**Data 624: Project 1**

**Time Series Analysis and Forecasting**

**Group 2: Alice Friedman, Kayleah Griffen, Josh Iden, Michael Ippolito**

Table of Contents

[Executive Summary 3](#_Toc138018185)

[Introduction 4](#_Toc138018186)

[Exploratory Data Analysis 5](#_Toc138018187)

[Loading the Data 5](#_Toc138018188)

[First Visualization of the Data 5](#_Toc138018189)

[Missing Values 6](#_Toc138018190)

[Correlation between variables 7](#_Toc138018191)

[Outliers 8](#_Toc138018192)

[Modeling and Forecasting 9](#_Toc138018193)

[Imputation 9](#_Toc138018194)

[Autocorrelation 9](#_Toc138018195)

[Modeling 10](#_Toc138018196)

[Conclusion 12](#_Toc138018197)

[Appendix 1: R Code for Modeling 13](#_Toc138018198)

[Appendix 2: R Code for Exploratory Data Analysis and Testing Frequency of 5 19](#_Toc138018199)

[Appendix 3: Time Series Model Detailed Explanations 110](#_Toc138018200)

# Executive Summary

The purpose of this project was to produce the most accurate time series forecasts on data provided. We were provided multiple different sets of time series data, with 12 of the variables required to be forecast, and in each case we were given 1622 periods and we were expected to predict 140 more. To accomplish this task, first exploratory data analysis was done to better understand the data. Following this step, if the data needed to be transformed to be better suited for modeling it was. Then the data was split up into training and testing data and multiple different models were experimented with in order to find the model that minimized the error in our predictions. Ultimately, we arrived at the best models for each time series being a simple exponential smoothing model and used these models to make the forecasts. The forecasts are provided in the attached excel sheet.

# Introduction

The objective of this project was to perform the appropriate analysis in order to forecast two variables (of five provided) each from six different time series sets, for a total of twelve time series sets to forecast. We were not given any explanation of what the data represents or why it is organized the way it is. We were provided a spreadsheet that contains 1622 periods of every variable in every set and were asked to forecast 140 periods. Both of the given and the predicted series have missing time periods in them. The time series sets are labeled S01, S02, S03, S04, S05 and S06 and each contains variables labeled V01, V02, V03, V05, and V07. Different variables are required to be forecast depending on the set, specified below:

S01 – Forecast Var01, Var02

S02 – Forecast Var02, Var03

S03 – Forecast Var05, Var07

S04 – Forecast Var01, Var02

S05 – Forecast Var02, Var03

S06 – Forecast Var05, Var07

This report is intended for colleagues from a variety of backgrounds and contains both technical and non-technical explanations of the work conducted. To keep the report concise, we have limited the technical descriptions throughout the report, but have provided them in the appendix. Also in the appendix is the full source code used to analyze the data and which lead us to our conclusions.

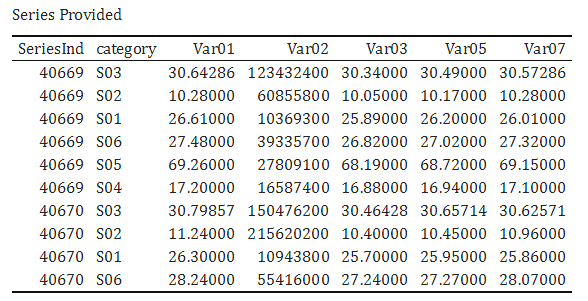
# 

# Exploratory Data Analysis

To start the project, the first step is getting familiar with the data. The steps outlined below were taken to get a better understanding of the dataset, given that we were not provided background on the data - this step is even more important.

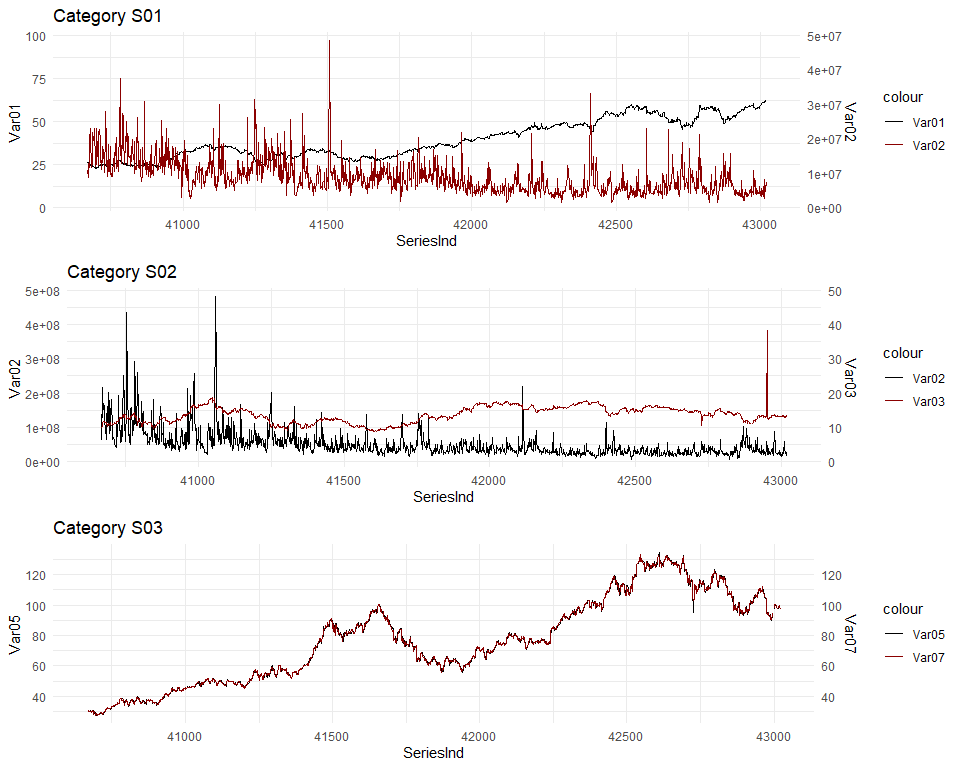
## Loading the Data

The data was provided as an excel (.xls) file. The columns provided were the series index, the category, and then a column for each variable. To conduct the data analysis and forecasting the open source software `r` was used. In order to begin processing the data, the data was read into `r` from github (where the provided data file was stored) and stored in a format in `r` called a dataframe. Below is a preview of the data, as it was read in from excel, to get an idea of the format.



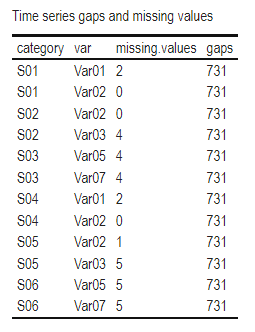
## First Visualization of the Data

To get an initial idea of the data, we created plots of the data with the x-axis as the SeriesInd and the y-axis as the value, examples of this are below. From this we were able to understand that some series exhibited trends while others did not. We also noticed some values that seemed to be outliers, a value far outside the usual range of the data.



## Missing Values

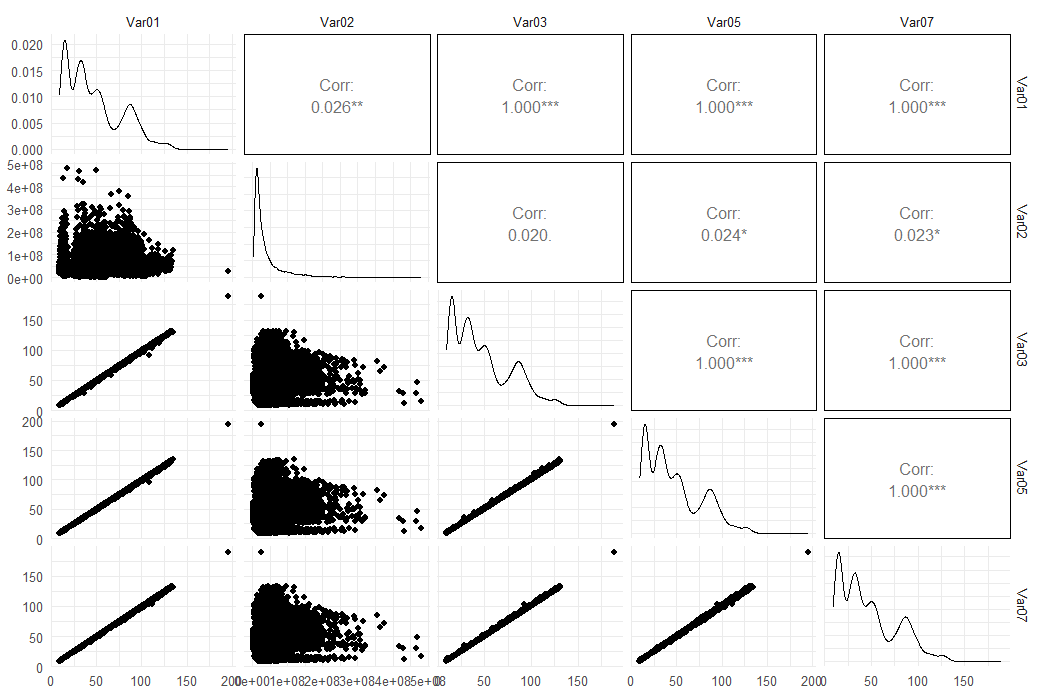
Upon doing a basic inspection of the data, we noticed two different kinds of missing data. There is random missing data in the provided sets, coded NA. There is also missing data in a patterned way, where the data is missing for the same time series indices across all of the sets. To better quantify these two types, we can look at how much data is missing for each. The table below shows how many values were missing in the original data (the NAs) as well as the gaps, which are the missing SeriesInds, for the data sets where we were expected to make a forecast.



The “missing.values” is relatively small, whereas the “gaps” represent quite a lot of missing data. As a part of our analysis, we found that 80% of the data was grouped in sets of 5, with the remaining 20% being in sets of 4,3,2, or 1. As a result of this - while we can not be sure what the data represents because we were not explicitly told - we are inclined to think the data represents something weekly with the standard work week days recorded and the weekends unrecorded, along with holidays or other off days not recorded. Going forward with our data analysis, we considered two different approaches. For one approach, we left the 1622 given periods and imputed, or filled in, the missing values that were given as NAs. For the second approach, we imputed the given NAs as well as created a cyclical five-day week. The reason for creating a five-day week rather than a seven-day week was to minimize the amount of imputing we would have to do which would add unnecessary bias into the data while adding a helpful feature, the frequency of 5, to see if there was a pattern.

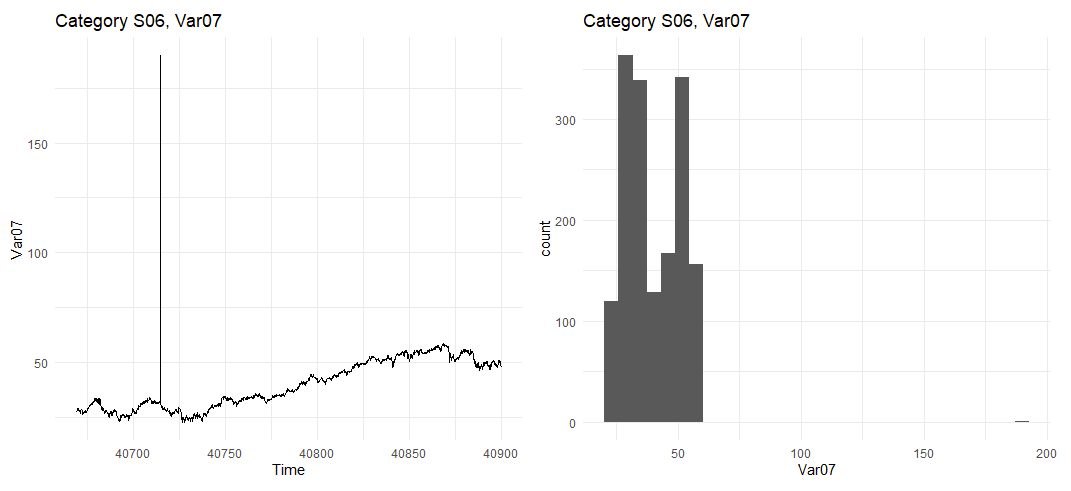
## Correlation between variables

An additional step in exploratory data analysis that is often helpful to better understand the data set is to generate pairwise plots showing the relationship between predictors. Based on these pairwise plots, we observed an extremely high degree of correlation between some predictors. Notably, variables Var01, Var03, Var05, and Var07 contain very similar values, suggesting that any missing values of one might be imputed using existing values of another. Without the context of what these variables represent, it is difficult to speculate on why they might be correlated as such, but it is possible that the data were collected by different observers using similar but slightly different methodologies or techniques. Despite this finding, when a value was missing we did not use other variables to predict it. In the plot below where the Corr is 1, represents a highly correlated variable, whichever variables interest at that Correlation are the variables that are highly correlated.



## Outliers

We discovered some obvious outliers in the data. Outliers adversely affect forecasts and, as such, should be either removed or replaced. Notably,outliers were discovered in the following variables, with four other variables having questionable values. One such example is in Category S06, variable V07, which clearly exhibits an outlying value of approximately about a quarter of the way into the series, shown in two different ways, below.



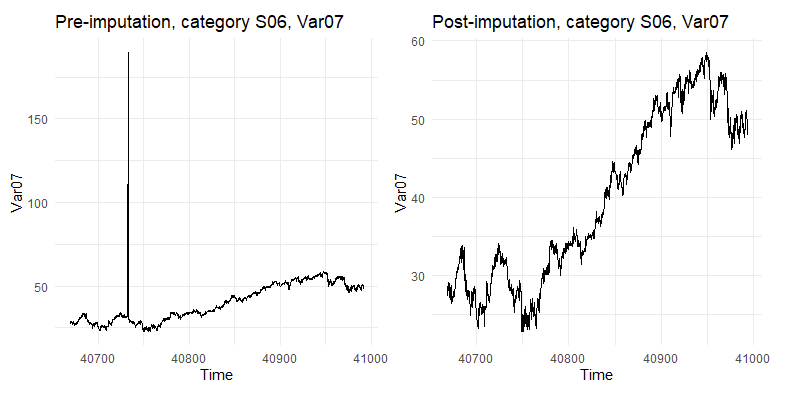
# 

# Modeling and Forecasting

Now that we have a better understanding of the data through the exploratory data analysis, we can take steps to prepare the data for modeling, experiment with different models and then make our forecasts.

## Imputation

To account for the missing values and outliers, we used a function in R called tsclean(), part of the forecast package developed by Hundman (2021). Below is an example of how the data was cleaned up. In the initial plot you can see clear spike in the data which was removed in the second plot.



## Autocorrelation

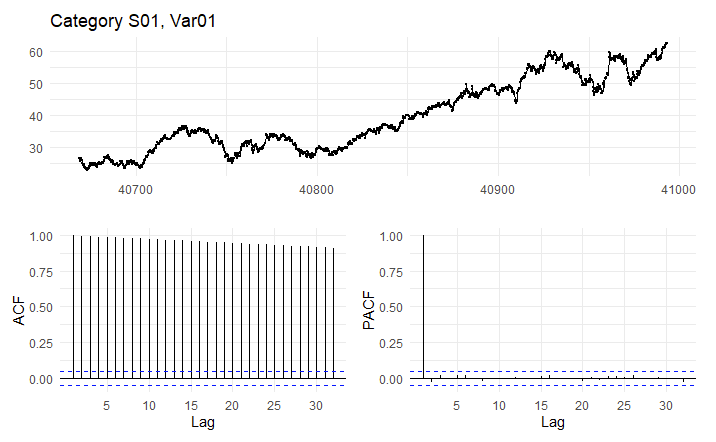
Some types of models are sensitive to data that is autocorrelated, that is, data that contains values which are related to previous values in some regular or predictable way. These models require that the data be modified such that they are "stationary," meaning that they appear to be randomly distributed and when plotted look like "white noise."

To identify autocorrelation patterns in the data, autocorrelation function (ACF) and partial autocorrelation function (PACF) plots were constructed. These plots illustrate the relationship between lagged time series values, i.e. comparing one value with the next value in the series, or between one value and those two or more positions later. Examining the patterns in the ACF and PACF plots helps the modeler determine what parameters to use as a basis when modeling.

ACF and PACF plots aid in evaluating whether the data is autocorrelated and, if so, whether it should be modified before modeling occurs. One such modification is "differencing," which converts time series data into the \*change\* in value over time. Once the data is "differenced" one or more times, it is no longer autocorrelated, and the time series should appear to be "white noise." Likewise, the ACF and PACF plots should exhibit no clear trend or pattern.

Beyond examining ACF and PACF plots, there are more formal means of evaluating whether differencing is required. These include the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test and the nsdiffs() R function for seasonal differencing.

The ACF and PACF plots for the variables in question exhibit similar patterns (see figure below), all of which indicate that, minimally, one iteration of non-seasonal differencing is likely required.



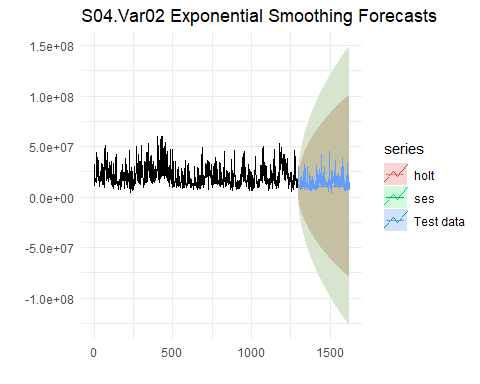
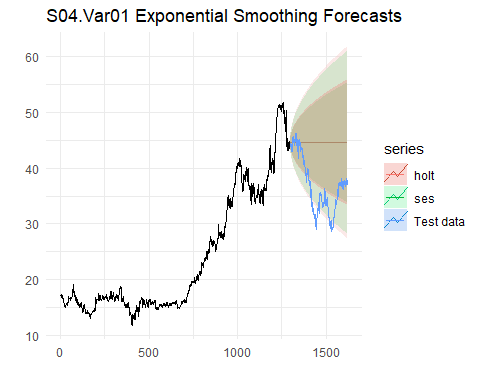
We further confirmed this by running KPSS tests (see table below). Seasonal differencing does not appear necessary, confirmed by running the nsdiff() function against the data set.

## Modeling

Two commonly used models for interpreting time series data are exponential smoothing and ARIMA models. Exponential smoothing is based on using exponentially decaying weighted averages of past observations to predict future values. The exponential smoothing model can be broken up into three parts: the trend, the seasonal and the error component.

ARIMA is a more complex model which includes autoregression (AR), integration through differencing (I), and a moving average (MA). ARIMA is suitable for cases when there is both seasonality and trend, and accounts for both in the process – time series data does not have to be stationary in order to be suitable for forecasting with ARIMA. Multiple versions of each of these models were attempted.

Based on the results of our modeling, the exponential smoothing models performed better across the board, with the exception of one variant, where the accuracy was only slightly higher than that of the corresponding ARIMA model. We compared the analysis of the original data with only imputing the NA values to imputing to a five day week and found the seasonal component in the five day week was so small that it did not improve the model. Ultimately, to maintain consistency and to conform to simpler models, we chose to use all of the exponential smoothing models over the slight gain in performance. Using the selected models, forecasts were generated for each variable and are included as a multi-tab Excel spreadsheet. Example forecasts are shown below. The black line is the training data, the blue line is the test data, the faded green line is the exponential smoothing data and the shaded areas are the confidence intervals.



Typically, models are evaluated not only on their accuracy but on how stationary the residual values are; that is, if the differences between the modeled values and the actual values exhibit a random pattern, this indicates a good model fit. This can be checked using a statistical test called the Ljung-Box test. It is noted that our exponential smoothing models produced subpar Ljung-Box results. The ARIMA models yielded better Ljung-Box results, but we nonetheless opted to choose the exponential smoothing models due to our observation that the ARIMA models appeared to overfit the data. Moreover, exponential smoothing models have the benefit of simplicity and typically perform better at forecasting non-seasonal time series.

# Conclusion

Based on our analysis, we conclude the following:

* Although there was no context given, the data is likely daily values, possibly stock or commodity prices, with values given for every four- or five-day cycle, likely corresponding to weekdays minus the occasional holiday.
* A small percentage of the data was missing, coded as NAs in the original data, which we accounted for by imputing.
* The data included a very small proportion of values that were well outside of the average, which we treated as outliers and replaced with more reasonable values.
* The data exhibited only slight seasonality, and, therefore, we opted to model it as non-seasonal.
* The data was autocorrelated, indicating that each successive value bore a relationship to the previous value. As such, the data was differenced during the modeling process where appropriate, such that changes in value were modeled rather than the values themselves.
* Prior to modeling, we segregated the data into training and test sets using an 80%-20% split.
* We modeled using exponential smoothing and ARIMA models to produce MAPE values for comparison.
* Based on the lower MAPE scores validated against the test set, we opted to select the exponential smoothing models, despite their suboptimal residual results, as the ARIMA models appeared to overfit the data.

# Appendix 1: R Code for Modeling

knitr::opts\_chunk$set(fig.path='Figs/', echo=FALSE, warning=FALSE, message=FALSE, cache=TRUE)  
library(httr)  
library(kableExtra)  
library(fpp2)  
library(imputeTS)  
library(tidyverse)  
library(urca)  
library(ggfortify)  
library(gridExtra)  
library(scales)  
library(flextable)  
library(openxlsx)  
set.seed(123)  
  
# Set minimal theme  
theme\_set(theme\_minimal())  
  
github\_link <- "https://github.com/klgriffen96/summer23\_data624/raw/main/project\_1/Data%20Set%20for%20Class.xls"  
temp\_file <- tempfile(fileext = ".xls")  
req <- GET(github\_link,   
 # write result to disk  
 write\_disk(path = temp\_file))  
  
df <- readxl::read\_excel(temp\_file)  
  
head(df, 10) |>  
 flextable()  
#Split the data into data frames by category and var name  
  
df\_long <- df %>% gather(key, value, -SeriesInd, -category)  
split\_data <- split(df\_long, f=list(df\_long$category, df\_long$key))  
  
  
#put the combo into a list so it can be run through  
list\_vars <- c(  
 "S01.Var01", "S01.Var02",  
 "S02.Var02", "S02.Var03",  
 "S03.Var05", "S03.Var07",  
 "S04.Var01", "S04.Var02",  
 "S05.Var02", "S05.Var03",  
 "S06.Var05", "S06.Var07"  
)  
  
#select list items based on the list vars and then turn each list item   
myts\_raw <- lapply(split\_data[list\_vars], function(x) {  
 x %>%   
 dplyr::select(value) %>%   
 slice(1:1622) %>% #removes the missing values we need to predict  
 ts()  
})  
  
  
#select list items based on the list vars and then turn each list item into a clean ts  
myts <- lapply(split\_data[list\_vars], function(x) {  
 x %>%   
 dplyr::select(value) %>%   
 slice(1:1622) %>% #removes the missing values we need to predict  
 ts() %>%   
 tsclean() %>%  
 na\_ma()   
})  
  
  
plot\_grid <- function(df){  
# Takes as input a dataframe which is a list of ts objects   
  
# Create an empty list to store the plots  
plot\_list <- list()  
  
# Generate the plots and store them in the plot\_list  
for (var in list\_vars) {  
 ts <- df[[var]]  
 plot <- autoplot(ts) + ggtitle(var) + scale\_y\_continuous(labels = scales::comma) # Customize y-axis labels  
 plot\_list[[var]] <- plot  
}  
  
# Arrange the plots in a grid  
grid\_arranged\_plots <- do.call(grid.arrange, c(plot\_list, ncol = 4))  
  
# Print the grid of plots  
print(grid\_arranged\_plots)  
}  
  
plot\_grid(myts\_raw)  
plot\_grid(myts)  
# Function to backtransorm differencing  
back\_diff <- function(diff\_t, t1){  
   
 # Append the initial value  
 init <- append(t1, diff\_t)  
   
 # Cumsum the diffed series with the inital value as the first item  
 backtransformed\_series <- cumsum(init)  
   
 # Return result   
 return(backtransformed\_series)  
   
 }  
   
  
# Create a function to test the ses and holt forecasts with test and train data  
ses\_test <- function(x, i, log=FALSE) {  
   
 # Determine the index to split the time series into train and test sets  
 split.index <- floor(0.8 \* length(x)) # 80% for training, 20% for testing  
   
 # Split the time series into train and test sets  
 train <- window(x, end = split.index)   
 test <- window(x, start = split.index + 1)  
   
 # Set the horizon  
 horizon <- length(test)  
   
 ######## FOR LOG TRANSFORMED  
 if(log==TRUE){  
 train <- log(train)  
 # Ses fit with training data  
 ses.fit <- ses(train, h = horizon)$mean %>%   
 exp() %>%   
 ts(start=split.index + 1)  
  
 # Test with test data BACKTRANSFORMED  
 ses\_res <- accuracy(ses.fit, test)['Test set', 'MAPE']  
   
 # Holt fit with training data  
 holt.fit <- holt(train, damped = TRUE, h = horizon)$mean %>%   
 exp() %>%   
 ts(start=split.index + 1)  
  
 # Test with test data  
 holt\_res <- accuracy(holt.fit, test)['Test set', 'MAPE']  
 }  
   
   
 ####### NO TRANSFORM  
 else{  
 # Ses fit with training data  
 ses.fit <- ses(train, h = horizon)  
  
 # Test with test data  
 ses\_res <- accuracy(ses.fit, test)['Test set', 'MAPE']  
   
 # Holt fit with training data  
 holt.fit <- holt(train, damped = TRUE, h = horizon)  
  
 # Test with test data  
 holt\_res <- accuracy(holt.fit, test)['Test set', 'MAPE']  
 }  
  
 ######################  
 # Create a plot  
 p <- autoplot(window(x, end=split.index)) +  
 autolayer(ses.fit, series = "ses") + #ses forecast  
 autolayer(holt.fit, alpha = 0.4, series = "holt") +  
 autolayer(test, series = "Test data") +  
 ggtitle(paste(list\_vars[i], "Exponential Smoothing Forecasts"))  
   
 # Return list of results  
 result <- list(ses\_MAPE = ses\_res, #1  
 holt\_MAPE = holt\_res #2  
 )  
   
 print(p)  
 return(result)  
}  
# Create the empty vectors  
smooth\_results <- vector(mode = "list", length = length(myts))  
ses\_MAPE <- vector("numeric", length = length(list\_vars))  
holt\_MAPE <- vector("numeric", length = length(list\_vars))  
ses\_p <- vector("numeric", length = length(list\_vars))  
holt\_p <- vector("numeric", length = length(list\_vars))  
  
# Run through the function to build the lists  
for (i in seq\_along(myts)) {  
 result <- ses\_test(myts[[i]], i, F)  
 ses\_MAPE[i] <- result[1]  
 holt\_MAPE[i] <- result[2]  
}  
# Create the empty vectors  
arima\_MAPE\_lambda0 <- vector(mode = "list", length = length(myts))  
arima\_MAPE\_lambda1 <- vector(mode = "list", length = length(myts))  
  
# Create a function to test the ses and holt forecasts with test and train data  
arima\_test <- function(x, i, lambda) {  
   
 # Determine the index to split the time series into train and test sets  
 split.index <- floor(0.8 \* length(x)) # 80% for training, 20% for testing  
   
 # Split the time series into train and test sets  
 train <- window(x, end = split.index)  
 test <- window(x, start = split.index + 1)  
   
 # Set the horizon  
 horizon <- length(test)  
   
 #auto arima fit  
 arima.fc <- train %>%   
 auto.arima(lambda = lambda) %>%  
 forecast(h=horizon)  
   
 # test results  
 result <- accuracy(arima.fc, test)['Test set', 'MAPE']  
   
 #plot residuals  
 #p <- checkresiduals(arima.fc)  
 p <- autoplot(arima.fc) +  
 autolayer(test) +  
 ggtitle(list\_vars[i])  
  
 print(p)  
   
 return(result)  
}  
  
  
## Run   
for (i in seq\_along(myts)) {  
 arima\_MAPE\_lambda0[i]<- do.call(arima\_test, list(myts[[i]], i, 0))  
 arima\_MAPE\_lambda1[i]<- do.call(arima\_test, list(myts[[i]], i, 1))  
}  
results\_df<- cbind(  
 list\_vars,  
 ses\_MAPE,  
 holt\_MAPE,  
 arima\_MAPE\_lambda0,  
 arima\_MAPE\_lambda1  
)  
# Print the results as a table  
results\_df # %>% flextable()  
SeriesInd <- df\_long$SeriesInd %>% unique()  
  
forecasts <- lapply(myts, function(x){  
  
 ses.fit <- ses(x, h = 140)  
   
 cbind(SeriesInd,   
 append(x, ses.fit$mean))  
})  
  
  
  
S01 <- merge(forecasts$S01.Var01, forecasts$S01.Var02, by = "SeriesInd")  
names(S01) <- c("SeriesInd", "Var01", "Var02")  
  
S02 <- merge(forecasts$S02.Var02, forecasts$S02.Var03, by = "SeriesInd")  
names(S02) <- c("SeriesInd", "Var02", "Var03")  
  
S03 <- merge(forecasts$S03.Var05, forecasts$S03.Var07, by = "SeriesInd")  
names(S03) <- c("SeriesInd", "Var05", "Var07")  
  
S04 <- merge(forecasts$S04.Var01, forecasts$S04.Var02, by = "SeriesInd")  
names(S04) <- c("SeriesInd", "Var01", "Var02")  
  
S05 <- merge(forecasts$S05.Var02, forecasts$S05.Var03, by = "SeriesInd")  
names(S05) <- c("SeriesInd", "Var02", "Var03")  
  
S06 <- merge(forecasts$S06.Var05, forecasts$S06.Var07, by = "SeriesInd")  
names(S06) <- c("SeriesInd", "Var05", "Var07")  
  
export <- list(  
 S01=S01,  
 S02=S02,  
 S03=S03,  
 S04=S04,  
 S05=S05,  
 S06=S06  
)  
#reference: https://www.statology.org/r-export-to-excel-multiple-sheets/  
#export each data frame to separate sheets in same Excel file  
openxlsx::write.xlsx(export, file = 'group2\_forecasts.xlsx')

# Appendix 2: R Code for Exploratory Data Analysis and Testing Frequency of 5

Load data

# Load data from git  
GET('https://github.com/klgriffen96/summer23\_data624/raw/main/project\_1/Data%20Set%20for%20Class.xls', write\_disk(tmpfile <- tempfile(fileext=".xls")))

## Response [https://raw.githubusercontent.com/klgriffen96/summer23\_data624/main/project\_1/Data%20Set%20for%20Class.xls]  
## Date: 2023-06-19 01:50  
## Status: 200  
## Content-Type: application/octet-stream  
## Size: 1.55 MB  
## <ON DISK> C:\Users\kayle\AppData\Local\Temp\RtmpC4knCr\file504c6ce4349c.xls

df\_orig <- readxl::read\_excel(tmpfile, skip=0)

Data preparation

# Set frequency  
the\_freq <- 5  
  
# Remove blank observations at end  
df <- df\_orig %>%  
 filter(SeriesInd <= 43021) %>%  
 arrange(SeriesInd, category)  
  
# Initial summary  
summary(df)

## SeriesInd category Var01 Var02   
## Min. :40669 Length:9732 Min. : 9.03 Min. : 1339900   
## 1st Qu.:41253 Class :character 1st Qu.: 23.10 1st Qu.: 12520675   
## Median :41846 Mode :character Median : 38.44 Median : 21086550   
## Mean :41843 Mean : 46.98 Mean : 37035741   
## 3rd Qu.:42430 3rd Qu.: 66.78 3rd Qu.: 42486700   
## Max. :43021 Max. :195.18 Max. :480879500   
## NA's :14 NA's :2   
## Var03 Var05 Var07   
## Min. : 8.82 Min. : 8.99 Min. : 8.92   
## 1st Qu.: 22.59 1st Qu.: 22.91 1st Qu.: 22.88   
## Median : 37.66 Median : 38.05 Median : 38.05   
## Mean : 46.12 Mean : 46.55 Mean : 46.56   
## 3rd Qu.: 65.88 3rd Qu.: 66.38 3rd Qu.: 66.31   
## Max. :189.36 Max. :195.00 Max. :189.72   
## NA's :26 NA's :26 NA's :26

# Fill in the existing NAs with Infs so we can distinguish them later  
df[is.na(df)] <- Inf  
  
# The next steps create a complete series of data, filling in gaps in SeriesInd across all categories  
  
# Create a sequence of values starting at the first value of SeriesInd and ending at the last  
dftmp2 <- data.frame(SeriesInd=seq(from=min(df$SeriesInd), to=max(df$SeriesInd)))  
  
# Create a tmp dataframe for categories  
dftmp3 <- data.frame(category=c('S01', 'S02', 'S03', 'S04', 'S05', 'S06'))  
  
# Create dataframe joining the complete set of SeriesInd to the categories  
dftmp4 <- dftmp2 %>%  
 merge(dftmp3, all=T) %>%  
 arrange(SeriesInd, category)  
  
# Join the original dataframe with the dataframe containing the full set of SeriesInd and categories  
df2 <- dftmp4 %>%  
 merge(df, by=c('SeriesInd', 'category'), all.x=T) %>%  
 arrange(SeriesInd, category)  
  
# Summary after creating complete dataframe  
summary(df2)

## SeriesInd category Var01 Var02   
## Min. :40669 Length:14118 Min. : 9.03 Min. : 1339900   
## 1st Qu.:41257 Class :character 1st Qu.:23.36 1st Qu.:12521025   
## Median :41845 Mode :character Median :38.55 Median :21091200   
## Mean :41845 Mean : Inf Mean : Inf   
## 3rd Qu.:42433 3rd Qu.:67.12 3rd Qu.:42518000   
## Max. :43021 Max. : Inf Max. : Inf   
## NA's :4386 NA's :4386   
## Var03 Var05 Var07   
## Min. : 8.82 Min. : 8.99 Min. : 8.92   
## 1st Qu.:22.68 1st Qu.:23.03 1st Qu.:23.04   
## Median :37.83 Median :38.24 Median :38.23   
## Mean : Inf Mean : Inf Mean : Inf   
## 3rd Qu.:66.33 3rd Qu.:66.75 3rd Qu.:66.78   
## Max. : Inf Max. : Inf Max. : Inf   
## NA's :4386 NA's :4386 NA's :4386

# Define which variables should be forecast for each category  
fcvars <- list(c(1, 2), c(2, 3), c(5, 7), c(1, 2), c(2, 3), c(5,7))

Exploratory data analysis

Examine missing values

# Look for missing values now that we have complete values for SeriesInd  
summary(df2)

## SeriesInd category Var01 Var02   
## Min. :40669 Length:14118 Min. : 9.03 Min. : 1339900   
## 1st Qu.:41257 Class :character 1st Qu.:23.36 1st Qu.:12521025   
## Median :41845 Mode :character Median :38.55 Median :21091200   
## Mean :41845 Mean : Inf Mean : Inf   
## 3rd Qu.:42433 3rd Qu.:67.12 3rd Qu.:42518000   
## Max. :43021 Max. : Inf Max. : Inf   
## NA's :4386 NA's :4386   
## Var03 Var05 Var07   
## Min. : 8.82 Min. : 8.99 Min. : 8.92   
## 1st Qu.:22.68 1st Qu.:23.03 1st Qu.:23.04   
## Median :37.83 Median :38.24 Median :38.23   
## Mean : Inf Mean : Inf Mean : Inf   
## 3rd Qu.:66.33 3rd Qu.:66.75 3rd Qu.:66.78   
## Max. : Inf Max. : Inf Max. : Inf   
## NA's :4386 NA's :4386 NA's :4386

# Create df to hold missing value summary  
dfmv <- data.frame(category=c(), var=c())  
  
# Examine missing value count  
for (i in seq(1, 6)) {  
 for (j in seq(1, 2)) {  
 gaps <- sum(is.na(df2[df2$category==paste0('S0', i), paste0('Var0', fcvars[[i]][j])]))  
 inf\_vals <- sum(is.infinite(df2[df2$category==paste0('S0', i), paste0('Var0', fcvars[[i]][j])]))  
 total\_vals <- length(df2[df2$category==paste0('S0', i), paste0('Var0', fcvars[[i]][j])])  
 dfmv <- rbind(dfmv, c(paste0('S0', i), paste0('Var0', fcvars[[i]][j]), inf\_vals, gaps))  
 }  
}  
colnames(dfmv) <- c('category', 'var', 'missing.values', 'gaps')  
dfmv %>%  
 kbl(caption='Time series gaps and missing values') %>%  
 kable\_classic(full\_width=F)

Gap Analysis

We looked for patterns in the data set to determine if there was any seasonality and to evaluate where there were gaps. We assumed SeriesInd was an integer representing the number of days since a certain date, a common one being January 1, 1900 (the “origin” date). Using this logic, we converted SeriesInd to a date and examined where gaps occurred. Using an origin date of January 1, 1900 placed gaps on Friday and Saturday. It seemed more reasonable to assume that the series excluded weekend days, so we shifted the weekdays such that the gaps would fall on weekends.

