Overview of Time series Analysis in R Using Local Restaurant Data and Several Methods

Kenna Schoeler

2017-11-28

Introduction and Motivation:

The purpose of this post is to briefly go over time series analysis in R. I am personally curious about how to analyze a univariate set of data across a span of time and I feel like a lot of the things that go into doing so would make for a very interesting post! I will make a short disclaimer to say that I'm only just familiarizing myself with time series, and I'm by no means a pro, and this post will exclude some of the more complex math and other "pro details"! I will however give a general survey for jumping into a time series analysis using R if you have at least 38-40 data points in your set, and the set isn't very volatile.

Background:

According to the department of statistics at Stockholm University the beginnings of time series analysis date back to the 1920s and 1930s when George Yule, J. Walker, Karl Pearson, and others worked to develop and greatly influence auto regressive modelling. Later in the 1970s, the book: "Time Series Analysis" by G.E.P. Box and G.M. Jenkins was published and within it, the authors gave a more completed method for time series modelling than had otherwise been given before, and the ARIMA method is often refered to as the Box-Jenkins method. Many different techniques for this type of analysis have since developed, including ARCH and GARCH procedures for parametarization of non constant variance, however my focus in this post will only cover analysis which requires a constant variance.

I'll give an example of what libraries and functions to use to decompose time series data, plot (in base R) to get a feel for what you're looking at when you have univariate data recorded over a span of time, and a couple forecasting methods.

Examples:

Let's start with some data that I have for a restaurant's popularity over the course of several years.

```
library(tseries)
```

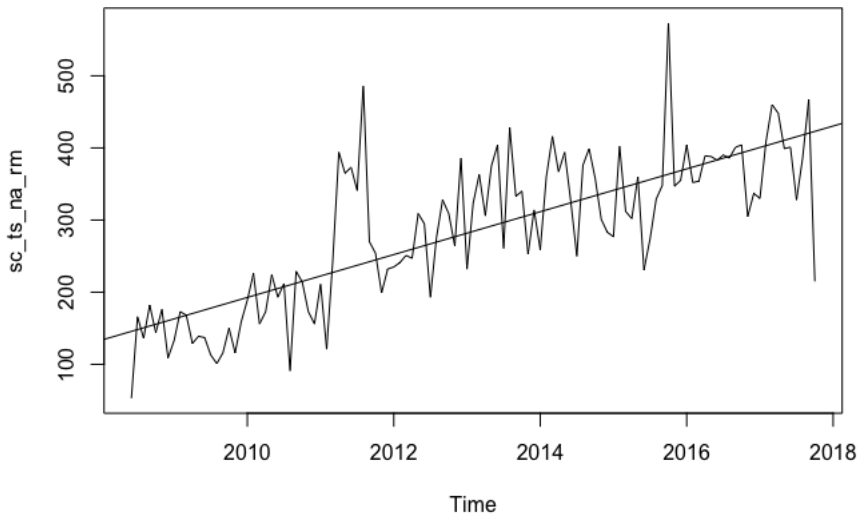
```
## Warning: package 'tseries' was built under R version 3.4.1
```

```
# Read in the data (seated_covers stands for the number of "seated covers" or
# you can consider this "number of guests" the restaurant took in every month
# over the course of nearly 10 years)
seated_covers <- read.csv(
  "~/Dropbox/stat133/stat133-hws-fall17/post02/seated_cvrs.csv")
```

For reproducibility purposes I've included all of the data below:

date	seated_cvrs
1/1/08	NA
2/1/08	NA
3/1/08	NA
4/1/08	NA

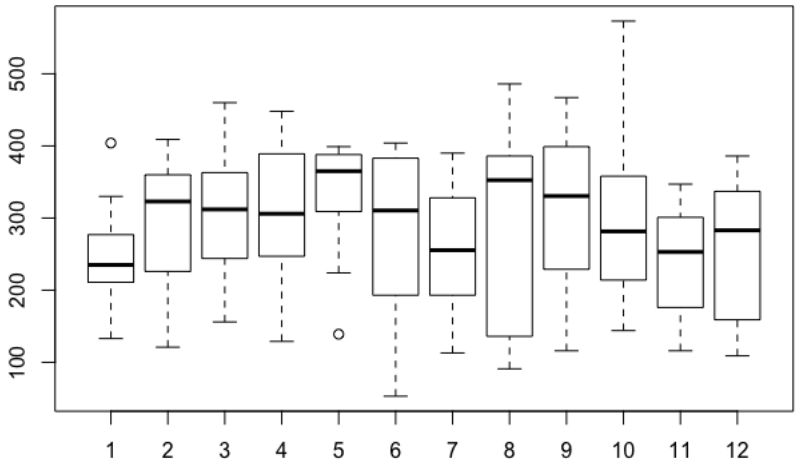
```
# To continue, make seated_covers a time series object
sc_ts <- ts(seated_covers$seated_cvrs, frequency = 12, start = 2008)
sc_ts_na_rm <- na.remove(sc_ts)

# Plot seated_covers to get an idea of how the trend looks
plot(sc_ts_na_rm)

# Add a line to observe the mean
abline(reg = lm(sc_ts_na_rm~time(sc_ts_na_rm)))
```




```
# Look at the summary info for the data
summary(sc_ts_na_rm)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    53.0   199.0   301.0   284.3   367.0   573.0
```

```
# Now use a box plot to check seasonality of the data
boxplot(sc_ts~cycle(sc_ts))
```



The first plot gives us a sense of how the business trend looks in general and the box plot gives us an idea for the seasonal fluctuations the restaurant experiences. 1-12 represent the months of the year and we can see that the month of May (5) is consistently a month that sees a high number of guests to the restaurant. This box plot also shows us outlier months where the number of guests visiting the restaurant happen to be exceptionally high or exceptionally low.

We continue on by adjusting the time series data in order to prepare it for use in the forecasting phase. This requires making the mean and the variance constant, known as making the data “stationary”.

Stationarity is important because when a sequence is stationary a lot of properties that work for independent random variables (ie Law of Large Numbers, and Central Limit Theorem) also work for the time series data.

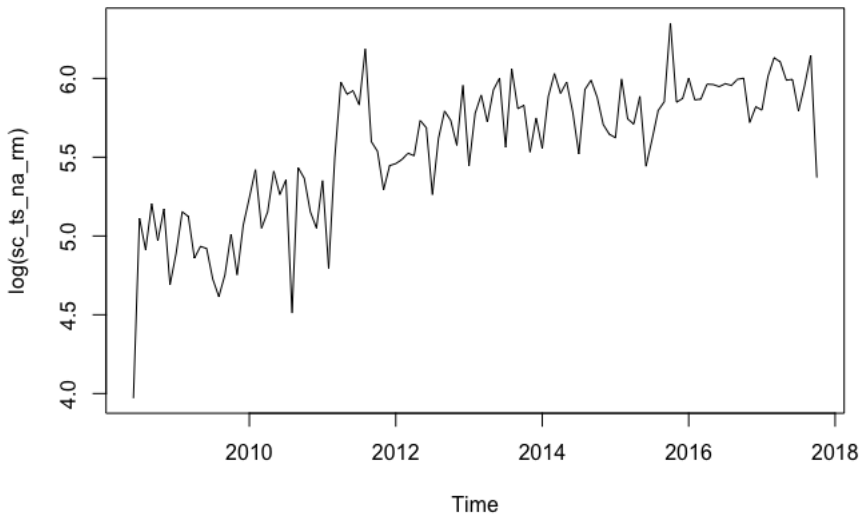
Sationarity is defined uniquely in that a sequence of data can only be stationary or not stationary. Similar to independence being defined uniquely.

We need stationarity so that the model we create to describe the data does not vary in accuracy at different points in time as it would’ve had we not differenced the data.

