Data Science Capstone: Refrigeration Case Temp Prediction

John Wallace

June 16, 2019

Executive Summary

This report has been created in partial fulfillment of the requirements for the Harvard EdX Data Science Program Certificate Capstone. This report is submitted as the "choose your own"" part of the Capstone. I chose a project which is closely related to my work. My background is in control systems for supermarket (or grocery) refrigeration cases. The goal of this project is to determine if a predictive model can be designed that can be used to detect issues in a refrigeration control system using only a temperature sensor which provides time series data that can be analyzed. Background information and details of the modeling algorithm chosen and the steps to develop the algorithm are noted below. An ARIMA (Auto-Regressive-Moving Average) model was developed which was able to forecast accurately a few time samples ahead which can then be compared to actual values and a determination made as to whether the system has a problem and an alarm triggered.

This report consists of 4 sections:

1) Introduction: Provides background of the project and details the steps used to create the model as well as information about the dataset
2) Methods & Analysis: Provides details about each step to create the model
3) Results: Lists the results of the model prediction and comparison to actual data
4) Conclusions: Commentary on the results as well as potential follow on work to improve the model

Introduction

The primary goal of a refrigeration case is to keep the food that is in the case at the proper temperatures. There are a variety of control systems that are used to insure that the food is kept at the proper temperature. Generally, a temperature sensor is placed in the case which provides a continuous reading of the temperature. The sensors output (the temperature) is then fed to an electronic control system that compares the sensor value to a desired setting and takes various actions (i.e. turning a compressor or control valve on or off) to keep the temperature of the case within the desired range. While the control systems work well, occasionally there are problems that occur that cause the system to not be able to maintain the desired temperature. The control systems generally evaluate the temperature sensor data to determine if there is a problem but in most cases, the evaluation is a static comparison to a pre-programmed value and is not adjusted to present conditions. The result is that sometimes the control system triggers an alarm condition too late (i.e. the problem has already occurred and the food must be thrown away because the temperature in the case was too high) or too soon (i.e. transients occur which cause the sensor reading to be out of range but return to normal after a short time). The goal of this project is to determine if it is feasible to develop a prediction algorithm which can predict what value the temperature should be in the near future (look ahead) based on past samples. If this algorithm could be developed it would be possible to predict the values of the temperature sensor say 5 to 10 minutes ahead and compare the actual to the predicted. If the actual was outside of a range around the predicted value, this could significant a problem with the refrigeration system and maintenance personnel alerted to correct the problem quickly.

The temperature sensor data used in this project is from a functioning control system. The temperature sensor value is sampled every 3 minutes and stored in a csv file. Since the data is a time series, time series techniques were investigated to create a predict model. Initially, a Long Short Time Memory (LSTM) Recurrent Neural Network was developed based on the Keras package (see reference) which utilized the tensorflow framework. Unfortunately, the LSTM model didn't appear to have good predictive capabilities as the output of the model was just a shifted version of the input samples. After some additional research, an Auto Regressive Integrated Moeing Average (ARIMA) model was developed. ARIMA models utilize information contain in the past samples of the time series data to predict future values. More information on ARIMA models can be found in the references. The ARIMA tutorial written by Ruslana Dalinina (January 10, 2017) on the datascience.com website was used as a guide to understand key parts of the ARIMA model as well as the steps involved to develop and test a model. Additionally, Time Series Analysis Using ARIMA Model In R also contains information about using R to create ARIMA models.

The following steps were used to create the model:

Step 1) The dataset is loaded from a csv file

Step 2) Preliminary data exploration is performed.
Note that the data is time series data and represents the temperature of the refrigeration case. sampled at approximately 3 minute intervals. The data is checked for mising values other issues and any required adjustments made. Visualization techniques are used to inspect the data.

Step 3) The data is cleansed. Data is checked for missing values and other issues and any required adjustments made. Moving averages are calculated to help smooth the data

Step 4) Decompose the data. Time series analysis relies heavily on detecting seasonality, trend and cycle We will look for these patterns in the data as that will help inform model development

Step 5) Create the model. The forecast package contains a function to create an ARIMA model. This package will be used to create an ARIMA model

Step 6) Use the model to forecast, check the results and calculate the rmse.

Methods & Analysis

Step 1) The dataset is loaded from a csv file.

As noted above, the data used for this project was downloaded from a refrigeration system operating in a supermarket. The data is represented as a column within a csv file with 1 row per sample. Samples are spaced at 3 minute intervals so there are 20 samples per hour.

First, required libraries are loaded

```
##############################################################
# Load the required libraries
##############################################################
library(tibble)
library(readr)

library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.5.2

## -- Attaching packages ------------------------------------------------------------------------- tidy

## v ggplot2 3.1.0     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.3.1
## v purrr   0.2.5     v forcats 0.3.0

## -- Conflicts ----------------------------------------------------------------------------- tidyverse_
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(readxl)
library (dplyr)
library (xts)
```

```
## Warning: package 'xts' was built under R version 3.5.3

## Loading required package: zoo

## Warning: package 'zoo' was built under R version 3.5.3

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

##
## Attaching package: 'xts'

## The following objects are masked from 'package:dplyr':
##
##     first, last
```

```r
library('ggplot2')
library('forecast')
```

```
## Warning: package 'forecast' was built under R version 3.5.3
```

```r
library('tseries')
```

```
## Warning: package 'tseries' was built under R version 3.5.3
```

```r
library(readr)
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.5.2

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

As noted above, the data used for this project was downloaded from a refrigeration system operating in a supermarket. The data is represented as a column within a csv file with 1 row per sample. Samples are spaced at 3 minute intervals so there are 20 samples per hour.

```r
# Note that the data file must be in the working directory
# (or add path information to the filename in the read_csv call)

CaseTempData <- read_csv("CapstoneTempData.csv")
```

```
## Parsed with column specification:
## cols(
##   CaseTemp = col_double()
## )
```

Step 2) Preliminary data exploration is performed.

Note that the data is time series data and represents the temperature of the refrigeration case sampled at approximately 3 minute intervals. The data is checked for missing values other issues and any required adjustments made. Visualization techniques are used to inspect the data.

As noted in the tutorial referenced in the introduction, An ARIMA model is specified by three order parameters (p, d, q). The auto-regressive component (AR(p)) refers to the use of past values in the regression equation for the series. p specifies the number of lags to be used. The d term represents how much differencing to use in the integral component. Differencing is performed by subtracting the current from previous values of the time series d times. This is important when a time series is not stationary (i.e there is a time based trend in the data). The moving average (MA(q)) component represent the combination of the previous error terms. The q determines the number of error terms to include. Data exploration is important to understand how the p, d and q terms should be set based on the characteristics of the data.

Look at the basic structure of the data and a few key statistics first.