It is also noted that a number of gaps fall on other weekdays, with the greatest number on Monday (32). This is consistent with a US-based calendar which includes a number of national holidays that fall on Monday (e.g. Memorial Day and Labor Day).

Because an origin date of January 1, 1900 yielded a week with gaps on Fridays and Saturdays, we performed a systematic search to find a reasonable origin that would align the gaps with Saturday and Sunday instead. Going under the assumption that this was a US calendar, we looked for origin dates that would place gaps on both January 1 and on December 25. The first such origin date that met these criteria was August 31, 1915, which would put our first data point in the year 2027. Therefore, we concluded that the data set either does not likely conform to a US calendar. We further concluded that the importance of the origin date was secondary and that the key observation was that the data is likely based on a seven-day week, with regular gaps on exactly two of those days. It is possible these are stock market prices, sales figures, or another such weekly metric.

We also performed a gap analysis on the prediction set to aid in determining whether “weekend” values would need to be imputed (filled). As shown in the table, there are no weekend days in the prediction set.

# Look for patterns in gaps in SeriesInd; first make a copy of df == dfga (gap analysis).  
# Filter by just a single category since this will give us a single set of SeriesInd values,  
# and the NAs in each category will all be in the same positions so it doesn't matter which category we choose.  
# Work from the theory that these are days of the week, and that that SeriesInd is the number of days since Jan 1, 1900.  
dfga <- df2 %>%  
 filter(category=='S01') %>%  
 mutate(date=as.Date(SeriesInd, origin='1900-01-01')+1) %>%  
 mutate(SeriesInd.mod7=SeriesInd %% 7) %>%  
 mutate(Day.of.week=weekdays(date)) %>%  
 group\_by(SeriesInd.mod7, Day.of.week) %>%  
 summarize(Gaps=sum(is.na(Var01)), Filled.vals=sum(!is.na(Var01)), .groups='keep') %>%  
 ungroup() %>%  
 arrange(SeriesInd.mod7) %>%  
 select(-SeriesInd.mod7)  
dfga %>%  
 kbl(caption='Gap analysis') %>%  
 kable\_classic(full\_width = F)

# Look at gaps in the SeriesInd values we need to predict;  
# first fill in values with -1 so we know which ones are missing  
df3 <- df\_orig %>%  
 filter(SeriesInd > 43021) %>%  
 arrange(SeriesInd, category) %>%  
 select(SeriesInd, category, Var01) %>%  
 mutate(Var01=-1)  
  
# Create a sequence of values starting at the first value of SeriesInd and ending at the last  
dftmp2 <- data.frame(SeriesInd=seq(from=min(df3$SeriesInd), to=max(df3$SeriesInd)))  
  
# Create a tmp dataframe for categories  
dftmp3 <- data.frame(category=c('S01', 'S02', 'S03', 'S04', 'S05', 'S06'))  
  
# Create dataframe joining the complete set of SeriesInd to the categories  
dftmp4 <- dftmp2 %>%  
 merge(dftmp3, all=T) %>%  
 arrange(SeriesInd, category)  
  
# Join the original dataframe with the dataframe containing the full set of SeriesInd and categories  
df3 <- dftmp4 %>%  
 merge(df3, by=c('SeriesInd', 'category'), all.x=T) %>%  
 arrange(SeriesInd, category)  
  
# Examine gaps  
dfga2 <- df3 %>%  
 filter(category=='S01') %>%  
 mutate(date=as.Date(SeriesInd, origin='1900-01-01')+1) %>%  
 mutate(SeriesInd.mod7=SeriesInd %% 7) %>%  
 mutate(Day.of.week=weekdays(date)) %>%  
 group\_by(SeriesInd.mod7, Day.of.week) %>%  
 summarize(Gaps=sum(is.na(Var01)), Filled.vals=sum(!is.na(Var01)), .groups='keep') %>%  
 ungroup() %>%  
 arrange(SeriesInd.mod7) %>%  
 select(-SeriesInd.mod7)  
dfga2 %>%  
 kbl(caption='Gap analysis - prediction set') %>%  
 kable\_classic(full\_width = F)

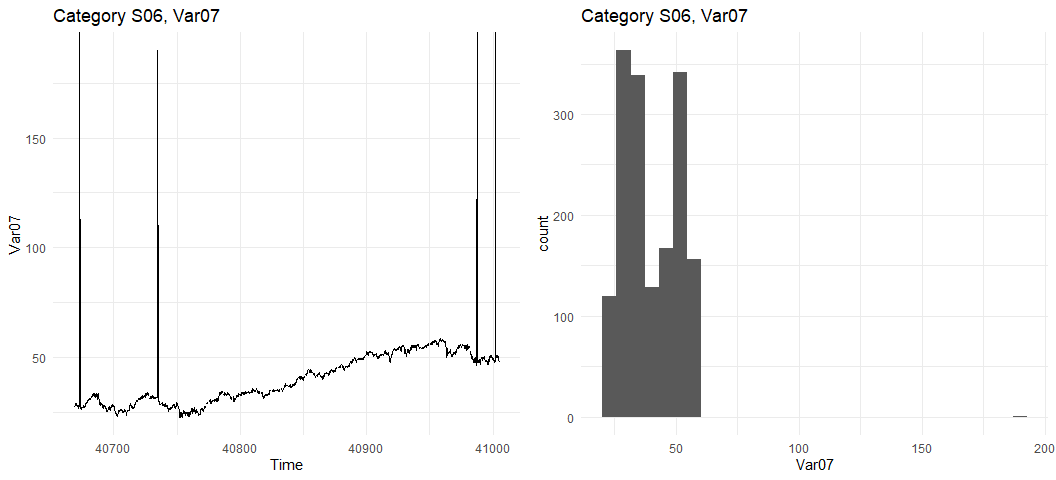
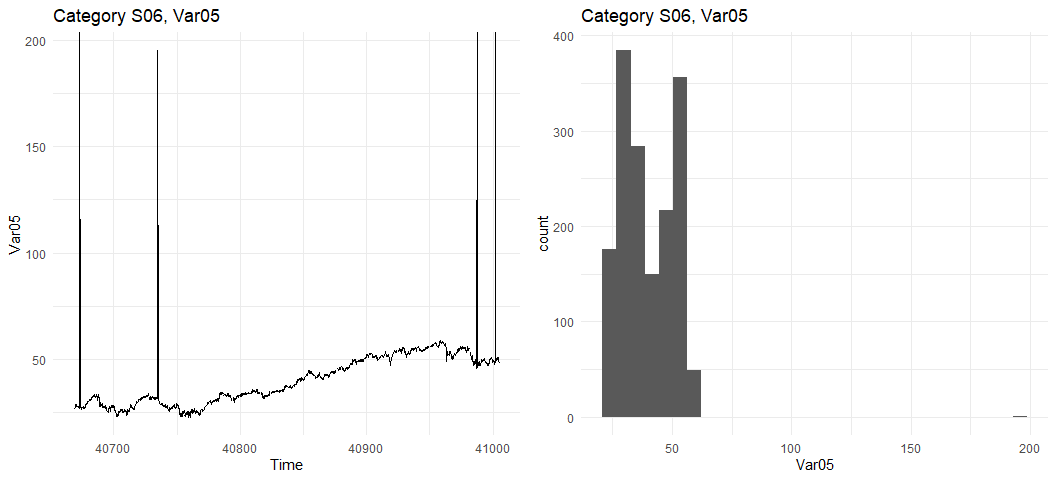
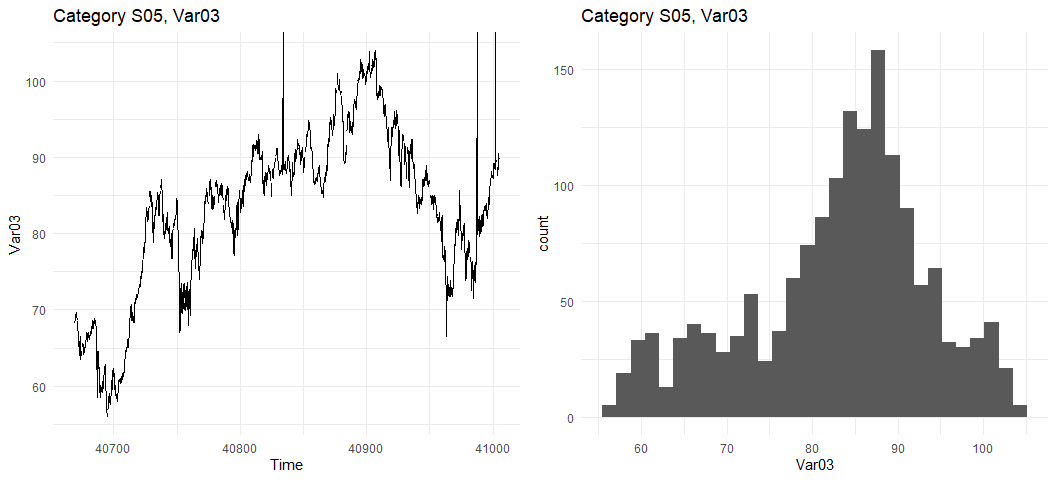
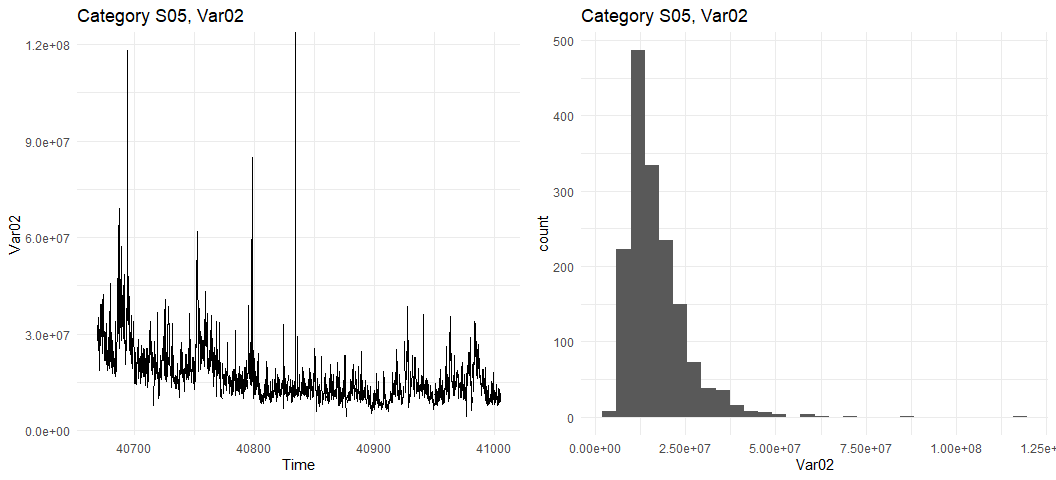
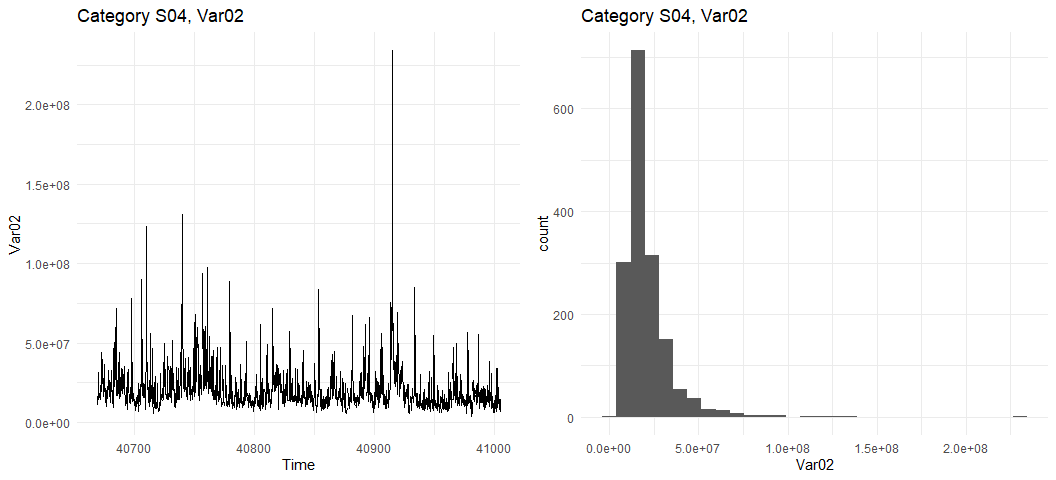
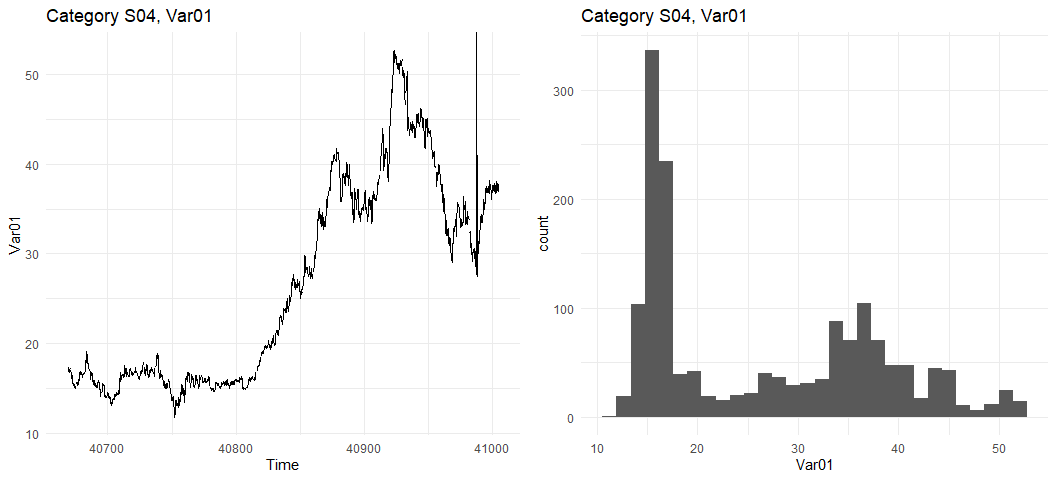
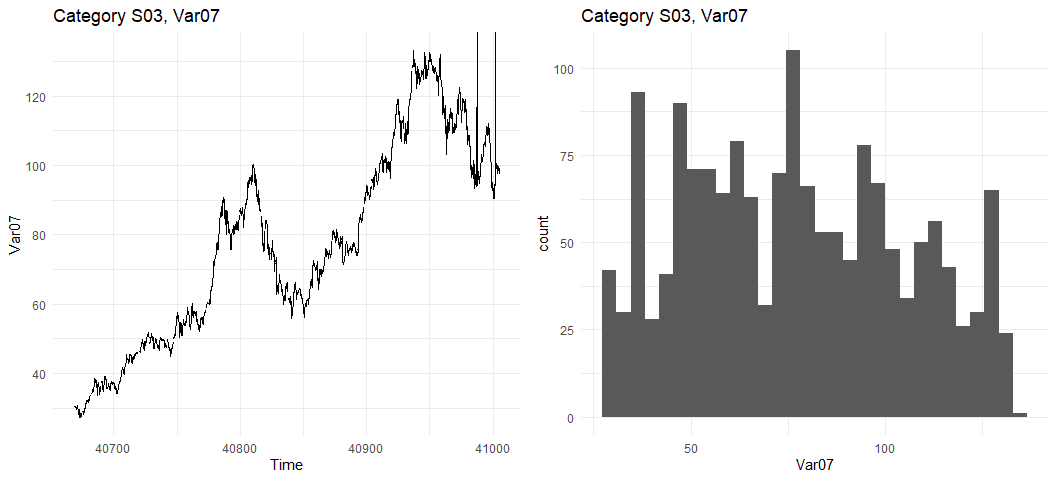
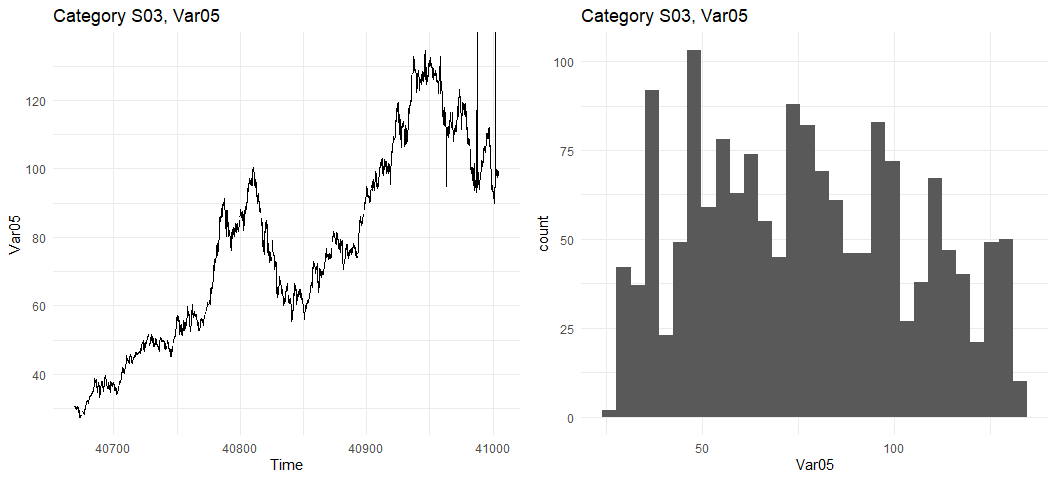
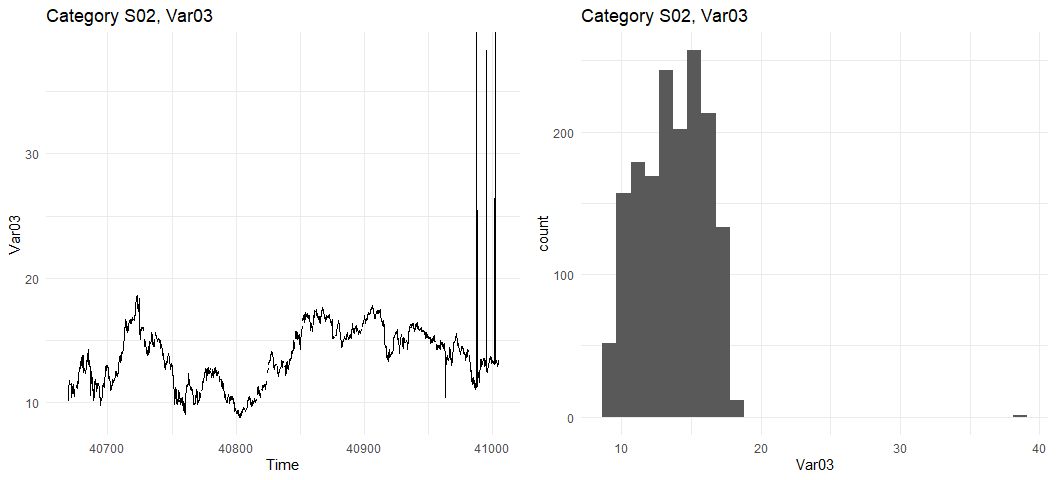
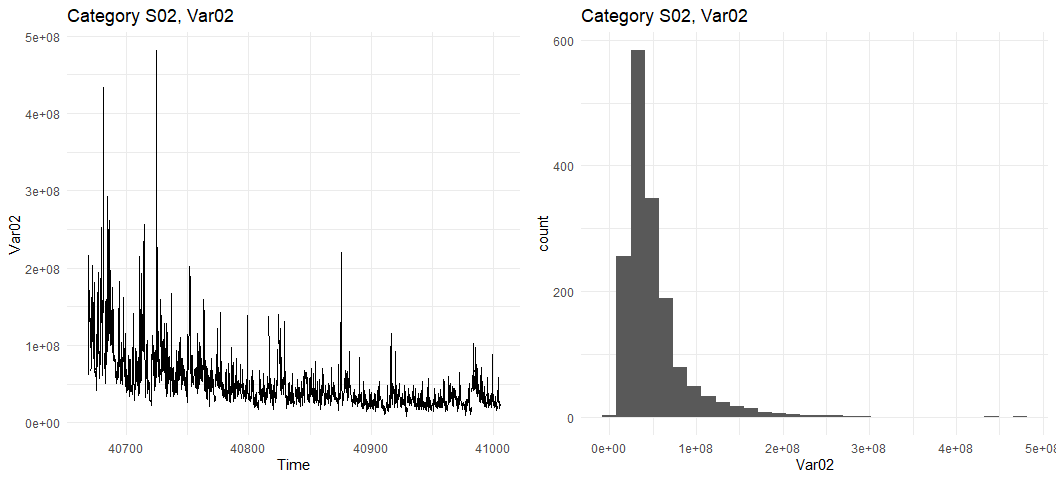
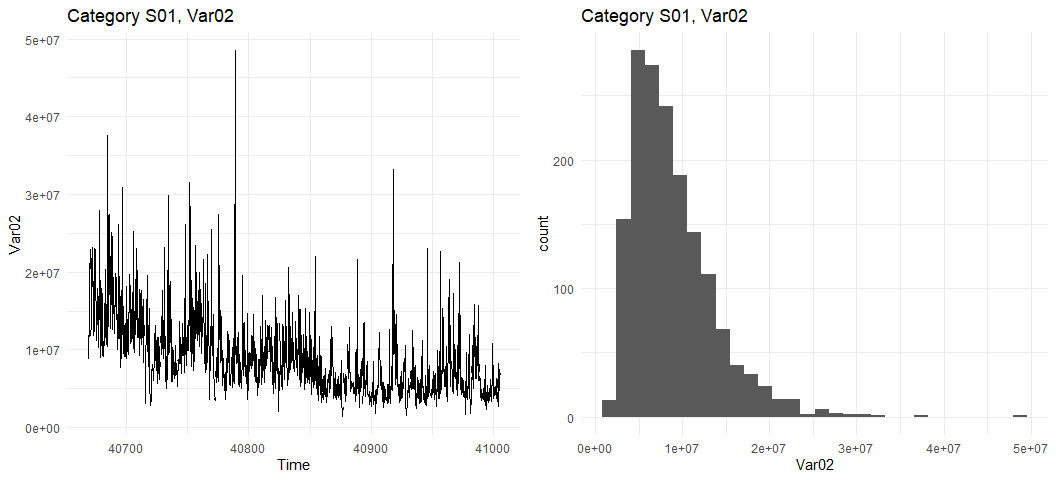
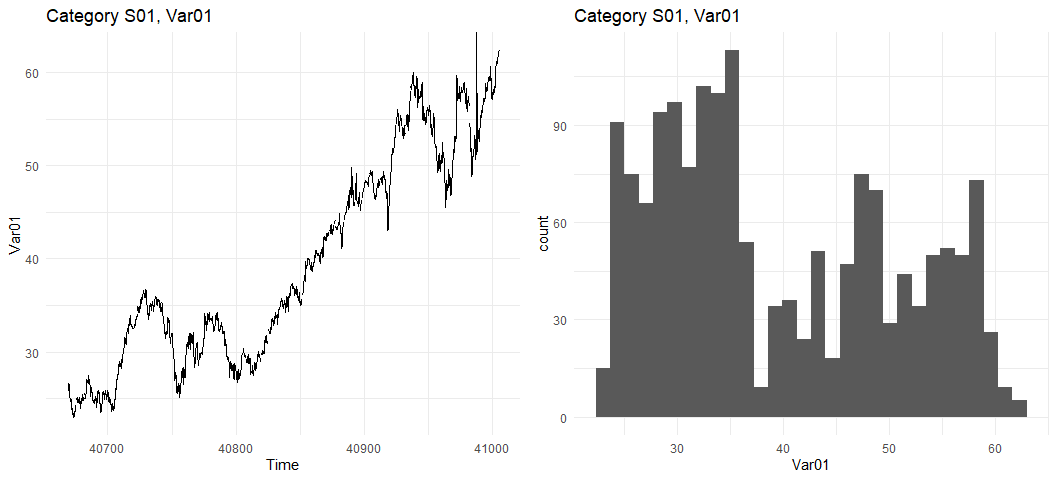
# Create full 5-day set of SeriesInds  
df4 <- df2 %>%  
 filter(category=='S01') %>%  
 mutate(date=as.Date(SeriesInd, origin='1900-01-01')+1) %>%  
 mutate(SeriesInd.mod7=SeriesInd %% 7) %>%  
 mutate(Day.of.week=weekdays(date)) %>%  
 filter(Day.of.week != 'Saturday' & Day.of.week != 'Sunday') %>%  
 select(SeriesInd, category)  
for (i in df4$SeriesInd) {  
 for (j in seq(2, 6)) {  
 df4 <- rbind(df4, data.frame(SeriesInd=i, category=paste0('S0', j)))  
 }  
}  
  
# Join the original dataframe with the dataframe containing the full set of SeriesInd and categories  
df2 <- df %>%  
 merge(df4, by=c('SeriesInd', 'category'), all=T) %>%  
 arrange(SeriesInd, category)  
summary(df2)

## SeriesInd category Var01 Var02   
## Min. :40669 Length:10086 Min. : 9.03 Min. : 1339900   
## 1st Qu.:41257 Class :character 1st Qu.:23.36 1st Qu.:12521025   
## Median :41845 Mode :character Median :38.55 Median :21091200   
## Mean :41844 Mean : Inf Mean : Inf   
## 3rd Qu.:42433 3rd Qu.:67.12 3rd Qu.:42518000   
## Max. :43021 Max. : Inf Max. : Inf   
## NA's :354 NA's :354   
## Var03 Var05 Var07   
## Min. : 8.82 Min. : 8.99 Min. : 8.92   
## 1st Qu.:22.68 1st Qu.:23.03 1st Qu.:23.04   
## Median :37.83 Median :38.24 Median :38.23   
## Mean : Inf Mean : Inf Mean : Inf   
## 3rd Qu.:66.33 3rd Qu.:66.75 3rd Qu.:66.77   
## Max. : Inf Max. : Inf Max. : Inf   
## NA's :354 NA's :354 NA's :354

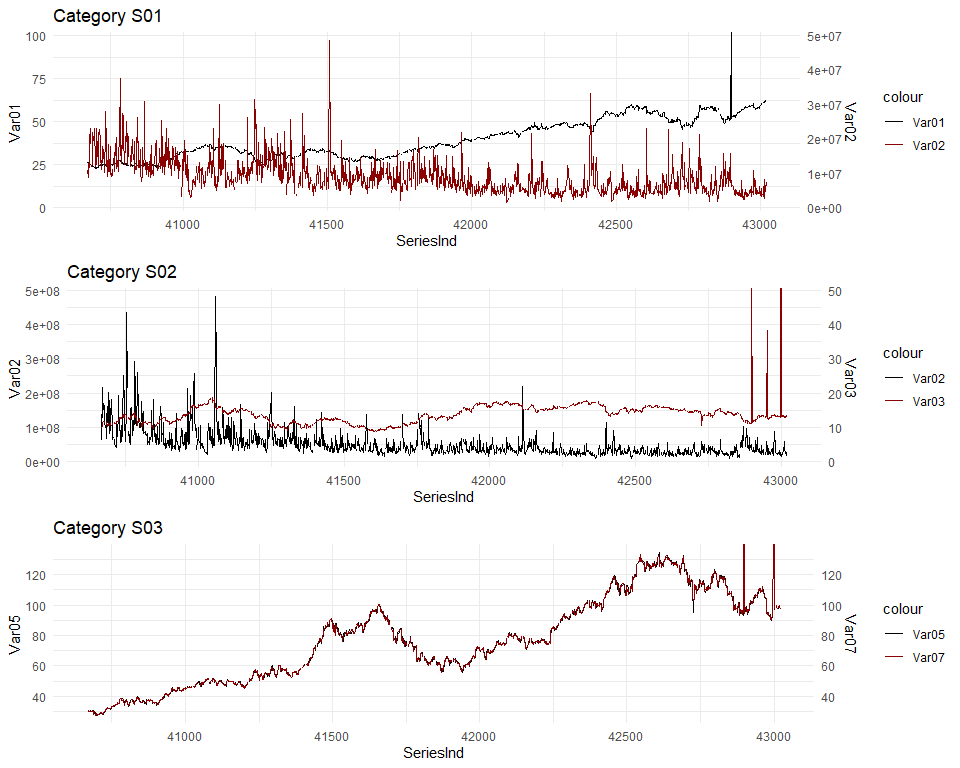
### 

# To work with the data more easily, initialize list to hold dataframes, one for each category  
dfcat = list()  
  
# Filter by category, selecting only those vars we're interested in for that category  
for (i in seq(1, 6)) {  
 dfcat[[i]] <- df2 %>%  
 filter(category == paste0('S0', i)) %>%  
 select(SeriesInd, !!paste0('Var0', fcvars[[i]][1]), !!paste0('Var0', fcvars[[i]][2]))  
}  
  
# Initialize ts objects; each element in each list will correspond to a category; e.g. ts1[[1]] will be S01, etc  
ts1 = list()  
ts2 = list()  
  
# Set start date to first date in series  
start\_date <- df2$SeriesInd[[1]]  
  
# Iterate over categories  
for (i in seq(1, 6)) {  
  
 # Create var names for the two variables we're interested in for this category  
 varname1 <- paste0('Var0', fcvars[[i]][1])  
 varname2 <- paste0('Var0', fcvars[[i]][2])  
  
 # Create time series for each variable  
 #ts1[[i]] <- ts(dfcat[[i]][dfcat[[i]][varname1] != Inf, varname1], frequency=the\_freq, start=start\_date)  
 #ts2[[i]] <- ts(dfcat[[i]][dfcat[[i]][varname2] != Inf, varname2], frequency=the\_freq, start=start\_date)  
 ts1[[i]] <- ts(dfcat[[i]][[varname1]], frequency=the\_freq, start=start\_date)  
 ts2[[i]] <- ts(dfcat[[i]][[varname2]], frequency=the\_freq, start=start\_date)  
  
 # Plot the time series  
 p1a <- ts1[[i]] %>%  
 autoplot() +  
 ggtitle(paste0('Category S0', i, ', ', varname1)) +  
 ylab(varname1)  
 p1b <- dfcat[[i]] %>%  
 filter(dfcat[[i]][!!varname1] != Inf & !is.na(dfcat[[i]][!!varname1])) %>%  
 ggplot() +  
 geom\_histogram(aes(x=eval(sym(varname1))), bins=30) +  
 ggtitle(paste0('Category S0', i, ', ', varname1)) +  
 xlab(varname1)  
 grid.arrange(p1a, p1b, ncol=2)  
 p2a <- ts2[[i]] %>%  
 autoplot() +  
 ggtitle(paste0('Category S0', i, ', ', varname2)) +  
 ylab(varname2)  
 p2b <- dfcat[[i]] %>%  
 filter(dfcat[[i]][!!varname2] != Inf & !is.na(dfcat[[i]][!!varname2])) %>%  
 ggplot() +  
 geom\_histogram(aes(x=eval(sym(varname2))), bins=30) +  
 ggtitle(paste0('Category S0', i, ', ', varname2)) +  
 xlab(varname2)  
 grid.arrange(p2a, p2b, ncol=2)  
  
}

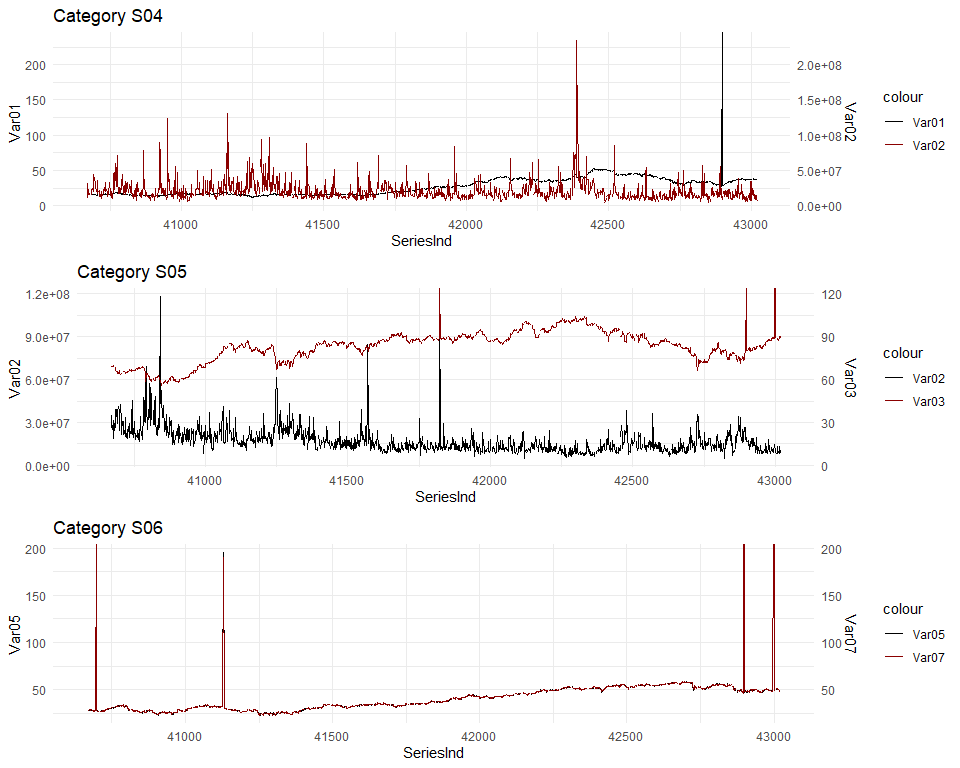
## Warning: Using one column matrices in `filter()` was deprecated in dplyr 1.1.0.  
## ℹ Please use one dimensional logical vectors instead.  
## ℹ The deprecated feature was likely used in the dplyr package.  
## Please report the issue at <]8;;https://github.com/tidyverse/dplyr/issueshttps://github.com/tidyverse/dplyr/issues]8;;>.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.



