ARIMA Time Series Forecasting

Introduction

ARIMA models are very popular in financial academic literature, particularly when examining long time series like commodity prices. Here, I use an ARIMA model in R as a demand forecasting tool.

First we import some packages that we'll need later:

```
require( dplyr )
require( forecast )
require( tseries)
```

Data

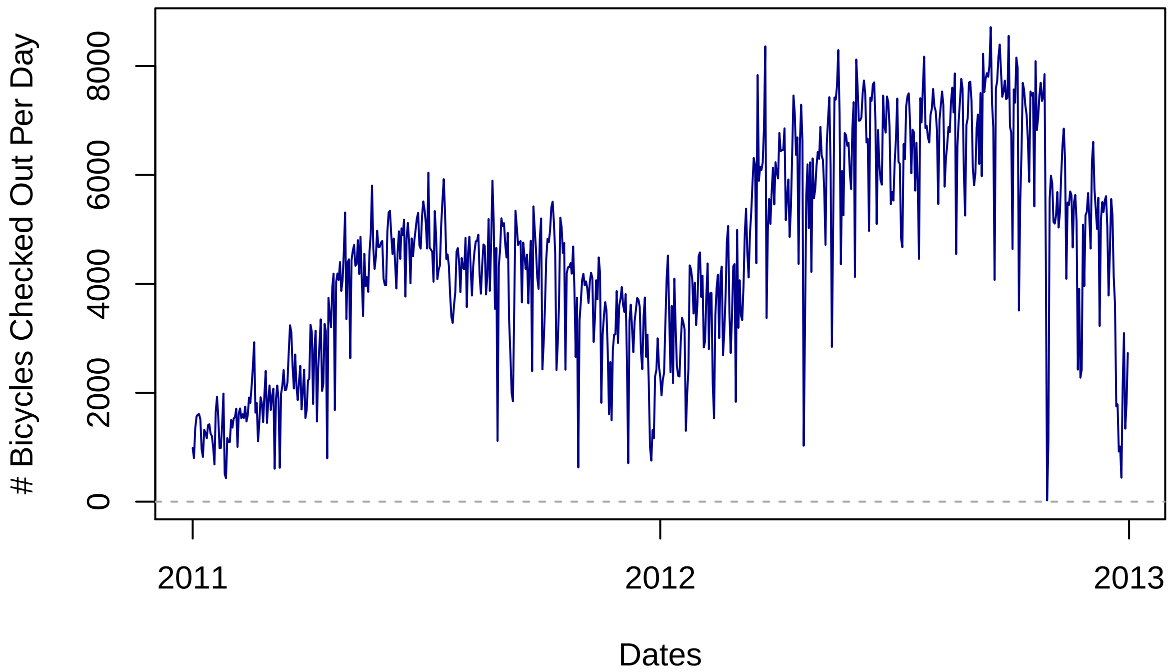
Here we'll import and examine the underlying data. We start by importing the data from our CSV file, and then also printing the first few rows in order to examine it:

```
Daily.Data = read.csv( file = 'day.csv' , header = TRUE )
head( Daily.Data )
```

```
##   instant     dteday season yr mnth holiday weekday workingday weathersit
## 1       1 2011-01-01      1  0    1       0       6          0          2
## 2       2 2011-01-02      1  0    1       0       0          0          2
## 3       3 2011-01-03      1  0    1       0       1          1          1
## 4       4 2011-01-04      1  0    1       0       2          1          1
## 5       5 2011-01-05      1  0    1       0       3          1          1
## 6       6 2011-01-06      1  0    1       0       4          1          1
##       temp    atemp      hum windspeed casual registered  cnt
## 1 0.344167 0.363625 0.805833 0.1604460    331        654  985
## 2 0.363478 0.353739 0.696087 0.2485390    131        670  801
## 3 0.196364 0.189405 0.437273 0.2483090    120       1229 1349
## 4 0.200000 0.212122 0.590435 0.1602960    108       1454 1562
## 5 0.226957 0.229270 0.436957 0.1869000     82       1518 1600
## 6 0.204348 0.233209 0.518261 0.0895652     88       1518 1606
```

Let's also examine the data visually by plotting it:

```
Daily.Data =
  Daily.Data %>%
  mutate( Format.Date = as.Date( dteday ) )

plot(  x = Daily.Data$Format.Date
       , Daily.Data$cnt
       , type = 'l'
       , col = 'darkblue'
       , xlab = 'Dates'
       , ylab = '# Bicycles Checked Out Per Day' )
abline( h = 0 , lwd = 1 , lty = 2 , col = 'darkgray' )
```