```
# Look at the structure of the data
str(CaseTempData)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    480 obs. of  1 variable:
##  $ CaseTemp: num  33.4 32.4 32.7 34 34.3 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   CaseTemp = col_double()
##   .. )
```

```
# Dataframe with 480 samples; Data Column is called "CaseTemp"

head(CaseTempData)
```

```
## # A tibble: 6 x 1
##   CaseTemp
##      <dbl>
## 1     33.4
## 2     32.4
## 3     32.7
## 4     34.0
## 5     34.3
## 6     32.8
```

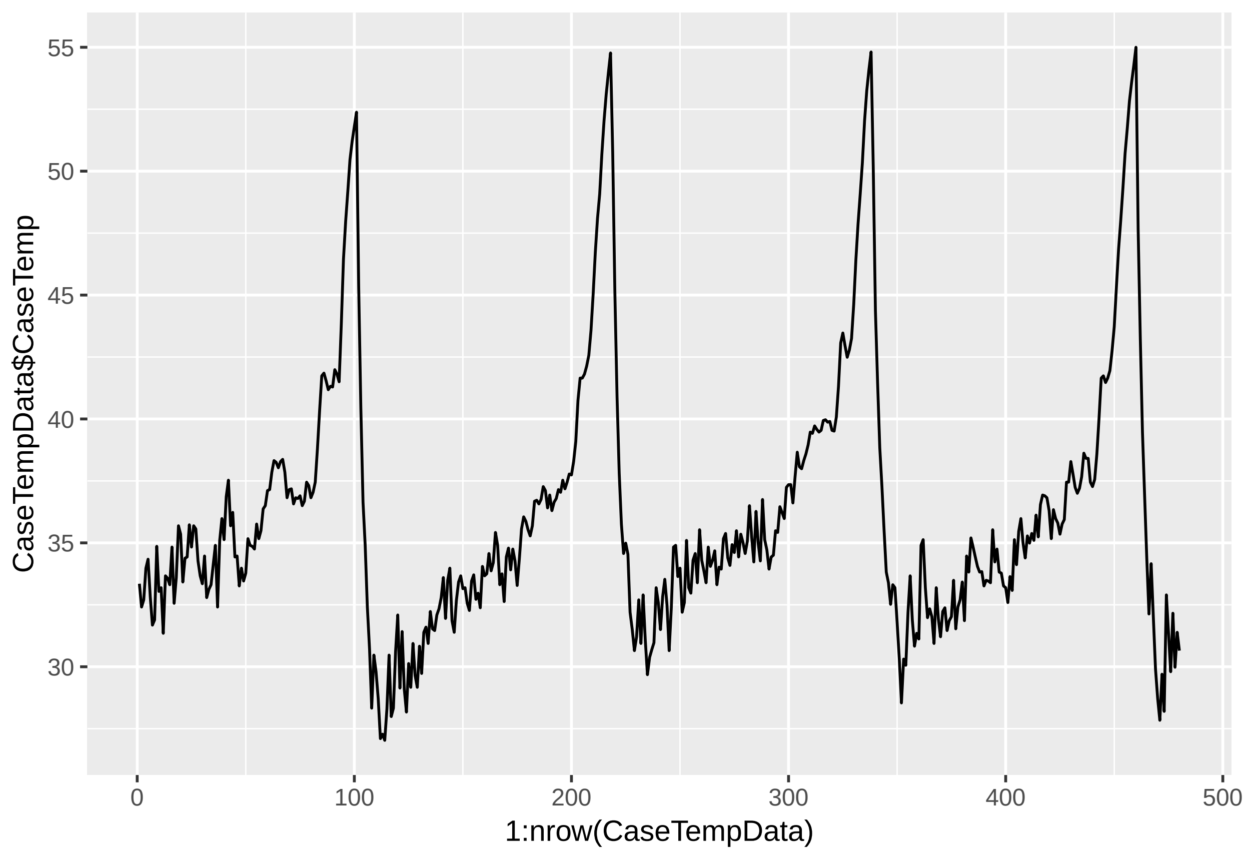
```
# Data is type dbl and have 1 decimal point (Note this data represents degree's fahrenheit)

# Look at the mean and standard deviation of the data
mean (CaseTempData$CaseTemp)
```

```
## [1] 36.24306
```

```
sd (CaseTempData$CaseTemp)
```

```
## [1] 5.336689
```

```
# Mean is 36.3 with a sd of 5.3

# Plot the raw data to see if there are any patterns
ggplot(CaseTempData, aes(x = 1:nrow(CaseTempData), y = CaseTempData$CaseTemp)) + geom_line()
```



