear to be small between them.

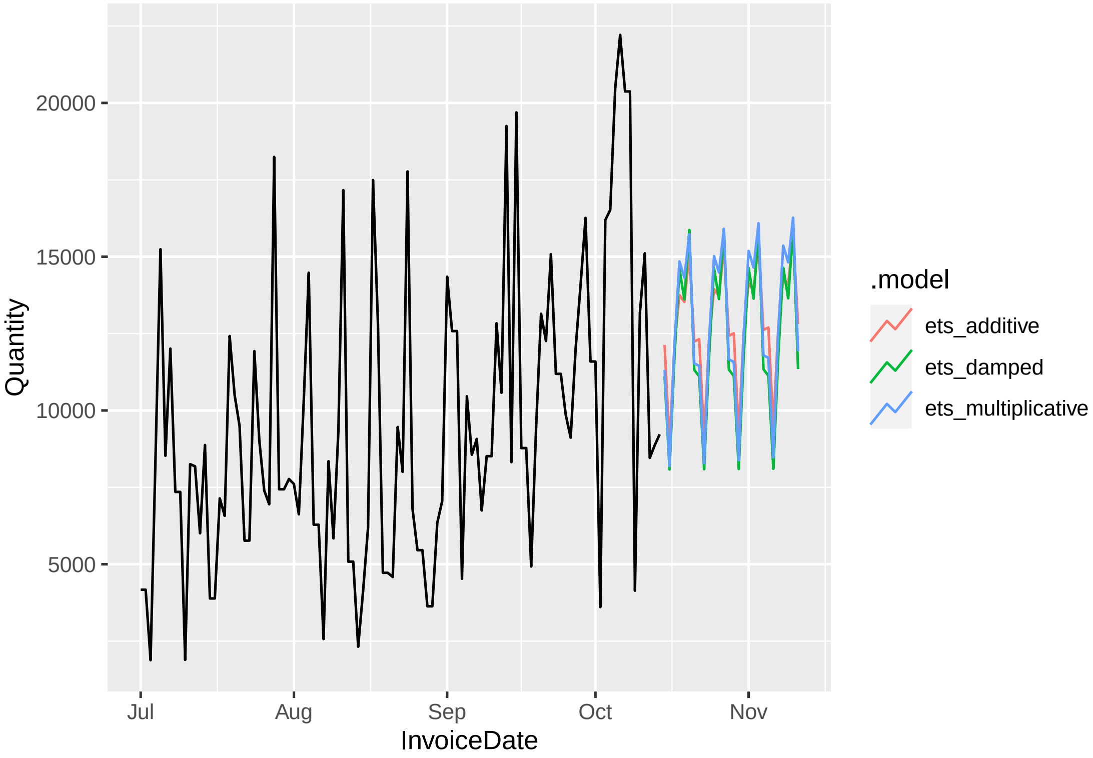

Figure 17: Trend and Seasonality Adjusted Forecasts

Perhaps the biggest observation is on how well the algorithm has performed when all 3 trained models get refitted as one-step forecasts and measured on the test data set. Their performance is almost identical with a slight edge on the multiplicative version of Exponential Smoothing. This is also ~10% improvement over the forecast error of Simple Exponential Smoothing and ~8% improvement over the SNAIVE algorithm.

5.3 Validation

At this point we feel that we have achieved some good enough scores and we have proceeded into fitting all above tuned methods against the entire online_store_fit data set. Note that we have yet to use the online_store_validate dataset in our testing , which we'll do now using multi step forecasts and compare all methods applied thus far.

Again the fable package allows us to fit all of the algorithms simultaneously and evaluate the best method.

Looking at the MAPE results we can observe that the damped trend version of the algorithm had surpassed all other methods with a MAPE score of 25.1% which is 11% improvement over the SNAIVE method.

Because it is difficult to visualise all five forecasts produced by each method, we proceed with plotting only the 28 days ahead multi step process from the ETS Damped one which produced the best score (see figure 17). We can observe that that our trained algorithm is performing fairly well in keeping up wit the weekly trend and seasonality pattern fo the previously unused validate dataset. Something that also catches our eye is how large the confidence intervals are. Perhaps our model doesn't generalise enough and there's still variability in the data that cannot be explained by our Exponential Smoothing model.

