University of Waterloo

STAT 929 — Forecasting Competition

Final Report

Dario Greco

April 2025

Scenario 1: Hydrological Forecast

We are given 576 observations at the monthly resolution of the level of a body of water. To begin we plot the time series, along with the ACF and a plot of each individual years water level:

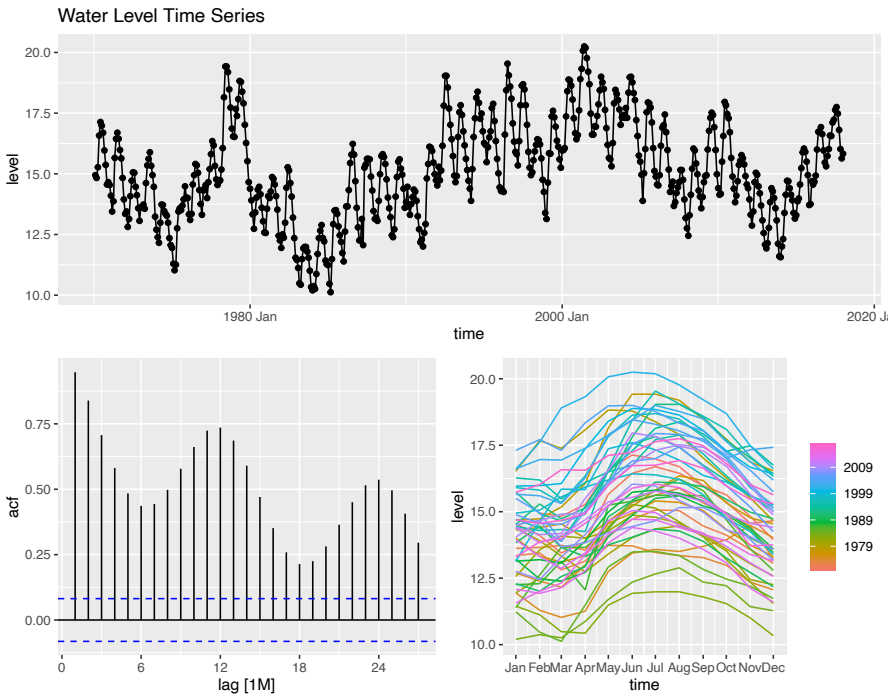

Figure 1: Time series of Water Level

It is clear from the ACF that the data is not stationary. The spikes are not exponentially decaying to 0, and we see spikes every 12 months suggesting some seasonality to the data. This likely has a natural explanation, as we would expect the level of a body of water to move with the seasons, something that is clear from the spectrum plot in the bottom right of Figure 1. We do the natural next step which is to take a look at the 12 month differenced data in hopes of reaching stationarity, as can be seen in Figure 2. The data still has signs of non-stationarity in both the time series plot and ACF. We take another monthly difference as can be seen in Figure 3, and we begin the process of fitting ARIMA models to the differenced time series.

We set a monthly difference of $d = 1$ and a seasonal difference of $D = 1$, and do a simple search through reasonable values of $p, q, P,$ and, Q given our ACF and PACF plots totaling 14 models. The models are then organized by AIC, where we do expanding window cross validation using $tsCV$ on the 6 best performing models. Table 1 shows the results of this process, where the CV error is measured with MSE.

Table 1: Top 6 ARIMA Models

Model	ARIMA_Order	AIC	BIC	CV MSE
arima13	(2,1,2)(1,1,1)[12]	630	660	60.17
arima5	(0,1,2)(0,1,1)[12]	633	650	61.65
arima6	(0,1,2)(1,1,1)[12]	634	655	61.40
arima8	(1,1,2)(1,1,1)[12]	635	661	60.80
arima12	(1,1,0)(0,1,2)[12]	635	653	60.09
arima2	(0,1,1)(0,1,1)[12]	637	650	60.08

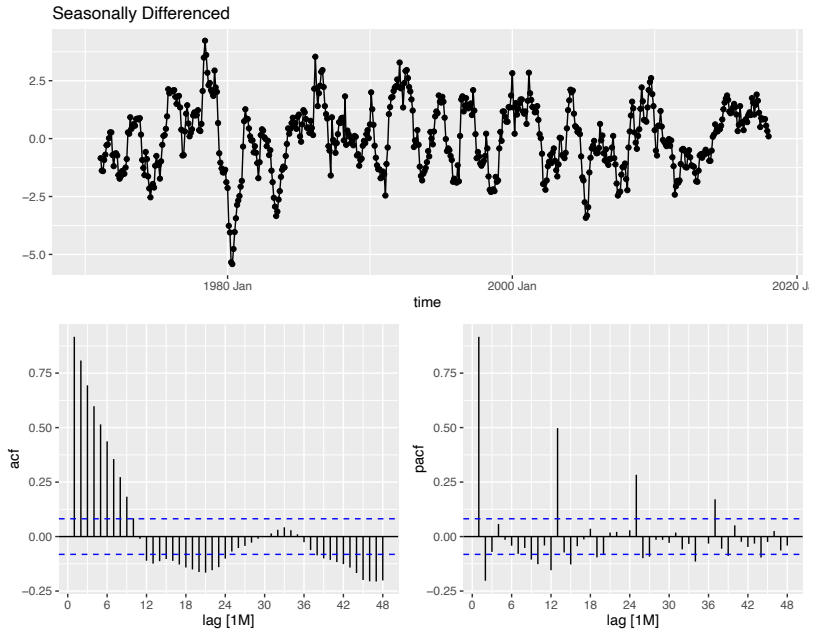

Figure 2: Seasonally Differenced Data

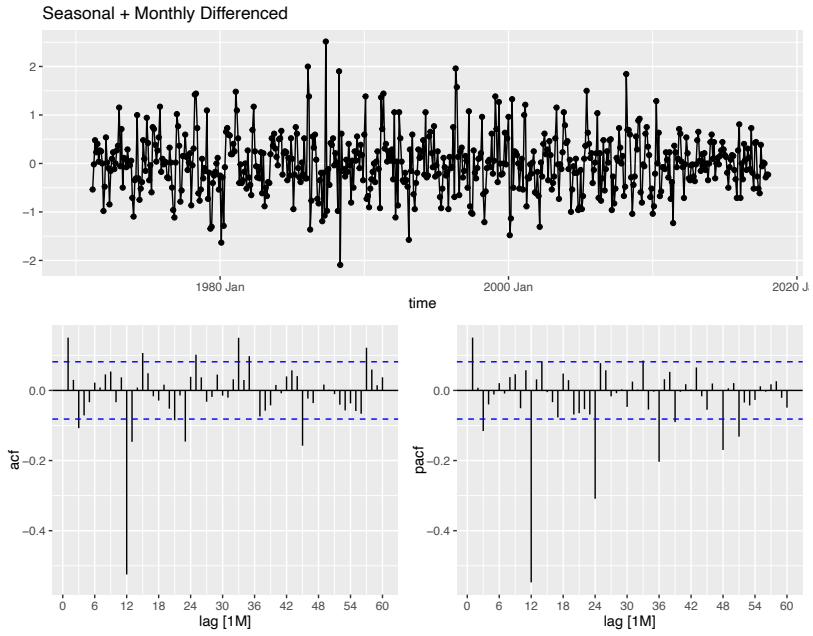

Figure 3: Seasonally and Monthly Differenced Data

As can be seen from Table 1, the models all have a similar cross validation error. The model labeled arima2 has both the lowest BIC and cross-validation error, along with a reasonably low AIC compared to the other models. We check the model residuals:

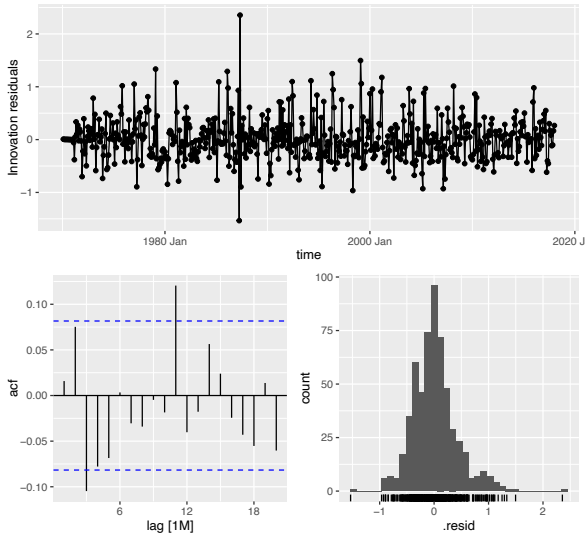

Figure 4: Time Series of Water Level

There is some outliers in the residual histogram, along with two spikes outside the 95% prediction interval, but overall the residuals look reasonable. The Ljung-Box test returns a p-value of 0.0108, rejecting the null hypothesis of normally distributed residuals at the 0.05 significance level. Compared to the other models in Table 1, these model diagnostics are comparable. We forecast 24 months ahead with the arima2 model as seen in Figure 5.

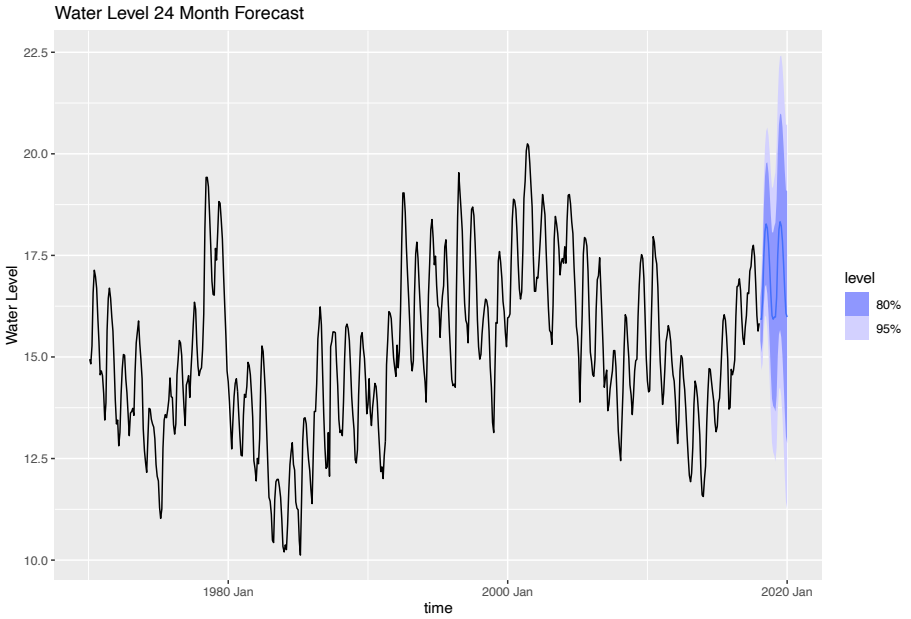

Figure 5: Water Level 24 Month Forecast

The forecast appears to capture the expected seasonality in the time series well, and the prediction intervals look reasonable.

Scenario 2: Financial Risk Forecast

We are now given 40 stocks from the New York stock exchange, each with 150 days of daily resolution. Our task is to forecast the lower 15% quantiles, Value-at-Risk, 10 steps ahead for each scenario. The error in our forecast is measured in the following way:

$$ERR = \frac{1}{400}\sum_{i=1}^{40}\sum_{j=1}^{10}\left(X_{i,150+j-\hat{q}_{i,j}}\right)\left(0.15 - \mathbf{1}\{X_{i,150+j} \leq \hat{q}_{i,j}\}\right) \tag{1}$$

As we know from Hansen (2001), and have seen in class, it is very difficult to get a substantially better result than a GARCH(1,1) when it comes to volatility models. Taking that as a given, we consider the best way to adjust the error distributions of our GARCH(1,1), and test over each of the 40 stocks in an automated manner.

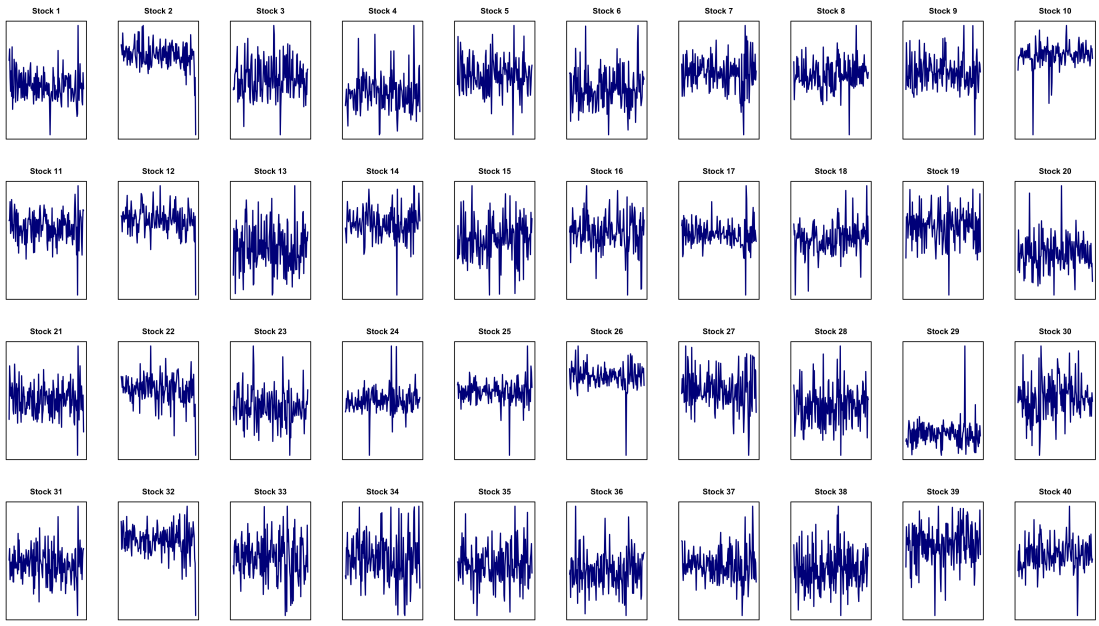

Figure 6: Time Series of Stocks

We consider three methods that cover the basic value at risk forecasting methods:

1. A standard GARCH(1,1) model where we assume the errors are normally distributed, and use the normal quantiles to forecast the VaR.
2. A standard GARCH(1,1) model where we, once again, assume the errors are normally distributed but use the empirical quantiles of the residuals in order to forecast VaR.
3. A GARCH(1,1) model where we assume the errors are from a skewed normal distribution and use the empirical quantiles of the residuals in order to forecast VaR.