```
# The plot shows that the data is cyclic with a repeating pattern appx every 120 samples
#   we will need to consider this when we create the ARIMA model
```

There are 480 samples in the data file representing 24 hours of data (480 samples x 1 hour per 20 samples). The mean of the data is 36.3 (degrees Fahrenheit) with a standard deviation of 5.3 degrees The plot shows that the data is cyclic with a repeating pattern appx every 120 samples (4 hours) Note that this repeating pattern is expected in a refrigeration system. A refrigeration case under normal operation will have ice buildup that must be melted periodically (called defrosting) for the system to function normally. During a defrost, the refrigeration is turned off for the case and the ice is allowed to melt. The temperature in the case will slowly rise during this time. At the conclusion of the defrost, the refrigeration is turned back on and the case temperature cools back down to a normal range.

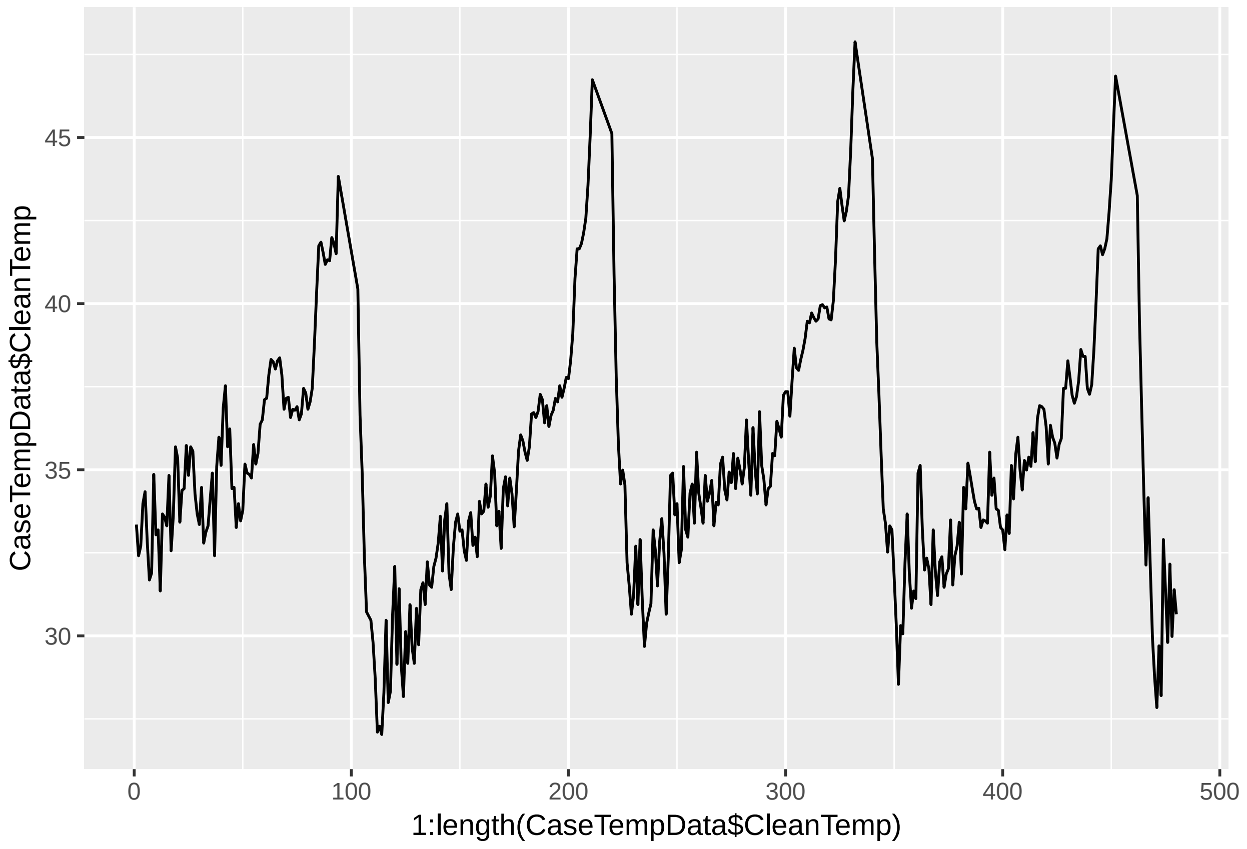
Step 3) The data is cleansed. Data is checked for missing values and other issues and any required adjustments made. Moving averages are calculated to help smooth the data

The function tsclean is used to help cleanse the data. This function estimates missing values and outlier replacements. Plot the data after it is clean.

```
# Create a "clean" data set by creating a new data column and add to the dataframe.  This will
#   remove any outliers which can cause issues when creating the mdoel.  Use the tsclean() function
#   that is part of the forecast package which identifies and replaces outliers via smoothing and
#   decomposition

Temp_ts <- ts(CaseTempData$CaseTemp)

CaseTempData$CleanTemp <- tsclean(Temp_ts)
```

```r
# And plot the Clean data
ggplot(CaseTempData$CleanTemp, aes(x = 1:length(CaseTempData$CleanTemp), y = CaseTempData$CleanTemp)) +
```

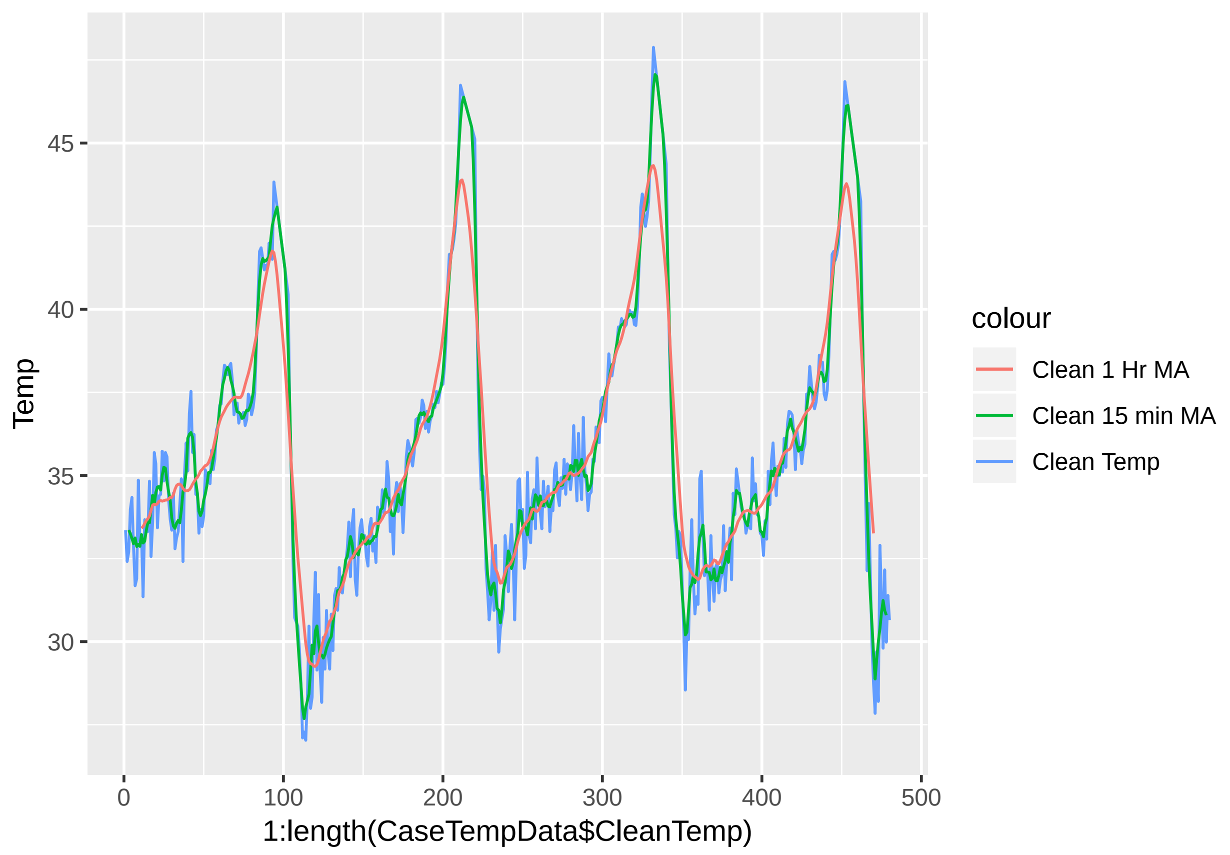
```
## Don't know how to automatically pick scale for object of type ts. Defaulting to continuous.
```



Next, use moving averages to smooth the data. Add a 15 minute (i.e. 5 samples) and 1 hour moving average and plot the resulting data.

