Time Series Analysis

Jeffrey Day

March 1, 2017

## Understanding The Components of a Time Series

This document discusses the components of a time series, the use of first order differencing to remove a trend, full Seasonal Decomposition and the potential uses of seasonal adjusted data. In additon, linear regression and ARIMA models are illustrated.

### Components of Time Series Data

A time series consists of the following components:

* **Trend** - is the base rate of increase/decrease underlying the time series data
* **Cyclic/Seasonal Effect** - is the variation above or below the base trend that repeats cyclically, such as by season
* **Unexplained** - is the variation from the base trend that cannot be explained by the cyclic effect

We wll use public available data from the Steam API. Specifically we will use the monthly average number of players for Counter-Strike from January 2013 thru January 2017.

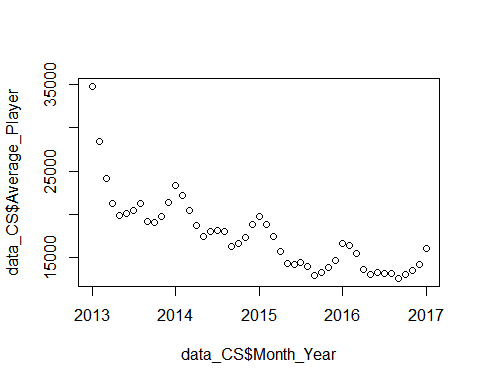
We will echo the code so it can be used as a reference or with subsequent work.

Begin by loading in the necessary libraries

library(forecast)  
library(tidyr)  
library(dplyr)  
library(ggplot2)  
library(stats)

Read in the data for Counter-Strike, filter to the desired example time series and date range.

# Use Public data from Steam API  
data <- read.csv("MonthlyPlayers.csv",sep = ",",header = TRUE,   
 stringsAsFactors = FALSE)  
  
# Use Counter Strike Average Monthly Player data  
data\_CS <- filter(data,data$Game\_Steam\_ID == 10 &   
 data$Month\_Year >=2013.000 &   
 Month\_Year < 2017.083)  
  
# Order the df by time  
data\_CS <- data\_CS[order(data\_CS$X,decreasing=TRUE),]  
  
# Let's see what the time series looks like  
plot(data\_CS$Month\_Year,data\_CS$Average\_Player)



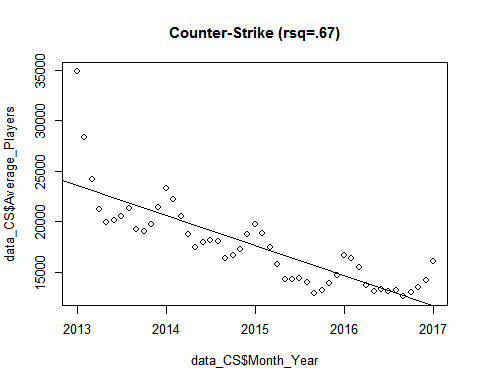
A look at the time series clearly shows declining monthly play but also what looks like seasonal ups and downs.

We can fit a linear regression line to this data and use time to get a feel for the Trend present in the data. It also makes it easier to see the potential seasonal aspects.

reg1 <- lm(data\_CS$Average\_Players~data\_CS$Month\_Year)  
summary(reg1)

##   
## Call:  
## lm(formula = data\_CS$Average\_Players ~ data\_CS$Month\_Year)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -2732.5 -1710.9 -522.3 1079.1 11258.3   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 6015112.8 605715.0 9.931 4.01e-13 \*\*\*  
## data\_CS$Month\_Year -2976.4 300.6 -9.902 4.40e-13 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 2480 on 47 degrees of freedom  
## Multiple R-squared: 0.676, Adjusted R-squared: 0.6691   
## F-statistic: 98.04 on 1 and 47 DF, p-value: 4.403e-13

par(cex=.8)  
plot(data\_CS$Month\_Year, data\_CS$Average\_Players,   
 main="Counter-Strike (rsq=.67)")  
abline(reg1)



The results are not bad for illustrating the trend and the seasonal aspects look pretty clear with the fit line as a reference. Before we formally break out the components of a time series let's first simply remove the trend by using first order differencing. This is also know as making the time series stationary (at least on the mean if not also variance).