# Plot time series on one graph  
p1 <- dfcat[[1]] %>%  
 ggplot(aes(x=SeriesInd)) +  
 geom\_line(aes(y=Var01, color='Var01')) +  
 geom\_line(aes(y=Var02 / 500000, color='Var02')) +  
 scale\_y\_continuous(sec.axis=sec\_axis(~ . \* 500000, name='Var02')) +   
 scale\_color\_manual(values=c('black', 'darkred')) +  
 ggtitle('Category S01')  
p2 <- dfcat[[2]] %>%  
 ggplot(aes(x=SeriesInd)) +  
 geom\_line(aes(y=Var02, color='Var02')) +  
 geom\_line(aes(y=Var03 \* 10000000, color='Var03')) +  
 scale\_y\_continuous(sec.axis=sec\_axis(~ . / 10000000, name='Var03')) +   
 scale\_color\_manual(values=c('black', 'darkred')) +  
 ggtitle('Category S02')  
p3 <- dfcat[[3]] %>%  
 ggplot(aes(x=SeriesInd)) +  
 geom\_line(aes(y=Var05, color='Var05')) +  
 geom\_line(aes(y=Var07, color='Var07')) +  
 scale\_y\_continuous(sec.axis=sec\_axis(~ ., name='Var07')) +   
 scale\_color\_manual(values=c('black', 'darkred')) +  
 ggtitle('Category S03')  
p4 <- dfcat[[4]] %>%  
 ggplot(aes(x=SeriesInd)) +  
 geom\_line(aes(y=Var01, color='Var01')) +  
 geom\_line(aes(y=Var02 / 1000000, color='Var02')) +  
 scale\_y\_continuous(sec.axis=sec\_axis(~ . \* 1000000, name='Var02')) +   
 scale\_color\_manual(values=c('black', 'darkred')) +  
 ggtitle('Category S04')  
p5 <- dfcat[[5]] %>%  
 ggplot(aes(x=SeriesInd)) +  
 geom\_line(aes(y=Var02, color='Var02')) +  
 geom\_line(aes(y=Var03 \* 1000000, color='Var03')) +  
 scale\_y\_continuous(sec.axis=sec\_axis(~ . / 1000000, name='Var03')) +   
 scale\_color\_manual(values=c('black', 'darkred')) +  
 ggtitle('Category S05')  
p6 <- dfcat[[6]] %>%  
 ggplot(aes(x=SeriesInd)) +  
 geom\_line(aes(y=Var05, color='Var05')) +  
 geom\_line(aes(y=Var07, color='Var07')) +  
 scale\_y\_continuous(sec.axis=sec\_axis(~ ., name='Var07')) +   
 scale\_color\_manual(values=c('black', 'darkred')) +  
 ggtitle('Category S06')  
#grid.arrange(p1, p2, p3, p4, p5, p6, nrow=6, ncol=1)  
grid.arrange(p1, p2, p3, nrow=3, ncol=1)

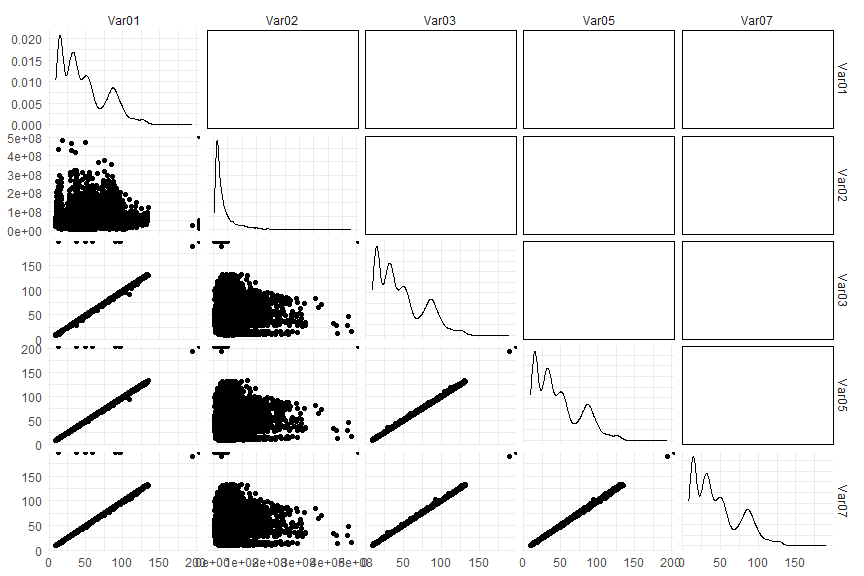


grid.arrange(p4, p5, p6, nrow=3, ncol=1)



# Look for correlation between variables  
GGally::ggpairs(df2[,3:7], progress=F)

## Registered S3 method overwritten by 'GGally':  
## method from   
## +.gg ggplot2

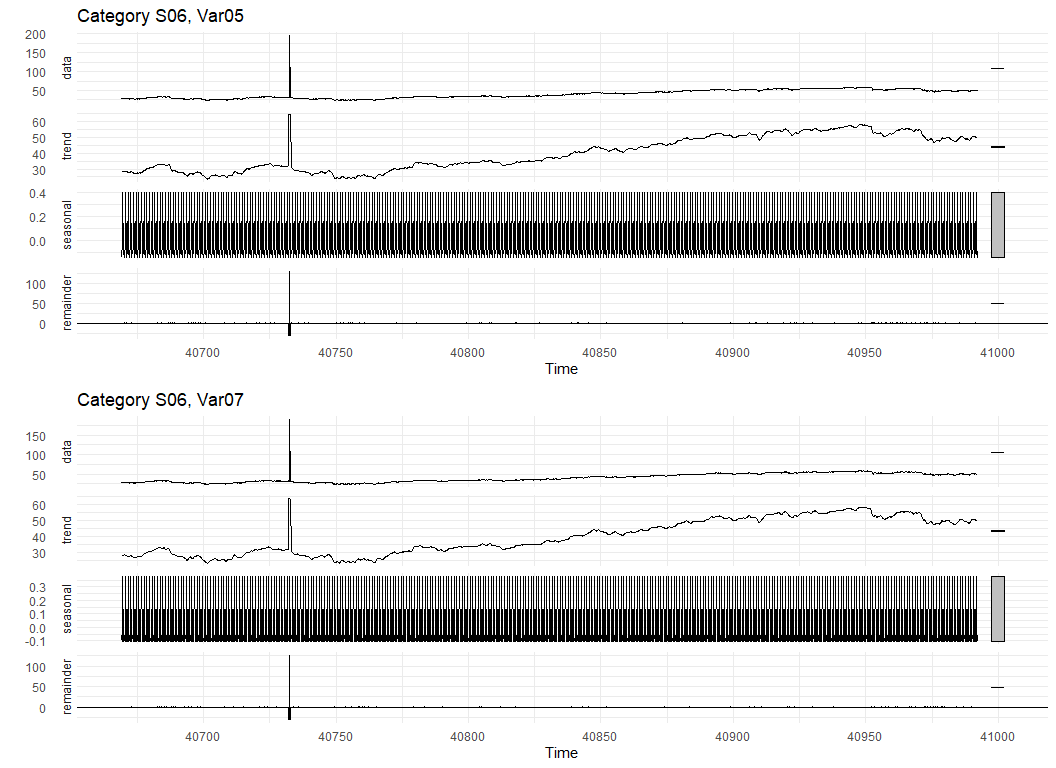
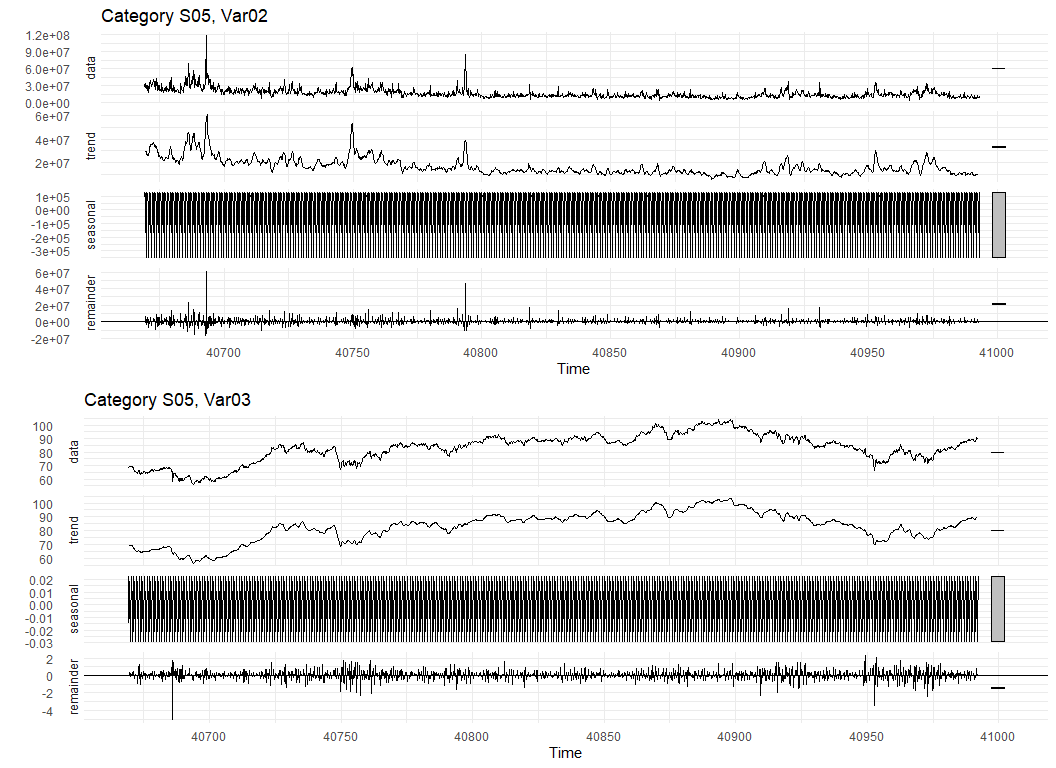
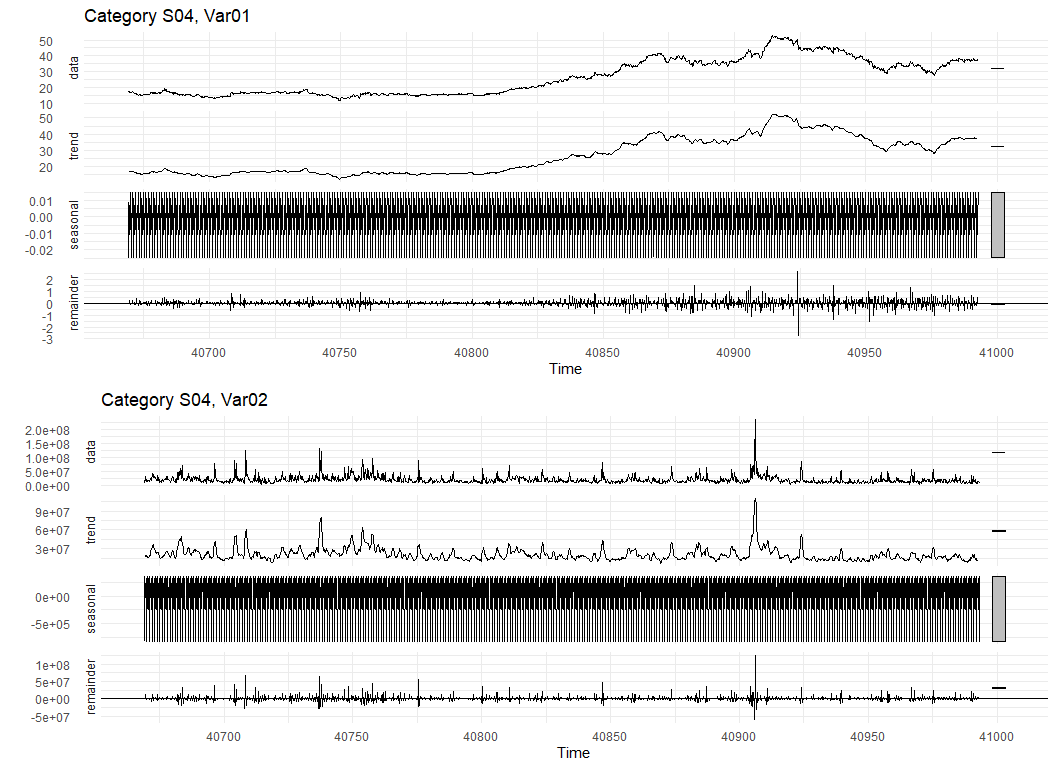
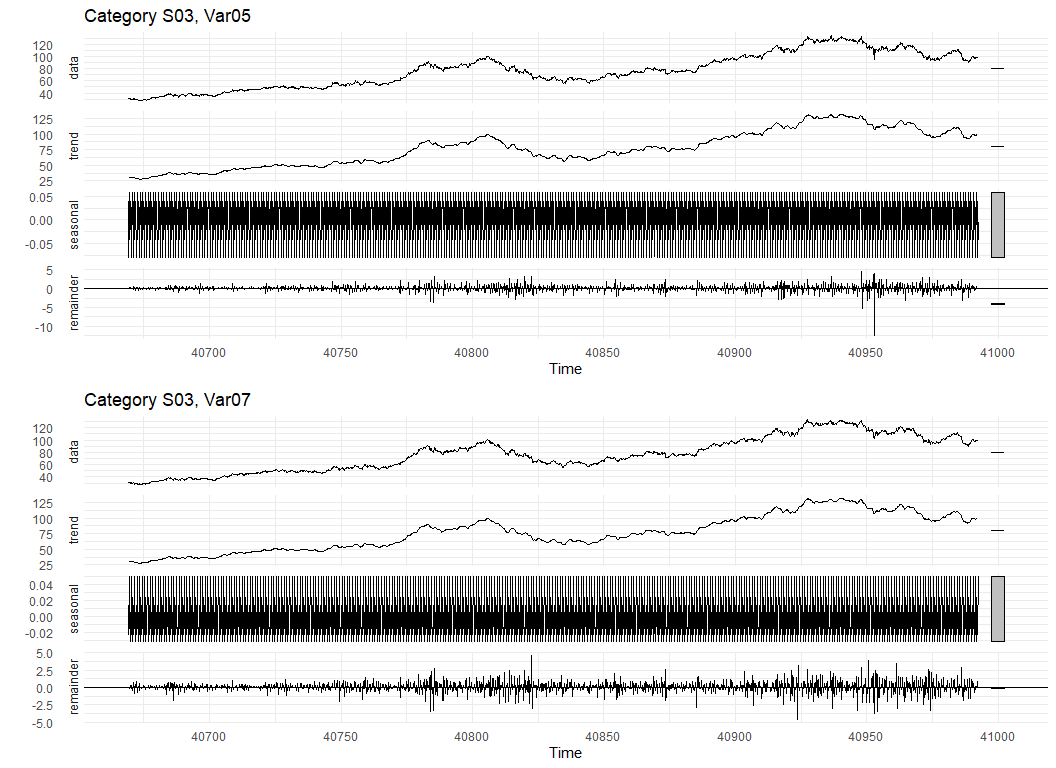
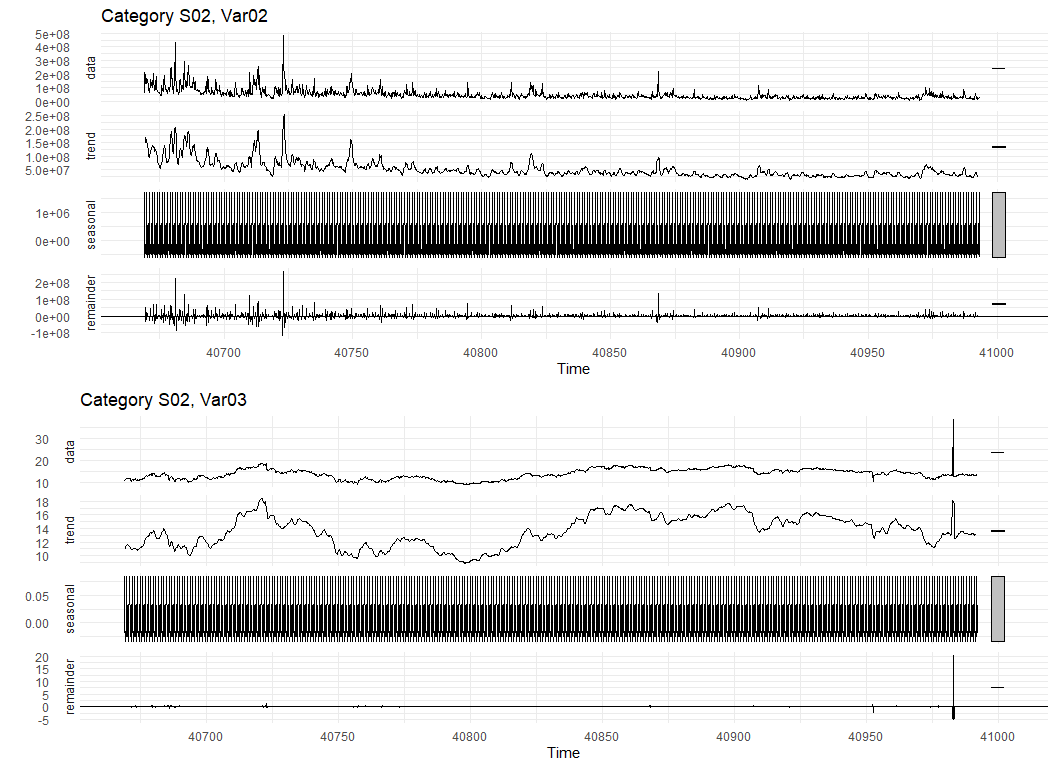
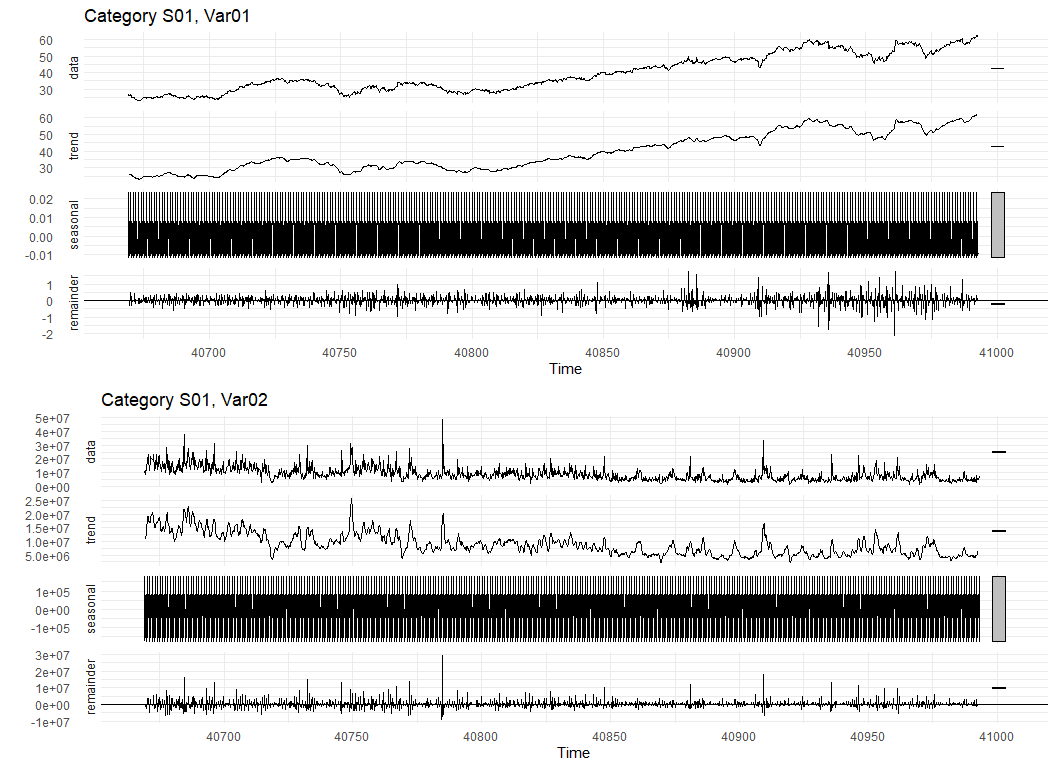


An additional step in exploratory data analysis that is often helpful to better understand the data set is to generate pairwise plots showing the relationship between predictors. Based on these pairwise plots (see below), we observed an extremely high degree of correlation between some predictors. Notably, variables Var01, Var03, Var05, and Var07 contain very similar values, suggesting that any missing values of one might be imputed using existing values of another. Without the context of what these variables represent, it is difficult to speculate on why they might be correlated as such, but it is possible that the data were collected by different observers using similar but slightly different methodologies or techniques.

We discovered some obvious outliers in the data. Outliers adversely affect forecasts and, as such, should be either removed or replaced. Notably,outliers were discovered in the following variables, with four other variables having questionable values.

One such example is in Category S06, variable V07, which clearly exhibits an outlying value of approximately 190 about a quarter of the way into the series.

# Create list to hold outlier counts  
dfout <- data.frame()  
  
# Look for outliers using decomposition plots  
for (i in seq(1, 6)) {  
  
 # Create var names for the two variables we're interested in for this category  
 varname1 <- paste0('Var0', fcvars[[i]][1])  
 varname2 <- paste0('Var0', fcvars[[i]][2])  
   
 # Filter out infinite values  
 dftmp1 <- dfcat[[i]] %>%  
 filter(!is.infinite(eval(sym(varname1)))) %>%  
 filter(!is.na(eval(sym(varname1))))  
 dftmp2 <- dfcat[[i]] %>%  
 filter(!is.infinite(eval(sym(varname2)))) %>%  
 filter(!is.na(eval(sym(varname2))))  
   
 # Create time series for each variable using xts; SeriesInd appears to be the days since 1900-01-01  
 ts1[[i]] <- ts(dftmp1[[varname1]], frequency=the\_freq, start=start\_date)  
 ts2[[i]] <- ts(dftmp2[[varname2]], frequency=the\_freq, start=start\_date)  
  
 if (the\_freq > 1) {  
   
 p1 <- ts1[[i]] %>%  
 decompose(type='additive') %>%  
 autoplot() +  
 ggtitle(paste0('Category S0', i, ', Var0', fcvars[[i]][1]))  
 p2 <- ts2[[i]] %>%  
 decompose(type='additive') %>%  
 autoplot() +  
 ggtitle(paste0('Category S0', i, ', Var0', fcvars[[i]][2]))  
 grid.arrange(p1, p2, ncol=1, nrow=2)  
   
 }  
   
 # Count of outliers > 3 SD  
 outct <- length(ts1[[i]][ts1[[i]] > mean(ts1[[i]]) + 3 \* sd(ts1[[i]]) | ts1[[i]] < mean(ts1[[i]]) - 3 \* sd(ts1[[i]])])  
 dfout <- rbind(dfout, data.frame(  
 Category=paste0('S0', i),  
 Variable=paste0('Var0', fcvars[[i]][1]),  
 Outliers=outct  
 ))  
 outct <- length(ts2[[i]][ts2[[i]] > mean(ts2[[i]]) + 3 \* sd(ts2[[i]]) | ts2[[i]] < mean(ts2[[i]]) - 3 \* sd(ts2[[i]])])  
 dfout <- rbind(dfout, data.frame(  
 Category=paste0('S0', i),  
 Variable=paste0('Var0', fcvars[[i]][2]),  
 Outliers=outct  
 ))  
   
}

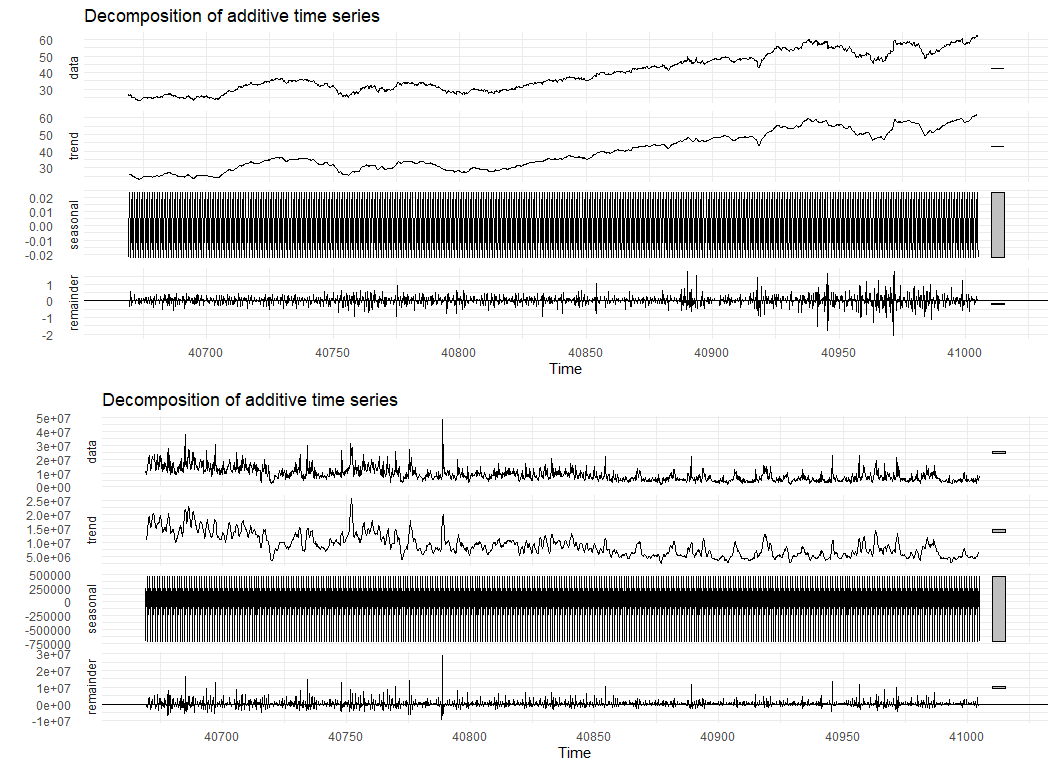


# Show outlier count  
dfout %>%  
 kbl(caption='Outliers beyond 3 standard deviations from the mean') %>%  
 kable\_classic(full\_width=F)

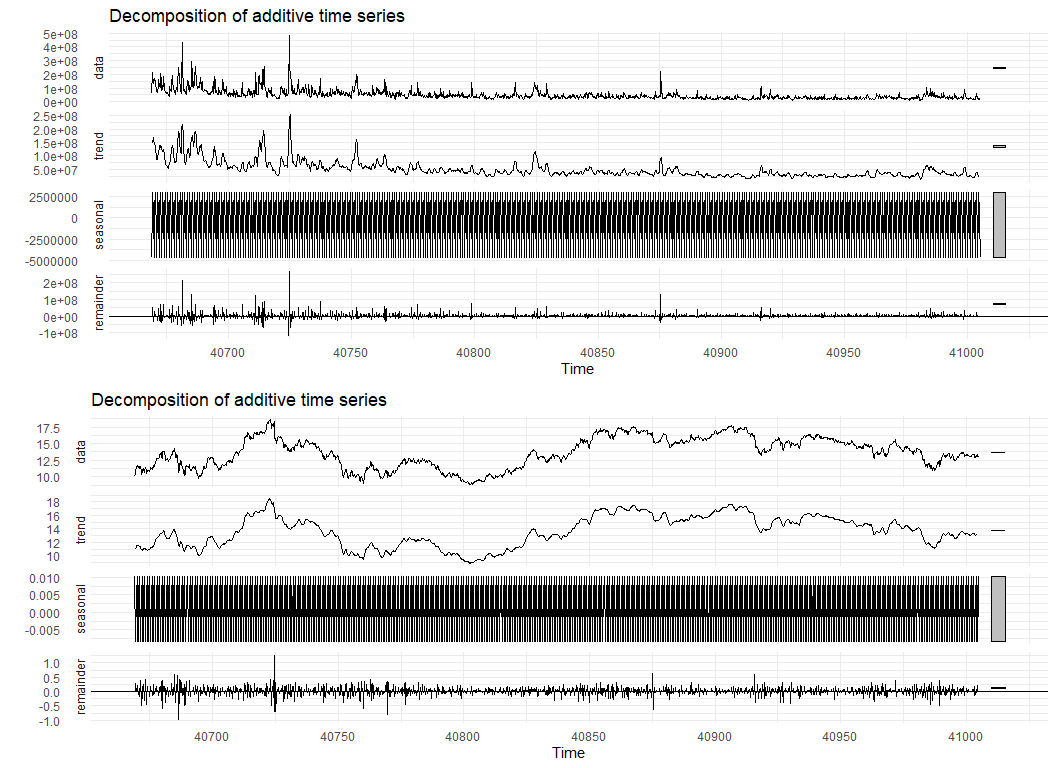
### 

# Create new ts objects that will have imputed values  
tsnew1 <- list()  
tsnew2 <- list()  
  
for(i in seq(1, 6)) {  
   
 # Create var names for the two variables we're interested in for this category  
 varname1 <- paste0('Var0', fcvars[[i]][1])  
 varname2 <- paste0('Var0', fcvars[[i]][2])  
   
 # Convert Infs to NAs  
 dftmp <- dfcat[[i]]  
 print(paste0(" changing ", sum(is.infinite(dftmp[[varname1]])), " infinite values to NA for ", varname1))  
 print(paste0(" changing ", sum(is.infinite(dftmp[[varname2]])), " infinite values to NA for ", varname2))  
 dftmp[varname1] = ifelse(is.infinite(dftmp[[varname1]]), NA, dftmp[[varname1]])  
 dftmp[varname2] = ifelse(is.infinite(dftmp[[varname2]]), NA, dftmp[[varname2]])  
  
 # Create time series for each variable  
 tsnew1[[i]] <- ts(dftmp[[varname1]], frequency=the\_freq, start=start\_date)  
 tsnew2[[i]] <- ts(dftmp[[varname2]], frequency=the\_freq, start=start\_date)  
  
 # Replace missing values  
 tsnew1[[i]] <- tsnew1[[i]] %>%  
 tsclean(replace.missing=T, lambda='auto')  
 tsnew2[[i]] <- tsnew2[[i]] %>%  
 tsclean(replace.missing=T, lambda='auto')  
  
 if (the\_freq > 1) {  
   
 # Decomp plots  
 p1 <- tsnew1[[i]] %>%  
 decompose(type='additive') %>%  
 autoplot()  
 p2 <- tsnew2[[i]] %>%  
 decompose(type='additive') %>%  
 autoplot()  
 grid.arrange(p1, p2, ncol=1, nrow=2)  
   
 }  
  
}

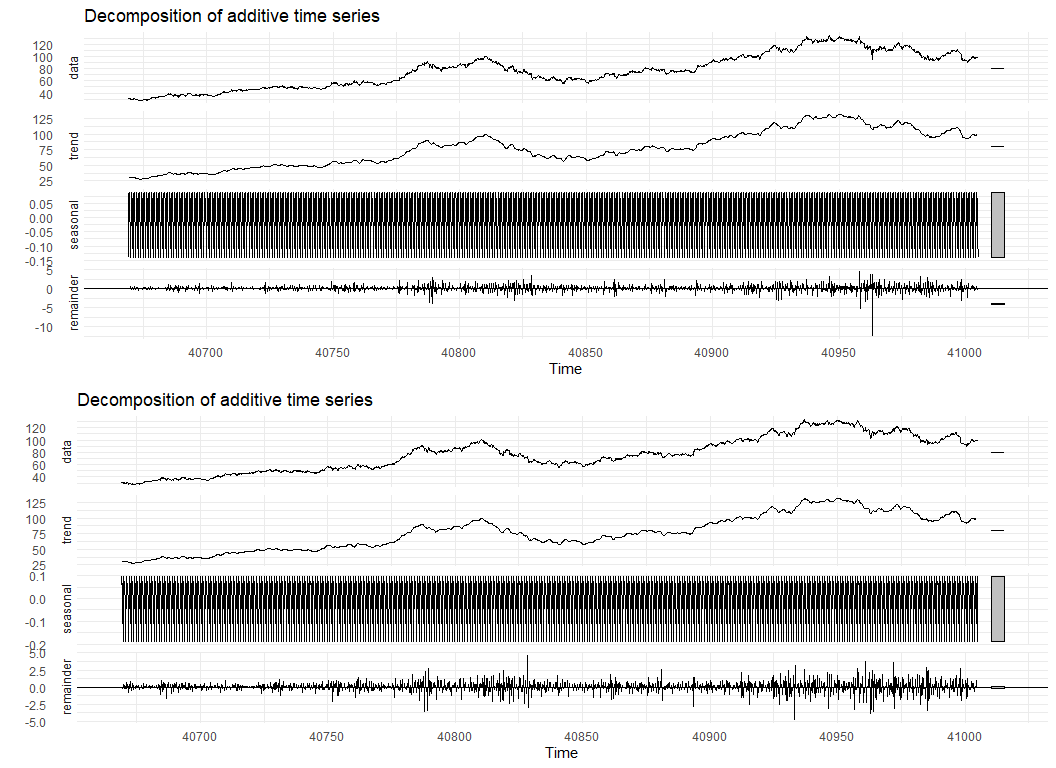
## [1] " changing 2 infinite values to NA for Var01"  
## [1] " changing 0 infinite values to NA for Var02"



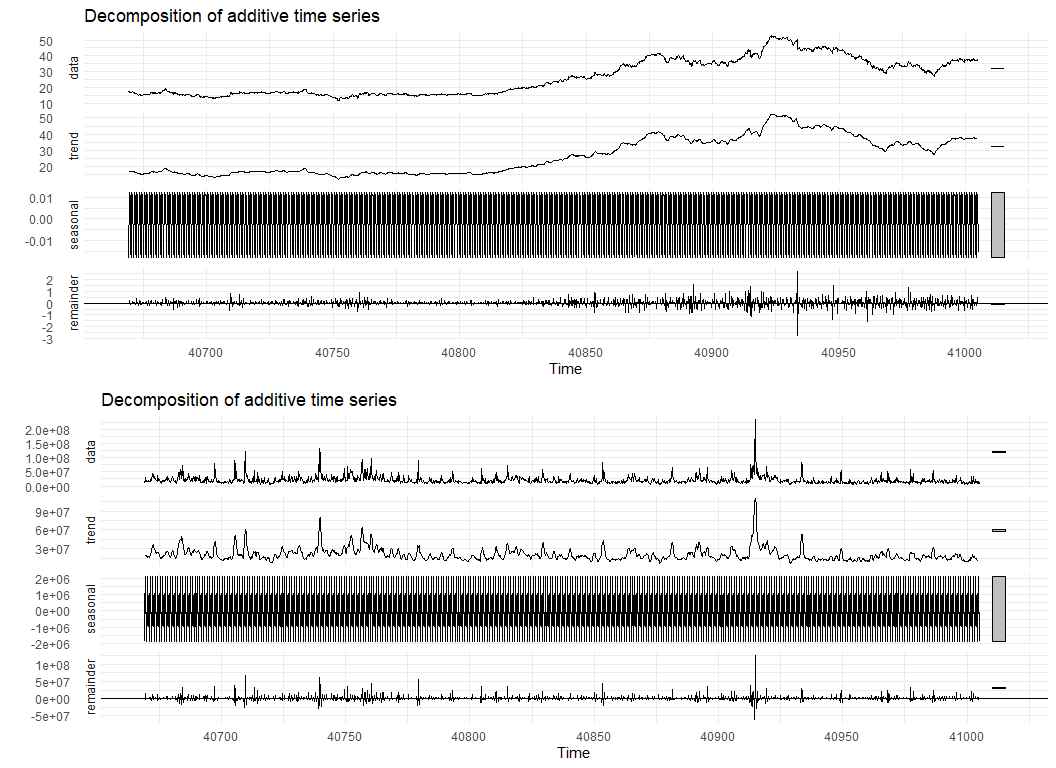
## [1] " changing 0 infinite values to NA for Var02"  
## [1] " changing 4 infinite values to NA for Var03"



