Exam July 5, 2024

Anthony Tricarico

1 Exercise 1

Autocorrelation in a time series refers to how much a variable is correlated to past (lagged) versions of itself. One way to look and analyze the autocorrelation visually in a time series is through a correlogram. A correlogram shows the degree of autocorrelation of a variable across different time lags. Importantly, correlograms produced with R also show significance threshold (blue dashed lines) to help us determine if the autocorrelation at different time lags is statistically significant.

In case we wanted to go deeper and use a more quantitative method, we could rely on the Ljung-Box test that allows us to determine whether or not the variable follows a white noise process (i.e., it is randomly distributed) or whether there are patterns across the different time lags. For this test the H_0 (null hypothesis) is that the series comes from a white noise process, whereas H_1 (alternative hypothesis) states exactly the opposite. A small p value for this test is indicative of more support for H_1 and therefore indicates that the variable does not follow a white noise process.

2 Exercise 2

> **Caution**
>
> For this question I found it hard to understand whether we are asked to produce forecasts out-of-sample or if the forecasting mentioned refers also to in-sample forecasts. Anyway, I added some possible answers with the one flagged as **"Best"** assuming that we are asked to produce out-of-sample forecasts.

2.0.1 Potential Answer

We can use methods like **Time Series Cross Validation** or a simpler **Train-Test Split**. In the first case, since the model is trained on a subset of the data and asked to produce a forecast on a value it has not been trained on, we can still ensure the fairness of the process while having a **true** value for the comparison. Similarly, for the **Train-Test Split** we can use the training subset to estimate the parameters of the model and then test how well it performs by evaluating the forecasts it produces for the periods contained in the **Test** subset. Then, also in this case, the model will be evaluated fairly and **true** values will be available for the comparison with the predicted ones.

2.0.2 Best Answer

Another viable alternative is the use of bootstrapping methods to estimate potential future values based on past ones. This approach starts from the last observed value in the time series and generates new ones by sampling from the residual distribution and adding the randomly sampled residual to the last observed value in the time series at time T. Then the procedure goes on employing the same approach to estimate the values Y_{T+2}, Y_{T+3}, ... Y_{T+i} where i indicates the number of values for the variable Y we want to generate after the last observed one.

3 Exercise 3

For this exercise, it is important to have the necessary libraries installed. In case you do not have the `tswge` library you can install it with the following function: `install.packages("tswge")`

3.1 Point a)

```
library(fpp3)
```

```
Registered S3 method overwritten by 'tsibble':
  method                from
  as_tibble.grouped_df dplyr

-- Attaching packages --------------------------------------- fpp3 1.0.1 --

v tibble      3.2.1     v tsibble     1.1.5
v dplyr       1.1.4     v tsibbledata 0.4.1
v tidyr       1.3.1     v feasts      0.4.1
v lubridate   1.9.3     v fable       0.4.1
v ggplot2     3.5.1

-- Conflicts ------------------------------------------------ fpp3_conflicts --
x lubridate::date()    masks base::date()
x dplyr::filter()      masks stats::filter()
x tsibble::intersect() masks base::intersect()
x tsibble::interval()  masks lubridate::interval()
x dplyr::lag()         masks stats::lag()
x tsibble::setdiff()   masks base::setdiff()
x tsibble::union()     masks base::union()
```

```
library(tswge)
```

```
Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo
```

```
Attaching package: 'tswge'

The following object is masked from 'package:datasets':

    uspop
```

```
df <- freight
```

```
View(df)
```

it is not a dataset, just a sequence of numbers, so we'll need to create first a sequence of dates and then move on to turn it into a coherent `tsibble` object.