### Making a Time Series Stationary

Most statistical forecasting methods are based on the assumption that the time series is stationary. A stationary series is relatively easy to predict as you simply predict that its future will be the same as the past.

Stationarizing a time series through differencing (where needed) is an important part of the process of fitting an ARIMA model. The I in ARIMA is exactly this. Integrated (I) - subtract time series with its lagged series to extract trends from the data and is the d in the ARIMA (p,d,q)

First let's talk a little about how to make a time series stationary, i.e., remove the trend. Most time series can be made stationary by using differencing.

First order differences usually do the trick, and nearly all time series can be made stationary with no more than a second order difference. Differencing may also help reveal seasonality.

Formally the equation is as follows:

1st Differencing (d=1) t= -

Doing first order differences can be useful when the time series lacks a full 4 cycles an a full decomposition is not possible.

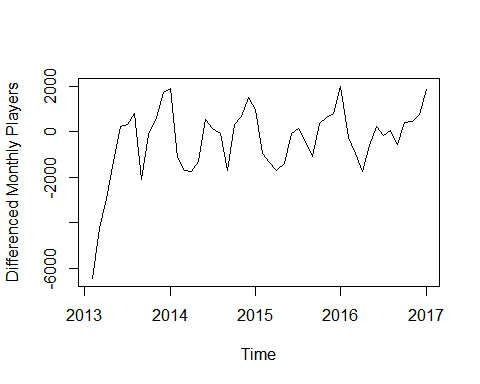
Let's do a first order differencing to remove the Trend in our data so we can see the rise and fall better by time. But first let's create a right proper time series object for R.

# Ceate Time Series of four years  
data1 <- ts(data\_CS$Average\_Players, start = c(2013,1),   
 frequency = 12)

Using the diff function with plot we can see the time series with the trend removed.

Basically the plot of the series *should* show a sideways line with no overall up or down movement over time.

plot(diff(data1),ylab="Differenced Monthly Players")



Yikes, initial launch really screws this up, but repeating pattern can be still be seen and the trend is no longer present (later we shall see that ARIMA confirms d=1 is the correct difference). Also, you can see how future prediction would be realtively easy given a few pieces of information. In this example the dips and dives appear to be seasonal but other series may show non-seasonal spikes (like we see with launch date here) around things like promotions, new content releases, or industry events. This differencing can make it easier to see. Furthermore, the trend can obscure the influence of these other variables due to changing means and variance due to increase/decrease in the dependent variable.

First order differencing can be useful (but not conclusive) when assessing the impact of a marketing program as over and above trend if the peak is greater than the prior same period over period time. If the peak or valley is appreciable different than the prior same time period, then something may be going on. If not, likely no out of normal impact.

### Decompose A Time Series

In order to fully decompose the series, We will use the stl function from the stats package to formally decompose the time series into **Trend**, **Seasonal**, and **Unexplained** (residual) components.

The stl function assumes an "Additive" model as opposed to a "Multiplicative" model.

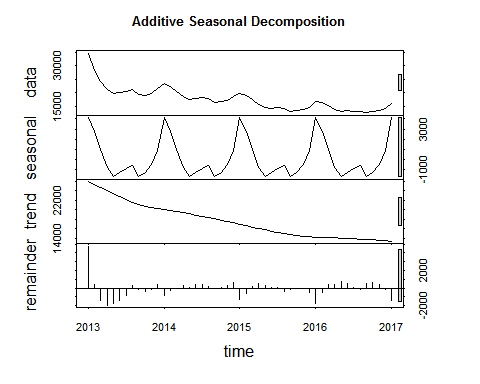
For additive models there is an implicit assumption that the different components affected the time series additively. For monthly data, an additive model assumes that the difference between the January and July values is approximately the same each year. In other words, the amplitude of the seasonal effect is the same each year. The model similarly assumes that the residuals are roughly the same size throughout the series -- they are a random component that adds on to the other components in the same way at all parts of the series.