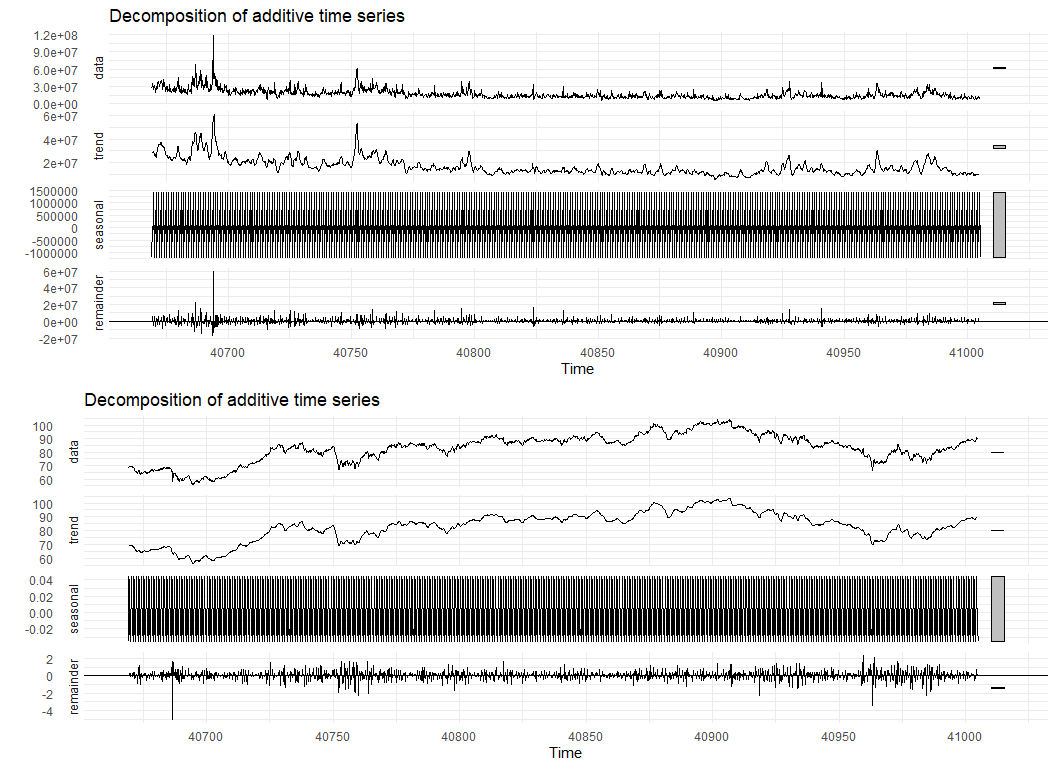
## [1] " changing 4 infinite values to NA for Var05"  
## [1] " changing 4 infinite values to NA for Var07"



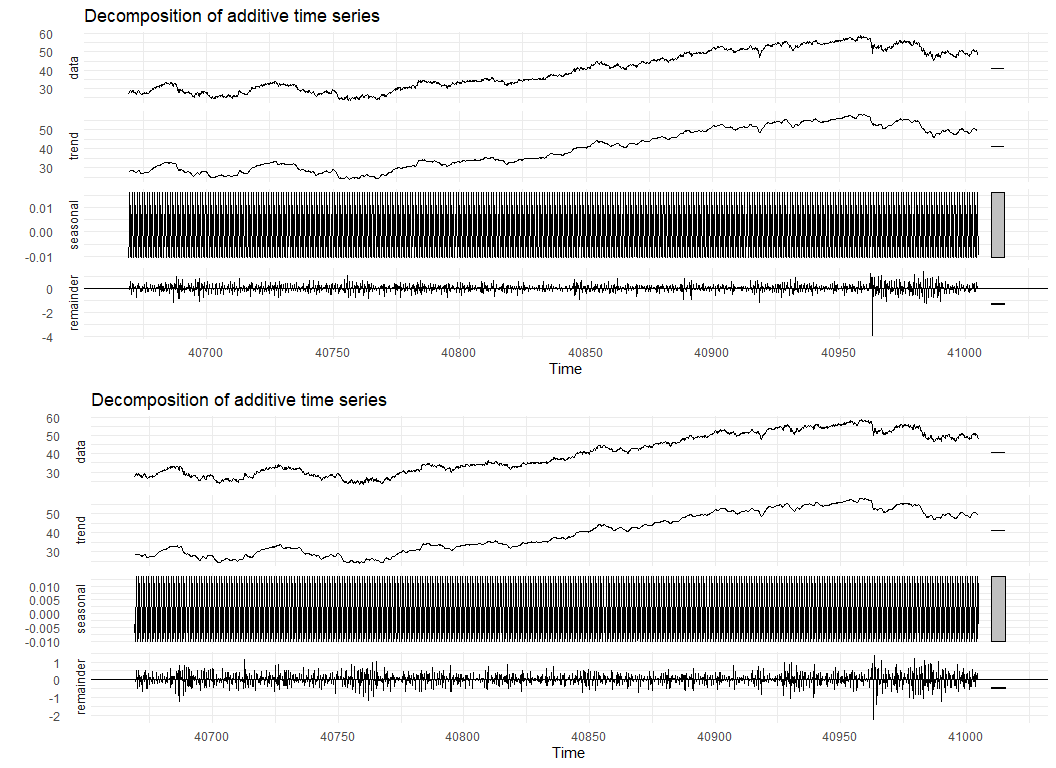
## [1] " changing 2 infinite values to NA for Var01"  
## [1] " changing 0 infinite values to NA for Var02"



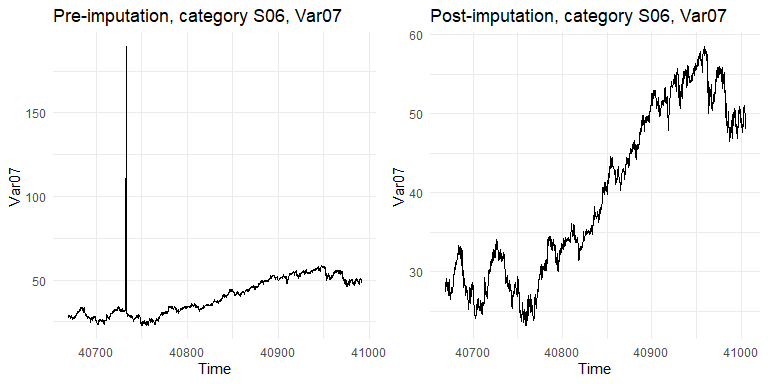
## [1] " changing 1 infinite values to NA for Var02"  
## [1] " changing 5 infinite values to NA for Var03"



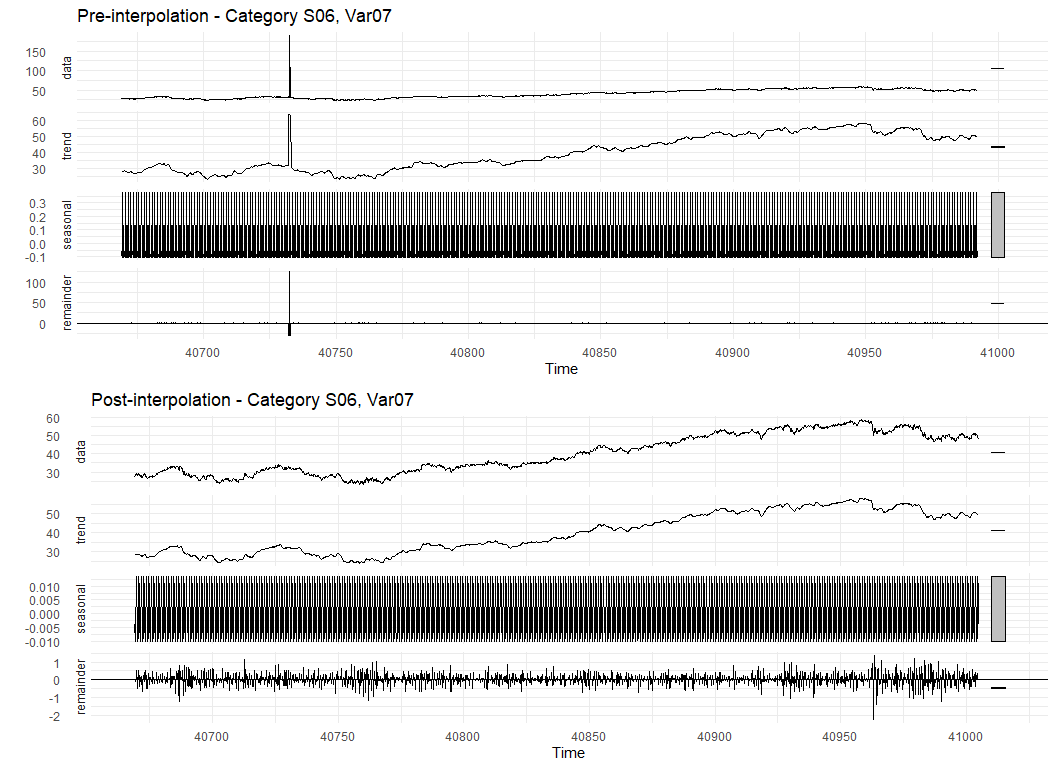
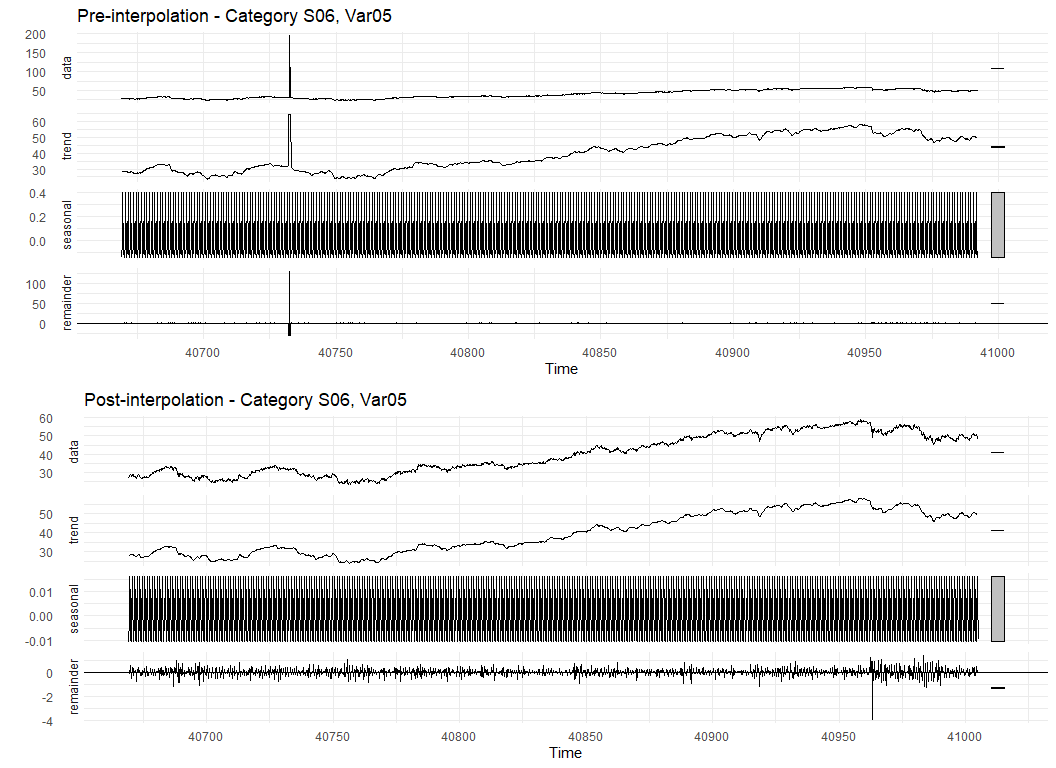
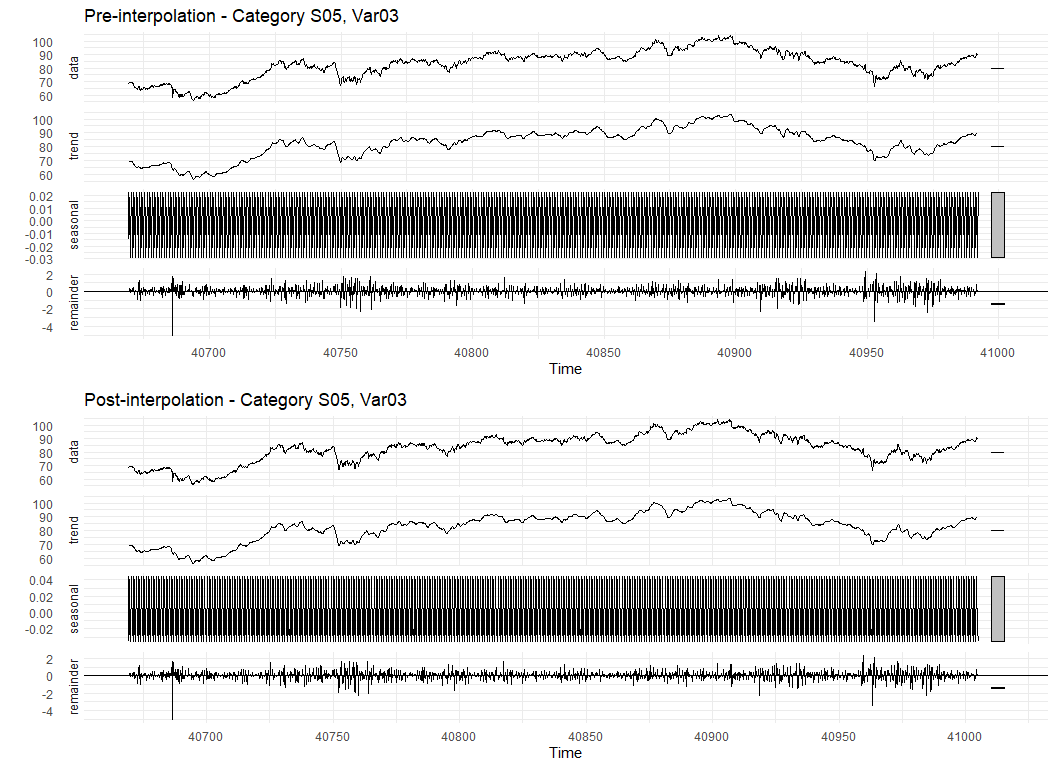
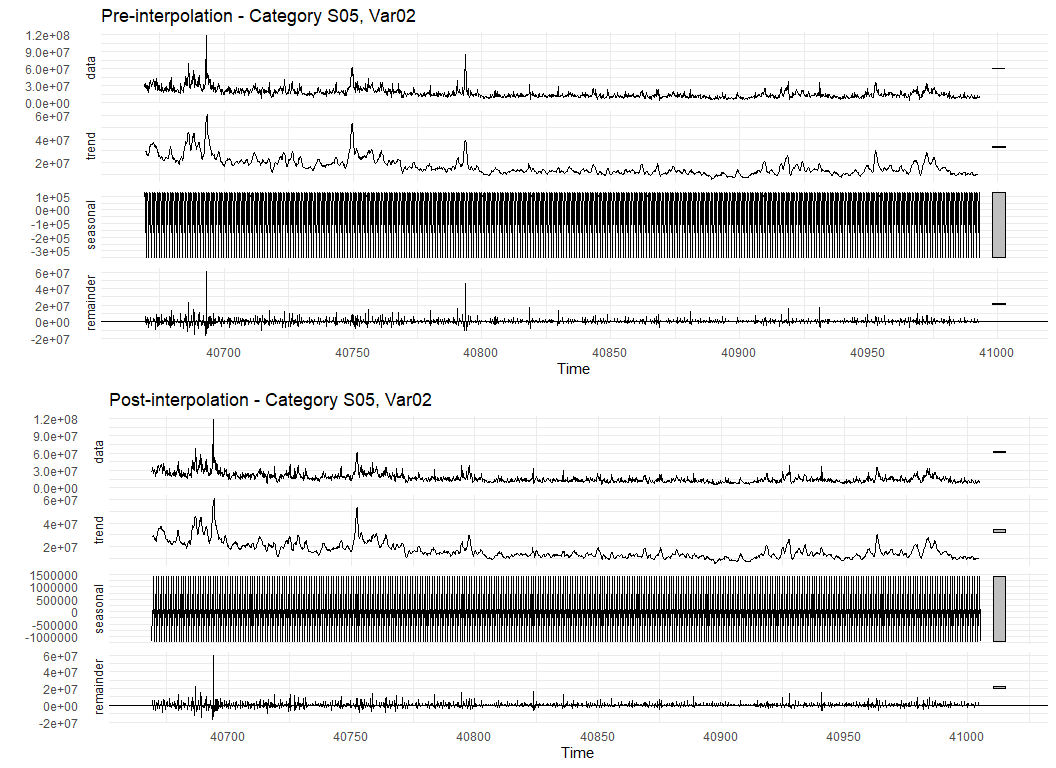
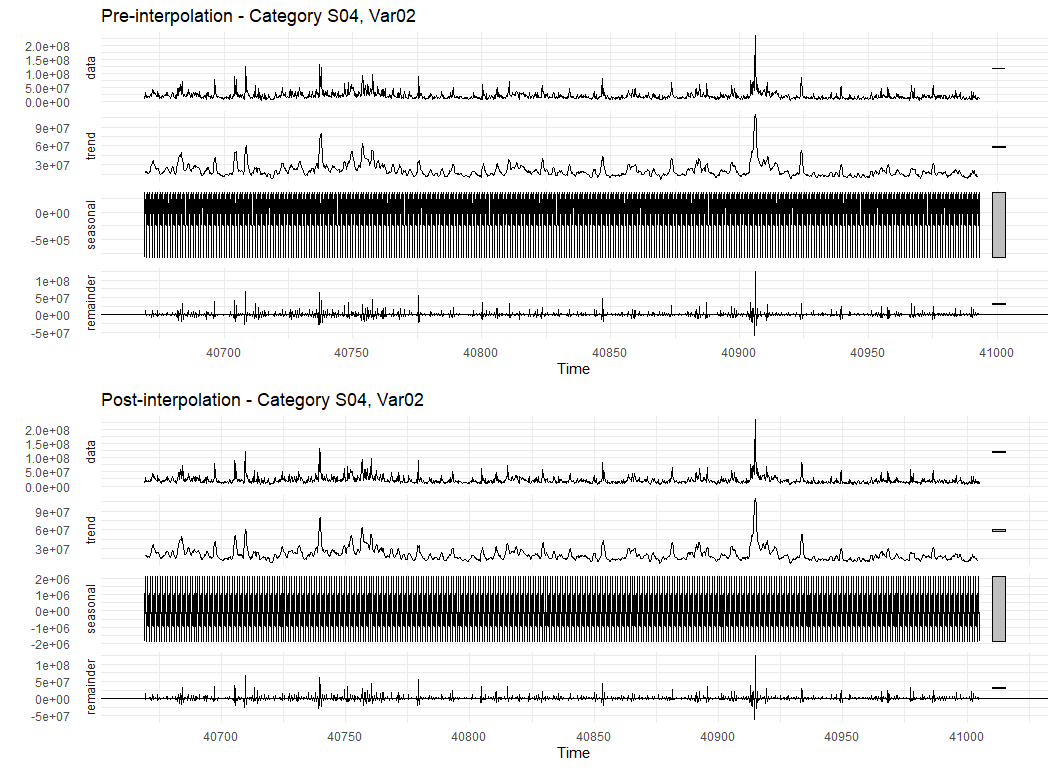
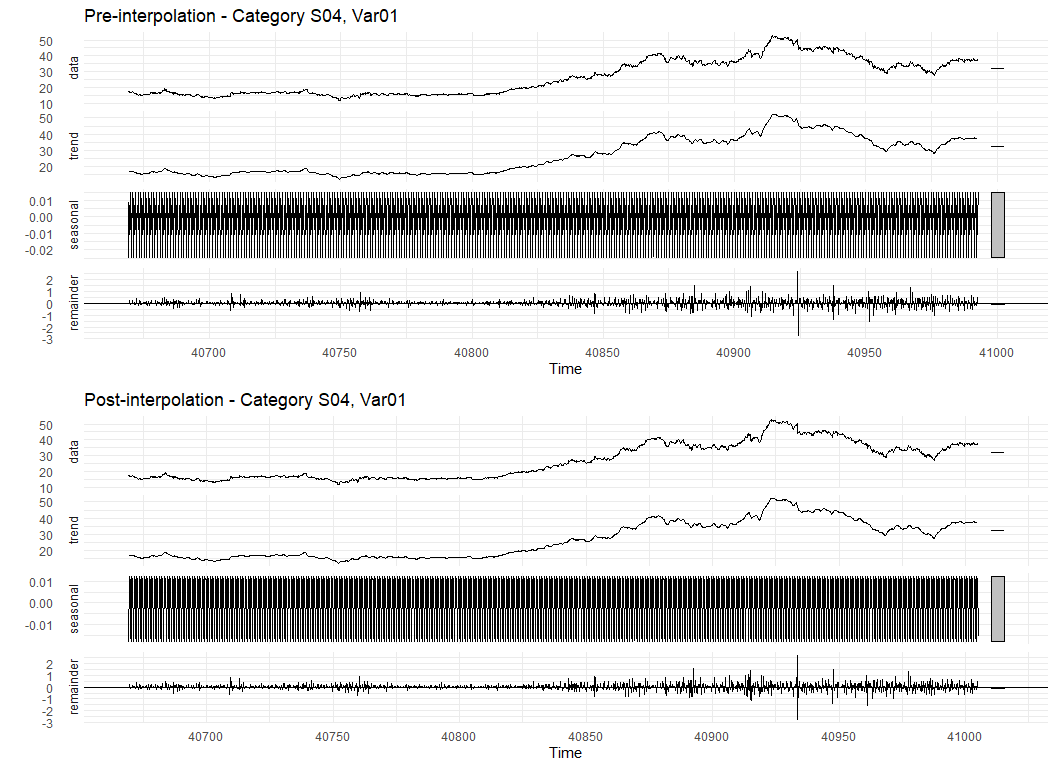
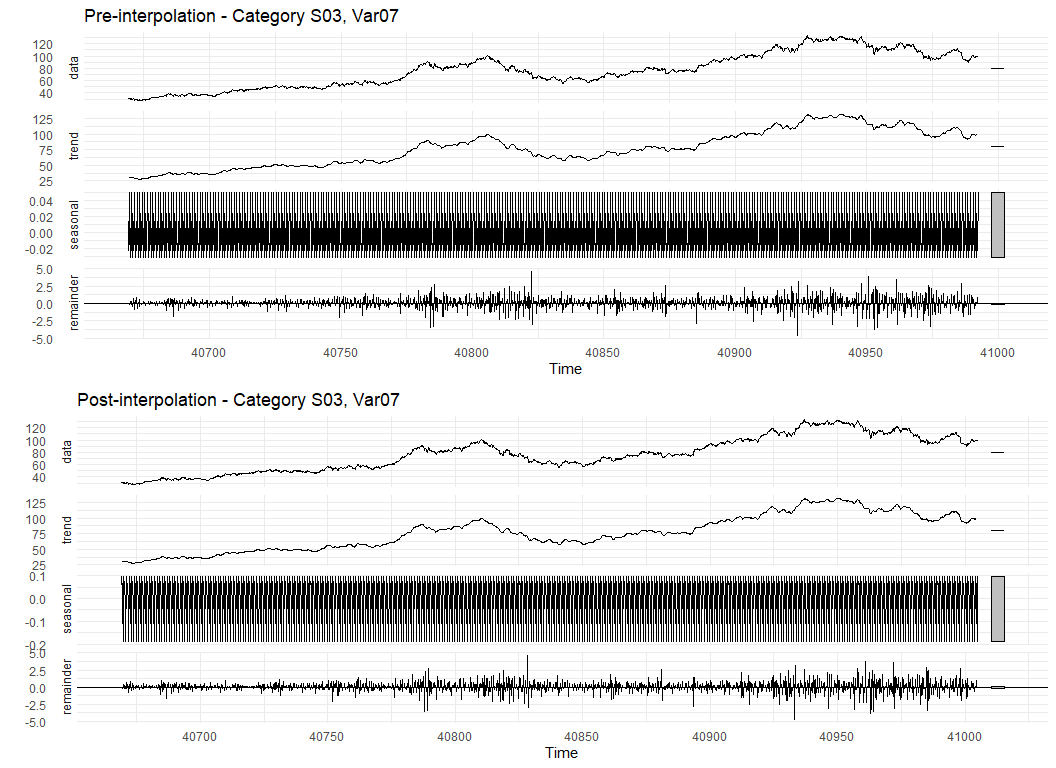
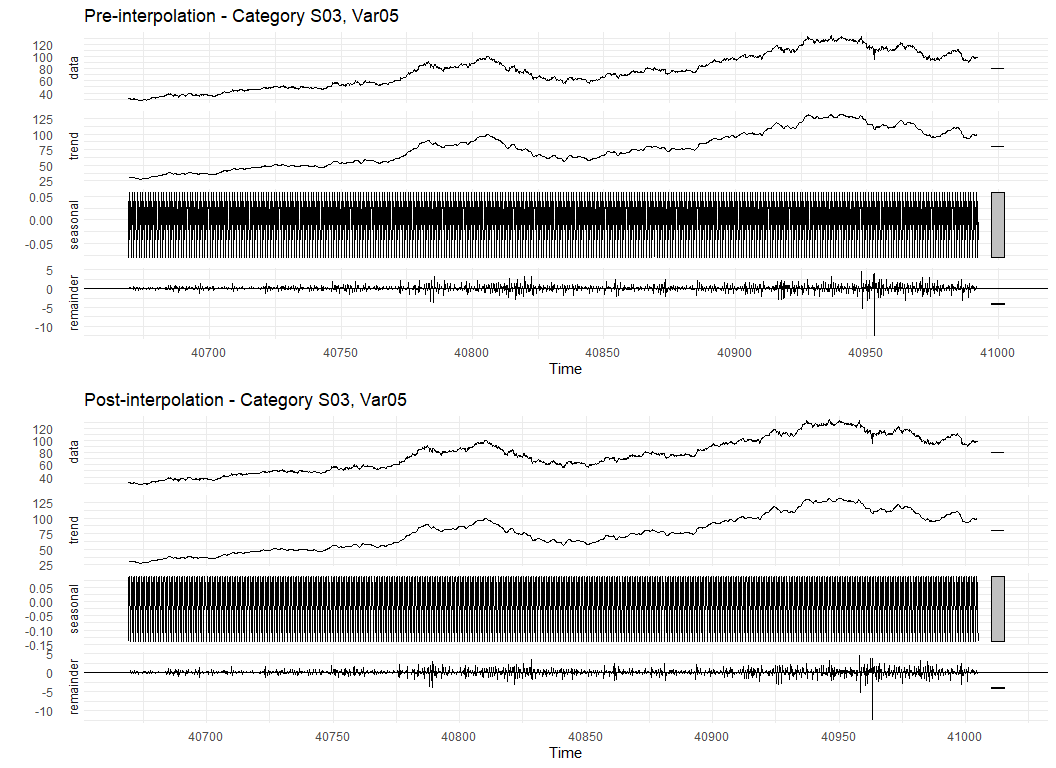
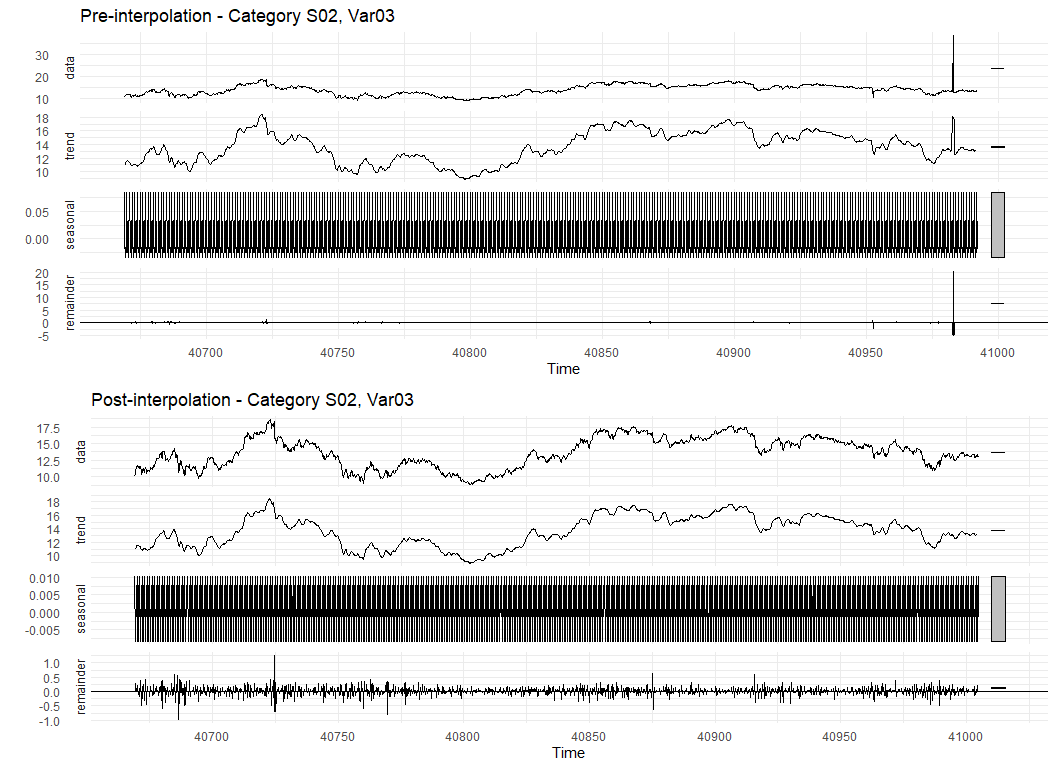
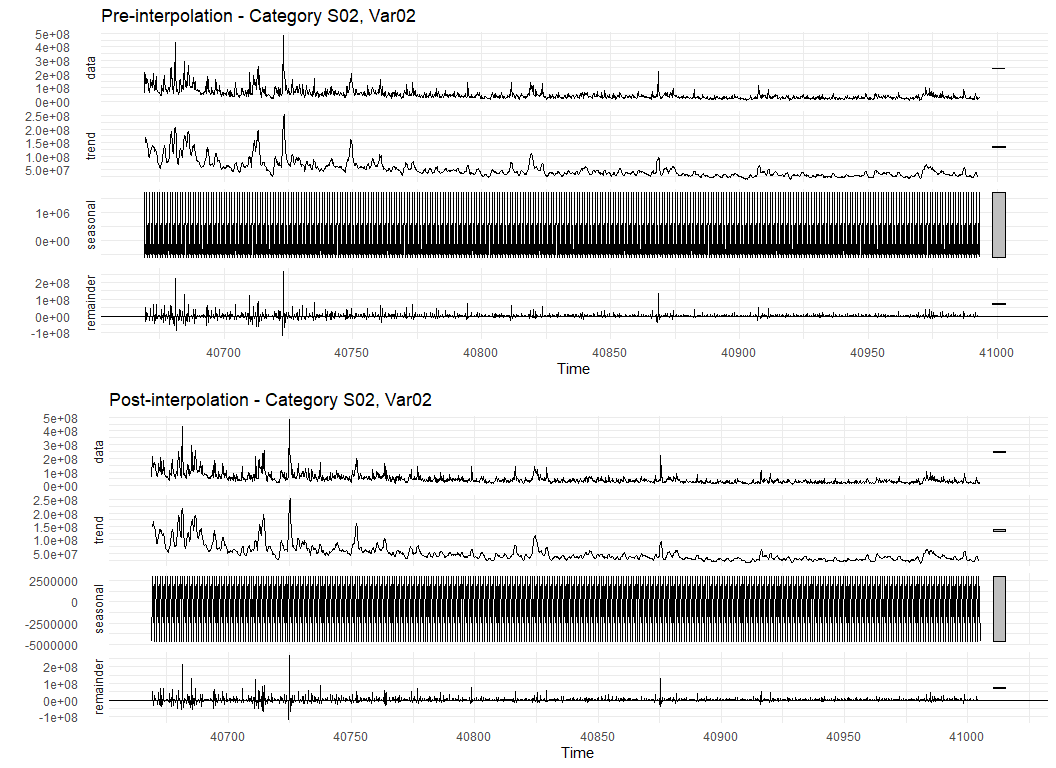
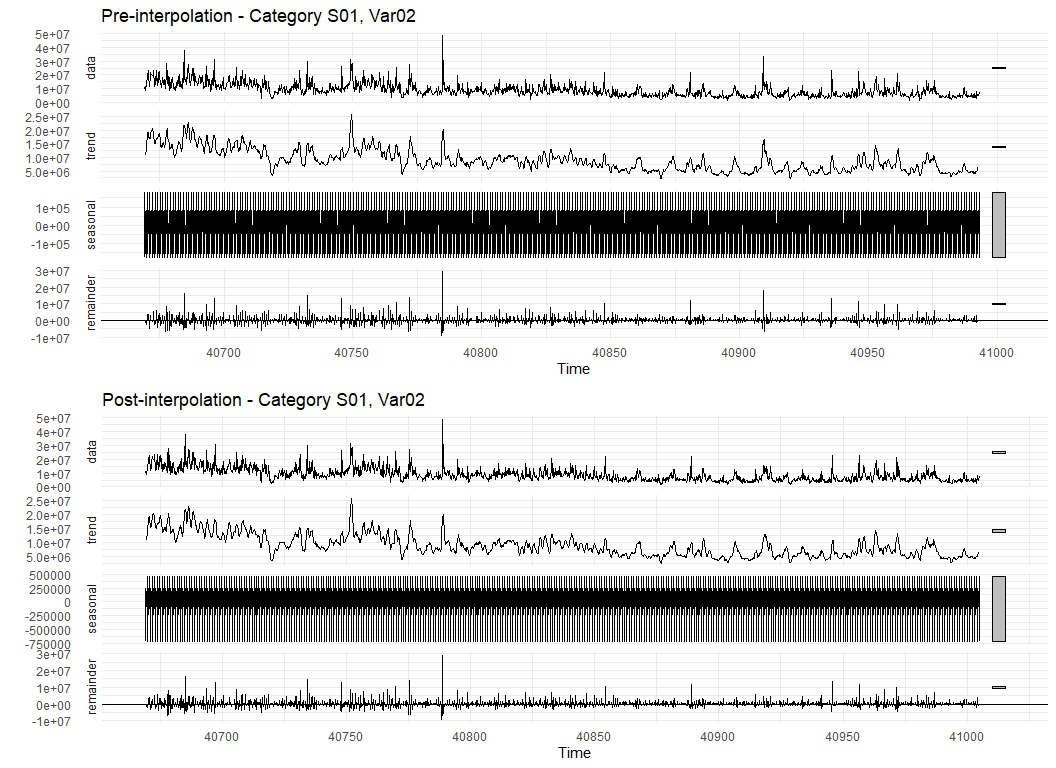
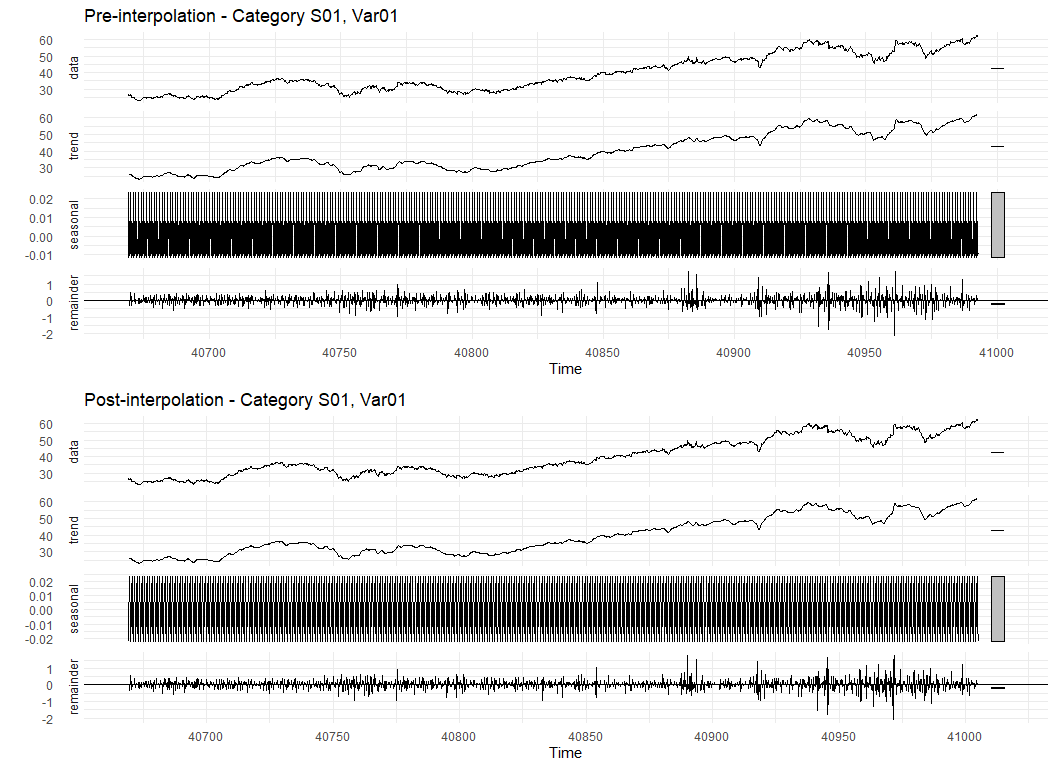
## [1] " changing 5 infinite values to NA for Var05"  
## [1] " changing 5 infinite values to NA for Var07"