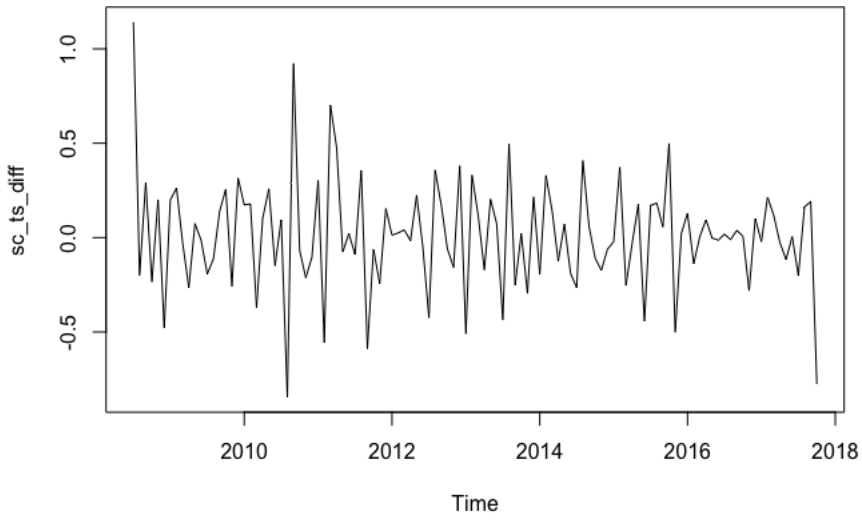
I like to think that the stationarization of data “levels the playing field”.

So:

```
# Do this by taking the log of the data within sc_ts, and plotting what you find
# to see if the plot becomes stationary
plot(log(sc_ts_na_rm))
```




```
# We can see that this plot is not stationary (does not have a constant mean
# with equal variance) so we change that!

# Now try to difference the plot to give it these qualities.
sc_ts_diff <- diff(log(sc_ts_na_rm))


# Here we can see that indeed we have what we want to proceed! The data has a
# constant mean and an equal variance.
plot(sc_ts_diff)
```



By taking the log of the data, we note the time series becomes additive rather than multiplicative. This should be kept in mind depending on what you plan to do with the time series.

Now that we've explored the data a bit, lets look into using an ARIMA method which stands for (Auto-Regressive Integrated Moving Average).

It is important to know that we need to calculate the coefficients for AR, I, and MA seperately.

Note: AR = p, I = d, and MA = q

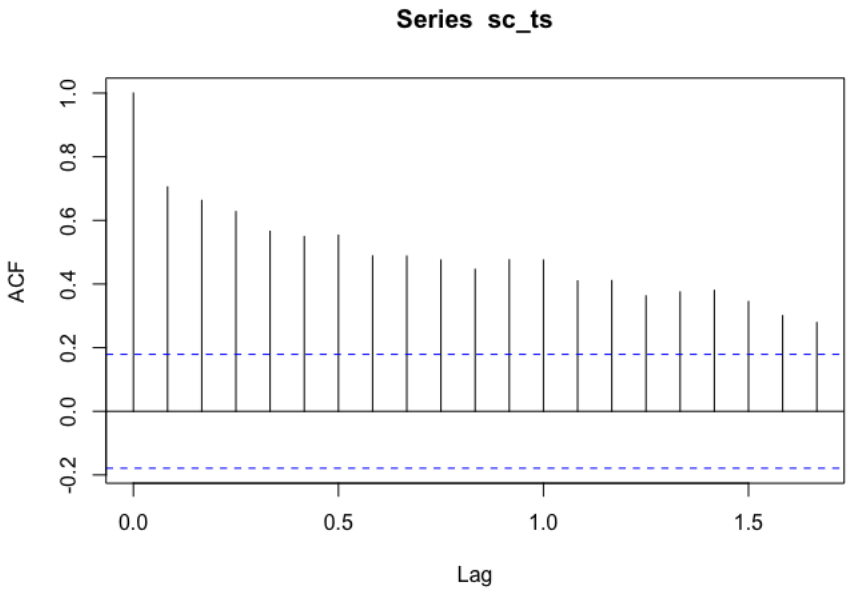
- Where (p) is the number of auto-regressive terms, or the order of auto-regressive components.
- Where (d) is the number of nonseasonal differences needed for stationarity which is the number of differencing operators.
- Where (q) is the number of forcast errors used in the prediction equation (the lag) which is the highest order coefficient of the moving average term.

We already know that d = 1 because it only took differencing of order = 1 to make the time series data stationary.