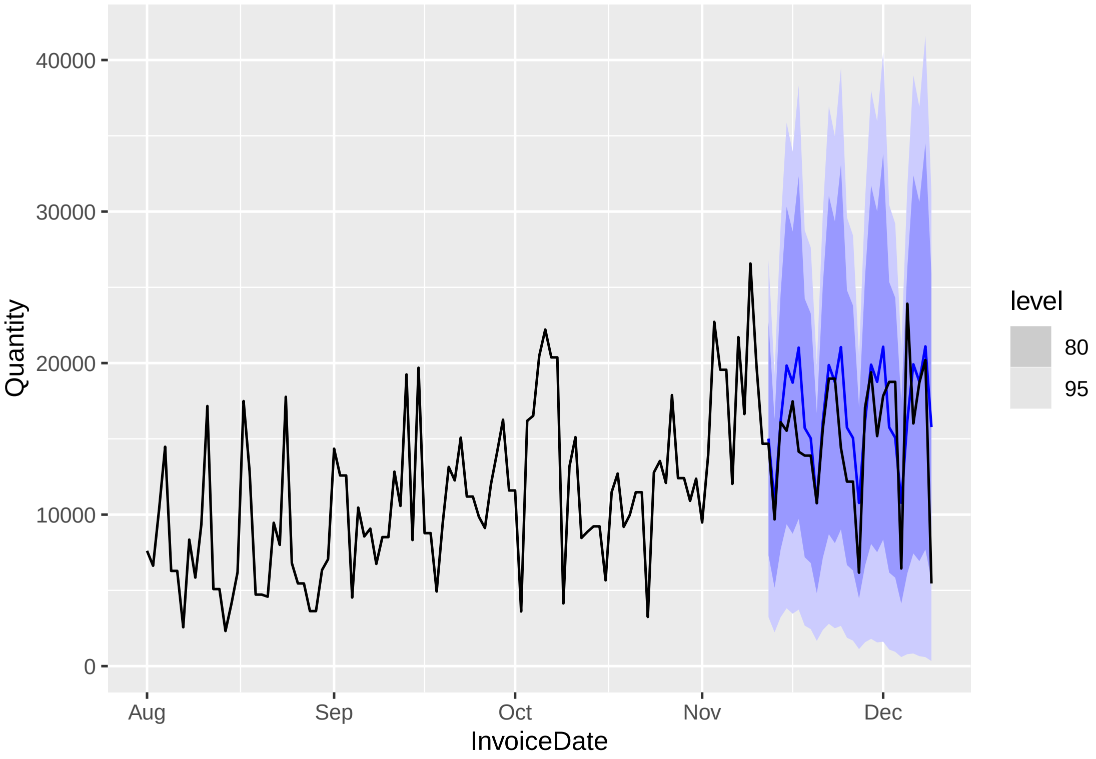

Figure 18: Multistep Exponential Smoothing Forecasts with Damped Trend and Multiplicative Error and Seasonality

6 Bootstrapping & Forecast Bagging

6.1 Bootstrapping

A useful technique in data science which allow us to avoid over-fitting our model is bootstrapping. In practice, we bootstrap the residuals of a time series in order to simulate future values of a series using a model. The process is the following:

First, the time series is transformed if necessary, and then decomposed into trend, seasonal and remainder components using STL.

Then we obtain shuffled versions of the remainder component to get bootstrapped remainder series. Because there may be autocorrelation present in an STL remainder series, we cannot simply use the re-draw procedure. Instead, we use a "blocked bootstrap," where contiguous sections of the time series are selected at random and joined together.

These bootstrapped remainder series are added to the trend and seasonal components, and the transformation is reversed to give variations on the original time series.

We implement the bootstrapping process bellow:

We generated 100 simulations of our time-series (online_store_fit) using the generate() fucntion from the fable model. We have used a block size of 28 to capture 4 weeks of data which is also the forecasting horizon we are aiming for. Figure 18 shows shows a visualisation fo these simulated series over the original data.