```r
# Let's add some smoothing to the data.  Do this by calculating moving averages.
# Since the data is a time series with samples every 3 minutes, lets do a 15 minute moving average
#  and a 1 hour moving average and see the results.  We use the ma function and add to the
#  original dataframe.  (note 5 samples = 15 minutes, 20 samples = 1 hour)
CaseTempData$CleanMA15 <- ma(CaseTempData$CleanTemp, order = 5)
CaseTempData$CleanMA60 <- ma(CaseTempData$CleanTemp, order = 20)

# Now plot the data to see how the smoothing looks

ggplot() +
  geom_line(data = CaseTempData$CleanTemp, aes(x = 1:length(CaseTempData$CleanTemp), y = CaseTempData$Cl
  geom_line(data = CaseTempData$CleanMA15, aes(x = 1:length(CaseTempData$CleanMA15), y = CaseTempData$Cl
  geom_line(data = CaseTempData$CleanMA60, aes(x = 1:length(CaseTempData$CleanMA60), y = CaseTempData$Cl
  ylab('Temp')
```

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

## Warning: Removed 4 rows containing missing values (geom_path).

## Warning: Removed 20 rows containing missing values (geom_path).
```



```
# We can see that the smoothing reomves some of the volativity but leaves the underlying pattern in pla
```

As the plots show, some of the original volitivity and outliers have been removed.

Step 4) Decompose the data. Time series analysis relies heavily on detecting seasonality, trend and cycle We will look for these patterns in the data as that will help inform model development

Note that by developing a deep understanding of the characteristics of the data, we will have a better understanding of how to choose the p, d and d values for the model. Use the adf (augumented Dickey-Fuller test) to check for stationarity.

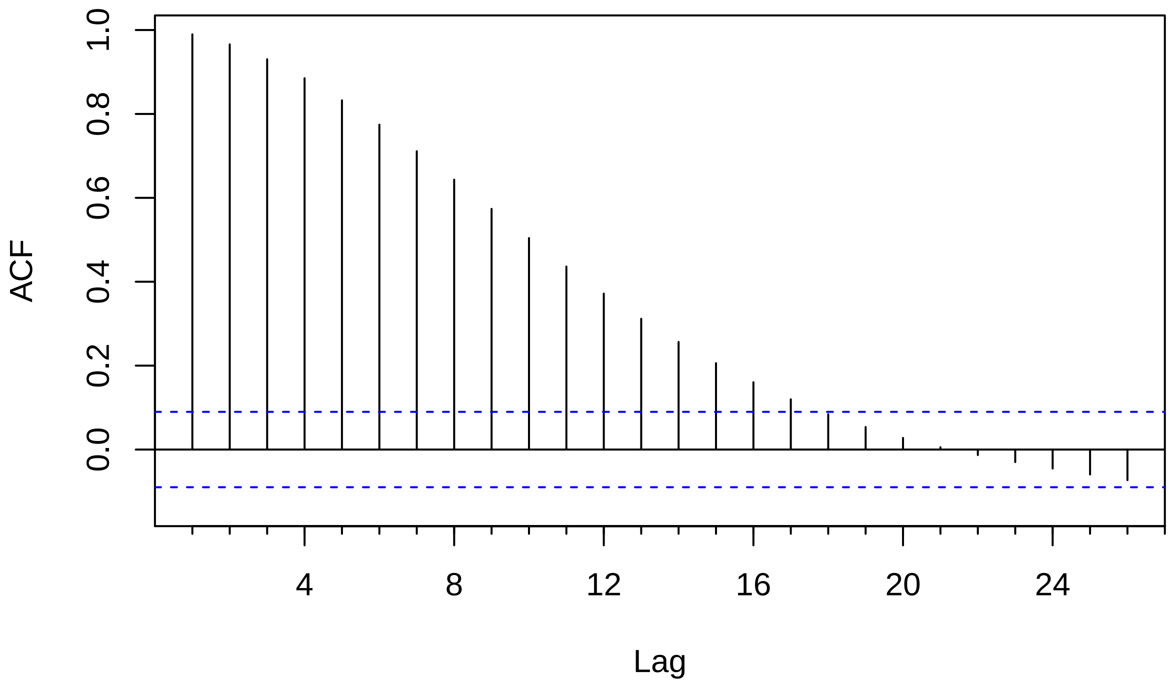
```
# Now check for stationarity   stationarity  means there is no time based trend in the data
# Use the augmented Dickey-Fuller (ADF) statistical test for stationarity.  The null hypothesis
#  assumes the series is non-stationary.
# The ADF procedure tests whether a change in Y can be explained by a lagged value and linear trend.
# If there is a presence os a trend component, the series is non-stationarity  and the null-hypothesis
#  will not be rejected.
CleanMA15Working <- ts(na.omit(CaseTempData$CleanMA15), frequency = 4)
adf.test(CleanMA15Working)
```

```
## Warning in adf.test(CleanMA15Working): p-value smaller than printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  CleanMA15Working
## Dickey-Fuller = -5.2483, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

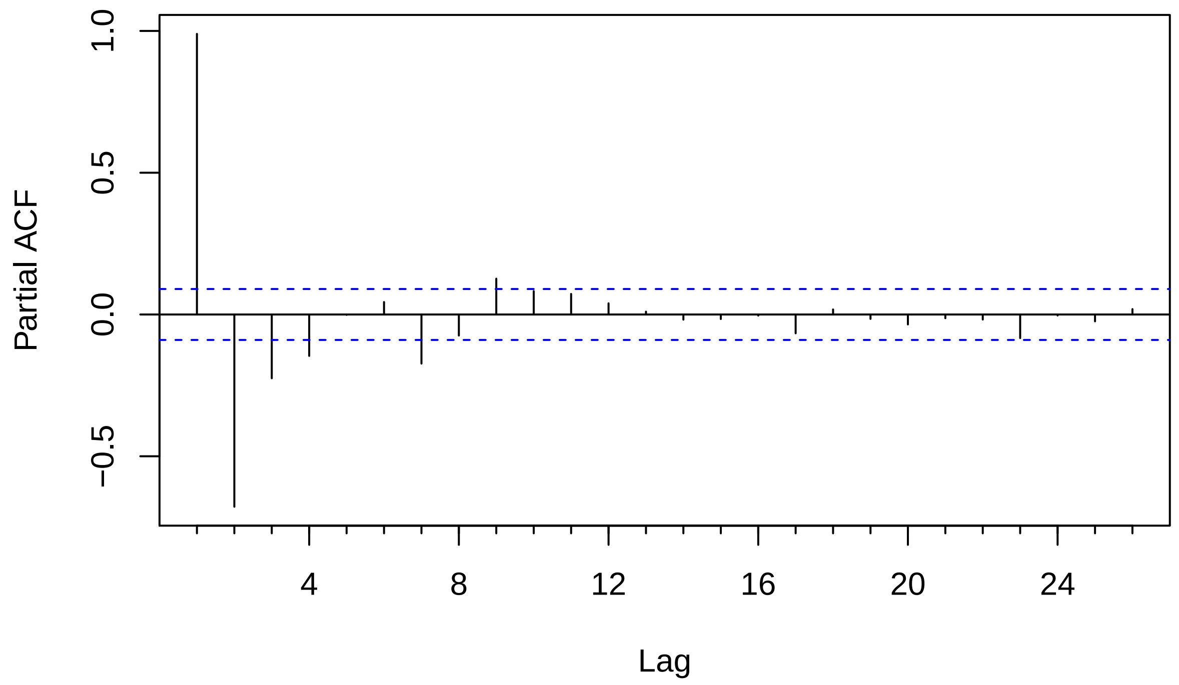
```
# Based on this test, it looks like that the data is stationarity  as the p value is less than 0.05
#  and the Null hypothesis is non-stationarity
```

Based on the initial test, it looks like the data is stationary but we need to do more analysis. We can also look at autocorrelations and a visual plot to see if there are correlations with the lagged data. If so, there might be some trend components. We will use both the Acf (autocorreleration plot) (shows all lags) and Pacf (shows partial lags) functions and visually inspect for correleration.