In essence, we are assuming that Method 1 is a baseline, and the other two methods provide some potential improvement. For example, Method 2 relieves us of the potentially (likely) naive assumption of forecasting with normally distributed errors in financial data. Method 3 goes one step further by immediately assuming the errors are non-normal, allowing for some skew, while also freeing ourselves from any potential restriction in forecasting with a parametric distribution. For each method the VaR was calculated with:

$$\text{VaR}(0.15) = \hat{\sigma}_{t+10}\hat{F^{-1}}(0.15)$$

where $\hat{\sigma}_{t+10}$ is the ten step ahead volatility estimate, and $\hat{F}$ is the distribution of the given methods residuals (either parametric or empirical).

One more note on the choice of the skewed normal distribution. In implementing these methods I found that the "snorm" error distribution in the *ugarchspec* model in R had the best convergence over all 40 stocks. Choosing some other, more complex asymmetric distributions, often ran into some computational issues. Given that we needed to find some way to automate our model testing over all 40 stocks I decided to go with the computationally favorable "snorm" distribution.

Given our 150 days of stock return data, we begin our expanding window at day 120, as I wanted a sufficient amount of data points to make a reasonable forecast, and forecast 10 days ahead, expanding our window by one day on each step. At each step the ERR is calculated in accordance with equation 1, and then averaged over each step once the expanding window procedure is complete. The model with the smallest average ERR is taken to be our best model for that stock.

Below we have the best model for each of the 40 stock time series:

Table 2: Best GARCH(1,1) for each stock.

Stock 1	**2**	**3**	**4**
Gaussian Parametric	Gaussian Nonparametric	Gaussian Nonparametric	Gaussian Parametric
5	**6**	**7**	**8**
Gaussian Parametric	Gaussian Nonparametric	Gaussian Parametric	Gaussian Nonparametric
9	**10**	**11**	**12**
Gaussian Nonparametric	Gaussian Nonparametric	Skew-Norm Nonparametric	Gaussian Nonparametric
13	**14**	**15**	**16**
Gaussian Nonparametric	Skew-Norm Nonparametric	Skew-Norm Nonparametric	Gaussian Parametric
17	**18**	**19**	**20**
Gaussian Nonparametric	Skew-Norm Nonparametric	Gaussian Nonparametric	Gaussian Nonparametric
21	**22**	**23**	**24**
Gaussian Nonparametric	Gaussian Parametric	Gaussian Nonparametric	Gaussian Nonparametric
25	**26**	**27**	**28**
Gaussian Parametric	Skew-Norm Nonparametric	Gaussian Parametric	Gaussian Parametric
29	**30**	**31**	**32**
Skew-Norm Nonparametric	Gaussian Parametric	Gaussian Parametric	Gaussian Parametric
33	**34**	**35**	**36**
Skew-Norm Nonparametric	Gaussian Nonparametric	Gaussian Parametric	Skew-Norm Nonparametric
37	**38**	**39**	**40**
Gaussian Parametric	Gaussian Parametric	Gaussian Parametric	Gaussian Nonparametric

And we can see that the non-parametric models were the best choice majority of the time. Observe the $VaR(0.15)$ forecasts of the first two stocks, using their given models, in Figures 7 and 8. The forecast for stock 1 appears reasonable given the past, however, it is unclear how well the forecast of stock 2 will do. Clearly the stock is experiencing a sharp downturn, and the model appears to be predicting a mean reversion in the volatility. Luckily, from observing Figure 6, there does not appear to be many other cases like this.

Potential improvements in the methods implemented above are to introduce some ARMA-GARCH modeling, more sophisticated residual assumptions, or residual bootstrapping.

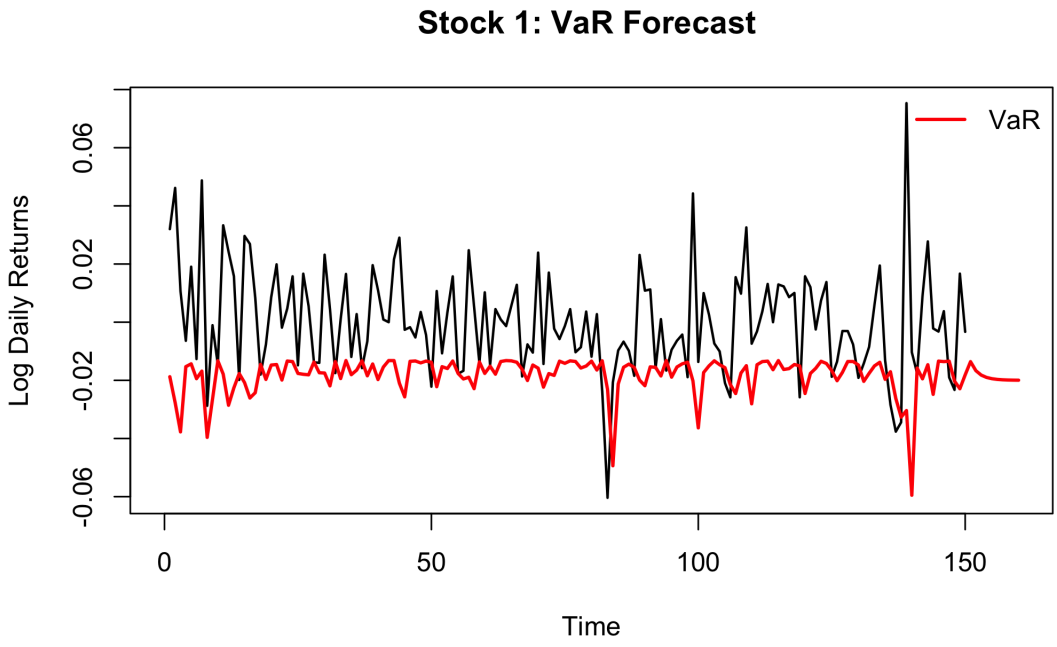

Figure 7: VaR Forecast of Stock 1

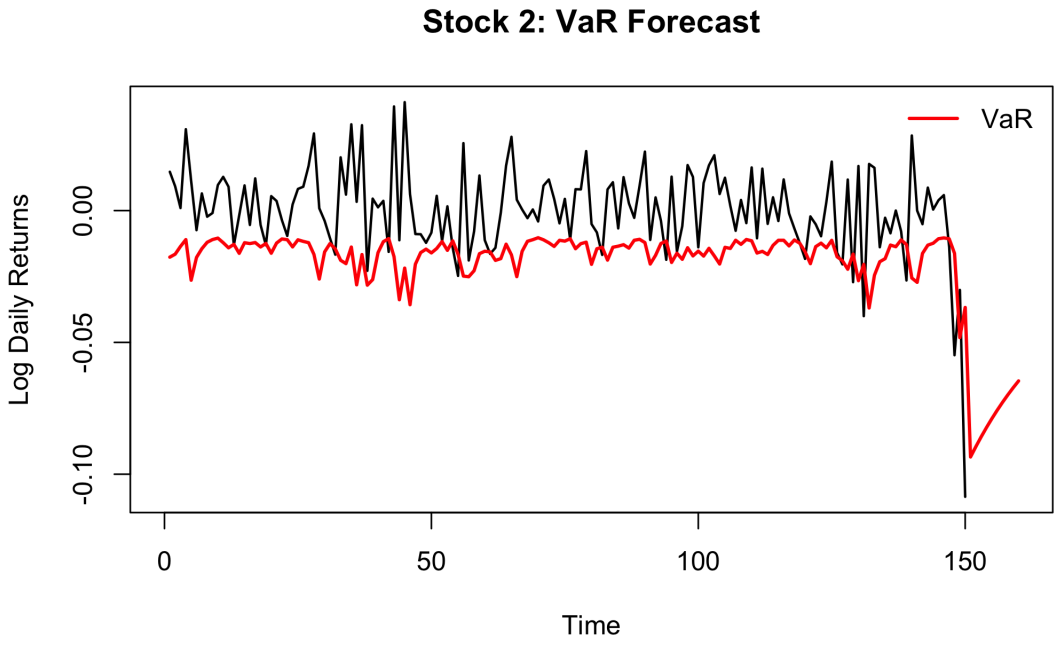

Figure 8: VaR Forecast of Stock 2

Scenario 3: Time Series Imputation

We are given the monthly resolution time series of the amount of beer produced in Australia. As can be seen from Figure 9, there is 30 missing data points in the middle of the time series that we are tasked with imputing, before we make a 24 month ahead forecast in Scenario 4. We rely on the *fpp3* and *imputeTS* package in R to aid us with both visualization and the imputation.

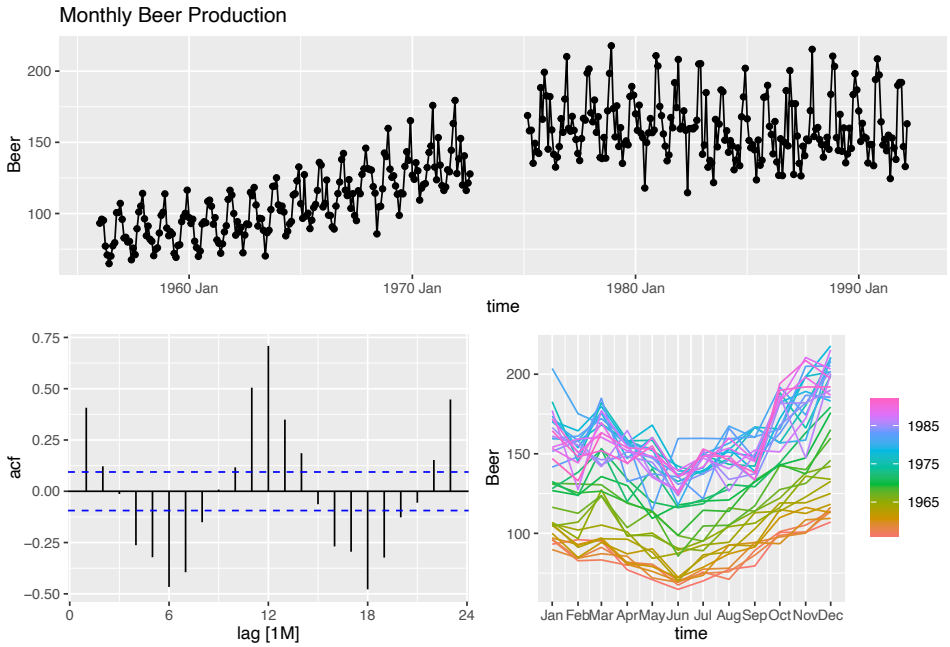

Figure 9: Monthly Beer Production Australia

It is clear from Figure 9 that there is some non-stationarity, by means of trend and seasonality, that will need to be addressed before continuing with the forecasting. Our goal is to find a suitable underlying model of the time series in order to applying Kalman Smoothing to the time series.

From the ACF and spectrum plot in Figure 9, seasonal differencing at either the 6 month or 12 month level seems reasonable, though the 12 month differencing fits our intuition of the time series more.

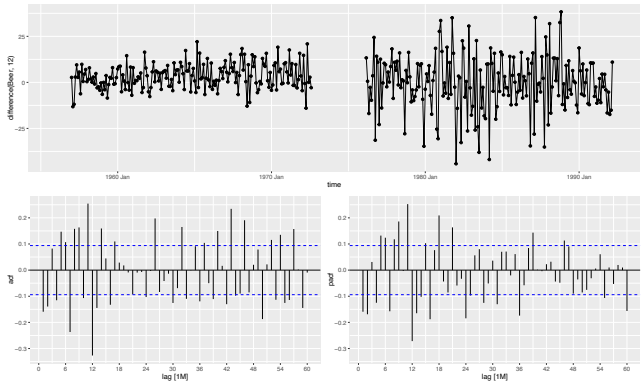

Figure 10: 12 Month Differenced Data

And from Figure 10 we can see that there is plenty of correlation that still needs to be taken care of. We

grid search for a suitable SARIMA state space model with values of p, q, P and Q ranging from 0 to 3, d and D between 0 and 1, and a seasonal difference at both the 6 month and 12 month level. The result is a test of 917 total models.

Table 3: Top SARIMA Model Candidates

Model	Order	Seasonal	AIC	LjungBox
Model 1	(2,1,3)	(0,1,1)[12]	2864.873	2.271000e–03
Model 2	(2,1,3)	(1,1,1)[12]	2866.579	1.026367e–03
Model 3	(3,1,3)	(0,1,1)[12]	2866.707	1.212096e–03
Model 4	(2,1,3)	(1,1,1)[12]	2868.519	6.795913e–04
Model 5	(2,1,3)	(1,1,2)[12]	2868.630	6.080426e–04

As can be seen from Table 3, the models with seasonal difference of 12 performed best in terms of AIC. The obvious issue is that all the models have quite low Ljung Box p-values. However, when we order the models by the largest p-value, Model 1 performs the best as well.

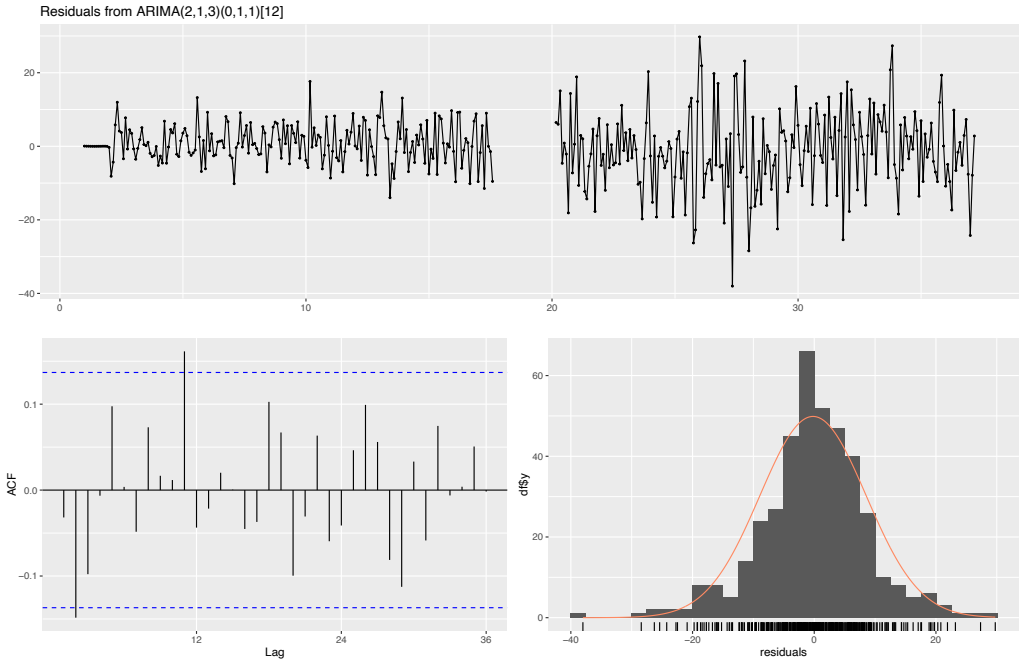

Figure 11: Model 1 Diagnostics

A check of the residual plots shows a reasonable ACF and histogram, though the increasing variance is somewhat concerning.

