Lab 2: E-Commerce Retail Sales Prediction

Liang Chen, Timothy Quek, Dili Wang

11/03/2019

Introduction

In the following time series analysis, we attempt to accurately model the Retail Sales data published by the Federal Reserve Econcomic Database (FRED) to predict quarterly E-commerce Retail Sales as a percent of total sales. The provided data entails E-commerce Retail Sales percentage for each quarter, ranging from Q4 of 1999 to Q2 of 2019. While we explore the use of simpler modeling techniques, such as polynomial linear regression, our analysis focuses comparing different versions of optimized AutoRegressive Integrated Moving Average models (ARIMA) with and without seasonal components on both the full data and the seasonally-adjusted data. Our in-sample models' fit are measured using various metrics, including Akaike's Information Criterion (AIC), Corrected Akaike's Information Criterion (AICc), Bayesian Information Criterion (BIC), and Root Mean Square Error. In addition, forcasting performance is also measured using a small testing subset of the last 6 quarters (Q1 of 2018 to Q2 of 2019) and measuring the RMSE of the forecasted data. Finally, we use our top performing models to predict sales percentages for a 6 quarter window, spanning from Q3 of 2019 to Q4 of 2020.

Part I

```
# Reading in the CSV file
df <- read.csv("ECOMPCTNSA.csv")
dfts = ts(data = df$ECOMPCTNSA, start = c(1999, 4), frequency = 4)
# Examine the data structure and the first and last 10 values
str(df)
```

```
## 'data.frame':    79 obs. of  2 variables:
##  $ DATE      : Factor w/ 79 levels "1999-10-01","2000-01-01",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ ECOMPCTNSA: num  0.7 0.8 0.8 0.9 1.2 1.1 1 1 1.3 1.3 ...
```

```
names(df)
```

```
## [1] "DATE"       "ECOMPCTNSA"
```

```
head(df)
```

```
##         DATE ECOMPCTNSA
## 1 1999-10-01        0.7
## 2 2000-01-01        0.8
## 3 2000-04-01        0.8
## 4 2000-07-01        0.9
## 5 2000-10-01        1.2
## 6 2001-01-01        1.1
```

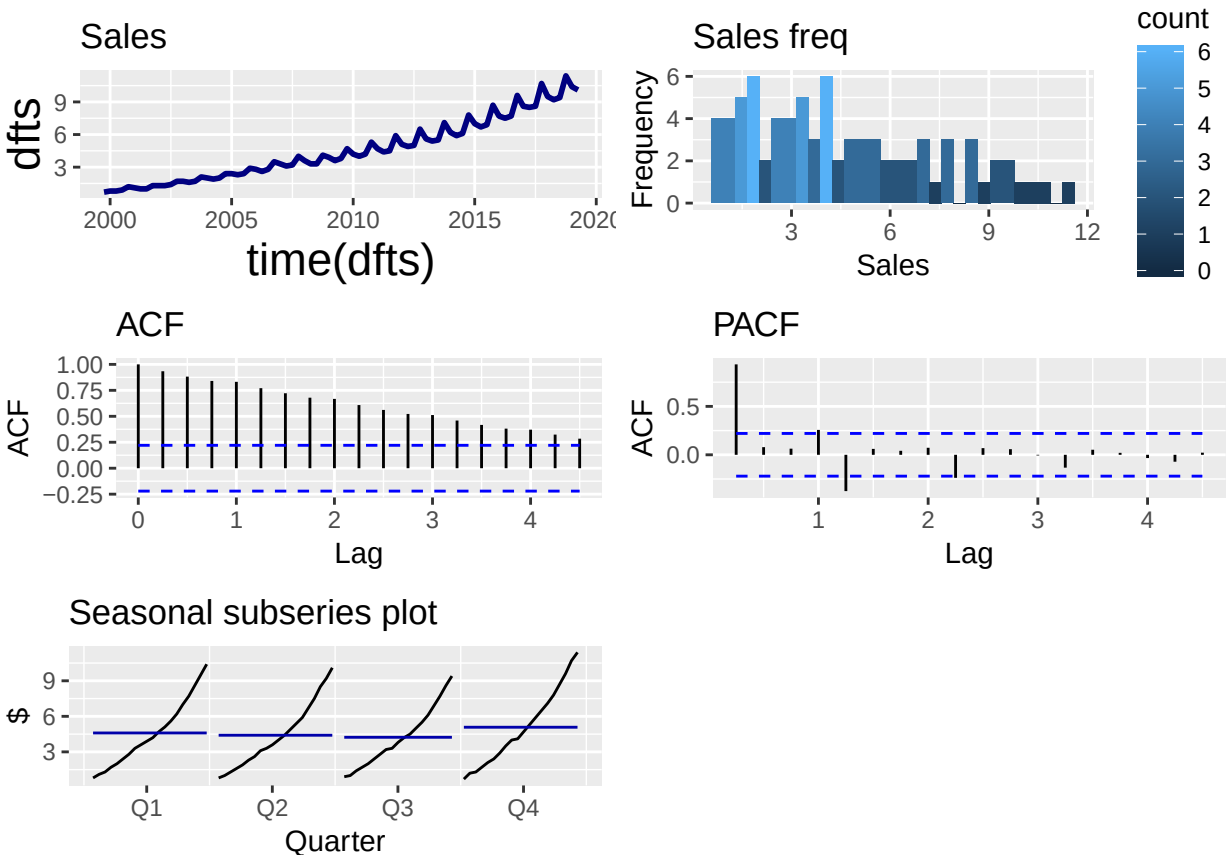
```
tail(df)
```

```
##          DATE ECOMPCTNSA
## 74 2018-01-01        9.5
## 75 2018-04-01        9.2
## 76 2018-07-01        9.4
## 77 2018-10-01       11.4
## 78 2019-01-01       10.4
```

```
## 79 2019-04-01       10.1
```

The CSV dataset from FRED is read into R as a dataframe. It comprises 2 variables, a DATE (read in as a factor) in YYYY-MM-DD format and a variable ECOMPCTNSA which contains the data for the E-Commerce Retail Sales, measured as a Percent of Total Sales. The dataframe is loaded as a time series and examined.

```
# Plot of the raw time series
p1 = ggplot(dfts, aes(x = time(dfts), y = dfts)) + geom_line(colour = "navy",
    size = 1) + ggtitle("Sales") + theme(axis.title = element_text(size = rel(1.5)))
# Histogram
p2 = ggplot(dfts, aes(x = dfts)) + geom_histogram(aes(fill = ..count..)) +
    ggtitle("Sales freq") + xlab("Sales") + ylab("Frequency")
# ACF plot
p3 = autoplot(acf(dfts, plot = FALSE)) + ggtitle("ACF")
# PACF plot
p4 = autoplot(pacf(dfts, plot = FALSE)) + ggtitle("PACF")
# Seasonal subseries plot
p5 = ggsubseriesplot(dfts) + ylab("$") + ggtitle("Seasonal subseries plot")
grid.arrange(p1, p2, p3, p4, p5, ncol = 2, nrow = 3)
```



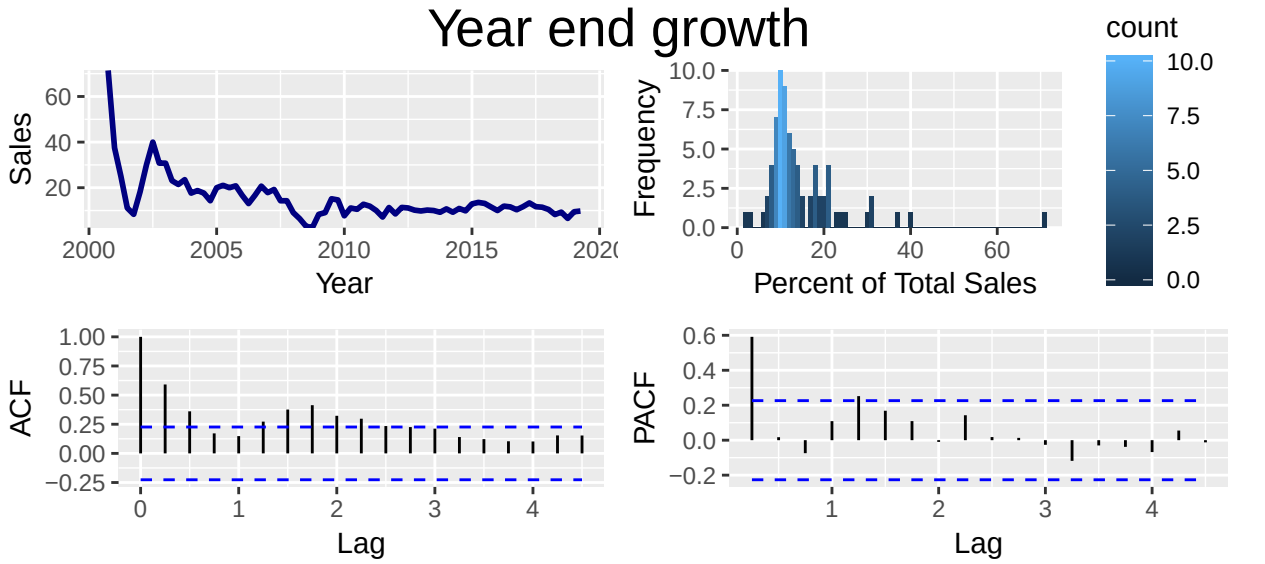
Based on the raw time series plot of the e-commerce retail sales (as a percentage of total sales) by time, we notice that there is a definite upward trend from 1999 and 2019 with seasonality in the data. The amplitude of oscillations, which represent the variations from the trend due to seasonality, also appear to increase as time increases. That is to say, for more recent data, the seasonality variations are greatest and the variation appears to decrease as we go backwards in time towards Q1 of 1999.

The histogram shows a greater number of quarters with lower e-commerce retail sales and a lower number of quarters with high e-commerce retail sales. The ACF gradually tapers off, while the PACF falls off sharply

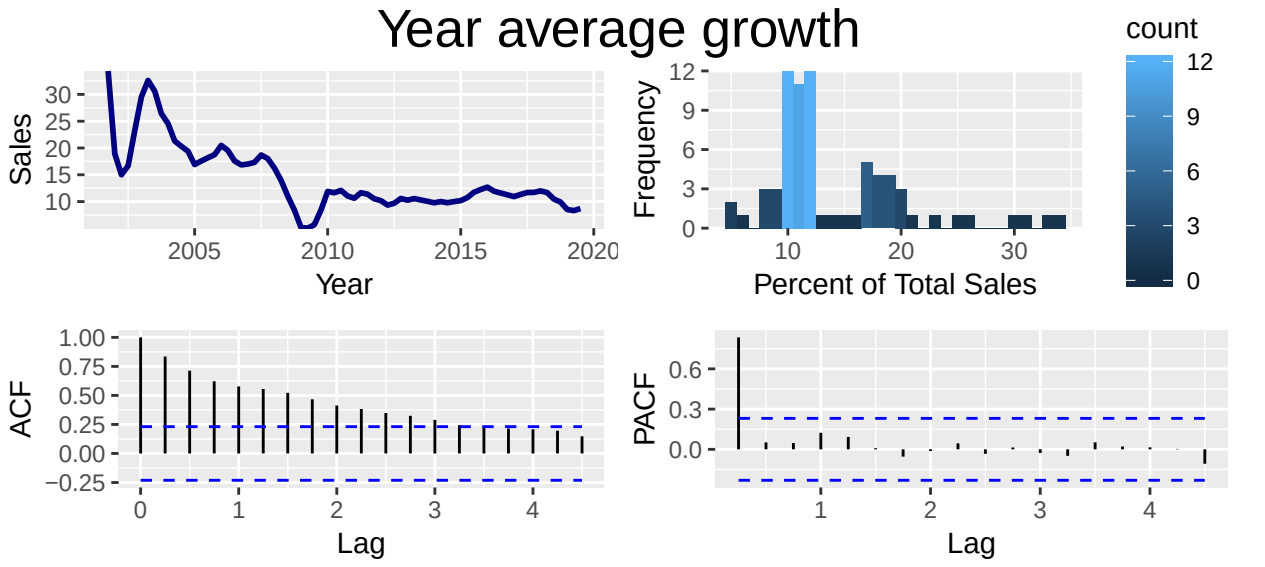
after the first time step. Higher PACF's noted at intervals of 4 quarters, and the seasonal subseries shows that there are higher e-commerce sales in the 4th quarter of each year. This is consistent with the seasonality noted in the raw plot. The gradual tapering off of the ACF is also clear evidence that the raw data is not stationary, which signifies that some non-zero order of differencing will be needed for ARIMA modeling in later sections.

```
# A function to produce a dataframe of a time series with
# time, sales, quarter and floor as the variables
DF <- function(ts) {
    data.frame(time = c(time(ts)), sales = c(ts), quarter = c(time(ts) -
        floor(time(ts))))
}
# A function to produce the year-ended growth rate,
# year-average growth rate, annualized quarter-ended growth
# rate, and annualized decade-ended growth rate
createTs <- function() {
    # Creating a list for the 4 types of growth rate
    results <- list()
    # YEAR ENDED GROWTH as a dataframe and time series
    year_ended_growth <- (df$ECOMPCTNSA[5:79] - df$ECOMPCTNSA[1:75])/df$ECOMPCTNSA[1:75] *
        100
    yeg = ts(data = year_ended_growth, start = c(2000, 4), frequency = 4)
    # YEAR AVERAGED GROWTH as a dataframe and time series
    year_averaged_growth_all = df$ECOMPCTNSA
    for (n in 79:8) {
        year_averaged_growth_all[n] = (mean(year_averaged_growth_all[(n -
            3):n]) - mean(year_averaged_growth_all[(n - 7):(n -
            4)]))/mean(year_averaged_growth_all[(n - 7):(n -
            4)]) * 100
    }
    year_averaged_growth = year_averaged_growth_all[8:79]
    yag = ts(data = year_averaged_growth, start = c(2001, 4),
        frequency = 4)
    # ANNUALIZED QUARTER ENDED GROWTH as a dataframe and time
    # series
    ann_quarter_ended_growth = (((df$ECOMPCTNSA[2:79]/df$ECOMPCTNSA[1:78])^4) -
        1) * 100
    # ANNUALIZED DECADE ENDED GROWTH as a dataframe and time
    # series
    aqeg = ts(data = ann_quarter_ended_growth, start = c(2000,
        1), frequency = 4)
    # Putting in the results into the 'results' list
    results$yeg <- yeg
    results$yag <- yag
    results$aqeg <- aqeg
    return(results)
}
# A function to produce plots for the various growth rates
PlotL <- function(ts, title) {
    # Growth rate
    p1 <- ggplot(ts, aes(x = time(ts), y = ts)) + geom_line(colour = "navy",
        size = 1) + xlab("Year") + ylab("Sales") + scale_y_continuous(expand = c(0,
        0))
    # Histogram
```

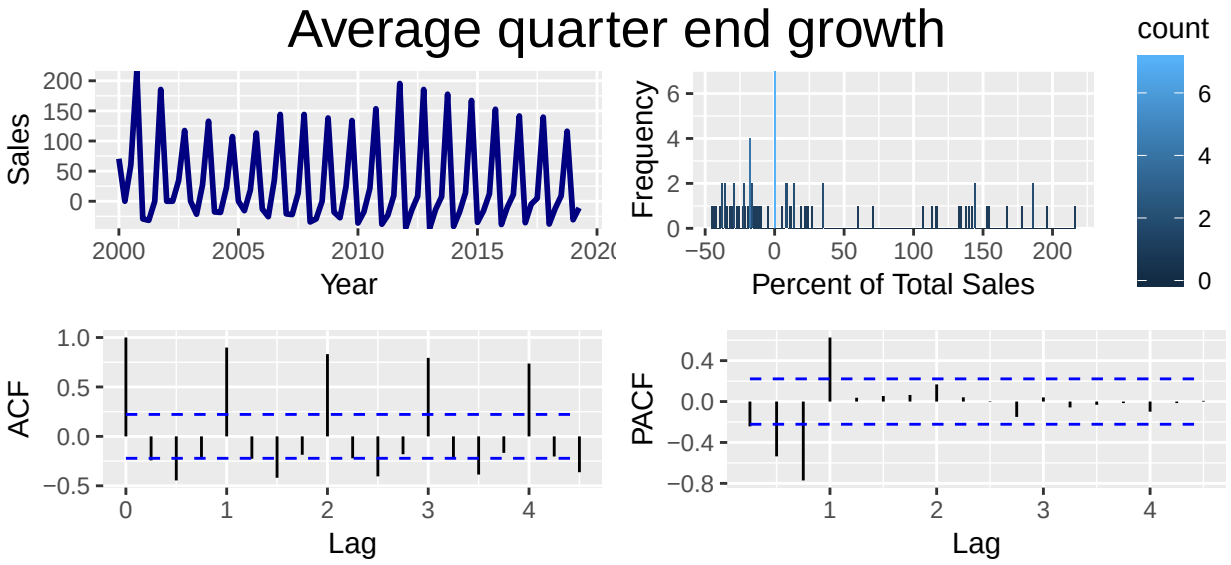
```
    p2 <- ggplot(ts, aes(x = ts)) + geom_histogram(aes(fill = ..count..),
        binwidth = 1) + xlab("Percent of Total Sales") + ylab("Frequency") +
        scale_y_continuous(expand = c(0, 0))
    # ACF
    p3 <- autoplot(acf(ts, plot = FALSE))
    # PACF
    p4 <- autoplot(pacf(ts, plot = FALSE)) + ylab("PACF")
    grid.arrange(p1, p2, p3, p4, ncol = 2, top = textGrob(title,
        gp = gpar(fontsize = 20, font = 1)))
}
PlotL(createTs()$yeg, "Year end growth")
```




```
PlotL(createTs()$yag, "Year average growth")
```




```
PlotL(createTs()$aqeg, "Average quarter end growth")
```



Next, quarterly series for year-ended growth, year-average growth, annualized quarter-ended growth, and annualized decade-ended growth are plotted.