```
# We can also use an autocorrelation plot(known as ACF) as a visual tool to confirm whether a series is
# The plot can also be useful in choosing the order parameters for the ARIMA model
# Note that if the series is correlated with its lages, then there is some trend or seasonal components
# The Acf & Pacf functions are from the forecast package and can detect auto correlerations

# Look at the autocorrelations with the function Acf.  Note that ACF plots can help in determing the or
#  the MA(q) model.

Acf(CleanMA15Working, main='')
```



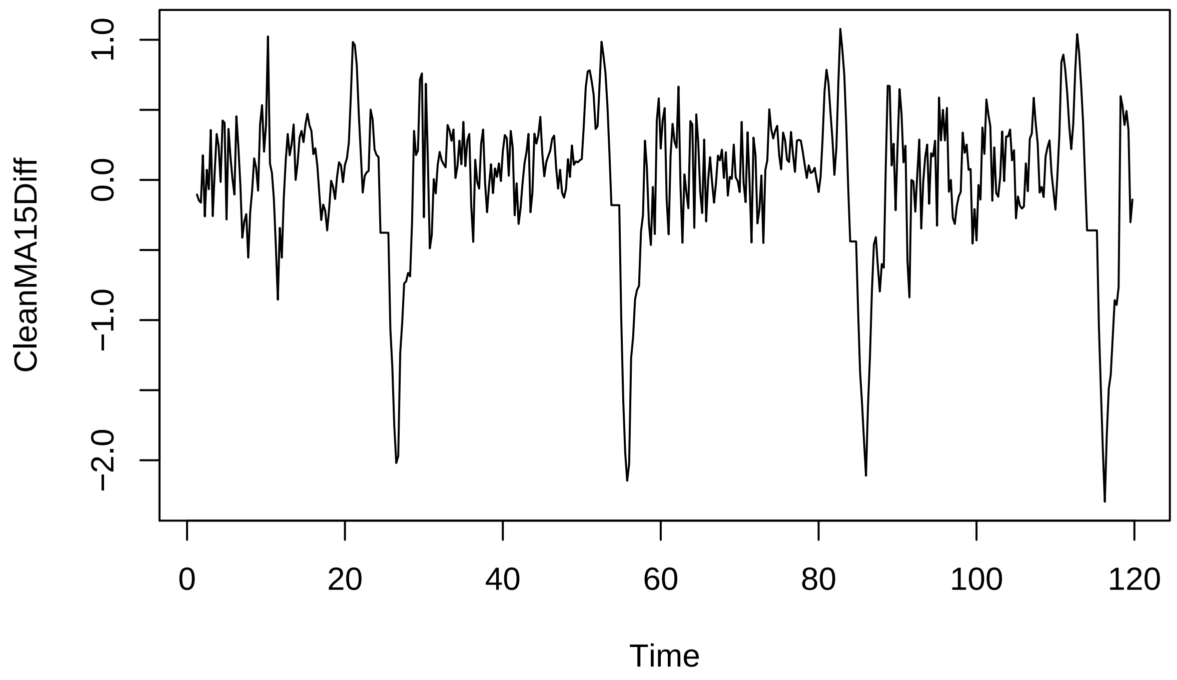
```
# The Acf shows a very clear indication of autocorrlations with many time lags as shown in the plot.
# (Note the blue bards in the plot show the 95% signifiance boundaries)