Considering the exhaustive search of the SARIMA parameters, we feel comfortable having Model 1 as the state space model for Kalman smoothing. We use the *kalman* function from *imptueTS* with Model 1 as the state space to obtain the imputation in Figure 12.

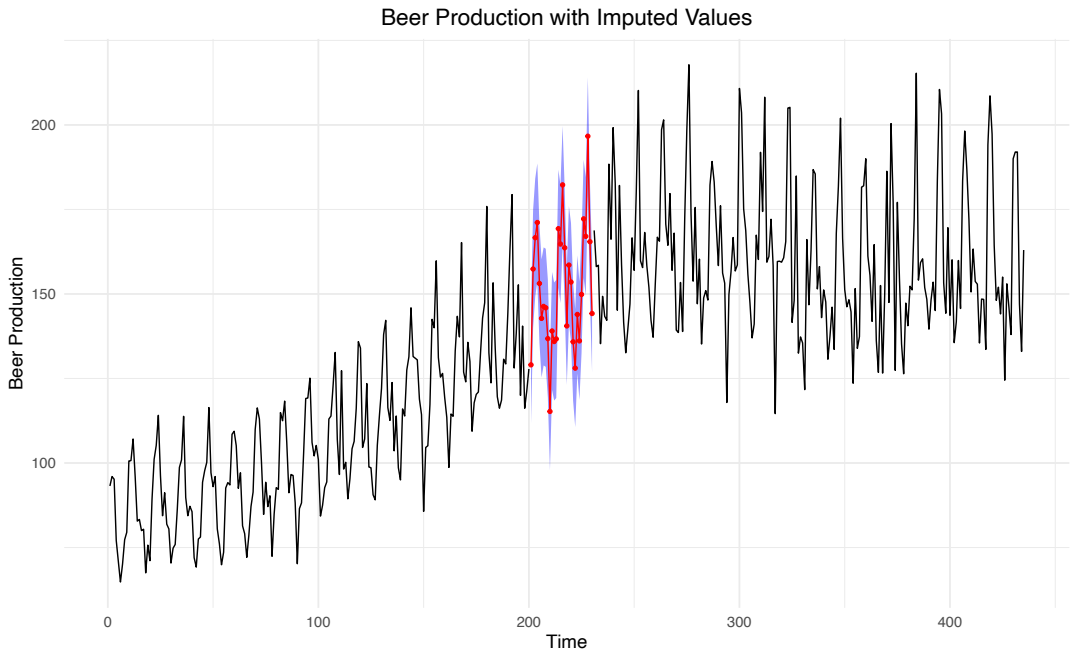

Figure 12: Kalman Smoothing with Model 1 State Space

where the 95% confidence intervals is computed from two standard deviations of the the state space model residuals.

We also test out other imputation methods available in the *imputeTS* library such as interpolation, an auto ARIMA state space for Kalman smoothing, and moving average as can be seen in Figure 13. It is clear however that none of these methods capture the obvious seasonality present in the data. As a result, we use the imputation of Kalman smoothing as seen in Figure 12 for our forecast.

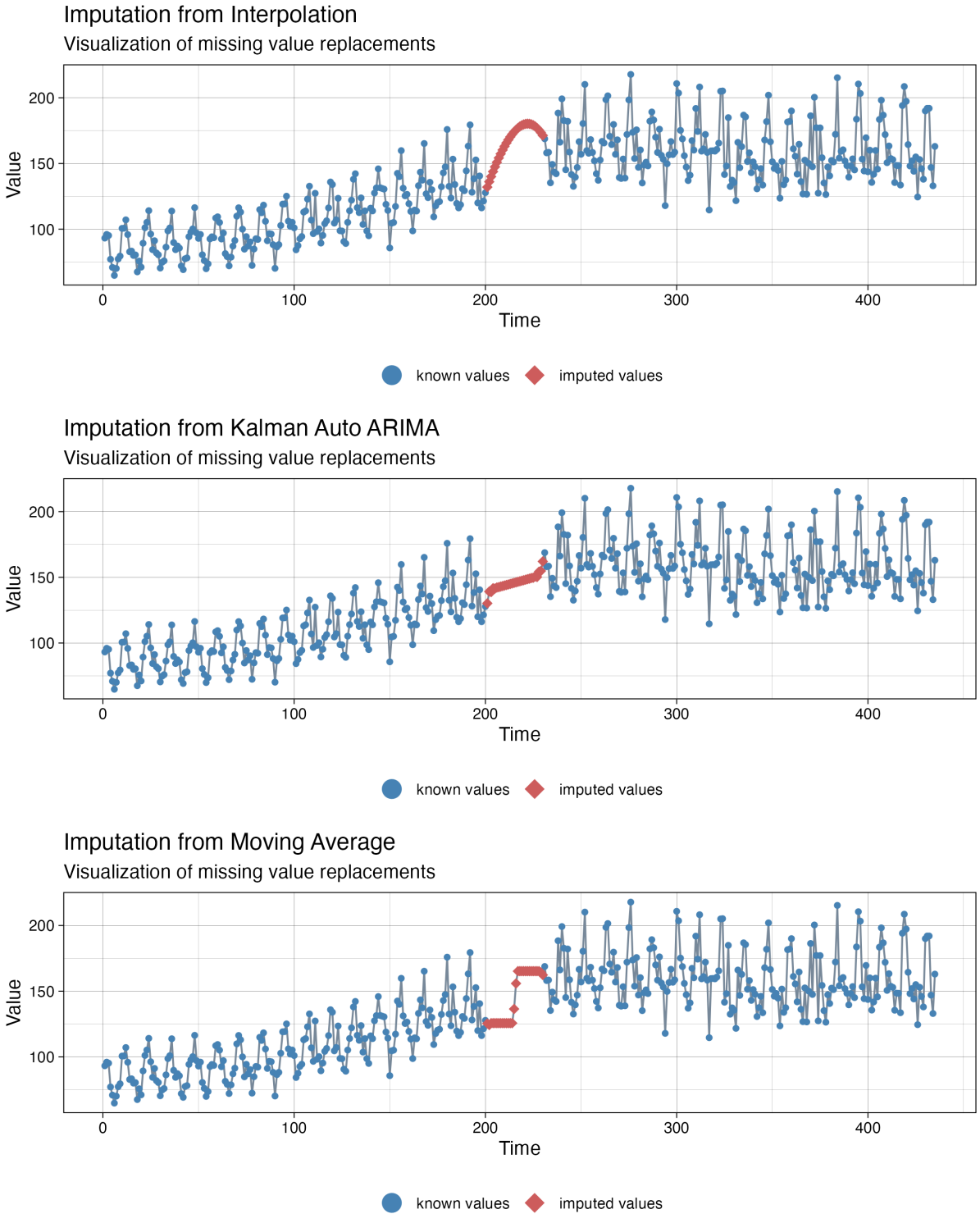

Figure 13: Other Imputation Methods

Scenario 4: Multivariate Time Series Forecast

We are now tasked with forecasting beer production in Australia 24 months ahead. We are given the four following time series, shown in Figure 14, on car production, steel production, gas consumption, and electricity consumption to use in our prediction as exogenous variables.

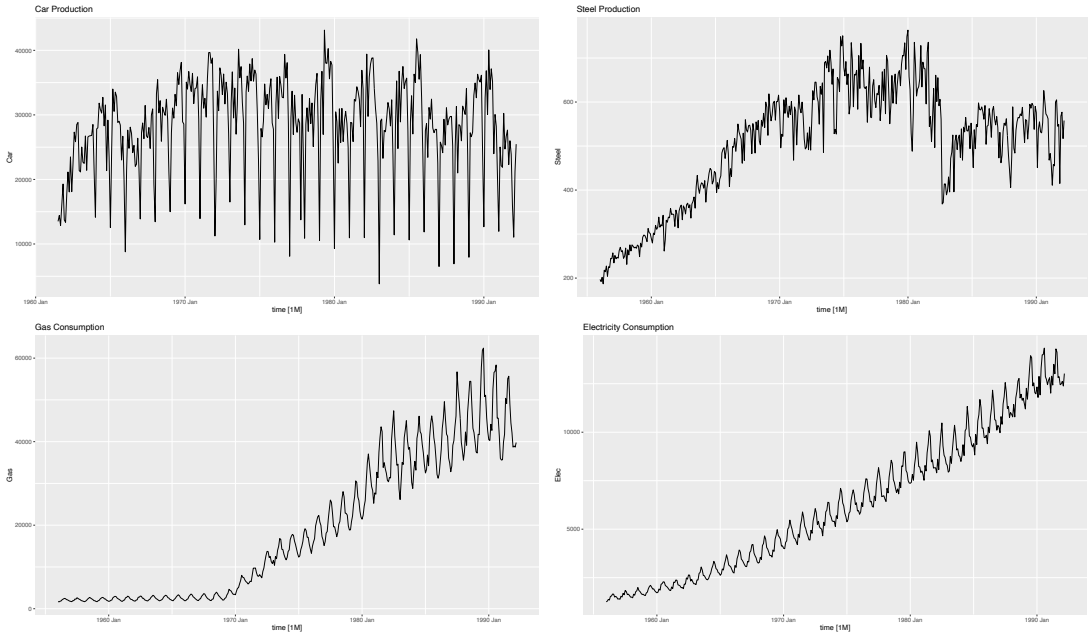

Figure 14: Car Production

With each of the steel, gas, and electricity time series plots we see the same increase in variance that we saw in the beer production time series of scenario 3. We decide to take a logarithmic transformation of each of those predictors, and the beer response, in hopes of stabilizing the variance.

We noticed that the car production time series only had 369 observations compared to 435 observations in the beer time series. Initially we only considered regression models with some combination of steel, gas, and electricity, with plans of considering car production if our models were struggling.

To begin we fit some simple linear regression models to the log of the beer time series with steel, gas, and electricity, all logged, as co-variates (assume moving forward that the response and predictors are logged).

Table 4: Linear Regression Results

Variable	**Estimate**	**Std. Error**	**t value**	**Pr(¿|t|)**
log(Intercept)	1.50551	0.16429	9.164	< 2e-16
Elec	-0.01945	0.05069	-0.384	0.701
Gas	0.11182	0.02337	4.785	2.35e-06
Steel	0.40452	0.03989	10.141	< 2e-16

From Table 4 we see that electricity is not a significant predictor. We fit another linear model of the beer time series with electricity and electricity squared as predictors and find each of the predictors to be significant. We fit an auto arima, with all the predictors in Table 4 as a baseline comparison model as seen in Figure 17

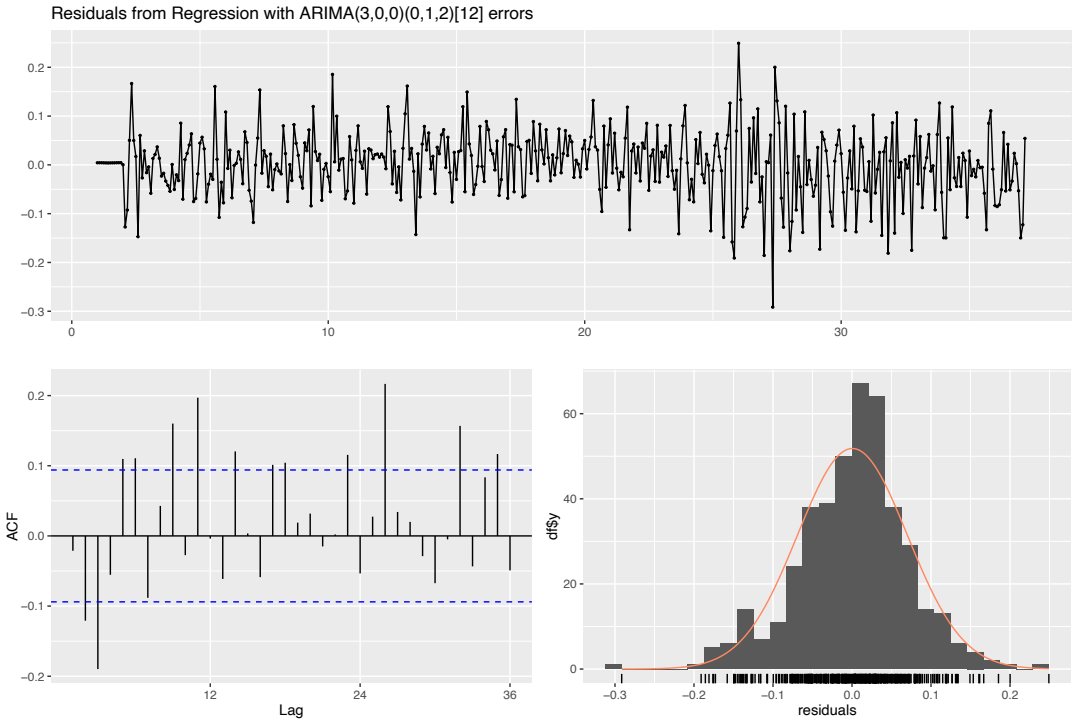

Figure 15: Auto Arima Model Diagnostics

The model diagnostics are clearly not great, with a clear correlation structure that should be estimated further. Our goal now is to find something better. We know from scenario 3, that Model 1 in Table 3 fits the data well. We now consider Model 1 with all the predictors in Table 4, each pair of predictors in the table, and one with electricity and electricity squared as predictors. In other words we begin to fit some ARMAX models on the data, using the other time series data as exogenous variables. In terms of how to forecast the exogenous variables, we simply take the auto.arima forecast as plug in values.