In many time series involving quantities (e.g. money, wheat production, ...), the absolute differences in the values are of less interest and importance than the percentage changes. For example, in seasonal data, it might be more useful to model that the July value is the same proportion higher than the January value in each year, rather than assuming that their difference is constant. Assuming that the seasonal and other effects act proportionally on the series is equivalent to a multiplicative model.

In my experience, a Multiplicative Model can be applied to a time seris that is additive and all is well. But the converse is not true.

Fortunately, multiplicative models are equally easy to fit to data as additive models! The trick to fitting a multiplicative model is to take logarithms of both sides of the model.

series\_parts <- stl(data1, s.window="periodic")   
# Let's plot the components  
plot(series\_parts,main="Additive Seasonal Decomposition")



On the plot, the top panel labeled **data** is Trend + Seasonal + Residual, i.e., the **actual data**

A quick look at the numbers may illustrative for some.

# Let's see the numbers  
series\_parts

## Call:  
## stl(x = data1, s.window = "periodic")  
##   
## Components  
## seasonal trend remainder  
## Jan 2013 4145.9206 25891.45 4777.09785  
## Feb 2013 2747.8720 25250.29 380.26295  
## Mar 2013 970.1746 24609.12 -1440.14321  
## Apr 2013 -813.8986 23996.81 -1978.45235  
## May 2013 -1717.6973 23384.50 -1765.99602  
## Jun 2013 -1252.3294 22783.21 -1385.99021  
## Jul 2013 -903.6239 22181.92 -796.86193  
## Aug 2013 -581.5193 21612.66 266.39132  
## Sep 2013 -1711.5020 21043.40 -144.10820  
## Oct 2013 -1274.0915 20719.06 -364.12105  
## Nov 2013 -492.3560 20394.72 -199.95887  
## Dec 2013 883.0502 20181.03 357.79048  
## Jan 2014 4145.9206 19967.33 -791.27421  
## Feb 2014 2747.8720 19751.07 -291.26467  
## Mar 2014 970.1746 19534.81 -19.93639  
## Apr 2014 -813.8986 19325.89 244.21112  
## May 2014 -1717.6973 19116.96 66.26409  
## Jun 2014 -1252.3294 18885.31 358.37874  
## Jul 2014 -903.6239 18653.66 364.47586  
## Aug 2014 -581.5193 18384.47 219.89991  
## Sep 2014 -1711.5020 18115.28 -75.27882  
## Oct 2014 -1274.0915 17826.53 84.03039  
## Nov 2014 -492.3560 17537.78 247.05464  
## Dec 2014 883.0502 17241.56 653.57106  
## Jan 2015 4145.9206 16945.34 -1323.99656  
## Feb 2015 2747.8720 16641.47 -588.01959  
## Mar 2015 970.1746 16337.60 131.88611  
## Apr 2015 -813.8986 16047.72 503.80197  
## May 2015 -1717.6973 15757.83 284.04331  
## Jun 2015 -1252.3294 15478.84 23.51773  
## Jul 2015 -903.6239 15199.85 106.08462  
## Aug 2015 -581.5193 14960.57 -393.62854  
## Sep 2015 -1711.5020 14721.29 -123.32447  
## Oct 2015 -1274.0915 14548.17 -45.19638  
## Nov 2015 -492.3560 14375.05 -37.98327  
## Dec 2015 883.0502 14281.30 -509.13309  
## Jan 2016 4145.9206 14187.56 -1690.18696  
## Feb 2016 2747.8720 14143.75 -494.97824  
## Mar 2016 970.1746 14099.94 358.69921  
## Apr 2016 -813.8986 14078.71 402.89583  
## May 2016 -1717.6973 14057.49 730.09792  
## Jun 2016 -1252.3294 13987.86 547.17956  
## Jul 2016 -903.6239 13918.23 98.06368  
## Aug 2016 -581.5193 13838.42 -109.44943  
## Sep 2016 -1711.5020 13758.61 537.37467  
## Oct 2016 -1274.0915 13670.71 606.03146  
## Nov 2016 -492.3560 13582.81 357.71328  
## Dec 2016 883.0502 13482.04 -148.65844  
## Jan 2017 4145.9206 13381.26 -1446.01420