The quarterly series for year-ended growth (the percentage change between one observation and the corresponding observation in the previous year) show a downward trend over time, reaching a minimum around 2009 (likely because of the financial crisis), with growth plateuaing around the 10% per year mark after that. Seasonality is removed by comparing only data from the same quarter. This is also seen in the histogram with the highest frequency being around the 10% mark. The reason for the spike in the first year ended growth reading is because of the high percentage change between Q4 1999 (which is a small number) and Q4 2000. No clear cut pattern is seen in the ACF and PACF plots.

The quarterly series for year-average growth (the average percentage change in growth over the year) shows a similar downward trend with time. As per the previous quarterly series, seasonality is removed by averaging the data for each year. As per the year-ended growth, the minimum in 2009 was followed by a plateau around the 10% per year mark after 2010. The ACF plot gradually tapers off, while the PACF falls off sharply after the first time step.

The quarterly series for the average quarter ended growth (percentage change in levels for successive quarters) exaggerates the seasonality by exponentiating the quarter to quarter growth to the power of 4 (to annualize it). One can see that the annualized percent growth between quarters among different years is remarkably consistent, with relatively lower changes in the 1st through 3rd quarters of each year, a spike in the 4th quarter (about 150-200%) of each year (compared to the 3rd quarter) and then a contraction between the 4th quarter and the 1st quarter of the next year. These changes are also reflected in the ACF and PACF plots, with strong correlations noted with a lag of 4 quarters.

The quarterly series for the average decade ended growth (percentage change in levels for successive decades) is more similar to the graphs for year-ended growth and year-average growth. One can see that the annualized percent growth drops over the years, gradually settling at the 10% mark (similar to what was noted above) as one approaches 2020. These changes are also reflected in the ACF and PACF plots, with strong correlations noted with a lag of 4 quarters. The ACF plot gradually tapers off, while the PACF falls off sharply after the first time step.

Part II

Dataframes containing subsets of the pre-2018 (ts_pre2018) and post-2018 (ts_post2018) observations were created from the original data.

```
# Creating the pre-2018 and post-2018 time series and
# dataframes
ts_pre2018 <- window(dfts, end = c(2017, 4))
ts_post2018 <- window(dfts, start = c(2018, 1))
df_pre2018 <- DF(ts_pre2018)
df_post2018 <- DF(ts_post2018)
```

A for loop and the poly function (which returns orthogonal polynomials of a particular degree) to fit time trend models between maximum degrees, n, with n ranging from 1 to 4 using the `lm()` function. Essentially, we attempted to find the best polynomial time trend model by fitting a linear regression to the data with polynomial terms for time from order 1 to maximum order n included in the linear model equation. For each of the maximum degrees, n, we fitted two types of linear models, totally 4x2=8 models:

1. models taking seasonality into account (mod_season) with the general form:

$$y(time) = c + \sum_{i=1}^{n} time^i + factor(quarter)$$

2. models not taking seasonality into account (mod) with the general form:

$$y(time) = c + \sum_{i=1}^{n} time^i$$

   It is important to note here that instead of using $I(time^n)$ terms within our linear models for higher order polynoimals, we opted to use the $poly(time, n)$ function, for order n. The reason for this because there are the terms $time$, $time^2$, $time^3$ and so on, could be highly correlated with each other. We actually attempted to originally fit using $I(time^n)$ terms but this resulted in model failure, due to correlation between terms, at order n=3. The poly() function allows us to avoid the correlated terms failure by producing orthogonal polynoimals, and therefore allows us to find the maximum order n needed for the best fitting polynomial time trend model.

We first compared the AIC and BIC of all 8 possible models. One can see that the lowest AIC / BIC were found in the 4th degree models that took seasonality into account. However, the differences between the 3rd and 4th degree models were small.

```
# Creating an empty list to hold the linear models and a
# dataframe to hold the AIC / BIC results
linear.models <- list()
IC_df <- data.frame(order = integer(), AIC = double(), BIC = double())
# Creating a for loop to produce the various linear models
for (i in 1:4) {
    # The models not taking seasonality into account
    mod = lm(sales ~ poly(time, i), data = df_pre2018)
    nam <- toString(i)
    # The models taking seasonality into account, using
    # factor(quarter) as a variable
    mod_season = lm(sales ~ poly(time, i) + factor(quarter),
        data = df_pre2018)
    nam_season <- paste(toString(i), "_season", sep = "")
    # Labelling the linear models
    linear.models[[nam]] <- mod
    linear.models[[nam_season]] <- mod_season
    # Binding the dataframe containing the AIC / BIC values
    # together
    IC_df <- rbind(IC_df, data.frame(highest.order = i, mod.AIC = round(AIC(mod),
```

```
        2), seasonal.mod.AIC = round(AIC(mod_season), 2), mod.BIC = round(BIC(mod),
        2), seasonal.mod.BIC = round(BIC(mod_season), 2)))
}
IC_df
```

```
##   highest.order mod.AIC seasonal.mod.AIC mod.BIC seasonal.mod.BIC
## 1             1  142.12           112.90  149.00           126.64
## 2             2  103.78            43.44  112.94            59.47
## 3             3   96.88            19.14  108.33            37.46
## 4             4   95.62            15.81  109.36            36.43
```

Next, we used anova() and CLM diagnostics to determine highest order of importance in polynomial fit for the non-seasonal models. Anova() is performed on the 1st order model. Based on the results above, one can see that the highest order of importance in the polynomial fit for the non-seasonal models was 3.

```
# Doing the anova test for the non seasonally adjusted models
# Looking at first order
Anova(linear.models[["1"]], test = "Chisq")
```

```
## Anova Table (Type II tests)
##
## Response: sales
##               Sum Sq Df F value    Pr(>F)
## poly(time, i) 417.68  1    1075 < 2.2e-16 ***
## Residuals      27.59 71
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Comparing second order with first order
anova(linear.models[["2"]], linear.models[["1"]], test = "Chisq")
```

```
## Analysis of Variance Table
##
## Model 1: sales ~ poly(time, i)
## Model 2: sales ~ poly(time, i)
##   Res.Df    RSS Df Sum of Sq  Pr(>Chi)
## 1     70 15.872
## 2     71 27.586 -1   -11.713 6.604e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Comparing third order with second order
anova(linear.models[["3"]], linear.models[["2"]], test = "Chisq")
```

```
## Analysis of Variance Table
##
## Model 1: sales ~ poly(time, i)
## Model 2: sales ~ poly(time, i)
##   Res.Df    RSS Df Sum of Sq Pr(>Chi)
## 1     69 14.051
## 2     70 15.872 -1   -1.8212 0.002785 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Comparing fourth order with third order
anova(linear.models[["4"]], linear.models[["3"]], test = "Chisq")
```

```
## Analysis of Variance Table
##
## Model 1: sales ~ poly(time, i)
## Model 2: sales ~ poly(time, i)
##   Res.Df    RSS Df Sum of Sq Pr(>Chi)
## 1     68 13.437
## 2     69 14.051 -1  -0.61419   0.0779 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The same is done for the seasonal models. In the results above, the highest order of importance in the polynomial fit for the seasonal models was also 3, although having a 4rth order term produces a somewhat significant χ^2 statistic. However, given that the p-value is quite close t 0.05 threshold, and there is not much difference in AIC/BIC between order 3 and order 4, it is likely that an order 4 term does not lead to a better fitting model.

```
# Doing the anova test for the seasonally adjusted models
# Looking at first order
Anova(linear.models[["1_season"]], test = "Chisq")
```

```
## Anova Table (Type II tests)
##
## Response: sales
##                 Sum Sq Df  F value    Pr(>F)
## poly(time, i)   418.38  1 1670.867 < 2.2e-16 ***
## factor(quarter)  10.56  3   14.056 3.157e-07 ***
## Residuals        17.03 68
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Comparing second order with first order
anova(linear.models[["2_season"]], linear.models[["1_season"]],
    test = "Chisq")
```

```
## Analysis of Variance Table
##
## Model 1: sales ~ poly(time, i) + factor(quarter)
## Model 2: sales ~ poly(time, i) + factor(quarter)
##   Res.Df     RSS Df Sum of Sq  Pr(>Chi)
## 1     67  6.3973
## 2     68 17.0268 -1    -10.63 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Comparing third order with second order
anova(linear.models[["3_season"]], linear.models[["2_season"]],
    test = "Chisq")
```

```
## Analysis of Variance Table
##
## Model 1: sales ~ poly(time, i) + factor(quarter)
## Model 2: sales ~ poly(time, i) + factor(quarter)
##   Res.Df    RSS Df Sum of Sq  Pr(>Chi)
## 1     66 4.4620
## 2     67 6.3973 -1   -1.9353 8.777e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

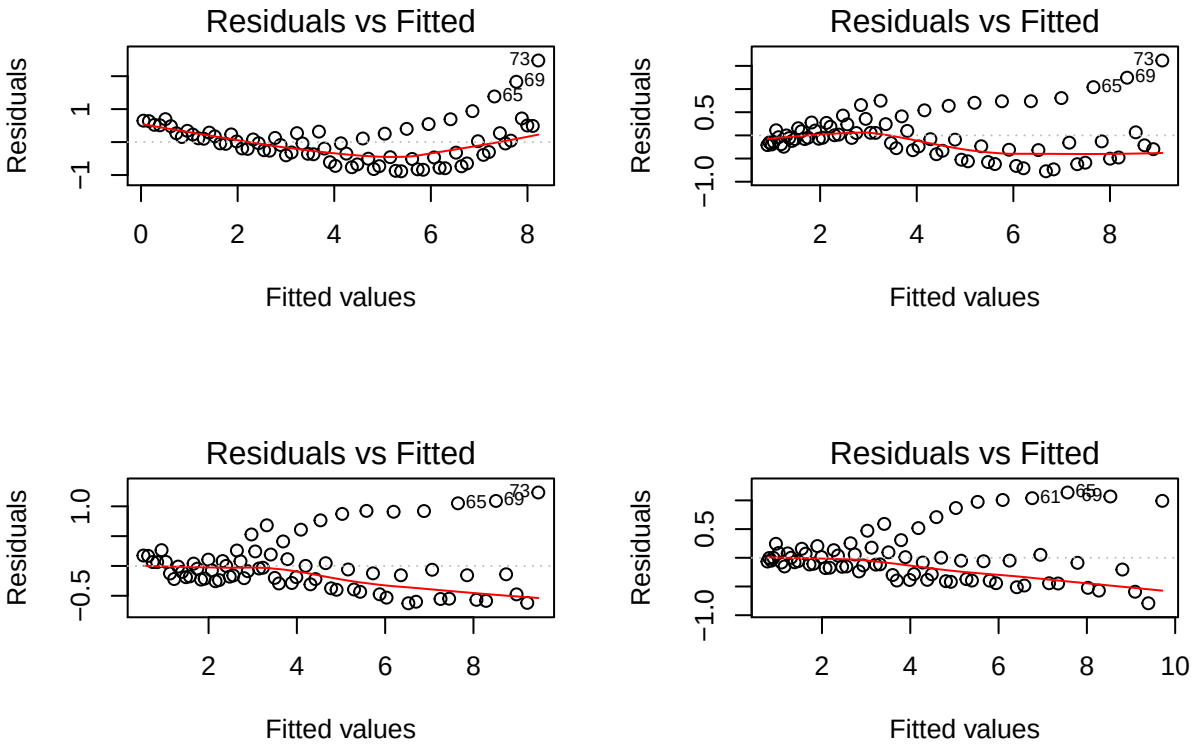
```r
# Comparing fourth order with third order
anova(linear.models[["4_season"]], linear.models[["3_season"]],
    test = "Chisq")
```

```
## Analysis of Variance Table
##
## Model 1: sales ~ poly(time, i) + factor(quarter)
## Model 2: sales ~ poly(time, i) + factor(quarter)
##   Res.Df    RSS Df Sum of Sq Pr(>Chi)
## 1     65 4.1479
## 2     66 4.4620 -1  -0.31405  0.02653 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

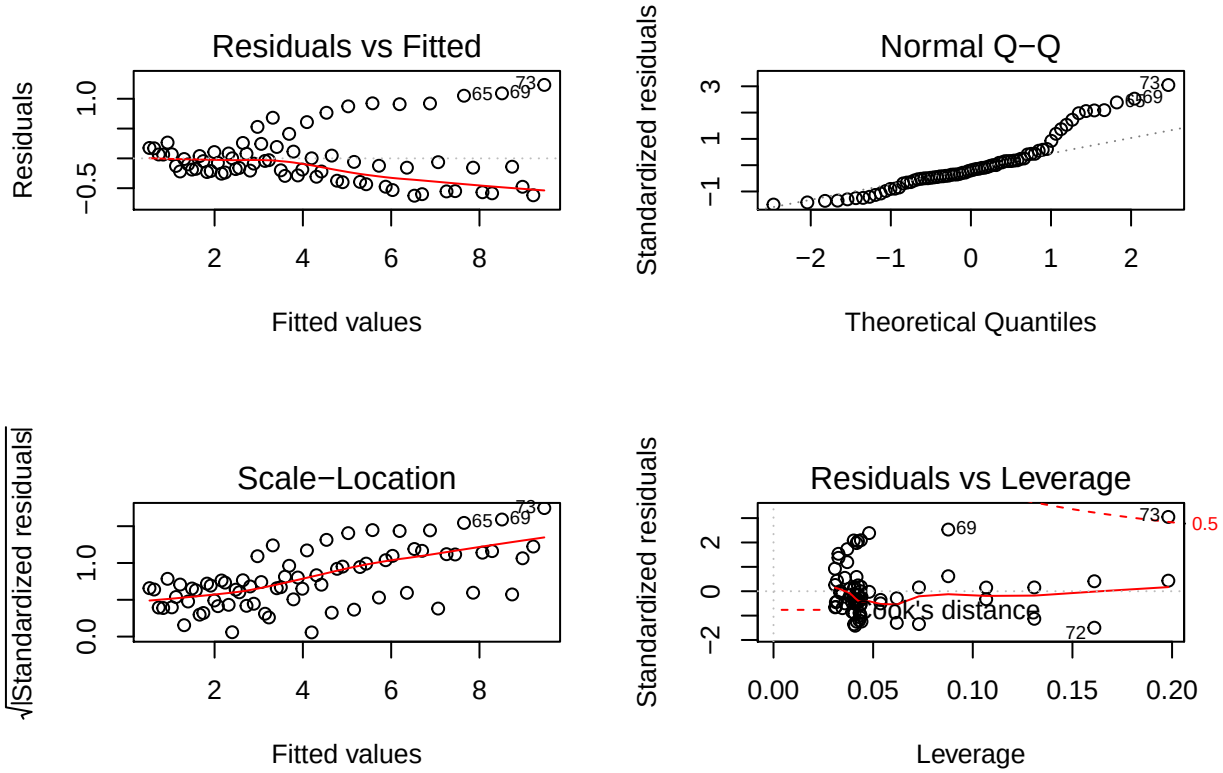
To compare the 4 different models of each type, we also plotted CLM diagnostic plots for comparison for all 8 models. We share some discussion on the comparisons between the 4 different models below, but we only display the Residuals vs Fitted plots for all 8 models and the 4 diagnostic plots for order 3 models below (to conserve some space).

First, we show some selected diagnostic plots from non-seasonal models:

```r
# Plotting the diagnostic plots for each of the models not
# taking seasonality into account Top left- 1st order, Top
# right- 2nd order, Bottom left- 3rd order, Bottom right- 4th
# order
par(mfrow = c(2, 2))
plot(linear.models[["1"]], which = 1)
plot(linear.models[["2"]], which = 1)
plot(linear.models[["3"]], which = 1)
plot(linear.models[["4"]], which = 1)
```



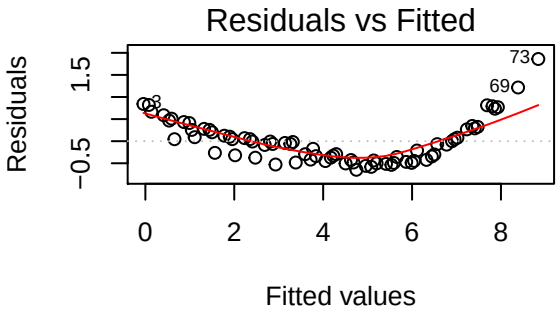
```
par(mfrow = c(2, 2))
plot(linear.models[["3"]])
```

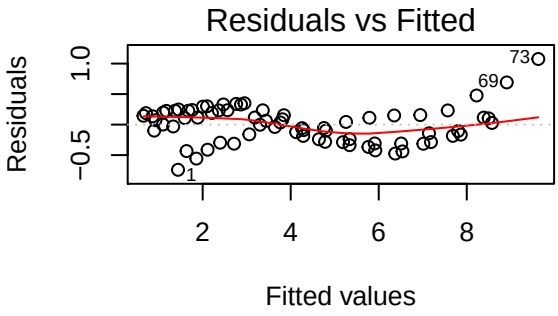


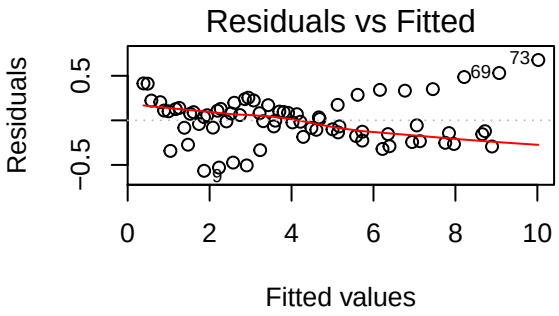
Comparing the diagnostic plots for the non-seasonal models, one sees a downward trend in the residuals vs fitted plots for the 2nd order and above non-seasonal models, which implies some degree of endogeneity. The normal Q-Q plots show devaiation from normality at the higher quantiles. The scale location plot shows greater heteroskedasticity at the higher fitted values, particularly for the 3rd and 4th order models. The residuals vs leverage plot do not show any datapoints with a Cook's distance above 1.

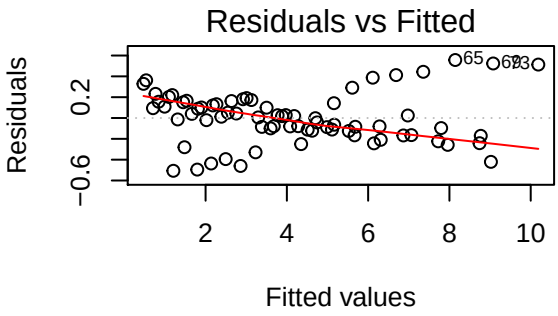
Next, we should some diagnostic plots from the 4 models that include the seasonal factor term:

```
# Plotting the diagnostic plots for each of the models taking
# seasonality into account Top left- 1st order, Top right-
# 2nd order, Bottom left- 3rd order, Bottom right- 4th order
par(mfrow = c(2, 2))
plot(linear.models[["1_season"]], which = 1)
plot(linear.models[["2_season"]], which = 1)
plot(linear.models[["3_season"]], which = 1)
plot(linear.models[["4_season"]], which = 1)
```

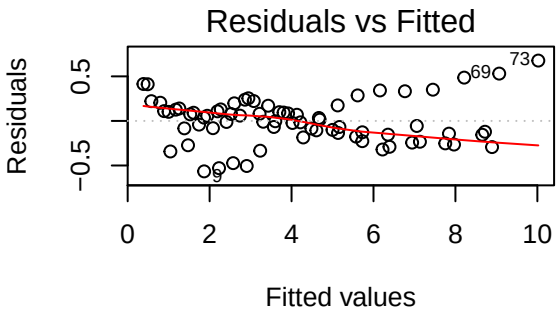


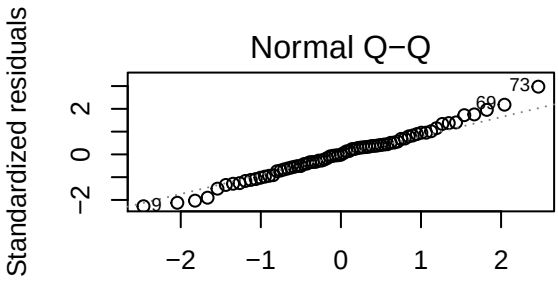


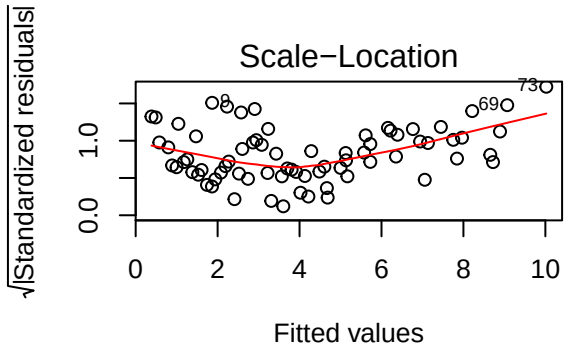


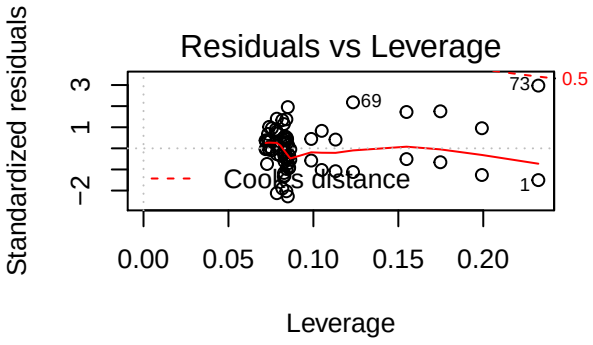



```
par(mfrow = c(2, 2))
plot(linear.models[["3_season"]])
```









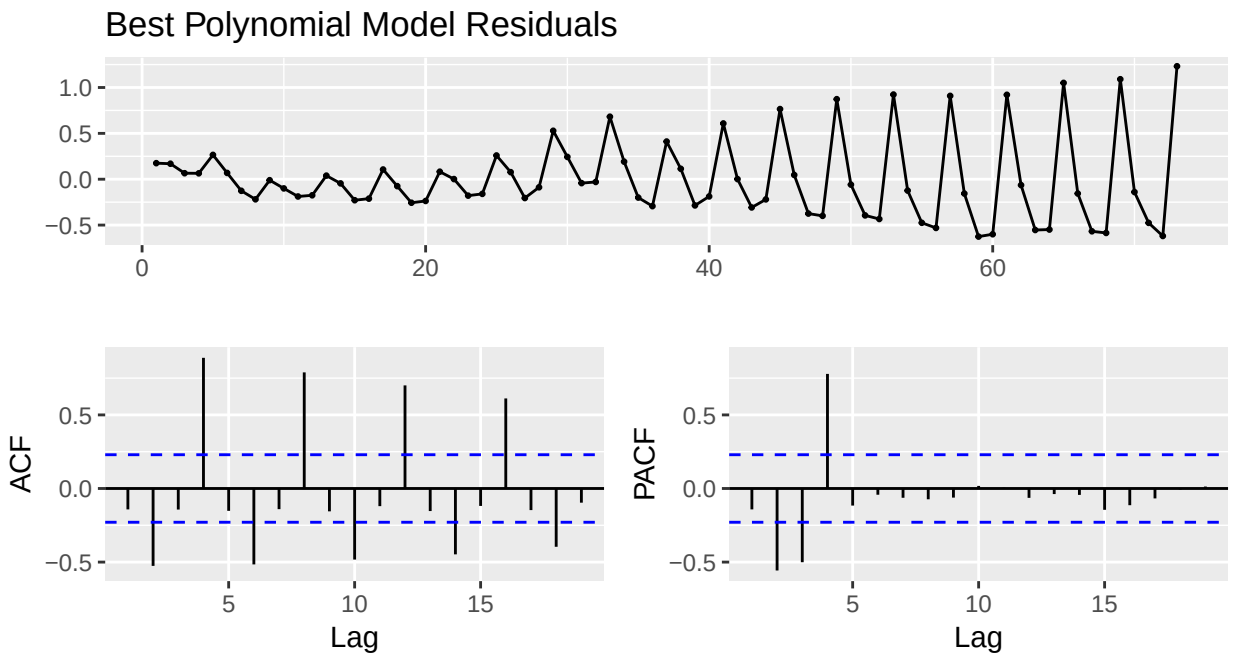
Comparing the diagnostic plots for the seasonal models, one sees a relatively flat residuals vs fitted plots for the seasonal models though the 4th order plot does show some decrease at higher fitted values. The normal Q-Q plots reveal that the residuals for the seasonal plots (particularly for the 3rd and 4th order

models) better conform to normality than the non-seasonal ones. The scale location plots appear to show lower heteroskedasticity for the seasonal models compared to the non seasonal models. Again, the residuals vs leverage plots do not show any datapoints with a Cook's distance above 1.

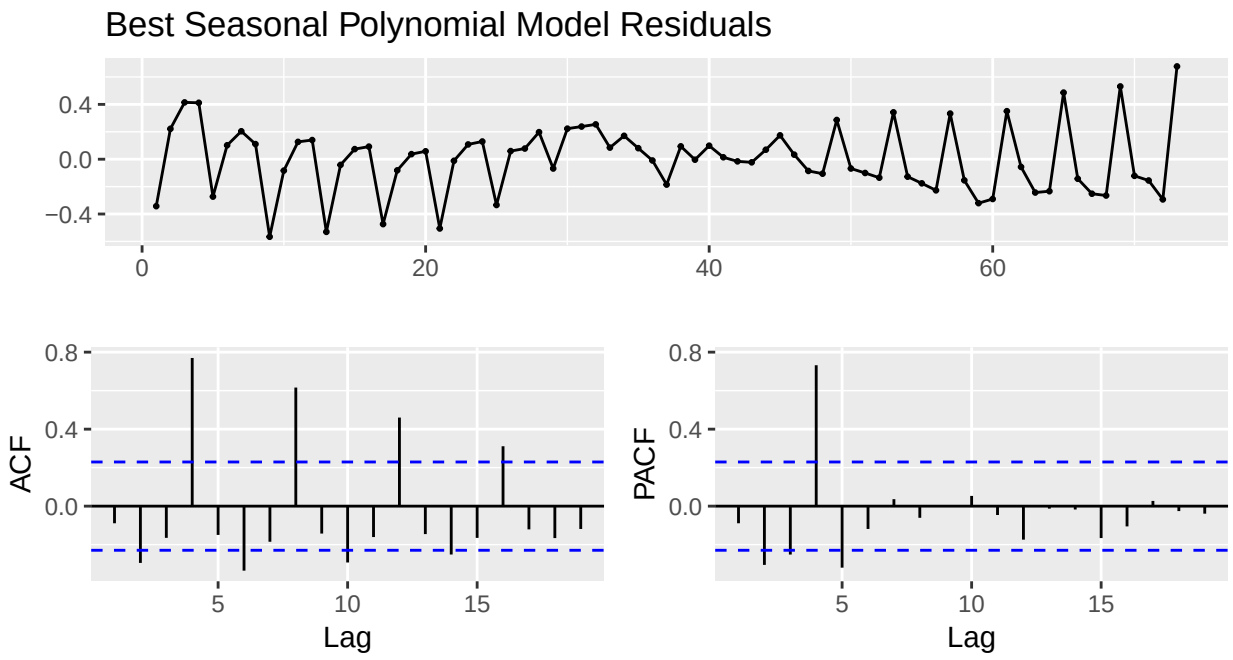
Hence, based on the results of the AIC, BIC, Anova tests, and diagnostic plots, we propose that the 3rd order models be used as our best models.

We next plot the ACF and PACF of the 3rd degree models. One notes that for both the non seasonal and seasonal models, there is a lag at 4, as would be expected if there were significant seasonality.

```
# Plotting the residuals
linear.models[["3"]]$residuals %>% ggtsdisplay(main = "Best Polynomial Model Residuals")
```



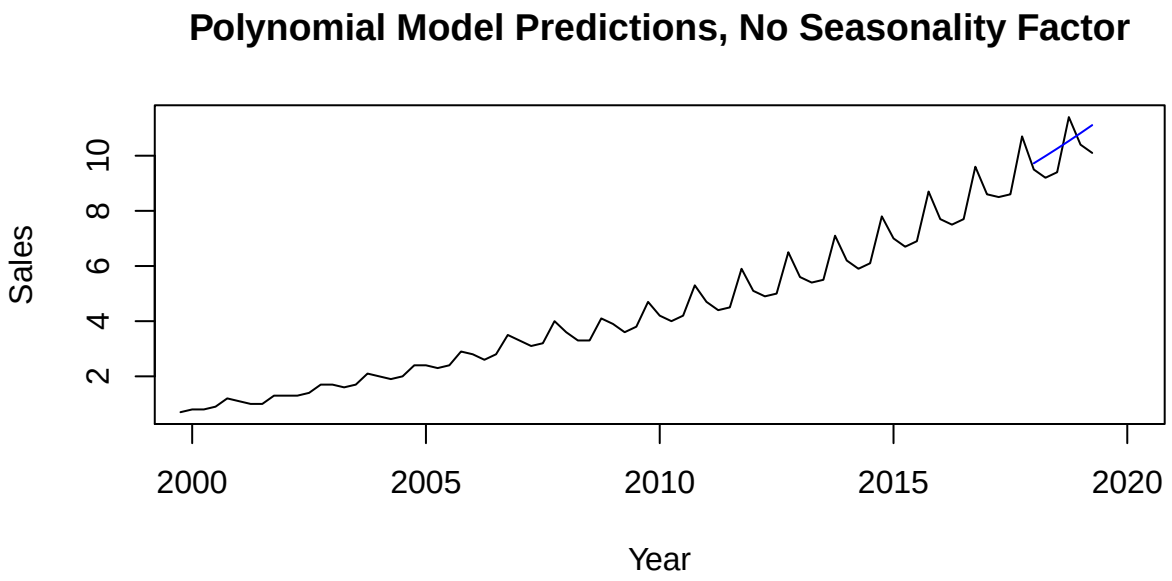

```
linear.models[["3_season"]]$residuals %>% ggtsdisplay(main = "Best Seasonal Polynomial Model Residuals")
```



This is because of how the linear model takes the seasonality into account is by adding in the mean seasonal effect over the time period. However, the true seasonal effect probably increases over time (as it is multiplicative). Hence, although adjusting for seasonality improves the overall fit of the model, the earlier time points for the seasonal model are overcorrected for, whereas the later time points are undercorrected for. This leaves an oscillation in the residuals which is picked up by the ACF and PACF.

Next, we use our (best) 3rd order model to predict the post 2018 values (not taking seasonality into account. We plot the predictions on a graph with the original data.