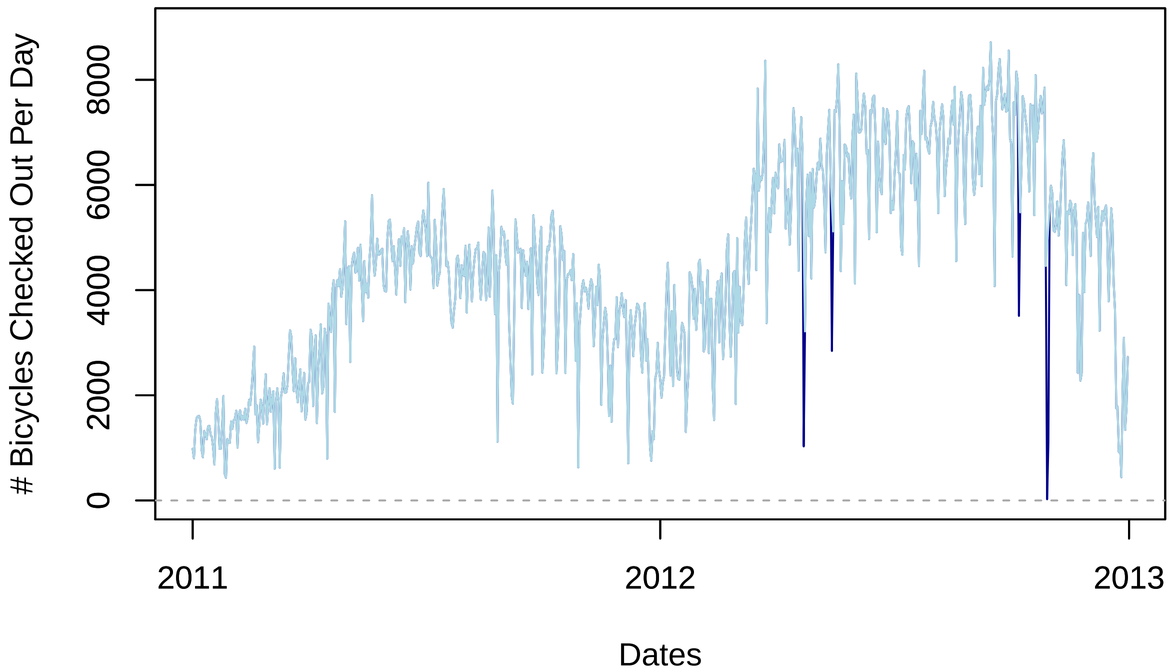
One thing that we can visually note from this plot is the huge variation in the data and the wide amount of dispersion. There are probably also some faulty observations here, as noted by some of the large and sudden spikes – including the one that seems to go to zero on one day. These are most likely errors and should be filtered out.

Luckily R provides us with a tool to easily clean and filter time series, using **tsclean**:

```
Daily.Data.Manipulated =
  Daily.Data %>%
  mutate( Time.Series.Counts = ts( cnt ) ) %>%
  mutate( Cleaned.Counts = tsclean( Time.Series.Counts ) )
```

Let's then plot this new data over the old data, for visual comparison:

```
ylim = c( 0 , 9000 )
plot(  x = Daily.Data.Manipulated$Format.Date
       , Daily.Data.Manipulated$cnt
       , type = 'l'
       , col = 'darkblue'
       , xlab = 'Dates'
       , ylab = '# Bicycles Checked Out Per Day'
       , ylim = ylim )

par(new = TRUE )
plot(  x = Daily.Data.Manipulated$Format.Date
       , Daily.Data.Manipulated$Cleaned.Counts
       , type = 'l'
       , col = 'lightblue'
       , xlab = NA
       , ylab = NA
       , ylim = ylim )

abline( h = 0 , lwd = 1 , lty = 2 , col = 'darkgray' )
```



This approach allows us to quickly and easily see the exact data points that were cleaned using the automated process. And these generally make sense – in fact, we see the one point that went to zero that we identified earlier was effectively cleaned out. Great.

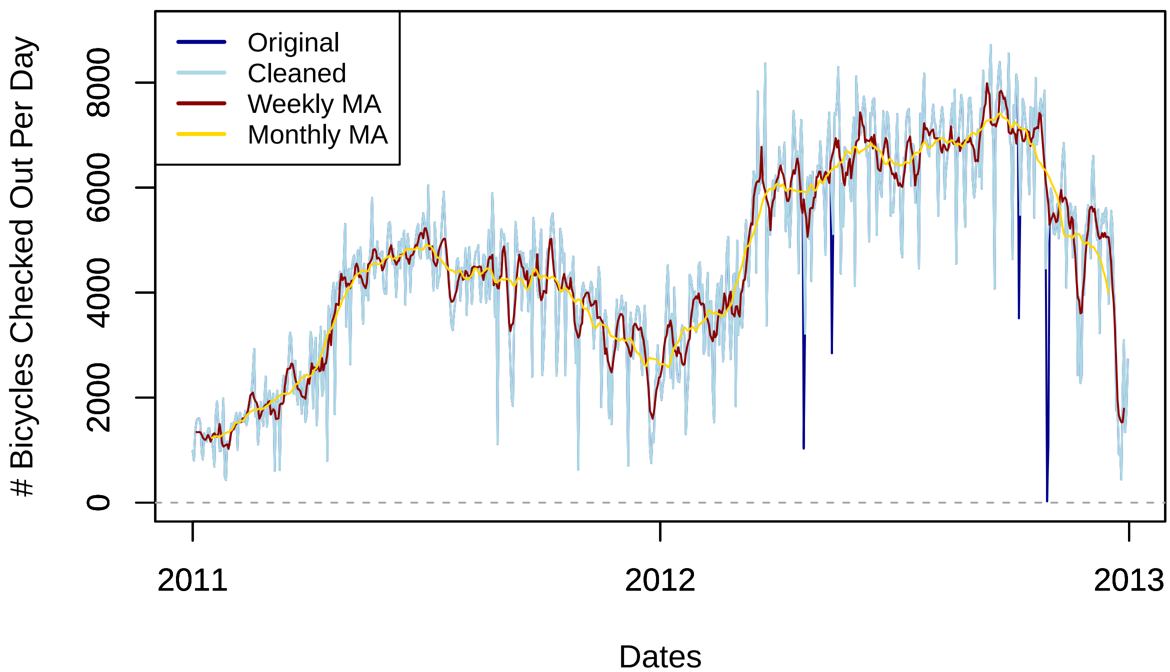
There's still one small feature or problem of the data, though, which is that it is still very volatile, even after the initial cleaning process. For further analysis, we should probably smooth it. One of the most common and convenient ways to smooth a time series like this would be to using a moving average, which we do in the below. Note how you can change the window size of your moving average:

```
Daily.Data.Manipulated =
  Daily.Data.Manipulated %>%
  mutate( Moving.Average.Weekly.Counts = ma( Cleaned.Counts , order = 7 ) ) %>%
  mutate( Moving.Average.Monthly.Counts = ma( Cleaned.Counts , order = 30 ) )
```

