AAEC4484/AAEC(STAT)5484: Applied Economic Forecasting

Your Name Here

Homework #4 - Spring 2025

Instructions: Where necessary ensure that your graphs and visuals have proper titles and axis labels. Refer to the output, whenever appropriate, when discussing the results. **Creativity (coupled with relevance) will be rewarded.**

Q1: Classical Multiplicative Decomposition

In this exercise, we will explore the classical decomposition. Unlike our example in class, we will try to use a multiplicative method.

Our variable of interest is *US Retail Sales: Restaurants and Other Eating Places* (symbol: `MRTSSM7225USN`) from the `FRED`.

1. **In a single step**, import the data from the `FRED` database using the `quantmod` package. Be sure to convert the data to a `tsibble` object and store it as `food`.
2. Produce an `autoplot` of the `food` series along with a `gg_subseries` plot. Use the `patchwork` package to combine the two plots.

- **Briefly** comment on any pattern you observe. Why do you suppose there was that noticeable drop in sales in March and April 2020?
- Kenneth suggested that a multiplicative decomposition is appropriate for this series? Do you agree with his assessment? **Remember to explain your reasoning. A simple yes or no will not suffice.**

To help with your titling, please see the series definition at the FRED website

3. With visuals and our observations in tow, we are ready to perform our decomposition. It would help to bear in mind that, with a multiplicative decomposition, the functional form is:

$$y_t = T_t \times S_t \times R_t$$

where T_t is the trend-cycle component, S_t is the seasonal component, and R_t is the remainder component.

Step i. Using an appropriate centered moving average, $m - MA$ model, extract the trend-cycle component, T_t. Recall that we might need to doubly smooth the series to get it to be symmetric.

Present a plot of your trend-cycle element on top of the original data.

Step ii. Using the trend-cycle computed above, compute and plot the **detrended** series

Step iii (a). Using a `TSLM` model, extract the seasonal component, S_t. Also, plot your seasonal component.

Step iii(b). Produce a plot of the actual data (in darkgray) and overlay the `seasonally-adjusted` variable. **Note: You will need to compute the `seasonally-adjusted` variable given your calculations in Step iii(a).**

Step iv. Lastly, compute and plot the `remainder` component.

Step v. Produce a plot of the data and three components. Store your plot as `plot1`. Do not present the plot here.

Please order your variables (graphs) as Data » Trend » Seasonal » Remainder.

Hint: I would like to see you

(i) make use of the `pivot_longer()` function to reshape your data. You can select the 4 columns of interest here.
(ii) convert your variable names to `factors` so that `R` respects the "hierarchy" and is not ordered alphabetically in the faceted graphs.
(iii) use the `facet_wrap()` function to stack your plots in a single column (so they resemble the results of the built-in function).
(iv) use the `ggtitle()` function to add the title "My Decomposition" to your plot.

Step v(b). Use the `classical_decomposition()` function, within the `model()` function, to create `R`'s version of the **multiplicative** decomposition of the series. Store this plot as `plot2`. Again, do not present the plot here.

- Add the title "R's Decomposition" to your plot.

Step vi. Present a 1x2 plot of `plot1` and `plot2`.

4. Now, confirm that your decomposition was correct by **plotting the multiplication of the three extracted components (on y-axis)** against the **actual data (on x-axis)**.

- Be sure to add in a 45^o line to see how the scatter plot stacks up against this line.
- Produce this plot using the `ggplot` function.
- Get fancy with it. Add a y-axis title that uses the `bquote()` function to display the $\widehat{y}$ symbol.

5. Produce an autoplot() of the `actual` data, food, and `autolayer()` your computed `trend-cycle` and `seasonally-adjusted` variables.

Ensure that your series are correctly labeled. You might also need to change the colors of each of the series.

Q2: Forecasting with Time Series Decomposition

In our analysis below, we will again pull data from the `FRED` database using the `quantmod` package. We will use the `getSymbols` command to import the *non-seasonally adjusted monthly U.S. unemployment* (`UNRATENSA`) data from `FRED`.