```
# Creating a dataframe with the time variable for post 2018
predict.data = data.frame(time = df_post2018[, 1])
# Fitting the 3rd degree time trend model to the pre-2018
# data WITHOUT taking seasonality into account
lm.3 = lm(formula = sales ~ poly(time, 3), data = df_pre2018)
# Predicting the values for post 2018 using our model (non
# seasonal)
lm.pred = predict(object = lm.3, newdata = predict.data, se = TRUE)
# Plotting the predicted values (in blue) with the pre-2018
# data (in black)
ts.plot(dfts, lty = 1, xlim = c(2000, 2020), xlab = "Year", ylab = "Sales",
    main = "Polynomial Model Predictions, No Seasonality Factor")
lines(ts(lm.pred$fit, frequency = 4, start = c(2018, 1)), lty = 1,
    col = "blue")
```




```
# Finding the RMSE
sqrt(mean((lm.pred$fit - df_post2018$sales)^2))
```

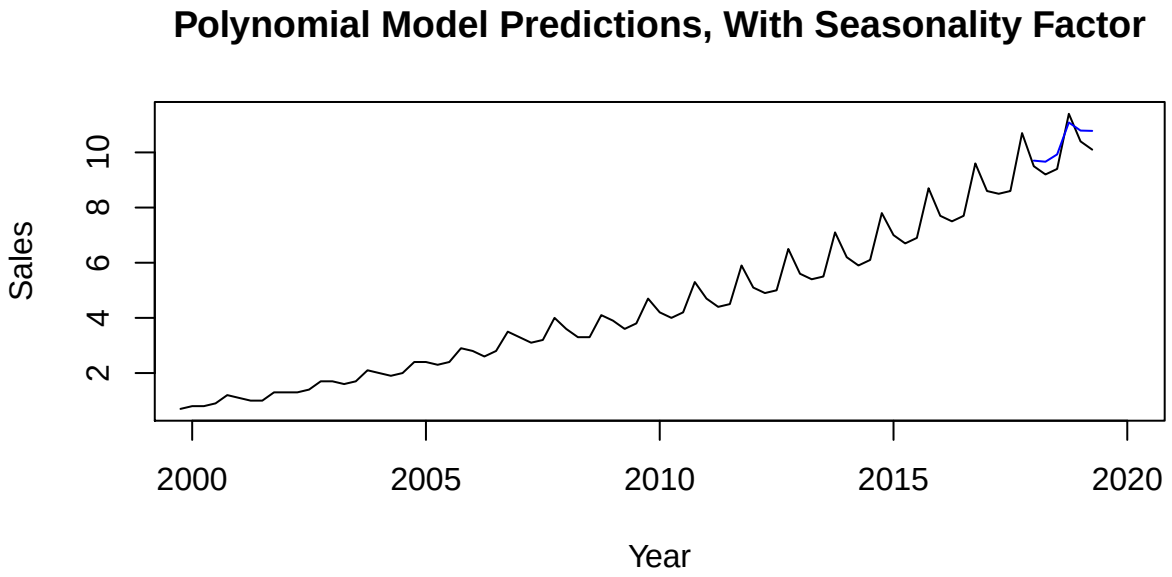
```
## [1] 0.7467474
```

One can see that this forecast predicts the trend, without the seasonality component, and the model is essentially unable to predict the actual data well. The RMSE for predictions on the post 2018 data is 0.7467.

Next, we use our (best) 3rd order model to predict the post 2018 values (this time, taking seasonality into account. We plot the predictions on a graph with the original data.

```
# Creating a dataframe with the time variable for post 2018
predict.data.season = data.frame(time = df_post2018[, 1], quarter = df_post2018[,
    3])
# Fitting the 3rd degree time trend model to the pre-2018
```

```r
# data
season.lm.3 = lm(formula = sales ~ poly(time, 3) + factor(quarter),
    data = df_pre2018)
# Predicting the values for post 2018 using our model
# (seasonal)
lm.pred.seasonal = predict(object = season.lm.3, newdata = predict.data.season,
    se = TRUE)
# Plotting the predicted values (in blue) with the pre-2018
# data (in black)
ts.plot(dfts, lty = 1, xlim = c(2000, 2020), xlab = "Year", ylab = "Sales",
    main = "Polynomial Model Predictions, With Seasonality Factor")
lines(ts(lm.pred.seasonal$fit, frequency = 4, start = c(2018,
    1)), lty = 1, col = "blue")
```




```r
# Finding the RMSE
sqrt(mean((lm.pred.seasonal$fit - df_post2018$sales)^2))
```

```
## [1] 0.4560327
```

One can see that this forecast predicts both the trend and the seasonality The RMSE is therefore lower than the predictions which did not take seasonality into account (0.4560). However, we look at the forecast (blue) compared to the post 2018 data, we see that the fit is quite poor, we can thus conclude that a polynomial time trend is likely insufficient in its complexity to model the FRED data.

Part III

In the code below, we take the pre-2018 series subset and we perform the following transformations:

1. We apply the `decompose()` function, using parameter type = 'multiplicative', which allows us to decompose our time series into trend, seasonal, and random noise components.
2. We next create a Seasonally-Adjusted pre-2018 time series by dividing the pre-2018 series by the seasonal component from the decomposition. In other words, we applied the following transformation:

$$ts\_2018.SA = \frac{ts\_2018}{seasonality}$$

where seasonality is the seasonal component of the decomposition performed in step 1. We note here that this method of creating a seasonally adjusted time series is equaivalent to applying the function `seasadj()` to the output of `decompose()`. However, knowing that we can simply factor in seasonality through division allows us to manipulate our forecast results later on.

3. We then repeat steps 1 and 2 to the entire time series, including the post 2018 data points. This allows us to compare the seasonally adjusted pre-2018 time series to the entire seasonally adjusted data set.

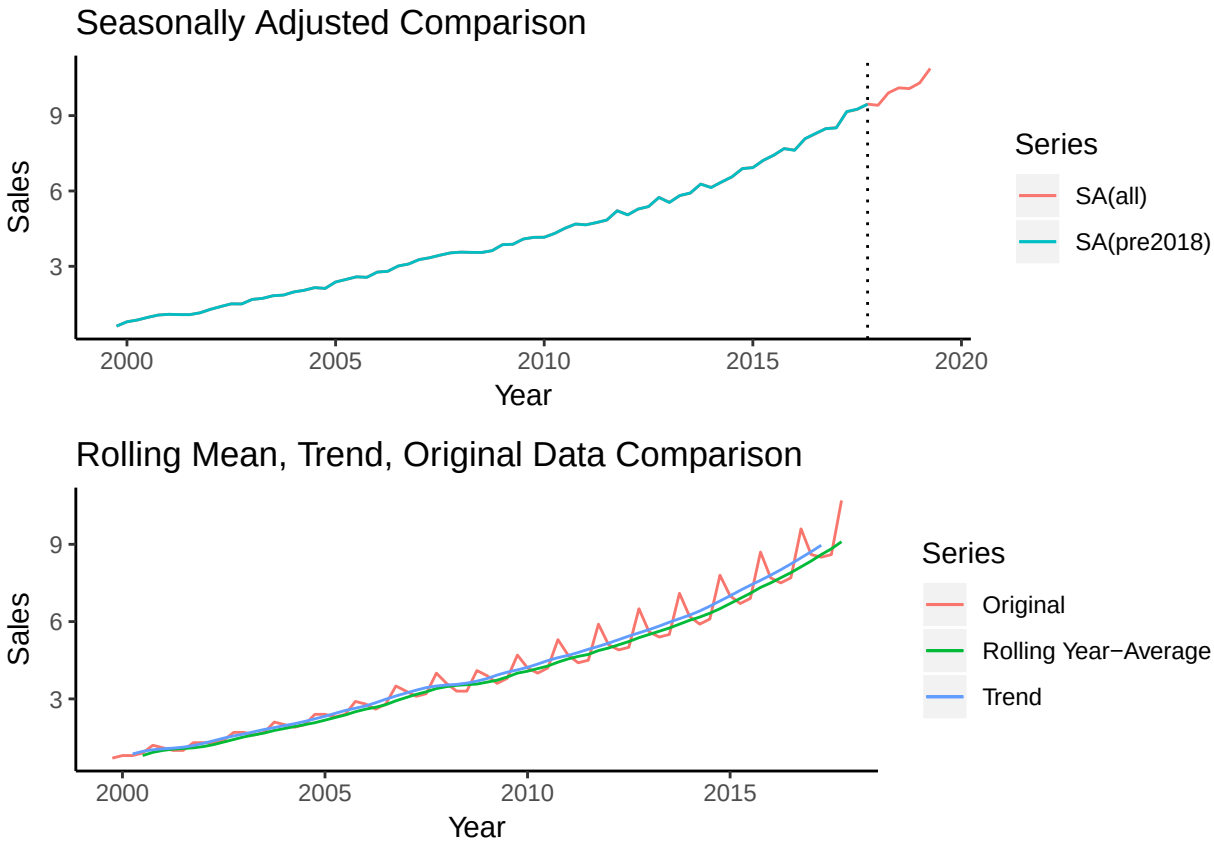
4. Additionally, we use the function `rollmean()` with k=4 parameter to create a quarterly rolling year-average series. The values for the rolling-year average comes from taking the average of a 4-quarter window, that includes the 3 previous quarters and the current quarter:

$$y_{rm(t)} = \frac{y_{t-3} + y_{t-2} + y_{t-1} + y_t}{4}$$

In the plots below, we first show the sesonally adjusted pre-2018 time series compared to the plot of the entire seasonally adjusted time series. In a second plot below, we show the rolling average, the trend, and the original series in comparison with each other.

```
p3 <- function(ts, ts0, ts1) {
    NSA <- DF(ts)
    # Decompose the trend and seasonal components of the pre-2018
    # series and use the seasonal factors to produce a
    # seasonally-adjusted series.

    # seasonal component of pre-2018 series
    SA.seasonal = decompose(ts, type = "mult")$seasonal
    # seasonally adjusted series, pre-2018
    SA <- DF(ts/SA.seasonal)
    # seasonal component of the entire series
    SA0.seasonal = decompose(ts0, type = "mult")$seasonal
    # seasonally adjusted entire series
    SA0 <- DF(ts0/SA0.seasonal)
    # With the pre-2018 data, produce a quarterly rolling
    # year-average series.
    RM <- ts(rollmean(c(ts), 4), frequency = 4, start = c(2000,
        3)) %>% DF()
    # Trend from decomposition
    TD <- decompose(ts, type = "mult")$trend %>% DF()
    # plot of seasonally-adjusted comparison
    p1 <- ggplot(data = NSA, aes(x = time, y = sales, col = Series)) +
        xlab("Year") + ylab("Sales") + geom_line(data = SA0,
        aes(y = sales, col = "SA(all)")) + geom_line(data = SA,
        aes(y = sales, col = "SA(pre2018)")) + geom_vline(xintercept = 2017.75,
        linetype = "dotted") + ggtitle("Seasonally Adjusted Comparison") +
        scale_y_continuous(labels = function(x) format(x, scientific = FALSE)) +
        theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
            panel.background = element_blank(), axis.line = element_line(colour = "black"))
    # plot of trend, rolling year-average, original data
    p2 <- ggplot(data = NSA, aes(x = time, y = sales, col = Series)) +
        xlab("Year") + ylab("Sales") + geom_line(data = NSA,
        aes(y = sales, col = "Original")) + geom_line(data = RM,
        aes(y = sales, col = "Rolling Year-Average")) + geom_line(data = TD,
        aes(y = sales, col = "Trend")) + ggtitle("Rolling Mean, Trend, Original Data Comparison") +
        scale_y_continuous(labels = function(x) format(x, scientific = FALSE)) +
        theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
```

```
            panel.background = element_blank(), axis.line = element_line(colour = "black"))
    grid.arrange(p1, p2, ncol = 1, nrow = 2)
    # return data frames to be used later:
    return(list(NSA = NSA, SA = SA, SA.seasonal = SA.seasonal,
        SA0.seasonal = SA0.seasonal, SA0 = SA0, RM = RM, TD = TD))
}
P3 <- p3(ts_pre2018, dfts, ts_post2018)
```



When we compare the two plots together, we see that the Rolling Year-Average and the Trend are quite similar, with the Rolling-Year Average having slightly lower values for later years where the oscillations due to seasonality have become higher. Both the Rolling Year-Average and the Trend are much smoother than either of the Seasonally-Adjusted series. When we decomposed the full data set, we essentially separated the full data into 3 multiplicative factors:

$$Full = Trend * Seasonality * Random$$

Therefore, we would expect the Trend to be much smoother than the Seasonally Adjusted series, as the Trend does not take into consideration the random noise that is extracted in the decomposition process. On the other hand, the Seasonally Adjusted series is given by:

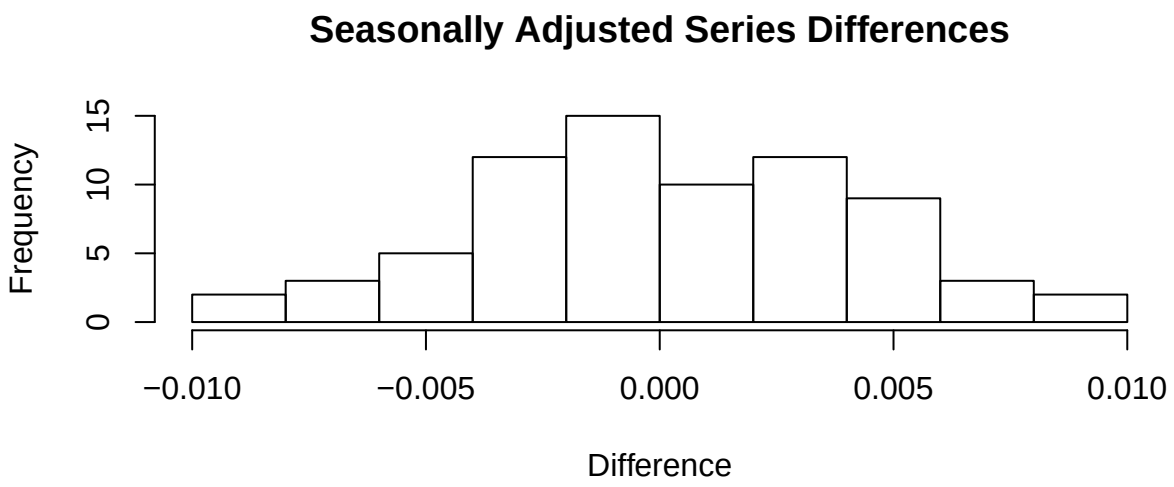
$$Seasonally\ Adjusted = \frac{Full}{Seasonality} = Trend * Random$$

and appears to have more noticeable oscillations, especially in the later years, because it **does** incorporate random noise from the decomposition process.

In addition, when we compare the Seasonally-Adjusted entire series to the Seasonally-Adjusted pre-2018 series in the top plot above, we see that the portion of the plot from both series pre-2018 lay almost entirely above each other. This signifies that the Seasonally-Adjusted series pre-2018 does not change significantly

when the 2018-2019 data points are added back. Or in other words, the seasonality of the pre-2018 data is very similar to the seasonality of the data from 2018-2019, such that the data points from 2018-2019 do not affect the seasonally adjusted pre-2018 values. To prove that the Seasonally-Adjusted pre-2018 data is very similar in the Seasonally-Adjusted pre-2018 series and the entire seasonally adjusted series, we plot the differences from the two Seasonally-Adjusted series below:

```
# differences between Seasonally Adjusted Data sets
hist(P3$SA$sales - P3$SA0[1:73, ]$sales, xlab = "Difference",
    ylab = "Frequency", main = "Seasonally Adjusted Series Differences")
```



From this historgram, we see that the majority of the differences between seasonally adjusted values from the pre-2018 series and from the entire series are very close to 0, which maximum and minimum differences close to +/- 0.1. This again supports the claim that the data points for the subset from 2018-2019 do not substantially affect the seasonality of the series pre 2018.

Part IV

In this section, we attempt to fit several versions of AutoRegressive Integrated Moving Average (ARIMA) Models to both the seasonally adjusted and non-seasonally adjusted pre-2018 time series data. The 6 data points extracted from Q2 of 2018 to Q4 of 2019 are used for "pseudo-out-of-sample" model performance evaluation. In each of our model optimization attempts, we use the `diff()` function to determine the appropriate d and/or D parameter. We then attempt to find a starting point for parameters p, q, P, and Q through diagnostic plots of ACF and PACF.

A. Seasonally Adjusted Model Fitting using Non-Seasonal ARIMA

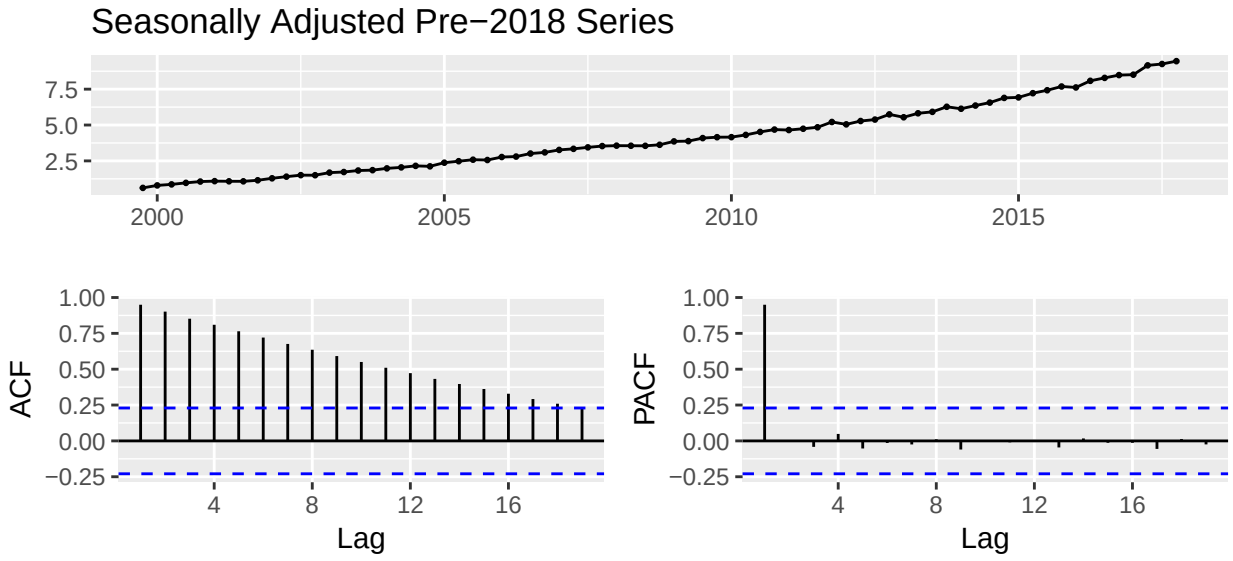
Our first instinct in finding a suitable ARIMA model for the Seasonally-Adjusted (SA) data is to use a ARIMA model with no seasonal components, since the seasonality has essentially been removed. We perform a series of steps to find the optimal ARIMA model with equation:

$$(1 - \phi_1 B - ... - \phi_p B^p)(1 - B)^d y_t = c + (1 + \theta_1 B + ... + \theta_q B^q)\epsilon_t$$

where p represents the order of the AutoRegressive model component (`AR(p)`), q represents the order of the Moving Average model component (`MA(q)`), d represents the necessary order of differencing, ϵ_t is the white noise term, and `B` represents the backshift operator with function $By_t = y_{t-1}$.

We start with examining the pre-2018 SA data and its ACF and PACF plots.

```
ts_pre2018.SA = ts(P3$SA$sales, start = c(1999, 4), frequency = 4)

ts_pre2018.SA %>% ggtsdisplay(main = "Seasonally Adjusted Pre-2018 Series")
```




```
ts_pre2018.SA %>% ur.kpss() %>% summary()
```

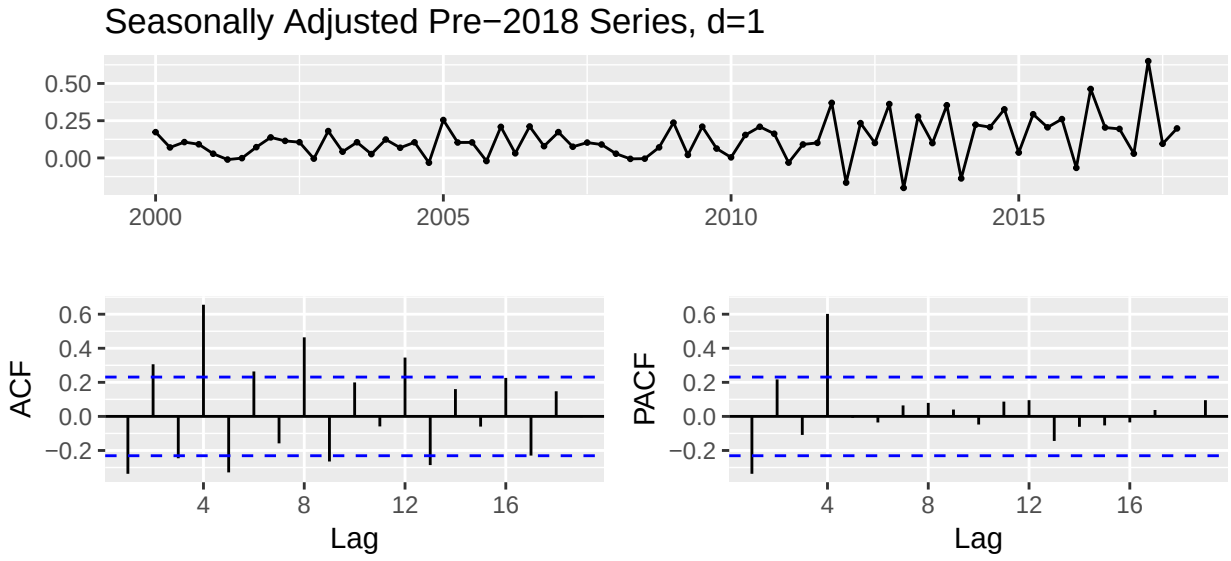
```
##
## ######################
## # KPSS Unit Root Test #
## ######################
##
## Test is of type: mu with 3 lags.
##
## Value of test-statistic is: 1.8722
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

We can tell immediately from the ACF that the data is not stationary, as the ACF would drop to 0 quite quickly for a stationary series but decreases slowly for a non-stationary series, which is shown in the correlogram plot above. To further confirm that the series is indeed not stationary, we conduct the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test, which has a null hypothesis that the data is stationary. From the KPSS test, we receive a high test statistic of 1.8722, which is enough to reject the null hypotesis that the data is stationary.

In order to properly find the orders p and q, we must first apply differencing to the data to obtain a stationary series. The number of times differencing is required will allow us to obtain the value of d.

To assess whether or not d = 1 is correct, we apply a first-order differencing to the data via the `diff()` function, and we examine the ACF, PACF and the KPSS test results.

```
ts_pre2018.SA %>% diff() %>% ggtsdisplay(main = "Seasonally Adjusted Pre-2018 Series, d=1")
```

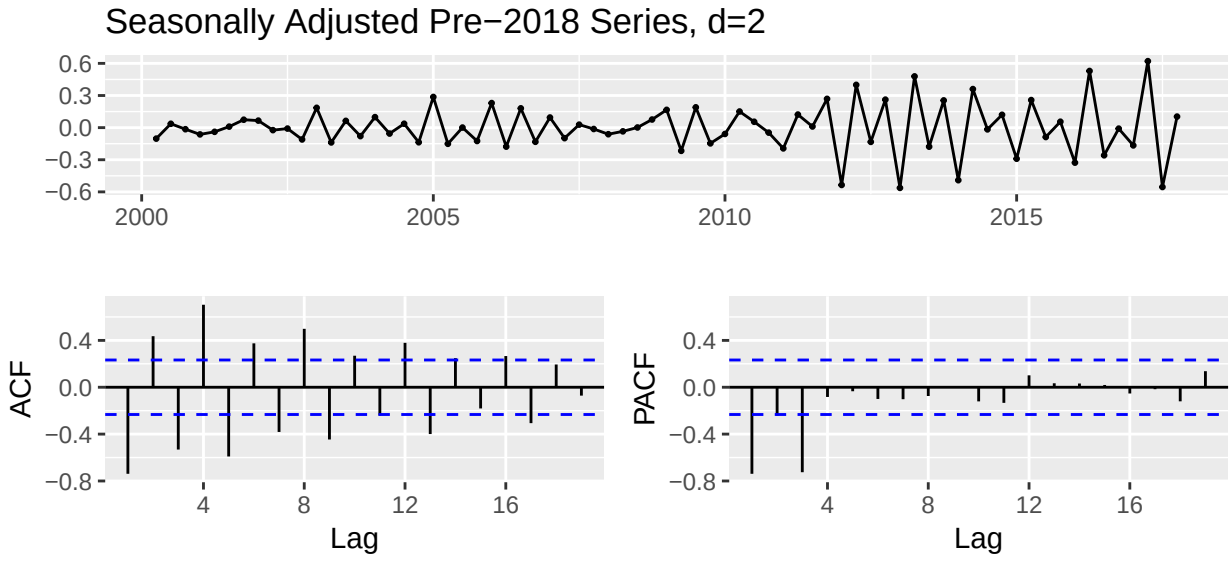



```
ts_pre2018.SA %>% diff() %>% ur.kpss() %>% summary()
```

```
##
## ######################
## # KPSS Unit Root Test #
## ######################
##
## Test is of type: mu with 3 lags.
##
## Value of test-statistic is: 1.1526
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

It is a little difficult to discern from the plots themselves whether or not the once differenced series is stationary, as the plot of the series itself shows only a slight increasing mean with time. However, the KPSS test produces a test statistic of 1.1526, which is large enough to reject the null hypothesis of stationary data at the 1 percent level. Given that a first order differencing is insufficient to produce a stationary series, d=1 is not the correct parameter value for an ARIMA model. Next, we see the effect of d=2:

```
ts_pre2018.SA %>% diff() %>% diff() %>% ggtsdisplay(main = "Seasonally Adjusted Pre-2018 Series, d=2")
```




```
ts_pre2018.SA %>% diff() %>% diff() %>% ur.kpss() %>% summary()
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: mu with 3 lags.
##
## Value of test-statistic is: 0.1003
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

Looking at the plot of the series post second-order differencing, we do not see a series that has an average that appears to drift up or down with time. Furthermore, we see that the KPSS test produces a much smaller test statistic (0.1003), which is insufficient to reject the null hypothesis of stationary data. Therefore, we surmise that the optimal parameter for order d is 2, and we check this using `ndiffs()` function, which takes the KPSS test into consideration:

```
ndiffs(ts_pre2018.SA)
```

```
## [1] 2
```

From the plots twice-differenced data, the ACF is oscilating and the PACF has a significant spike at lag 3 but no other significant spikes beyond lag 3. This is a strongly suggestive of a AR(3) component, and this allows us to start with Arima(3,2,0) model as our first guess. However, being aware that ACF and PACF plots are not very useful for parameter predictions of ARIMA models where $p \neq 0$ and $q \neq 0$, we use a for loop below to test other paramters for p and q. Since we have a pretty good guess for p, we allow p to vary from 2 to 4, which is +/-1 our best guess for p from the diagnostic plots. Since we do not have a good guess for q, we allow q to vary from 0 to 2.

For the p,q values tried in each of the versions of ARIMA modeling in the for-loop below, we also calculate the AIC, AICc and the BIC as a measure of the quality of fit of each model. Our goal is to find the model that minimizes these criteria values in an attempt to select the optimized orders of p and q. Furthermore, we priorize the minimizing the BIC value, which penalizes more heavily for higher complexity, as we believe that a simpler model with comparable performance is more preferred, if it has comparable performance metrics to

a more complex model.

Additionally, we test in-sample and pseudo-out-of-sample model performance using RMSE calculations. The RMSE calculations have two versions -

1. We calculate the RMSE of the fitted/predicted values compared to the actual values in the SA data. This is denoted by sa.in.rmse and sa.out.rmse.
2. We also calculate the RMSE of the fitted/predicted values compared to the actual values *with the seasonality added back in through multiplication*. Since the SA data is constructed by dividing by the seasonality, we find this RMSE by multiply the fitted/predicted values by their respective seasonality. The reason for doing this is so that we have in sample and out of sample RMSEs that are more comparable to the models fitted to the NSA data, which takes seasonality into consideration. In the table below, these additioanl RMSE values are simply called in.rmse and out.rmse.

```
# initializing results table
mod_df.SA <- data.frame(p = integer(), d = integer(), q = integer(),
    AIC = double(), AICc = double(), BIC = double(), sa.in.rmse = double(),
    sa.out.rmse = double(), in.rmse = double(), out.rmse = double())
for (p in 2:4) {
    for (q in 0:2) {
        # for each p, q, fit ARIMA model
        mod <- Arima(ts_pre2018.SA, order = c(p, 2, q), method = "ML")
        # forecast model for next 6 quarters, to match 2018-2019 data
        f = forecast(mod, h = 6)
        # calculate RMSE on seasonally adjusted values
        sa.insample.rmse = (mean((mod$fitted - P3$SA$sales)^2))^0.5
        sa.pred.rmse = mean((f$mean - P3$SA0$sales[74:79])^2)^0.5
        # calculate RMSE on actual values by multiplying back in
        # seasonality factor
        insample.rmse = (mean((mod$fitted * P3$SA.seasonal -
            c(ts_pre2018))^2))^0.5
        pred.rmse = mean((f$mean * P3$SA0.seasonal[74:79] - c(ts_post2018))^2)^0.5
        read <- data.frame(p = as.integer(p), d = 2, q = as.integer(q),
            in.rmse = insample.rmse, out.rmse = pred.rmse, sa.in.rmse = sa.insample.rmse,
            sa.out.rmse = sa.pred.rmse, BIC = mod$bic, AICc = mod$aicc,
            AIC = mod$aic)
        mod_df.SA <- rbind(mod_df.SA, read)
    }
}
mod_df.SA <- mod_df.SA %>% dplyr::arrange(BIC, in.rmse, AICc,
    AIC, p, q)
print(head(mod_df.SA, 10))
```

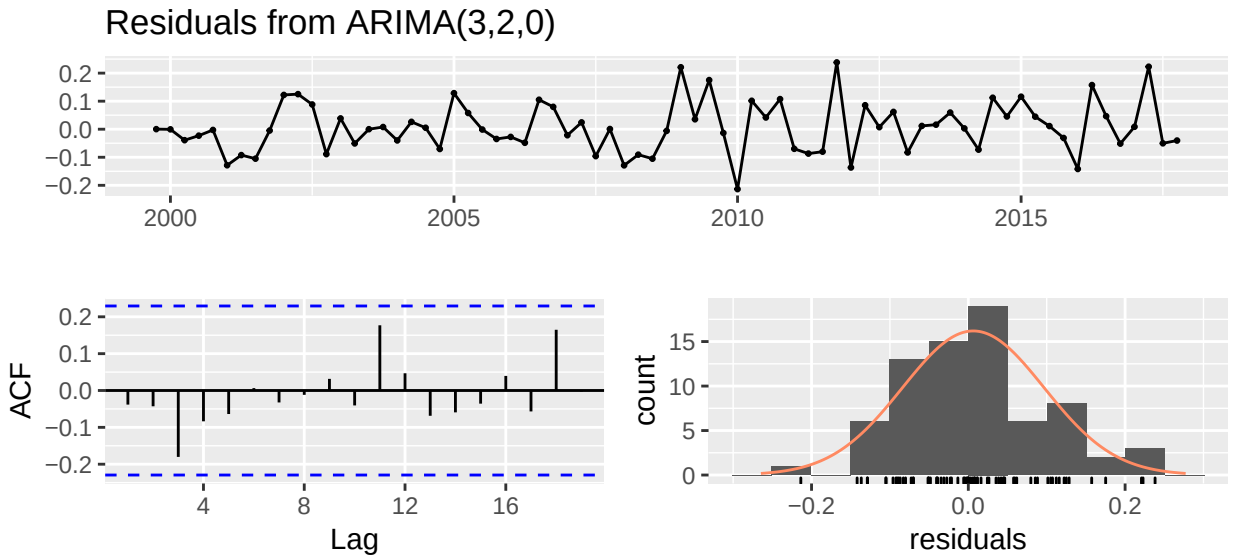
```
##   p d q    in.rmse  out.rmse sa.in.rmse sa.out.rmse        BIC       AICc
## 1 3 2 0 0.08978189 0.2436863 0.08990669   0.2347017 -117.42402 -125.86868
## 2 4 2 0 0.08964908 0.2412478 0.08987646   0.2320716 -113.21391 -123.60423
## 3 3 2 1 0.08952582 0.2368724 0.09011006   0.2272052 -112.87507 -123.26539
## 4 4 2 1 0.08758632 0.1945476 0.08827318   0.1840904 -111.32439 -123.58797
## 5 3 2 2 0.08857905 0.2299786 0.08944499   0.2186375 -109.59490 -121.85848
## 6 4 2 2 0.08732622 0.1965701 0.08789836   0.1862435 -107.60940 -121.67038
## 7 2 2 2 0.11429509 0.2755513 0.11217473   0.2624264  -82.85943  -93.24975
## 8 2 2 1 0.11923649 0.2158371 0.11676591   0.2057459  -82.02459  -90.46925
## 9 2 2 0 0.14786079 0.1879356 0.14512974   0.1824672  -56.92443  -63.35427
##          AIC
## 1 -126.47474
## 2 -124.52731
```

```
## 3 -124.18847
## 4 -124.90047
## 5 -123.17098
## 6 -123.44816
## 7  -94.17283
## 8  -91.07531
## 9  -63.71247
```

The results of our model versions and their performances are shown in the table above, where the ARIMA model versions are ordered by their BIC values sorted from lowest to highest. We notice that the RMSE calculations in the event of having seasonality versus not having seasonality is actually quite comparable. After the BIC, we choose to prioritize the in-sample RMSE in the table above because there are 73 data points in sample vs only 6 data points in the pseudo-out-of-sample subset. Therefore, we believe model performance based on in-sample RMSE prevents us from choosing a model that overfits to the test data. We can also rely more on in-sample RMSE since we found in **Part III** that the seasonlity is similar between the in-sample data points and the pseudo-out-of-sample subset.