Let's add these features to our plot to see how it worked:

```
ylim = c( 0 , 9000 )
plot(  x = Daily.Data.Manipulated$Format.Date
       , Daily.Data.Manipulated$cnt
       , type = 'l'
       , col = 'darkblue'
       , xlab = 'Dates'
       , ylab = '# Bicycles Checked Out Per Day'
       , ylim = ylim )

par(new = TRUE )
plot(  x = Daily.Data.Manipulated$Format.Date
       , Daily.Data.Manipulated$Cleaned.Counts
       , type = 'l'
       , col = 'lightblue'
       , xlab = NA
       , ylab = NA
       , ylim = ylim )
```

```
par( new = TRUE )
plot(  x = Daily.Data.Manipulated$Format.Date
       , Daily.Data.Manipulated$Moving.Average.Weekly.Counts
       , type = 'l'
       , col = 'darkred'
       , xlab = NA
       , ylab = NA
       , ylim = ylim )

par( new = TRUE )
plot(  x = Daily.Data.Manipulated$Format.Date
       , Daily.Data.Manipulated$Moving.Average.Monthly.Counts
       , type = 'l'
       , col = 'gold'
       , xlab = NA
       , ylab = NA
       , ylim = ylim )

abline( h = 0 , lwd = 1 , lty = 2 , col = 'darkgray' )

legend( 'topleft'
        , legend = c( 'Original' , 'Cleaned' , 'Weekly MA' , 'Monthly MA'  )
        , lwd = 2
        , col = c( 'darkblue' , 'lightblue' , 'darkred' , 'gold' )
        , cex = 0.8 )
```

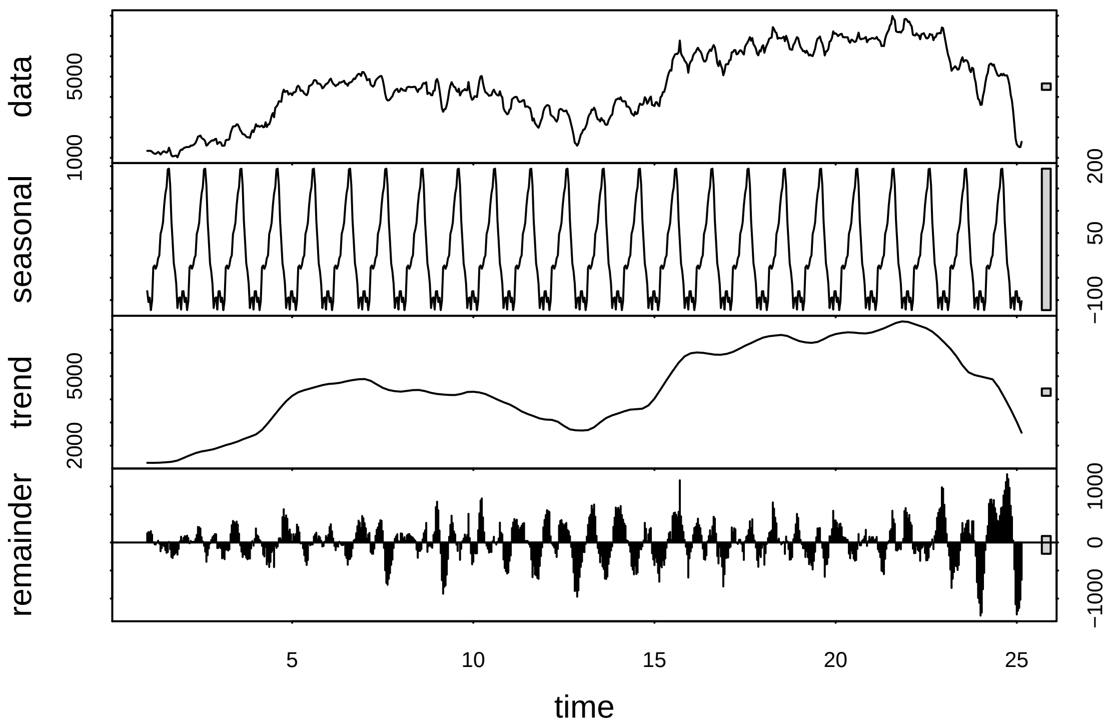


The wider a moving average widnow you use, the smoother your data will be – this is confirmed by our plot, where the monthly moving average line is clearly smoother than the weekly. Going forward, however, the rest of our analysis will only use the data resulting from the weekly moving average.

Time Series Data Decomposition

We can use several functions to very easily remove the seasonal component from the underlying data:

```
Converted.Counts =
  Daily.Data.Manipulated$Moving.Average.Weekly.Counts %>%
  na.omit %>%
  ts( frequency = 30 )

Decomposed.Series =
  Converted.Counts %>%
  stl( s.window = 'periodic' )

Deasonalized.Counts =
  seasadj( Decomposed.Series )

plot( Decomposed.Series )
```



Stationarity

As with many statistical tools, an ARIMA model assumes that its inputs will be stationary. The ADF test (Augmented Dickey-Fuller) is a statistical test that allows us to verify whether or not this is true for the time series in question:

```
adf.test( Converted.Counts , alternative  = 'stationary'  )
```