We can create a sequence of dates using the following approach.

```
dates <- seq.Date(from = as.Date("2011-01-01"), to = as.Date("2020-12-01"), by = "month")
```

Then we can go on and create a `tsibble` using the omonymous function and specifying that we want to create two columns (`"series"` and "`dates`") and we assign to them the specific values of the time series (`df`) and the sequence of dates we just created (`dates`)

```
df_ts <- tsibble("series" = df,
        "dates" = dates,
        index = "dates") %>%
  mutate(dates = yearmonth(dates))
```

3.2 Point b)

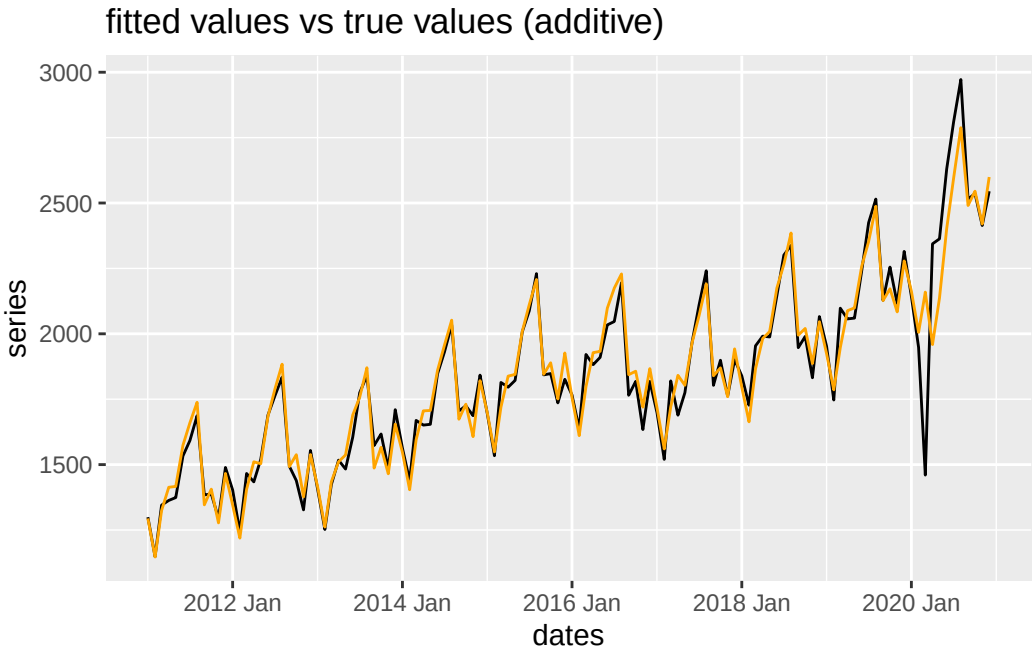
Now we are asked to fit the additive and multiplicative HW's model to the time series created in Section 3.1. The way we do this is by using the `model` function along with the `ETS` function to specify the required models.

```
add_model <- df_ts %>%
  model(
    "additive" = ETS(series ~ error("A") + trend("A") + season("A"))
  )

mult_model <- df_ts %>%
  model(
    "multiplicative" = ETS(series ~ error("M") + trend("A") + season("M"))
  )
```