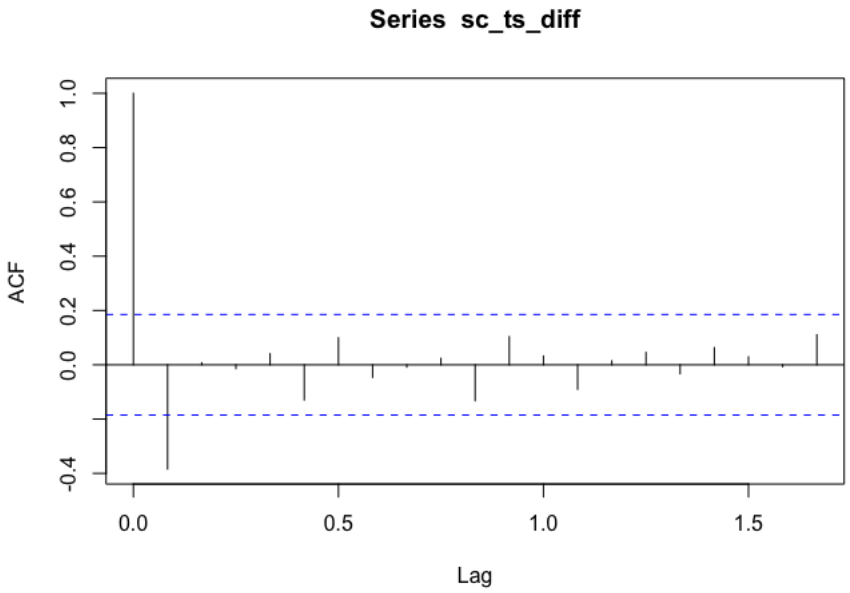
We continue by finding the values of p and q by using R's auto-correlation function `acf()`

To do this step install the package `tseries`:

```
# install.packages("tseries")
library(tseries)

# Now first look at the autocorrelation of the data without any transformations
# made to it
# If the data you're working with contains NA's or missing data you can pass
# the NA's so that the function doesn't fail
plot(acf(sc_ts, na.action = na.pass))
```




```
# We can see that none of the lines are inverted so lets look at the
# stationarized data
plot(acf(sc_ts_diff, na.action = na.pass))
```



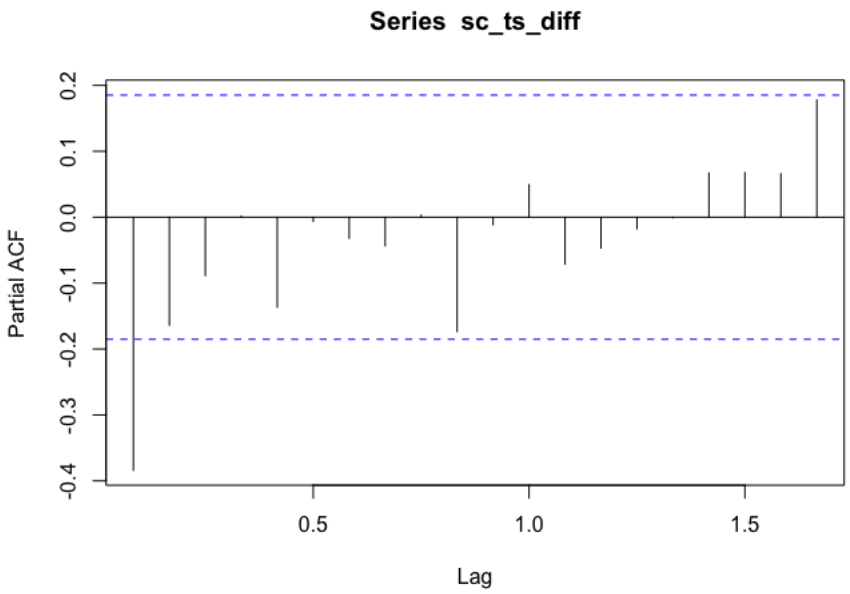

```
# Looking at this plot we can see that the auto-regressive coefficient q is
# equal to 0.

# Without getting too far into the "pro-level" details of it, we see this
# because it is the last positive line before the first inverted negative line
# (a spike at the 0th line) which is what we are looking for.
```

So we have that q = 0

- When using the `acf()` in R to find p, take the position of the line before the first inverted line. We are looking for the lag with the most notable spike.