To evaluate the model, we forecast 24 months ahead 5 times using the expanding window cross validation method. That is, we start from training on the first $435 - 5(25)$ observations and forecast ahead 24 steps 5 times, expanding the window by 24 steps each time.

Table 5: Cross-Validated MSEs for ARIMAX Models

Model	**Regressors**	**CV MSE**
ARMAX 2	Gas, Steel	0.006415
ARMAX 1	Elec, Gas, Steel	0.006467
ARMAX 5	Elec, Gas	0.006474
ARMAX 3	Elec, Elec^2	0.006489
ARMAX 4	Steel, Elec	0.006507
Model 1 baseline	NA	0.006515

As can be seen from Table 5, none of the models really differentiate themselves in terms of MSE, though all the models that include an exogenous variable perform better than the baseline. The ARMAX 2 with Gas and Steel exogenous variables does quite well and has improved diagnostics compared to Model 1 in Table 3, as can be seen in Figure 16.

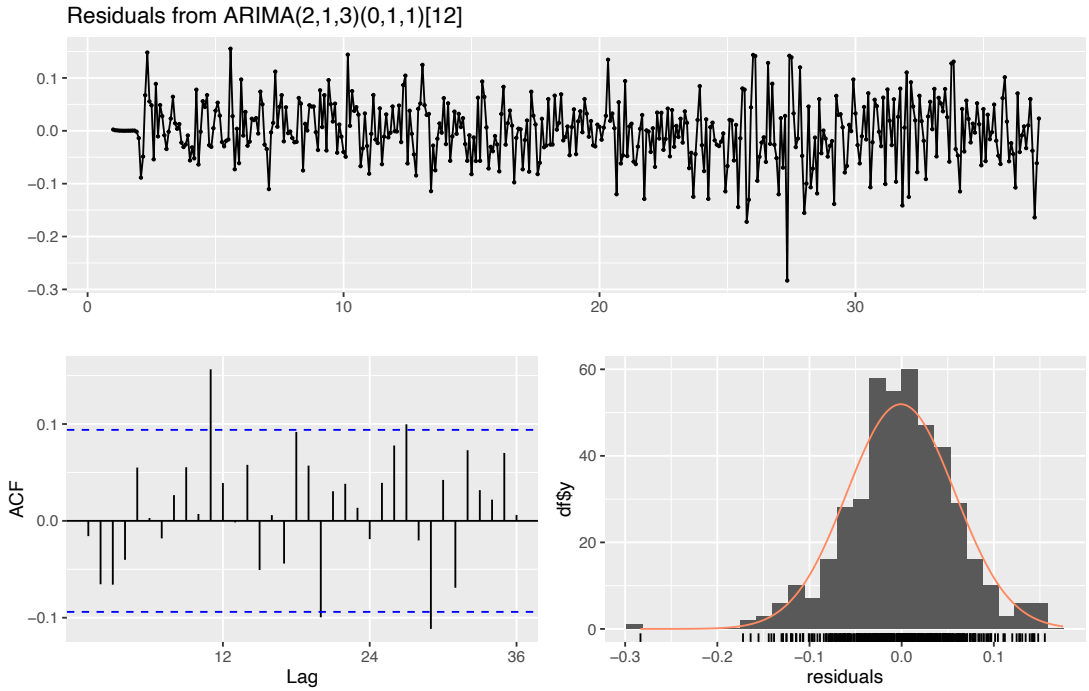

Figure 16: ARMAX Model Diagnostics

We proceed to train ARMAX 2 on all the data, forecast all its regressors using auto.arima, and forecast the entire series 24 months ahead. (Note that we had to transform the predictions back to the original time scale as we had taken a log transformation earlier)

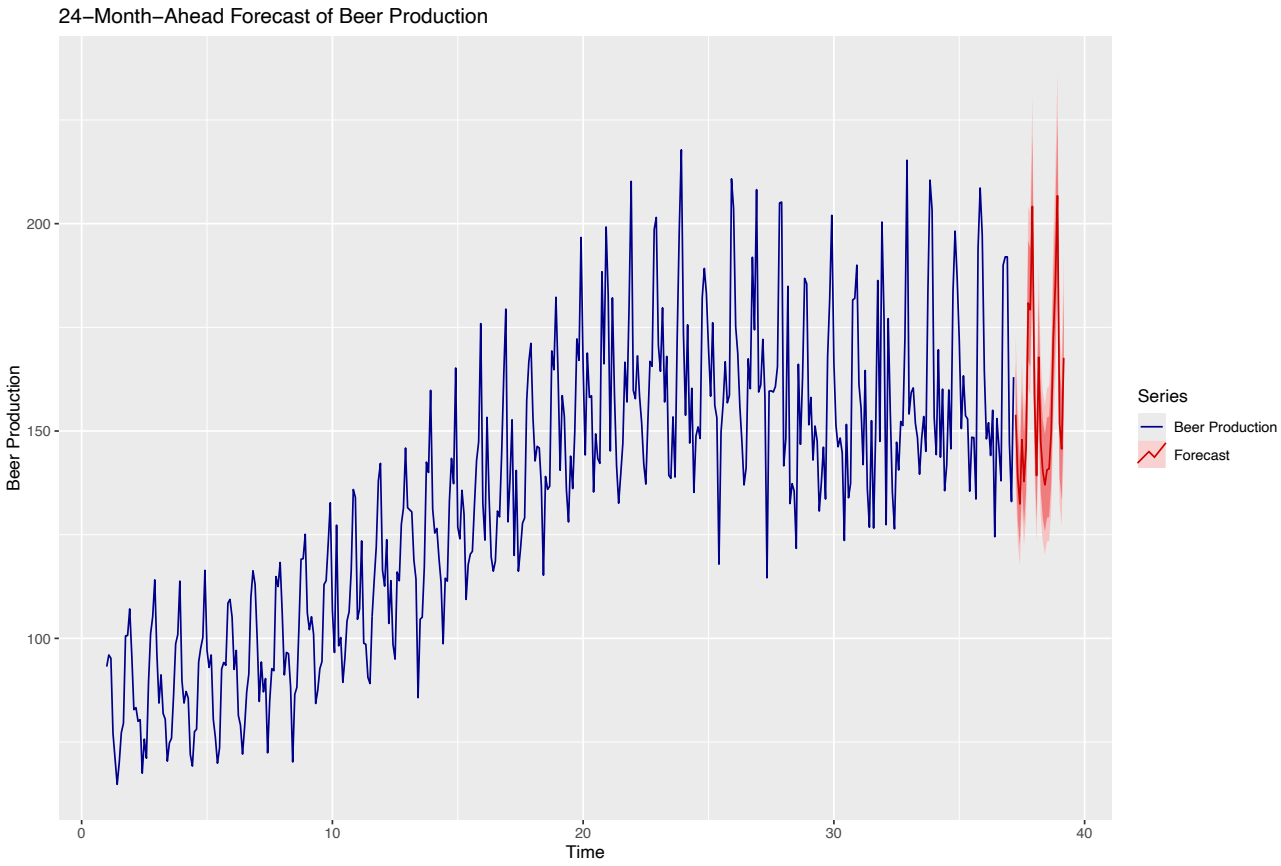

Figure 17: Beer Production Forecast

Once again the forecast looks quite reasonable, with the 95% prediction intervals being quite small. The forecast appears to capture the expected seasonality well, especially the peaks and valley in the summer and winter months.

Scenario 5: Long Range Forecasting

We are given the standardized half hourly resolution measurements of the concentration of an air pollutant in three different cities over 53 days. Our goal is to forecast one week ahead or 336 half hours. This is considered a "long-range forecast".

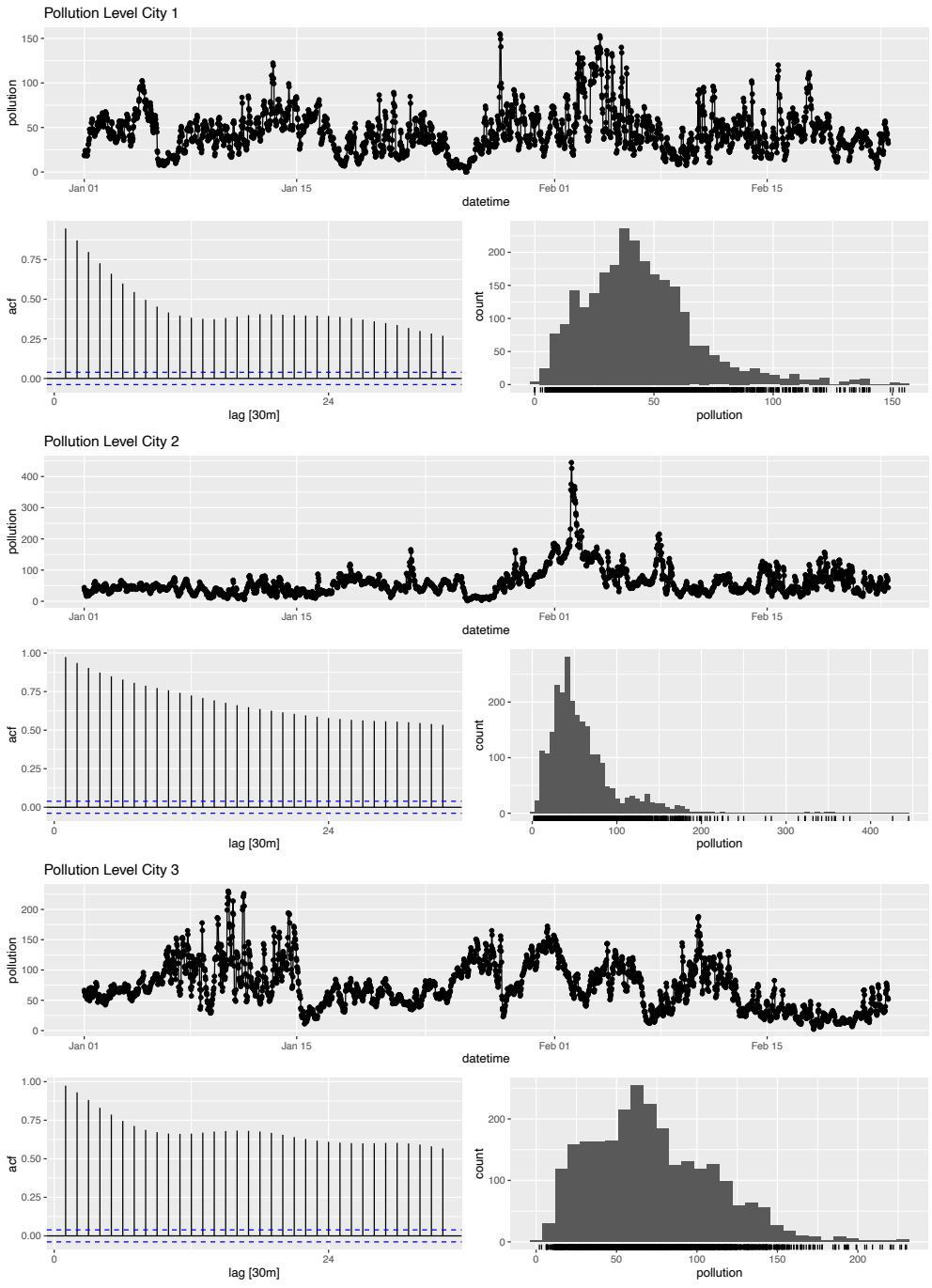

Figure 18: Pollution Level of City 1,2, and 3

As can be seen from Figure 18, none of the time series appear to be even remotely stationary.

Though it may be hard to see from the raw time series plots, it seems reasonable to assume that there is more than one level of seasonality to the data here. For example, a daily cycle for pollution seems reasonable given the daily cycle of temperature in a city. A weekly cycle also seems plausible, as, for example, people may use their cars in the city more on a weekend. To illustrate this point further we can use the *STL()* function from the *fpp3* package to decompose a given time series into its seasonality plus trend components.

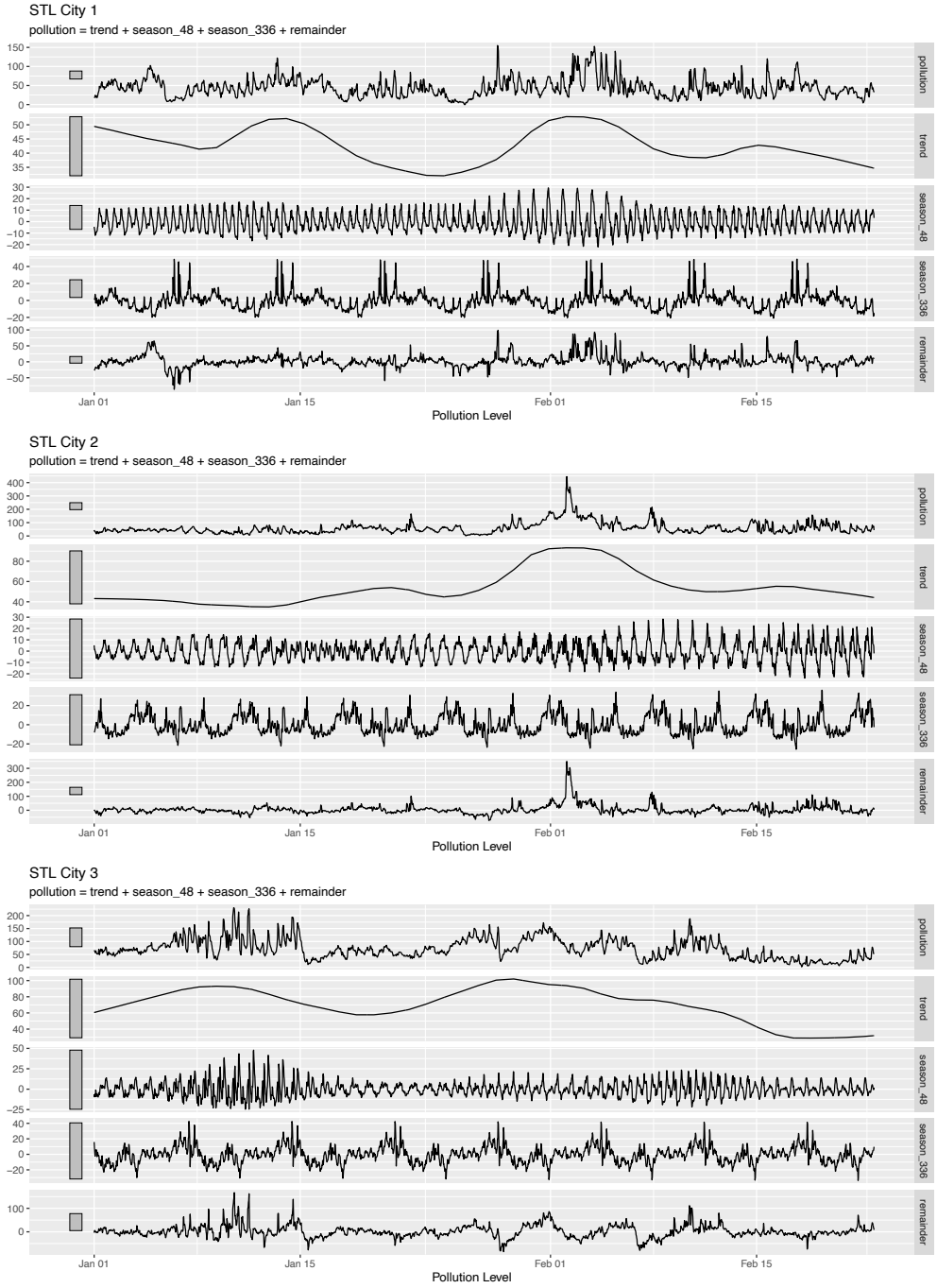

Figure 19: STL decomposition of City 1,2, and 3

For each of the time series we can see some plausible cycles at the 48 half hour (one day) and 336 half hour (one week) level, up to some changes in variance. Dynamic harmonic regression with multiple seasonal components will allow us to tackle these multiple layers of seasonality [1]. It is also questionable whether an ARMA model will be able to handle such a high level of seasonality, in the 336 step case, let alone capture multiple seasons.