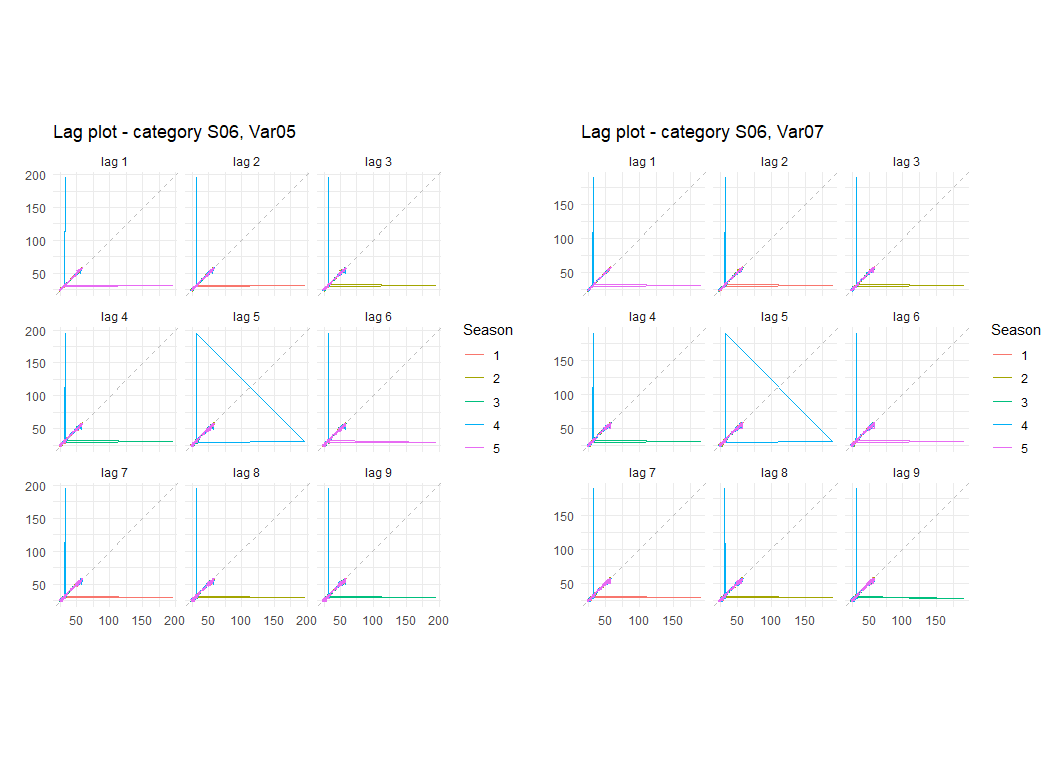
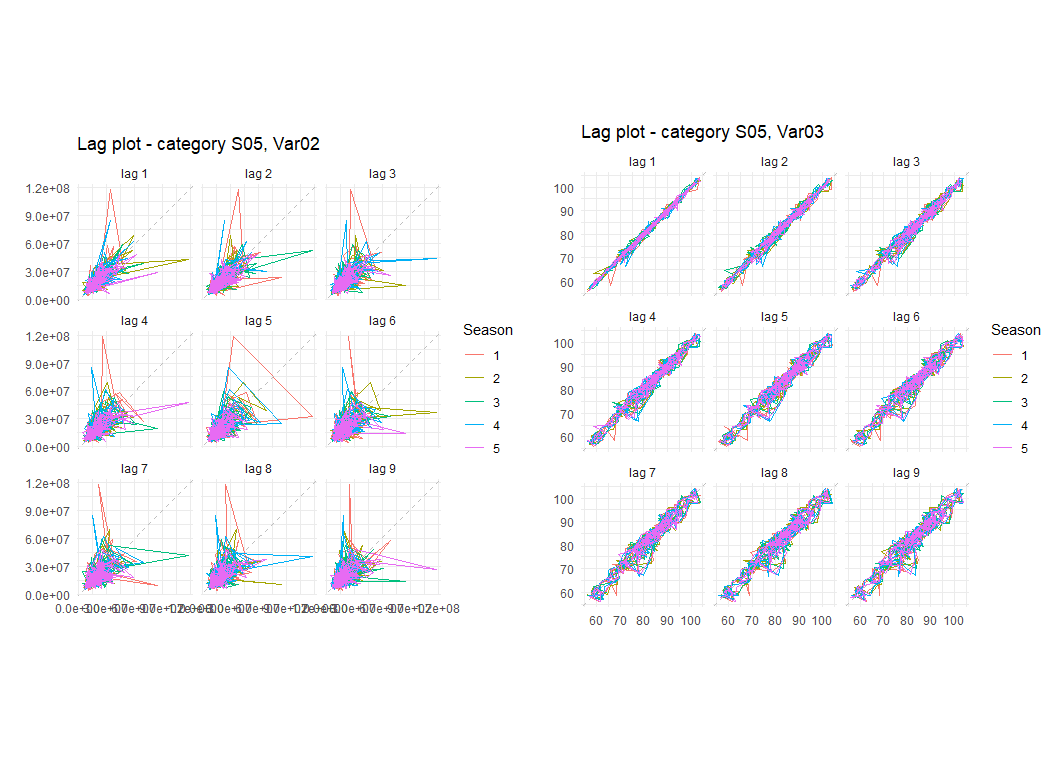
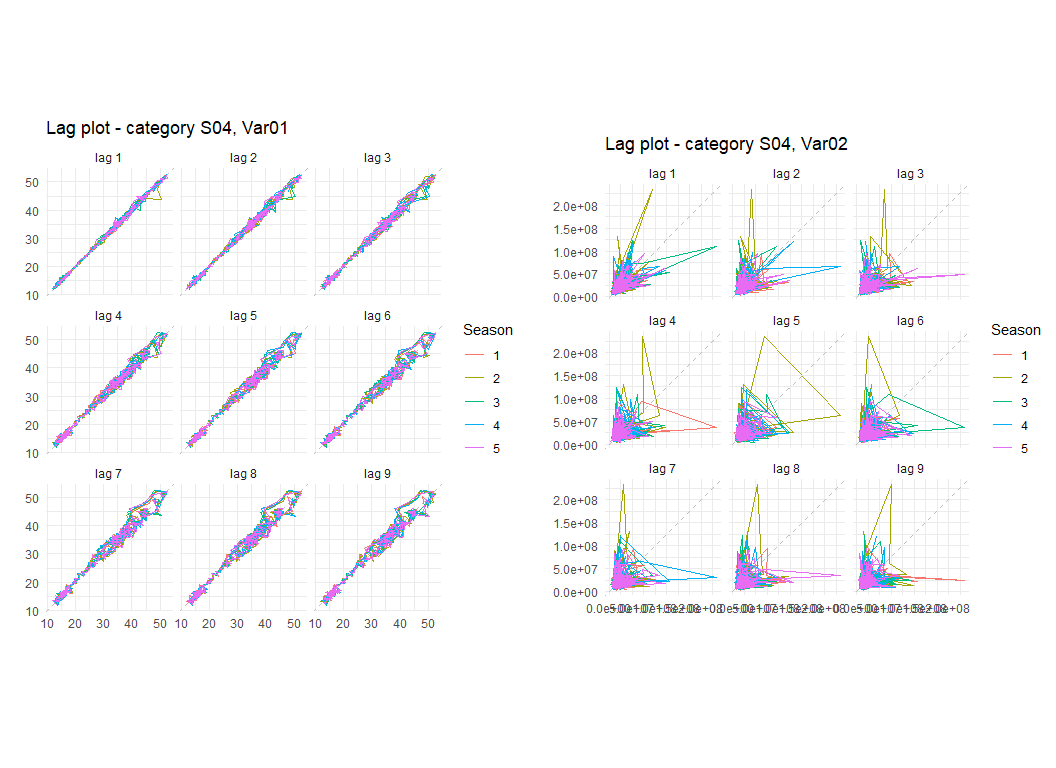
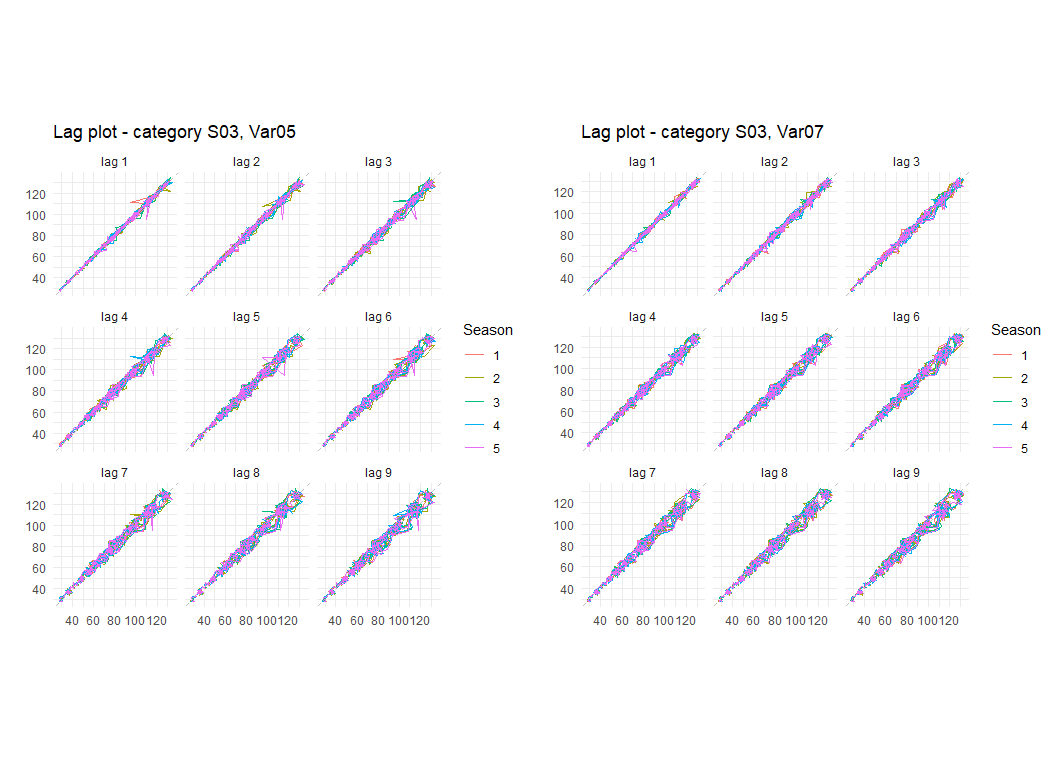
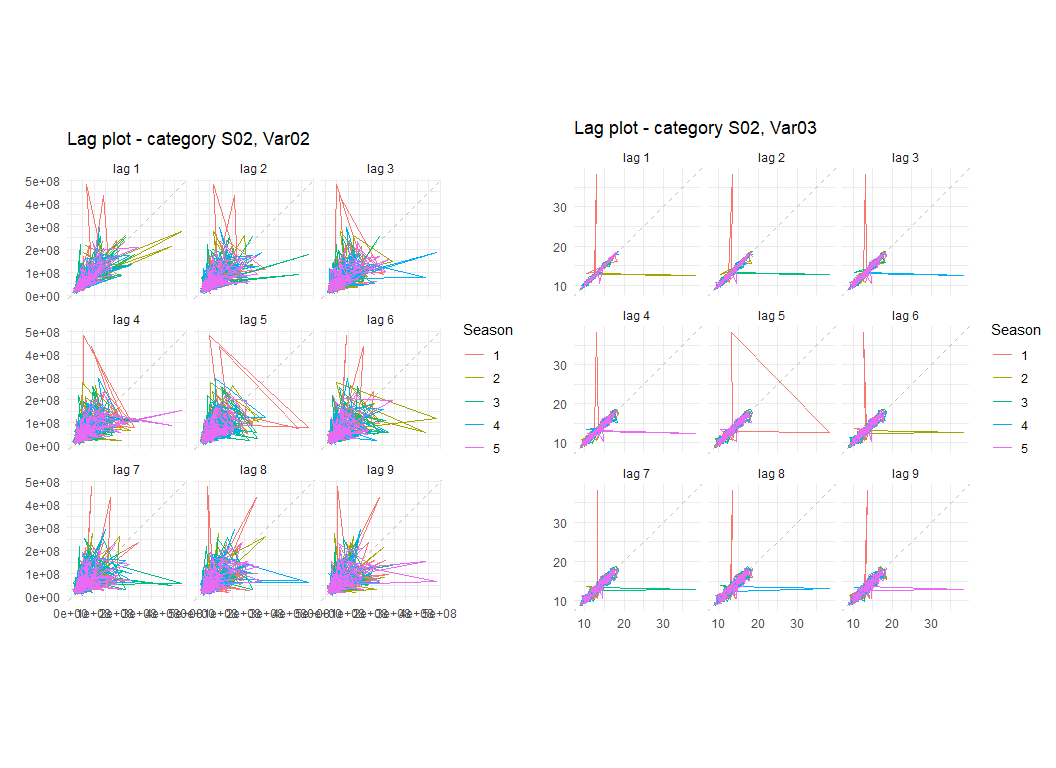
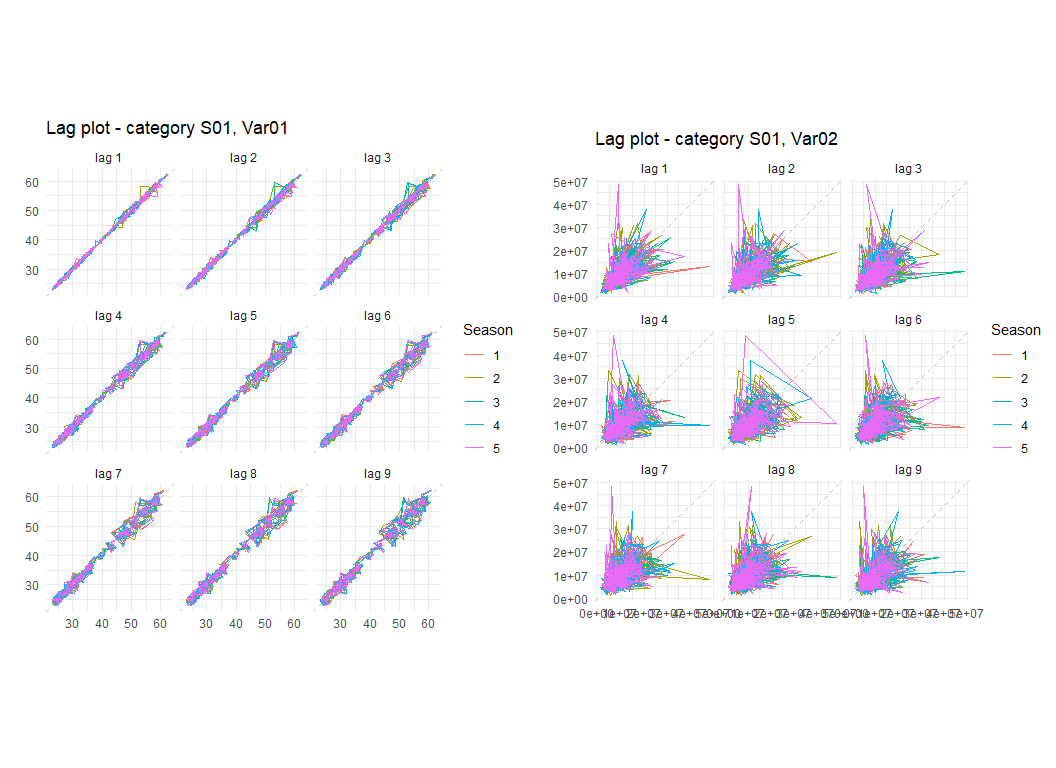
# Pre- and post-imputation examples  
i <- 6  
p1 <- ts2[[i]] %>%  
 autoplot() +  
 ggtitle(paste0('Pre-imputation, category S0', i, ', ', varname2)) +  
 ylab(varname2)  
p2 <- tsnew2[[i]] %>%  
 autoplot() +  
 ggtitle(paste0('Post-imputation, category S0', i, ', ', varname2)) +  
 ylab(varname2)  
grid.arrange(p1, p2, nrow=1, ncol=2)



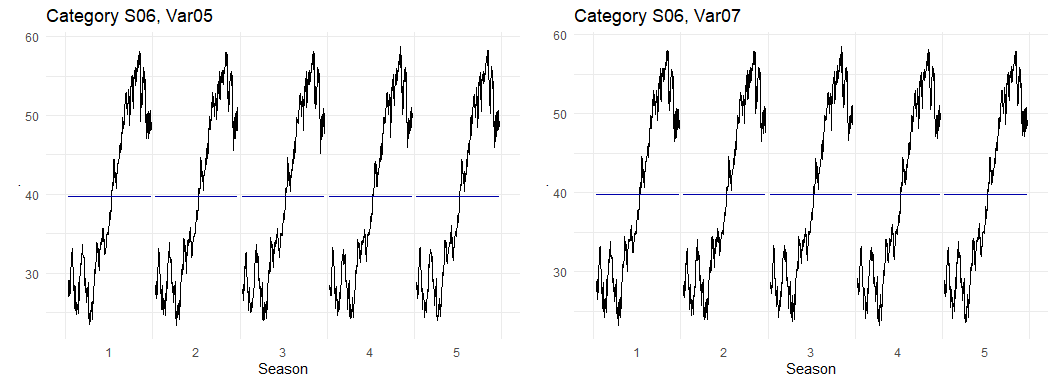
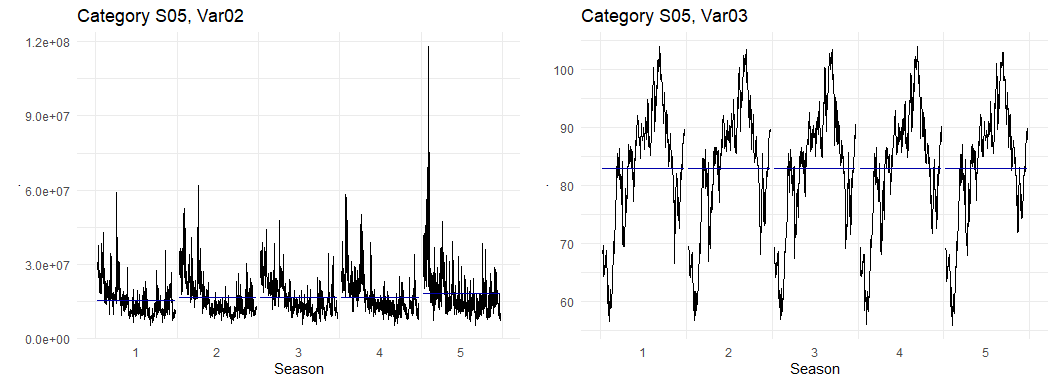
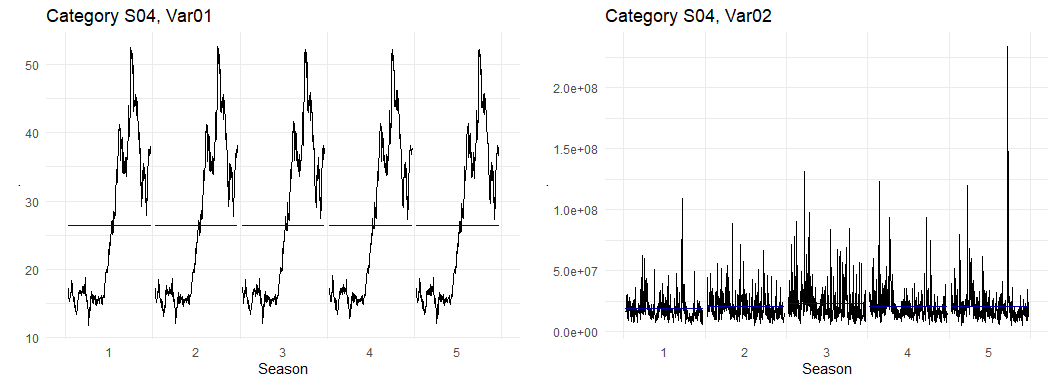
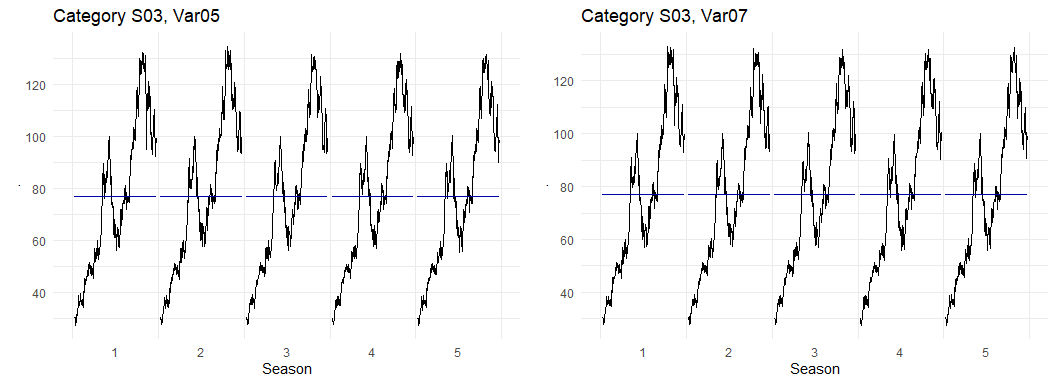
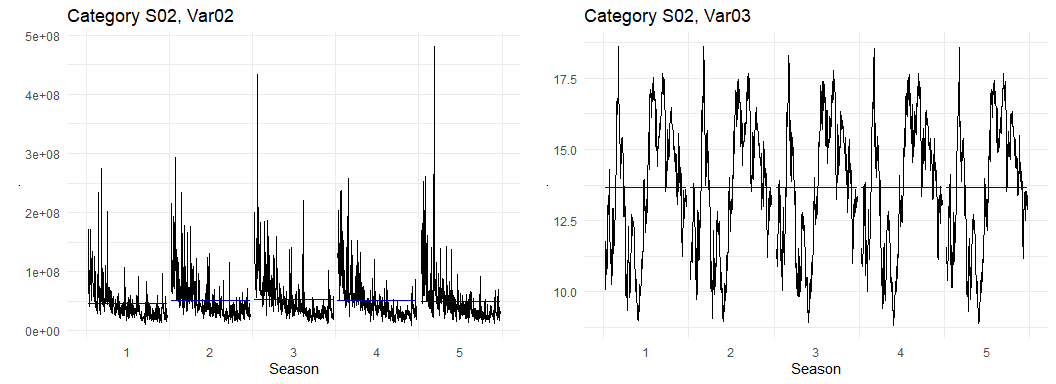
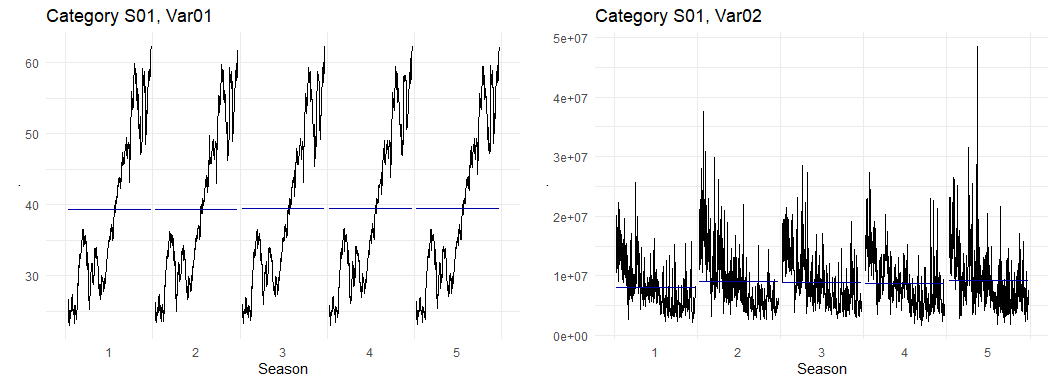
if (the\_freq > 1) {  
  
 # Compare plots of pre- and post-interpolation  
 for (i in seq(1, 6)) {  
   
 # First var  
 p1 <- (ts1[[i]] %>% decompose(type='additive')) %>%  
 autoplot() +  
 ggtitle(paste0('Pre-interpolation - Category S0', i, ', Var0', paste0(fcvars[[i]][1])))  
 p2 <- (tsnew1[[i]] %>% decompose(type='additive')) %>%  
 autoplot() +  
 ggtitle(paste0('Post-interpolation - Category S0', i, ', Var0', paste0(fcvars[[i]][1])))  
 grid.arrange(p1, p2, ncol=1, nrow=2)  
   
 # Second var  
 p3 <- (ts2[[i]] %>% decompose(type='additive')) %>%  
 autoplot() +  
 ggtitle(paste0('Pre-interpolation - Category S0', i, ', Var0', paste0(fcvars[[i]][2])))  
 p4 <- (tsnew2[[i]] %>% decompose(type='additive')) %>%  
 autoplot() +  
 ggtitle(paste0('Post-interpolation - Category S0', i, ', Var0', paste0(fcvars[[i]][2])))  
 grid.arrange(p3, p4, ncol=1, nrow=2)  
   
 }  
  
}



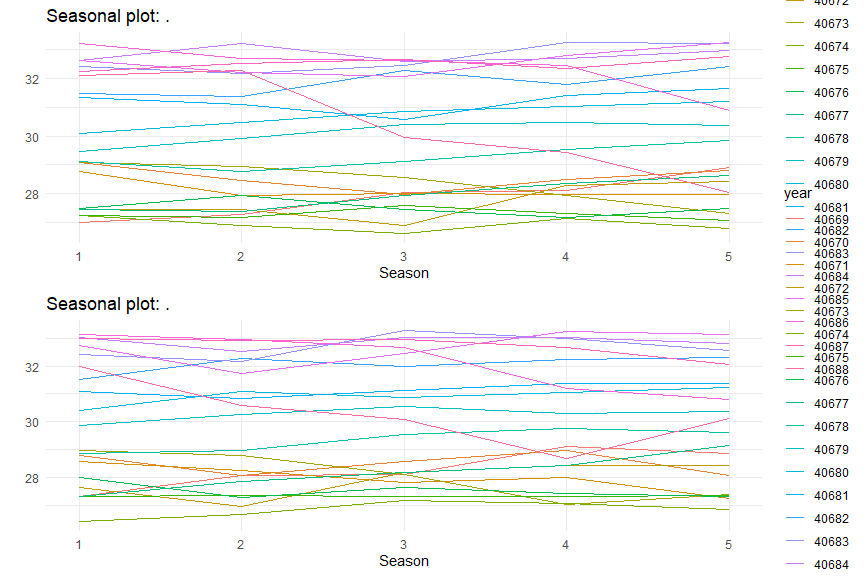
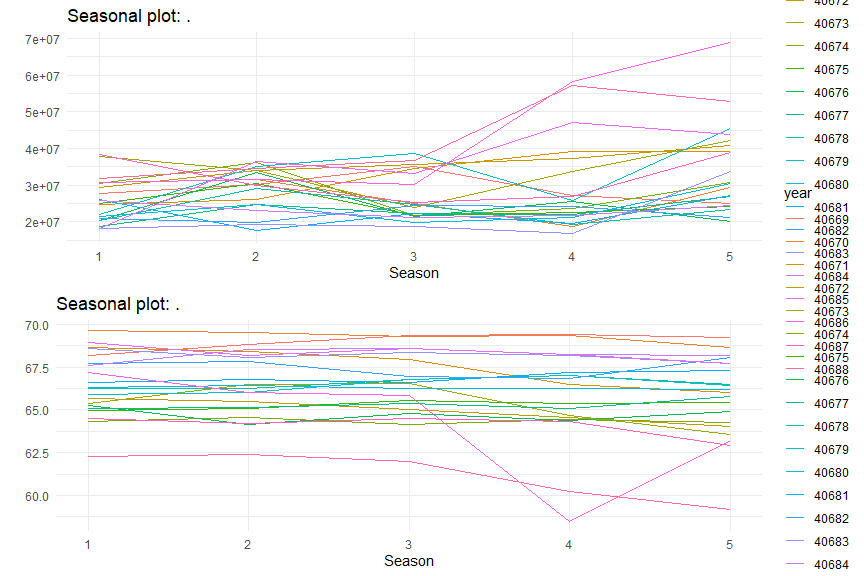
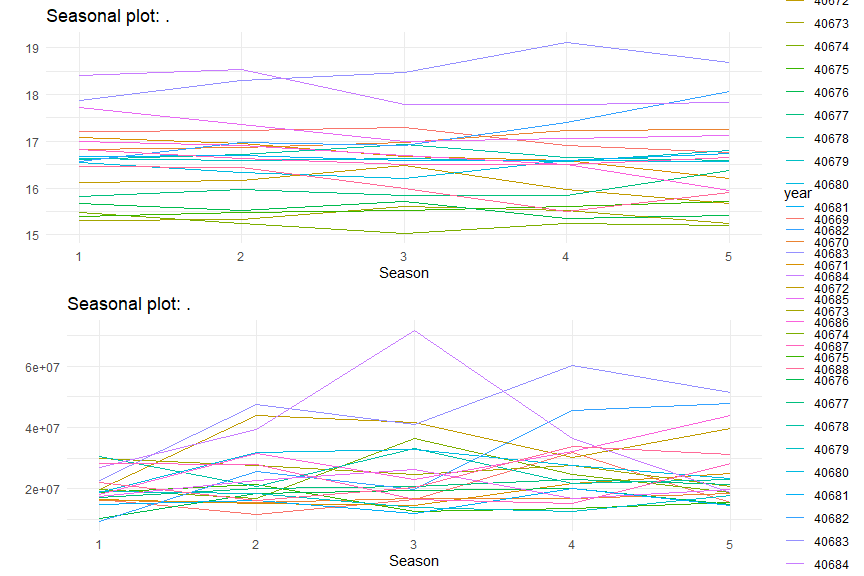
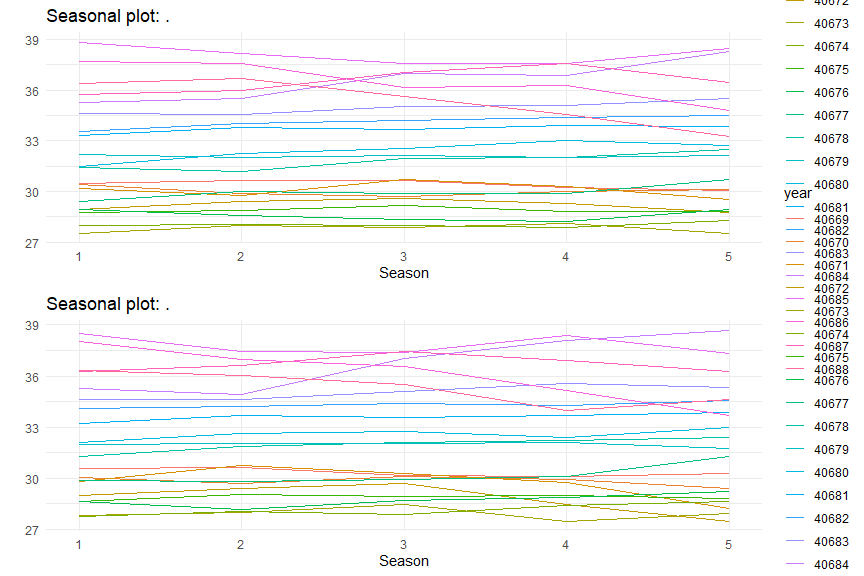
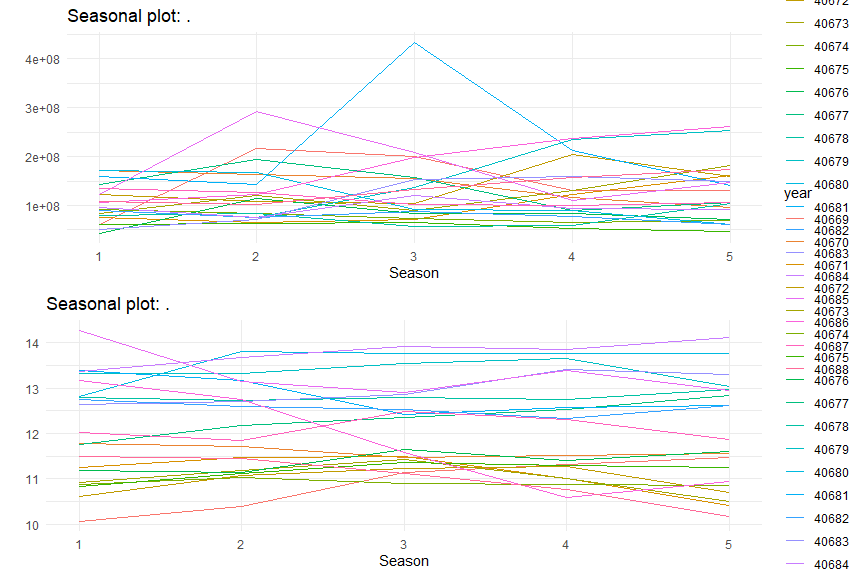
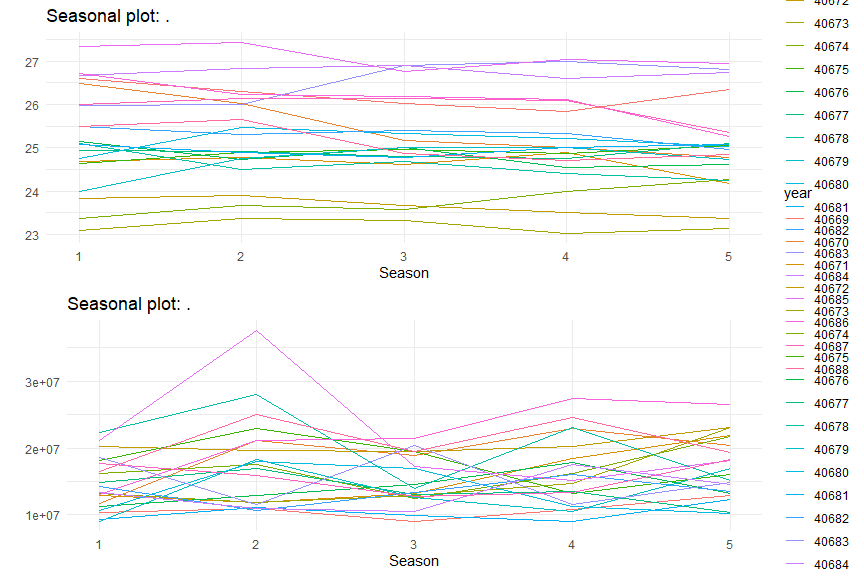
# Lag plots - I don't think these would be useful  
for (i in seq(1, 6)) {  
   
 p1 <- ts1[[i]] %>%  
 gglagplot() +  
 ggtitle(paste0('Lag plot - category S0', i, ', Var0', fcvars[[i]][1]))  
 p2 <- ts2[[i]] %>%  
 gglagplot() +  
 ggtitle(paste0('Lag plot - category S0', i, ', Var0', fcvars[[i]][2]))  
 grid.arrange(p1, p2, ncol=2, nrow=1)  
   