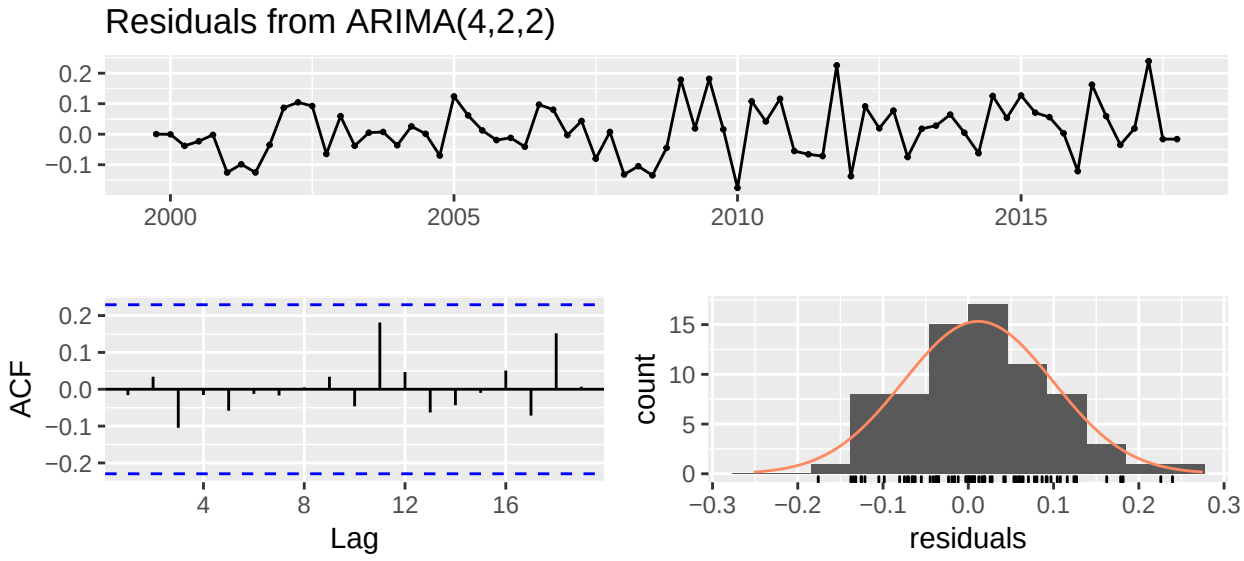
From the results, we see that ARIMA(4,2,2) model has the best in-sample RMSE performance and one of the best pseudo-out-of sample performances, but the ARIMA(3,2,0) model has the best BIC, AICc and AIC scores. We also compare their residuals below:

```
checkresiduals(Arima(ts_pre2018.SA, order = c(3, 2, 0), method = "ML"))
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(3,2,0)
## Q* = 3.7781, df = 5, p-value = 0.5818
##
## Model df: 3.   Total lags used: 8
```

```
checkresiduals(Arima(ts_pre2018.SA, order = c(4, 2, 2), method = "ML"))
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(4,2,2)
## Q* = 1.3939, df = 3, p-value = 0.707
##
## Model df: 6.   Total lags used: 9
```

Both sets of residuals pass the L-jung Box Test with relatively high p-values and fairly normally distributed values, although the ARIMA(4,2,2) residuals appear to be better normally distributed. In choosing between the two models, we are more likely to lean towards the ARIMA(3,2,0) model. While ARIMA(4,2,2) has slightly better RMSE calculations, it is not a significant enough of a gain for us to choose the more complex AIRMA(4,2,2) model over the simpler ARIMA(3,2,0).

B. Seasonally Adjusted Model Fitting using SARIMA

Given that the Seasonally-Adjusted data shows some oscillations due to seasonally, we also consider the possibility that a SARIMA model, or ARIMA model with seasonal components can be used to fit the Seasonally-Adjusted series. It is important to note however, that when we try `auto.arima()` on the Seasonally-Adjusted pre-2018 series, with the seasaonal parameter turned on, we are still provided with a non-seasonal arima model, which is strong evidence that a SARIMA model is not likely the most suitable for the Seasonally-Adjusted series.

```
auto.arima(ts_pre2018.SA, seasonal = T, stepwise = F, approximation = F)
```

```
## Series: ts_pre2018.SA
## ARIMA(3,2,0)
##
## Coefficients:
##           ar1      ar2      ar3
##       -1.0785  -0.9496  -0.8127
## s.e.   0.0740   0.0978   0.0714
##
## sigma^2 estimated as 0.008678:  log likelihood=67.24
## AIC=-126.47   AICc=-125.87   BIC=-117.42
```

Nevertheless, we attempt a home-grown approach to find a suitable SARIMA model below. The SARIMA

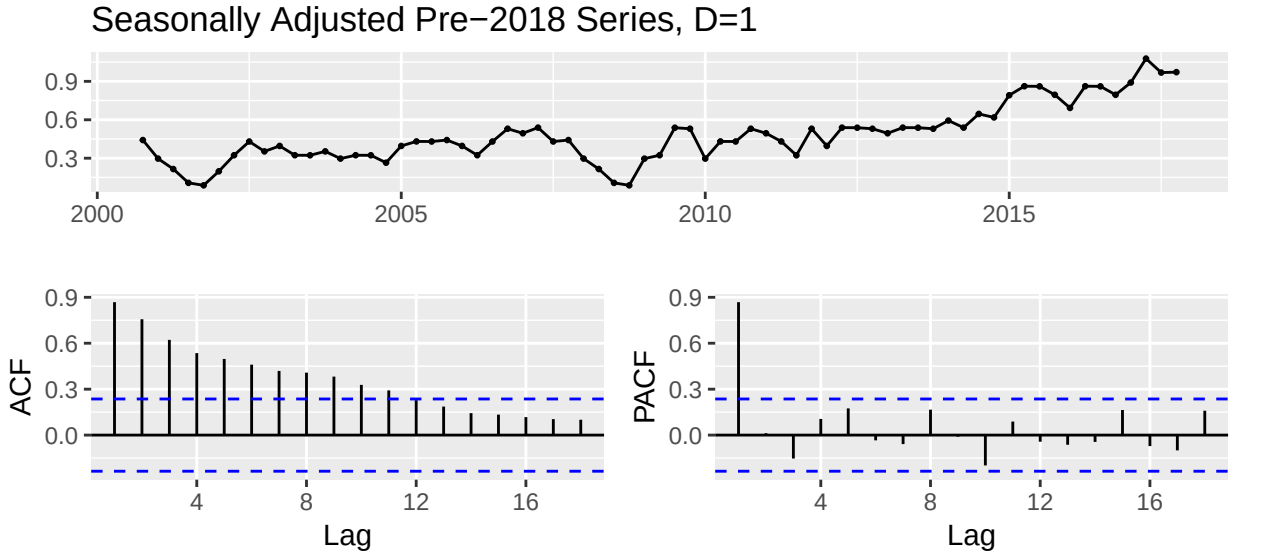
model has the form

$$\Phi(B^s)\phi(B)(1-B^s)^D(1-B)^d y_t = \Theta_Q(B^s)\theta_q(B)\epsilon_t$$

where the variables p,d,q have the same representation as the model form in the non-seasonal ARIMA case. Additional, there are parameters P, D, and Q, which represent Seasonal AutoRegressive order, Seasonal differencing order, and Seasonal moving average order. S represents the seasonal period, in this case being 4 since the data is quarterly.

For a SARIMA model on the SA data, we first apply a seasonal difference to the data, using lag = 4 to signify quarterly seasonality.

```
ts_pre2018.SA %>% diff(lag = 4) %>% ggtsdisplay(main = "Seasonally Adjusted Pre-2018 Series, D=1")
```

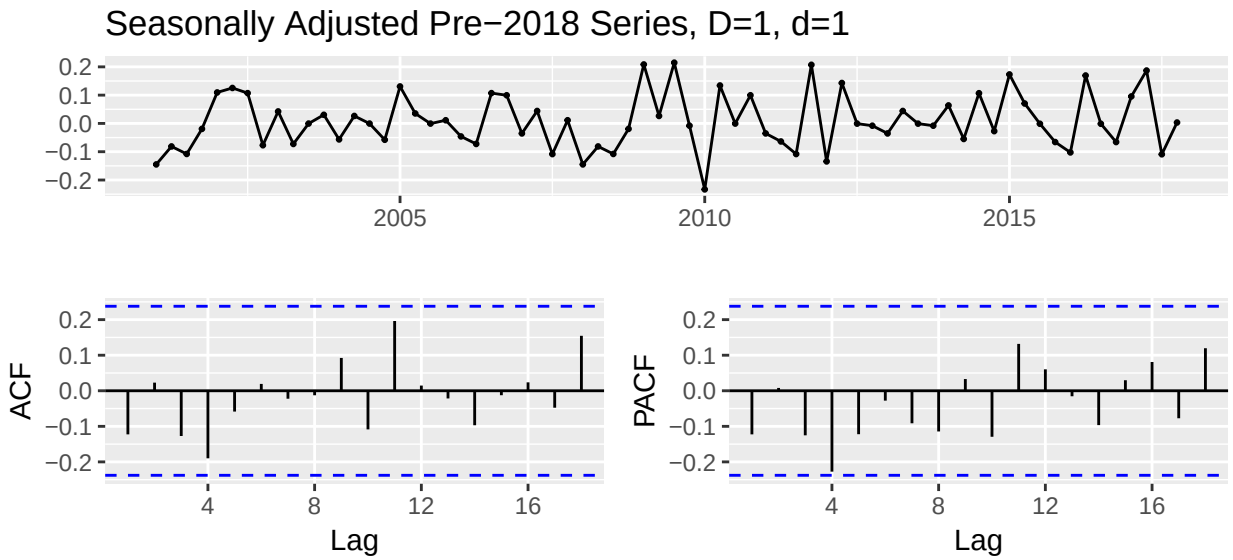



```
ts_pre2018.SA %>% diff(lag = 4) %>% ur.kpss() %>% summary()
```

```
##
## ######################
## # KPSS Unit Root Test #
## ######################
##
## Test is of type: mu with 3 lags.
##
## Value of test-statistic is: 1.3155
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

We find that the resulting seasonally differenced data is not stationary, by the both the plots-based assessment and the KPSS test with a high test-statistic. This results means that first order seasonal differencing (D=1) is insufficient to produce a stationary series, and we must consider first order non-seasonal differencing. Next we apply d=1 on top of the D=1 data set:

```
ts_pre2018.SA %>% diff(lag = 4) %>% diff() %>% ggtsdisplay(main = "Seasonally Adjusted Pre-2018 Series,
```




```
ts_pre2018.SA %>% diff(lag = 4) %>% diff() %>% ur.kpss() %>%
    summary()
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: mu with 3 lags.
##
## Value of test-statistic is: 0.1436
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

Additionally applying non-seasonaly first order differencing does produce a stationary data set with a KPSS test statistic of 0.1436, which is too small to reject the null hypothesis that the resulting data is stationary. Therefore, we attempt SARIMA models using D=1 and d=1 in the for-loop below. From the ACF and PACF plots, we have no significant spikes, and therefore are not easily able to guess proper p, q, P and Q values. To ensure that we test as many values of these 4 parameters as possible, we range all 4 from 0 to 3. We also calculate both types of in-sample and pseudo-out-of-sample RMSE values (with and without seasonality factored in), along with extracting the BIC, AICc and the AIC from the test models. Since there are a large number of models tried in the for loops below, we decided this time to sort by increasing BIC, as we would like to find the SARIMA model with comparable fit while having the lowest complexity. We pull the top 10 models with the best BIC values below.

```
mod_df.SA.2 <- data.frame(p = integer(), d = integer(), q = integer(),
    P = integer(), D = integer(), Q = integer(), AIC = double(),
    AICc = double(), BIC = double(), sa.in.rmse = double(), sa.out.rmse = double(),
    in.rmse = double(), out.rmse = double())

for (p in 0:3) {
    for (q in 0:3) {
        for (P in 0:3) {
```

```
            for (Q in 0:3) {
                # tryCatch() is used to skip over trial parameters that
                # cannot produce a model
                possibleError <- tryCatch(mod <- Arima(ts_pre2018.SA,
                  order = c(p, 1, q), seasonal = list(order = c(P,
                    1, Q), 4), method = "ML"), error = function(e) e)
                if (!inherits(possibleError, "error")) {
                  f = forecast(mod, h = 6)
                  # calculate RMSE on seasonally adjusted values
                  sa.insample.rmse = (mean((mod$fitted - P3$SA$sales)^2))^0.5
                  sa.pred.rmse = mean((f$mean - P3$SA0$sales[74:79])^2)^0.5
                  # calculate RMSE on actual values by multiplying back in
                  # seasonality factor
                  insample.rmse = (mean((mod$fitted * P3$SA.seasonal -
                    c(ts_pre2018))^2))^0.5
                  pred.rmse = mean((f$mean * P3$SA0.seasonal[74:79] -
                    c(ts_post2018))^2)^0.5
                  read <- data.frame(p = as.integer(p), d = 1,
                    q = as.integer(q), P = as.integer(P), D = 1,
                    Q = as.integer(Q), in.rmse = insample.rmse,
                    out.rmse = pred.rmse, sa.in.rmse = sa.insample.rmse,
                    sa.out.rmse = sa.pred.rmse, BIC = mod$bic,
                    AICc = mod$aicc, AIC = mod$aic)
                  mod_df.SA.2 <- rbind(mod_df.SA.2, read)
                }
            }
        }
    }
}
mod_df.SA.2 <- mod_df.SA.2 %>% dplyr::arrange(BIC, in.rmse, sa.in.rmse,
    AICc, AIC, p, q)
print(head(mod_df.SA.2, 10))
```

```
##    p d q P D Q    in.rmse  out.rmse sa.in.rmse sa.out.rmse       BIC
## 1  0 1 0 0 1 0 0.09230958 0.2180023 0.09321679   0.2107546 -120.6847
## 2  0 1 0 0 1 1 0.09073364 0.1717518 0.09134147   0.1614035 -119.0524
## 3  0 1 0 1 1 0 0.09078278 0.1776321 0.09143687   0.1680079 -118.9276
## 4  1 1 0 0 1 0 0.09151680 0.2176034 0.09259438   0.2103411 -117.3629
## 5  0 1 1 0 1 0 0.09148396 0.2189115 0.09260491   0.2116980 -117.3473
## 6  1 1 0 0 1 1 0.08929264 0.1661609 0.09007770   0.1553396 -116.6341
## 7  0 1 1 0 1 1 0.08922117 0.1684325 0.09008668   0.1576892 -116.6183
## 8  1 1 0 1 1 0 0.08929737 0.1717066 0.09012653   0.1617305 -116.5640
## 9  0 1 1 1 1 0 0.08922475 0.1741255 0.09013915   0.1642389 -116.5429
## 10 0 1 0 2 1 0 0.09073909 0.1734276 0.09132872   0.1631234 -114.8489
##         AICc       AIC
## 1  -122.8436 -122.9042
## 2  -123.3068 -123.4914
## 3  -123.1820 -123.3666
## 4  -121.6173 -121.8019
## 5  -121.6017 -121.7863
## 6  -122.9177 -123.2927
## 7  -122.9019 -123.2769
## 8  -122.8475 -123.2225
## 9  -122.8264 -123.2014
```

```
## 10 -121.1324 -121.5074
```

Based on the top 10 models with the best BIC values, we see that the RMSE calcuations are comparable to those of the non-seasonal ARIMA models in the previous section. The BIC, AIC, and AICc values for the best models in this table are all slightly higher than our chosen model ARIMA(3,2,0). Therefore, we conclude that the ARIMA(3,2,0), with lowest BIC, AIC, and AICc values and comparable RMSE calculations is still the best model for the Seasonally-Adjusted data. We also note here that had we sorted by in-sample RMSE, we would have gotten more complex SARIMA models with better RMSE but much worse BIC, AIC, and AICc. Therefore, we believe that choosing by RMSE alone is insufficient to determine the best performing model, and ultimate, ARIMA(3,2,0) provides the best balance between having the best RMSE and BIC, AIC, and AICc measurements.