Now for p:

```
# Instead of acf() we use pacf() to find partial autocorrelation.
plot(pacf(sc_ts_diff, na.action = na.pass))
```



The partial auto correlation function returens a negative tail. This is suggestive that the time series data needs extra differencing. You can continue on to do things like difference the mean, use a Box-Cox Transformation, or any number of other methods, but here I will choose to use the auto.arima function to aid in my forecasting. This function comes from the forecast package so we will need to call its library.

```
library(forecast)
```

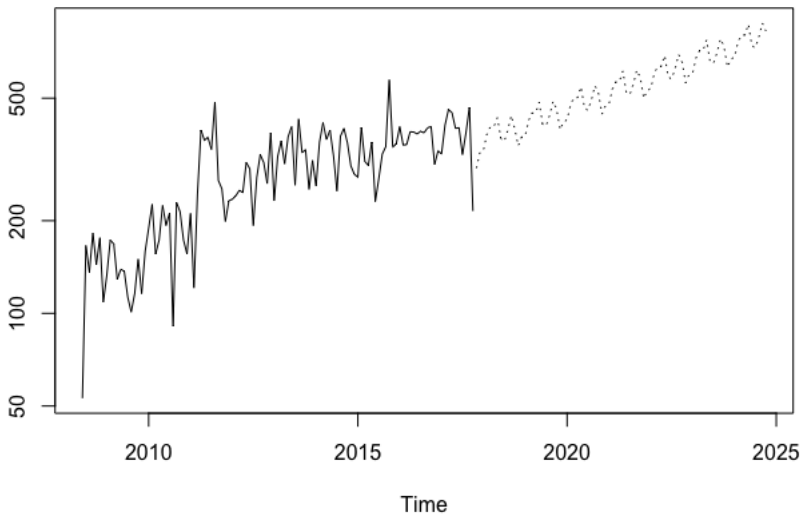
```
## Warning: package 'forecast' was built under R version 3.4.2
```

```
auto.arima(sc_ts_na_rm)
```

```
## Series: sc_ts_na_rm
## ARIMA(1,1,1) with drift
##
## Coefficients:
##          ar1      ma1   drift
##       0.2388  -0.9275  2.4009
## s.e.  0.1099   0.0496  0.6545
##
## sigma^2 estimated as 4170:  log likelihood=-624.96
## AIC=1257.93   AICc=1258.3   BIC=1268.8
```

Using the information given by the auto.arima function we can now attempt to plot a forecast for restaurant popularity using this information as a guide.

```
sc_ts_fitted <- arima(log(sc_ts_na_rm), c(1, 1, 1), seasonal = list(order = c(1, 1, 1), period = 12))

# How far to predict into the future
pred <- predict(sc_ts_fitted, n.ahead = 7*12)

# The code below accounts for the fact that we took the log earlier
ts.plot(sc_ts_na_rm, 2.718^pred$pred, log = "y", lty = c(1, 3))
```



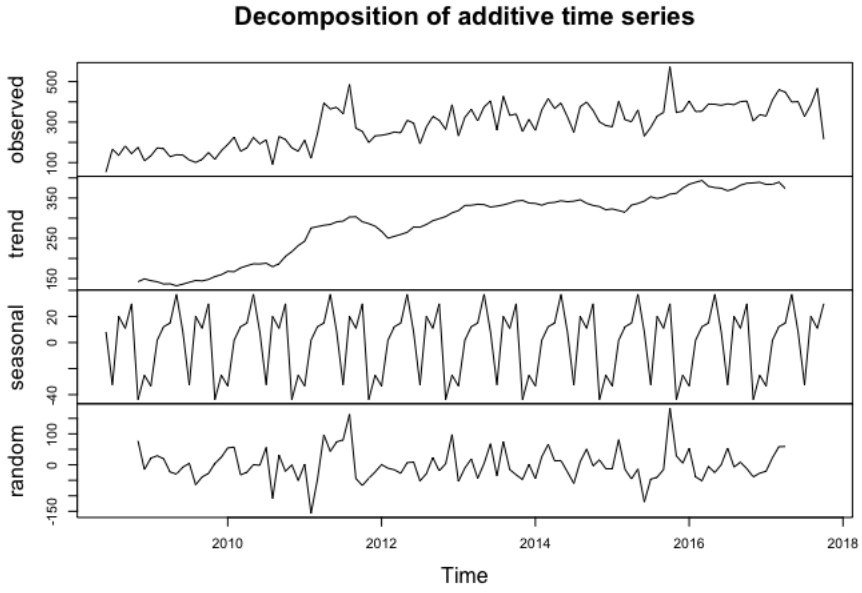
This is good, and it seems to make sense but I want to try another time series analysis.