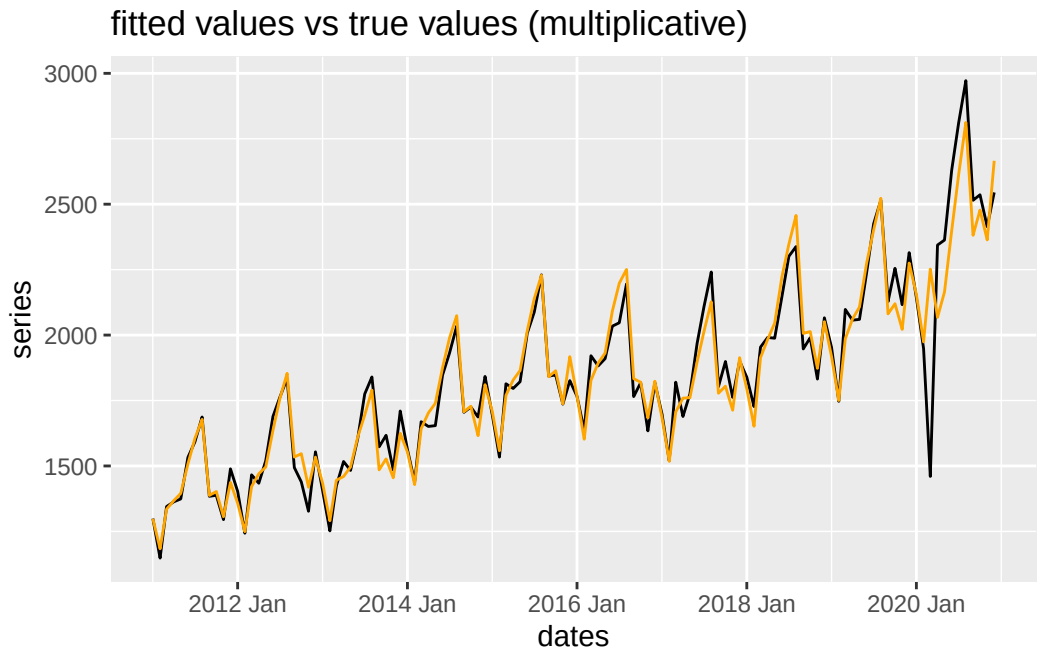
3.3 Point c)

```
ggplot(augment(add_model), aes(x = dates)) +
  geom_line(aes(y = series, color = "black"), color = "black") +
  geom_line(aes(y = .fitted, color = "orange"), color = "orange") +
  labs(title = "fitted values vs true values (additive)")
```

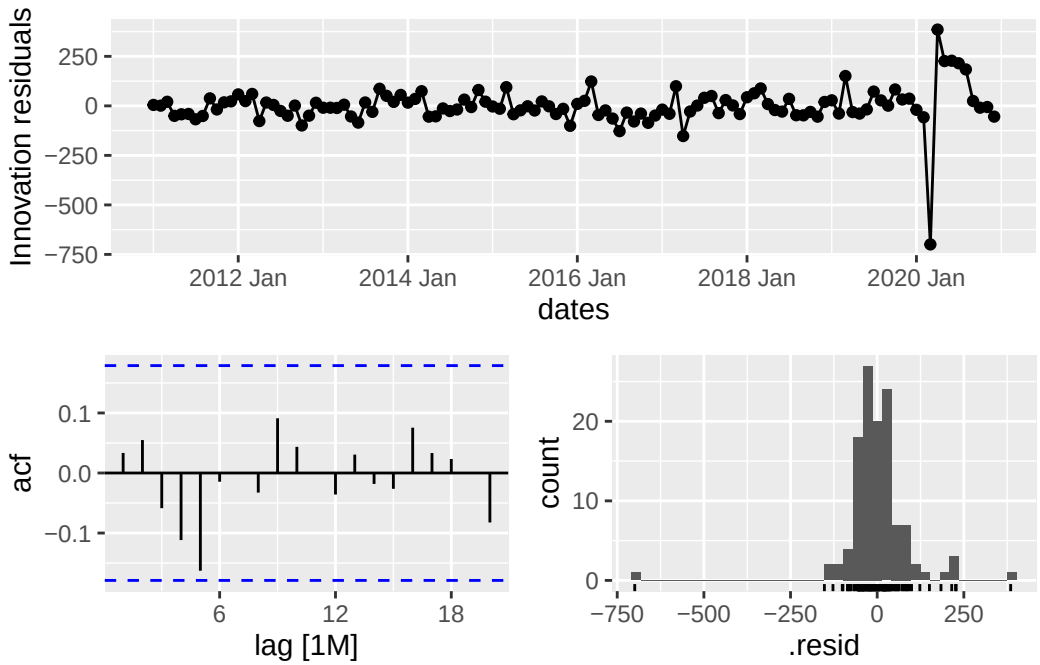


The series appear to have a seasonality of varying intensity and this is not captured very well by the additive model, which is better to model constant seasonal patterns. However, as we can see from the plot below representing the multiplicative model, when there are relevant shocks due to unexpected exogenous factors (i.e., a pandemic) the multiplicative model tends to incorporate this pattern into subsequent periods. In this case we can see that the fitted values from the multiplicative model are pretty much always below the true values).