Dynamic harmonic regression (DHR) allows us to specify some K Fourier terms for each seasonal component, and we model the short term errors with an ARMA model, in a similar way to the ARMAX case.

To build the dynamic harmonic regression models we specify the amount of daily and weekly Fourier terms using a grid search on each from 1 to 6, and order by AICc. So we test 36 dynamic harmonic regression models for each time series in total. We also fit an auto ARIMA to the full time series, and manually specify 5 other plausible models based off the returned parameters. For each time series we then take 3-4 top performing DHR models, 2 ARIMA models (one of them being auto ARIMA), and perform cross validation.

The cross validation involves going back 5×336 steps and forecasting a week ahead, expanding our training set by 336 steps each time for a total of 5 windows. The MSE for each model was measured for each window and averaged. The best performing model was used as the forecasting model for that time series.

Model	**RMSE**
DHR1	23.7
DHR2	23.2
DHR3	23.2
DHR4	23.5
auto_arima	24.0
manual_arima	24.1

Table 6: CV Results for City 1

Model	**RMSE**
DHR2	49.3
DHR1	46.6
DHR2	48.7
auto_arima	45.5
manual_arima	48.8

Table 7: CV Results for City 2

Model	**RMSE**
DHR1	33.5
DHR2	30.1
DHR3	33.2
auto_arima	35.8
manual_arima	36.0

Table 8: CV Results for City 3

The best performing model for city 1 was a DHR model with 3 daily Fourier terms, 1 weekly Fourier term, and trend component. For city 2 the auto ARIMA was the best performing model. The DHR models for city 3 performed much better than the ARIMA models, with the best model having 3 daily Fourier terms, 1 weekly Fourier term, and no trend.

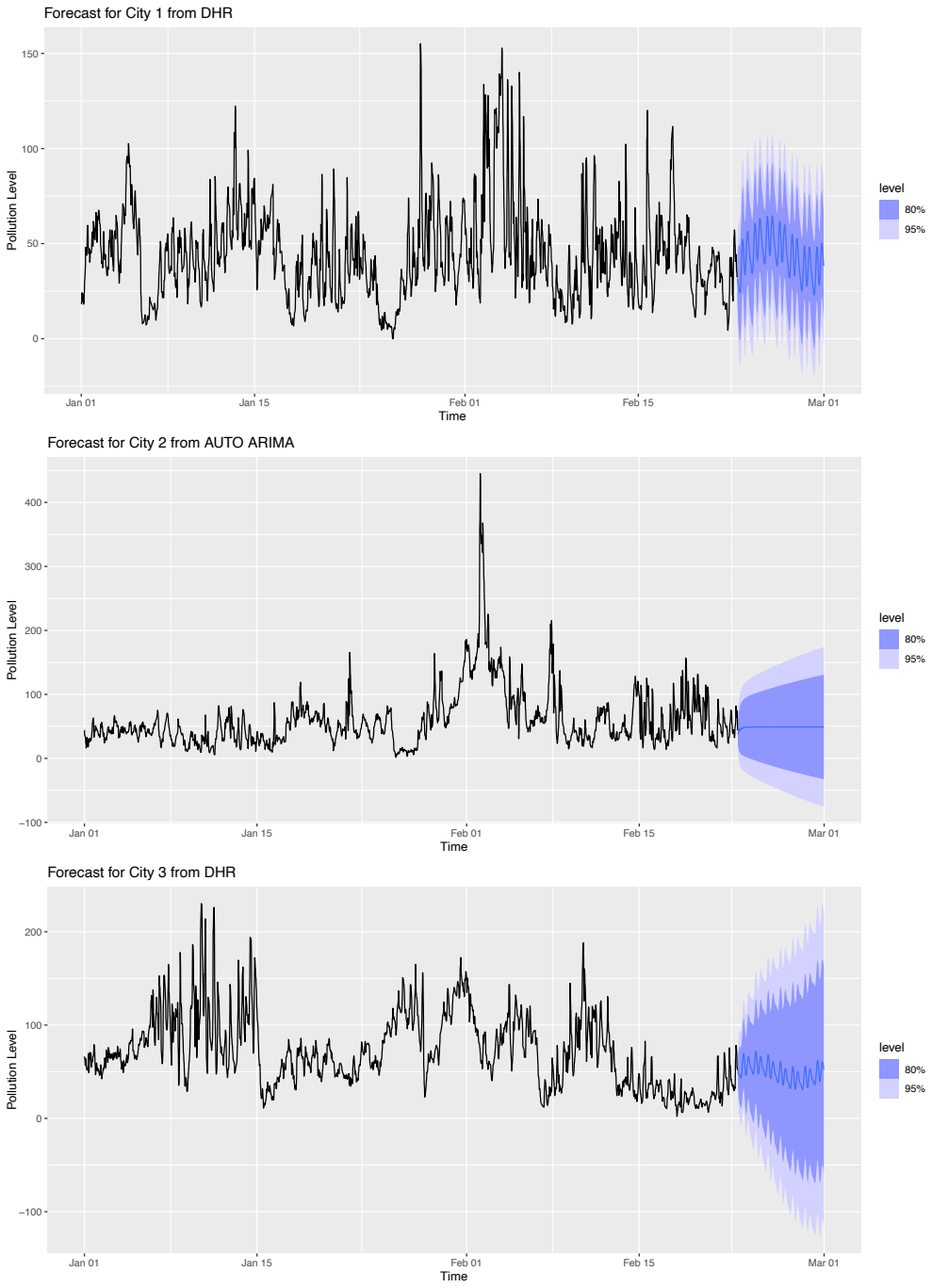

Figure 20: Pollution Forecasts for City 1,2, and 3

The forecast for both city 1 and city 3 are capturing some form of seasonality that we would expect. Both seem "plausible" and pass the eye test, but if our predicted cycles ever get mis-aligned with the underlying cycles, our test error will be poor. City 2 has, at first glance, a less impressive forecast. The forecast fluctuates slightly at the beginning but then reverts to a horizontal line. Given the complex seasonality of this forecast however, it is may not be unwise to use such a forecast.

Another thing to note is how much larger the prediction intervals are for city 3 compared to the other two. The prediction intervals are getting so wide that they dip down into negative values of pollution. This also

applies to city 2. On the other hand, the city 1 prediction intervals are quite tight and only briefly dip below 0.

We used methods described by Hyndman and Athanasopoulos. Particularly chapter 9.9, 10.5, and 12.1 [2], and support tools from the `fpp3` R package [1]. The course textbook was also a valuable reference. [3].

References

[1] Rob J. Hyndman. *fpp3: Data for "Forecasting: Principles and Practice" (3rd Edition)*. R package version 1.0.1.9000. 2025. URL: https://github.com/robjhyndman/fpp3.

[2] Rob J. Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. 3rd. OTexts, 2021. URL: https://otexts.com/fpp3/.

[3] Shumway. *Time Series Analysis*. 2023. URL: http://www.stat.ucla.edu/~frederic/415/S23/tsa4.pdf.

Appendix A: R Code for Scenario 1

The following R script was used to fit the models for Scenario 1.

```
library(astsa)
library(fpp3)
library(tidyverse)
library(forecast)

# READ: Code was inspired by Hyndman fpp ed 3 section 9.9.
# I thought his code style was nice

# Read and prep the data
library(readr)
hydro_df <- read_csv("Data/HydroData/hyd_post.txt")
hydro_ts = ts(hydro_df[,2])

# Exploratory Data Analysis ----------------------------------------------

# look at time series plot and corresponding acf
# Notice the seasonality
plot(hydro_df, type = "l", xlab = "time", ylab = "level")
acf(hydro_df[,2], lag = 48)

hydro_tsibble <- hydro_df %>%
  rename(time = 1, level = 2) %>%
  mutate(time = yearmonth(time)) %>%  # sets the default to 1970. For the modeling
      below
  as_tsibble(index = time)

# Here is a nice version of the plot using the tsibble
hydro_tsibble %>% gg_tsdisplay() + labs(title = "Water Level Time Series")


# We take a 12 month difference and we can see the natural peaks in PACF at 12
hydro_tsibble %>% gg_tsdisplay(difference(level, 12),
                               plot_type = "partial", lag_max = 48) +
                  labs(title = "Seasonally Differenced") +
                  ylab("")

# Data is still non-stationary so we take the first difference
hydro_tsibble %>% gg_tsdisplay(difference(level, 12) %>% difference(1),
                               plot_type = "partial", lag_max = 60) +
                  labs(title = "Seasonal + Monthly Differenced") +
```

```
                  ylab("")

# SARIMA Model Fitting ------------------------------------------------------

# Fit any reasonable model
fit <- hydro_tsibble %>% model(
  arima1  = ARIMA(level ~ pdq(1,1,0) + PDQ(1,1,0)),
  arima2  = ARIMA(level ~ pdq(0,1,1) + PDQ(0,1,1)),
  arima3  = ARIMA(level ~ pdq(1,1,1) + PDQ(1,1,1)),
  arima4  = ARIMA(level ~ pdq(2,1,0) + PDQ(1,1,0)),
  arima5  = ARIMA(level ~ pdq(0,1,2) + PDQ(0,1,1)),
  arima6 =  ARIMA(level ~ pdq(0,1,2) + PDQ(1,1,1)),
  arima7  = ARIMA(level ~ pdq(2,1,1) + PDQ(1,1,1)),
  arima8  = ARIMA(level ~ pdq(1,1,2) + PDQ(1,1,1)),
  arima9  = ARIMA(level ~ pdq(1,1,1) + PDQ(2,1,0)),
  arima10  = ARIMA(level ~ pdq(1,1,1) + PDQ(0,1,2)),
  arima11 = ARIMA(level ~ pdq(0,1,1) + PDQ(2,1,1)),
  arima12 = ARIMA(level ~ pdq(1,1,0) + PDQ(0,1,2)),
  arima13 = ARIMA(level ~ pdq(2,1,2) + PDQ(1,1,1)),
  arima14 = ARIMA(level ~ 1 + pdq(2, 1, 1) + PDQ(4,1,1),
                  order_constraint = p + q + P + Q <= 10 & (constant + d + D <=
                      10)) # for fun
  #auto = ARIMA(level, stepwise = FALSE, approx = FALSE)
)

fit %>%  pivot_longer(everything(), names_to = "Model name",
                      values_to = "Orders")
glance(fit) %>%  arrange(AICc) %>%  select(.model:BIC)

# Check individual residual plots
fit %>%  select(arima13) %>%  gg_tsresiduals(lag=20)
fit %>%  select(arima5) %>%  gg_tsresiduals(lag=20)
fit %>%  select(arima6) %>%  gg_tsresiduals(lag=20)
fit %>%  select(arima8) %>%  gg_tsresiduals(lag=20)
fit %>%  select(arima2) %>%  gg_tsresiduals(lag=20)

# Check normality of innovations
augment(fit) %>%
  filter(.model == "arima13")  %>%
  features(.innov, ljung_box, lag = 24, dof = 6)

augment(fit) %>%
  filter(.model == "arima5")  %>%
  features(.innov, ljung_box, lag = 24, dof = 3)

augment(fit) %>%
  filter(.model == "arima6")  %>%
  features(.innov, ljung_box, lag = 24, dof = 4)

augment(fit) %>%
  filter(.model == "arima8") %>%
  features(.innov, ljung_box, lag = 24, dof = 5)

augment(fit) %>%
  filter(.model == "arima2") %>%
  features(.innov, ljung_box, lag = 24, dof = 2)

# Exponential Fitting -------------------------------------------------------
```

```
# Fit the ETS model
ets_model <- hydro_tsibble %>%
  model(ETS(level))

# View the model summary
report(ets_model)

ets_model |>
  gg_tsresiduals(lag_max = 16)

# Very bad results
augment(ets_model) |>
  features(.innov, ljung_box, lag = 16)

# Cross Validation  ------------------------------------------------------

# I want to use the tsCV function from the forecast library so I need to make
    functions corresponding to my models in question
# It takes quite a long time to run the cv values so we test only the top six
    models
library(tictoc)

tic("Running the SARIMA CV")
forecast_hydro_A <- function(x, h) {forecast(Arima(x, order = c(2, 1, 2), seasonal
    = list(order = c(1, 1, 1), period = 12)), h=h)}
cv_errors_A <- tsCV(hydro_ts, forecast_hydro_A, h = 24)
MSE_CV_A <- sum(apply(cv_errors_A, 2, function(x) mean(x^2, na.rm = TRUE)))

forecast_hydro_B <- function(x, h) {forecast(Arima(x, order = c(0, 1, 2), seasonal
    = list(order = c(0, 1, 1), period = 12)), h=h)}
cv_errors_B <- tsCV(hydro_ts, forecast_hydro_B, h = 24)
MSE_CV_B <- sum(apply(cv_errors_B, 2, function(x) mean(x^2, na.rm = TRUE)))

forecast_hydro_C <- function(x, h) {forecast(Arima(x, order = c(1, 1, 2), seasonal
    = list(order = c(1, 1, 1), period = 12)), h=h)}
cv_errors_C <- tsCV(hydro_ts, forecast_hydro_C, h = 24)
MSE_CV_C <- sum(apply(cv_errors_C, 2, function(x) mean(x^2, na.rm = TRUE)))

forecast_hydro_D <- function(x, h) {forecast(Arima(x, order = c(0, 1, 2), seasonal
    = list(order = c(1, 1, 1), period = 12)), h=h)}
cv_errors_D <- tsCV(hydro_ts, forecast_hydro_D, h = 24)
MSE_CV_D <- sum(apply(cv_errors_D, 2, function(x) mean(x^2, na.rm = TRUE)))

forecast_hydro_E <- function(x, h) {forecast(Arima(x, order = c(0, 1, 1), seasonal
    = list(order = c(0, 1, 1), period = 12)), h=h)}
cv_errors_E <- tsCV(hydro_ts, forecast_hydro_E, h = 24)
MSE_CV_E <- sum(apply(cv_errors_E, 2, function(x) mean(x^2, na.rm = TRUE)))

forecast_hydro_F <- function(x, h) {forecast(Arima(x, order = c(1, 1, 0), seasonal
    = list(order = c(0, 1, 2), period = 12)), h=h)}
cv_errors_F <- tsCV(hydro_ts, forecast_hydro_F, h = 24)
MSE_CV_F <- sum(apply(cv_errors_F, 2, function(x) mean(x^2, na.rm = TRUE)))

toc()

# Final Forecast ----------------------------------------------------------
# Note: The time in this plot is made up and was set above in tsibble format

forecast(fit, h=24) %>%
```

```
  filter(.model=='arima2') %>%
  autoplot(hydro_tsibble) +
  labs(title = "Water Level 24 Month Forecast",
       y="Water Level")

forecast_senario_1 = forecast(fit, h=24) %>%
  filter(.model=='arima2')
```