At this point I'll decide to move on to a decomposition of the time series data to get a better overall idea of what is happening.

Use a `decomposition()` function to view the different features of the time series. This will show us the original observed data plotted over time, the trend line, the seasonality, and random factors

(ie things out of control like road work that might limit the number of guests to the restaurant during a particuar month).

```
# Attain the decomposition of the additive time series by using the decompose()
# function and then plotting the result
dec_sc_ts <- decompose(sc_ts_na_rm)
plot(dec_sc_ts)
```

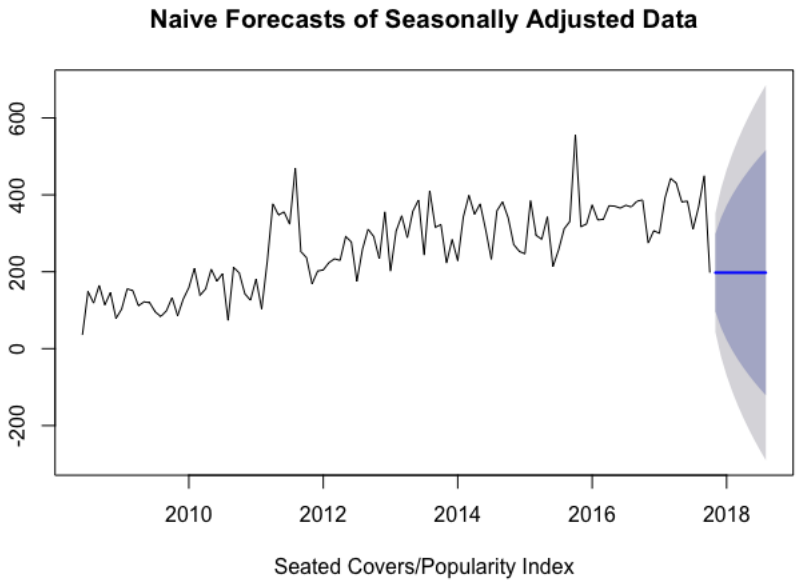


Now that we can see the different components to our time series data, we can also start to consider forcasting with the decomposition as well.

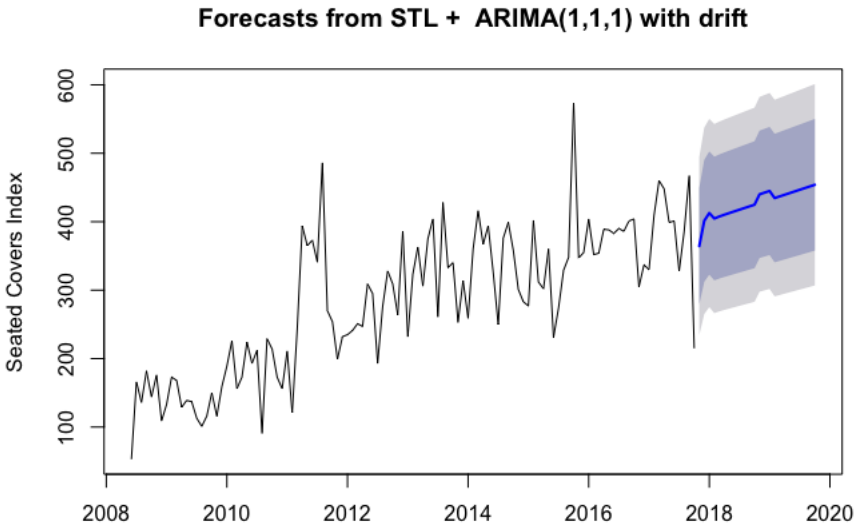
This requires the use of the stl() function from the 'forecast' package.

