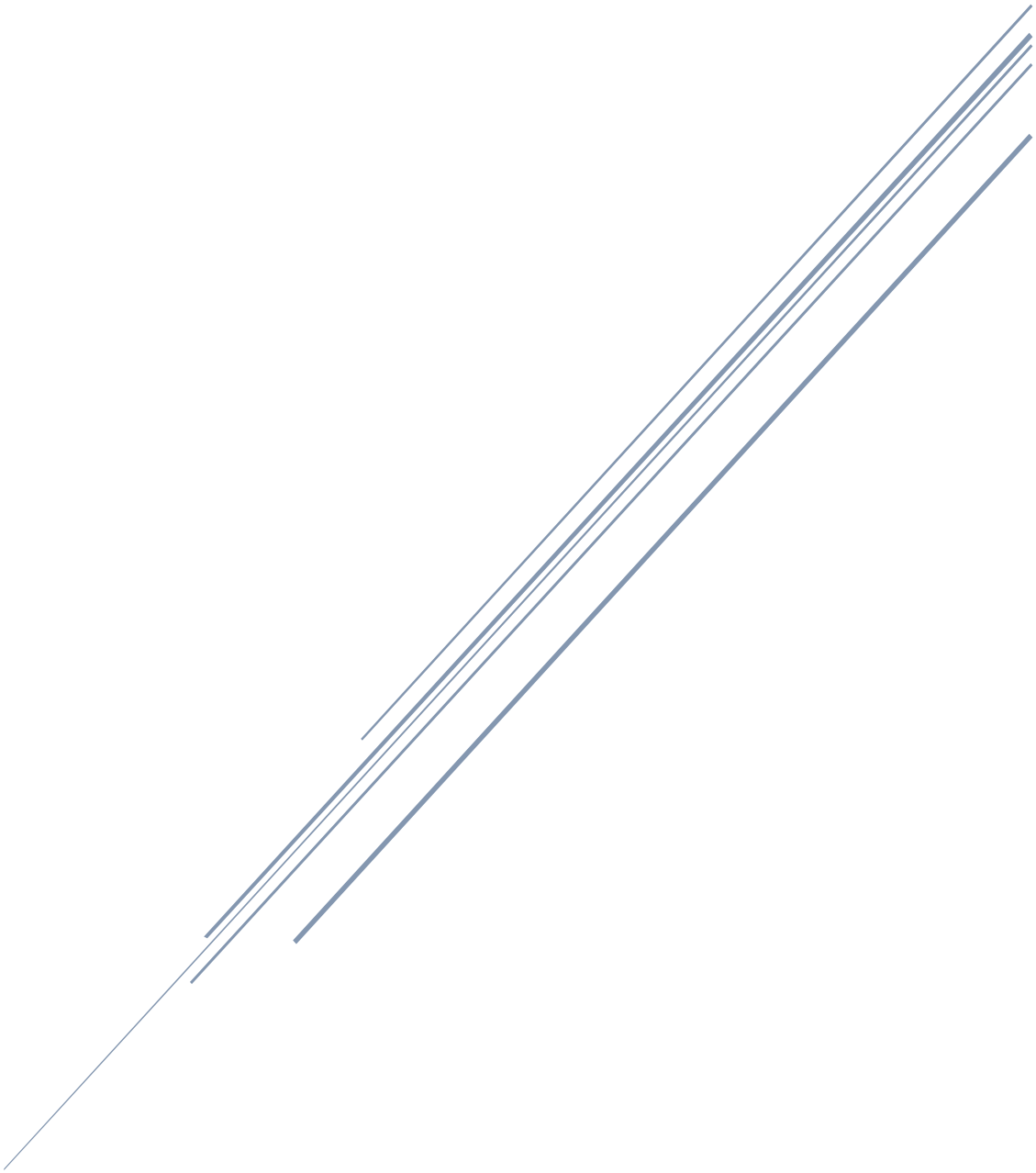
MSCI 523

Forecasting Coursework – Part 2

Rivyesch Ranjan, NN5-034, 36330520



Executive Summary

Time series forecasting was performed on the data NN5-034. The time series was split into in sample and out of sample sets using a ration of 90%:10%. Four types of time series forecasting algorithms were used to produce a 14-day future forecast. The algorithms used were exponential smoothing, ARIMA, regression and neural networks. Models were developed manually and automatically for each of these methods and then compared. For each forecasting method the best model created was highlighted. The best model for ARIMA, regression and neural network came from the manual building process showing that manually built models often outperform automatically built models. Only for exponential smoothing did the automatic model outperform the manually built model. The performance of models was evaluated on the basis of the error metrics MAE and sMAPE which are both relatively unbiased compared to other error metrics. Cross validation RMSE was calculated to verify that the developed models were not overfitting and would exhibit similar performance for different forecast origins. The best exponential smoothing model used a multiplicative error and additive seasonality state space configuration with a small smoothing factor for level (0.0663) and seasonalty (0.0001). Meanwhile, the best performing ARIMA model was $(1,1,1)(0,1,1)_7$ which made use of non-seasonal AR and MA components and a seasonal MA component. Regression model that employed information about trend, day in a week seasonality, quarterly seasonality, bank holidays, missing values, outliers and the first two lags gave extremely low errors. The NNAR(28,2,k) was identified to be the neural network that is optimal, however it was later found to have suffered from overfitting. Each of these best models were compared to the benchmark mean, naïve and seasonal naïve models. All except for the neural network outperformed the benchmark model making exponential smoothing, regression and ARIMA suitable models for the forecasting of this time series. The model that overall was the best in terms of having the lowest errors was the $ARIMA(1,1,1)(0,1,1)_7$ model. This was the best recommended model. The regression model was also put forward as a recommended model due to its ability in incorporating external factors into the model and versatility in dealing with time series that have different characteristics such as structural breaks.

Table of Contents

Introduction

Time series data "NN5-034" related to the daily ATM cash withdrawals in England over the course of two years is analysed and studied. Forecasting models that are able to accurately forecast the cash withdrawals 14 days into the future would provide useful information to the relevant banks such that they can deposit the necessary amount of money in the ATMs. This would improve customer satisfaction and also reduce the opportunity cost for the banks as excessive deposits could be served in other profitable investment and ventures. The objective of this study is to explore the forecasting capabilities of several advanced forecasting techniques.

Methodology

Train-Validation Split

The time series data is cleaned and pre-processed similarly to what was outlined in the previous report with the only exception being that zero values in the time series were removed and then imputed by linear interpolation. This was done here so that the number of potential models that could be built is not restricted to just additive models.

The cleaned data is converted into a time series object with a frequency of 7 which represents a day in a week seasonality using the ts() function. For purposes of model building, the time series is split using the ts_split() function such that there is in-sample data and out-sample data. The in-sample data is used to fit the model and comprises of just under 90% of the time series. Meanwhile, the remaining approximately 10% of the time series which corresponds to the last 74 observations of the time series is withheld from the training process. This out-sample data is used to validate the forecasting model built. Considering the desired forecast horizon is 14 steps into the future this length of validation which is much larger than the forecast horizon is justified. It should be highlighted that there is no test set that exists for this study.

Evaluation/Error Metrics

The ability of a time series forecasting model to predict the future is defined by its performance. The three main metrics used in this study to determine the performance of a model is mean absolute error (MAE), root mean square error (RMSE) and symmetric mean absolute error (sMAPE).

MAE shows how much inaccuracy should be expected from the forecast on average but is ineffective for data with extreme error. RMSE is also based on the same ideology as MAE but is especially suitable when the residuals found are normally distributed. It is especially sensitive to outliers in the prediction which makes it useful as the occurrence of large outliers in predictions are undesirable. sMAPE is another evaluation measure that is expressed as a percentage. This makes it useful for comparing forecast that are on different scales. sMAPE overcomes the asymmetry of MAPE which penalises models that over-forecast more heavily. It does, however, introduce another kind of asymmetry. The sMAPE will be higher for underpredictions as compared to overpredictions. Another drawback is that it is unstable when the true value and forecasted value are close to zero. Overall, the lower each of these metrics are, the higher the accuracy of the model's forecast.

Cross validation is used to check and identify if the model is overfitting. Furthermore, it provides information on the reliability of the model's forecasting performance for different forecast horizon.

Benchmark models

To demonstrate the usefulness of more advanced forecasting techniques three simple benchmark models are created. These benchmark models are the Mean method, Naïve method and Seasonal Naïve method. Of the three benchmark models, the seasonal naïve model seems the most capable as it has the lowest error metrics for both in and out of sample data. The more advanced forecasting

models can only be considered to be more useful if they are capable of forecasting more accurately than the benchmark models.

Table 1: Performance Metrics of Benchmark Models

Model	Residuals RMSE	Cross-validation RMSE	In/Out sample	RMSE	MAE	sMAPE
Mean model	11.5868	11.6557	Train	11.5868	9.3285	-
			Validation	11.1874	8.9174	0.2731
Naïve model	12.2308	15.9858	Train	12.2308	9.3088	-
			Validation	10.6042	8.9174	0.2736
Seasonal naïve model	11.1377	11.2593	Train	11.1377	6.8579	-
			Validation	9.9022	7.8264	0.2433

Exponential Smoothing Model

Four important exponential smoothing methods are explored below. The main hyperparameter for each method is varied to find the optimal. This corresponds to the model that gives the lowest MAE and RMSE. The other hyperparameters, if any, are not specified when building the model. Hence, it is automatically selected during the training process of the model. The range of values the main hyperparameter is varied for each of the four main methods are detailed in Table 14 in the Appendix A. The choice of the range is based on the traditional restriction for the state space models.

Simple exponential smoothing is suitable for forecasting data with no clear trend or seasonal pattern. Therefore, the time series is differenced to remove seasonality first and then differenced to remove the trend. This data is then fit into the SES using the ses() function with a horizon length equal to the length of the validation set. The optimal alpha is 0.004 which gives a RMSE of 15.36598. The small value of alpha reflects the small changes over time in the estimated level which results in a series of fitted values that are smooth.

Holt's linear trend method allows for forecasting of data with a constant increasing or decreasing trend. The time series is differenced to remove seasonality and then fit into the model using the holt() function. The optimal beta is 5e-04 which gives a RMSE of 9.452893. This method tends to over forecast especially for longer forecast horizons since trends rarely stay constant. For this reason, the inclusion of a damping factor is explored. The range of the damping coefficient is restricted to 0.8 to 0.98. This is due to it being impractical to use a low damping coefficient that is too strong whilst a damping coefficient of 1 would be identical to the model created in the previous section. The optimal phi value is 0.964 for which the RMSE of the fitted model is 9.416307.

Holt-Winters' seasonal method it is meant for data with seasonality, no differencing was done to the time series. The original time series was fit using the ets() function for gamma values ranging from 0 to 0.996. The optimal gamma was determined to be 0.007 which gives a RMSE of 5.63274.

The optimal hyperparameters from the experimentation done above can be viewed in Figures 14 to 17 in Appendix A and are summarized in the Table 2 below.

Table 2: Optimal ETS Hyperparameters

Parameter	Optimal value
Alpha	0.004
Beta	5.00E-04
Phi	0.964
gamma	0.006

These optimal values are used to build a ets model using the ets() function. There are several configuration of the state space model that can be trained where the type of error, trend and seasonality can either be additive, multiplicative or non-existent. Various possible model combinations are explored and the findings are tabulated below in Table 3.

Table 3: Performance Metrics of Candidate ETS Models

Model	AICc	P value	Residuals RMSE	CV RMSE	In/Out sample	RMSE	MAE	sMAPE
MAM	7013.43	2.2E-16	0.6775	0.2790	Train	8.9732	5.5651	-
					Validation	5.1547	4.4572	0.6775
MMM	7008.98	2.2E-16	0.2753	9.8067	Train	9.0499	5.5515	-
					Validation	9.0499	5.5515	0.4033
MAA	7001.94	2.2E-16	0.2756	9.2566	Train	8.6212	5.4569	-
					Validation	7.7141	6.7057	0.9881
AAA	7152.15	2.2E-16	8.5797	9.2410	Train	8.5797	5.3837	-
					Validation	7.6845	6.7001	0.9920

"AAA" is a candidate model since it has reasonably low error metrics and the difference between the cross validation RMSE and residual RMSE is not very significant unlike the models "MMM" and "MAA". The low discrepancy between cross validation and residual RMSE indicates the model will have similar accuracy for different forecast horizon and is therefore reliable. The residuals are still moderately high but normally distributed based on Figure 1. There is still a large amount of autocorrelation that remains in the residuals as there are significant spikes at the first four lags. The p-value from the Ljung-Box test gives a very low value which confirms that the residuals are indeed autocorrelated.

"MAM" is the best ETS model since it has the lowest MAE and RMSE for the validation set. Furthermore, the RMSE obtained through cross validation is extremely low indicating this model performs better than the rest consistently. The AICc of this model is reasonably low, albeit not the lowest of the models trained. The residuals are much lower than that of the "AAA" model and is normally distributed as seen in Figure 2. However, this model still has autocorrelation in the residuals with there being significant spikes at the first three lags. The statistical test gives a low p-value which confirms this.