1. Following the codes above, store the series as a `ts` object called `unrate`.
2. Next, produce an `autoplot()` of the `unrate.raw` series and **briefly** explain the dynamics you observe. Were there any potentially concerning outliers?
3. Given the unusually volatile unemployment observed during the COVID-19 pandemic, we will remove 2020 and beyond from our sample. In the codes that follow, drop all data points after December 2019. Overwrite the original `unrate` variable.
4. Report `autoplot()`, `ACF()`, and `gg_subseries` of the new `unrate` data. Discuss any noticeable trend and/or seasonality in the data. Also, based on your plot, are you comfortable with using an additive model? **Be sure to explain why or why not.**

Hints: - You might want to set the `lag_max = 36` in your ACF plot. - Yes, it is very likely that you do not observe any strong seasonality in the data and ACF plots. You might want to use the `gg_subseries` to confirm this.

5. Now assign the values in `unrate` up to and including December 2016 as a training set called `unrate.train`. Assign the values **after** December 2016 to a variable called `unrate.test`.

Use a graph to show the split between the training and test sets.

6. Using the **training set** above and an `STL` decomposition with fixed seasonal windows and `robust = TRUE`, present the plot of the decomposition of the `unrate.train` series. You will find it best to store the model result first.
7. Superimpose the `seasonally-adjusted` and `trend-cycle` series onto the training data.

You might need to change your line colors to ensure that each series is visible.

8. Use an `arima` method to produce predictions of the **seasonally adjusted** portion of the unemployment data over the length of the test period.

- Please store your model forecast results in a variable called `arima.unrate`.
- Create an `autoplot` of `arima.unrate` (be sure to include the data from the training set).

9. Now, produce a forecast of the seasonal component of the unemployment data over the test period. For simplicity, we will use the seasonal naïve method. Save the forecasts as `arima.seas.unrate` and produce a plot of the test data and the forecasted values.
10. Combine the forecasts from (8) and (9) to produce a "reseasonalized" forecasts of the original unemployment data. Store this forecast as `pred.unrate`. Next, produce a plot displaying the reseasonalized forecast and the actual test data (`unrate.test`).

Hint: You might find it helpful to (i) convert `arima.unrate` and `arima.seas.unrate` to tibble objects; (ii) `select` the index and the `.mean` columns from each; (iii) join the two tibbles by the index column; (iv) sum the two `.mean` columns to get the reseasonalized forecast.

11. Briefly discuss your observation from the plot above.

Note: The emphasis of your discussion should be on how well the `arima` forecast from the STL model captures the dynamics of the actual data that we tried to forecast.

12. One of the most desirable properties of the `STL` function is that we can allow for changing seasonality (and flexible, yet robust, trend-cycle decomposition). Again, using the **training set** and an `STL` decomposition with `robust = TRUE`, present the plot of the decomposition of the `unrate.train` series. You can allow the function to select the trend and seasons windows automatically.

Again, you will find it best to store the model result first. Save this as `fit2`.

13. Comment on any discernible differences between the two decompositions (in Parts 6 and 12). For completeness, you should produce the decompositions as a 1x2 plot.

Ensure each plot has a proper title to help with the comparison.

Using the `decomposition_model()` function

You would have noticed that reseasonalizing the forecasts can become cumbersome. Also, it is not always easy to back-transform the results to the original scale. To help with this, we can use the `decomposition_model()` function to dictate the forecast method to be used on the seasonally adjusted component. The forecasts are then conducted (using the additive model) and transformed with proper confidence intervals added.

14. For both the **fixed** and **changing seasonality** models with `robust = TRUE` (Parts 6 & 12, respectively), use the `naive`, `drift`, `arima`, and `Holt`'s methods to forecast the seasonally adjusted component. Save your models into a variable called `mod.fit` before forecasting. Store your forecasts as `mod.all`.

Note:

- **Please name all 8 models appropriately in a single `model()` command.**
- You are not required to plot the forecasts here; you just need to store them.

15. Produce a plot of the model forecasts (stored in `mod.all`) for all 8 models and the *test data*. **Be sure to turn the PIs off.**

Which method appears to do the best job (visually) of forecasting the test portion of the data?

16. Compare the accuracy of your forecasts.

- Which model is chosen by the `RMSE`? How about `MAPE`? As usual, be sure to explain how you arrived at your conclusion.
- Produce your model accuracy statistics as a `kable` with digits rounded to 3 dps.

17. Do the residuals from the "preferred model" stored in `mod.fit` appear to be white noise? Supplement your discussion using the `gg_tsresidual` function and a Ljung-Box test with a lag of $2 \times m$.