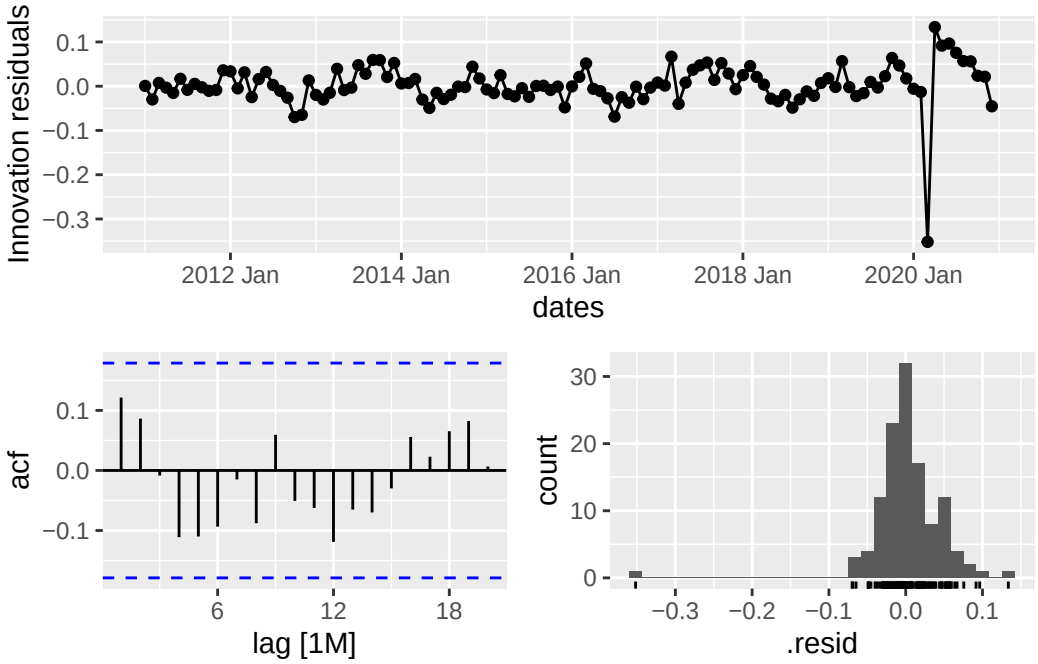
```
ggplot(augment(mult_model), aes(x = dates)) +
  geom_line(aes(y = series, color = "black"), color = "black") +
  geom_line(aes(y = .fitted, color = "orange"), color = "orange") +
  labs(title = "fitted values vs true values (multiplicative)")
```




```
gg_tsresiduals(add_model)
```




```
gg_tsresiduals(mult_model)
```




```
accuracy(add_model)$RMSE
```

```
[1] 96.37689
```

```
accuracy(mult_model)$RMSE
```

```
[1] 100.4274
```

Also, given the accuracy measures produced above, we can see that the additive model performs better than the multiplicative one since it has the lower RMSE between the two models.

3.4 Point d)

This can be done easily using the `forecast` function and the model under consideration.

```
forecast(add_model, h = 12)
```

```
# A fable: 12 x 4 [1M]
# Key:     .model [1]
   .model      dates
   <chr>       <mth>
 1 additive 2021 Jan
 2 additive 2021 Feb
 3 additive 2021 Mar
 4 additive 2021 Apr
 5 additive 2021 May
```

```
 6 additive 2021 Jun
 7 additive 2021 Jul
 8 additive 2021 Aug
 9 additive 2021 Sep
10 additive 2021 Oct
11 additive 2021 Nov
12 additive 2021 Dec
# i 2 more variables: series <dist>, .mean <dbl>
```