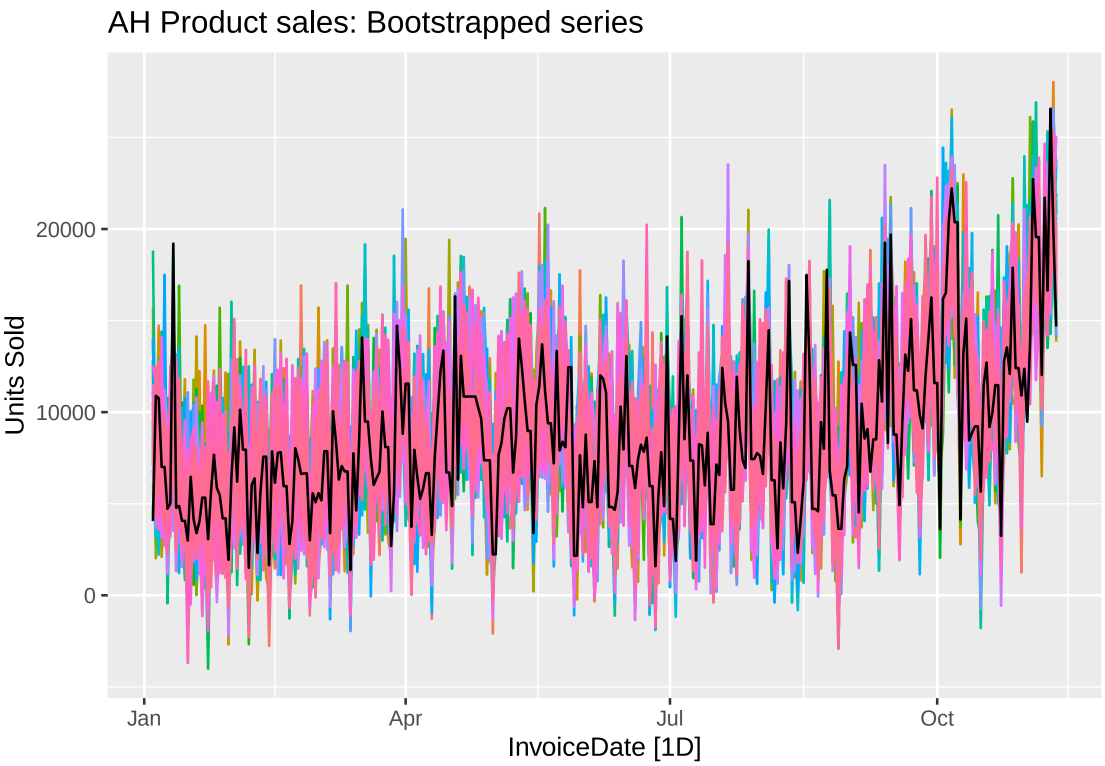

Figure 19: 100 Bootstrapped Time seres

For each of these series, we fit an ETS model and generate multiple 28 days multi step forecasts . This is a time-consuming process as there are a large number of series. Figure 19 shows these forecasts generated. It took a couple of mutes to complete on a normal windows laptop with 8GB of RAM.