Appendix B: R Code for Scenario 2

The following R script was used to fit the models for Scenario 2.

```
library(rugarch)
library(tidyverse)
library(sn) # for skew normal error dist
library(astsa)

# Load in Data
fold_path <- "Data/Stock/"
stock_list_ts <- vector("list", 40)

# Loop over stock1.txt to stock40.txt
for (i in 1:40) {
  file_name <- paste0(fold_path, "stock", i, ".txt")
   stock <- read.csv(file_name)
   ts_data <- ts(stock[,2])
  stock_list_ts[[i]] <- ts_data
}

# Plot all the stocks
#for(i in 1:40){
  #tsplot(stock_list_ts[[i]])
#}

# All Stocks ------------------------------------------------------------------
# stock_list_ts is a list of all the stocks as a time series object
best_models_vec <- vector("character", length(stock_list_ts))

for(k in 1:40){

 # Pull one stock to test method
  one_stock <- stock_list_ts[[k]]

  # Trying to minimize this error
  calc_ERR <- function(actual, forecast, alpha = 0.15) {
    mean((actual - forecast) * (alpha - (actual <= forecast)))
  }

  # The models we will use. Standard Garch (1,1) with different error
      distributions.
  garch_spec_gauss <- ugarchspec(
    variance.model = list(model = "sGARCH", garchOrder = c(1,1)),
    mean.model = list(armaOrder = c(0,0)),
    distribution.model = "norm"
  )

  garch_spec_snorm <- ugarchspec(
    variance.model = list(model = "sGARCH", garchOrder = c(1,1)),
```

```
  mean.model = list(armaOrder = c(0,0)),
  distribution.model = "snorm"
)

# We will fit an expanding window approach starting with 120 observations
window_exp = seq(120, 140, 1)
n.ahead = 10
alpha = 0.15

niter = length(window_exp)

ERR_gauss <- numeric()
ERR_gauss_np <- numeric()
ERR_snorm_np <- numeric()
for(i in 1:niter){
  # define our expanding window
  train_size <- window_exp[i]
  train_data <- one_stock[1:train_size] # expanding window approach
  test_data <- one_stock[(train_size + 1):(train_size + 10)] # We want to test
      on ten steps ahead

  # Fit some models
  garch_fit_gauss <- ugarchfit(spec = garch_spec_gauss, data = train_data)
  garch_fit_snorm <- ugarchfit(spec = garch_spec_snorm, data = train_data)

  # Forecast
  garch_fc_gauss <- ugarchforecast(garch_fit_gauss, n.ahead = 10)
  garch_fc_snorm <- ugarchforecast(garch_fit_snorm, n.ahead = 10)

  # Get the estimated vol
  vol_fc_gauss <- sigma(garch_fc_gauss)
  vol_fc_snorm <- sigma(garch_fc_snorm)

  # VaR approaches

  # standard parametric for gauss
  VaR_Gauss_para <- qnorm(alpha)*vol_fc_gauss

  # non-parametric gauss
  gauss_resid <- residuals(garch_fit_gauss, standardize = TRUE)
  VaR_Gauss_non_para <- quantile(gauss_resid, probs = 0.15)*vol_fc_gauss

  # non-parametric snorm
  snorm_resid <- residuals(garch_fit_snorm, standardize = TRUE)
  VaR_snorm_non_para <- quantile(snorm_resid, probs = 0.15)*vol_fc_snorm

  # Get the ERR
  ERR_gauss[i] <- calc_ERR(test_data, VaR_Gauss_para)
  ERR_gauss_np[i] <- calc_ERR(test_data, VaR_Gauss_non_para)
  ERR_snorm_np[i] <- calc_ERR(test_data,VaR_snorm_non_para )

}

mean_ERR_gauss     <- mean(ERR_gauss, na.rm = TRUE)
mean_ERR_gauss_np  <- mean(ERR_gauss_np, na.rm = TRUE)
mean_ERR_snorm_np  <- mean(ERR_snorm_np, na.rm = TRUE)

# Combine into a named vector
ERR_summary <- c(
```

```
    "GARCH(1,1) Gaussian Parametric" = mean_ERR_gauss,
    "GARCH(1,1) Gaussian Nonparametric" = mean_ERR_gauss_np,
    "GARCH(1,1) Skew-Norm Nonparametric" = mean_ERR_snorm_np
  )

  # Find the best model (lowest ERR)
  best_model <- names(ERR_summary)[which.min(ERR_summary)]
  cat("Best model based on lowest ERR:", best_model)
  best_models_vec[k] <- best_model

}

# GET FORECASTS ------------------------------------------------------------

# we want to take our best model and train it on the full dataset for the 10 ahead

VaR_forecasts <- list()

# Loop through each the "best model" stock list and fit the best model accoridngly
.
for(k in 1:40) {

  one_stock <- stock_list_ts[[k]]
  best_model <- best_models_vec[k]
  if (best_model == "GARCH(1,1) Gaussian Parametric") {

    garch_fit <- ugarchfit(spec = garch_spec_gauss, data = one_stock)
    garch_fc <- ugarchforecast(garch_fit, n.ahead = 10)

    sigma_fc <- sigma(garch_fc)
    VaR_forecast <- qnorm(alpha) * sigma_fc

  } else if (best_model == "GARCH(1,1) Gaussian Nonparametric") {

    garch_fit <- ugarchfit(spec = garch_spec_gauss, data = one_stock)
    garch_fc <- ugarchforecast(garch_fit, n.ahead = 10)

    sigma_fc <- sigma(garch_fc)

    gauss_resid <- residuals(garch_fit, standardize = TRUE)
    q_resid <- quantile(gauss_resid, probs = alpha)

    VaR_forecast <- q_resid * sigma_fc

  } else if (best_model == "GARCH(1,1) Skew-Norm Nonparametric") {

    garch_fit <- ugarchfit(spec = garch_spec_snorm, data = one_stock)
    garch_fc <- ugarchforecast(garch_fit, n.ahead = 10)

    sigma_fc <- sigma(garch_fc)

    snorm_resid <- residuals(garch_fit, standardize = TRUE)
    q_resid <- quantile(snorm_resid, probs = alpha)

    VaR_forecast <- q_resid * sigma_fc
  }

  # Store forecast for stock k
  VaR_forecasts[[k]] <- VaR_forecast
```

```
}



# Forecast Plotting + Other Viz
    ------------------------------------------------------------------

# get stock 1 fc
VaR_for_stock1 <- as.vector(VaR_forecasts[[1]])
time1 <- seq(151, 160, 1)

# plot it
plot(stock_list_ts[[1]], ylab = "Log Daily Returns Stock 1",
     xlab = "Time",
     main = "VaR(0.15) Forecast for Stock 1", xlim = c(0, 160))
lines(time1, VaR_for_stock1, col = "red")


# Add the historical vol for better presentation
garch_fit_1 <- ugarchfit(spec = garch_spec_gauss, data = stock_list_ts
    [[1]][1:150])
sigma_estimates_1 <- sigma(garch_fit_1)
historical_VaR_1 <- qnorm(alpha) * sigma_estimates_1  # parametric quantile


VaR_series_1 <- c(as.numeric(historical_VaR_1), as.numeric(VaR_forecasts[[1]]))


# Plot
plot(stock_list_ts[[1]], type = "l", col = "black", lwd = 1.5,
     xlab = "Time", ylab = "Log Daily Returns", main = "Stock 1: VaR Forecast",
     xlim = c(0, 160))

lines(1:160, VaR_series_1, col = "red", lwd = 2)
legend("topright", legend = c("VaR"),
       col = c( "red"), lwd = 2, bty = "n")

# stock 2 fc
VaR_for_stock2 <- as.vector(VaR_forecasts[[2]])
time2 <- seq(151, 160, 1)

# plot it
plot(stock_list_ts[[2]], ylab = "Log Daily Returns Stock 2",
     xlab = "Time",
     main = "VaR(0.15) Forecast for Stock 2", xlim = c(0, 160))
lines(time2, VaR_for_stock2, col = "red")

# Add the historical vol
garch_fit_2 <- ugarchfit(spec = garch_spec_gauss, data = stock_list_ts
    [[2]][1:150])
sigma_estimates_2 <- sigma(garch_fit_2)
resid_2 <- residuals(garch_fit_2, standardize = TRUE)
q_resid_2 <- quantile(resid_2, probs = alpha)
historical_VaR_2 <- q_resid_2 * sigma_estimates_2


VaR_series_2 <- c(as.numeric(historical_VaR_2), as.numeric(VaR_forecasts[[2]]))
```

Appendix C: R Code for Scenario 3

The following R script was used to fit the models for Scenario 3.

```
library(astsa)
library(fpp3)
library(tidyverse)
library(forecast)
library(imputeTS)

prod_targ = read.csv("Data/ProdData/prod_target.txt")
prod_target_ts = ts(prod_targ[,2])
prod_targ <- prod_targ %>%
  select(V1, V2) %>%
  rename(time = V1, Beer = V2) %>%
  mutate(time = yearmonth(time)) %>% # This is a random time.
  as_tsibble(index = time)

# Look at this plot. What an amazing package
prod_targ %>% gg_tsdisplay() + labs(title = "Monthly Beer Production")

prod_targ %>% gg_tsdisplay(difference(Beer, 6) ,
                           plot_type = c("partial"), lag_max = 60)
prod_targ %>% gg_tsdisplay(difference(Beer, 12) ,
                           plot_type = c("partial"), lag_max = 60)


beer_ts <- ts(prod_targ$Beer, frequency = 12)



# Model Fitting Algo for State Space ------------------------------------

results <- list()
model_num <- 1

# Grid search parameters
p_vals <- 0:3
d_vals <- 0:1
q_vals <- 0:3
P_vals <- 0:3
Q_vals <- 0:3
D <- 1
seasonal_periods <- c(6, 12)

# Grid search
for (s in seasonal_periods) {
  for (d in d_vals) {
    for (p in p_vals) {
      for (q in q_vals) {
        for (P in P_vals) {
          for (Q in Q_vals) {
            try({
              fit <- Arima(beer_ts,
                           order = c(p, d, q),
                           seasonal = list(order = c(P, D, Q), period = s),
                           include.constant = FALSE)

              lb_pval <- Box.test(residuals(fit), lag = 24,
```

```
                                    fitdf = p + q + P + Q, type = "Ljung")$p.value

              results[[model_num]] <- list(
                model = fit,
                order = c(p, d, q),
                seasonal = c(P, D, Q),
                seasonal_period = s,
                AIC = AIC(fit),
                LjungBox_p = lb_pval
              )
              model_num <- model_num + 1
            }, silent = TRUE)
          }
        }
      }
    }
  }
}


results_df <- bind_rows(lapply(results, function(x) {
  tibble(
    order = paste0("(", paste(x$order, collapse=","), ")"),
    seasonal = paste0("(", paste(x$seasonal, collapse=","), ")[", x$seasonal_
        period, "]"),
    AIC = x$AIC,
    LjungBox_p = x$LjungBox_p
  )
}))


# Fits from the model fitting algorithm -----------------------------------

fit1 <- Arima(beer_ts, order = c(2, 1, 3),
              seasonal = list(order = c(1, 1, 1), period = 6),
              include.constant = FALSE)
checkresiduals(fit1)

# These other two fits had a reasoanable fit
fit2 <- Arima(beer_ts, order = c(2, 1, 3),
              seasonal = list(order = c(0, 1, 1), period = 12),
              include.constant = FALSE)
checkresiduals(fit2)
fit3 <- Arima(beer_ts, order = c(3, 1, 3),
              seasonal = list(order = c(0, 1, 1), period = 12),
              include.constant = FALSE)
checkresiduals(fit3)

# Kalman Smoothing --------------------------------------------------------

# Basic attempt
fit1_imp = na_kalman(beer_ts)
tsplot(fit1_imp)

# Fit with my own state space model from above
mod_ss = arima(beer_ts, order = c(2,1,3),
               seasonal = list(order = c(0, 1, 1),
                               period = 12))
```

```r
fit2_imp = na_kalman(beer_ts, model = mod_ss$model, smooth = TRUE)
ggplot_na_imputations(beer_ts, fit2_imp)


# Try other imputation algorithms
fit3_imp <- na_interpolation(prod_target_ts, option = "stine")
ggplot_na_imputations(beer_ts,fit3_imp) + labs(title = "Imputation from
    Interpoltaion")

fit4_imp <- na_kalman(prod_target_ts, model = "auto.arima")
ggplot_na_imputations(beer_ts,fit4_imp) + labs(title = "Imputation from Kalman
    Auto Arima")

fit5_imp <- na_ma(prod_target_ts)
ggplot_na_imputations(beer_ts,fit5_imp) + labs(title = "Imputation from Moving
    Average")


# Confidence intervals ----------------------------------------------------

resid_ss = as.vector(fit2$residuals)
sd_imp = sd(resid_ss, na.rm = TRUE)

# Get the value and index of imputed values
miss_idx = which(is.na(as.vector(beer_ts)))

imp_val = fit2_imp[miss_idx]
CI_up = imp_val + 2*sd_imp
CI_lw = imp_val - 2*sd_imp



plot(as.vector(beer_ts), type = "l", ylab = "Beer Production", xlab = "Time")
points(miss_idx, imp_val, col = "red", type = "o")
points(miss_idx, CI_up, col = "blue", type = "l")
points(miss_idx, CI_lw, col = "blue", type = "l")
legend("topleft", legend = c("Imputed Value", "95% Confidence Interval"),
       fill= c("red", "blue"))


# Visual  -----------------------------------------------------------------
library(ggplot2)

# Set up vectors
beer_vec <- as.vector(beer_ts)
time_idx <- seq_along(beer_vec)

# Make a df for ggplot
df <- data.frame(
  Time = time_idx,
  Beer = beer_vec
)

# Get the imputations
imp_df <- data.frame(
  Time = miss_idx,
  Imputed = imp_val,
  CI_Lower = CI_lw,
```

```
  CI_Upper = CI_up
)

ggplot(df, aes(x = Time, y = Beer)) +
  geom_line(color = "black") +
  geom_ribbon(data = imp_df, aes(x = Time, ymin = CI_Lower, ymax = CI_Upper),
              fill = "blue", alpha = 0.4, inherit.aes = FALSE) +
  geom_line(data = imp_df, aes(x = Time, y = Imputed), color = "red", size = 0.5)
      +
  geom_point(data = imp_df, aes(x = Time, y = Imputed), color = "red", size = 1.2)
       +
  labs(title = "Beer Production with Imputed Values",
       x = "Time", y = "Beer Production") +
  theme_minimal(base_size = 14) +
  theme(plot.title = element_text(hjust = 0.5))
```