We can run a multiplicative decomposition and get seasonal factors. Here we use the decompose package.

test1<-decompose(data1, type = "multiplicative",filter=NULL)  
# Let's see that seasonal factors  
test1$figure

## [1] 1.1685798 1.1391819 1.0723631 0.9761823 0.9194329 0.9425664 0.9498628  
## [8] 0.9757572 0.9011081 0.9244788 0.9719417 1.0585449

# Place the seasonal factor back into the analysis df  
  
data\_CS$SFM <- test1$seasonal

The seasonal number (or factor) could be used to adjust or remove seasonality from KPI numbers, if a true sense of progress or lack thereof is desired as seasonality might otherwise obscure the real assesment.

In addition, if we are doing forecasting and INFERENCE is desired we could use the decomp information in more intuitive statistical models to help us understand the relationship to the outcome variable.

In fact, let's do that right here and now as an illustration.

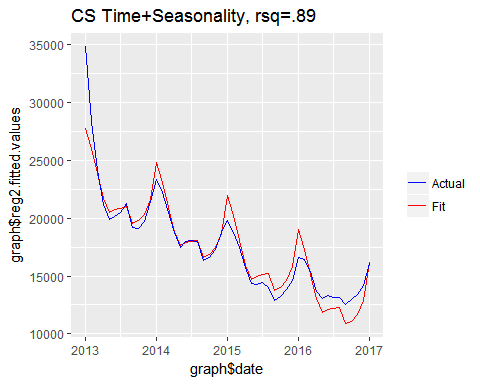
# Attach the Seasonal Factor (all rows, 1st column)  
data\_CS$SF <- series\_parts$time.series[,1]  
# Let's do a linear regression and include the seasonal factor  
# Does a reasonable job along with time with rsq=.89  
  
reg2 <- lm(data\_CS$Average\_Players~data\_CS$Month\_Year+data\_CS$SF)  
summary(reg2)

##   
## Call:  
## lm(formula = data\_CS$Average\_Players ~ data\_CS$Month\_Year + data\_CS$SF)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -2414.3 -696.5 -235.6 283.7 7077.2   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 5.847e+06 3.568e+05 16.389 < 2e-16 \*\*\*  
## data\_CS$Month\_Year -2.893e+03 1.771e+02 -16.340 < 2e-16 \*\*\*  
## data\_CS$SF 1.071e+00 1.130e-01 9.476 2.19e-12 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1459 on 46 degrees of freedom  
## Multiple R-squared: 0.8902, Adjusted R-squared: 0.8855   
## F-statistic: 186.5 on 2 and 46 DF, p-value: < 2.2e-16

reg2.predictions<-predict(reg2,newdata=data\_CS)  
reg2\_error <- sqrt((sum((data\_CS$Average\_Players-reg2.predictions)^2))/nrow(data\_CS))  
reg2\_error

## [1] 1413.514

# Plot the results  
fitted.values = data.frame(reg2$fitted.values)  
original.values = data.frame(data\_CS$Average\_Players)  
graph <- data.frame(fitted.values, original.values)  
graph$date = data\_CS$Month\_Year  
ggplot(data=graph)+   
 geom\_line(aes(y=graph$reg2.fitted.values, colour="Fit",x = graph$date)) +  
 geom\_line(aes(y=graph$data\_CS.Average\_Players, colour="Actual" ,  
 x = graph$date)) +  
 scale\_colour\_manual("",  
 breaks = c("Actual","Fit"),  
 values = c("blue","red")) +  
 ggtitle("CS Time+Seasonality, rsq=.89")



Let's see what the multiplicative decomp gives us.