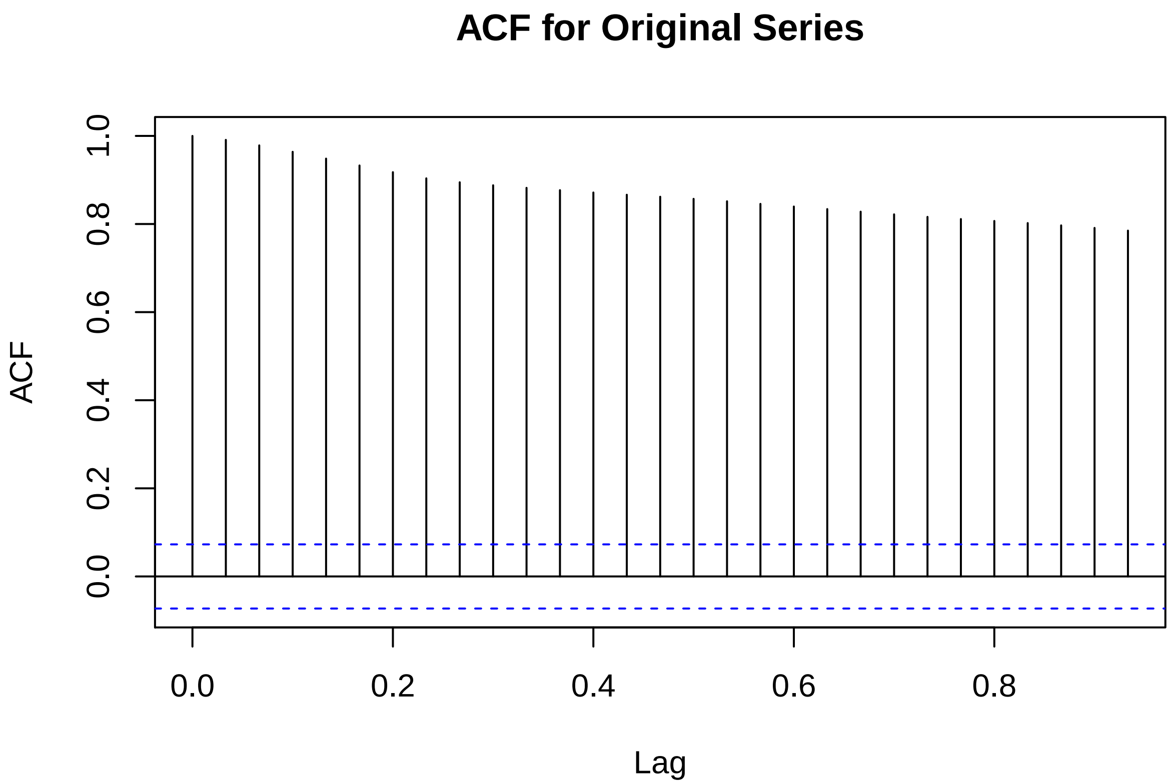
```
## Warning in adf.test(Converted.Counts, alternative = "stationary"): p-value
## greater than printed p-value

##
##  Augmented Dickey-Fuller Test
```

```
##
## data:  Converted.Counts
## Dickey-Fuller = -0.2557, Lag order = 8, p-value = 0.99
## alternative hypothesis: stationary
```

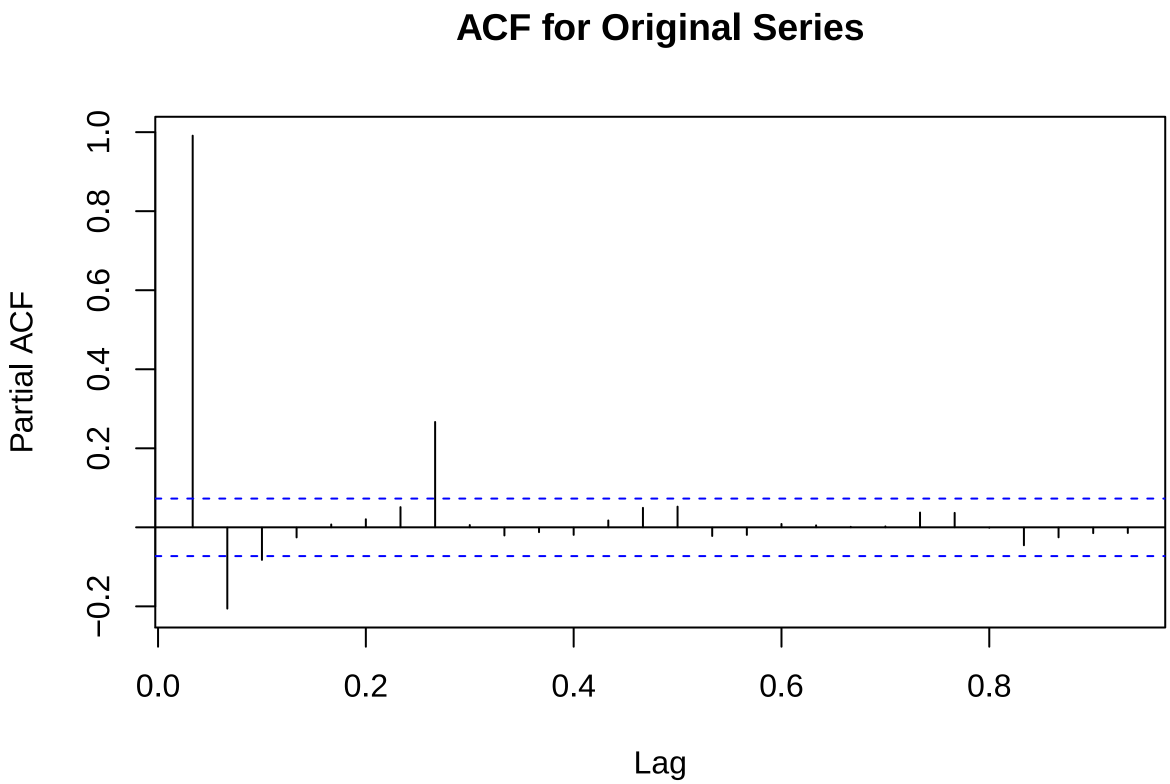
If the p-value of the ADF test were less than 0.05, we could say that the time series was stationary, but it clearly is not. This also checks with a visual inspection of the time series – it clearly does not look stationary.