Appendix D: R Code for Scenario 4

The following R script was used to fit the models for Scenario 4.

```
library(astsa)
library(fpp3)
library(tidyverse)
library(forecast)
library(imputeTS)

# Scenario 4 ------------------------------------------------------------------

# Read in the data
prod_1 = read.csv("Data/ProdData/prod_1.txt")
prod_1 = prod_1 %>%
  select(V1, V2) %>%
  rename(time = V1, Car = V2) %>%
  mutate(time = yearmonth(time)) %>%
  as_tsibble(index = time)

prod_1 %>% gg_tsdisplay(plot_type = c("auto"))
prod_1 %>% autoplot(Car) + labs(title = "Car Production")

prod_2 = read.csv("Data/ProdData/prod_2.txt")
prod_2 = prod_2 %>%
  select(V1, V2) %>%
  rename(time = V1, Steel = V2) %>%
  mutate(time = yearmonth(time)) %>%
  as_tsibble(index = time)

prod_2 %>% autoplot(Steel) + labs(title = 'Steel Production')
prod_2 %>% gg_tsdisplay(log(Steel),plot_type = c("auto"))

eng_1 = read.csv("Data/ProdData/eng_1.txt")
eng_1 = eng_1 %>%
  select(V1, V2) %>%
  rename(time = V1, Gas = V2) %>%
  mutate(time = yearmonth(time)) %>%
  as_tsibble(index = time)

eng_1 %>% gg_tsdisplay(Gas, (plot_type = c("auto")))
eng_1 %>% autoplot(Gas) + labs(title = "Gas Consumption")
```

```r
eng_1 %>% gg_tsdisplay(log(Gas), (plot_type = c("auto")))

eng_2 = read.csv("Data/ProdData/eng_2.txt")
eng_2 = eng_2 %>%
  select(V1, V2) %>%
  rename(time = V1, Elec = V2) %>%
  mutate(time = yearmonth(time)) %>%
  as_tsibble(index = time)


eng_2 %>% gg_tsdisplay(Elec,  plot_type = "auto")
eng_2 %>% autoplot(Elec) + labs(title = "Electricity Consumption")
eng_2 %>% gg_tsdisplay(log(Elec),  plot_type = "auto")

# And let us recall our orginal time series
ggplot_na_imputations(beer_ts, fit2_imp)
autoplot(log(fit2_imp))

# Model Fitting ------------------------------------------------------------
# With all the time series data in these scenario we see an increase in variance
# We want to adjust for that by taking the log. The rest of the non-stationarity
    will be
# dealt with by taking the difference.

# First we fit some regression models to see what the best predictors could be

# Regression Fitting -------------------------------------------------------

y = log(fit2_imp)

Elec = log(eng_2$Elec)
Gas  = log(eng_1$Gas)
Steel = log(prod_2$Steel)

xreg1 = cbind(
  Elec  = Elec,
  Gas = Gas,
  Steel = Steel
)

xreg2 = cbind(Gas = Gas,
              Steel = Steel)
xreg3 = cbind(Elec = Elec,
              Elec2 = Elec^2)
xreg4 = cbind(Steel = Steel,
              Elec = Elec)
xreg5 = cbind(Elec = Elec,
              Gas = Gas)

model_lm1 = lm(y ~ xreg[,"Elec"] + I(xreg[, "Elec"]^2))
summary(model_lm1)

model_lm2 = lm(y ~ xreg[, "Gas"])
summary(model_lm2)

model_lm3 = lm(y ~ xreg[, "Steel"])
summary(model_lm3)

model_lm4 = lm(y ~ xreg)
```

```
summary(model_lm4)

model_lm5 = lm(y ~ xreg[,"Gas"] + xreg[, "Steel"])
summary(model_lm5)

log_Steel = as.vector(log(prod_2$Steel))
log_Gas = as.vector(log(eng_1$Gas))
log_Elec = as.vector(log(eng_2$Elec))

# Insepct the residuals from some of the models
acf(model_lm4$residuals)
naive_mod = arima(y, order = c(0,1,0),
      seasonal = list(order = c(0, 1, 0),
                      period = 12), xreg = xreg)
checkresiduals(naive_mod)
# Modelling ARIMA errors ------------------------------------------------

# Both fail diagnostics
fit = auto.arima(log_beer_ts, xreg = log_Steel, stepwise = FALSE)
checkresiduals(fit)

fit2 = auto.arima(log_beer_ts, stepwise = FALSE)
checkresiduals(fit2)

# Try regression with the ARIMA model we used in Scenario 3
# Test out some model fits with the new regressors
fit_reg_1 = arima(
  y,
  order    = c(2, 1, 3),
  seasonal = c(0, 1, 1)

)

summary(fit_reg_1)
checkresiduals(fit_reg_1)

fit_reg_2 = arima(
  y,
  order = c(2,1,3),
  seasonal = c(0,1,1),
  xreg = xreg
)
checkresiduals(fit_reg_2)


fit_reg_3 = arima(
  y,
  order = c(2,1,3),
  seasonal = c(0,1,1),
  xreg = xreg2
)
checkresiduals(fit_reg_3)

fit_reg_4 = arima(
  y,
  order = c(2,1,3),
  seasonal = c(0,1,1),
  xreg = xreg3
)
```

```
checkresiduals(fit_reg_4)

fit_reg_5 = arima(
  y,
  order = c(2,1,3),
  seasonal = c(0,1,1),
  xreg = xreg4
)
checkresiduals(fit_reg_5)

fit_reg_6 = arima(
  y,
  order = c(2,1,3),
  seasonal = c(0,1,1),
  xreg = xreg5
)
checkresiduals(fit_reg_6)


model_reg = list(fit_reg_1, fit_reg_2, fit_reg_3, fit_reg_4, fit_reg_5, fit_reg_6)

# Validation ------------------------------------------------------------------
candidate_models = list(
  list(name = "xreg1", desc = "Elec, Gas, Steel", xreg = xreg1),
  list(name = "xreg2", desc = "Gas, Steel", xreg = xreg2),
  list(name = "xreg3", desc = "Elec, Elec^2", xreg = xreg3),
  list(name = "xreg4", desc = "Steel, Elec", xreg = xreg4),
  list(name = "xreg5", desc = "Elec, Gas", xreg = xreg5)
)

# For each of the regressors we will simply take the auto arima as the
forecast_regressor = function(x, h) {
  forecast(auto.arima(x), h = h)$mean
}

results = data.frame(Model = character(), Description = character(), MSE = numeric
    (), stringsAsFactors = FALSE)

# Number of rolling validation sets
num_splits = 5

for (candidate in candidate_models) {
  model_name = candidate$name
  desc = candidate$desc
  xreg_full = as.matrix(ts(candidate$xreg, start = start(y), frequency = frequency
      (y)))

  n_total = length(y)
  mse_vals = numeric(num_splits)

  for (k in 1:num_splits) {
    end_train_index = n_total - h * (num_splits - k + 1)

    train_y = window(y, end = time(y)[end_train_index])
    test_y  = window(y, start = time(y)[end_train_index + 1], end = time(y)[end_
        train_index + h])

    train_xreg = window(xreg_full, end = time(y)[end_train_index])
    fc_xreg = forecast_regressors(train_xreg, h)
```

```
    fit_model = Arima(train_y, order = c(2,1,3), seasonal = c(0,1,1),
                      xreg = train_xreg, include.constant = TRUE, method = "ML")

    fc_y = forecast(fit_model, h = h, xreg = fc_xreg)

    acc = accuracy(fc_y, test_y)
    mse_vals[k] = acc["Test set", "RMSE"]^2
    print("here")
  }

  avg_mse = mean(mse_vals)

  results = rbind(results, data.frame(Model = model_name,
                                      Description = desc,
                                      MSE = avg_mse,
                                      stringsAsFactors = FALSE))
}

results = results %>% arrange(MSE)
print(results)

# Add in the baseline model
mse_vals_baseline = numeric(5)

for (k in 1:num_splits) {
  end_train_index = n_total - h * (num_splits - k + 1)

  train_y = window(y, end = time(y)[end_train_index])
  test_y  = window(y, start = time(y)[end_train_index + 1], end = time(y)[end_
      train_index + h])

  fit_baseline = Arima(train_y, order = c(2,1,3), seasonal = c(0,1,1),
                       include.constant = TRUE, method = "ML")

  fc_y = forecast(fit_baseline, h = h)

  acc = accuracy(fc_y, test_y)
  mse_vals_baseline[k] = acc["Test set", "RMSE"]^2
}

avg_mse_baseline = mean(mse_vals_baseline)

# Add to results table
results = rbind(results, data.frame(Model = "baseline",
                                    Description = "ARIMA(2,1,3)(0,1,1)[12] with no
                                        xreg",
                                    MSE = avg_mse_baseline,
                                    stringsAsFactors = FALSE))

# FINAL MODEL ------------------------------------------------------------
# get the predictions for the future
fc_Gas = forecast_regressor(Gas, h)
fc_Steel = forecast_regressor(Steel, h)

# Fit the final model
fc_xreg2 = cbind(Gas = fc_Gas, Steel = fc_Steel)
fit_final_4 = Arima(y, order = c(2,1,3), seasonal = c(0,1,1),
                    xreg = xreg2, include.constant = TRUE, method = "ML")
```

```
# Seems reasonable
fc_y = forecast(fit_final_4, h = h, xreg = fc_xreg2)
autoplot(fc_y) +
  ggtitle("24-Day-Ahead Forecast using xreg2 (Gas and Steel)") +
  ylab("log(Beer Production)") +
  xlab("Time")

# Transfrom back to original scale
fc_y_exp = fc_y
fc_y_exp$mean = exp(fc_y$mean)
fc_y_exp$lower = exp(fc_y$lower)
fc_y_exp$upper = exp(fc_y$upper)

autoplot(fit2_imp, series = "Beer Production") +
  autolayer(fc_y_exp, series = "Forecast", PI = TRUE) +
  ggtitle("24-Month-Ahead Forecast of Beer Production") +
  xlab("Time") +
  ylab("Beer Production") +
  scale_color_manual(values = c("Beer Production" = "darkblue",
                                "Forecast" = "red")) +
  guides(colour = guide_legend(title = "Series"))
```