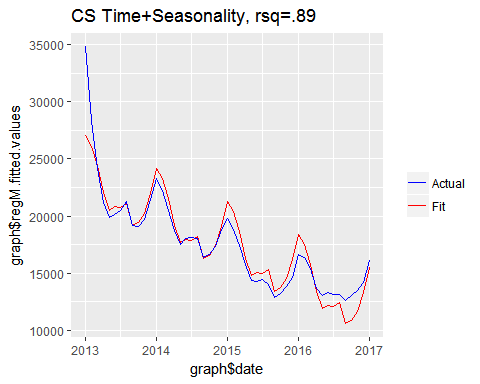
regM <- lm(data\_CS$Average\_Players~data\_CS$Month\_Year+data\_CS$SFM)  
summary(regM)

##   
## Call:  
## lm(formula = data\_CS$Average\_Players ~ data\_CS$Month\_Year + data\_CS$SFM)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -1754.1 -666.3 -354.1 259.1 7802.0   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 5781762.2 372031.2 15.54 < 2e-16 \*\*\*  
## data\_CS$Month\_Year -2871.7 184.5 -15.56 < 2e-16 \*\*\*  
## data\_CS$SFM 22198.5 2494.2 8.90 1.45e-11 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1519 on 46 degrees of freedom  
## Multiple R-squared: 0.881, Adjusted R-squared: 0.8758   
## F-statistic: 170.2 on 2 and 46 DF, p-value: < 2.2e-16

regM.predictions<-predict(regM,newdata=data\_CS)  
regM\_error <- sqrt((sum((data\_CS$Average\_Players-regM.predictions)^2))/nrow(data\_CS))  
regM\_error

## [1] 1472.094

# Plot the results  
fitted.values = data.frame(regM$fitted.values)  
original.values = data.frame(data\_CS$Average\_Players)  
graph <- data.frame(fitted.values, original.values)  
graph$date = data\_CS$Month\_Year  
ggplot(data=graph)+   
 geom\_line(aes(y=graph$regM.fitted.values, colour="Fit",x = graph$date)) +  
 geom\_line(aes(y=graph$data\_CS.Average\_Players, colour="Actual" ,  
 x = graph$date)) +  
 scale\_colour\_manual("",  
 breaks = c("Actual","Fit"),  
 values = c("blue","red")) +  
 ggtitle("CS Time+Seasonality, rsq=.89")



Both yield similiar if not exactly the same result. Now let's see if we include launch date along with time and seasonal factor and see what that does to our model.

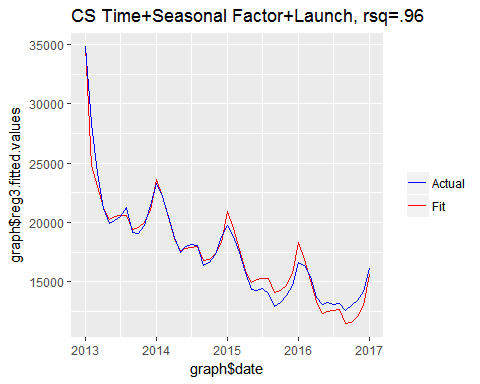
data\_CS$Launch <-ifelse(data\_CS$statMonth=="Jan" & data\_CS$statYear == 2013, 1, 0)  
reg3 <- lm(data\_CS$Average\_Players~data\_CS$Month\_Year+data\_CS$SF+data\_CS$Launch)  
summary(reg3)

##   
## Call:  
## lm(formula = data\_CS$Average\_Players ~ data\_CS$Month\_Year + data\_CS$SF +   
## data\_CS$Launch)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -1663.8 -471.8 -77.3 432.0 3536.1   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 5.373e+06 2.301e+05 23.349 < 2e-16 \*\*\*  
## data\_CS$Month\_Year -2.658e+03 1.142e+02 -23.274 < 2e-16 \*\*\*  
## data\_CS$SF 8.706e-01 7.449e-02 11.688 3.15e-15 \*\*\*  
## data\_CS$Launch 8.533e+03 1.003e+03 8.512 6.27e-11 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 913 on 45 degrees of freedom  
## Multiple R-squared: 0.9579, Adjusted R-squared: 0.9551   
## F-statistic: 341.7 on 3 and 45 DF, p-value: < 2.2e-16

reg3.predictions<-predict(reg3,newdata=data\_CS)  
reg3\_error <- sqrt((sum((data\_CS$Average\_Players-reg3.predictions)^2))/nrow(data\_CS))  
reg3\_error

