AAEC 4484/AAEC(STAT) 5484: Applied Economic Forecasting

Your Name Here

Homework #5 - Spring 2024

Instructions: In all cases, please ensure, where necessary, that your graphs and visuals have proper titles and axis labels. Refer to the output, whenever appropriate, when discussing the results. **Lastly, remember that creativity (coupled with relevance) will be rewarded.**

Q1: Classical Decomposition

In the class notes, we explored the manual computation/coding of the Classical decomposition of an Additive series. Here, I would like you to reproduce that same idea but for a series with a multiplicative pattern.

This exercise requires that you pull data on "Advance Retail Sales: Retail Trade" (symbol: `RSXFSN`) from the St. Louis Federal Reserve's economic database, `FRED` (https://fred.stlouisfed.org/).

1. Import the *non-seasonally adjusted Advance Retail Sales: Retail Trade* (`RSXFSN`) data from `FRED`. Declare as a `tsibble` object using the appropriate manipulations. Store this series as `retail`.
2. Plot a graph of the `retail` series and **briefly** comment on any pattern you observe. Do you agree that a multiplicative decomposition is appropriate for this series? Why or why not? *To help with your titling, please see the series definition at the FRED website*
3. With these observations, we are now 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 $m - MA$ 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. Using regression analysis, extract the seasonal component, S_t. Also, plot your seasonal component.

Step iii(b). Using the fitted values (our predicted seasonal component), S_t from Step iii, *compute* and *plot* the `seasonally-adjusted` variable. **Overlay the original data as well.**

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`.

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

This will require that you convert your variable names to `factors` so that `R` respects the "hierarchy" and is not ordered alphabetically in the faceted graphs. Your graphs should be stacked in a single column (so they resemble the results of the built-in function).

Step v(b). Use the `classical_decomposition()` function to create the **multiplicative** decomposition of the series. Store this plot as `plot2`.

Step vi. Use the `gridarrange()` function to create a 2 column plot of your two plots(`plot1`,`plot2`) stored above.

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

5. Produce an autoplot() of the `actual` data, retail, 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.

6. Isaac was neither particularly pleased nor impressed with the simplicity of the manual decomposition I introduced in class. He is convinced that the U.S. Census Bureau's method for seasonal adjustments is **far** superior (his thoughts, not mine).

He has asked you to compare your manual decomposition with the seasonally adjusted series from the FRED database. Return to the FRED website and download the **seasonally adjusted series** for the same retail sales data. Your ticker of interest is `RSXFS`.

Your tasks are as follows:

a. Import the `RSXFS` series and produce a time plot of this seasonally adjusted series. Overlay *your* calculated seasonally adjusted series from Step iii(b). Save your plot as `s.plot1`.

b. Since both series overlap at points, it might be challenging to discern the differences. Producing a plot of the deviations between the two series would also help. This will help you see how well your manual decomposition captures the seasonality in the official series (I am referring to `RSXFS` as the official series).

Hint: You might find it most straightforward to proceed as follows:
i. Return to your `retail` series and `select` only the columns corresponding to your index and `Seasonally Adjusted` series columns.
ii. `Pipe` and left join this with the `RSXFS` series from `FRED` on the index column.
iii. Create a new column called `diff` that captures the difference between the two series. Produce your plot and save it as `s.plot2`.

c. Produce a 2 row plot of `s.plot1` and `s.plot2` using the `grid.arrange()` function.

d. Does your classical decomposition happen to capture the general dynamics of the FRED series? Why or why not?

Hint: Your answer should come back to the disadvantages of the classical decomposition and the potential difference in the method used by the U.S. Census Bureau (the folks responsible for collecting this data) for seasonal adjustments. You will find this link helpful to help the quality of your responses: https://www.census.gov/topics/research/seasonal-adjustment.html

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.

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 $2 \times m$.