A time series can usually be made stationary through differencing, and differencing is a component of the ARIMA model. What we need to do next is identify by how much we need to difference the data. For this we'll start by looking at some autocorrelation plots:

```r
acf( Converted.Counts , main = 'ACF for Original Series'  )
```




```r
pacf( Converted.Counts , main = 'ACF for Original Series'  )
```



Based on the large spike in the PACF plot at the first lag, let's try that lag to difference our time series, and see what that looks like:

```
Diffed.1 = diff( Deasonalized.Counts , differences = 1 )

adf.test( Diffed.1 , alternative = 'stationary' )
```

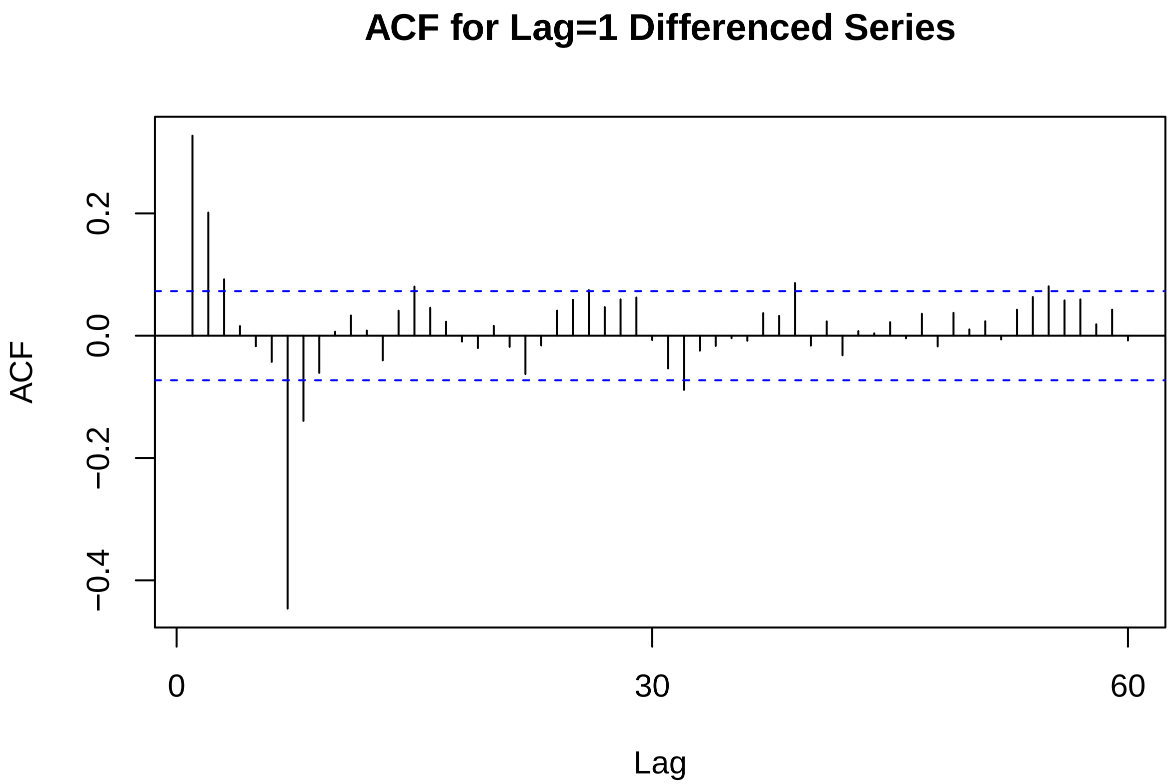
```
## Warning in adf.test(Diffed.1, alternative = "stationary"): p-value smaller
## than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  Diffed.1
## Dickey-Fuller = -9.9255, Lag order = 8, p-value = 0.01
## alternative hypothesis: stationary
```

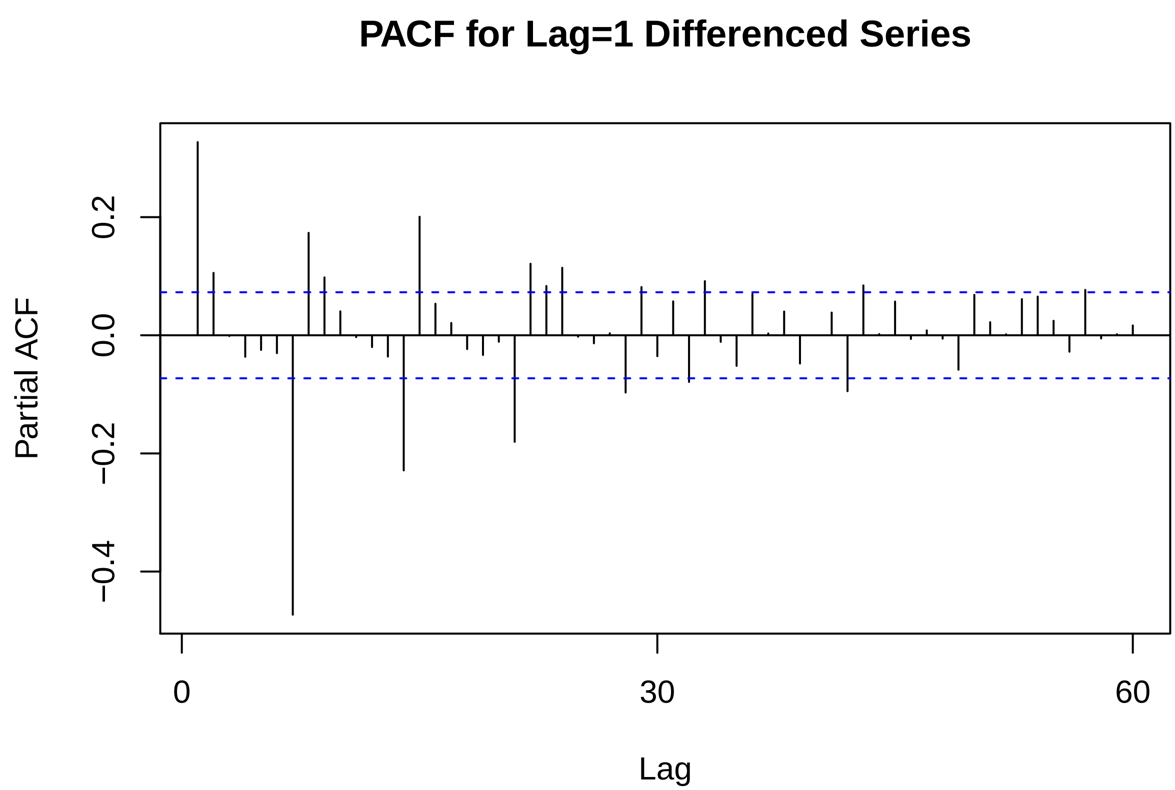
Note that our p-value is now much much lower – we can say with statistical significance that the time series is now stationary.