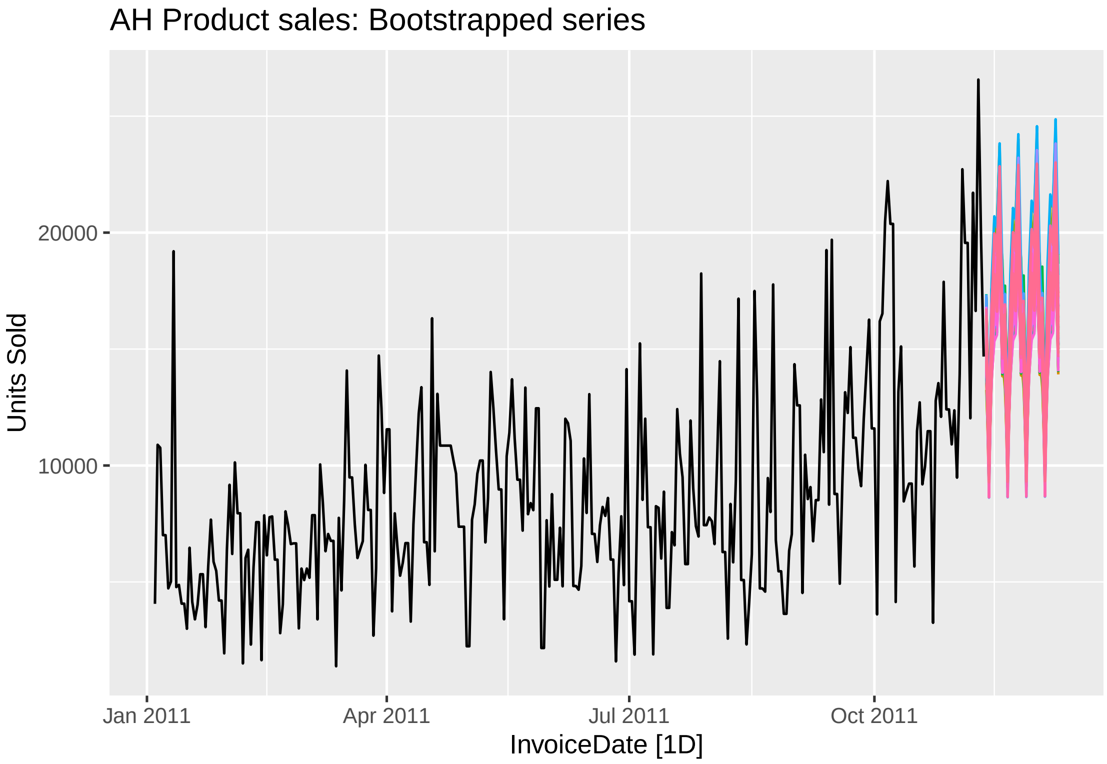

Figure 20: Simulated Forecasts from Bootsrapped Time Series

6.2 Forecast Bagging

One use for these bootstrapped time series is to improve forecast accuracy. If we produce forecasts from each of the additional time series, and average the resulting forecasts, we get better forecasts than if we simply forecast the original time series directly. This is called “bagging” which stands for “bootstrap aggregating.”

We’ve managed to improve our previous MAPE slightly which of course is a welcomed achievement but more importantly it provides us with the reassurance that we have not overfitted our model and would probably perform well on forecasting 28 days of sales for this product group on data that the algorithm has not yet seen. Figure 20 visually compares the bagged forecast (orange), with directly applied ETS damped (before bootstrapping) and with the actual values fo the last 28 days of the dataset.