# Pacf (partical autocorrelation plots) display correlations between a variable and it's lags not expla
#  by previous lags.  (PACF plots are useful when determing the order of the AR(p) model
Pacf(CleanMA15Working, main='')
```



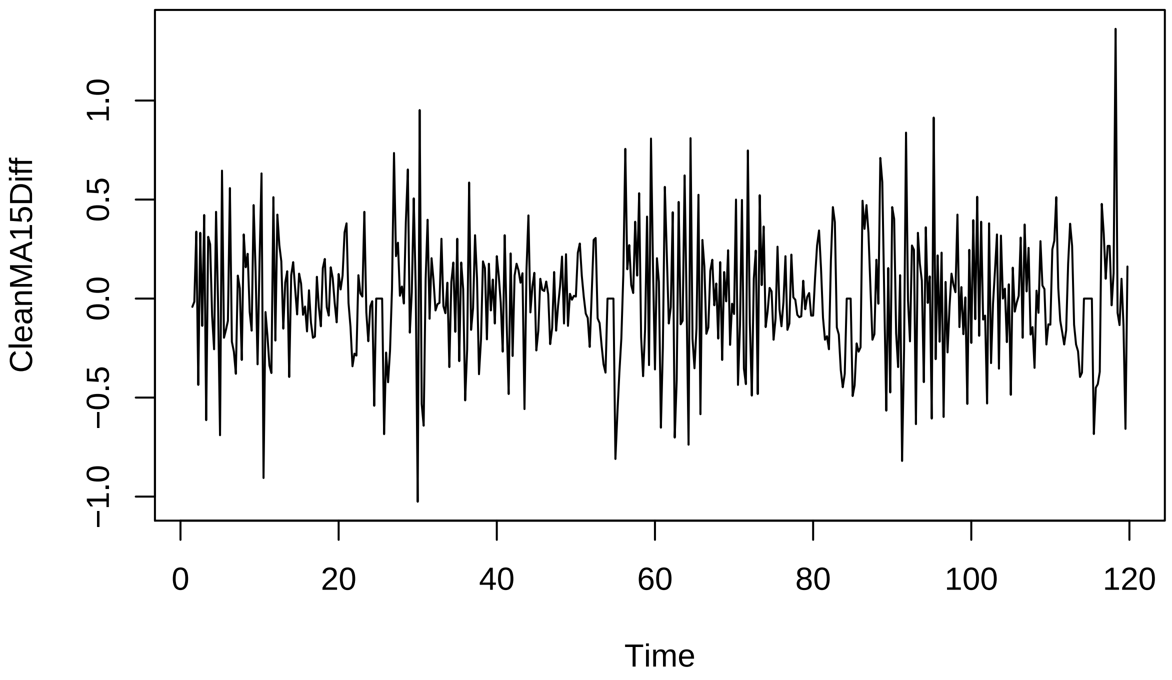
```
# The PACF shows a significant correlation only for the 1-3 lag
```

There appear to be several autocorrelations with previous lags (shown by the spikes at various time lags). Let's use differencing to remove the autocorrelations. Try with a difference or both 1 and 2 samples (d).

```
# Since there does appear to be some autocorrelations with prevous lags, let's look at the adf
#  with a difference of 1
CleanMA15Diff <- diff(CleanMA15Working, differences = 1)
plot(CleanMA15Diff)
```



```r
# From the plot there still appears to be some trends, let's try a difference of 2
CleanMA15Diff <- diff(CleanMA15Working, differences = 2)
plot(CleanMA15Diff)
```



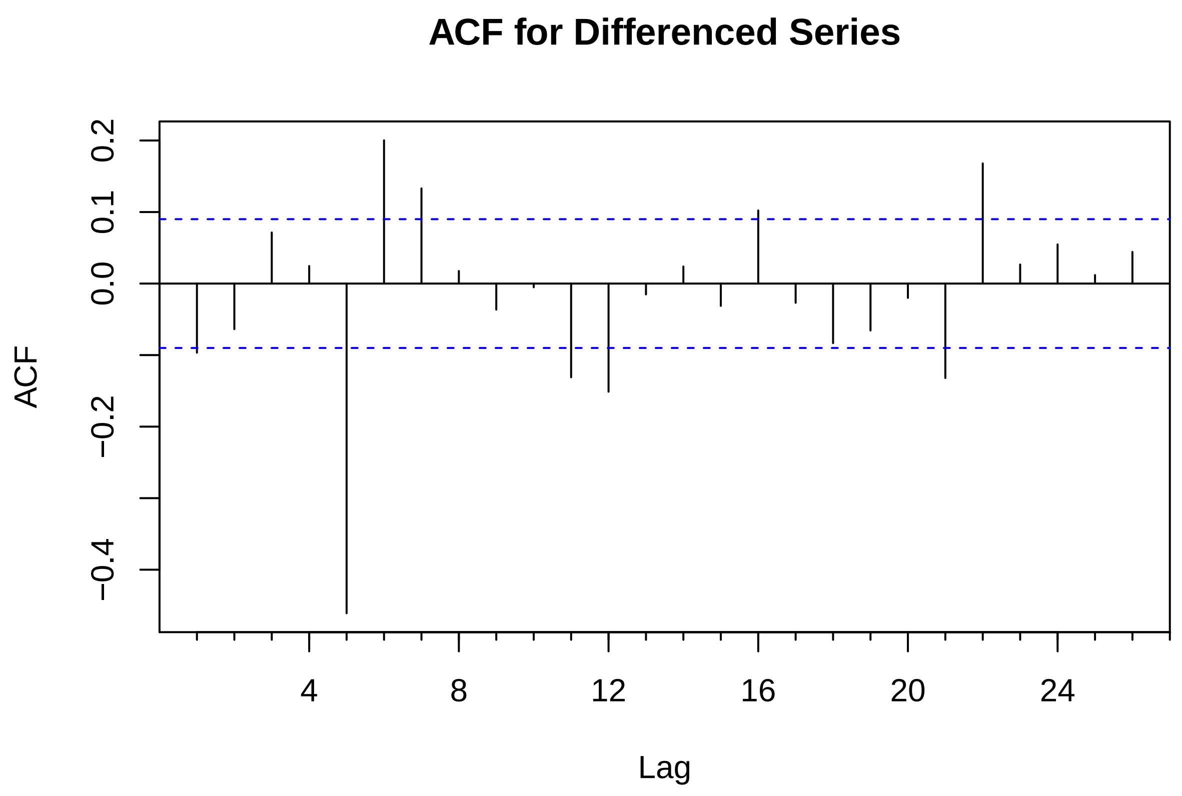
```r
# with a difference of 2, there doesn't appear to be any signficant correlations
# Do an adf test to confirm
adf.test(CleanMA15Diff, alternative = "stationary")
```