I will plot each method option so a comparision can be made:

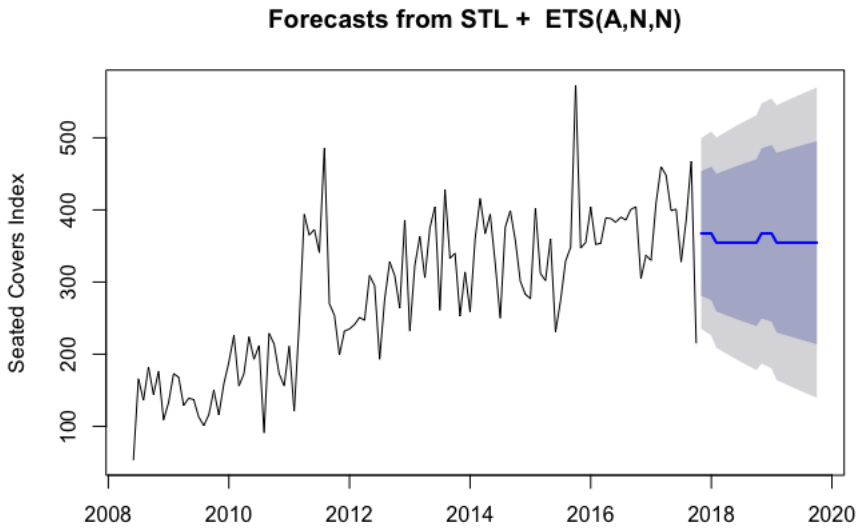
```r
library(forecast)

sc_ts_fit <- stl(
  sc_ts_na_rm, t.window = 5, s.window = "periodic", robust = TRUE)
# t.window is optional but gives us control over how much "wiggle" the plot
# shows

sc_ts_adj <- seasadj(sc_ts_fit)
# make a seasonal adjustment to the stl fit by using function seasadj()

plot(
  naive(sc_ts_adj), xlab = "Seated Covers/Popularity Index",
  main = "Naive Forecasts of Seasonally Adjusted Data")
```



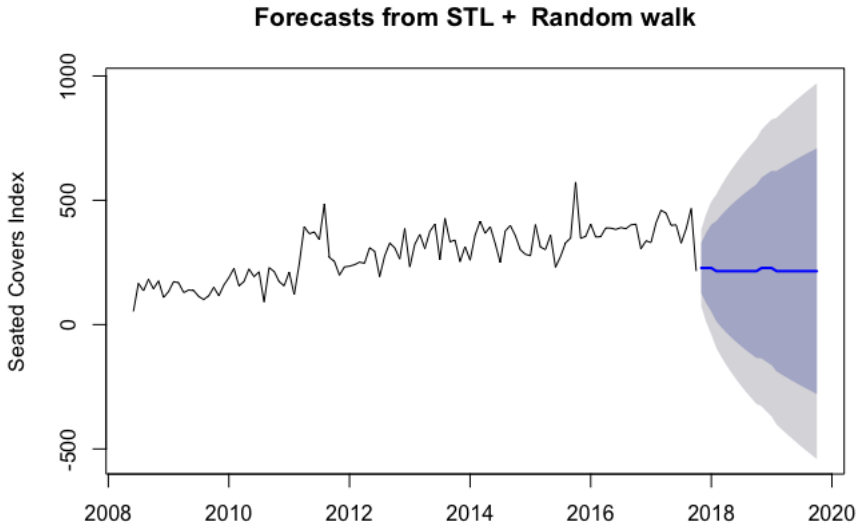

```r
sc_ts_forecast <- forecast(sc_ts_fit, method = "arima")
# Use method arima which is basically inducing a polynomial trend of order d
# into the forecast function

plot(sc_ts_forecast, ylab = "Seated Covers Index")
```