4 Exercise 4

4.1 Point a)

Here we create a sequence of daily dates as before and then we add it to the `ozona` dataset before turning it into a `tsibble` object using the `as_tsibble` function.

```
dates <- seq.Date(from = as.Date("2019-06-01"), to = as.Date("2019-07-31"), by = "day")
```

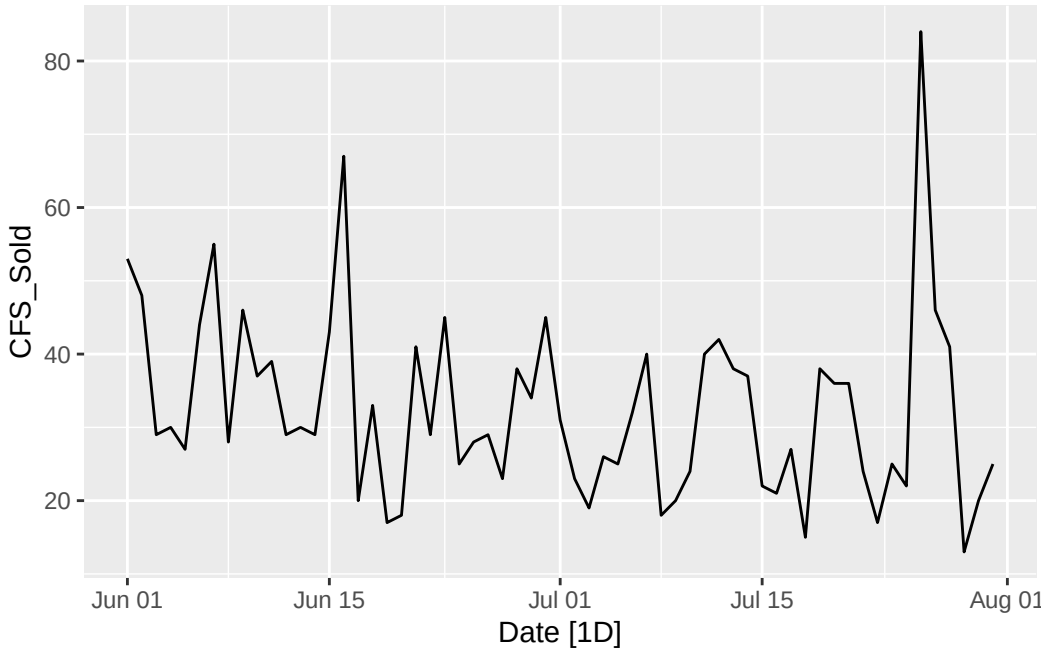
```
df <- ozona

df_ts <- df %>%
  mutate(Date = dates) %>%
  as_tsibble(index = Date)
```

4.2 Point b)

```
autoplot(df_ts)
```

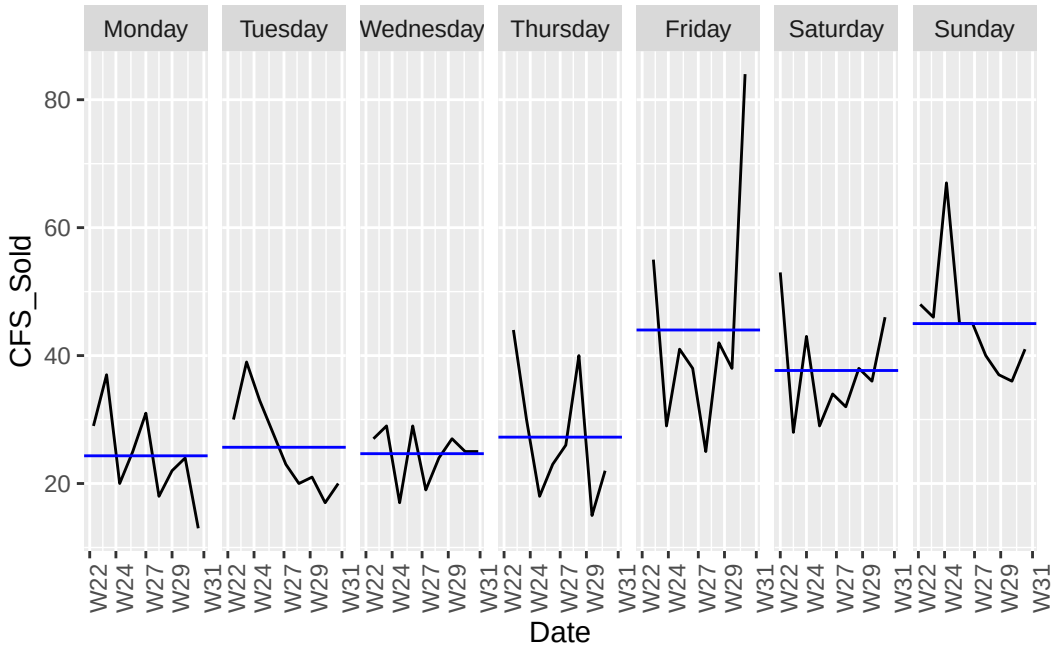
```
Plot variable not specified, automatically selected `.vars = CFS_Sold`
```