}



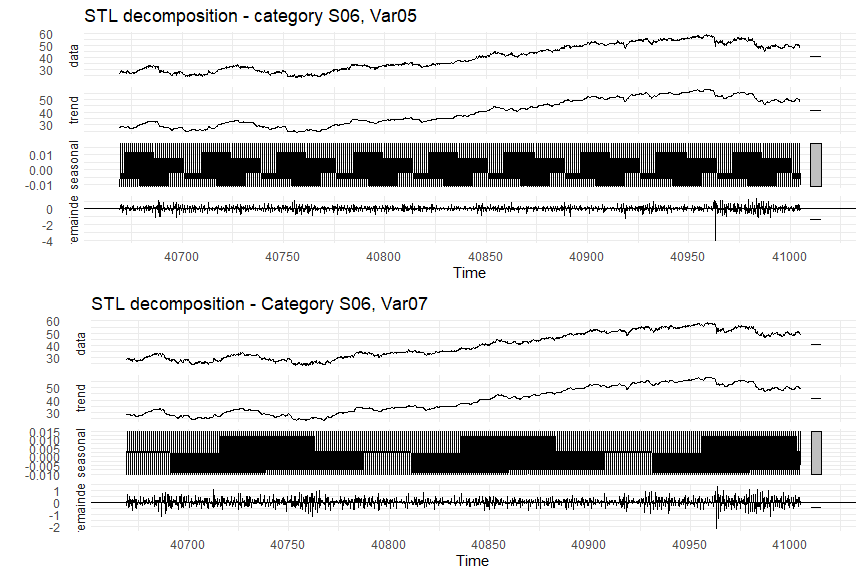
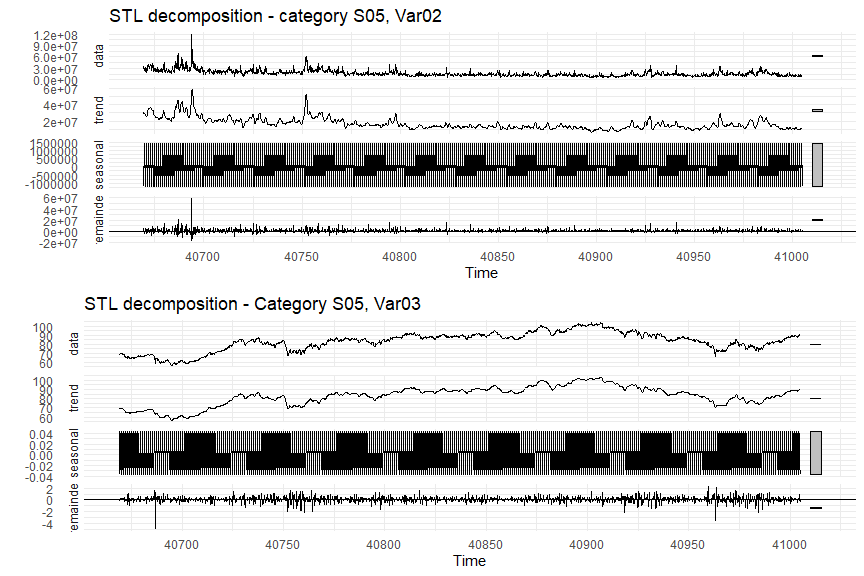
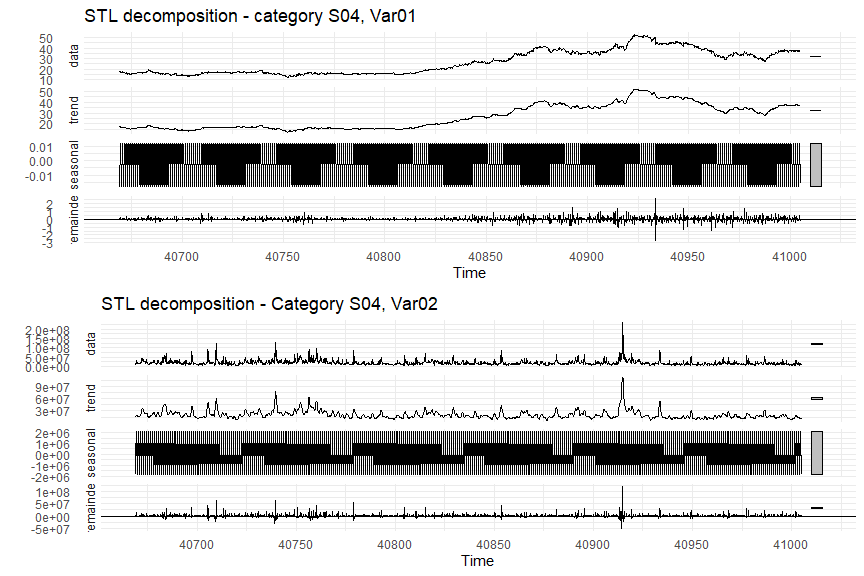
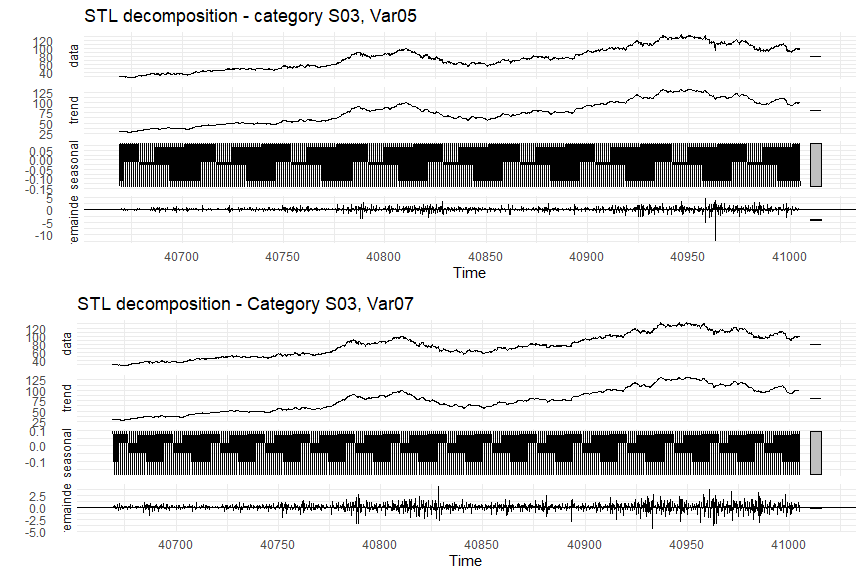
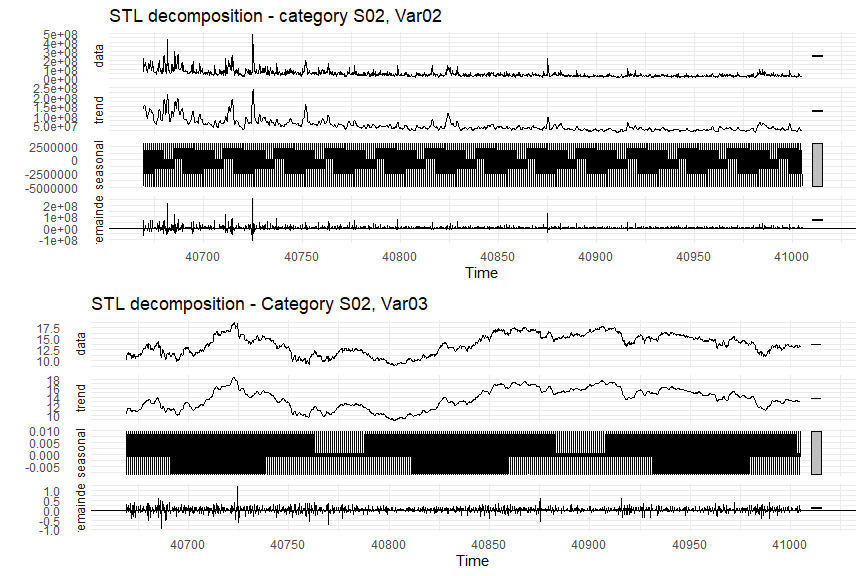
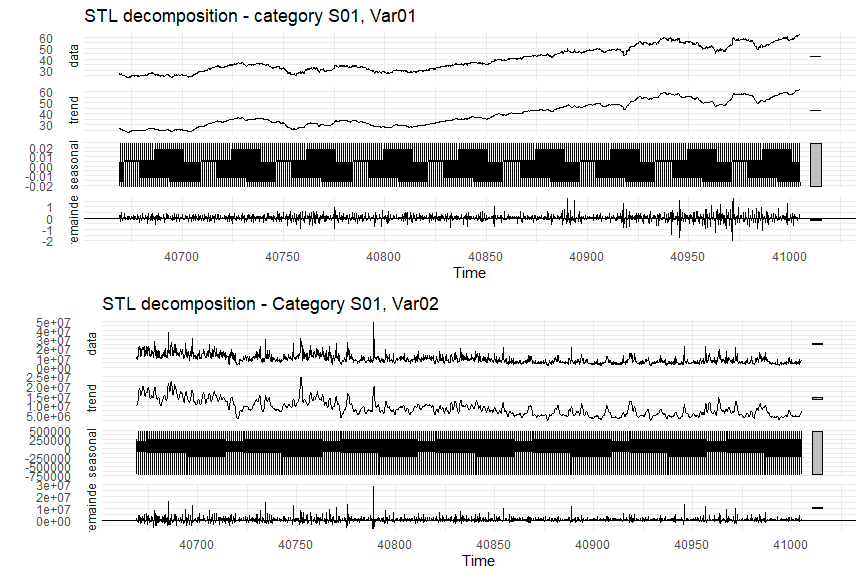
if (the\_freq > 1) {  
   
 # Seasonal subseries plots  
 for (i in seq(1, 6)) {  
   
 p1 <- tsnew1[[i]] %>%  
 ggsubseriesplot(main=paste0('Category S0', i, ', Var0', fcvars[[i]][1]))  
 p2 <- tsnew2[[i]] %>%  
 ggsubseriesplot(main=paste0('Category S0', i, ', Var0', fcvars[[i]][2]))  
 grid.arrange(p1, p2, ncol=2, nrow=1)  
   
 }  
   
}



if (the\_freq > 1) {  
  
 # I don't think these will be useful  
 for (i in seq(1, 6)) {  
 p1 <- tsnew1[[i]] %>%  
 head(100) %>%  
 ggseasonplot()  
 p2 <- tsnew2[[i]] %>%  
 head(100) %>%  
 ggseasonplot()  
 grid.arrange(p1, p2, nrow=2, ncol=1)  
 }  
   
}



if (the\_freq > 1) {  
   
 # Decomposition  
 for (i in seq(1, 6)) {  
   
 p1 <- tsnew1[[i]] %>%   
 head(3000) %>%  
 stl(s.window='periodic') %>%  
 autoplot() +  
 ggtitle(paste0('STL decomposition - category S0', i, ', Var0', paste0(fcvars[[i]][1])))  
 p2 <- tsnew2[[i]] %>%   
 head(3000) %>%  
 stl(s.window='periodic') %>%  
 autoplot() +  
 ggtitle(paste0('STL decomposition - Category S0', i, ', Var0', paste0(fcvars[[i]][2])))  
 grid.arrange(p1, p2, ncol=1, nrow=2)  
   
 }  
   
}



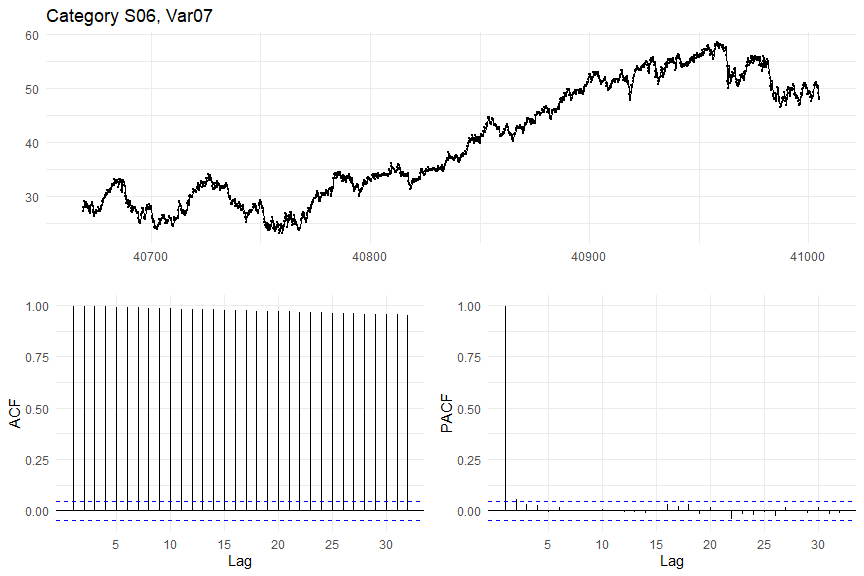
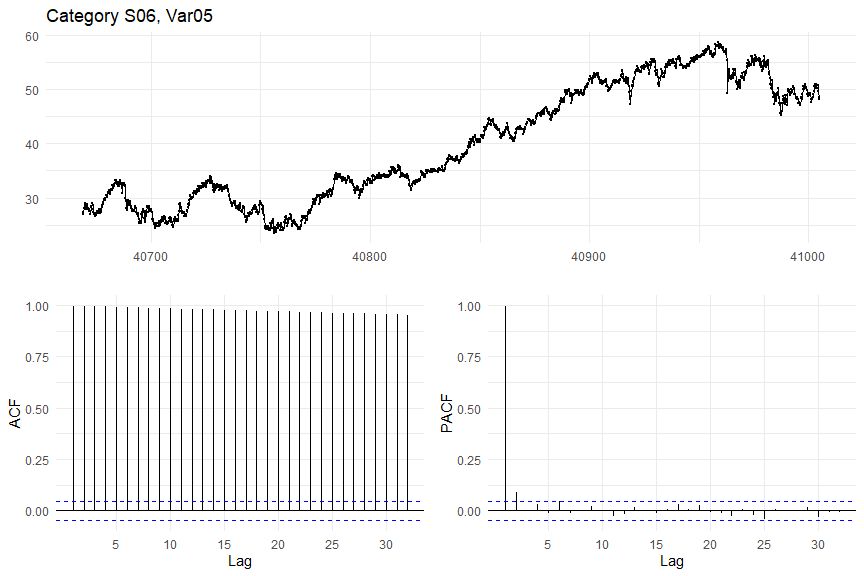
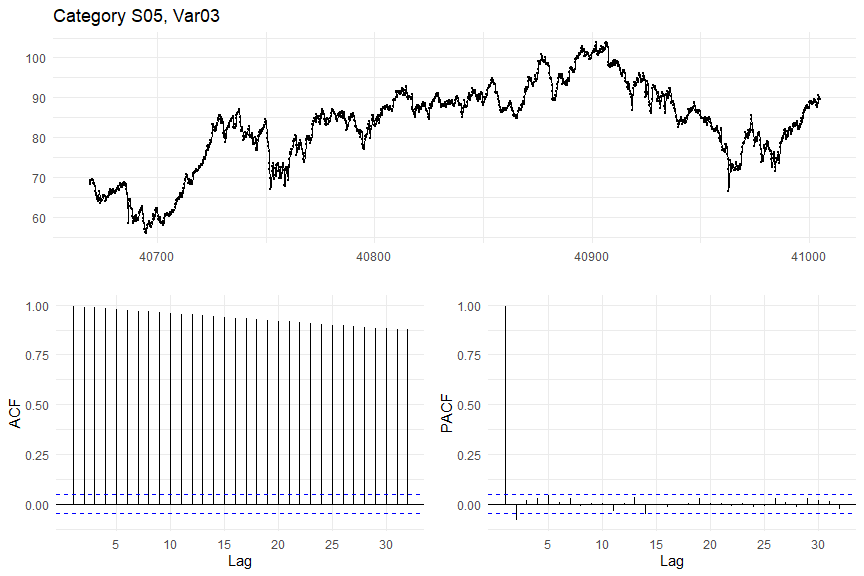
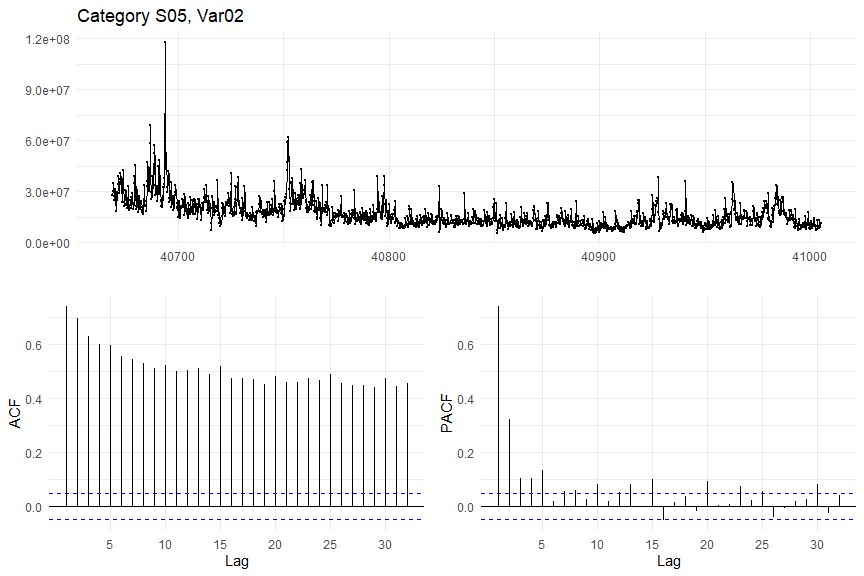
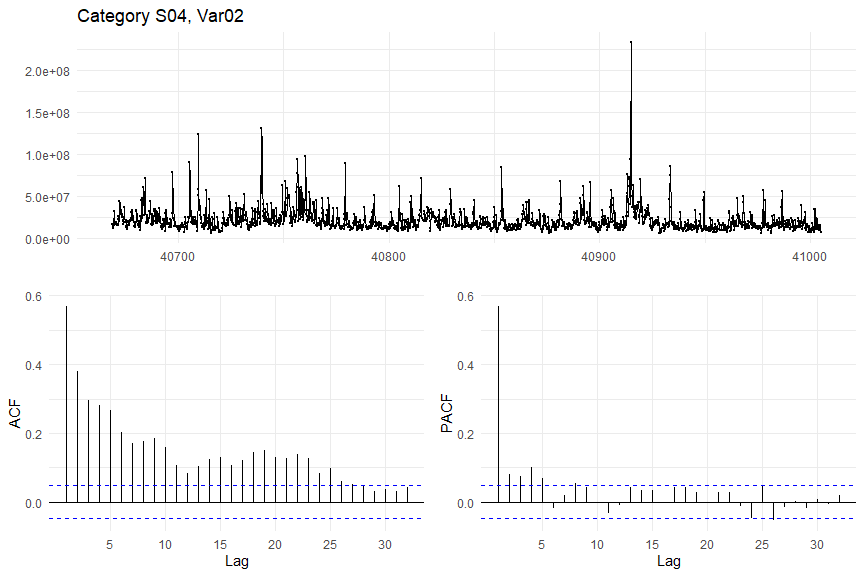
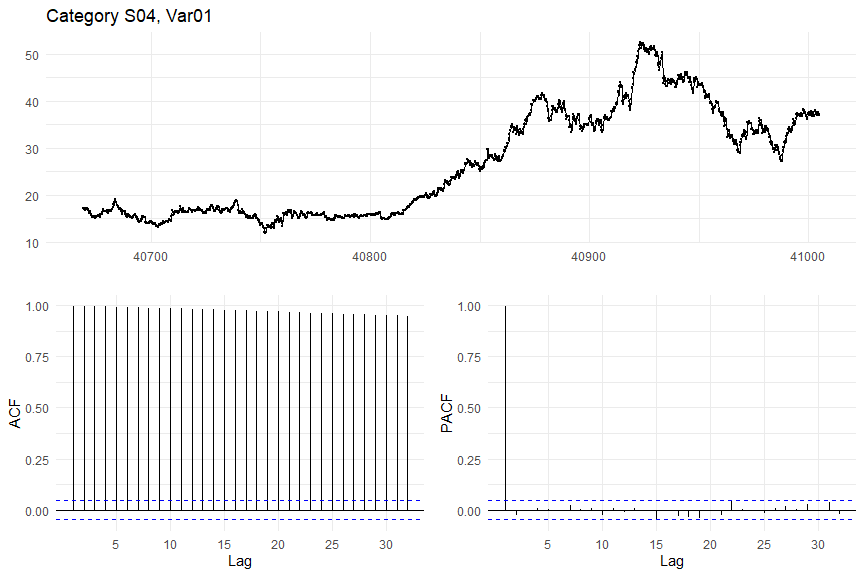
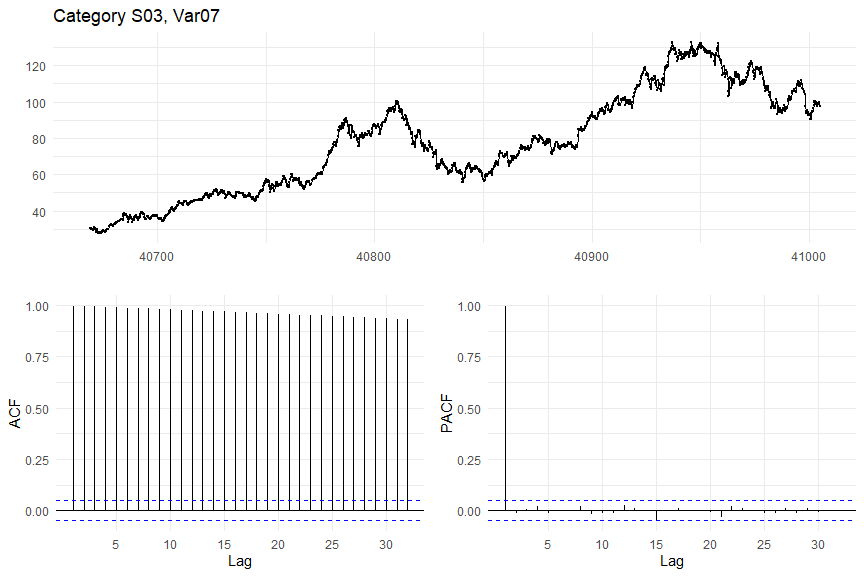
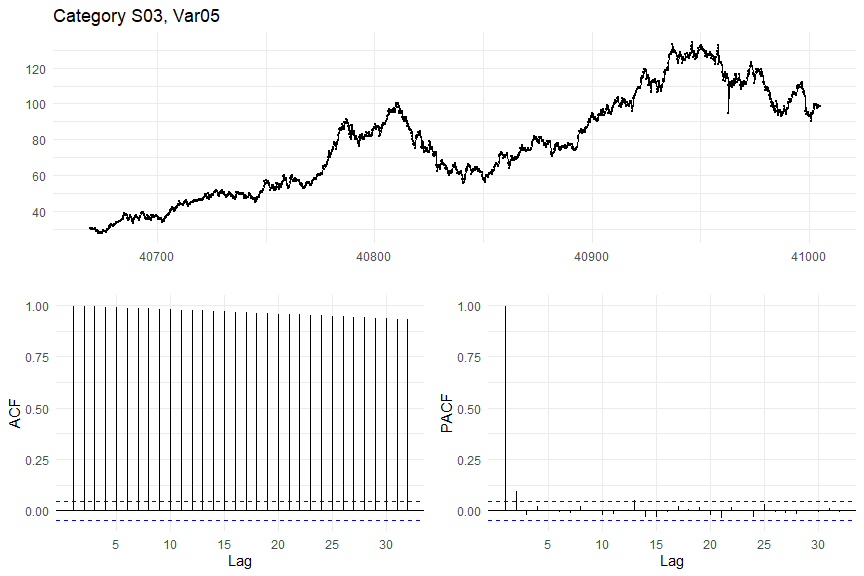
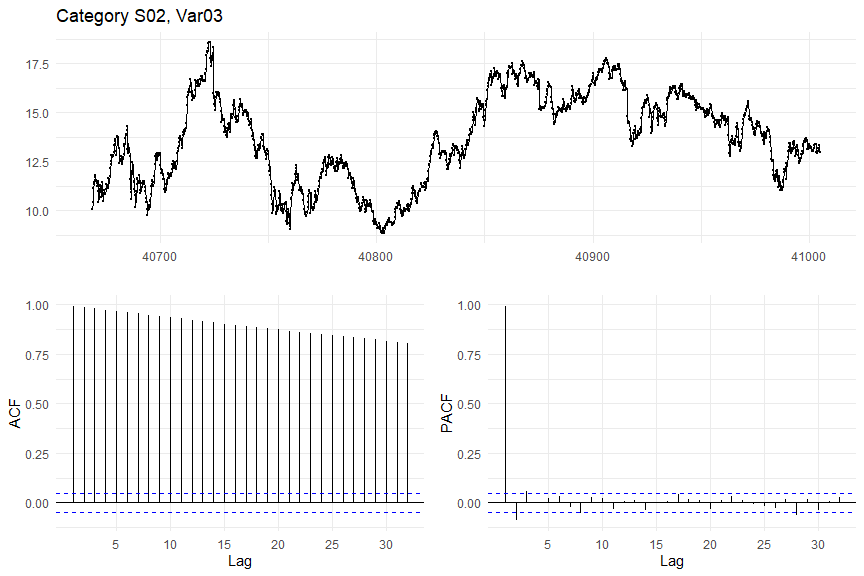
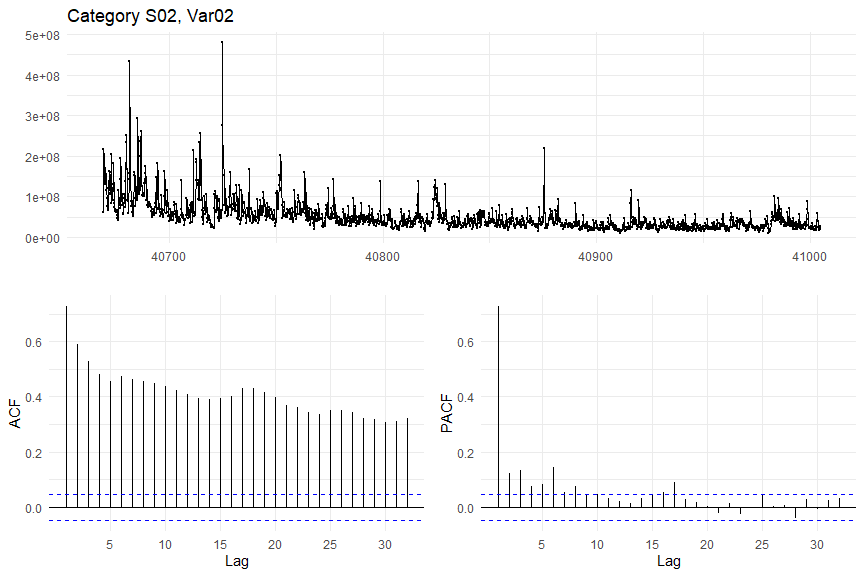
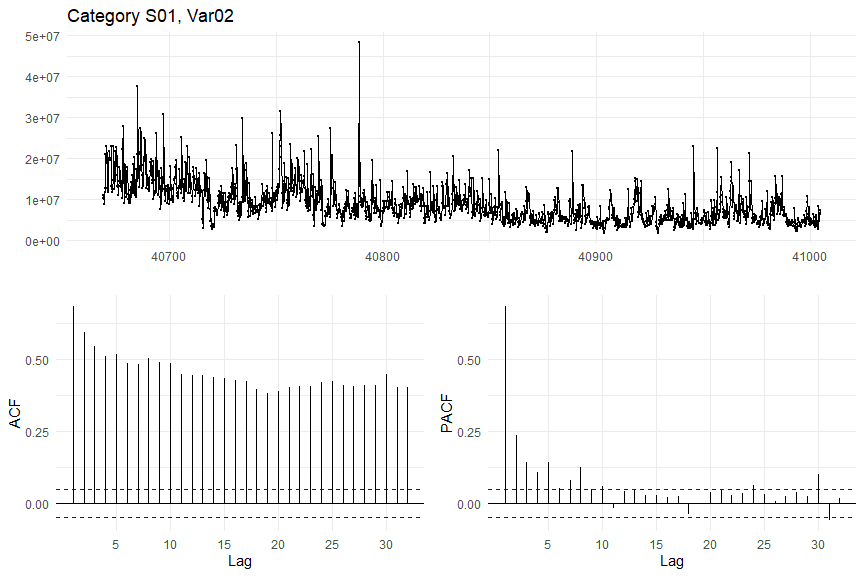
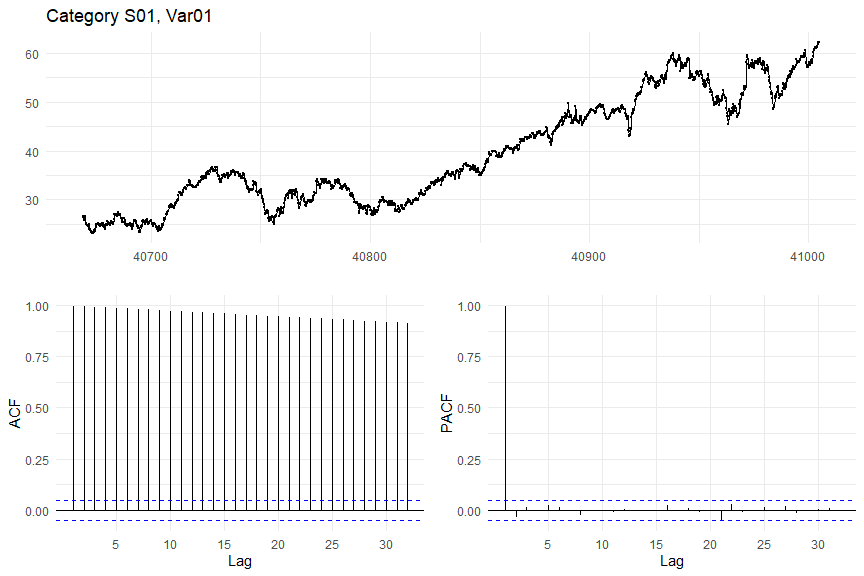
Some types of models are sensitive to data that is autocorrelated, that is, data that contains values which are related to previous values in some regular or predictable way. These models require that the data be modified such that they are “stationary,” meaning that they appear to be randomly distributed and when plotted look like “white noise.”

To identify autocorrelation patterns in the data, autocorrelation function (ACF) and partial autocorrelation function (PACF) plots can be constructed. These plots illustrate the relationship between lagged time series values, i.e. comparing one value with the the next value in the series, or the values two or more positions later. Examining the patterns in the ACF and PACF plots helps the modeler determine what parameters to use as a basis when modeling.

ACF and PACF plots aid in evaluating whether the data is autocorrelated and, if so, whether it should be modified before modeling occurs. One such modification is “differencing,” which converts time series data into the *change* in value over time. Once the data is “differenced,” it is no longer autocorrelated, and the time series should appear to be “white noise.” Likewise, the ACF and PACF plots should exhibit no clear trend or pattern.