From our analysis on the various models fit to the Seasonally Adjusted series, we conclude the the ARIMA(3,2,0) model with the following coefficients:

```
mod_SA <- Arima(ts_pre2018.SA, order = c(3, 2, 0), method = "ML")
summary(mod_SA)
```

```
## Series: ts_pre2018.SA
## ARIMA(3,2,0)
##
## Coefficients:
##           ar1      ar2      ar3
##       -1.0785  -0.9497  -0.8127
## s.e.   0.0740   0.0978   0.0714
##
## sigma^2 estimated as 0.008678:  log likelihood=67.24
## AIC=-126.47   AICc=-125.87   BIC=-117.42
##
## Training set error measures:
##                       ME       RMSE        MAE        MPE     MAPE
## Training set 0.006331224 0.08990669 0.06872051 -0.1388562 2.285228
##                    MASE        ACF1
## Training set 0.1431051 -0.03827568
```

is our best model for the Seasonally Adjusted Data set.

C. Non-Seasonally Adjusted Model Fitting using SARIMA

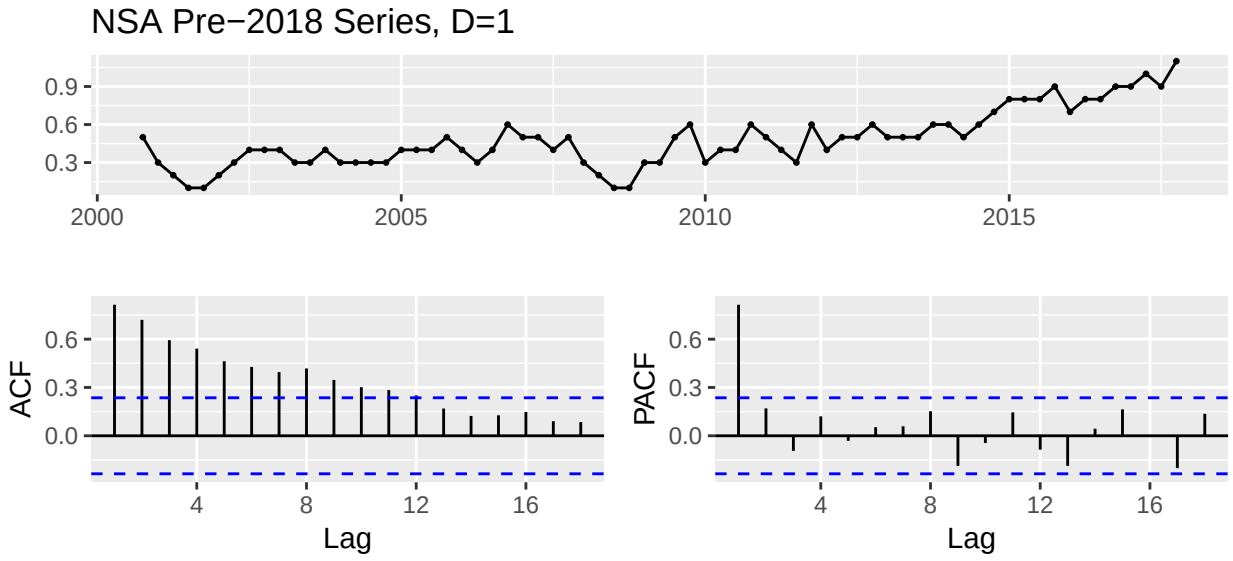
In this section, we will attempt to find the best SARIMA model to fit the NSA data. Again, we use the SARIMA model equation:

$$\Phi(B^s)\phi(B)(1-B^s)^D(1-B)^d y_t = \Theta_Q(B^s)\theta_q(B)\epsilon_t$$

and attempt to optimize the parameters P, D, Q, p, d, and q.

From our EDA, we know that our pre-2018 NSA time series is not stationary, as the ACF is gradually decreasing. Our first step in SARIMA modeling will be to apply first order seasonal differencing to the NSA series:

```
ts_pre2018 %>% diff(lag = 4) %>% ggtsdisplay(main = "NSA Pre-2018 Series, D=1")
```

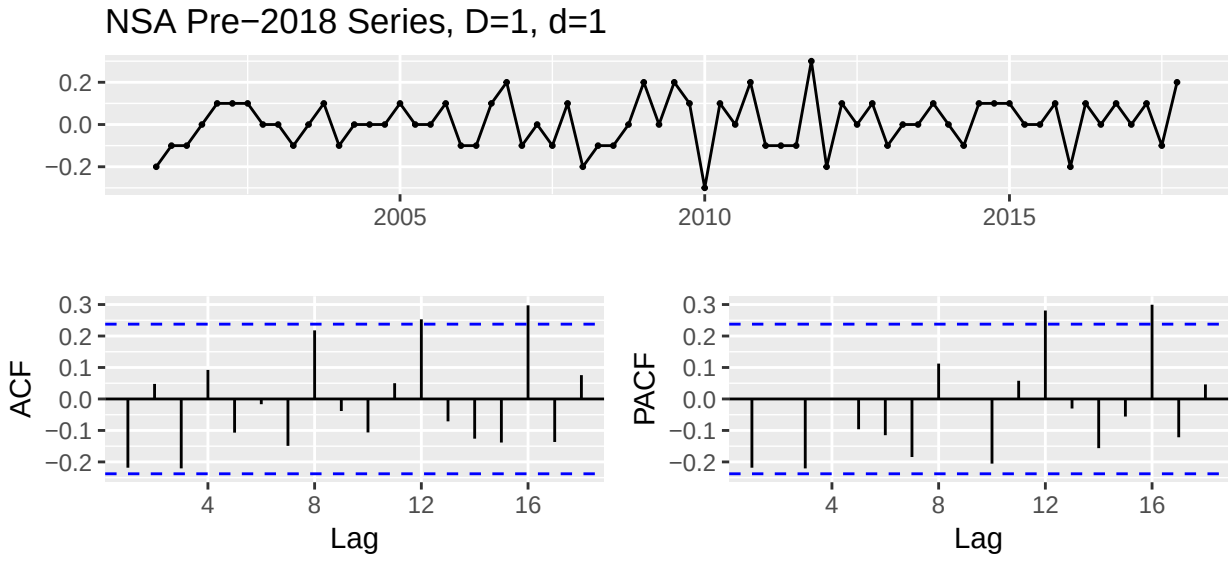



```
ts_pre2018 %>% diff(lag = 4) %>% ur.kpss() %>% summary()
```

```
##
## ######################
## # KPSS Unit Root Test #
## ######################
##
## Test is of type: mu with 3 lags.
##
## Value of test-statistic is: 1.3154
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

From the plots and the KPSS test with a rather large test statistic of 1.3154, we conclude that the seasonally differenced NSA data is not stationary. We then check whether or not and additional first order non-seasonal differencing will produce stationary series:

```
ts_pre2018 %>% diff(lag = 4) %>% diff() %>% ggtsdisplay(main = "NSA Pre-2018 Series, D=1, d=1")
```




```
ts_pre2018 %>% diff(lag = 4) %>% diff() %>% ur.kpss() %>% summary()
```

```
##
## ########################
## # KPSS Unit Root Test #
## ########################
##
## Test is of type: mu with 3 lags.
##
## Value of test-statistic is: 0.225
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.347 0.463  0.574 0.739
```

While there are a couple significant spikes in the ACF and PACF plots, the resulting series does pass the KPSS test with a small enough test-statistics to fail rejection of the null hypothesis. We can also check that first order seasonally is sufficient using `nsdiffs()` function

```
ts_pre2018 %>% nsdiffs()
```

```
## [1] 1
```

and that an addtional non-seasonal first order differencing is sufficient using `nidffs()`:

```
ts_pre2018 %>% diff(lag = 4) %>% ndiffs()
```

```
## [1] 1
```

Now that have a pretty good guess for the d and D parameters, which are both equal to 1, we can use the ACF and PACF plots to determine a starting point for p, q, P and q parameters. In the non-differenced original data with plots in **Part I**, we see one spike one at lag 1 and one at lag 4 in the PACF plot, which are strongly suggestive of:

-an AR(1) seasonal component -an AR(1) nonseasonal component

Then the once seasonally differenced data, there is a significant spike at lag 1 in the PACF, which is again strongly suggestive of -an AR(1) non seasonal component.

From the diagnostic plots, we believe that the model ARIMA(1,1,0)(1,1,0)[4] would be a good candidate. However, since the PACF/ACF is not good for diagnosing plots where $p \neq 0$ and $q \neq 0$, we also vary values

for p, P, q and P in a for loop. Since we have good guesses for p and P, given that we see spikes for the AR(1) seasonal and non-seasonal components, we vary p and P from 0 to 2, ranges that represent +/-1. The q and Q ranges are slighly larger, since we are not able to discern q and Q from plots alone.

In addition to BIC, AICc, and AIC, we calculate the in-sample and pseudo-out-of-sample RMSE values again. This time, since we are measuring performance on the actual NSA data, we only calculate one type of RMSE which takes into consideration the difference between fitted/predicted and actual data. The models are sorted by increasing BIC, as we try over 140 different, so we are again looking for the model with comparable performance but least complexity. In the table below, we pull the top 10 performing models based on BIC.

```
mod_df.NSA <- data.frame(p = integer(), d = integer(), q = integer(),
    P = integer(), D = integer(), Q = integer(), AIC = double(),
    AICc = double(), BIC = double(), in.rmse = double(), out.rmse = double())

for (p in 0:2) {
    for (q in 0:3) {
        for (P in 0:2) {
            for (Q in 0:3) {
                # tryCatch() is used to skip over trial parameters that
                # cannot produce a model
                possibleError <- tryCatch(mod <- Arima(ts_pre2018,
                  order = c(p, 1, q), seasonal = list(order = c(P,
                    1, Q), 4), method = "ML"), error = function(e) e)
                if (!inherits(possibleError, "error")) {
                  f = forecast(mod, h = 6)
                  insample.rmse = (mean(mod$residuals^2))^0.5
                  pred.rmse = (mean((f$mean - c(ts_post2018))^2))^0.5
                  read <- data.frame(p = as.integer(p), d = 1,
                    q = as.integer(q), P = as.integer(P), D = 1,
                    Q = as.integer(Q), in.rmse = insample.rmse,
                    out.rmse = pred.rmse, BIC = mod$bic, AIC = mod$aic,
                    AICc = mod$aicc)
                  mod_df.NSA <- rbind(mod_df.NSA, read)
                }
            }
        }
    }
}
mod_df.NSA <- mod_df.NSA %>% dplyr::arrange(BIC, in.rmse, AICc,
    AIC, p, q)
print(head(mod_df.NSA, 10))
```

```
##    p d q P D Q    in.rmse  out.rmse        BIC       AIC      AICc
## 1  1 1 0 1 1 2 0.09212240 0.3240677 -101.25288 -112.3504 -111.3827
## 2  0 1 1 1 1 2 0.09253209 0.3201047 -100.73018 -111.8277 -110.8600
## 3  0 1 0 1 1 1 0.09761548 0.3537139 -100.21383 -106.8724 -106.4974
## 4  0 1 0 1 1 2 0.09620385 0.3459102 -100.09218 -108.9702 -108.3353
## 5  1 1 0 2 1 1 0.09305758 0.2901912  -98.87058 -109.9681 -109.0004
## 6  1 1 1 1 1 2 0.09078998 0.3265112  -98.86553 -112.1826 -110.8055
## 7  0 1 0 2 1 1 0.09672978 0.3355465  -98.84335 -107.7214 -107.0865
## 8  0 1 1 2 1 1 0.09352777 0.2928870  -98.47477 -109.5723 -108.6046
## 9  1 1 0 2 1 2 0.08960655 0.3670225  -98.44328 -111.7603 -110.3833
## 10 0 1 0 2 1 2 0.08991367 0.3853004  -98.32879 -109.4263 -108.4586
```

Of the top 10 models sorted by BIC, ARIMA(1,1,0)(2,1,2)[4] has the best in sample RMSE while ARIMA(1,1,0)(1,1,2)[4] has the best BIC, AICc and AIC values. However, while in this case, we would be

tempted to choose ARIMA(1,1,0)(2,1,2)[4], we must also note that this more complex model has one of the worst pseudo-out-of-sample RMSE calculations. This means that while ARIMA(1,1,0)(2,1,2)[4] fits the in-sample test data rather well, it suffers from overfitting as it does not generalize well to out of sample data points.

Unlike the models fit to Seasonally-Adjusted data, where we can rely on in-sample RMSE since the seasonality for in-sample data points and pseudo-out-of-sample subset are similar, the NSA data which contains seasonality is much more complex and we therefore need to find a model that best balances pseudo-out-of-sample RMSE and in sample RMSE. One model with comparable in-sample accuracy to the best BIC model, ARIMA(1,1,0)(1,1,2)[4], but has better pseudo-out-of-sample RMSE is ARIMA(1,1,0)(2,1,1)[4]. However, in graphical analysis (not shown here) of the forecast values between ARIMA(1,1,0)(1,1,2)[4] and ARIMA(1,1,0)(2,1,1)[4], we did not see much difference. Given that their RMSE values are still quite comparable, we opted to go with the simpler ARIMA(1,1,0)(1,1,2)[4] based on BIC values, since the gain in RMSE is not significant.

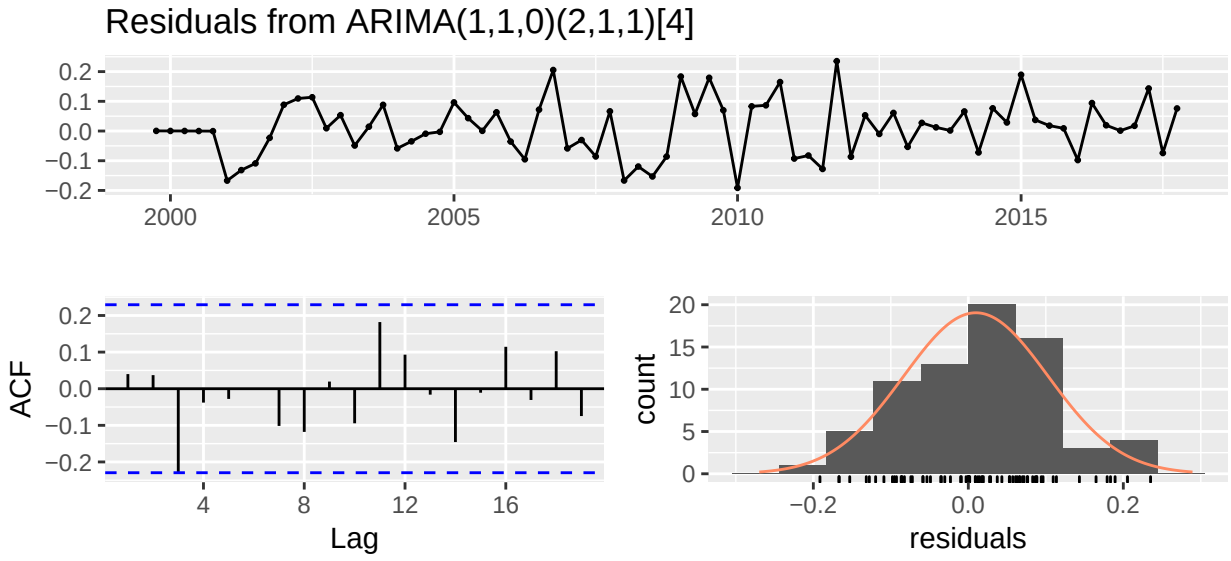
We conclude here that the best fitting SARIMA model on the NSA series is ARIMA(1,1,0)(1,1,2)[4] with model paramters:

```
mod_NSA <- Arima(ts_pre2018, order = c(1, 1, 0), seasonal = list(order = c(2,
    1, 1), 4), method = "ML")
summary(mod_NSA)
```

```
## Series: ts_pre2018
## ARIMA(1,1,0)(2,1,1)[4]
##
## Coefficients:
##           ar1    sar1    sar2     sma1
##       -0.2608  0.6788  0.3092  -0.8976
## s.e.   0.1237  0.1486  0.1338   0.1680
##
## sigma^2 estimated as 0.009877:  log likelihood=59.98
## AIC=-109.97   AICc=-109   BIC=-98.87
##
## Training set error measures:
##                       ME       RMSE        MAE        MPE     MAPE
## Training set 0.009755398 0.09305758 0.07288885 -0.0671081 2.459908
##                    MASE       ACF1
## Training set 0.1514859 0.03989992
```

lastly, we check the residuals of our best model:

```
checkresiduals(mod_NSA)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,1,0)(2,1,1)[4]
## Q* = 6.6531, df = 4, p-value = 0.1554
##
## Model df: 4.   Total lags used: 8
```

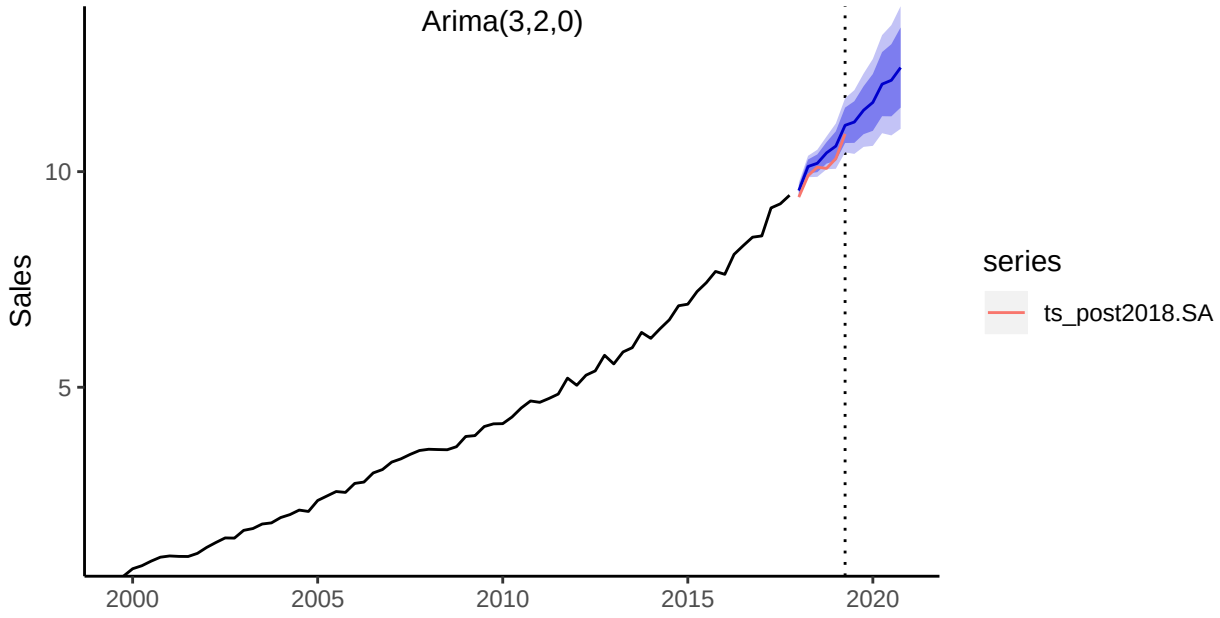
Again, the residuals pass the L-jung Box Test and appear to be generally random noise. The ACF show no signifcant spikes and the residuals appear mostly normal. This allows us to conclude that ARIMA(1,1,0)(1,1,2)[4] is generally good for making predictions.

Part V

Using the best models for the SA data (ARIMA(3,2,0)) and for the NSA data (ARIMA(1,1,0)(1,1,2)[4]), we generate forecasts for the remainder of 2019 (Q3 and Q4) and the entirety of 2020. Since our model is fit to pre-2018 data, we use `forecast()` function to generate the forecasts for these 6 quarters. In each of the forecasts below, which we display graphically, we use parameter `h = 12`, where the first 6 forecast points are generated forecasts for the pseudo-out-of-sample data points, and the last 6 forecast points correspond to the predicted values for the remainder of 2019 and the entirety of 2020.

First we forecast using the best model from the SA data fitting, ARIMA(3,2,0). First we graph the forecast values on just the seasonally adjusted data -

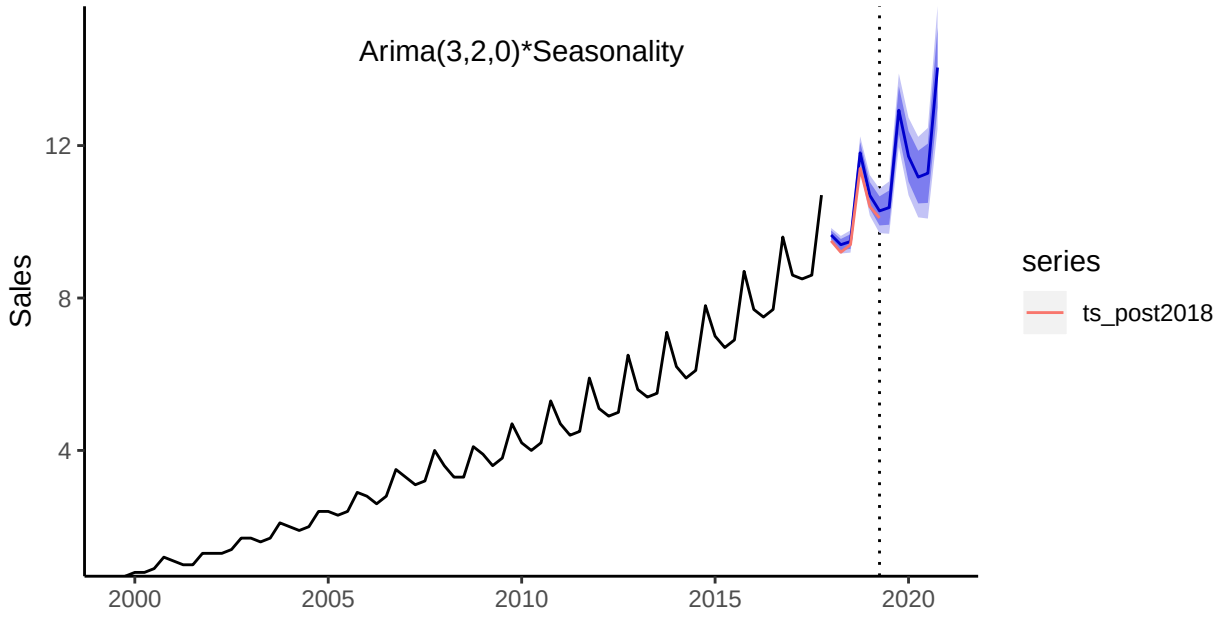
```
# create Seasonally Adjusted post 2018 data
ts_post2018.SA = window(ts(P3$SA0$sales, start = c(1999, 4),
    frequency = 4), start = c(2018, 1))

label1 = "Arima(3,2,0)"
p1 <- ggplot(aes(x = time(ts_pre2018.SA), y = ts_pre2018.SA),
    data = ts_pre2018.SA) + xlab("") + ylab("Sales") + geom_line() +
    geom_vline(xintercept = 2019.25, linetype = "dotted") + geom_forecast(forecast(mod_SA,
    h = 12)) + scale_y_continuous(expand = c(0, 0)) + autolayer(ts_post2018.SA) +
    theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        panel.background = element_blank(), axis.line = element_line(colour = "black")) +
    annotate("text", x = 2010, y = 13.5, label = label1)
p1
```



We can see that the first 6 forecasted values deviate slightly from the seasonally adjusted pseudo-out-of-sample data points (Q1 of 2018 to Q2 of 2019), but otherwise the forcast for the 2019 and 2020 additional quarters seem to keep the general trend of the data. We do note however, that the confidence intervals appear to increase as time increases.

To compare the performance of the SA model, ARIMA(3,2,0), to the best model fit from the NSA data, we also plot the forecast of the SA model with the seasonality factored back in. We take the seasonality from the entire data set, but we partition it to quarters which correspond to those of the forecasted time values. Given the forecast on the same ARIMA(3,2,0), we multiply the mean, upper and lower bounds by the seasonality of the entire data set, and we plot the adjusted forecast values along with the original data below:

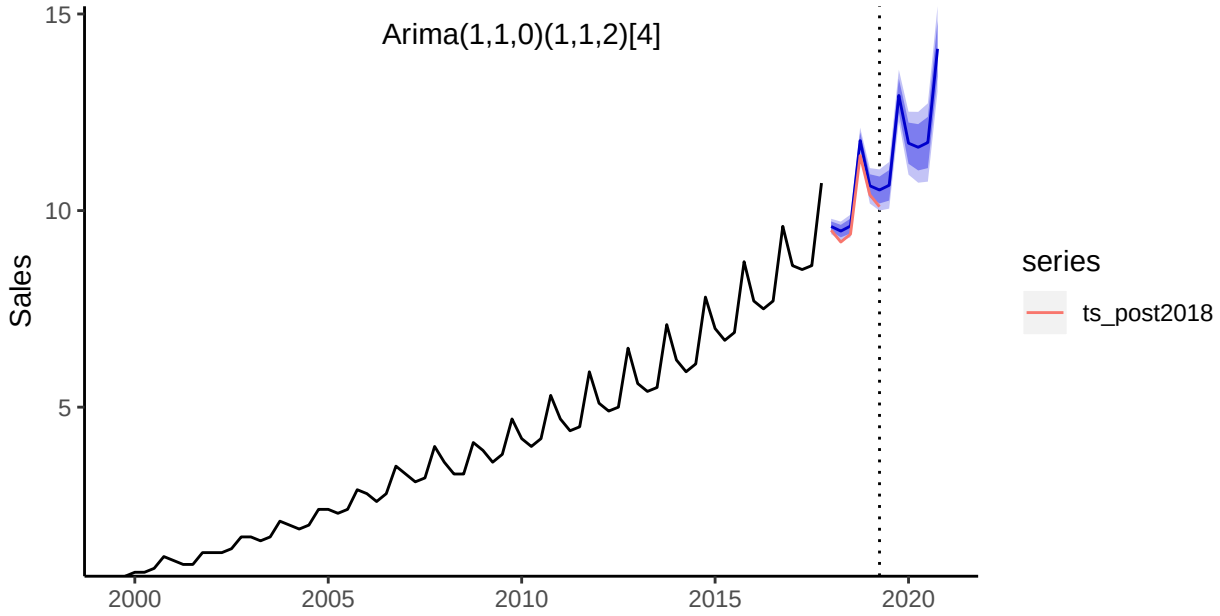
```
# adjustment to the forecast object by seasonality factor
forecast.obj = forecast(mod_SA, h = 12)
forecast.obj$mean = forecast.obj$mean * P3$SA0.seasonal[62:73]
forecast.obj$lower = forecast.obj$lower * P3$SA0.seasonal[62:73]
forecast.obj$upper = forecast.obj$upper * P3$SA0.seasonal[62:73]

label1 = "Arima(3,2,0)*Seasonality"
p <- ggplot(aes(x = time(ts_pre2018), y = ts_pre2018), data = ts_pre2018) +
    xlab("") + ylab("Sales") + geom_line() + geom_vline(xintercept = 2019.25,
    linetype = "dotted") + geom_forecast(forecast.obj) + scale_y_continuous(expand = c(0,
    0)) + autolayer(ts_post2018) + theme(panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(), panel.background = element_blank(),
    axis.line = element_line(colour = "black")) + annotate("text",
    x = 2010, y = 14.5, label = label1)
p
```



Here, we can see that the forecasted values fit quite nicely with the post 2018 pseudo-out-of-sample data points. The forcasted values show that the seasonal oscillations increase in amplitude, which matches the trend from the entire data set, where amplitude of oscillations does appear to be larger for later years than earlier years. Furthermore, the trend is clearly preserved (which we know from the previous graph on the SA data)

Lastly, we plot the forecasted values from the best (ARIMA(1,1,0)(1,1,2)[4]) fitted to the NSA data below.

```
label1 = "Arima(1,1,0)(1,1,2)[4]"
p <- ggplot(aes(x = time(ts_pre2018), y = ts_pre2018), data = ts_pre2018) +
    xlab("") + ylab("Sales") + geom_line() + geom_vline(xintercept = 2019.25,
    linetype = "dotted") + geom_forecast(forecast(mod_NSA, h = 12)) +
    scale_y_continuous(expand = c(0, 0)) + autolayer(ts_post2018) +
    theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        panel.background = element_blank(), axis.line = element_line(colour = "black")) +
    annotate("text", x = 2010, y = 14.5, label = label1)
p
```



We can see from this graph that although the general trend appears to be preserved, the forecasted values do not appear to be as close of a fit to the post-2018 pseudo-out-of-sample data points as the ARIMA(3,2,0) model. In particular, the trough of the forecast oscillations do not dip as far as that of the 2019 Q2 data point, and thus the fit appears to be slightly worse.

Conclusion

Based on our final comparison of the two best models fitted to the SA and the NSA, we see that the ARIMA(3,2,0) model appears to outperform the (ARIMA(1,1,0)(1,1,2)[4]), even when we factor the seasonality back in, when comparing the forecasted values to the post-2018 data points. This is perhaps no surprise, as the SA data trial models had significantly better pseudo-out-of-sample RMSEs (~0.2) compared to those of the NSA data trial models (~0.3 or greater). While the large difference between pseudo-out-of-sample RMSEs because we chose to pick the best NSA models based on BIC first, we also note here that the best pseudo-out-of-sample RMSE was 0.2316 for all NSA SARIMA models we attempted, provided by ARIMA(1,1,3)(1,1,1)[4] shown in the table row below:

```
mod_df.NSA <- mod_df.NSA %>% dplyr::arrange(out.rmse, BIC, in.rmse,
    AICc, AIC, p, q)
print(head(mod_df.NSA, 1))
```

```
##   p d q P D Q    in.rmse  out.rmse       BIC       AIC      AICc
## 1 1 1 3 1 1 1 0.09078217 0.2315999 -93.87901 -109.4156 -107.5489
```

This would mean that out of all 140+ SARIMA models we attempted on the NSA data, the model with the best pseudo-out-of-sample RMSEs still performed worse than almost every non-seasonal ARIMA model we attempted on the SA data. In additional, the top SA models had generally better BIC, AIC, and AICc values than those of the top NSA models.

Based on all of the performance statistics we took into consideration - BIC, AIC, AICc, RMSEs for both in-sample and pseudo-out-of-sample, we conclude that fitting to seasonally adjusted data does appear to result in a better model than fitting to the raw data with clear seasonality. One main reason for this is that applying a seasonal adjustment transformation significantly simplifies the data and allows us to find a simpler, better fitting model. The raw data may contain too much noise due to the seasonality that makes finding a better fitting model much more difficult.

In **Part III** we found that the Seasonally-Adjusted values do not change significantly between the pre-2018

series and the entire time series, which provides us with sufficient evidence that the seasonality is somewhat persistent and fairly unchanged for the next 6 quarters. It is with this analysis that we make the strong assumption that we can transform the predicted values from the non-seasonal ARIMA(3,2,0) model to predict seasonal raw data by simply multiplying by entire series seasonality. This technique could work well for forecasting near term perdictions, but could be very dangerous for frecasting long term. If we were to apply the same technique to future predictions, we should re-apply the seasonal adjustment once we accumulated more yearly data, to verify that the seasonlity has not drastically changed.