There appears to be some seasonality as every week follows roughly the same pattern. At the beginning of the week there are, on average, less CFS sold. We can check this further with a subseries plot.

```
gg_subseries(df_ts, period = "1w")
```

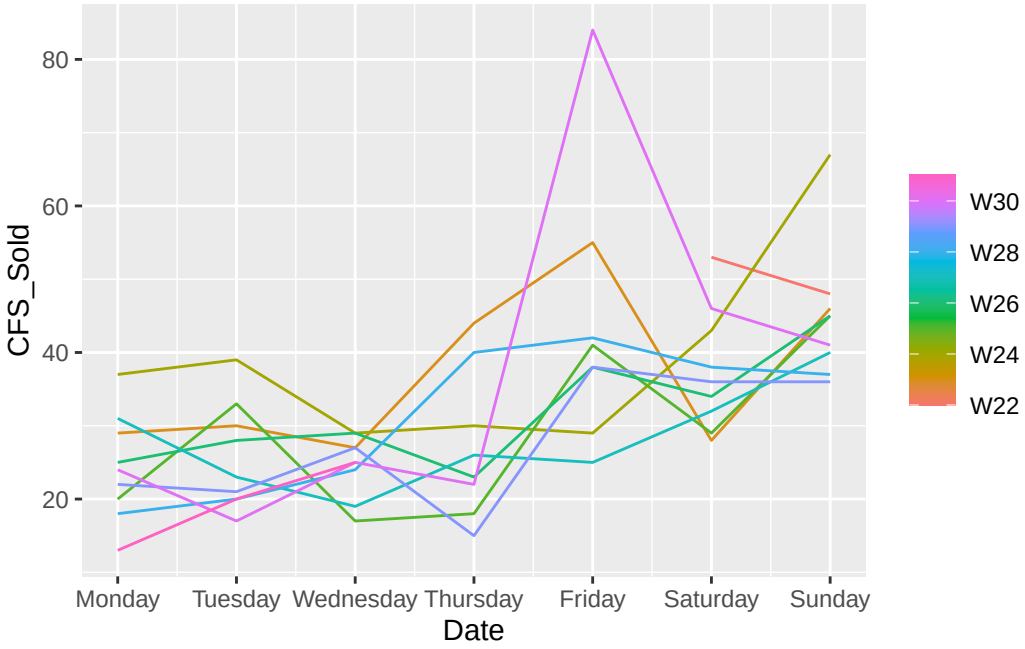
```
Plot variable not specified, automatically selected `y = CFS_Sold`
```



There does not appear to be any significant trend across the different days, however, it is clear that the number of CFS sold is higher on average during the weekend, compared to other days of the week.

```
gg_season(df_ts, period = "1w")
```

```
Plot variable not specified, automatically selected `y = CFS_Sold`
```



Apart from week 27 and week 24 all other weeks show a peak on Friday.