Let's now take our differenced time series and look at it in the context of our ACF and PACF plots as before:

```
Acf( Diffed.1 , main='ACF for Lag=1 Differenced Series')
```




```r
Pacf(Diffed.1 , main='PACF for Lag=1 Differenced Series')
```



These plots can be used to identify the order of the move aveage used by the model – see for example the large spike at lag = 1 on the PACF plot.

Although we have been identifying our own model parameters along the way, it's also important to note that R can do this for us if we ask it to:

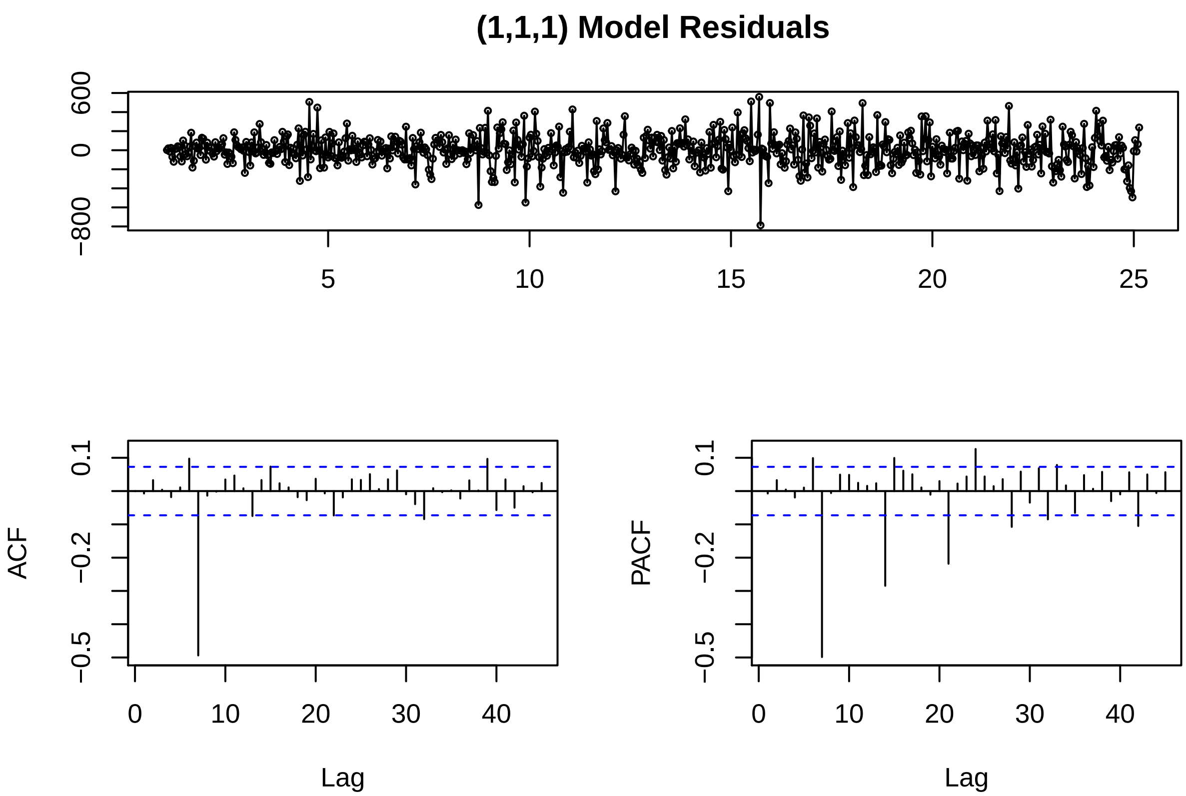
```
Fit.Atuo <- auto.arima( Deasonalized.Counts , seasonal = FALSE )
Fit.Atuo
```

```
## Series: Deasonalized.Counts
## ARIMA(1,1,1)
##
## Coefficients:
##          ar1      ma1
##       0.5510  -0.2496
## s.e.  0.0751   0.0849
##
## sigma^2 estimated as 26180:  log likelihood=-4708.91
## AIC=9423.82   AICc=9423.85   BIC=9437.57
```

Interesting and also comforting: the automated process recommends a (1,1,1) specification for the ARIMA model, which checks with what we identified through our manual analysis above.