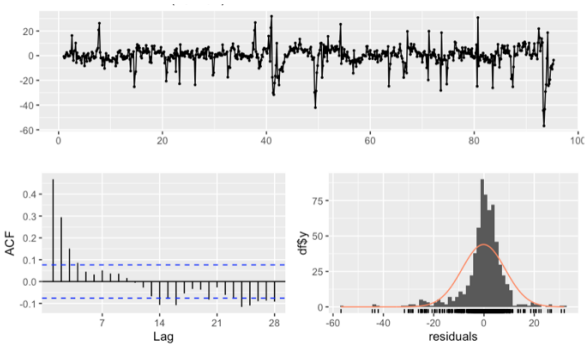


Figure 1: Residuals of AAA ETS Model

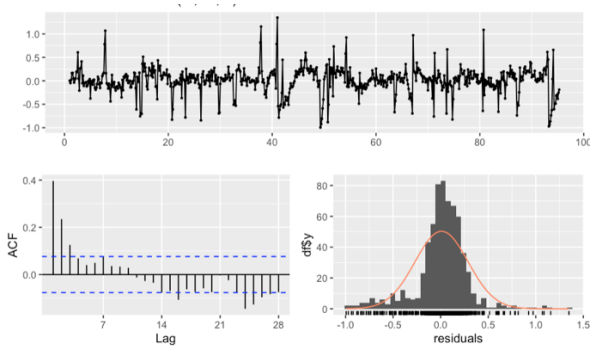


Figure 2: Residuals of MAM ETS Model

An automatic ETS is built to compare the performance of the manually built ETS. Whilst the manually built ETS involves finding the optimal hyperparameters one at a time, the automatic model does this automatically and decides on the best combination of hyperparameters based on the combination that gives the lowest AIC value. The automatically built ETS model has an alpha of 0.0663, gamma of 1E-04 and a configuration of "MNA". The results show that the MAE and RMSE is lower for the train and validation set for the automatically built model. The

AICc of the automatic model is also lower indicating that it has a better fit compared to the manually built model. However, the automatic model like the manual models still have autocorrelation in the residuals as depicted in Figure 18 in Appendix A. This suggest that ETS model's forecast, regardless of if built manually or automatically, will be inefficient as there is information left over in the residuals which should have been accounted for in the model.

Table 4: Performance Metrics of Automatic ETS Model

Model	AICc	P value	Residuals RMSE	CV RMSE	In/Out sample	RMSE	MAE	sMAPE
Auto	6943.10	1.10E-08	0.2640	10.8367	Train	7.9184	4.7888	-
					Validation	4.4731	3.6603	0.1920

ARIMA

The function tsdisplay() is used on the seasonal and trend differenced data. There is a significant spike at lag 1 in the ACF which suggest a non-seasonal MA(1) component. There is also a significant spike at lag 7 in both the ACF and PACF plots which points to a seasonal MA(1) component, although this may be higher considering there are several significant spikes at every 7 intervals in the PACF plot. A significant spike at lag 1 in the PACF suggest that there is an AR(1) non-seasonal component.

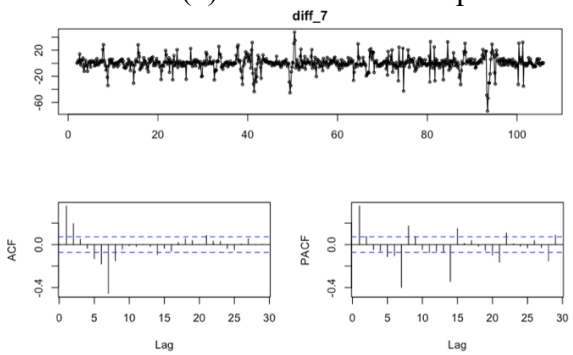


Figure 3: Seasonal Differenced Time Series

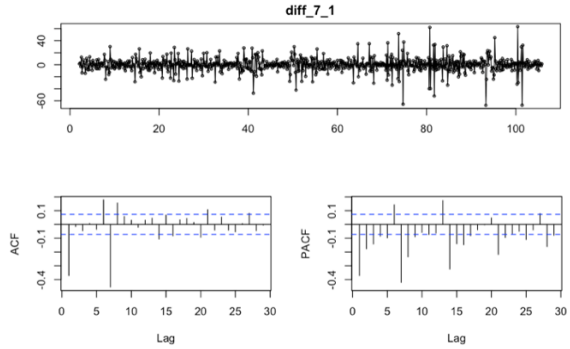


Figure 4: Seasonal and Trend Differenced Time Series

However, the initial ARIMA model that will be created will be a simple ARIMA(0,1,1)(0,1,1)$_7$ model. Ggtsdisplay() function is used to visualize the residuals of the fitted model and its corresponding ACF and PACF. It can be seen there are still significant spike at lag 1 in both ACF and PACF in Figure 5. There also appears to be significant lags at lags 3 to 6 for both ACF and PACF. Hence, it is sensible to include an AR(1) component such that an ARIMA(1,1,1)(0,1,1)$_7$ is fit. Figure 6 shows the residual of the newly fit model. It is evident there are no significant spikes in the ACF and PACF, although there are a few time lags for which the spikes are close to the significance level. Overall, the latest ARIMA model captures the trend and seasonality in the time series accurately enough to give reliable forecast.

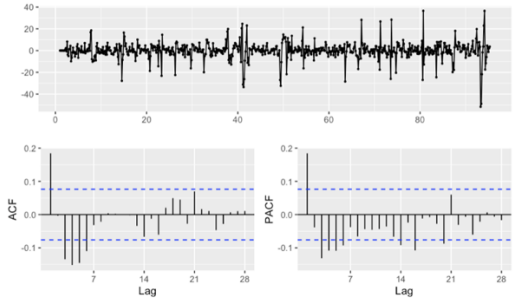


Figure 5: ARIMA(0,1,1)(0,1,1)$_7$ Residuals

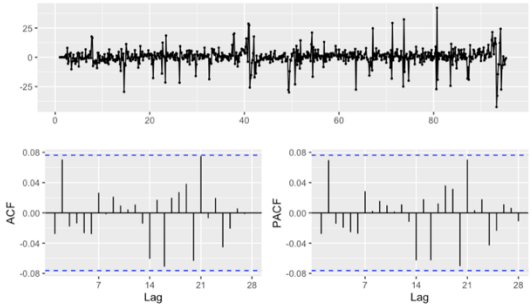


Figure 6: ARIMA(1,1,1)(0,1,1)$_7$ Residual

Whilst a suitable ARIMA model has already been found, a few additional ARIMA models with different parameters are trained to check and verify that there are no other configurations that perform better. The findings are listed in Table 15 in Appendix B . Table 5 shows the simplest ARIMA along with three candidate ARIMA models that perform well. The two new candidate models are found by changing the components of AR, MA and order of differencing incrementally as demonstrated above. ARIMA(1,1,1)$(0,1,2)_7$ has identical error metrics for the train and validation sets as compared to the candidate model ARIMA(1,1,1)$(0,1,1)_7$ found above. However, ARIMA(1,1,1)$(0,1,1)_7$ is still regarded as the optimal model since it has a slightly better AICc and more importantly a lower cross validation RMSE. The results for the various other configurations of ARIMA trained are tabulated in the appendix.

Table 5: Performance Metrics of Candidate ARIMA Models

ARIMA models:	AICc	Residuals RMSE	CV RMSE	In/Out sample	RMSE	MAE
ARIMA (0,1,1) $(0,1,1)_7$	4587.39	7.8725	9.4450	Train	7.8295	4.6691
				Validation	6.4620	5.9527
ARIMA (1,1,1) $(0,1,1)_7$	4508.05	7.4795	8.5418	Train	7.3433	4.4175
				Validation	3.4743	2.7294
ARIMA (1,1,1) $(1,1,1)_7$	4509.07	7.4375	8.5638	Train	7.3082	4.3839
				Validation	3.4814	2.7424
ARIMA (1,1,1) $(0,1,2)_7$	4508.95	7.4356	8.5837	Train	7.3077	4.3818
				Validation	3.4668	2.7296

ARIMA model is also built automatically using auto.arima() function. The automatic method selects the best ARIMA based on the lowest AICc possible. The AICc of 4507.47 seen in the table below is lower than all the candidate models AICc considered. Although the fit of the ARIMA model is the best of all models considered, the MAE and RMSE of the manually built and selected ARIMA model is lower, and hence better than that of the automatic ARIMA model. This is especially true for the error metric applied on the validation set where it can be seen the MAE of the automatic model is more than eight times greater than the MAE of any of the manual candidate models. Considering the MAE and RMSE for the train set is only slightly higher for the automatic model, it is inferred that the automatic ARIMA overfits which results in poor generalization ability and forecasting performance.

Table 6: Performance Metric of Automatic ARIMA

ARIMA models:	AICc	Residuals RMSE	CV RMSE	In/Out sample	RMSE	MAE
ARIMA $(0,0,3)(2,1,1)_7$	4507.47	-	-	Train	7.3841	4.4465
				Validation	24.5033	24.5033

Regression

An important aspect of building the regression models in this study was the inclusion of other information other than the past observations of the time series. The motivation behind creating this additional information in the form of several dummy variables is that the inclusion of some or all of these dummy variables may lead to more accurate forecast. Columns that indicate the day of the week, week number and month number of each instance are created based on the date information of each instance. From there dummy variables can be created for weekly,

monthly and quarterly seasonality. Those instances in the time series that were identified as outliers in the previous study are represented accordingly using another dummy variable. The same is done for those instances that fall on dates that are bank holidays in the UK and those instances with missing values in the time series prior to data cleaning. It should be noted that a dummy "step" variable for structural breaks was not created since the previous study did not find any. Distributed lags for the first 30 lags of the time series are created where the data for each respective lag is contained within a separate column. The last information that needs to be included is the trend of the time series. There are several options that can be used, and this was explored and detailed in the table below. The trend that fits the time series the best and gives the lowest MAE is the spline trend with a Box-Cox transform. Hence, this trend is included as part of the additional information.

Table 7: Performance Metrics of Different Trends

Trend	RMSE	MAE	MAPE
Linear	14.51935	10.9493	32.44022
Exponential	35.45832	32.54654	89.03387
Quadratic	14.10543	11.10265	35.26562
Cubic	14.34198	10.94511	33.24183
Spline	10.4882	8.920421	29.93119
Spline (w Box-Cox Transform)	10.52698	8.920265	29.11847

From the previous study, it was identified that there is a day in a week seasonality and a trend in the time series data. Hence, the base regression model (Rbase) is fit using the lm() function with day in a week seasonality dummy variables and spline with Box-Cox transform trend as the independent variables. Various models are trained where each model consists of the independent variables in the base model along with one or more of the dummy variables or distributed time lags. This is done to explore which variables are significant and improve the fit of the regression model. The decision if the newly included variable is significant or not is based on the p-value of that variable. A predictor that has a low p-value is likely to be a meaningful addition to the model because changes in the predictor's value are related to changes in the response variable. The independent variables used in each model explored and the variable's significance are shown in the Table 16 in Appendix C.

Both weekly seasonality and trend that were used in the base model are significant predictors. Quarterly seasonality is a useful predictor whereas monthly seasonality was found to be not significant. The remaining dummy variables created for missing values, outliers and bank holidays were all found to be significant predictors, justifying the decision to create these dummy variables and include them in the model. The previous study found there to be the highest autocorrelation between time lags of 7 and hence it was included in the model. However, it was found to not be a significant predictor. All predictors that were found to be significant, namely weekly seasonality, trend, quarterly seasonality, NA, outliers and bank holidays, were used as independent variables in the first candidate model (R1). The residuals of the first candidate model along with the rolling average MAE is shown below. Whilst the residuals are normally distributed it is clear that there is still some autocorrelation in the residuals since lag 1 and lag 2 are above the significance level. The Ljung-Box test confirms that there is indeed autocorrelation in the residuals as the p-value is greater than the significance level 0.05. Also, the VIF of the independent variables are checked to ensure there is no multicollinearity.

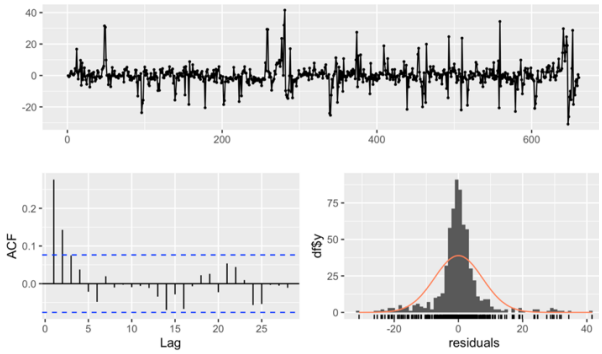


Figure 7: Residuals of Regression Model R1

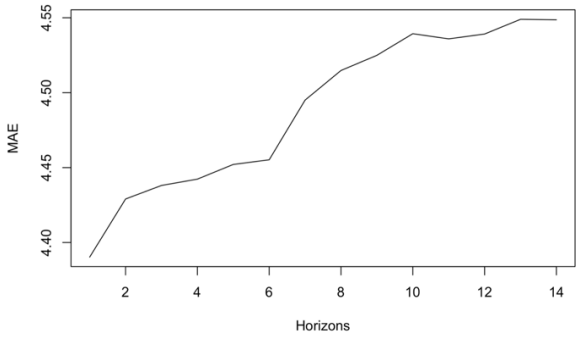


Figure 8: Rolling Horizon MAE for Model R1

A second candidate model (R2) is trained using all the independent variables in the first candidate model but with the inclusion of the distributed lags for lag1 and lag 2. The resulting model's residuals are shown below and it can be seen that there are no longer any autocorrelations in the residuals. This is verified by the p-value from the Ljung-Box test which is greater than 0.05. The residuals are also normally distributed. Furthermore, the rolling origin MAE for the second candidate model is consistently lower for all horizon lengths from zero to 14 compared to the first candidate model. This suggest that the latest model produces more accurate forecast.