As shown in the figures, there is some trending to most variables that would indicate that differencing is needed prior to modeling. One possible exception is Var02 in categories

# ACF/PACF plots - needed for ARIMA modeling  
for (i in seq(1, 6)) {  
  
 p1 <- tsnew1[[i]] %>%  
 ggtsdisplay(plot.type='partial', main=paste0('Category S0', i, ', Var0', fcvars[[i]][1]))  
 p2 <- tsnew2[[i]] %>%  
 ggtsdisplay(plot.type='partial', main=paste0('Category S0', i, ', Var0', fcvars[[i]][2]))  
 p1  
 p2  
   
}



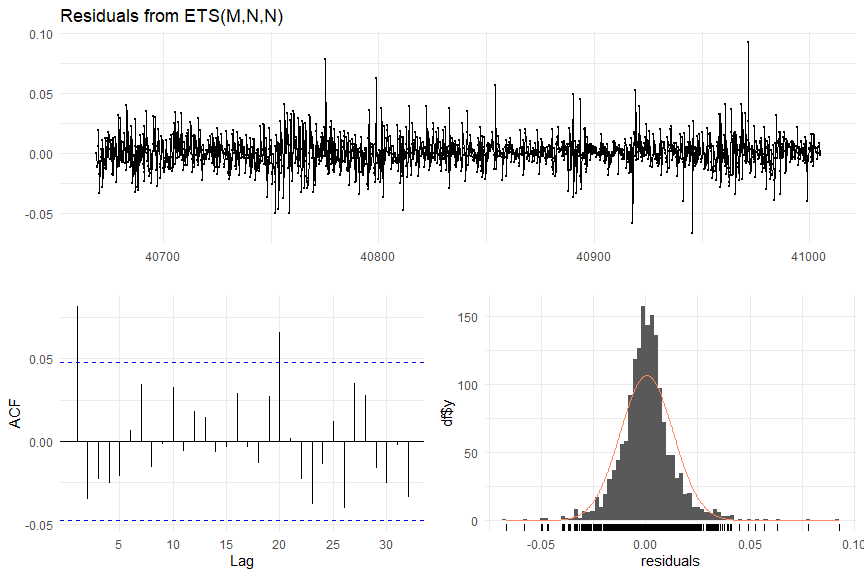
# Init df to hold kpss test stats to determine whether differencing is needed  
dfdiffs <- data.frame(matrix(nrow=0, ncol=6))  
colnames(dfdiffs) = c('Category', 'Variable', 'KPSS.test.statistic', 'Differencing.required',   
 'Number.of.seasonal.differences', 'Seasonal.differencing.required')  
  
# Differencing - all need differencing except Var02  
for (i in seq(1, 6)) {  
  
 # First variable in this category  
 tmp\_kpss <- summary(tsnew1[[i]] %>% ur.kpss())@teststat  
 if (the\_freq > 1) {  
 tmp\_nsdiffs <- tsnew1[[i]] %>% nsdiffs()  
 }  
 else {  
 tmp\_nsdiffs <- 0  
 }  
 diff\_req <- 'no'  
 seas\_diff\_req <- 'no'  
 if (tmp\_kpss > 1) { # test statistic is given in percent, if it exceeds 1%, differencing is required  
 diff\_req <- 'yes'  
 }  
 if (tmp\_nsdiffs > 0) { # nsdiffs() gives the number of seasonal differences required  
 seas\_diff\_req <- 'yes'  
 }  
 dfdiffs <- rbind(dfdiffs, data.frame(  
 Category=paste0('S0', i),  
 Variable=paste0('V0', fcvars[[i]][1]),  
 KPSS.test.statistic=tmp\_kpss,  
 Differencing.required=diff\_req,  
 Number.of.seasonal.differences=tmp\_nsdiffs,  
 Seasonal.differencing.required=seas\_diff\_req  
 ))  
  
 # Second variable in this category  
 tmp\_kpss <- summary(tsnew2[[i]] %>% ur.kpss())@teststat  
 if (the\_freq > 1) {  
 tmp\_nsdiffs <- tsnew2[[i]] %>% nsdiffs()  
 }  
 else {  
 tmp\_nsdiffs <- 0  
 }  
 diff\_req <- 'no'  
 seas\_diff\_req <- 'no'  
 if (tmp\_kpss > 1) { # test statistic is given in percent, if it exceeds 1%, differencing is required  
 diff\_req <- 'yes'  
 }  
 if (tmp\_nsdiffs > 0) { # nsdiffs() gives the number of seasonal differences required  
 seas\_diff\_req <- 'yes'  
 }  
 dfdiffs <- rbind(dfdiffs, data.frame(  
 Category=paste0('S0', i),  
 Variable=paste0('V0', fcvars[[i]][1]),  
 KPSS.test.statistic=tmp\_kpss,  
 Differencing.required=diff\_req,  
 Number.of.seasonal.differences=tmp\_nsdiffs,  
 Seasonal.differencing.required=seas\_diff\_req  
 ))  
  
}  
  
# Show table  
dfdiffs %>%  
 kbl(caption='Differencing requirements') %>%  
 kable\_classic(full\_width=F)

# Create new list to hold differenced ts objects  
tsdiff1 <- list()  
tsdiff2 <- list()  
  
# Now actually do the differencing  
for (i in seq(1, 6)) {  
   
 # First variable in category i  
 tsdiff1[[i]] <- diff(tsnew1[[i]])  
 tmp\_kpss <- summary(tsdiff1[[i]] %>% ur.kpss())@teststat  
 dfdiffs[2 \* i - 1, 'KPSS.test.statistic'] <- tmp\_kpss  
 diff\_req <- 'no'  
 if (tmp\_kpss > 1) { # test statistic is given in percent, if it exceeds 1%, differencing is required  
 diff\_req <- 'yes'  
 }  
 dfdiffs[2 \* i - 1, 'Differencing.required'] <- diff\_req  
  
 # Second variable in category i  
 tsdiff2[[i]] <- diff(tsnew1[[i]])  
 tmp\_kpss <- summary(tsdiff1[[i]] %>% ur.kpss())@teststat  
 dfdiffs[2 \* i, 'KPSS.test.statistic'] <- tmp\_kpss  
 diff\_req <- 'no'  
 if (tmp\_kpss > 1) { # test statistic is given in percent, if it exceeds 1%, differencing is required  
 diff\_req <- 'yes'  
 }  
 dfdiffs[2 \* i, 'Differencing.required'] <- diff\_req  
}  
  
# Show updated table  
dfdiffs %>%  
 kbl(caption='Differencing requirements') %>%  
 kable\_classic(full\_width=F)

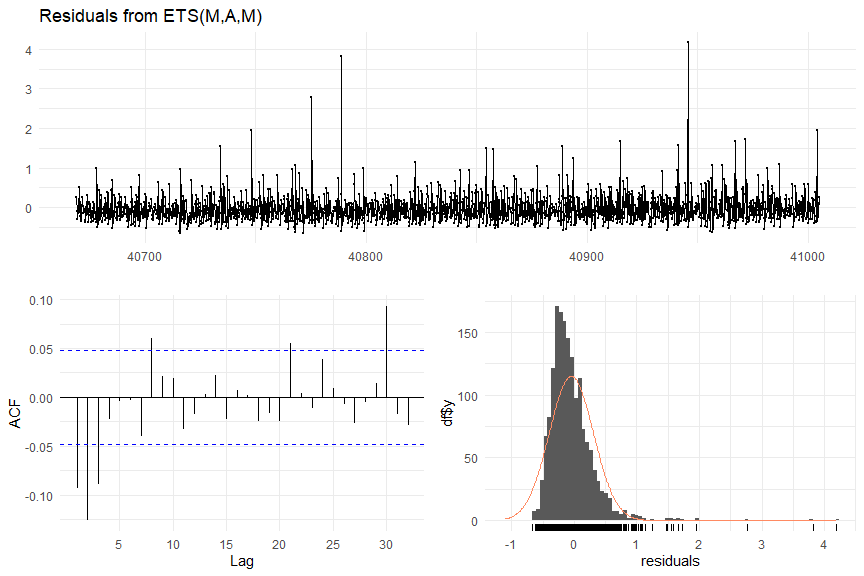
# Prepare data frame for results  
#dfr <- data.frame(matrix(nrow=0, ncol=11))  
#colnames(dfr) <- c('category', 'var', 'model', 'method', 'ME', 'RMSE', 'MAE', 'MPE', 'MAPE', 'MASE', 'ACF1')  
dfr <- data.frame(matrix(nrow=0, ncol=6))  
colnames(dfr) <- c('Category', 'Variable', 'Model', 'Method', 'MAPE', 'Ljung.Box')

# Create list to store ETS fit  
fit\_ets1 <- list()  
fit\_ets2 <- list()  
  
# ETS  
for (i in seq(1, 6)) {  
   
 # First variable in category i  
 fit\_ets1[[i]] <- ets(tsnew1[[i]])  
 dfr <- rbind(dfr, data.frame(  
 Category=paste0('S0', i),   
 Variable=paste0('V0', fcvars[[i]][1]),   
 Model='ETS',   
 Method=fit\_ets1[[i]]$method,   
 MAPE=accuracy(fit\_ets1[[i]])[5],  
 Ljung.Box=0 # temp, will fill in later when calculating residuals  
 ))  
   
 # Second variable in category i  
 fit\_ets2[[i]] <- ets(tsnew2[[i]])  
 dfr <- rbind(dfr, data.frame(  
 Category=paste0('S0', i),   
 Variable=paste0('V0', fcvars[[i]][2]),   
 Model='ETS',   
 Method=fit\_ets2[[i]]$method,  
 MAPE=accuracy(fit\_ets2[[i]])[5],  
 Ljung.Box=0 # temp, will fill in later when calculating residuals  
 ))  
   
}

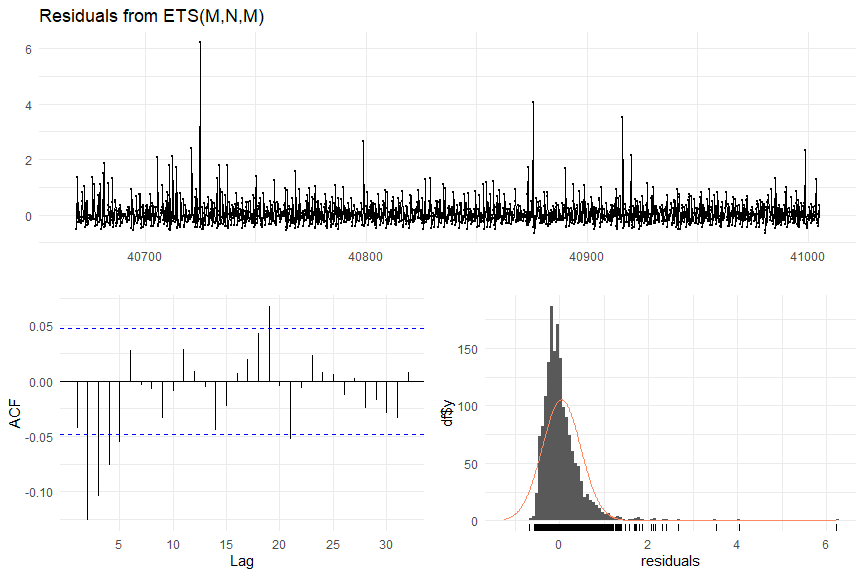
# Display residual plots  
for (i in seq(1, 6)) {  
 tmp\_res <- checkresiduals(fit\_ets1[[i]])  
 dfr[i \* 2 - 1, 'Ljung.Box'] <- tmp\_res$p.value  
 tmp\_res <- checkresiduals(fit\_ets2[[i]])  
 dfr[i \* 2, 'Ljung.Box'] <- tmp\_res$p.value  
}



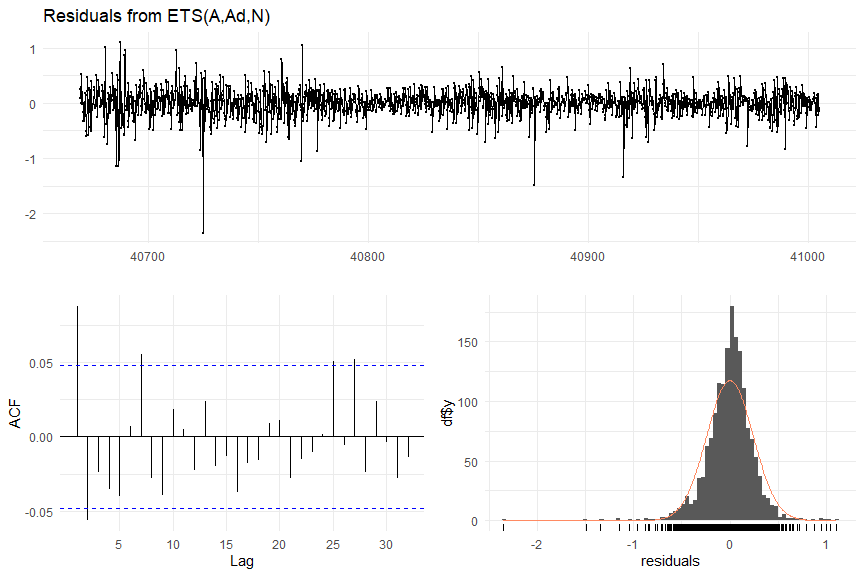
##   
## Ljung-Box test  
##   
## data: Residuals from ETS(M,N,N)  
## Q\* = 20.285, df = 10, p-value = 0.02667  
##   
## Model df: 0. Total lags used: 10



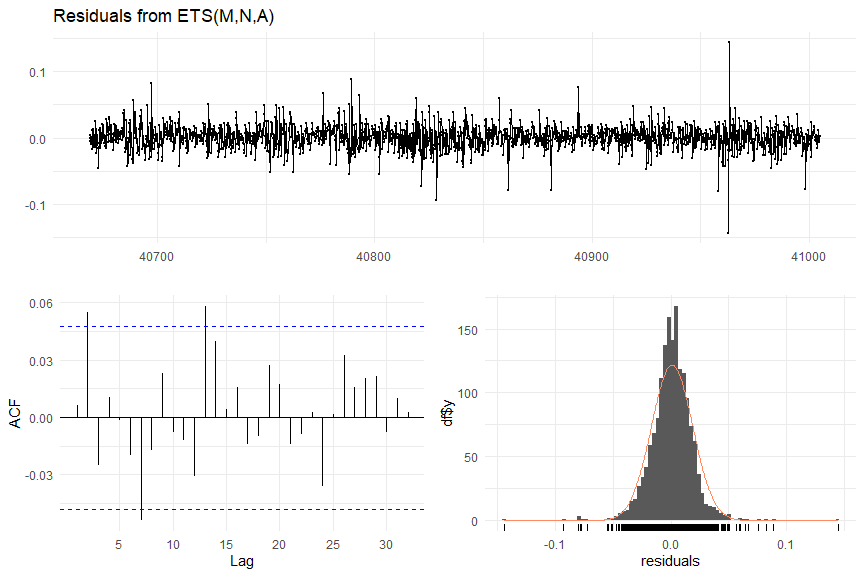
##   
## Ljung-Box test  
##   
## data: Residuals from ETS(M,A,M)  
## Q\* = 66.04, df = 10, p-value = 2.562e-10  
##   
## Model df: 0. Total lags used: 10



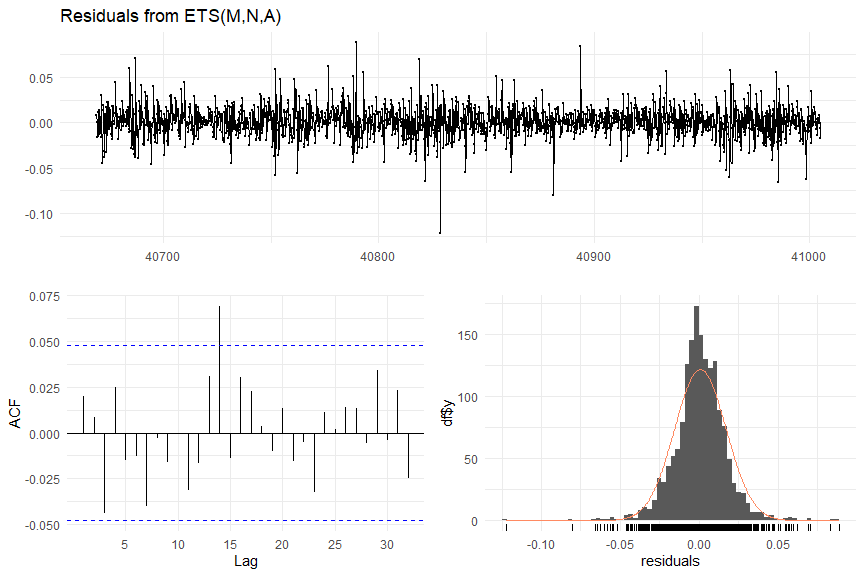
##   
## Ljung-Box test  
##   
## data: Residuals from ETS(M,N,M)  
## Q\* = 66.585, df = 10, p-value = 2.015e-10  
##   
## Model df: 0. Total lags used: 10



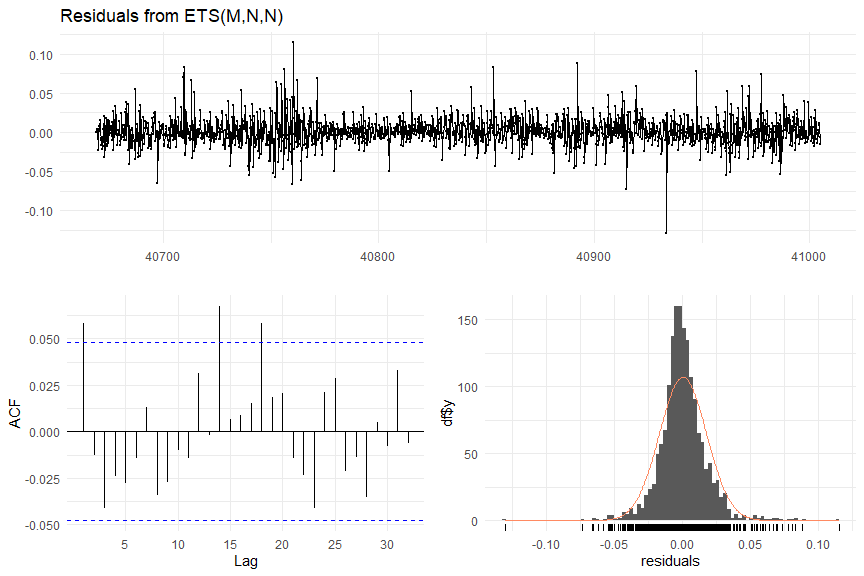
##   
## Ljung-Box test  
##   
## data: Residuals from ETS(A,Ad,N)  
## Q\* = 33.823, df = 10, p-value = 0.0001979  
##   
## Model df: 0. Total lags used: 10



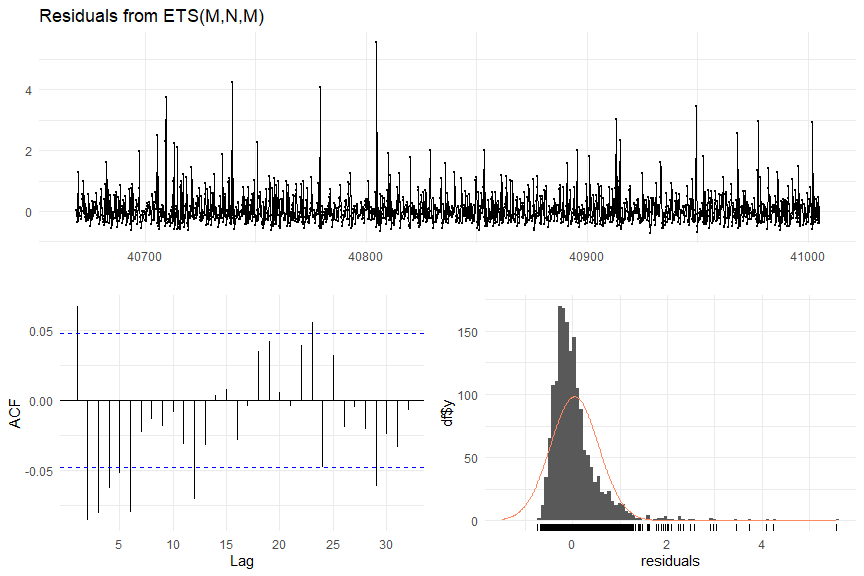
##   
## Ljung-Box test  
##   
## data: Residuals from ETS(M,N,A)  
## Q\* = 13.536, df = 10, p-value = 0.1952  
##   
## Model df: 0. Total lags used: 10



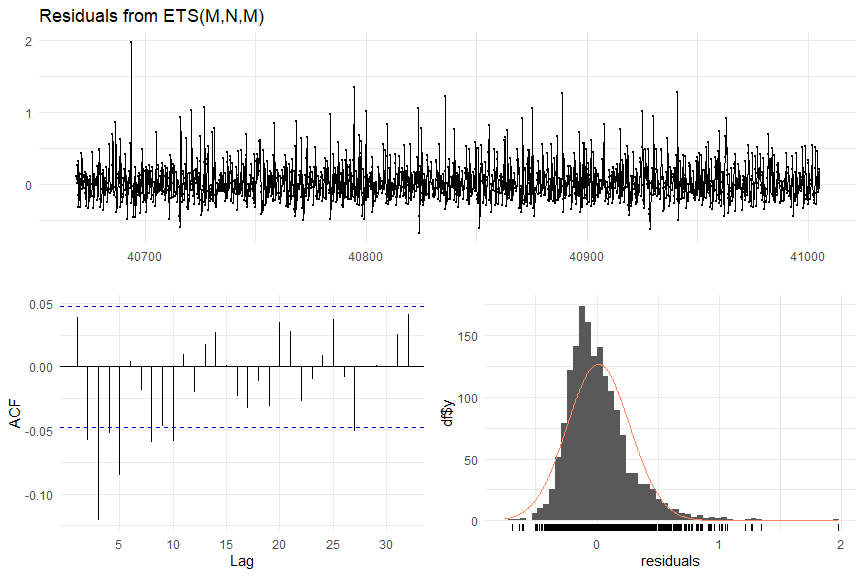
##   
## Ljung-Box test  
##   
## data: Residuals from ETS(M,N,A)  
## Q\* = 8.9033, df = 10, p-value = 0.5413  
##   
## Model df: 0. Total lags used: 10



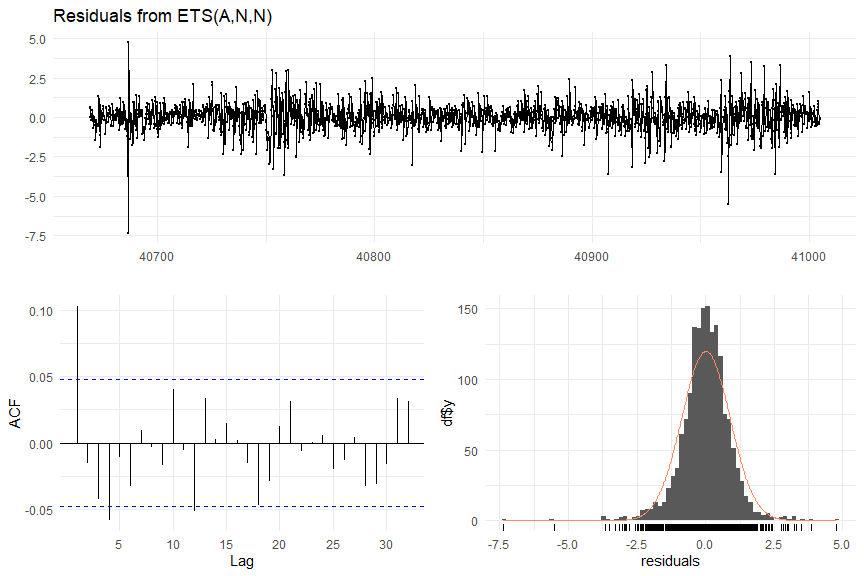
##   
## Ljung-Box test  
##   
## data: Residuals from ETS(M,N,N)  
## Q\* = 15.125, df = 10, p-value = 0.1276  
##   
## Model df: 0. Total lags used: 10



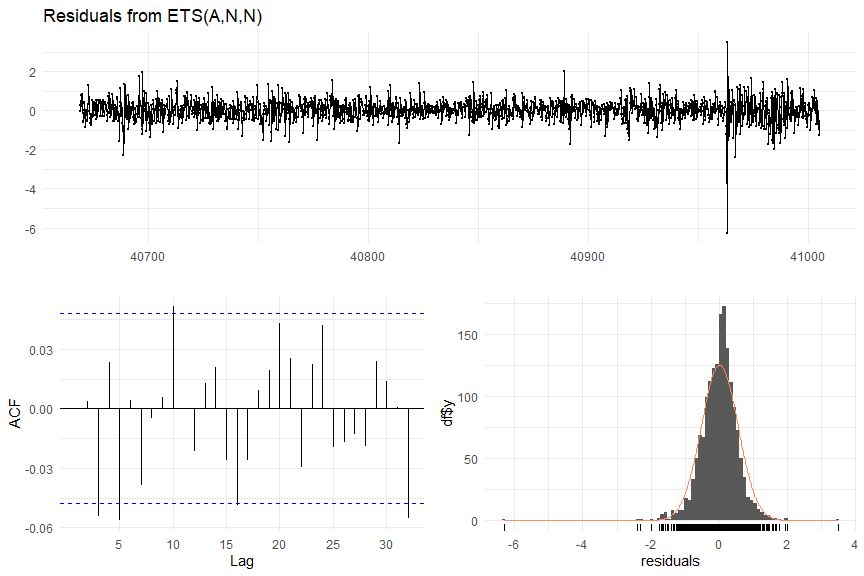
##   
## Ljung-Box test  
##   
## data: Residuals from ETS(M,N,M)  
## Q\* = 54.629, df = 10, p-value = 3.703e-08  
##   
## Model df: 0. Total lags used: 10



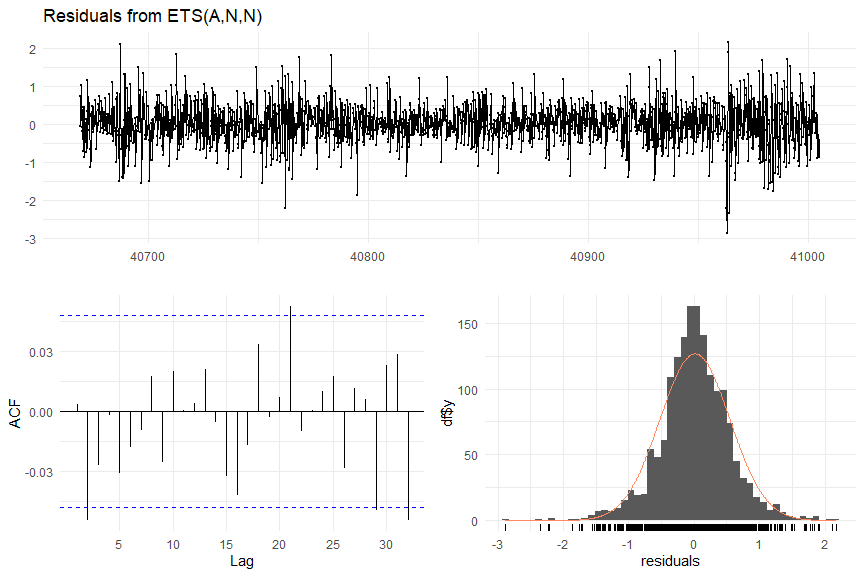
##   
## Ljung-Box test  
##   
## data: Residuals from ETS(M,N,M)  
## Q\* = 65.926, df = 10, p-value = 2.695e-10  
##   
## Model df: 0. Total lags used: 10



##   
## Ljung-Box test  
##   
## data: Residuals from ETS(A,N,N)  
## Q\* = 32.435, df = 10, p-value = 0.0003387  
##   
## Model df: 0. Total lags used: 10



##   
## Ljung-Box test  
##   
## data: Residuals from ETS(A,N,N)  
## Q\* = 18.391, df = 10, p-value = 0.04871  
##   
## Model df: 0. Total lags used: 10



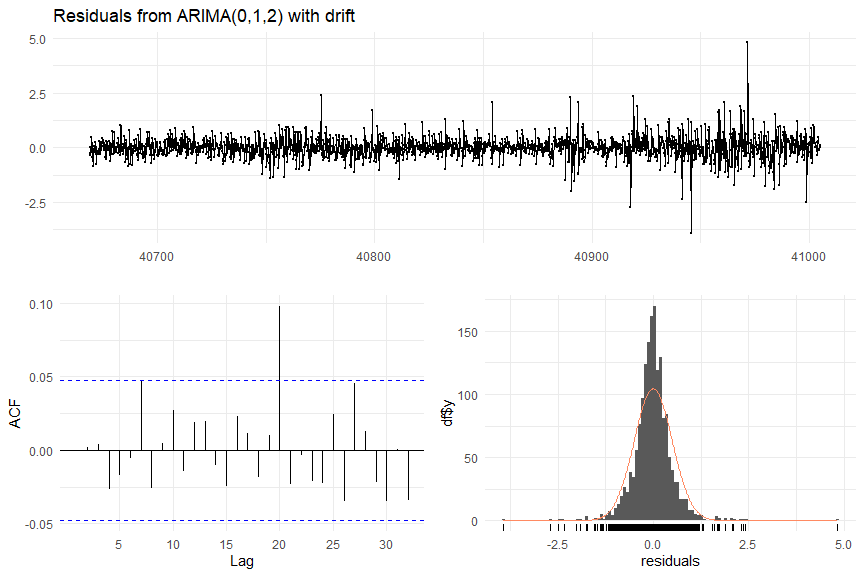
##   
## Ljung-Box test  
##   
## data: Residuals from ETS(A,N,N)  
## Q\* = 10.804, df = 10, p-value = 0.373  
##   
## Model df: 0. Total lags used: 10

For the most part, residuals look like white noise and are normally distributed.

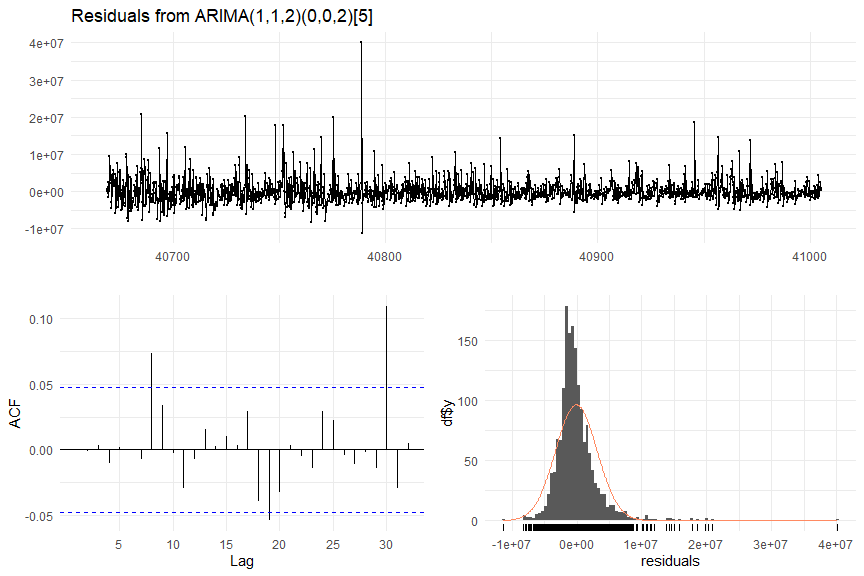
# Display results  
dfr %>%  
 kbl(caption='ETS Modeling Results') %>%  
 kable\_classic(full\_width=F)

# Create list to store ETS fit  
fit\_arima1 <- list()  
fit\_arima2 <- list()  
  
# Function to return friendly name of ARIMA method using the fit returned by the model  
ret\_arima\_name <- function (fit) {  
 tmp\_name <- paste0(  
 'ARIMA(', fit$arma[1],   
 ',', fit$arma[6],  
 ',', fit$arma[2],  
 ')(', fit$arma[3],  
 ',', fit$arma[7],  
 ',', fit$arma[4],  
 ')'  
 )  
 if ('drift' %in% names(fit$coef)) {  
 tmp\_name <- paste(tmp\_name, ' with drift')  
 }  
 return(tmp\_name)  
}  
  
# ARIMA modeling  
for (i in seq(1, 6)) {  
   
 fit\_arima1[[i]] <- auto.arima(tsnew1[[i]])  
 dfr <- rbind(dfr, data.frame(  
 Category=paste0('S0', i),   
 Variable=paste0('V0', fcvars[[i]][1]),   
 Model='ARIMA',   
 Method=ret\_arima\_name(fit\_arima1[[i]]),   
 MAPE=accuracy(fit\_arima1[[i]])[5],  
 Ljung.Box=0 # temp, will fill in later when calculating residuals  
 ))  
 fit\_arima2[[i]] <- auto.arima(tsnew2[[i]])  
 dfr <- rbind(dfr, data.frame(  
 Category=paste0('S0', i),   
 Variable=paste0('V0', fcvars[[i]][2]),   
 Model='ARIMA',   
 Method=ret\_arima\_name(fit\_arima2[[i]]),  
 MAPE=accuracy(fit\_arima1[[i]])[5],  
 Ljung.Box=0 # temp, will fill in later when calculating residuals  
 ))  
   
}

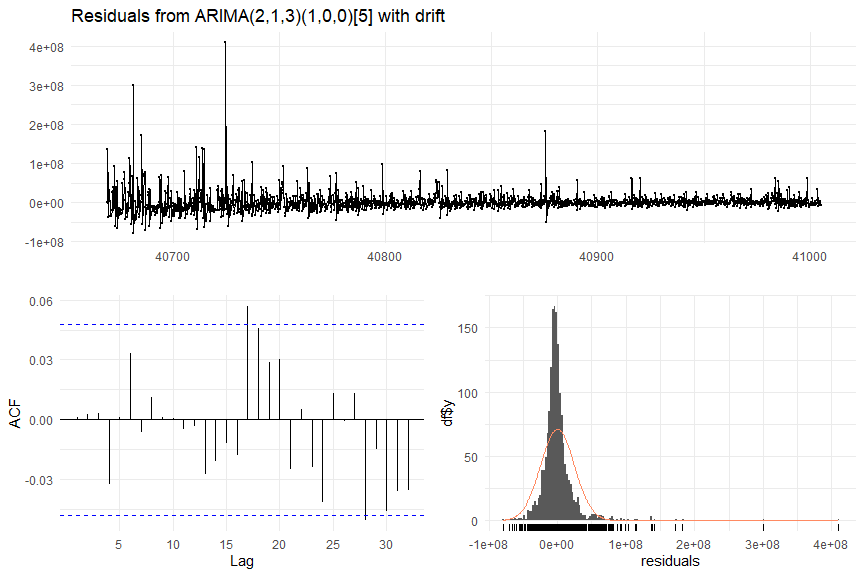
# Display residual plots  
for (i in seq(1, 6)) {  
 tmp\_res <- checkresiduals(fit\_arima1[[i]])  
 dfr[i \* 2 - 1 + 12, 'Ljung.Box'] <- tmp\_res$p.value  
 tmp\_res <- checkresiduals(fit\_arima2[[i]])  
 dfr[i \* 2 + 12, 'Ljung.Box'] <- tmp\_res$p.value  
}



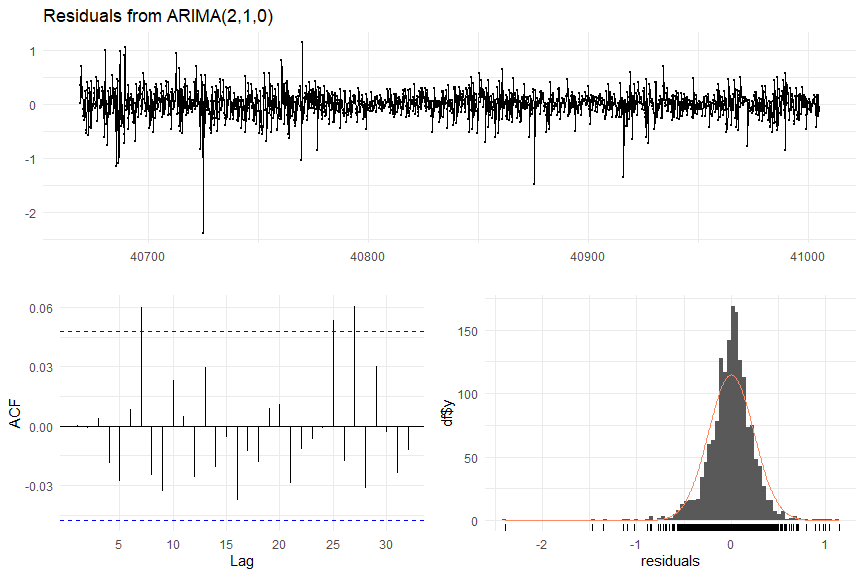
##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(0,1,2) with drift  
## Q\* = 7.9375, df = 8, p-value = 0.4396  
##   
## Model df: 2. Total lags used: 10



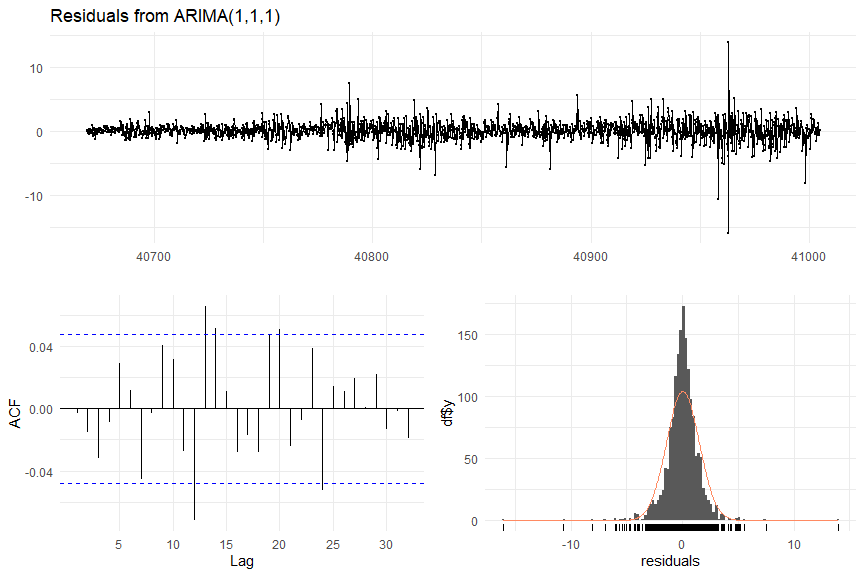
##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(1,1,2)(0,0,2)[5]  
## Q\* = 11.455, df = 5, p-value = 0.04307  
##   
## Model df: 5. Total lags used: 10



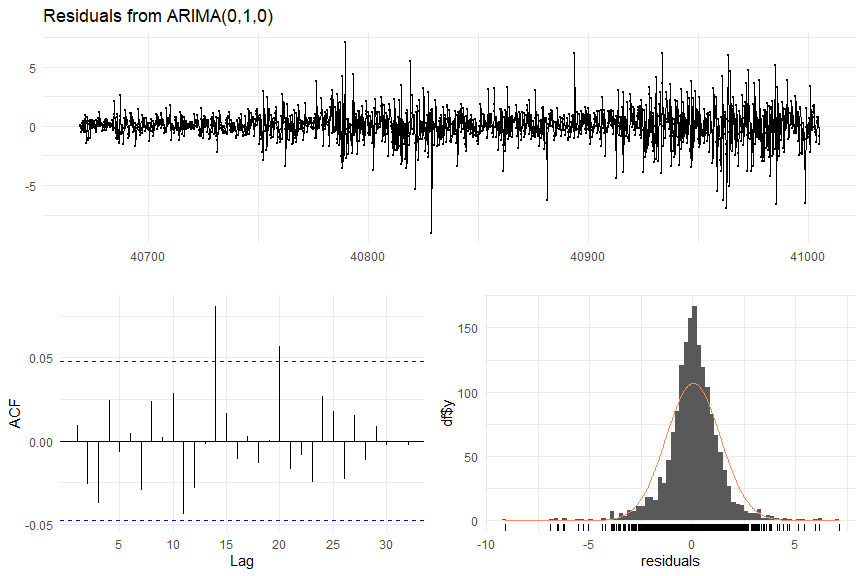
##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(2,1,3)(1,0,0)[5] with drift  
## Q\* = 3.9806, df = 4, p-value = 0.4086  
##   
## Model df: 6. Total lags used: 10