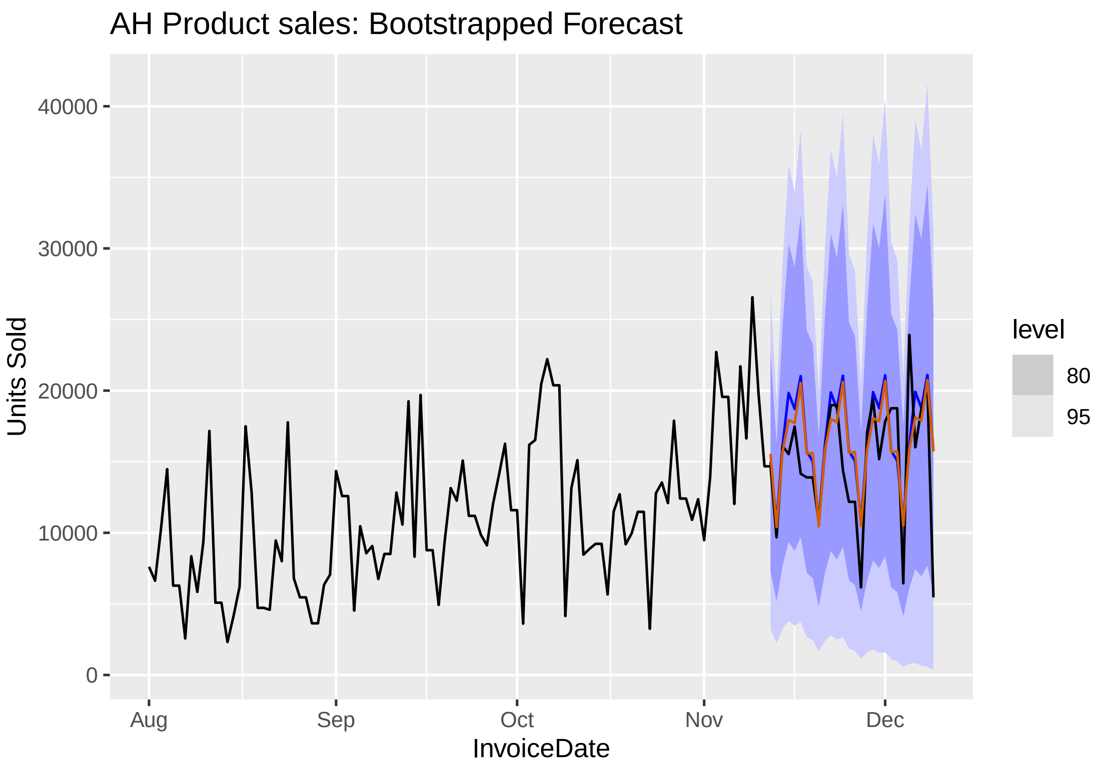

Figure 21: Bagged ETS Forecast vs Non Bootstrapped ETS vs Actual Sales Values

7 Conclusion and further discussion

In this analysis, we have assumed the role of a data scientist, assigned with the task of developing a time series forecasting model of $h = 28$ time horizon. The dataset analysed was an extract from a UK online store's sales database which contained a year's worth of sales recorded for over three thousand items on a daily granularity level. Due to the large volume of independent time-series in the dataset, we've focused our efforts to products of high revenue and large volume of orders which have the majority of impact on both the company's top and bottom lines. After a thorough clean up of the dataset we've proceed with feature extraction from the time-series of trend and seasonal nature. Those features where later used to fine tune the algorithms used to during the fitting process of our model on the train dataset.

The algorithms tested where a) The naive method (SNAIVE) with a trend and seasonal component applied, b) Linear Regression (TSLM) with trend and seasonality, and c) the Exponential Smoothing algorithm in its multiple versions for trend and seasonality adjustments. Exponential Smoothing and specifically the Damped Trend with multiplicative error and seasonality had outperformed all other methods by producing a smaller MAPE when tested and validated against hold-out part of our dataset reserved for that final estimation. Moreover, we've applied time series bootstrapping with 100 simulations of block size = 28 to ensure are performance score was not the result of over fitting our model. Our results where in line with the theory suggesting that on average, bagging gives better forecasts than just applying ETS() directly (Bergmeir, Hyndman, and Benítez 2016). However it comes with a performance cost as it more complex data structures it could take a significant amount of time to calculate.

Time series forecasting is a vast and complex topic in data science. With that in mind there's further analysis that coud potentially improve prediction on this dataset and which we haven't included in this instance. More specifically, our task was to forecast on a $h = 28$ horizon, however a time-series cross validation would have allowed us to estimate what it the optimal horizon to forecast to minimise error. We've have used 3 of the most popular forecasting algorithms to generate our predictions, however it would be useful in future analysis to further test popular algorithms such as ARIMA, Prophet and TBATS. Additionally, our dataset was a multivariate one which we coerced into a univariate time series. More advanced machine learning techniques such as Non-Liner Regression and clustering could have provided further insights on the interaction between variables in the dataset as well as providing more sophisticated classification of product and potentially customer groups.

References

Bergmeir, Christoph, Rob J. Hyndman, and José M. Benítez. 2016. "Bagging Exponential Smoothing Methods Using STL Decomposition and Box–Cox Transformation." *International Journal of Forecasting* 32 (2): 303–12. https://doi.org/https://doi.org/10.1016/j.ijforecast.2015.07.002.

Brown, R. G. 1959. *Statistical Forecasting for Inventory Control*. McGraw/Hill.

Cleveland, Cleveland, R. B. 1990. "STL: A Seasonal-Trend Decomposition Procedure Based on Loess." *Journal of Official Statistics* 6 (1): 3–33.

Holt, Charles C. 2004. "Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages." *International Journal of Forecasting* 20 (1): 5–10. https://doi.org/https://doi.org/10.1016/j.ijforecast.2003.09.015.

Hyndman, & Athanasopoulos, R. J. 2021. *Forecasting: Principles and Practice, 3rd Edition*. OTexts: Melbourne, Australia. https://otexts.com/fpp3/.

Sheather, S. J. 2009. *A Modern Approach to Regression with r*. Springer.

Unwin, A. 2015. *Graphical Data Analysis with r. Chapman*. Chapman; Hall/CRC.

Wang, Smith, X. 2006. "Data Mining and Knowledge Discovery." *Journal of Official Statistics* 13 (3): 335–64.

Winters, Peter R. 1960. "Forecasting Sales by Exponentially Weighted Moving Averages." *Management Science* 6 (3): 324–42. https://EconPapers.repec.org/RePEc:inm:ormnsc:v:6:y:1960:i:3:p:324-342.