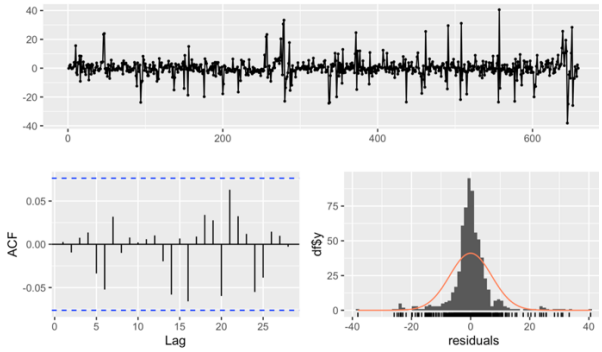


Figure 9: Residuals of Regression Model R2

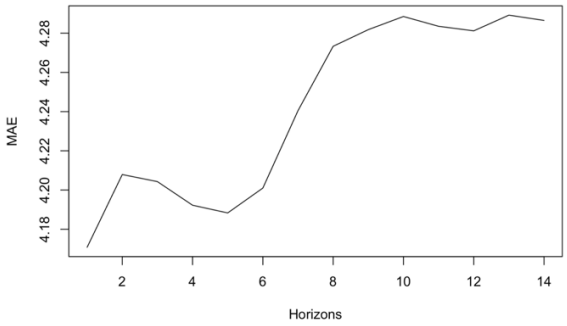


Figure 10: Rolling Horizon MAE for Model R2

The table below compares the two candidate models and the base regression model. The AICc and BIC is lowest for the model R2. This indicates that R2 is a better fit model compared to the rest. It's also noteworthy to compare the in and out of sample errors of the models. Again, model R2 performs the best, giving the lowest MAE errors for the train and validation sets. Based on the adjusted R-squared metric model R2 has the best goodness of fit of the three models and the inputs used in R2 manages to explain 62.53% of the variance in the time series.

Table 8: Performance Metrics of Candidate Regression Models

Model	AICc	Adjusted R-Squared	p-value	Rolling Error MAE	In/Out Sample	MAE	sMAPE
Rbase	4627.08	0.5283	2.20E-16	4.7048	Train	15.3163	-
					Validation	7.8951	0.2303
R1	4559.33	0.5866	3.94E-08	4.4896	Train	10.7200	-
					Validation	9.3217	0.2833
R2	4484.94	0.6253	0.8436	4.2421	Train	8.8255	-

					Validation	6.8098	0.1952

An automatic regression model is created using backwards stepwise regression. The independent variables selected were similar to that used in R2 except the automatic model also made use of lags 14, 16, 18, 20 and 21. Despite using more predictors, the automatic model still had a lower adjusted R-square and higher training and validation MAE compared to the best manually built regression model R2. It did, however, manage to get the lowest AICc. Nevertheless, the improved goodness of fit does not translate to lower errors in the forecast which could be attributed to the automatic model suffering from overfitting. The p-value from the Ljung-Box test was greater than 0.05 and so it can be concluded the residuals from the automatic model is not autocorrelated. The ACF plot in Figure 20 in Appendix C agrees with the conclusion of the statistical test.

Table 9: Performance Metric of Automatic Regression Model

Model	AICc	Adjusted R-Squared	p-value	Rolling Error MAE	In/Out Sample	MAE	sMAPE
Backward Selection	4309.20	0.6167	0.781	-	Train	8.9131	-
					Validation	6.9281	0.1978

Neural Network

Unlike the previous techniques considered, neural networks do not require the data being fitted to be stationary. Therefore, the original undifferenced time series is fed into a neural network auto regression (NNAR) model for training. This is done using the nnetar() function which only considers feed-forward networks with one hidden layer. There are three parameters which are number of non-seasonal lagged inputs (p), number of seasonal lagged inputs (P) and nodes in the hidden layer (k) that can be optimised. In the hyperparameter tuning only p and P are varied whilst k is calculated automatically based on the values of p and P.

The first neural network trained is a simple one with p=1 and P=1. The residuals in Figure 21 in Appendix D are relatively high and the ACF shows significant spikes at both random lags and periodic intervals. This suggest that using P=1 is not sufficient to represent the seasonality in the time series. This is further evidenced by the Ljung-Box test which gives a small p-value less than 0.05. The neural network was retrained with P=4 to check if this captures the seasonality. The p-value given by the statistical test is greater than 0.05 which implies that the residuals of the time series are independent. However, there remains few significant spikes at lag 2 and some larger lags as seen in Figure 22 in Appendix D. This infers that more non-seasonal lagged inputs may be required as inputs to the model and that making judgments solely based on the Ljung-Box test p-value can be misleading.

The previous study found that lag intervals of 7 had the highest autocorrelation. Hence, NNAR models with p set to 7, 14, 21 and 28 are trained. It was found that of these models those that used 14, 21 and 28 non-seasonal lags as inputs resulted in low residuals.

The models shown in the table are regarded as the best candidate models. The residuals for all the candidate models were approximately normally distributed. Besides that, the p-values for each of these candidate models are greater than the significance level which implies that the residuals are not autocorrelated. Models NNAR(14,2,k) and NNAR(14,3,k) both have relatively low residuals and not much autocorrelation in the residuals. NNAR(28,1,k) is similar

except its residuals are much lower. NNAR(21,1,k), NNAR(21,2,k) and NNAR(28,2,k) were considered better models since there were no signs of autocorrelation in the residuals. This implies that these model's forecast will be efficient as there is no information left over which should have been accounted for in the model. These three models also had very low residuals. The model that was found to be optimal is NNAR(28,2,k). The reasoning behind this is that there is no autocorrelation in the residuals and the distribution of the residuals are approximately normally distributed. This model configuration has the lowest RMSE and MAE for the train set and validation set of all the candidate models.

Table 10: Performance Metrics of Candidate Neural Network Models

Model	p-value	Residuals RMSE	Cross-validation RMSE	In/Out sample	RMSE	MAE	sMAPE
NNAR(14,2,k)	0.2022	4.8651	10.2108	Train	4.6865	2.9583	-
				Validation	13.4808	11.3744	0.5122
NNAR(14,3,k)	0.2164	4.8469	10.2281	Train	4.4013	2.7230	-
				Validation	13.3539	10.3813	0.5006
NNAR(21,1,k)	0.9410	3.3984	10.1721	Train	2.8457	1.6686	-
				Validation	13.4815	10.1465	0.4661
NNAR(21,2,k)	0.4251	2.0797	10.1504	Train	2.9429	1.7823	-
				Validation	15.8665	12.9844	0.5741
NNAR(28,1,k)	0.4299	2.1394	10.1753	Train	1.6182	0.9351	-
				Validation	9.3925	8.4718	0.4527
NNAR(28,2,k)	0.2307	1.9684	10.2258	Train	1.6028	0.8954	-
				Validation	9.2119	7.1979	0.4516

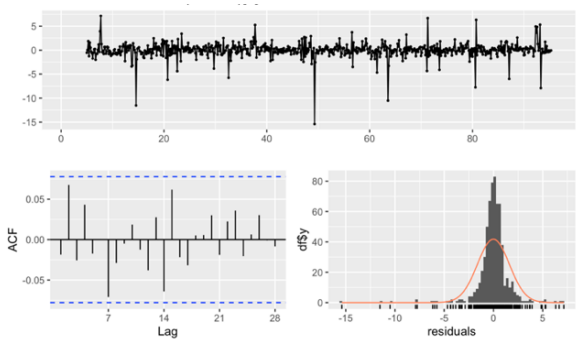


Figure 11: Residuals of NNAR(28,2,k)

An automatic NNAR model was also created. The automatic model found NNAR(28,1,14)$_7$ to be the optimal model. This is rather similar to the parameters found through the manual model building process. Figure 23 in Appendix D shows the residuals were relatively low and this model had no autocorrelation in the residuals. It also passed the Ljung-box test as it concluded the residuals are not autocorrelated. The MAE and RMSE for the automatically built model is slightly higher than the optimal model selected prior for both the in and out of sample data sets. This highlights that whilst the automatic process is fast and does train a highly capable neural network, manually tweaking the parameters can lead to an even more optimal model that produces more accurate time series forecast.

Table 11: Performance Metrics of Automatic Neural Network

Model	p-value	Residuals RMSE	Cross-validation RMSE	In/Out sample	RMSE	MAE	sMAPE
NNAR(28,1,k)	0.6141	4.8525	10.2241	Train	1.6145	0.9160	-
				Validation	10.1757	8.8312	0.5176

Model Comparison

The table below summarises the error metrics MAE and sMAPE for the best models selected for both in and out of sample data sets. Furthermore, the RMSE computed through cross validation is also displayed.

Table 12: Summary of Performance Metrics of Best Models for Each Algorithm

Model	Residuals RMSE	CV RMSE	In/Out sample	MAE	sMAPE
Auto ETS	0.2640	10.8367	Train	4.7888	-
			Validation	3.6603	0.1920
ARIMA(1,1,1) $(0,1,1)_7$	7.4795	8.5418	Train	4.4175	-
			Validation	2.7294	0.1330
NNAR(28,2,k)	1.9684	10.2258	Train	0.8954	-
			Validation	7.1979	0.4516
R2	7.0921	4.2421	Train	8.825514	-
			Validation	6.809765	0.1952
Mean model	11.5868	11.6557	Train	9.3285	-
			Validation	8.9174	0.2731
Naïve model	12.2308	15.9858	Train	9.3088	-
			Validation	8.9174	0.2736
Seasonal naïve model	11.1377	11.2593	Train	6.8579	-
			Validation	7.8264	0.2433

The results from the table clearly indicate that the Auto ETS, ARIMA(1,1,1) $(0,1,1)_7$, and regression model R2 all outperform the three benchmark models in every error metric considered for both the in and out of sample data. This is quite promising as it demonstrates the more advanced models do indeed model the characteristics of the time series better through the use of their respective features. All three models are suitable models for the given forecasting problem.

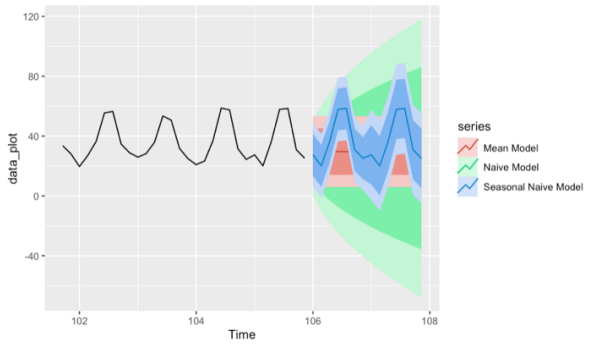


Figure 12: Forecast of Benchmark Models

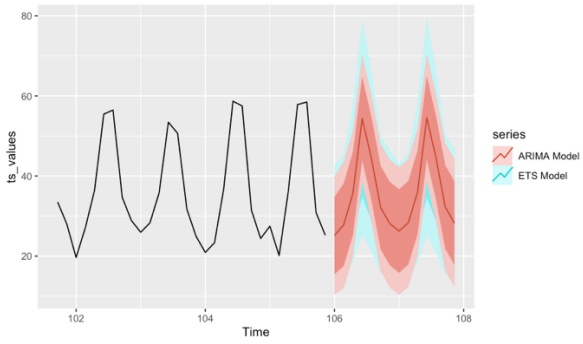


Figure 13: Forecast of ARIMA and ETS models

The neural network trained performs better than the benchmark models in every measure except sMAPE. The higher sMAPE highlights a key problem in the training and selection of neural networks. The neural networks with high number of lagged inputs gave residuals that are uncorrelated but by doing so ended up overfitting the model. This is evidenced by the extreme discrepancy between the RMSE of the residuals and the RMSE from cross validation. On the other hand, those that made use of fewer lagged inputs did not overfit but instead had high amounts of residuals that were autocorrelated.