## [1] 874.931

fitted.values = data.frame(reg3$fitted.values)  
original.values = data.frame(data\_CS$Average\_Players)  
graph <- data.frame(fitted.values, original.values)  
graph$date = data\_CS$Month\_Year  
ggplot(data=graph)+   
 geom\_line(aes(y=graph$reg3.fitted.values, colour="Fit",x = graph$date)) +  
 geom\_line(aes(y=graph$data\_CS.Average\_Players, colour="Actual" ,  
 x = graph$date)) +  
 scale\_colour\_manual("",  
 breaks = c("Actual","Fit"),  
 values = c("blue","red")) +  
 ggtitle("CS Time+Seasonal Factor+Launch, rsq=.96")



Our RMSE has dropped to 875 - Not bad at all. But we see that our fit starts to degrade as we progress through time, especially at the end. This may be due to the trend goofing up the correlations with things as the trend decreases. This is the reason extrapolating a regression model to a time series with a trend is potentially problematic as the future rolls onward. If INFERENCE is a key reason for the development of a model, then this may be okay. If not, ARIMA likely is a better choice to model the data.

But let's see if ARIMA can beat this.

### Fit a Seasonal Arima Model

Arima does a lot of the work for us. It will assess and automatically perform the differencing needed (d), this is the I in ARIMA.

Next it will extract the influence of the previous periods' values on the current period (p), this is the AR in ARIMA.

This is done through developing a regression model with the time lagged period values as independent or predictor variables

Lastly, it extracts the influence of the previous period's error terms on the current period's error (q), this is the MA in ARIMA. Be careful here as it is the moving average of the error terms, not the typical moving average we are all accustom to.

We will use the auto.arima function from the forecast package.

arima\_fit <- auto.arima(data1)  
summary(arima\_fit)

## Series: data1   
## ARIMA(1,1,0)(1,0,0)[12]   
##   
## Coefficients:  
## ar1 sar1  
## 0.8731 0.7843  
## s.e. 0.0976 0.0852  
##   
## sigma^2 estimated as 650494: log likelihood=-394.95  
## AIC=795.9 AICc=796.45 BIC=801.52  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 101.6616 781.4527 600.7118 0.5203055 3.296113 0.2172337  
## ACF1  
## Training set 0.100085

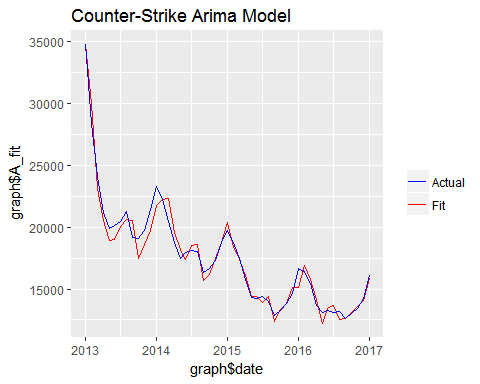
# Caluclate the predicted values  
# Get RMSE to assess (it is shown in summary but just for illustration  
# we do it here as well)  
  
arima\_pred <- predict(arima\_fit)  
arima\_pred <-as.data.frame(arima\_pred)  
arima\_error <- sqrt((sum((arima\_fit$residuals)^2)) / nrow(data\_CS))

The best fit ARIMA model (1,1,0) (1,0,0) [12] tells us that the trend can be removed with first order (d=1) differencing (I) and that an Auto Regression (AR) (p=1) value of 1 is predictive of the current value (loosely we can think of this as our time variable equivalent) and no error term averaging (q=0) (MA) was needed. In addition, the seasonal results (the second set of p,d,q info) say there is a autoregressive (AR=1) seasonal factor at lag 12 [12], i.e, auto regressive with one year ago. Which of course our seasonal decomp showed was true. And tracing our first order differencing through time shows as well.

And it appears we have beaten the regression model above by a little with an RMSE of 781. And we have not included Launch as a covariate.