```
## Warning in adf.test(CleanMA15Diff, alternative = "stationary"): p-value
## smaller than printed p-value
```

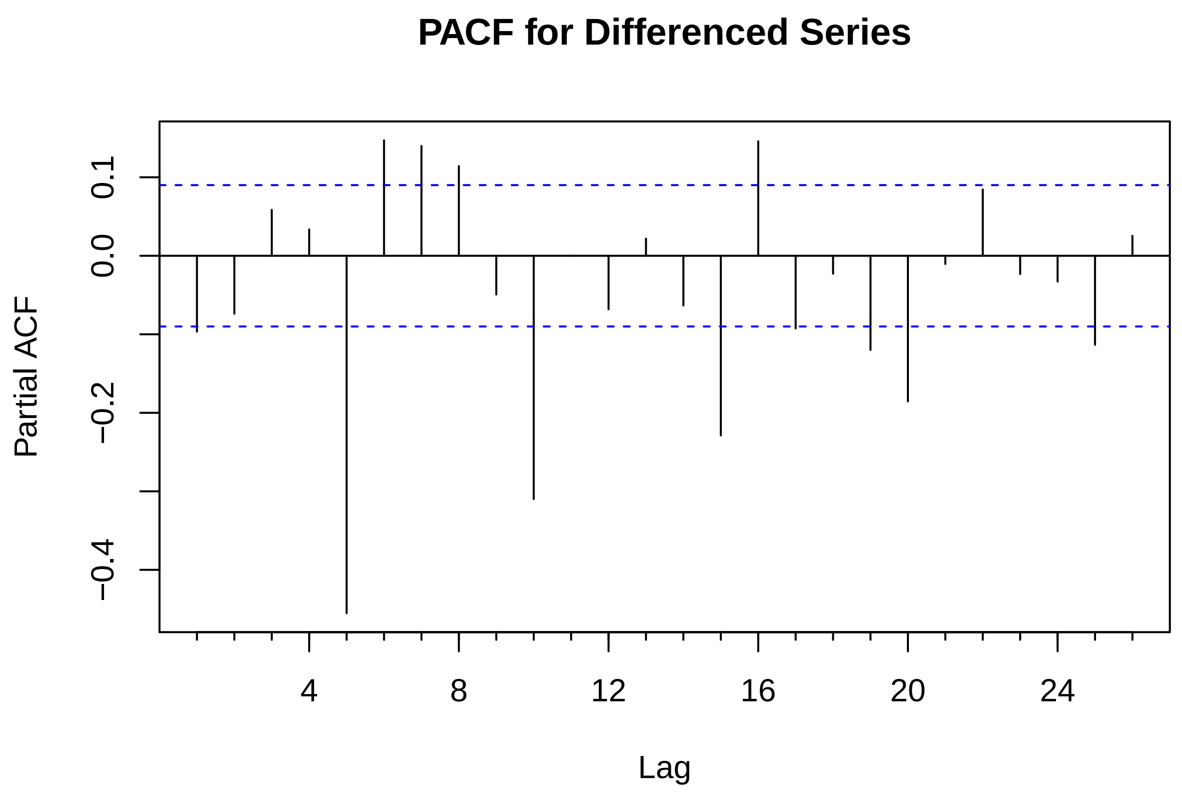
```
##
##  Augmented Dickey-Fuller Test
##
## data:  CleanMA15Diff
## Dickey-Fuller = -6.8158, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

Using a difference of 2 eliminates most of the autocorrelations and we can assume stationarity. We can use Acf and Pacf to understand where significant autocorrelations might be in the original data to help set the p values.

```r
# The adf test on the differenced data rejeccts the null hypotheses of non-stationarity which
# suggests that a differencing of order 2 is sufficient for the model

# Note that spikes at particular lags of the differenced series can help decide the choie of p or q for
Acf(CleanMA15Diff, main="ACF for Differenced Series")
```



```r
# significant auto correlations at lag 5, 6, 7 and beyond
Pacf(CleanMA15Diff, main="PACF for Differenced Series")
```



```
# pacf shows a significant pspike at 5
```

Acf shows correlations at lags of 5, and beyond. The Pacf shows a significant spike at 5 and other spikes at 10 and beyond.

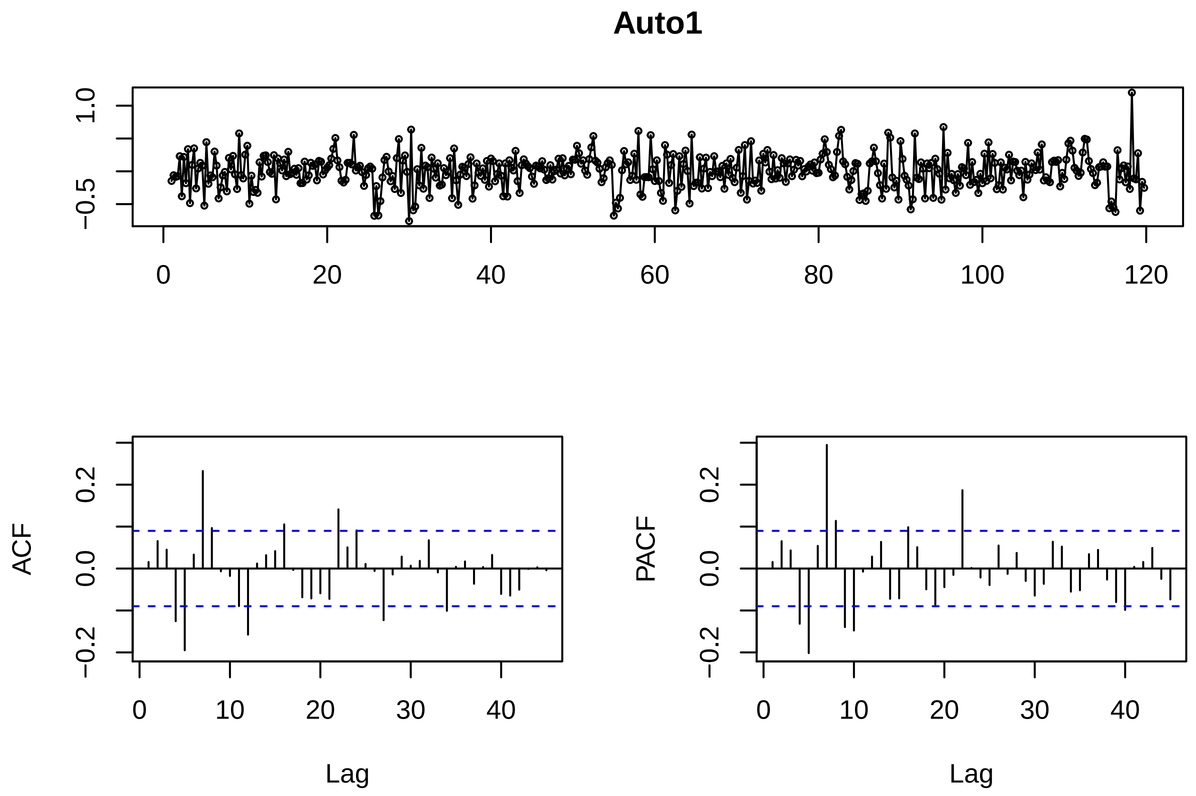
Step 5) Create the model. The forecast package contains a function to create an ARIMA model. This package will be used to create an ARIMA model