Of the three suitable models, the best model is the ARIMA model which gives the lowest MAE and sMAPE errors. The use of autoregressive terms and moving average terms in a seasonal ARIMA model was able to forecast out of sample data with really high accuracy. Furthermore, the discrepancy between the residual RMSE and CV RMSE is minimal which suggest that the accuracy of the forecast will be as impressive regardless of the forecast origin. The final forecast for the ARIMA model is shown in the Table 13 below. The next best model is the ETS that was automatically trained. Whilst it has an extremely low validation MAE and the second lowest sMAPE, it is likely it will not perform as well if the forecast origin is shifted. This is because the cross validation RMSE is much larger than the RMSE of the residuals of the trained model. The forecast for these two models are illustrated in Figure 13 above where it can be seen the ARIMA forecast has a smaller prediction interval compared to the ETS forecast. From the results, the regression model is more reliable as although it has a slightly higher MAE and sMAPE than the ETS model, it has a much lower cross validation RMSE. The performance of the regression could potentially benefit from being trained with a longer time series. It is also possible that an important external factor that affects the time series was not considered and used as an independent variable to the regression model.

Table 13: Final 14 Day Forecast of Best ARIMA model

Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
25.12402	27.81674	35.66635	54.34178	44.42165	32.11804	28.08995
Day 8	Day 9	Day 10	Day 11	Day 12	Day 13	Day 14
26.24371	28.3005	35.91997	54.51212	44.56184	32.24733	28.21529

Conclusion

Based on the finding from this study ARIMA models and Regression models are good algorithms for modelling and forecasting time series that display a trend and seasonality. First and foremost, any of the four advanced models need to outperform the base model to even be considered a suitable model for the forecasting task at hand. The reason behind this is it would make little sense to recommend a model that is more complex but performs worse than a simpler model that is computationally less costly. The seasonal ARIMA$(1,1,1)(0,1,1)_7$ was the undisputed best forecasting model of the many models explored in this study. It neither overfits or underfits, giving a forecast that is reliable and consistent. The results of the optimal regression model that makes use of information regarding trend, day in a week seasonality, quarterly seasonality, missing values, outliers, bank holidays and the first two lags also shows promise. The use of these external factors makes it more versatile as it could serve to be beneficial for time series that have different characteristics such as structural breaks

References

https://analyticsindiamag.com/a-guide-to-different-evaluation-metrics-for-time-series-forecasting-models/

https://towardsdatascience.com/choosing-the-correct-error-metric-mape-vs-smape-5328dec53fac

https://medium.com/analytics-vidhya/benchmarking-methods-for-deep-learning-based-time-series-forecast-ec45f78b61e2

Appendix A - ETS

Table 14: Range of ETS Hyperparameters

Models	Parameters	Start	Stop	Step
Simple Exponential Smoothing	Alpha	0	1	0.001
Holt's Linear Trend Method	Beta	0	0.004	0.0001
Holt's Linear Trend Method with Damping	Phi	0.8	0.98	0.001
Holt-Winters' Seasonal Method	Gamma	0	0.996	0.001

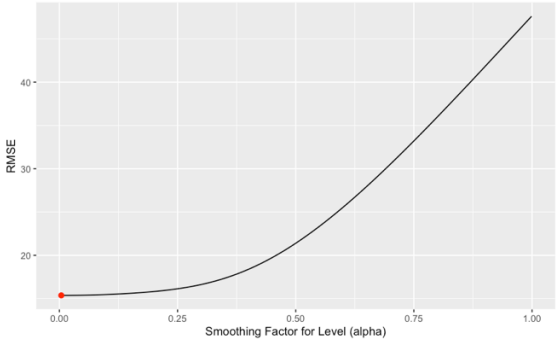


Figure 14: Hyperparameter Tuning for Alpha

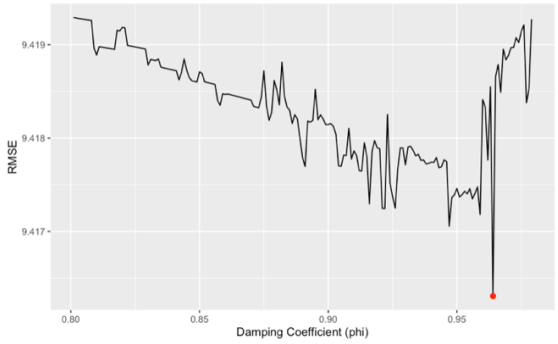


Figure 16: Hyperparameter Tuning for Phi

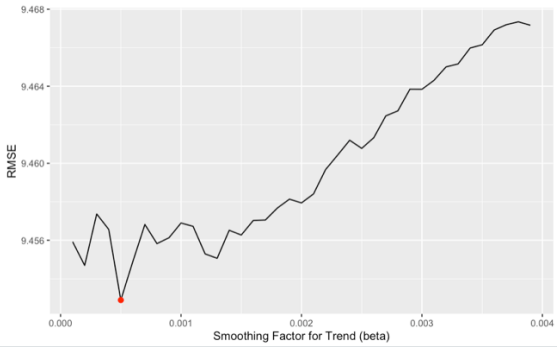


Figure 15: Hyperparameter Tuning for Beta

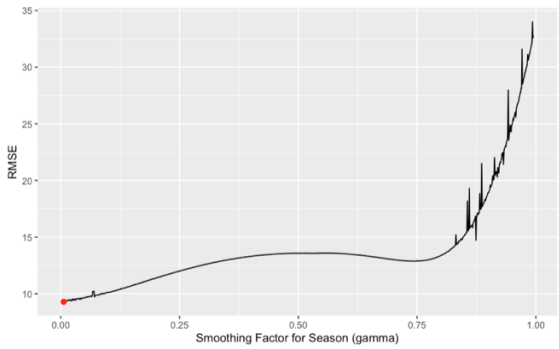


Figure 17: Hyperparameter Tuning for Gamma

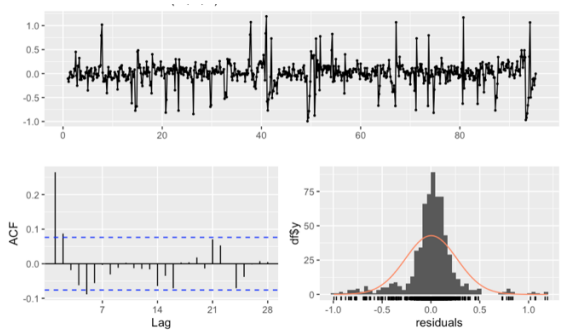


Figure 18: Residuals of Automatic ETS Model

Appendix B - ARIMA

Table 15: Other ARIMA Models

ARIMA models:	AICc	Residuals RMSE	CV RMSE	In/Out sample	RMSE	MAE
ARIMA (0,1,1) $(1,1,1)_7$	4589.04	7.8722	9.4111	Train	7.8243	4.6756
				Validation	6.4753	5.9733
ARIMA (1,1,2) $(0,1,1)_7$	4507.95	7.4704	8.6598	Train	7.3309	4.3771
				Validation	3.5076	2.7680
ARIMA (2,1,1) $(0,1,1)_7$	4507.38	7.4677	8.7350	Train	7.3287	4.3669
				Validation	3.5120	2.7735
ARIMA (1,1,2) $(0,1,2)_7$	4508.88	7.4299	8.7241	Train	7.2932	4.3412
				Validation	3.5192	2.7700
ARIMA (2,1,1) $(0,1,2)_7$	4508.31	7.4278	8.7312	Train	7.2897	4.3290
				Validation	3.5264	2.7740
ARIMA (2,1,1) $(1,1,1)_7$	4508.42	7.4296	8.5536	Train	7.2904	4.3312
				Validation	3.5401	2.7864
ARIMA (1,1,2) $(1,1,1)_7$	4508.99	7.4318	8.5862	Train	7.2938	4.3429
				Validation	3.5338	2.7830
ARIMA (2,1,2) $(0,1,1)_7$	4508.54	7.4644	8.5924	Train	7.3291	4.3657
				Validation	3.5029	2.7736
ARIMA (2,1,2) $(0,1,2)_7$	4509.44	7.4233	8.6763	Train	7.2871	4.3266
				Validation	3.5044	2.7556

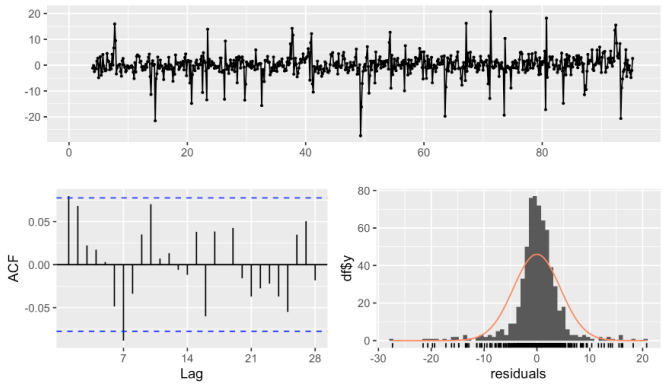


Figure 19: Residuals of Automatic ARIMA Model

Appendix C - Regression

Table 16: Variable Selection

Model No.	Weekly Seasonality	Trend	Quarterly Seasonality	Monthly Seasonality	NA	Outliers	Bank Holiday	Lag 7
1 (Base)	Y	Y	-	-	-	-	-	-
2	Y	Y	Y	-	-	-	-	-
3	Y	Y	-	N	-	-	-	-
4	Y	Y	Y	N	-	-	-	-
5	Y	Y	-	-	Y	-	-	-
6	Y	Y	-	-	-	Y	-	-
7	Y	Y	-	-	-	-	Y	-
8	Y	Y	-	-	-	-	-	N

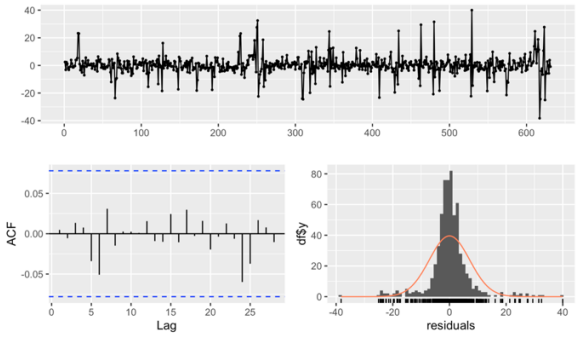


Figure 20: Residuals of Automatic Regression Model

Appendix D – Neural Networks

Table 17: Other Neural Network Models

Model	p-value	Residuals RMSE	Cross-validation RMSE	In/Out sample	RMSE	MAE	sMAPE

NNAR(1,1,k)	0.0087	8.4793	10.0154	Train	8.1927	5.5046	-
				Validation	10.0085	7.7315	0.3387
NNAR(1,4,k)	0.5871	7.5816	9.8155	Train	7.2596	4.7420	-
				Validation	10.4720	9.2444	0.3031
NNAR(2,0,k)	0.0000	9.0765	12.4463	Train	8.7869	6.2581	-
				Validation	9.6201	7.4634	0.3133
NNAR(2,1,k)	0.0000	8.1092	9.9093	Train	7.9793	5.3653	-
				Validation	8.3820	6.6336	0.3490
NNAR(2,2,k)	0.0001	6.9688	9.7711	Train	7.7328	5.1334	-
				Validation	11.8993	9.3212	0.3450
NNAR(2,3,k)	0.1368	7.5435	9.8883	Train	7.1720	4.7170	-
				Validation	13.1002	10.6401	0.3567
NNAR(2,4,k)	0.4868	7.2361	9.8353	Train	6.9536	4.5321	-
				Validation	11.2771	9.2249	0.3191
NNAR(5,0,k)	0.0000	7.8937	10.5923	Train	7.5324	5.0684	-
				Validation	4.8077	3.6507	0.1853
NNAR(5,1,k)	0.0446	7.1322	9.6849	Train	6.8052	4.4329	-
				Validation	6.9015	5.8780	0.2388
NNAR(5,2,k)	0.0101	6.9594	9.4531	Train	6.7364	4.3536	-
				Validation	12.3808	10.1092	0.4385
NNAR(5,3,k)	0.1829	6.8564	9.5243	Train	6.5360	4.2400	-
				Validation	12.9077	9.8685	0.3824
NNAR(7,0,k)	0.0043	6.9451	9.7586	Train	6.7022	4.3312	-
				Validation	7.2641	6.1017	0.2395
NNAR(7,1,k)	0.0173	6.9925	9.7153	Train	6.7460	4.3262	-
				Validation	7.5983	6.5844	0.2672
NNAR(7,2,k)	0.0018	6.7078	9.8334	Train	6.5313	4.2126	-
				Validation	14.1621	11.6399	0.4912
NNAR(14,1,k)	0.2141	4.9951	10.0247	Train	4.6961	2.9863	-
				Validation	14.5215	12.0429	0.3888

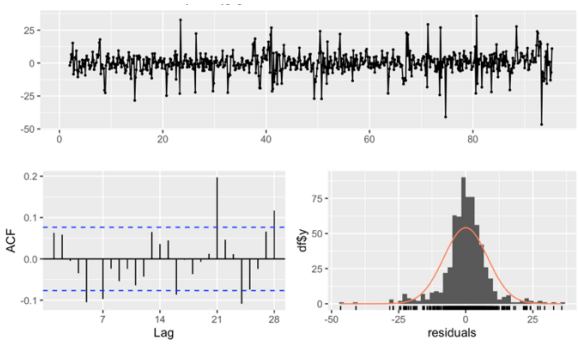


Figure 21: Residuals of NNAR(1,1,k)