Evaluate the ARIMA Model

```
tsdisplay( residuals( Fit.Atuo ) , lag.max = 45 , main = '(1,1,1) Model Residuals' )
```

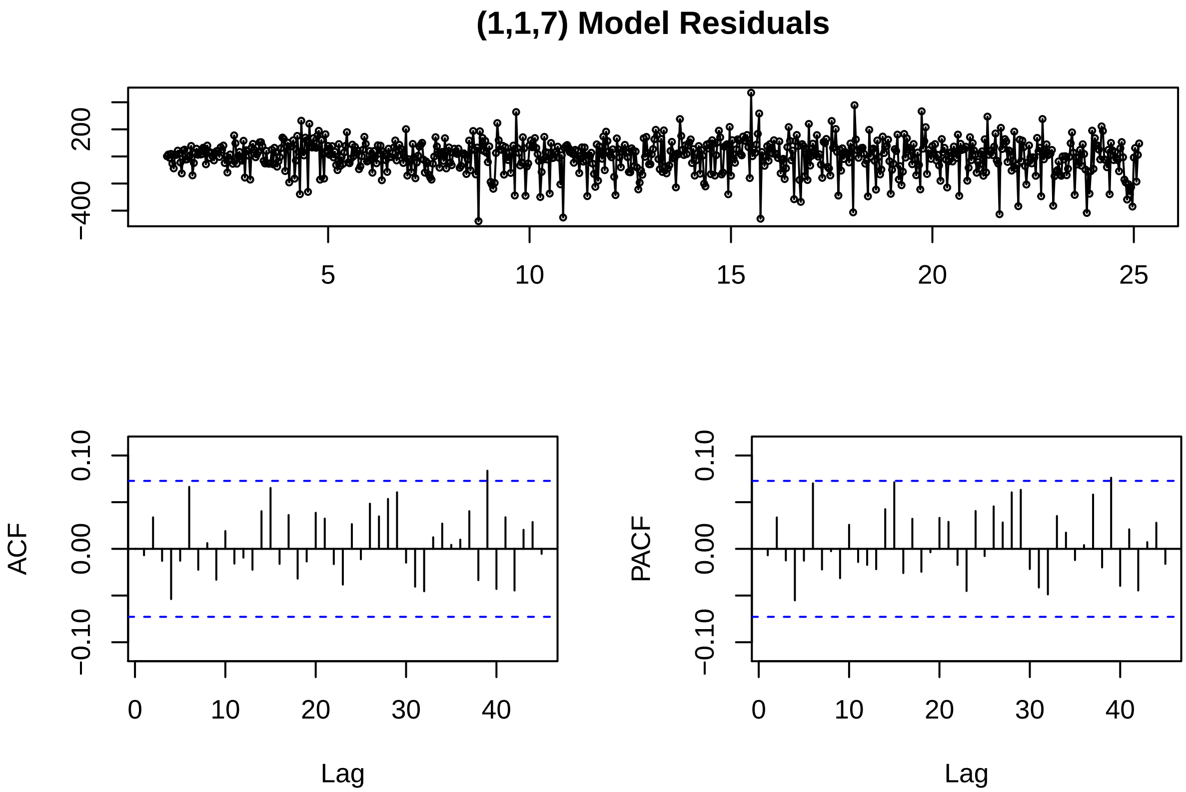


Note the spike in the ACF and PACF plots at lag = 7. This can be a strong indication that our model might be better specified with a different specification. So we can repeat the fitting process.

```
Fit.Manual.1 <- arima( Deasonalized.Counts , order = c(1,1,7) )
Fit.Manual.1
```

```
##
## Call:
## arima(x = Deasonalized.Counts, order = c(1, 1, 7))
##
## Coefficients:
##          ar1     ma1     ma2     ma3     ma4     ma5     ma6      ma7
##       0.2803  0.1465  0.1524  0.1263  0.1225  0.1291  0.1471  -0.8353
## s.e.  0.0478  0.0289  0.0266  0.0261  0.0263  0.0257  0.0265   0.0285
##
## sigma^2 estimated as 14392:  log likelihood = -4503.28,  aic = 9024.56
```

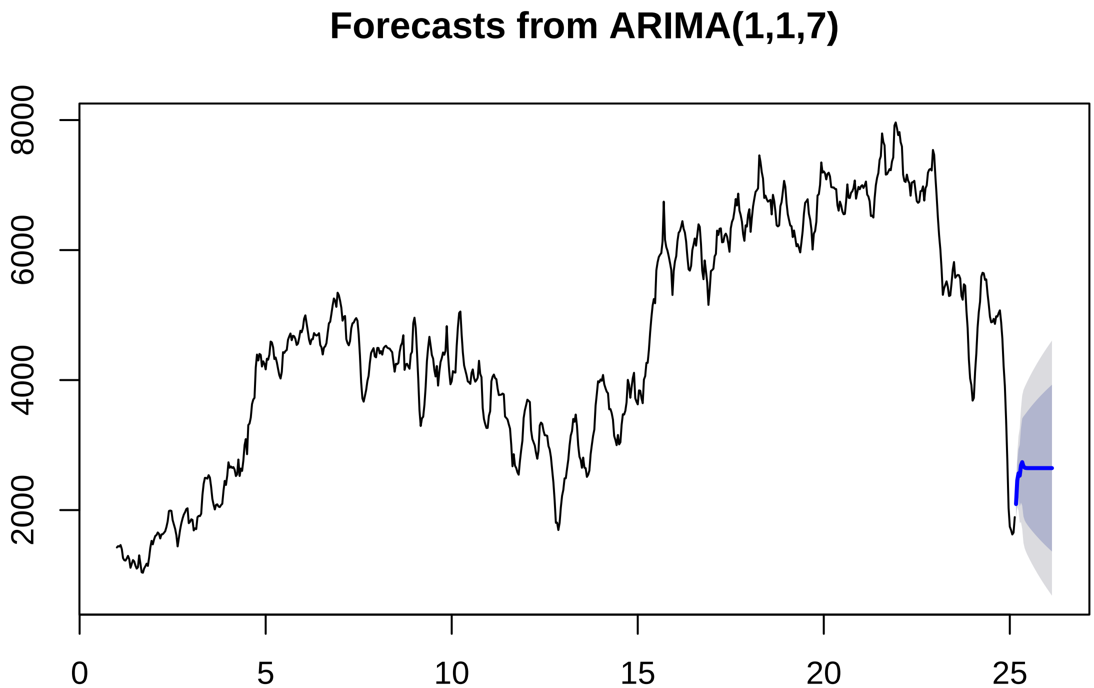
```
tsdisplay( residuals( Fit.Manual.1 ) , lag.max = 45 , main = '(1,1,7) Model Residuals' )
```