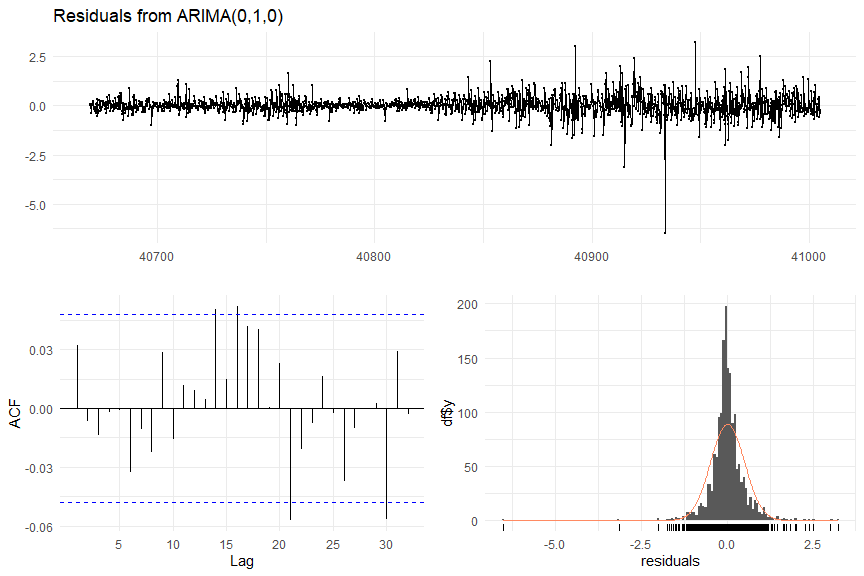
##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(2,1,0)  
## Q\* = 11.977, df = 8, p-value = 0.1522  
##   
## Model df: 2. Total lags used: 10



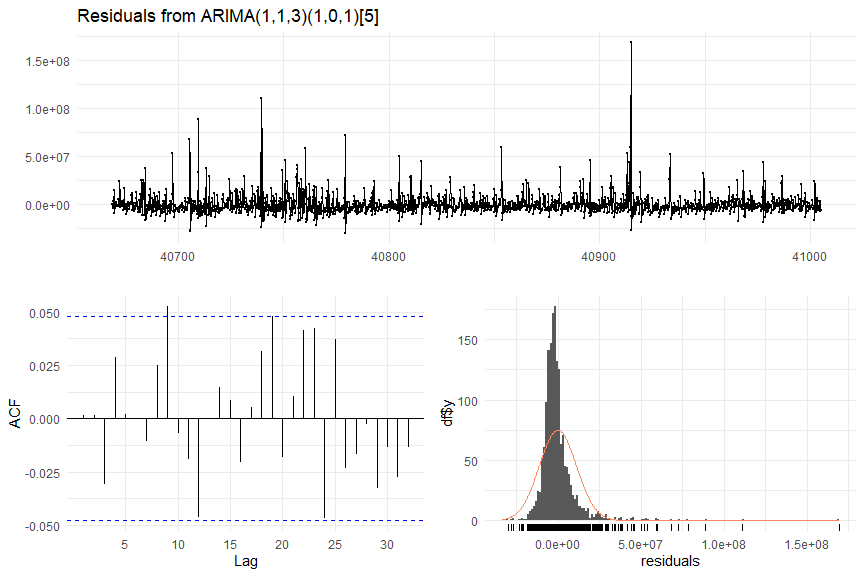
##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(1,1,1)  
## Q\* = 11.894, df = 8, p-value = 0.156  
##   
## Model df: 2. Total lags used: 10



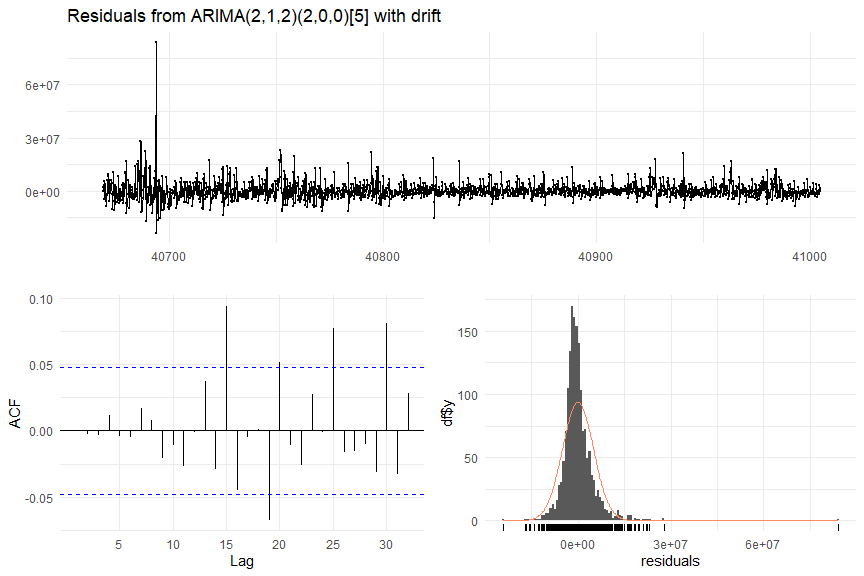
##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(0,1,0)  
## Q\* = 8.5574, df = 10, p-value = 0.5746  
##   
## Model df: 0. Total lags used: 10



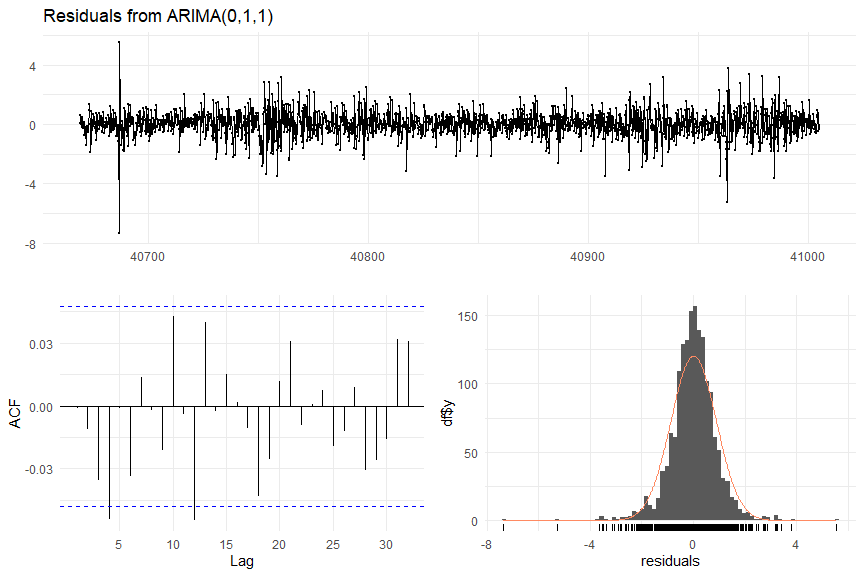
##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(0,1,0)  
## Q\* = 6.8036, df = 10, p-value = 0.7439  
##   
## Model df: 0. Total lags used: 10



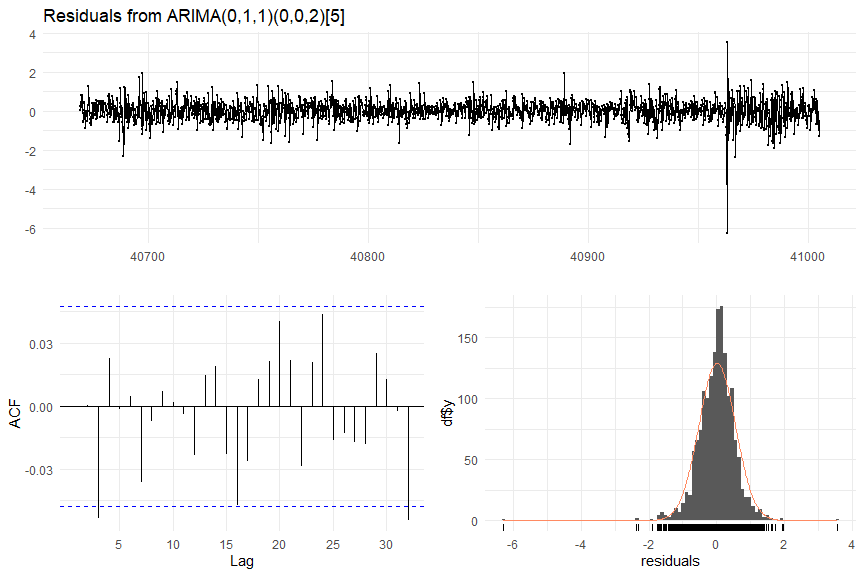
##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(1,1,3)(1,0,1)[5]  
## Q\* = 9.0584, df = 4, p-value = 0.05966  
##   
## Model df: 6. Total lags used: 10



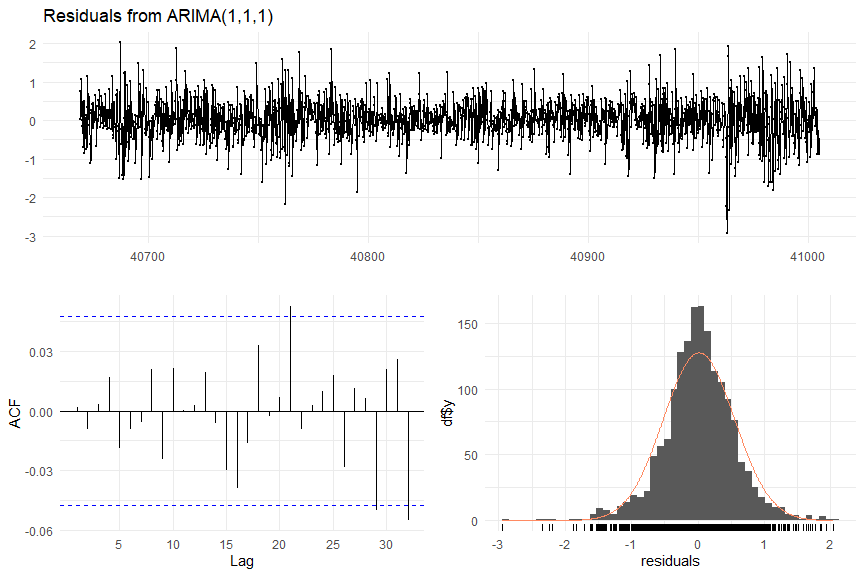
##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(2,1,2)(2,0,0)[5] with drift  
## Q\* = 1.8642, df = 4, p-value = 0.7607  
##   
## Model df: 6. Total lags used: 10



##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(0,1,1)  
## Q\* = 13.457, df = 9, p-value = 0.143  
##   
## Model df: 1. Total lags used: 10



##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(0,1,1)(0,0,2)[5]  
## Q\* = 8.1228, df = 7, p-value = 0.3219  
##   
## Model df: 3. Total lags used: 10



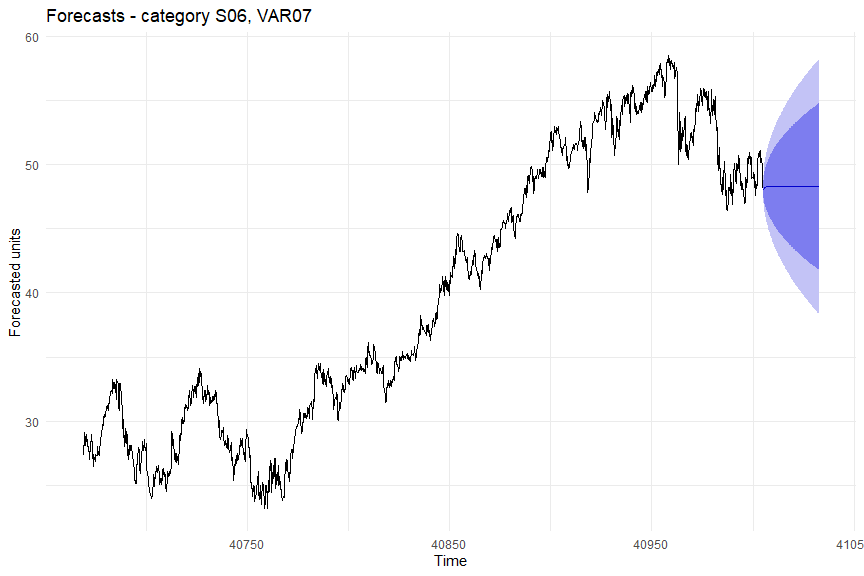
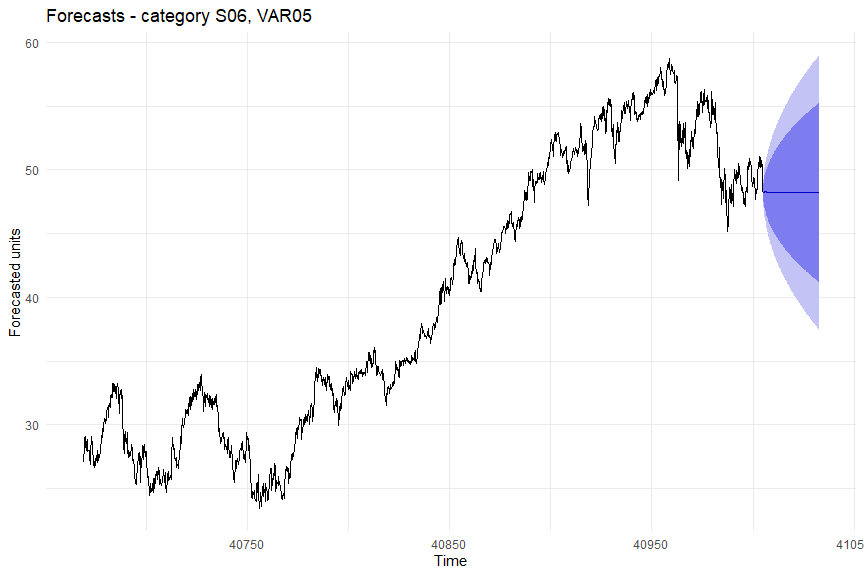
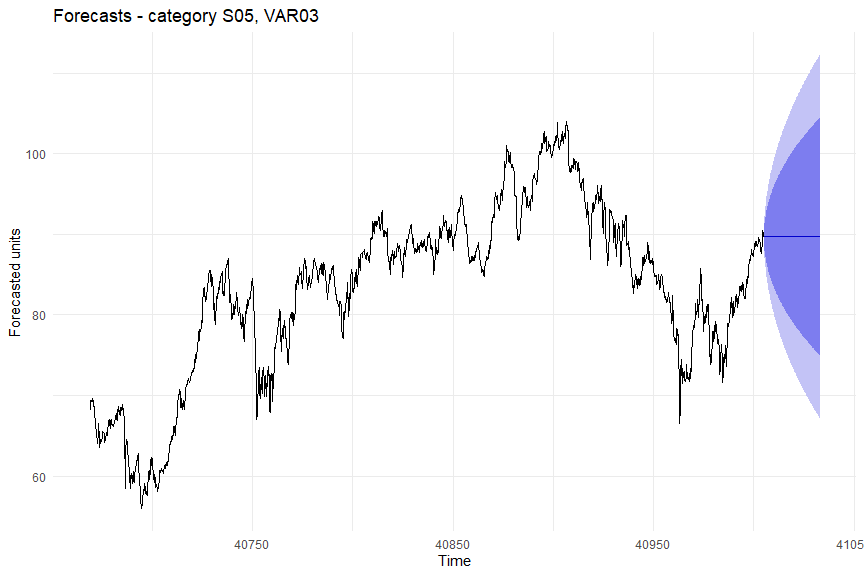
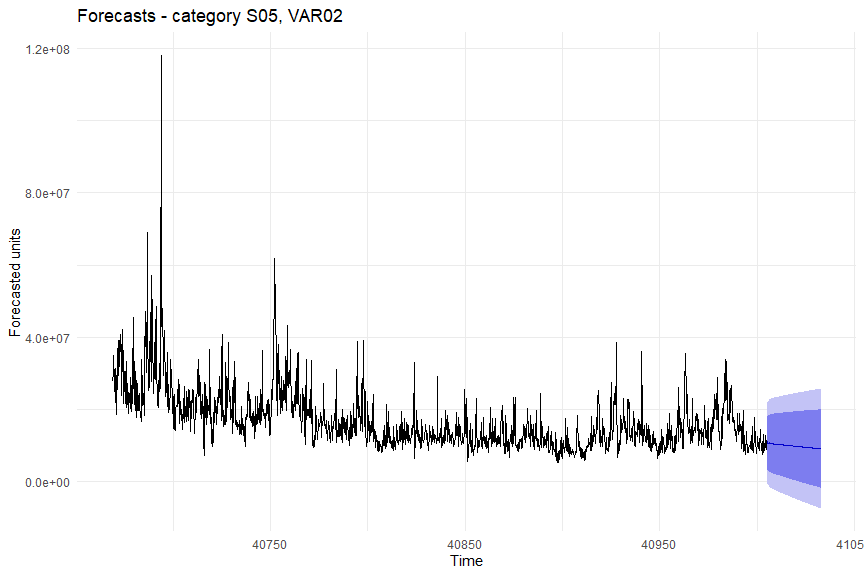
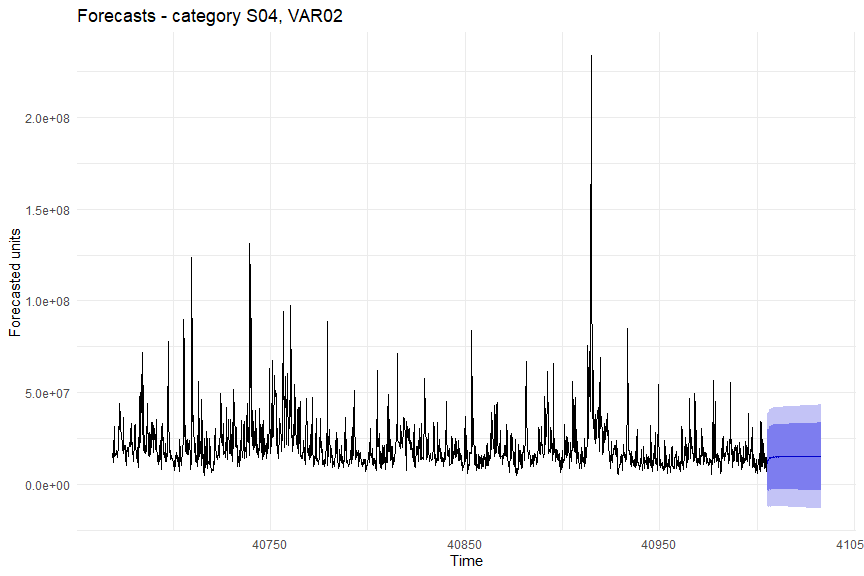
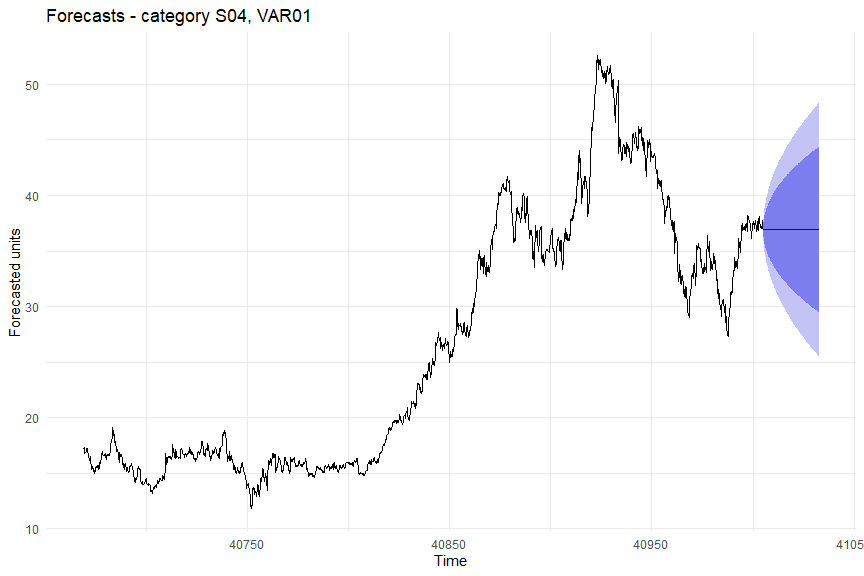
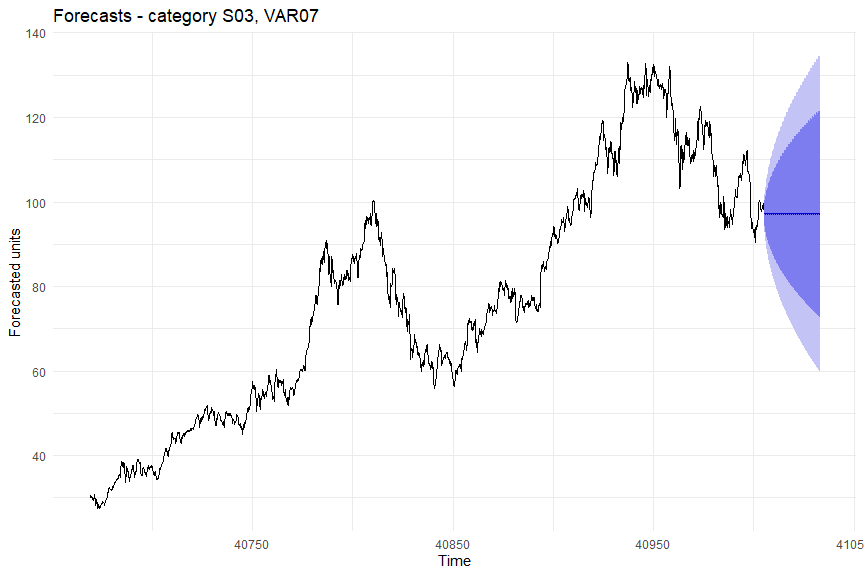
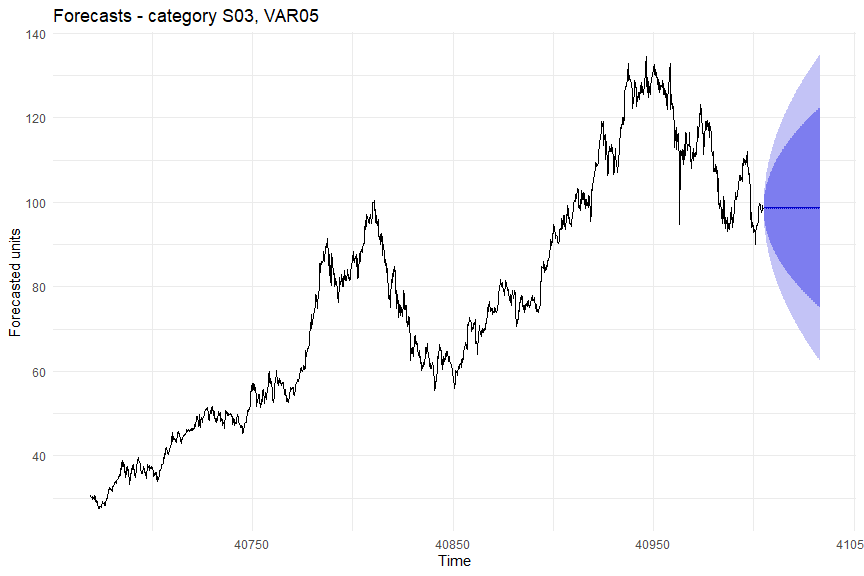
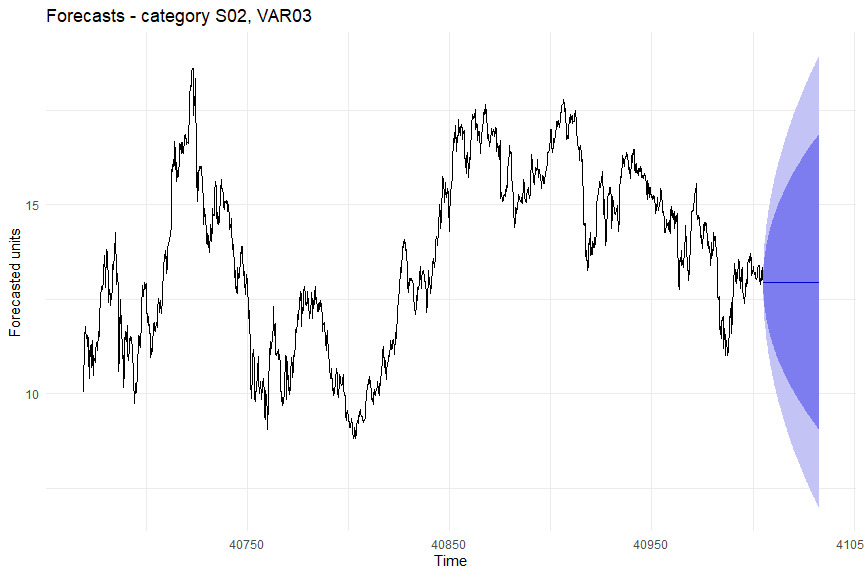
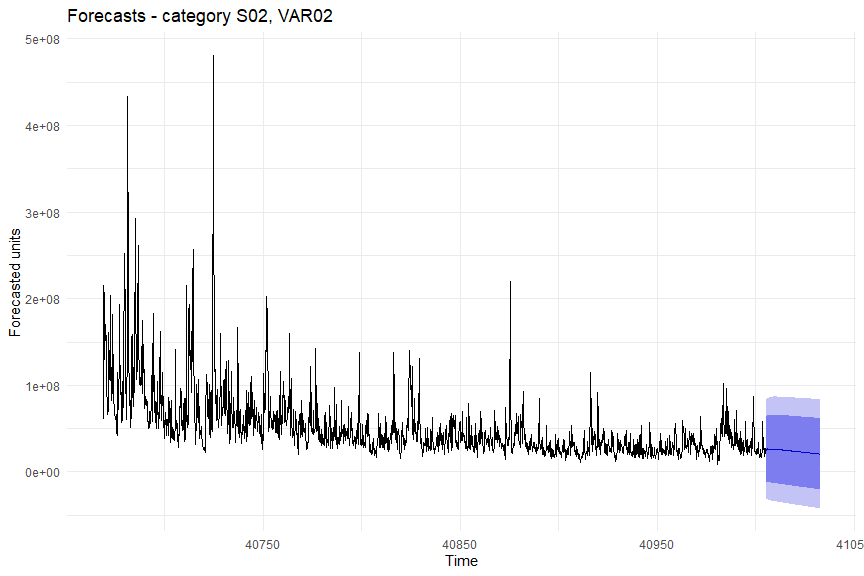
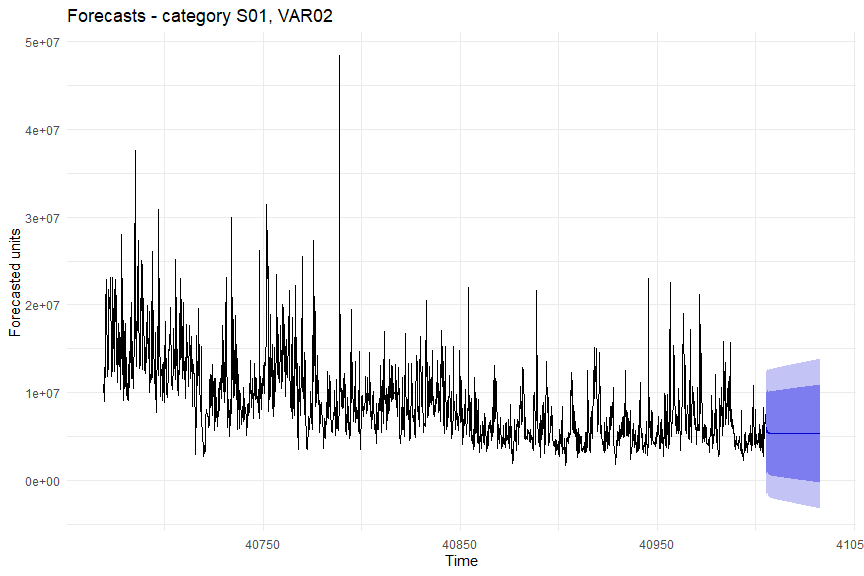
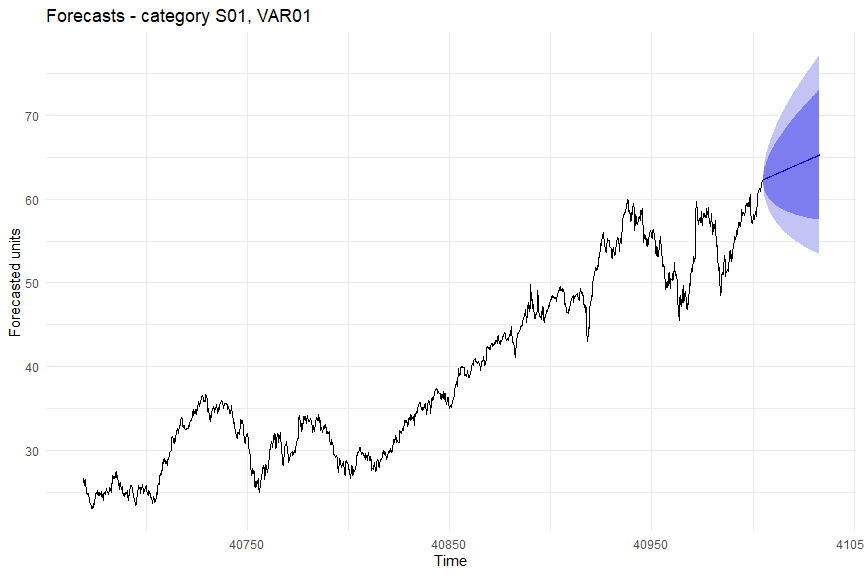
##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(1,1,1)  
## Q\* = 3.9985, df = 8, p-value = 0.8573  
##   
## Model df: 2. Total lags used: 10

As with the ETS models, residuals look like white noise and are normally distributed.

# Display results  
dfr %>%  
 arrange(Category, Variable, Model) %>%  
 kbl(caption='Modeling Results - ETS and ') %>%  
 kable\_classic(full\_width=F)

# Manually choose best model for now  
Selected.model <- c(1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0)  
  
# Choose the model with the lower MAPE for each category/var combination  
#dfr2 <- dfr %>%  
# #filter(Ljung.Box > 0.05) %>%  
# group\_by(Category, Variable) %>%  
# slice\_min(MAPE)  
dfr2 <- dfr %>%  
 arrange(Category, Variable, Model) %>%  
 cbind(Selected.model) %>%  
 filter(Selected.model==1) %>%  
 select(-Selected.model)  
colnames(dfr2) <- c('Category', 'Variable', 'Model', 'Method', 'MAPE', 'Ljung.Box')  
dfr2 %>%  
 kbl(caption='Best-performing models') %>%  
 kable\_classic(full\_width=F)

# Create variable to store forcasts  
fc1 <- list()  
fc2 <- list()  
  
# Create data frame to store forecasts  
dffc <- data.frame(matrix(nrow=0, ncol=3))  
colnames(dffc) <- c('Category', 'Variable', 'Forecast')  
  
for (i in seq(1, 6)) {  
   
 # First var in category  
 if (dfr2[2 \* i - 1, 'Model'] == 'ETS') {  
 fc1[[i]] <- fit\_ets1[[i]] %>% forecast(h=140)  
 } else {  
 fc1[[i]] <- fit\_arima1[[i]] %>% forecast(h=140)  
 }  
 p1 <- fc1[[i]] %>%  
 autoplot() +  
 ylab('Forecasted units') +  
 ggtitle(paste0('Forecasts - category S0', i, ', VAR0', fcvars[[i]][1]))  
 print(p1)  
  
 # First var in category  
 if (dfr2[2 \* i, 'Model'] == 'ETS') {  
 fc2[[i]] <- fit\_ets2[[i]] %>% forecast(h=140)  
 } else {  
 fc2[[i]] <- fit\_arima2[[i]] %>% forecast(h=140)  
 }  
 p2 <- fc2[[i]] %>%  
 autoplot() +  
 ylab('Forecasted units') +  
 ggtitle(paste0('Forecasts - category S0', i, ', VAR0', fcvars[[i]][2]))  
 print(p2)  
   
 # Store forecasts in df  
 dffc <- rbind(dffc, data.frame(  
 Category=paste0('S0', i),  
 Variable=paste0('V0', fcvars[[i]][1]),  
 Forecast=data.frame(fc1[[i]])['Point.Forecast']  
 ))  
 dffc <- rbind(dffc, data.frame(  
 Category=paste0('S0', i),  
 Variable=paste0('V0', fcvars[[i]][2]),  
 Forecast=data.frame(fc2[[i]])['Point.Forecast']  
 ))  
  
}



#df\_orig %>%  
# filter(SeriesInd > 43021) %>%  
# select(SeriesInd) %>%  
  
#dffc %>%  
# merge()  
  
# Display a few rows  
dffc %>%  
 head(10) %>%  
 kbl(caption='Forecast values (first 10 values)') %>%  
 kable\_classic(full\_width=F)

# Write forecasts  
  
#reference: https://www.statology.org/r-export-to-excel-multiple-sheets/  
#export each data frame to separate sheets in same Excel file  
#openxlsx::write.xlsx(forecasts, file = 'mydata.xlsx')

# Appendix 3: Time Series Model Detailed Explanations

**Exponential Smoothing**

Exponential Smoothing is based on using exponentially decaying weighted averages of past observations to predict future values. The exponential smoothing model can be broken up into three parts, the trend, the seasonal and the error component. Trend means a slope to the data and seasonal means that there is a regularly spaced in time repeating pattern in the data. The trend and seasonal components each have three settings in an exponential smoothing model.

The trend component could be not present, it could be additive, or it could be additive and damped. The first case is simple, if the trend component is not present there is no trend in the data. If the trend component is additive it means that there is a slope to the data, which could be changing throughout. The forecasted values would have a constant slope starting at the first predicted value with the last estimated slope value. In contrast, an additive damped model accounts for the fact that the additive model alone tends to over or under forecast so there is a damping parameter which eventually leads to a flat line in the future.

Next, the seasonal component could be not present, could be additive, or could be multiplicative. If the seasonal component is not present then there is no equally spaced in time repeating pattern in the data. If the seasonal component is additive it means that the seasonal component is constant throughout time whereas if it is multiplicative the seasonal component could grow or shrink in time.

Lastly, there is the error component. The error can be additive or multiplicative as well. This means that to predict the next value the error can either be added to the predicted values or multiplied by the predicted values.

Now that the trend, seasonal, and error components that exponential smoothing can include have been explained, the ets() function in r can be used to select what combination of the components is best and also to estimate the parameters for the model. Once the model has been selected, the forecast() function can be called with the model as the input to get our predictions.

**ARIMA**

Another commonly used time series forecasting model is called ARIMA. The fundamental difference is that rather than using components for trend and seasonality, ARIMA uses autocorrelations within the data to forecast future values. There are three parts that can make up an ARIMA, the autoregressive (AR), the differencing/integration (I), and the moving average (MA) parts.

A stationary model is one who could be measured at any time, and the time would not have to do with the value observed. To make a non-stationary dataset into a stationary dataset, differencing can be used. This means subtracting subsequent values or values at a seasonal interval. A unit root test can be used to assess if differencing is required. One specific unit test is called the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test where the null hypothesis asserts that the data is stationary and the alternative hypothesis is that the data is not stationary. Normally, autoregressive and moving average models are restricted to being stationary.

Starting with the autoregressive, AR, part of ARIMA, the future values are based on lagged values of the past. Depending on how many lagged values are used, is the order, p. Next, the moving average, MA, part of ARIMA, uses the historic error values in the model. Depending on how far many values back in error terms are used, that is the order, q. When integration/differencing is included, this is called a “non-seasonal” ARIMA model. To help determine the p and q values that are appropriate for the data, an ACF and a PACF (partial autocorrelation) plot can be used. This can be useful because the auto.arima() function does not check for all possible values of p,d, and q. So if the optimal values are not even checked, you would get a less accurate model.

**ETS vs. ARIMA**

In order to choose which model is better for a given time series data set, a train and test dataset can be created. Once the predicted models are obtained, they can be compared to the test with a value called “MAPE” Mean average percentage error. Whichever has the lower MAPE is superior.