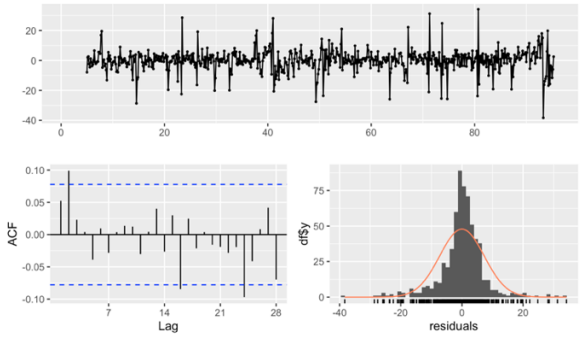


Figure 22: Residuals of NNAR(1,4,k)

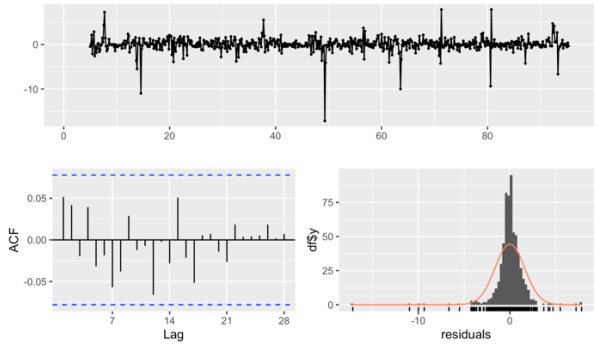


Figure 23: Residuals of Automatic Neural Network Model

Appendix E – ETS Code

```
library(dplyr)
library(ggplot2)
library(forecast)
library(smooth)
library(TSstudio)

# Import data, split data into train and validation test
data <- read.csv('cleaned_data.csv',header=T)
dates <- data$date_obj

# create ts object
ts_data <- ts(data$values_no_na, frequency = 7)

# split time series into train and validation sets (90:10 ratio)
total_obs <- length(ts_data)
split_ts <- ts_split(ts.obj = ts_data, sample.out = round(0.1*total_obs))
train_set <- split_ts$train
val_set <- split_ts$test

# verify length of each set
length(train_set)
length(val_set)
```

```
# Differencing time series
diff_train_7 <- diff(train_set, lag = 7)
diff_val_7 <- diff(val_set, lag = 7)
diff_train_7_1 <-  diff(diff(train_set, lag = 7), lag = 1)
diff_val_7_1 <- diff(diff(val_set, lag = 7), lag = 1)

# Single Exponential Smoothing (Simple)
alpha <- seq(0, 1, 0.001)
alpha <- head(tail(alpha, -1), -1)
ses_RMSE_7_1 <- NA
ses_RMSE_7_1_train <- NA

for (i in seq_along(alpha)){
  fit <- ses(diff_train_7_1, alpha = alpha[i], h = 14)
  ses_RMSE_7_1[i] <- forecast::accuracy(fit, diff_val_7_1)[2,2]
}

alpha_ses_RMSE_7_1 <- data_frame(alpha, ses_RMSE_7_1)
min_alpha_ses_RMSE_7_1 <- filter(alpha_ses_RMSE_7_1, ses_RMSE_7_1 ==
min(ses_RMSE_7_1))

# plot RMSE against alpha values
ggplot(alpha_ses_RMSE_7_1, aes(alpha, ses_RMSE_7_1)) +
  geom_line() +
  geom_point(data = min_alpha_ses_RMSE_7_1, aes(alpha, ses_RMSE_7_1), size = 2, color
= "red") +
  xlab("Smoothing Factor for Level (alpha)") +
  ylab("RMSE")

min_alpha_ses_RMSE_7_1

fit <- ses(diff_train_7_1, alpha = 0.008, h = 14)
ses_cv_e <- tsCV(diff_train_7_1, ses, alpha = 0.008, h = 14)
sqrt(mean(ses_cv_e^2, na.rm=TRUE))
sqrt(mean(residuals(ses(diff_train_7_1, alpha = 0.008, h = 14))^2, na.rm=TRUE))

autoplot(fit) +
  ggtitle("Forecast of ses() with optimal alpha") +
  ylab("values")

# Double Exponential Smoothing (without damping)
beta <- seq(0, 0.008, 0.0001)
beta <- head(tail(beta, -1), -1)
holt_RMSE_7 <- NA

for (i in seq_along(beta)){
  fit <- holt(diff_train_7, beta = beta[i], h = 14)
  holt_RMSE_7[i] <- forecast::accuracy(fit, diff_val_7)[2,2]
}
```

```
beta_holt_RMSE_7 <- data_frame(beta, holt_RMSE_7)
min_beta_holt_RMSE_7 <- filter(beta_holt_RMSE_7, holt_RMSE_7 ==
min(holt_RMSE_7))

# plot RMSE against beta values
ggplot(beta_holt_RMSE_7, aes(beta, holt_RMSE_7)) +
  geom_line() +
  geom_point(data = min_beta_holt_RMSE_7, aes(beta, holt_RMSE_7), size = 2, color =
"red") +
  xlab("Smoothing Factor for Trend (beta)") +
  ylab("RMSE")

min_beta_holt_RMSE_7

fit <- holt(diff_train_7, beta = 0.0079, h = 14)

beta_holt_cv_e <- tsCV(diff_train_7, holt, beta = 0.0079, h = 14)
sqrt(mean(beta_holt_cv_e^2, na.rm=TRUE))
sqrt(mean(residuals(holt(diff_train_7, beta = 0.0079, h = 14))^2, na.rm=TRUE))

autoplot(fit) +
  ggtitle("Forecast of holt() with optimal beta") +
  ylab("values")

# Double Exponential Smoothing (with damping)
phi <- seq(0.80, 0.98, 0.001)
phi <- head(tail(phi, -1), -1)
holt_RMSE_7_phi <- NA

for (i in seq_along(phi)){
  fit <- holt(diff_train_7, phi = phi[i], h = 14, damped = TRUE)
  holt_RMSE_7_phi[i] <- forecast::accuracy(fit, diff_val_7)[2,2]
}

phi_holt_RMSE_7 <- data_frame(phi, holt_RMSE_7_phi)
min_phi_holt_RMSE_7 <- filter(phi_holt_RMSE_7, holt_RMSE_7_phi ==
min(holt_RMSE_7_phi))

# plot RMSE against phi values
ggplot(phi_holt_RMSE_7, aes(phi, holt_RMSE_7_phi)) +
  geom_line() +
  geom_point(data = min_phi_holt_RMSE_7, aes(phi, holt_RMSE_7_phi), size = 2, color =
"red") +
  xlab("Damping Coefficient (phi)") +
  ylab("RMSE")

min_phi_holt_RMSE_7

fit <- holt(diff_train_7, phi = 0.964, h = 14)
```

```
phi_holt_cv_e <- tsCV(diff_train_7, holt, phi = 0.964, h = 14, damped = TRUE)
sqrt(mean(phi_holt_cv_e^2, na.rm=TRUE))
sqrt(mean(residuals(holt(diff_train_7, phi = 0.964, h = 14, damped = TRUE))^2,
na.rm=TRUE))

autoplot(fit) +
  ggtitle("Forecast of holt() with optimal phi") +
  ylab("values")

# Triple Exponential Smoothing
gamma <- seq(0, 0.992, 0.001)
gamma <- head(tail(gamma, -1), -1)
hw_RMSE <- NA

for (i in seq_along(gamma)){
  # fit ets on original
  fit <- hw(train_set, gamma = gamma[i], h = 14)
  hw_RMSE[i] <- forecast::accuracy(fit, val_set)[2,2]
}

# create dataframes for plotting purposes
gamma_hw_RMSE <- data_frame(gamma, hw_RMSE)
min_hw_RMSE <- filter(gamma_hw_RMSE, hw_RMSE == min(hw_RMSE))

# plot RMSE against gamma values
ggplot(gamma_hw_RMSE, aes(gamma, hw_RMSE)) +
  geom_line() +
  geom_point(data = min_hw_RMSE, aes(gamma, hw_RMSE), size = 2, color = "red") +
  xlab("Smoothing Factor for Season (gamma)") +
  ylab("RMSE")

min_hw_RMSE

fit <- hw(train_set, gamma = 0.007, h = 14)

phi_holt_cv_e <- tsCV(train_set, hw, gamma = 0.007, h = 14, damped = TRUE)
sqrt(mean(phi_holt_cv_e^2, na.rm=TRUE))
sqrt(mean(residuals(hw(train_set, gamma = 0.007, h = 14, damped = TRUE))^2,
na.rm=TRUE))

autoplot(fit) +
  ggtitle("Forecast of holt() with optimal gamma") +
  ylab("values")

# Triple Exponential Smoothing (seasonality type)

# Case 1: multiplicative error, additive trend, multiplicative seasonality
optimal_ets_MAM <- ets(train_set, model = "MAM", alpha = 0.008, beta = 0.0079, phi =
0.964, gamma = 0.007)
forecast_optimal_ets <- forecast(optimal_ets_MAM, h = 14)
```

```
autoplot(forecast_optimal_ets)
forecast::accuracy(forecast_optimal_ets, val_set)
ets_func <- function(x, h) {forecast(ets(x, model = "MAM",
                                alpha = 0.008, beta = 0.0079, phi = 0.964, gamma = 0.007), h = h)}
cv_model <- tsCV(train_set, ets_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(ets(train_set, model = "MAM", alpha = 0.008, beta = 0.0079,
phi = 0.964, gamma = 0.007),
                    h = 14))^2, na.rm=TRUE))

Metrics::smape(val_set, forecast(optimal_ets_MAM, h = length(val_set))$mean)
summary(optimal_ets_MAM)
checkresiduals(optimal_ets_MAM)

# Case 2: multiplicative error, multiplicative trend, multiplicative seasonality
optimal_ets_MMM <- ets(train_set, model = "MMM", alpha = 0.008, beta = 0.0079, phi =
0.964, gamma = 0.007)
forecast_optimal_ets <- forecast(optimal_ets_MMM, h = 14)
autoplot(forecast_optimal_ets)
forecast::accuracy(forecast_optimal_ets, val_set)
ets_func <- function(x, h) {forecast(ets(x, model = "MMM",
                                alpha = 0.008, beta = 0.0079, phi = 0.964, gamma = 0.007), h = h)}
cv_model <- tsCV(train_set, ets_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(ets(train_set, model = "MMM", alpha = 0.008, beta = 0.0079,
phi = 0.964, gamma = 0.007),
                    h = 14))^2, na.rm=TRUE))

Metrics::smape(val_set, forecast(optimal_ets_MMM, h = length(val_set))$mean)
summary(optimal_ets_MMM)
checkresiduals(optimal_ets_MMM)

# Case 3: multiplicative error, additive trend, additive seasonality
optimal_ets_MAA <- ets(train_set, model = "MAA", alpha = 0.008, beta = 0.0079, phi =
0.964, gamma = 0.007)
forecast_optimal_ets <- forecast(optimal_ets_MAA, h = 14)
autoplot(forecast_optimal_ets)
forecast::accuracy(forecast_optimal_ets, val_set)
ets_func <- function(x, h) {forecast(ets(x, model = "MAA",
                                alpha = 0.008, beta = 0.0079, phi = 0.964, gamma = 0.007), h = h)}
cv_model <- tsCV(train_set, ets_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(ets(train_set, model = "MAA", alpha = 0.008, beta = 0.0079, phi
= 0.964, gamma = 0.007),
                    h = 14))^2, na.rm=TRUE))

Metrics::smape(val_set, forecast(optimal_ets_MAA, h = length(val_set))$mean)
summary(optimal_ets_MAA)
checkresiduals(optimal_ets_MAA)
```

```
# Case 4: additive error, additive trend, additive seasonality
optimal_ets_AAA <- ets(train_set, model = "AAA", alpha = 0.008, beta = 0.0079, phi =
0.964, gamma = 0.007)
forecast_optimal_ets <- forecast(optimal_ets_AAA, h = 14)
autoplot(forecast_optimal_ets)
forecast::accuracy(forecast_optimal_ets, val_set)
ets_func <- function(x, h) {forecast(ets(x, model = "AAA",
                                alpha = 0.008, beta = 0.0079, phi = 0.964, gamma = 0.007), h = h)}
cv_model <- tsCV(train_set, ets_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(ets(train_set, model = "AAA", alpha = 0.008, beta = 0.0079, phi
= 0.964, gamma = 0.007),
                    h = 14))^2, na.rm=TRUE))

Metrics::smape(val_set, forecast(optimal_ets_AAA, h = length(val_set))$mean)
summary(optimal_ets_AAA)
checkresiduals(optimal_ets_AAA)

# Optimal ETS
optimal_ets <- ets(train_set, model = "MAA", alpha = 0.004, beta = 5e-04,
                   phi = 0.964, gamma = 0.006)
summary(optimal_ets)
forecast_optimal_ets <- forecast(optimal_ets, h = length(val_set))
autoplot(forecast_optimal_ets) +
  ggtitle("Forecast of ets() with optimal parameters") +
  ylab("values")
accuracy(forecast_optimal_ets, val_set)

ets_func <- function(x, h) {forecast(ets(x, model = "MAA",
                                alpha = 0.004, beta = 5e-04,
                                phi = 0.964, gamma = 0.006), h = h)}
cv_model <- tsCV(ts_data, ets_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))

sqrt(mean(residuals(forecast(ets(ts_data, model = "ANA", alpha = 0.004,
                           beta = 5e-04, phi = 0.964, gamma = 0.006),
                    h = 14))^2, na.rm=TRUE))

# automated model
auto_ets <- ets(train_set, model = "ZZZ")
forecast_auto_ets <- forecast(auto_ets, h = 14)
autoplot(forecast_auto_ets)
forecast::accuracy(forecast_auto_ets, val_set)
ets_func <- function(x, h) {forecast(ets(x, model = "ZZZ"), h = h)}
cv_model <- tsCV(train_set, ets_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(ets(train_set, model = "ZZZ"), h = 14))^2, na.rm=TRUE))

Metrics::smape(val_set, forecast(auto_ets, h = length(val_set))$mean)
summary(forecast_auto_ets)
```

```
checkresiduals(forecast_auto_ets)
```