# Plot the actual versus the predicted as visual  
A\_fit <-(data\_CS$Average\_Players-arima\_fit$residuals)  
fitted.values <- data.frame(A\_fit)  
original.values = data.frame(data\_CS$Average\_Players)  
graph <- data.frame(fitted.values, original.values)  
graph$date = data\_CS$Month\_Year  
ggplot(data=graph)+   
 geom\_line(aes(y=graph$A\_fit, colour="Fit",x = graph$date)) +  
 geom\_line(aes(y=graph$data\_CS.Average\_Players, colour="Actual" ,  
 x = graph$date)) +  
 scale\_colour\_manual("",  
 breaks = c("Actual","Fit"),  
 values = c("blue","red")) +   
 ggtitle("Counter-Strike Arima Model")

## Don't know how to automatically pick scale for object of type ts. Defaulting to continuous.



It seems a little weird as the peaks appear to lag the actual peaks.

Let's include the covariate *Launch* to see if that helps.

covariates <- data\_CS[c("Launch")]  
arima\_fit2 <-auto.arima(data1, xreg=covariates)  
  
summary(arima\_fit2)

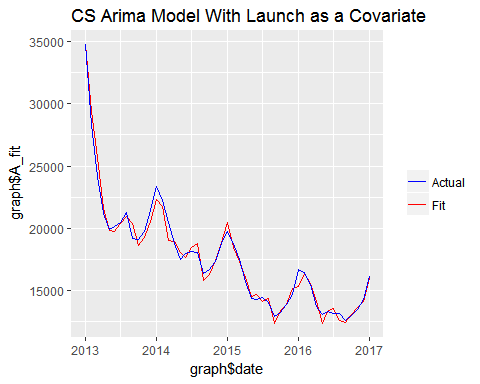
## Series: data1   
## ARIMA(1,1,0)(1,0,0)[12]   
##   
## Coefficients:  
## ar1 sar1 Launch  
## 0.4994 0.8701 4098.0157  
## s.e. 0.1642 0.0549 737.3125  
##   
## sigma^2 estimated as 365568: log likelihood=-382.61  
## AIC=773.23 AICc=774.16 BIC=780.71  
##   
## Training set error measures:  
## ME RMSE MAE MPE MAPE MASE  
## Training set 37.8612 579.4188 434.0961 0.2493504 2.429494 0.156981  
## ACF1  
## Training set 0.1009867

# Caluclate the predicted values  
# Get RMSE to assess (it is shown in summary but just for illustration  
# we do it here)  
  
arima\_pred <- predict(arima\_fit2,newxreg=covariates,se.fit=FALSE)  
arima\_pred <-as.data.frame(arima\_pred)  
arima\_error <- sqrt((sum((arima\_fit2$residuals)^2)) / nrow(data\_CS))  
arima\_error

## [1] 579.4188

# Plot the actual versus the predicted as visual  
A\_fit <-(data\_CS$Average\_Players-arima\_fit2$residuals)  
fitted.values <- data.frame(A\_fit)  
original.values = data.frame(data\_CS$Average\_Players)  
graph <- data.frame(fitted.values, original.values)  
graph$date = data\_CS$Month\_Year  
ggplot(data=graph)+   
 geom\_line(aes(y=graph$A\_fit, colour="Fit",x = graph$date)) +  
 geom\_line(aes(y=graph$data\_CS.Average\_Players, colour="Actual" ,  
 x = graph$date)) +  
 scale\_colour\_manual("",  
 breaks = c("Actual","Fit"),  
 values = c("blue","red")) +  
 ggtitle("CS Arima Model With Launch as a Covariate")

## Don't know how to automatically pick scale for object of type ts. Defaulting to continuous.



That improves the model even more (RMSE=579) and brings the peak prediction back into line. ARIMA seems to be the right weapon here if INFERENCE is not a part of the gig.

But, this may be a total overfit as with ARIMA, "The advantage is that, with enough elements regressed and averaged, you can fit an approximation to almost any time series you like, to whatever precision you like. It basically means that you may fit the data magnificently, but the ARIMA fit could still be total nonsense."

As Frankie Valli says:

You're just too good to be true  
I can't take my eyes off you  
You'd be like heaven to touch  
I wanna hold you so much  
At long last love has arrived  
And I thank God I'm alive  
You're just too good to be true  
Can't take my eyes off you