Appendix E: R Code for Scenario 5

The following R script was used to fit the models for Scenario 5.

```
library(astsa)
library(fpp3)
library(tidyverse)
library(forecast)
library(fable)

# Get the pollution data
city1 = as.ts(read.csv("Data/PollutionData/pollutionCity1.txt")[,2])
city2 = as.ts(read.csv("Data/PollutionData/pollutionCity2.txt")[,2])
city3 = as.ts(read.csv("Data/PollutionData/pollutionCity3.txt")[,2])

# Had to make up some date
# Go to the tsibble world
start_date <- ymd_hm("2024-01-01 00:00")
time_seq <- seq(from = start_date, by = "30 mins", length.out = length(city1))

# Inspect the time series
pollution_ts_1 <- tibble(
  datetime = time_seq,
  pollution = city1
) %>%
  as_tsibble(index = datetime)

pollution_ts_1 %>% gg_tsdisplay(pollution, plot_type = "histogram") +
  labs(title = "Pollution Level City 1")

pollution_ts_2 <- tibble(
  datetime = time_seq,
  pollution = city2
) %>%
  as_tsibble(index = datetime)
```

```r
pollution_ts_2 %>% gg_tsdisplay(pollution, plot_type = "histogram") +
  labs(title = "Pollution Level City 2")

pollution_ts_3 <- tibble(
  datetime = time_seq,
  pollution = city3
) %>%
  as_tsibble(index = datetime)

pollution_ts_3 %>% gg_tsdisplay(pollution, plot_type = "histogram") +
  labs(title = "Pollution Level City 3")

# We can use the STL function to look at the seasonal decomposition of the time
    series
pollution_ts_1  %>%
  model(
    STL(pollution ~ season(period = 48) +
          season(period = 336),
        robust = TRUE)
  ) %>%
  components() %>%
  autoplot() + labs(title = "STL City 1", x = "Pollution Level")

pollution_ts_2 %>%
  model(
    STL(pollution ~ season(period = 48) +
          season(period = 336),
        robust = TRUE)
  ) %>%
  components() %>%
  autoplot() + labs(title = "STL City 2", x = "Pollution Level")

pollution_ts_3 %>%
  model(
    STL(pollution ~ season(period = 48) +
          season(period = 336),
        robust = TRUE)
  ) %>%
  components() %>%
  autoplot() + labs(title = "STL City 3", x = "Pollution Level")

# First Time Series ----------------------------------------------------------

# Get some baseline model
fit_baseline_sarima_1 = auto.arima(pollution_ts_1)
summary(fit_baseline_sarima_1)
fit_baseline_exp_1 = pollution_ts_1 %>% model(ETS(pollution))
glance(fit_baseline_exp_1)

# We want to try some of the dynamic harmonic regression models to capture the two
    modes of seasonality
daily_terms <- 1:6
weekly_terms <- 1:6

param_grid <- expand.grid(K_daily = daily_terms, K_weekly = weekly_terms)
results <- param_grid %>%
  mutate(AICc = NA_real_)
for (i in 1:nrow(param_grid)) {
```

```
  K_daily <- param_grid$K_daily[i]
  K_weekly <- param_grid$K_weekly[i]


  fit <- pollution_ts_1 %>%
    model(DHR = ARIMA(pollution ~ fourier(48, K = K_daily) +
                        fourier(336, K = K_weekly) + trend()))

  model_aicc <- glance(fit) %>% pull(AICc)

  # Store result
  results$AICc[i] <- model_aicc

  cat("Fitted K_daily =", K_daily, "K_weekly =", K_weekly, "-> AICc:", model_aicc,
      "\n")

}

# Take the top three models in regards to AIC

fit_dhr1_1 <- pollution_ts_1 %>%
  model(DHR = ARIMA(pollution ~ fourier(48, K = 4) +
                      fourier(336, K = 2) + trend()))

fit_dhr1_2 <- pollution_ts_1 %>%
  model(DHR = ARIMA(pollution ~ fourier(48, K = 3) +
                      fourier(336, K = 1) + trend()))

fit_dhr1_3 <- pollution_ts_1 %>%
  model(DHR = ARIMA(pollution ~ fourier(48, K = 3) +
                      fourier(336, K = 2) + trend()))


# Try some SARIMA that are around the auto arima parameter s

city1_sarima_models <- pollution_ts_1 %>%
  model(
    M1 = ARIMA(pollution ~ 1 + pdq(4,0,1) + PDQ(0,0,2,48)),
    M2 = ARIMA(pollution ~ 1 + pdq(3,0,1) + PDQ(0,0,2,48)),
    M3 = ARIMA(pollution ~ 1 + pdq(4,0,2) + PDQ(0,0,1,48)),
    M4 = ARIMA(pollution ~ 1 + pdq(2,0,2) + PDQ(0,0,2,48)),
    M5 = ARIMA(pollution ~ 1 + pdq(4,0,1) + PDQ(1,0,1,48))
  )

# The best models are M1 and M2 for AICc


# CROSS VALIDATION CITY 1 ------------------------------------------------
# We do expanding window, moving the window obver 336 observations each time
h <- 336
n_total <- nrow(pollution_ts_1)
n_windows <- 5

results <- list()

for (i in 0:(n_windows - 1)) {

  end_train <- n_total - h * (n_windows - i)
```

```
  start_test <- end_train + 1
  end_test <- start_test + h - 1

  train_ts <- pollution_ts_1[1:end_train, ]
  test_ts  <- pollution_ts_1[start_test:end_test, ]

  fit_models <- train_ts %>%
    model(
      auto_arima = ARIMA(pollution ~ 1 + pdq(4,0,1) + PDQ(0,0,2,48)),
      DHR1 = ARIMA(pollution ~ fourier(48, 4) + fourier(336, 2) + trend()),
      DHR2 = ARIMA(pollution ~ fourier(48, 3) + fourier(336, 1) + trend()),
      DHR3 = ARIMA(pollution ~ fourier(48, 3) + fourier(336, 1) + trend()),
      DHR4 = ARIMA(pollution ~ fourier(48, 6) + fourier(336, 4) + trend()),
      manual_arima = ARIMA(pollution ~ 1 + pdq(3,0,1) + PDQ(0,0,2,48))
    )

  fc <- forecast(fit_models, h = h)
  acc <- accuracy(fc, test_ts) %>% mutate(window = i + 1) # Gives back RMSE which
     is fine

  results[[i + 1]] <- acc
  print(paste("Done window", i + 1))
}

final_results <- bind_rows(results)

summary_results <- final_results %>%
  select(-window) %>%
  group_by(.model) %>%
  summarise(across(where(is.numeric), mean, na.rm = TRUE), .groups = "drop")

# Look at the results. All teh models did reasonable
print(summary_results)

# City 2 ------------------------------------------------------------------
# We follow a similar method from above
# Get some baseline model
fit_baseline_sarima_2 = auto.arima(pollution_ts_2)
summary(fit_baseline_sarima_2)
fit_baseline_exp_2 = pollution_ts_2 %>% model(ETS(pollution))
glance(fit_baseline_exp_2)

# We want to try some of the dynamic harmonic regression models to capture the two
    modes of seasonality
daily_terms <- 1:6
weekly_terms <- 1:6

param_grid <- expand.grid(K_daily = daily_terms, K_weekly = weekly_terms)
results2 <- param_grid %>%
  mutate(AICc = NA_real_)
for (i in 1:nrow(param_grid)) {

  K_daily <- param_grid$K_daily[i]
  K_weekly <- param_grid$K_weekly[i]


  fit <- pollution_ts_2 %>%
    model(DHR = ARIMA(pollution ~ fourier(48, K = K_daily) +
                        fourier(336, K = K_weekly) + trend()))
```

```
  model_aicc <- glance(fit) %>% pull(AICc)

  # Store result
  results2$AICc[i] <- model_aicc

  cat("Fitted K_daily =", K_daily, "K_weekly =", K_weekly, "-> AICc:", model_aicc,
      "\n")

}

# Get some of the best models for AIC
fit_dhr2_1 <- pollution_ts_2 %>%
  model(DHR = ARIMA(pollution ~ fourier(48, K = 3) +
                      fourier(336, K = 1) + trend()))

fit_dhr2_2 <- pollution_ts_2 %>%
  model(DHR = ARIMA(pollution ~ fourier(48, K = 4) +
                      fourier(336, K = 1) + trend()))

fit_dhr2_3 <- pollution_ts_2 %>%
  model(DHR = ARIMA(pollution ~ fourier(48, K = 3) +
                      fourier(336, K = 3) + trend()))


# Try some SARIMA that are around the auto arima

city2_sarima_models <- pollution_ts_2 %>%
  model(
    M1 = ARIMA(pollution ~ 1 + pdq(3,1,3) + PDQ(0,0,1,48)),
    M2 = ARIMA(pollution ~ 1 + pdq(3,1,2) + PDQ(0,0,1,48)),
    M3 = ARIMA(pollution ~ 1 + pdq(2,1,3) + PDQ(0,0,1,48)),
    M4 = ARIMA(pollution ~ 1 + pdq(2,1,2) + PDQ(0,0,1,48)),
    M5 = ARIMA(pollution ~ 1 + pdq(4,1,3) + PDQ(0,0,1,48))
  )

# Cross Validation Time Series 2 ----------------------------------------
results2 <- list()

for (i in 0:(n_windows - 1)) {

  end_train <- n_total - h * (n_windows - i)
  start_test <- end_train + 1
  end_test <- start_test + h - 1

  train_ts2 <- pollution_ts_2[1:end_train, ]
  test_ts2  <- pollution_ts_2[start_test:end_test, ]

  fit_models2 <- train_ts2 %>%
    model(
      auto_arima = ARIMA(pollution),
      DHR1 = ARIMA(pollution ~ fourier(48, 3) + fourier(336, 1) + trend()),
      DHR2 = ARIMA(pollution ~ fourier(48, 4) + fourier(336, 1) + trend()),
      DH32 = ARIMA(pollution ~ fourier(48, 3) + fourier(336, 3) + trend()),
      manual_arima = ARIMA(pollution ~ 1 + pdq(4,1,3) + PDQ(0,0,1,48))
    )

  fc2 <- forecast(fit_models2, h = h)
  acc2 <- accuracy(fc2, test_ts2) %>% mutate(window = i + 1)
```

```
  results2[[i + 1]] <- acc2
  print(paste("Done window", i + 1))
}

# Combine results from all windows
final_results2 <- bind_rows(results2)

# Summarize accuracy over all windows
summary_results2 <- final_results2 %>%
  select(-window) %>%
  group_by(.model) %>%
  summarise(across(where(is.numeric), mean, na.rm = TRUE), .groups = "drop")

# Print average performance
print(summary_results2)

# City 3 ------------------------------------------------------------------

fit_baseline_sarima_3 = auto.arima(pollution_ts_3)
summary(fit_baseline_sarima_3)
fit_baseline_exp_3 = pollution_ts_3 %>% model(ETS(pollution))
glance(fit_baseline_exp_3)

# We want to try some of the dynamic harmonic regression models to capture the two
    modes of seasonality
daily_terms <- 1:6
weekly_terms <- 1:6

param_grid <- expand.grid(K_daily = daily_terms, K_weekly = weekly_terms)
results3 <- param_grid %>%
  mutate(AICc = NA_real_)
for (i in 1:nrow(param_grid)) {

  K_daily <- param_grid$K_daily[i]
  K_weekly <- param_grid$K_weekly[i]


  fit <- pollution_ts_3 %>%
    model(DHR = ARIMA(pollution ~ fourier(48, K = K_daily) +
                        fourier(336, K = K_weekly)))

  model_aicc <- glance(fit) %>% pull(AICc)

  # Store result
  results3$AICc[i] <- model_aicc

  cat("Fitted K_daily =", K_daily, "K_weekly =", K_weekly, "-> AICc:", model_aicc,
      "\n")

}
# These were the top modles with respect to AIC
fit_dhr3_1 <- pollution_ts_3 %>%
  model(DHR = ARIMA(pollution ~ fourier(48, K = 3) +
                      fourier(336, K = 1) ))

fit_dhr3_2 <- pollution_ts_3 %>%
  model(DHR = ARIMA(pollution ~ fourier(48, K = 5) +
                      fourier(336, K = 1)))
```

```r
fit_dhr3_3 <- pollution_ts_3 %>%
  model(DHR = ARIMA(pollution ~ fourier(48, K = 3) +
                      fourier(336, K = 2)))

# Try some SARIMA that are around teh auto arima parameter s

city3_sarima_models <- pollution_ts_3 %>%
  model(
    M1 = ARIMA(pollution ~ 1 + pdq(1,1,0) + PDQ(0,0,2,48)),
    M2 = ARIMA(pollution ~ 1 + pdq(1,1,1) + PDQ(0,0,2,48)),
    M3 = ARIMA(pollution ~ 1 + pdq(1,1,1) + PDQ(0,0,2,48)),
    M4 = ARIMA(pollution ~ 1 + pdq(2,1,0) + PDQ(0,0,2,48)),
    M5 = ARIMA(pollution ~ 1 + pdq(2,1,1) + PDQ(0,0,2,48))
  )
city3_sarima_models %>% glance()
# Cross Validation Time Series 3 ----------------------------------------
h <- 336
n_total <- nrow(pollution_ts_3)
n_windows <- 5

results3 <- list()

for (i in 0:(n_windows - 1)) {

  end_train <- n_total - h * (n_windows - i)
  start_test <- end_train + 1
  end_test <- start_test + h - 1

  train_ts3 <- pollution_ts_3[1:end_train, ]
  test_ts3  <- pollution_ts_3[start_test:end_test, ]

  fit_models3 <- train_ts3 %>%
    model(
      auto_arima = ARIMA(pollution ~ 1+  pdq(1,1,0) + PDQ(0,0,2,48)),
      DHR1 = ARIMA(pollution ~ fourier(48, 3) + fourier(336, 2)),
      DHR2 = ARIMA(pollution ~ fourier(48, 3) + fourier(336, 1)),
      DHR3 = ARIMA(pollution ~ fourier(48, 5) + fourier(336, 2)),
      manual_arima = ARIMA(pollution ~ 1 + pdq(1,1,1) + PDQ(0,0,2,48))
    )

  fc3 <- forecast(fit_models3, h = h)
  acc3 <- accuracy(fc3, test_ts3) %>% mutate(window = i + 1)

  results3[[i + 1]] <- acc3
  print(paste("Done window", i + 1))
}


final_results3 <- bind_rows(results3)

summary_results3 <- final_results3 %>%
  select(-window) %>%
  group_by(.model) %>%
  summarise(across(where(is.numeric), mean, na.rm = TRUE), .groups = "drop")


print(summary_results3)
```

```r
# FORECASTS ------------------------------------------------------------
# City 1
best_model_1 <- pollution_ts_1 %>%
  model(DHR2 = ARIMA(pollution ~ fourier(48, 3) + fourier(336, 1) + trend()))

fc_best <- forecast(best_model_1, h = 336)

fc_best %>%
  autoplot(pollution_ts_1) +
  ggtitle("Forecast from Best Model (DHR2)") +
  xlab("Time") + ylab("Pollution Level")

# City 2
best_model_2 <- pollution_ts_2 %>%
  model(auto_arima = ARIMA(pollution))

fc_best2 <- forecast(best_model_2, h = 336)

fc_best2 %>%
  autoplot(pollution_ts_2) +
  ggtitle("Forecast from Best Model ARIMA AUTO") +
  xlab("Time") + ylab("Pollution Level")

# City 3
best_model_3 <- pollution_ts_3 %>%
  model(DHR2 = ARIMA(pollution ~ fourier(48, 3) +
                       fourier(336, 1)))

fc_best3 <- forecast(best_model_3, h = 336)

fc_best3 %>%
  autoplot(pollution_ts_3) +
  ggtitle("Forecast from Best Model (DHR2)") +
  xlab("Time") + ylab("Pollution Level")
```