Appendix F – ARIMA Code

```
library(dplyr)
library(ggplot2)
library(forecast)
library(Metrics)
library(smooth)

# Import data, pre-processing and splitting into train and validation set
data <- read.csv('cleaned_data.csv',header=T)
dates <- data$date_obj

# create ts object
ts_data <- ts(data$values_no_na, frequency = 7)

# split time series into train and validation sets (90:10 ratio)
total_obs <- length(ts_data)
split_ts <- ts_split(ts.obj = ts_data, sample.out = round(0.1*total_obs))
train_set <- split_ts$train
val_set <- split_ts$test

# verify length of each set
length(train_set)
length(val_set)

# Differencing data
diff_train_7 <- diff(train_set, lag = 7)
diff_val_7 <- diff(val_set, lag = 7)

diff_train_7_1 <-  diff(diff(train_set, lag = 7), lag = 1)
diff_val_7_1 <- diff(diff(val_set, lag = 7), lag = 1)

diff_7 <- diff(ts_data, lag = 7)
diff_7_1 <- diff(ts_data, lag = 7) %>% diff()

tsdisplay(diff_7_1)
tsdisplay(diff_7)

# Manual ARIMA models

fit <- Arima(train_set, order = c(0,1,1), seasonal = c(0,1,1))
fit
checkresiduals(fit)
ggtsdisplay(residuals(fit))
autoplot(forecast(fit, h = length(val_set)))
forecast::accuracy(forecast(fit), val_set)[,c(2,3,5,6)]
```

```
arima_func <- function(x, h) {forecast(Arima(x, order = c(0,1,1), seasonal = c(0,1,1)), h =
h)}
cv_model <- tsCV(ts_data, arima_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(Arima(ts_data, order = c(0,1,1), seasonal = c(0,1,1)), h = 14))^2,
na.rm=TRUE))
Metrics::smape(val_set, forecast(Arima(ts_data, order = c(0,1,1), seasonal = c(0,1,1)))$fitted)

fit <- Arima(train_set, order = c(1,1,1), seasonal = c(0,1,1))
fit
checkresiduals(fit)
ggtsdisplay(residuals(fit))
autoplot(forecast(fit, h = length(val_set)))
forecast::accuracy(forecast(fit), val_set)[,c(2,3,5,6)]
arima_func <- function(x, h) {forecast(Arima(x, order = c(1,1,1), seasonal = c(0,1,1)), h =
h)}
cv_model <- tsCV(ts_data, arima_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(Arima(ts_data, order = c(1,1,1), seasonal = c(0,1,1)), h = 14))^2,
na.rm=TRUE))
Metrics::smape(val_set, forecast(Arima(ts_data, order = c(1,1,1), seasonal = c(0,1,1)))$fitted)

fit <- Arima(train_set, order = c(1,1,1), seasonal = c(1,1,1))
fit
checkresiduals(fit)
ggtsdisplay(residuals(fit))
autoplot(forecast(fit, h = length(val_set)))
forecast::accuracy(forecast(fit), val_set)[,c(2,3,5,6)]
arima_func <- function(x, h) {forecast(Arima(x, order = c(1,1,1), seasonal = c(1,1,1)), h =
h)}
cv_model <- tsCV(ts_data, arima_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(Arima(ts_data, order = c(1,1,1), seasonal = c(1,1,1)), h = 14))^2,
na.rm=TRUE))
Metrics::smape(val_set, forecast(Arima(ts_data, order = c(1,1,1), seasonal = c(1,1,1)))$fitted)

fit <- Arima(train_set, order = c(0,1,1), seasonal = c(1,1,1))
fit
checkresiduals(fit)
ggtsdisplay(residuals(fit))
autoplot(forecast(fit, h = length(val_set)))
forecast::accuracy(forecast(fit), val_set)[,c(2,3,5,6)]
arima_func <- function(x, h) {forecast(Arima(x, order = c(0,1,1), seasonal = c(1,1,1)), h =
h)}
cv_model <- tsCV(ts_data, arima_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(Arima(ts_data, order = c(0,1,1), seasonal = c(1,1,1)), h = 14))^2,
na.rm=TRUE))
Metrics::smape(val_set, forecast(Arima(ts_data, order = c(0,1,1), seasonal = c(1,1,1)))$fitted)
```

```
fit <- Arima(train_set, order = c(1,1,1), seasonal = c(0,1,2))
fit
checkresiduals(fit)
ggtsdisplay(residuals(fit))
autoplot(forecast(fit, h = length(val_set)))
forecast::accuracy(forecast(fit), val_set)[,c(2,3,5,6)]
arima_func <- function(x, h) {forecast(Arima(x, order = c(1,1,1), seasonal = c(0,1,2)), h =
h)}
cv_model <- tsCV(ts_data, arima_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(Arima(ts_data, order = c(1,1,1), seasonal = c(0,1,2)), h = 14))^2,
na.rm=TRUE))
Metrics::smape(val_set, forecast(Arima(ts_data, order = c(1,1,1), seasonal = c(0,1,2)))$fitted)

fit <- Arima(train_set, order = c(1,1,2), seasonal = c(0,1,1))
fit
checkresiduals(fit)
ggtsdisplay(residuals(fit))
autoplot(forecast(fit, h = length(val_set)))
forecast::accuracy(forecast(fit), val_set)[,c(2,3,5,6)]
arima_func <- function(x, h) {forecast(Arima(x, order = c(1,1,2), seasonal = c(0,1,1)), h =
h)}
cv_model <- tsCV(ts_data, arima_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(Arima(ts_data, order = c(1,1,2), seasonal = c(0,1,1)), h = 14))^2,
na.rm=TRUE))
Metrics::smape(val_set, forecast(Arima(ts_data, order = c(1,1,2), seasonal = c(0,1,1)))$fitted)

fit <- Arima(train_set, order = c(2,1,1), seasonal = c(0,1,1))
fit
checkresiduals(fit)
ggtsdisplay(residuals(fit))
autoplot(forecast(fit, h = length(val_set)))
forecast::accuracy(forecast(fit), val_set)[,c(2,3,5,6)]
arima_func <- function(x, h) {forecast(Arima(x, order = c(2,1,1), seasonal = c(0,1,1)), h =
h)}
cv_model <- tsCV(ts_data, arima_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(Arima(ts_data, order = c(2,1,1), seasonal = c(0,1,1)), h = 14))^2,
na.rm=TRUE))
Metrics::smape(val_set, forecast(Arima(ts_data, order = c(2,1,1), seasonal = c(0,1,1)))$fitted)

fit <- Arima(train_set, order = c(1,1,2), seasonal = c(0,1,2))
fit
checkresiduals(fit)
ggtsdisplay(residuals(fit))
autoplot(forecast(fit, h = length(val_set)))
forecast::accuracy(forecast(fit), val_set)[,c(2,3,5,6)]
arima_func <- function(x, h) {forecast(Arima(x, order = c(1,1,2), seasonal = c(0,1,2)), h =
h)}
```

```
cv_model <- tsCV(ts_data, arima_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(Arima(ts_data, order = c(1,1,2), seasonal = c(0,1,2)), h = 14))^2,
na.rm=TRUE))
Metrics::smape(val_set, forecast(Arima(ts_data, order = c(1,1,2), seasonal = c(0,1,2)))$fitted)

fit <- Arima(train_set, order = c(2,1,1), seasonal = c(0,1,2))
fit
checkresiduals(fit)
ggtsdisplay(residuals(fit))
autoplot(forecast(fit, h = length(val_set)))
forecast::accuracy(forecast(fit), val_set)[,c(2,3,5,6)]
arima_func <- function(x, h) {forecast(Arima(x, order = c(2,1,1), seasonal = c(0,1,2)), h =
h)}
cv_model <- tsCV(ts_data, arima_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(Arima(ts_data, order = c(2,1,1), seasonal = c(0,1,2)), h = 14))^2,
na.rm=TRUE))
Metrics::smape(val_set, forecast(Arima(ts_data, order = c(2,1,1), seasonal = c(0,1,2)))$fitted)

fit <- Arima(train_set, order = c(2,1,1), seasonal = c(1,1,1))
fit
checkresiduals(fit)
ggtsdisplay(residuals(fit))
autoplot(forecast(fit, h = length(val_set)))
forecast::accuracy(forecast(fit), val_set)[,c(2,3,5,6)]
arima_func <- function(x, h) {forecast(Arima(x, order = c(2,1,1), seasonal = c(1,1,1)), h =
h)}
cv_model <- tsCV(ts_data, arima_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(Arima(ts_data, order = c(2,1,1), seasonal = c(1,1,1)), h = 14))^2,
na.rm=TRUE))
Metrics::smape(val_set, forecast(Arima(ts_data, order = c(2,1,1), seasonal = c(1,1,1)))$fitted)

fit <- Arima(train_set, order = c(1,1,2), seasonal = c(1,1,1))
fit
checkresiduals(fit)
ggtsdisplay(residuals(fit))
autoplot(forecast(fit, h = length(val_set)))
forecast::accuracy(forecast(fit), val_set)[,c(2,3,5,6)]
arima_func <- function(x, h) {forecast(Arima(x, order = c(1,1,2), seasonal = c(1,1,1)), h =
h)}
cv_model <- tsCV(ts_data, arima_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(Arima(ts_data, order = c(1,1,2), seasonal = c(1,1,1)), h = 14))^2,
na.rm=TRUE))
Metrics::smape(val_set, forecast(Arima(ts_data, order = c(1,1,2), seasonal = c(1,1,1)))$fitted)

fit <- Arima(train_set, order = c(2,1,2), seasonal = c(0,1,1))
fit
```

```
checkresiduals(fit)
ggtsdisplay(residuals(fit))
autoplot(forecast(fit, h = length(val_set)))
forecast::accuracy(forecast(fit), val_set)[,c(2,3,5,6)]
arima_func <- function(x, h) {forecast(Arima(x, order = c(2,1,2), seasonal = c(0,1,1)), h =
h)}
cv_model <- tsCV(ts_data, arima_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(Arima(ts_data, order = c(2,1,2), seasonal = c(0,1,1)), h = 14))^2,
na.rm=TRUE))
Metrics::smape(val_set, forecast(Arima(ts_data, order = c(2,1,2), seasonal = c(0,1,1)))$fitted)

fit <- Arima(train_set, order = c(2,1,2), seasonal = c(0,1,2))
fit
checkresiduals(fit)
ggtsdisplay(residuals(fit))
autoplot(forecast(fit, h = length(val_set)))
forecast::accuracy(forecast(fit), val_set)[,c(2,3,5,6)]
arima_func <- function(x, h) {forecast(Arima(x, order = c(2,1,2), seasonal = c(0,1,2)), h =
h)}
cv_model <- tsCV(ts_data, arima_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(Arima(ts_data, order = c(2,1,2), seasonal = c(0,1,2)), h = 14))^2,
na.rm=TRUE))
Metrics::smape(val_set, forecast(Arima(ts_data, order = c(2,1,2), seasonal = c(0,1,2)))$fitted)

# Auto ARIMA model

fit <- auto.arima(train_set)
fit
checkresiduals(fit)
autoplot(forecast(fit, h = 14))
forecast::accuracy(forecast(fit), length(val_set))[,c(2,3,5,6)]

arima_func <- function(x, h) {forecast(auto.arima(train_set), h = h)}
cv_model <- tsCV(ts_data, arima_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(auto.arima(ts_data), h = 14))^2, na.rm=TRUE))

Metrics::smape(test, forecast(auto.arima(ts_data))$fitted)
```