This looks much better! With a proper model calibrated, we can now move on to forecasting.

Forecasting

```
Forecast.Manual.1 <- forecast( Fit.Manual.1 , h = 30 )

plot( Forecast.Manual.1 )
```



The fan shows the forecast provided by our ARIMA model. The blue prediction line is relatively flat, despite the variation in the historical data. The forecast is clearly a naive model, but the example illustrates the process and the inspections necessary to calibrate an ARIMA model.

Improvements

Now, without even backtesting it, we can probably guess that our prediction in the previous section was not a very good one. The manner in which it becomes immediately flat, given all the variation in the historical data, seems very suspicious.

If we recall the manner in which we calibrated this model, however, it might make the reason for this a little more obvious. Recall that we used the deseasonalized counts in order to calibrate the model . . . in other words, the model that we calibrated and then used to predict with does not "see" any of the seasonality in the actual data.

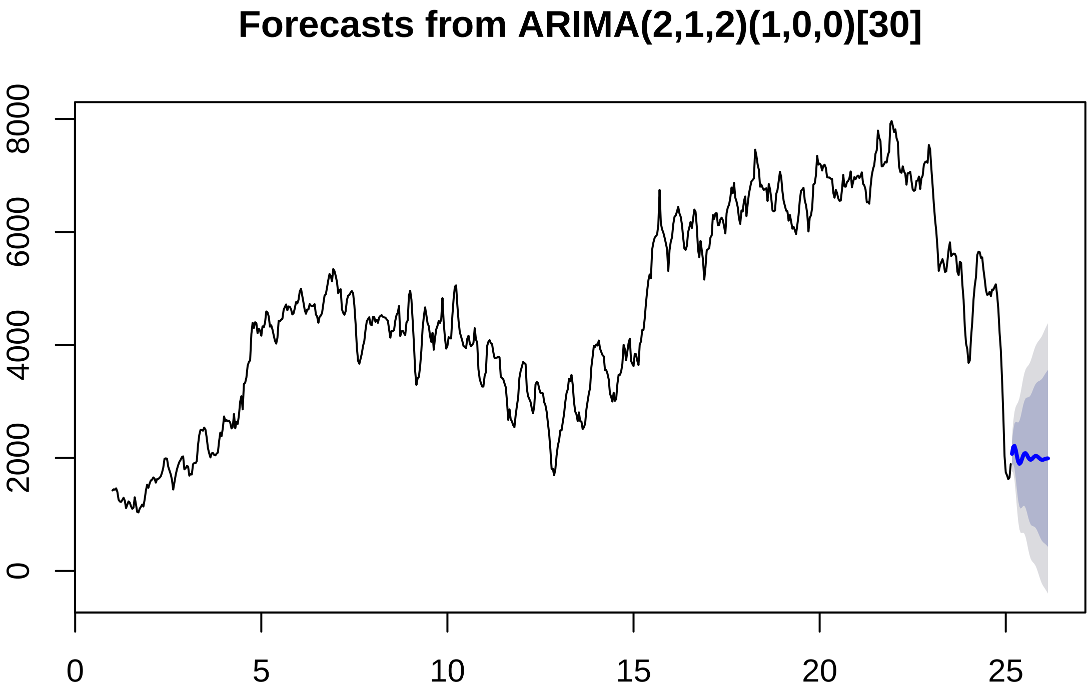
We could do this in two ways: first, we could go back to the original seasonal data. Or, alternatively (and what we will do here) is to allow for seasonality in the ARIMA model:

```
New.Fit.With.Seasonality = auto.arima( Deasonalized.Counts , seasonal = TRUE )
New.Fit.With.Seasonality
```

```
## Series: Deasonalized.Counts
## ARIMA(2,1,2)(1,0,0)[30]
##
## Coefficients:
##          ar1      ar2      ma1     ma2    sar1
##       1.3644  -0.8027  -1.2903  0.9146  0.0100
## s.e.  0.0372   0.0347   0.0255  0.0202  0.0388
##
## sigma^2 estimated as 24810:  log likelihood=-4688.59
## AIC=9389.17   AICc=9389.29   BIC=9416.68
```

Here we take the very same deseasonalized data and fit it to a new (2,1,2) model. Now, we could go through the same manual validation process we went through earlier, but for our purposes here we'll just accept this automatically calibrated model and use it for our prediction:

```
Forecast.Auto.Arima = forecast( New.Fit.With.Seasonality , h = 30 )

plot( Forecast.Auto.Arima )
```



Interesting! We see some more seasonal variation in the prediction here, which is probably appropriate given the patterns in the historical time series.

Conclusion

In conclusion, we have very briefly described how to use R in order to fit an ARIMA time series model, including manual investigation of model residuals and ACF/PACF plots. Finally, given a fully calibrated model, we demonstrated how to use an ARIMA model for demand forecasting.