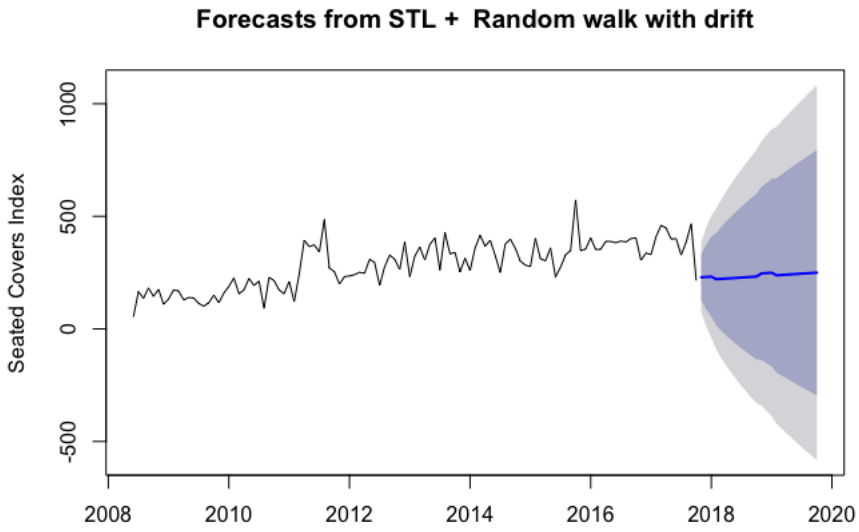

```r
sc_ts_forecast <- forecast(sc_ts_fit, method = "ets")
# Use method ets (uses AAA version of exponential smoothing algorithm)

plot(sc_ts_forecast, ylab = "Seated Covers Index")
```




```r
sc_ts_forecast <- forecast(sc_ts_fit, method = "naive")
# Use method naive (where the last periods actual numbers are used in current
# forecast) Used for highly seasonal data.

plot(sc_ts_forecast, ylab = "Seated Covers Index")
```




```r
sc_ts_forecast <- forecast(sc_ts_fit, method = "rwdrift")
# Use method drift (a variation of the naive method that allows for forecasts to
# increase and decrease over time) The amount of change or "drift" over time is
# set to be the historical average.

plot(sc_ts_forecast, ylab = "Seated Covers Index")
```



All of these plots are useful but it is up to the R user to determine what is of importance and why. Your analysis depends on the data you have, and what you are interested in knowing.

Visuals:

I chose to include visuals throughout the post so that they could aid the explanation of ideas as I go rather than place all of my images in one section.

Discussion:

Though this all might seem a little complex, there are a few general steps to follow when attempting a time series analysis:

- **Step 1**: Read in the data, make it a time series object, and do some exploratory analysis with plot(), abline(), boxplot() and summary().
- **Step 2**: Work on stationarization of the data, this process can be a little time consuming and there is not quite one fix-all way to do this, so experiment with adding MA values if needed, differencing the mean, etc.
- **Step 3**: Choose which type of model fitting works best for your data. This is open ended and really depends on the data set (and how much time you have) but ultimately it is important to play around and look at examples to get a feel for what steps to take and what functions to use.
- **Step 4**: Make your forecast! Use a combination of techniques if necessary and see what the future holds. If what you get seems wrong, revise until you're able to make a few inferences.

Conclusions:

Time series analysis is a complicated subject especially if you are just beginning (like me) so keep in mind that the more familiar you become with the underlying concepts of time series forecasting the better your intuition will be for making accurate predictions. This was a quick survey of some of the things you can do in R to begin an analysis of a time dependant sequence of data, and by no means did I even really scratch the surface. There is interesting math behind each function I've used here and it is worth further investigation/research!

While scouring the internet for insight I did happen to discover Rob Hyndman of the University of Melbourne who seems to be the go-to guy for all things time series analysis! Check out the link for more about him.

The take home message here is that breaking down a time series can be somewhat complicated, but it is an interesting way to delve into the analysis of univariate data, and often times we don't have much more to go off of except the passing of time. So I feel like its good to begin to understand how to use the many different methods for such investigation, and this post is a brief intro intended to pique the reader's interest in further study.

References:

- https://align-alytics.com/seasonal-decomposition-of-time-series-by-loessan-experiment/
- https://www.jstor.org/stable/2669408?seq=1#page_scan_tab_contents Introduction portion of: Time Series and Forecasting: Brief History and Future Research Ruey S. Tsay Journal of the American Statistical Association Vol. 95, No. 450 (Jun., 2000), pp. 638
- https://onlinecourses.science.psu.edu/stat510/
- https://people.duke.edu/~rnau/411arim.htm
- http://people.duke.edu/~rnau/411fcst.htm
- https://www.stat.berkeley.edu/~brill/Papers/encysbs.pdf
- http://www.statistics.su.se/english/research/time-series-analysis/a-brief-history-of-time-series-analysis-1.259451
- https://www.wessa.net/download/stl.pdf