Appendix G – Regression Code

```
library(ggplot2)
library(forecast)
library(dplyr)
library(imputeTS)
library(smooth)
library(Metrics)
library(lubridate)
```

```
library(timeDate)
library(TSstudio)
library(timetk)
library(tidyquant)
library(car)

# Import data and split into train and validation set
data <- read.csv('NN5-034.csv',header=T)[-1:-3,]

# All missing values are treated as NA
data[data == ''] <- NA
colnames(data) <- c('date','values')
data$date_obj <- dmy(data$date)

# set start date and frequency (daily data so set to 365.25)
data$ts <- ts(data$values, frequency = 365.25, start = c(1996, 3, 18))

# Replace zero values with NA
data$values <- as.numeric(data$values)
data$ts <- ifelse(data$values == 0, NA, data$values)

# Linear interpolation to fill missing values in time series
data$ts_no_na <- na_interpolation(data$ts, option = "linear")

ts_data <- ts(data$ts_no_na, frequency = 7)

ts_df <- data.frame(value = ts_data, date = data$date_obj)

complete_df <- data.frame(ts_no_na = ts_data, date_obj = data$date_obj, values =
data$values, ts_no_na = data$ts_no_na)

# detect and create dummy variables for outliers
stl_decomposition <- stl(complete_df$ts_no_na, 7)
plot(stl_decomposition)
ts_comp <- as.data.frame(stl_decomposition$time.series)
tsdisplay(ts_comp$remainder)

tsoutliers(complete_df$ts_no_na)

boxplot(complete_df$ts_no_na)

outliers <- which(abs(ts_comp$remainder) > 2*sd(ts_comp$remainder))
remainder_df <- as.data.frame(ts_comp$remainder)
colnames(remainder_df) <- "remainders"
remainder_df$outliers <- NA
remainder_df$outliers[outliers] <- remainder_df$remainders[outliers]

# create dummy variables for outlier
complete_df$outliers <- ifelse(is.na(remainder_df$outliers), 0, 1)
```

```
# create dummy variables for NA
complete_df$na <- ifelse(is.na(complete_df$values), 1, 0)

# Create dummy variables for bank holidays
complete_df$bank_holidays <- as.integer(complete_df$date_obj %in%
as.Date(holidayLONDON(1996:1998)))

data_transform <- complete_df

model_df <- data.frame(y = ts_data)
model_df$day_num <- wday(data_transform$date_obj)
model_df$week_num <- week(data_transform$date_obj)

model_df$week_num[model_df$week_num == 5] <- 1
model_df$week_num[model_df$week_num == 6] <- 2
model_df$week_num[model_df$week_num == 7] <- 3
model_df$week_num[model_df$week_num == 8] <- 4
model_df$week_num[model_df$week_num == 9] <- 1
model_df$week_num[model_df$week_num == 10] <- 2
model_df$week_num[model_df$week_num == 11] <- 3
model_df$week_num[model_df$week_num == 12] <- 4
model_df$week_num[model_df$week_num == 13] <- 1
model_df$week_num[model_df$week_num == 14] <- 2
model_df$week_num[model_df$week_num == 15] <- 3
model_df$week_num[model_df$week_num == 16] <- 4
model_df$week_num[model_df$week_num == 17] <- 1
model_df$week_num[model_df$week_num == 18] <- 2
model_df$week_num[model_df$week_num == 19] <- 3
model_df$week_num[model_df$week_num == 20] <- 4
model_df$week_num[model_df$week_num == 21] <- 1
model_df$week_num[model_df$week_num == 22] <- 2
model_df$week_num[model_df$week_num == 23] <- 3
model_df$week_num[model_df$week_num == 24] <- 4
model_df$week_num[model_df$week_num == 21] <- 1
model_df$week_num[model_df$week_num == 22] <- 2
model_df$week_num[model_df$week_num == 23] <- 3
model_df$week_num[model_df$week_num == 24] <- 4
model_df$week_num[model_df$week_num == 25] <- 1
model_df$week_num[model_df$week_num == 26] <- 2
model_df$week_num[model_df$week_num == 27] <- 3
model_df$week_num[model_df$week_num == 28] <- 4
model_df$week_num[model_df$week_num == 29] <- 1
model_df$week_num[model_df$week_num == 30] <- 2
model_df$week_num[model_df$week_num == 31] <- 3
model_df$week_num[model_df$week_num == 32] <- 4
model_df$week_num[model_df$week_num == 33] <- 1
model_df$week_num[model_df$week_num == 34] <- 2
model_df$week_num[model_df$week_num == 35] <- 3
model_df$week_num[model_df$week_num == 36] <- 4
model_df$week_num[model_df$week_num == 37] <- 1
```

```
model_df$week_num[model_df$week_num == 38] <- 2
model_df$week_num[model_df$week_num == 39] <- 3
model_df$week_num[model_df$week_num == 40] <- 4
model_df$week_num[model_df$week_num == 41] <- 1
model_df$week_num[model_df$week_num == 42] <- 2
model_df$week_num[model_df$week_num == 43] <- 3
model_df$week_num[model_df$week_num == 44] <- 4
model_df$week_num[model_df$week_num == 45] <- 1
model_df$week_num[model_df$week_num == 46] <- 2
model_df$week_num[model_df$week_num == 47] <- 3
model_df$week_num[model_df$week_num == 48] <- 4

# identify month number
model_df$month_num <- month(data_transform$date_obj)

model_df$month_num[model_df$month_num == 1] <- 1
model_df$month_num[model_df$month_num == 2] <- 1
model_df$month_num[model_df$month_num == 3] <- 1
model_df$month_num[model_df$month_num == 4] <- 2
model_df$month_num[model_df$month_num == 5] <- 2
model_df$month_num[model_df$month_num == 6] <- 2
model_df$month_num[model_df$month_num == 7] <- 3
model_df$month_num[model_df$month_num == 8] <- 3
model_df$month_num[model_df$month_num == 9] <- 3
model_df$month_num[model_df$month_num == 10] <- 4
model_df$month_num[model_df$month_num == 11] <- 4
model_df$month_num[model_df$month_num == 12] <- 4

model_df$lag_1 <- scale(dplyr::lag(as.numeric(ts_data), n = 1))
model_df$lag_2 <- scale(dplyr::lag(as.numeric(ts_data), n = 2))
model_df$lag_3 <- scale(dplyr::lag(as.numeric(ts_data), n = 3))
model_df$lag_4 <- scale(dplyr::lag(as.numeric(ts_data), n = 4))
model_df$lag_5 <- scale(dplyr::lag(as.numeric(ts_data), n = 5))
model_df$lag_6 <- scale(dplyr::lag(as.numeric(ts_data), n = 6))
model_df$lag_7 <- scale(dplyr::lag(as.numeric(ts_data), n = 7))
model_df$lag_8 <- scale(dplyr::lag(as.numeric(ts_data), n = 8))
model_df$lag_9 <- scale(dplyr::lag(as.numeric(ts_data), n = 9))
model_df$lag_10 <- scale(dplyr::lag(as.numeric(ts_data), n = 10))
model_df$lag_11 <- scale(dplyr::lag(as.numeric(ts_data), n = 11))
model_df$lag_12 <- scale(dplyr::lag(as.numeric(ts_data), n = 12))
model_df$lag_13 <- scale(dplyr::lag(as.numeric(ts_data), n = 13))
model_df$lag_14 <- scale(dplyr::lag(as.numeric(ts_data), n = 14))
model_df$lag_15 <- scale(dplyr::lag(as.numeric(ts_data), n = 15))
model_df$lag_16 <- scale(dplyr::lag(as.numeric(ts_data), n = 16))
model_df$lag_17 <- scale(dplyr::lag(as.numeric(ts_data), n = 17))
model_df$lag_18 <- scale(dplyr::lag(as.numeric(ts_data), n = 18))
model_df$lag_19 <- scale(dplyr::lag(as.numeric(ts_data), n = 19))
model_df$lag_20 <- scale(dplyr::lag(as.numeric(ts_data), n = 20))
model_df$lag_21 <- scale(dplyr::lag(as.numeric(ts_data), n = 21))
model_df$lag_22 <- scale(dplyr::lag(as.numeric(ts_data), n = 22))
```

```
model_df$lag_23 <- scale(dplyr::lag(as.numeric(ts_data), n = 23))
model_df$lag_24 <- scale(dplyr::lag(as.numeric(ts_data), n = 24))
model_df$lag_25 <- scale(dplyr::lag(as.numeric(ts_data), n = 25))
model_df$lag_26 <- scale(dplyr::lag(as.numeric(ts_data), n = 26))
model_df$lag_27 <- scale(dplyr::lag(as.numeric(ts_data), n = 27))
model_df$lag_28 <- scale(dplyr::lag(as.numeric(ts_data), n = 28))
model_df$lag_29 <- scale(dplyr::lag(as.numeric(ts_data), n = 29))
model_df$lag_30 <- scale(dplyr::lag(as.numeric(ts_data), n = 30))

# create weekly seasonality dummy variable
model_df$day_num1 <- 0
model_df$day_num2 <- 0
model_df$day_num3 <- 0
model_df$day_num4 <- 0
model_df$day_num5 <- 0
model_df$day_num6 <- 0
model_df$day_num7 <- 0

model_df$day_num1[model_df$day_num == 2] <- 1
model_df$day_num2[model_df$day_num == 3] <- 1
model_df$day_num3[model_df$day_num == 4] <- 1
model_df$day_num4[model_df$day_num == 5] <- 1
model_df$day_num5[model_df$day_num == 6] <- 1
model_df$day_num7[model_df$day_num == 7] <- 1

# create monthly seasonality dummy variables
model_df$week1 <- 0
model_df$week2 <- 0
model_df$week3 <- 0
model_df$week4 <- 0

model_df$week1[model_df$week_num == 2] <- 1
model_df$week2[model_df$week_num == 3] <- 1
model_df$week4[model_df$week_num == 4] <- 1

# create quarter seasonality dummy variables
model_df$quarter1 <- 0
model_df$quarter2 <- 0
model_df$quarter3 <- 0
model_df$quarter4 <- 0

model_df$quarter1[model_df$month_num == 2] <- 1
model_df$quarter2[model_df$month_num == 3] <- 1
model_df$quarter4[model_df$month_num == 4] <- 1

# Include other dummy variables
model_df$na <- data_transform$na
model_df$bank_holidays <- data_transform$bank_holidays
model_df$outliers <- data_transform$outliers
```

```
model_df <- select(model_df, -day_num, -week_num, -month_num)

# split time series into train and validation sets (90:10 ratio)
total_obs <- length(ts_data)
split_ts <- ts_split(ts.obj = ts_data, sample.out = round(0.1*total_obs))
train_set <- split_ts$train
val_set <- split_ts$test

# verify length of each set
length(train_set)
length(val_set)

train_df <- model_df[1:length(train_set),]
test_start_point <- length(train_set) + 1
test_df <- model_df[test_start_point:total_obs,]

train_ts_df <- ts_df[1:length(train_set),]
test_ts_df <- ts_df[test_start_point:total_obs,]

# Exploring different types of trend
linear_trend <- lm(value ~ as.numeric(date), data = train_ts_df)
forecast::accuracy(predict(linear_trend, test_ts_df), test_ts_df$value)[,c(2,3,5)]

exponential_trend <- lm(log(value) ~ as.numeric(date), data = train_ts_df)
forecast::accuracy(predict(exponential_trend, test_ts_df), test_ts_df$value)[,c(2,3,5)]

quadratic_trend <- lm(value ~ poly(as.numeric(date), 2), data = train_ts_df)
forecast::accuracy(predict(quadratic_trend, test_ts_df), test_ts_df$value)[,c(2,3,5)]

cubic_trend <- lm(value ~ poly(as.numeric(date), 3), data = train_ts_df)
forecast::accuracy(predict(cubic_trend, test_ts_df), test_ts_df$value)[,c(2,3,5)]

spline_model <- splinef(train_set, h = 14)
forecast::accuracy(spline_model, val_set)[,c(2,3,5)]

spline_model_box <- splinef(train_set, h = 14, lambda = "auto")
forecast::accuracy(spline_model_box, val_set)[,c(2,3,5)]

plot(linear_trend$fitted.values)
plot(quadratic_trend$fitted.values)
plot(cubic_trend$fitted.values)
plot(exponential_trend$fitted.values)
plot(spline_model$fitted)
plot(spline_model_box$fitted)

train_df$spline_model_box <- scale(spline_model_box$fitted)

spline_model_box_test <- splinef(val_set, h = 14, lambda = "auto")
test_df$spline_model_box <- scale(spline_model_box_test$fitted)
```

```
spline_model_box_all <- splinef(ts_data, h = 14, lambda = "auto")
model_df$spline_model_box <- scale(spline_model_box_all$fitted)

reg_initial <- lm(y ~ day_num1 + day_num2 + day_num3 + day_num4 + day_num5 +
                    day_num7 + spline_model_box,
                  data = train_df)

summary(reg_initial)
AICc(reg_initial)
BIC(reg_initial)
checkresiduals(reg_initial)
vif(reg_initial)

reg_quarter <- lm(y ~ day_num1 + day_num2 + day_num3 + day_num4 + day_num5 +
                    day_num7 + spline_model_box + quarter1 + quarter2 + quarter4,
                  data = train_df)

summary(reg_quarter)
checkresiduals(reg_quarter)
vif(reg_quarter)

reg_month <- lm(y ~ day_num1 + day_num2 + day_num3 + day_num4 + day_num5 +
                  day_num7 + spline_model_box + week1 + week2 + week4,
                data = train_df)


summary(reg_month)
checkresiduals(reg_month)
vif(reg_month)

reg_quarter_month <- lm(y ~ day_num1 + day_num2 + day_num3 + day_num4 + day_num5
+
                          day_num7 + spline_model_box + quarter1 + quarter2 + quarter4 +
                          week1 + week2 + week4,
                        data = train_df)

summary(reg_quarter_month)
checkresiduals(reg_quarter_month)
vif(reg_quarter_month)

reg_missing <- lm(y ~ day_num1 + day_num2 + day_num3 + day_num4 + day_num5 +
                    day_num7 + spline_model_box + na,
                  data = train_df)

summary(reg_missing)
vif(reg_missing)
checkresiduals(reg_missing)

reg_outliers <- lm(y ~ day_num1 + day_num2 + day_num3 + day_num4 + day_num5 +
                     day_num7 + spline_model_box + outliers,
```

```
                data = train_df)

summary(reg_outliers)
vif(reg_outliers)
checkresiduals(reg_outliers)

reg_bank <- lm(y ~ day_num1 + day_num2 + day_num3 + day_num4 + day_num5 +
                day_num7 + spline_model_box + bank_holidays,
              data = model_df)

summary(reg_bank)
checkresiduals(reg_bank)
vif(reg_bank)

reg_lag_7 <- lm(y ~ day_num1 + day_num2 + day_num3 + day_num4 + day_num5 +
                day_num7 + spline_model_box + lag_7,
              data = model_df)

summary(reg_lag_7)
checkresiduals(reg_lag_7)
vif(reg_lag_7)

reg_optimal_1 <- lm(y ~ day_num1 + day_num2 + day_num3 + day_num4 + day_num5 +
                    day_num7 + spline_model_box + quarter1 + quarter2 + quarter4 +
                    na  + bank_holidays + outliers, data = train_df)

summary(reg_optimal_1)
AICc(reg_optimal_1)
BIC(reg_optimal_1)
vif(reg_optimal_1)
checkresiduals(reg_optimal_1)

# regression with variables found to be significant + lags based on ACF
# added lags: 1, 2
reg_optimal_2 <- lm(y ~ day_num1 + day_num2 + day_num3 + day_num4 + day_num5 +
                    day_num7 + spline_model_box + quarter1 + quarter2 + quarter4 +
                    na  + bank_holidays + outliers + lag_1 + lag_2, data = train_df)

summary(reg_optimal_2)
AICc(reg_optimal_2)
BIC(reg_optimal_2)
vif(reg_optimal_2)
checkresiduals(reg_optimal_2)
sqrt(mean(residuals(reg_optimal_2)^2, na.rm=TRUE))

auto_train_df <- na.omit(train_df)
auto_reg_0 <- lm(y ~ 1, data = auto_train_df)
auto_reg_1 <- lm(y ~ ., data = auto_train_df)

auto_backward <- step(auto_reg_1, formula(auto_reg_1), direction = "backward", trace = 0)
```

```
summary(auto_backward)

# Last 14 values
last_14 <- tail(train_df, 14)

auto_backward_full <- lm(formula = y ~ lag_1 + lag_2 + lag_14 + lag_16 + lag_18 + lag_20
+
                         lag_21 + day_num1 + day_num2 + day_num3 + day_num4 + day_num5 +
                         day_num7 + quarter1 + quarter2 + quarter4 + na + bank_holidays +
                         outliers + spline_model_box, data = auto_train_df)

summary(auto_backward_full)
AICc(auto_backward_full)
BIC(auto_backward_full)

# in sample
forecast::accuracy(predict(auto_backward_full, last_14), last_14$y)
# out of sample
forecast::accuracy(predict(auto_backward_full, test_df), test_df$y)
Metrics::smape(test_df$y, predict(auto_backward_full, test_df))

checkresiduals(auto_backward_full)

# in sample
forecast::accuracy(predict(reg_optimal_1, last_14), last_14$y)
# out of sample
forecast::accuracy(predict(reg_optimal_1, test_df), test_df$y)
Metrics::smape(test_df$y, predict(reg_optimal_1, test_df))

# rolling origin
h <- 14
lag_no <- 0
start_point <- lag_no + 1
end_point <- length(train_set) - h - 1
medium_noise_length <- length(ts_data)
num_times_predict <- end_point - start_point + 1

medium_noise_forecasts <- matrix(NA, nrow=num_times_predict, ncol=h)
medium_noise_holdout <- matrix(NA, nrow=num_times_predict, ncol=h)
medium_noise_RMSE <- matrix(NA, nrow=num_times_predict, ncol=1)

for(i in start_point:end_point){
  train_start <- start_point
  train_end <- h + i - start_point
  our_train_set <- train_df[train_start:train_end, ]
  if (length(train_set) - train_end != 14){
    test_start <- train_end + 1
    test_end <- test_start + h - 1

    our_test_set <- train_df[test_start:test_end, ]
```

```
    medium_noise_holdout[i - start_point,] <- our_test_set$y

    current_reg <- reg_optimal_1

    current_reg_pred <- predict(current_reg, our_test_set)
    medium_noise_forecasts[i - start_point,] <- current_reg_pred
    medium_noise_RMSE[i - start_point,] <- sqrt(mean((our_test_set$y -
current_reg_pred)^2))

  } else{
    break
  }
}

complete_cases_holdout <- complete.cases(medium_noise_holdout)
medium_noise_holdout <- medium_noise_holdout[complete_cases_holdout,]
medium_noise_forecasts <- medium_noise_forecasts[complete_cases_holdout,]

complete_cases_forecasts <- complete.cases(medium_noise_forecasts)
medium_noise_holdout <- medium_noise_holdout[complete_cases_forecasts,]
medium_noise_forecasts <- medium_noise_forecasts[complete_cases_forecasts,]

plot(colMeans(abs(medium_noise_holdout - medium_noise_forecasts)),  type = "l",
     ylab = "MAE", xlab = "Horizons",
     main = "MAE of each horizon")
mean(colMeans(abs(medium_noise_holdout - medium_noise_forecasts)))
```

Appendix H – Neural Network Code

```
library(forecast)
library(TSstudio)

# Import data, pre-processing and splitting into train and validation set
data <- read.csv('cleaned_data.csv',header=T)
dates <- data$date_obj

# create ts object
ts_data <- ts(data$values_no_na, frequency = 7)

# split time series into train and validation sets (90:10 ratio)
total_obs <- length(ts_data)
split_ts <- ts_split(ts.obj = ts_data, sample.out = round(0.1*total_obs))
train_set <- split_ts$train
val_set <- split_ts$test

# verify length of each set
length(train_set)
length(val_set)
```

```
# Automatic Neural Network
ann1 <-nnetar(train_set)
print(ann1)

accnfcst<-forecast(ann1,h=14)
autoplot(accnfcst)

fcast <- forecast(ann1, PI=TRUE, h=14)
autoplot(fcast)

forecast::accuracy(forecast(ann1), val_set)[,c(2,3,5,6)]
checkresiduals(ann1)
ggtsdisplay(ann1$residuals)

# Manual Neural Networks

p = 1
P = 1
ann2 <-nnetar(train_set, p, P=P)
ann2
forecast::accuracy(forecast(ann2), val_set)[,c(2,3,5,6)]
checkresiduals(ann2)
ggtsdisplay(ann2$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann2, h=length(val_set))$mean)

p = 7
P = 1
ann3 <-nnetar(train_set, p, P=P)
ann3
forecast::accuracy(forecast(ann3), val_set)[,c(2,3,5,6)]
checkresiduals(ann3)
ggtsdisplay(ann3$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann3, h=length(val_set))$mean)

p = 7
P = 2
ann4 <-nnetar(train_set, p, P=P)
ann4
forecast::accuracy(forecast(ann4), val_set)[,c(2,3,5,6)]
checkresiduals(ann4)
ggtsdisplay(ann4$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
```

```
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann4, h=length(val_set))$mean)

p = 2
P = 2
ann5 <-nnetar(train_set, p, P=P)
ann5
forecast::accuracy(forecast(ann5), val_set)[,c(2,3,5,6)]
checkresiduals(ann5)
ggtsdisplay(ann5$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann5, h=length(val_set))$mean)

p = 7
P = 0
ann6 <-nnetar(train_set, p, P=P)
ann6
forecast::accuracy(forecast(ann6), val_set)[,c(2,3,5,6)]
checkresiduals(ann6)
ggtsdisplay(ann6$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann6, h=length(val_set))$mean)

p = 14
P = 1
ann7 <-nnetar(train_set, p, P=P)
ann7
forecast::accuracy(forecast(ann7), val_set)[,c(2,3,5,6)]
checkresiduals(ann7)
ggtsdisplay(ann7$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann7, h=length(val_set))$mean)

p = 14
P = 2
ann8 <-nnetar(train_set, p, P=P)
ann8
forecast::accuracy(forecast(ann8), val_set)[,c(2,3,5,6)]
checkresiduals(ann8)
```

```
ggtsdisplay(ann8$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann8, h=length(val_set))$mean)

p = 2
P = 3
ann9 <-nnetar(train_set, p, P=P)
ann9
forecast::accuracy(forecast(ann9), val_set)[,c(2,3,5,6)]
checkresiduals(ann9)
ggtsdisplay(ann9$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann9, h=length(val_set))$mean)

p = 2
P = 1
ann10 <-nnetar(train_set, p, P=P)
ann10
forecast::accuracy(forecast(ann10), val_set)[,c(2,3,5,6)]
checkresiduals(ann10)
ggtsdisplay(ann10$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann10, h=length(val_set))$mean)

p = 5
P = 1
ann11 <-nnetar(train_set, p, P=P)
ann11
forecast::accuracy(forecast(ann11), val_set)[,c(2,3,5,6)]
checkresiduals(ann11)
ggtsdisplay(ann11$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann11, h=length(val_set))$mean)

p = 5
P = 2
ann12 <-nnetar(train_set, p, P=P)
ann12
```

```
forecast::accuracy(forecast(ann12), val_set)[,c(2,3,5,6)]
checkresiduals(ann12)
ggtsdisplay(ann12$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann12, h=length(val_set))$mean)

p = 5
P = 3
ann13 <-nnetar(train_set, p, P=P)
ann13
forecast::accuracy(forecast(ann13), val_set)[,c(2,3,5,6)]
checkresiduals(ann13)
ggtsdisplay(ann13$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann13, h=length(val_set))$mean)

p = 5
P = 0
ann14 <-nnetar(train_set, p, P=P)
ann14
forecast::accuracy(forecast(ann14), val_set)[,c(2,3,5,6)]
checkresiduals(ann14)
ggtsdisplay(ann14$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann14, h=length(val_set))$mean)

p = 2
P = 0
ann15 <-nnetar(train_set, p, P=P)
ann15
forecast::accuracy(forecast(ann15), val_set)[,c(2,3,5,6)]
checkresiduals(ann15)
ggtsdisplay(ann15$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann15, h=length(val_set))$mean)

p = 2
P = 4
```

```
ann16 <-nnetar(train_set, p, P=P)
ann16
forecast::accuracy(forecast(ann16), val_set)[,c(2,3,5,6)]
checkresiduals(ann16)
ggtsdisplay(ann16$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann16, h=length(val_set))$mean)

p = 14
P = 3
ann17 <-nnetar(train_set, p, P=P)
ann17
forecast::accuracy(forecast(ann17), val_set)[,c(2,3,5,6)]
checkresiduals(ann17)
ggtsdisplay(ann17$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann17, h=length(val_set))$mean)

p = 21
P = 1
ann18 <-nnetar(train_set, p, P=P)
ann18
forecast::accuracy(forecast(ann18), val_set)[,c(2,3,5,6)]
checkresiduals(ann18)
ggtsdisplay(ann18$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann18, h=length(val_set))$mean)

p = 28
P = 1
ann19 <-nnetar(train_set, p, P=P)
ann19
forecast::accuracy(forecast(ann19), val_set)[,c(2,3,5,6)]
checkresiduals(ann19)
ggtsdisplay(ann19$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann19, h=length(val_set))$mean)
```

```
p = 21
P = 2
ann20 <-nnetar(train_set, p, P=P)
ann20
forecast::accuracy(forecast(ann20), val_set)[,c(2,3,5,6)]
checkresiduals(ann20)
ggtsdisplay(ann20$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann20, h=length(val_set))$mean)

p = 28
P = 2
ann21 <-nnetar(train_set, p, P=P)
ann21
forecast::accuracy(forecast(ann21), val_set)[,c(2,3,5,6)]
checkresiduals(ann21)
ggtsdisplay(ann21$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann21, h=length(val_set))$mean)

p = 1
P = 4
ann22 <-nnetar(train_set, p, P=P)
ann22
forecast::accuracy(forecast(ann22), val_set)[,c(2,3,5,6)]
checkresiduals(ann22)
ggtsdisplay(ann22$residuals)
nn_func <- function(x, h) {forecast(nnetar(x, p, P=P, h = h))}
cv_model <- tsCV(ts_data, nn_func, h = 14)
sqrt(mean(cv_model^2, na.rm=TRUE))
sqrt(mean(residuals(forecast(nnetar(ts_data, p, P=P, h = 14)))^2, na.rm=TRUE))
Metrics::smape(val_set, forecast(ann22, h=length(val_set))$mean)
```