Note that the forecast package allows a user to explicitly specify the order of the model or to automatically generate a model based on an analysis and optimization algorithms. As a first try, use the auto.arima function to create the model and look at the results.

```
# First, let's use the auto arima which will automatically generate a set of optimal (p,d,q) values.  N
#  the auto.arima model TBD TBD- Tie in the exploratory work...
# Note that we are using seasonal = FALSE as there is no seasonal pattern in the data so we want to res
#  to non-seasonal models.
fit <- auto.arima(CleanMA15Working, seasonal=FALSE)
fit
```

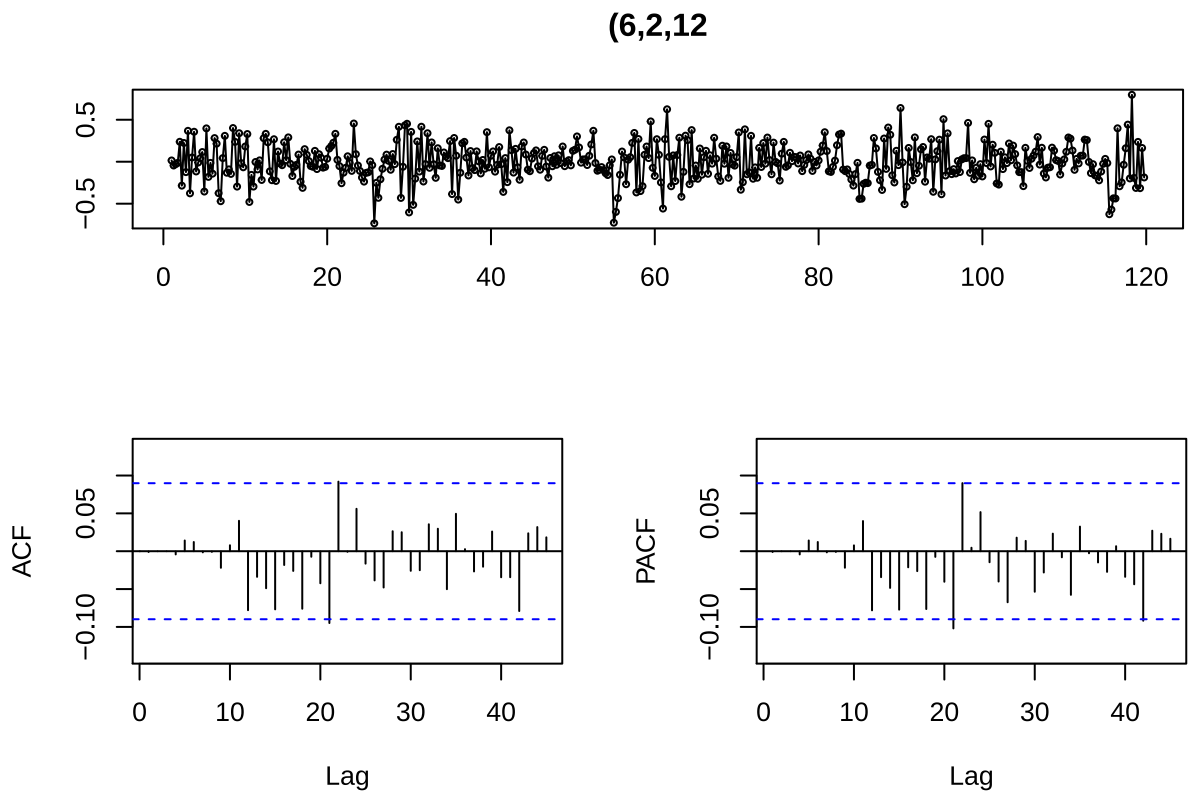
```
## Series: CleanMA15Working
## ARIMA(5,0,3) with non-zero mean
##
## Coefficients:
##          ar1     ar2     ar3      ar4      ar5     ma1     ma2     ma3
##       0.0757  0.9978  0.6247  -0.3235  -0.4453  1.8754  1.4533  0.3052
## s.e.  0.1125  0.0657  0.1041   0.0431   0.0639  0.1216  0.1936  0.1182
##          mean
##       35.7319
## s.e.   0.7570
##
## sigma^2 estimated as 0.06625:  log likelihood=-30.1
## AIC=80.2   AICc=80.67   BIC=121.85
```

```
# Let's see how the model performs.  If the model is good, we should expect no significant

tsdisplay(residuals(fit), lag.max=45, main="Auto1")
```



We do see some autocorrelations present at lag 5, 6 and 7. Refit the model this time specifying the order with the order= parameter in the call. Note that the following code was iterated several times trying different values for the order (p,q,d) and observing the results. The order noted in the code 6,2,12 obtained the best results (least autocorrelations).

```
# We do see autocorrelations present at lag 5, 6 and 7; re-fit the model

fit2<- arima(CleanMA15Working, order=c(6,2,12))
tsdisplay(residuals(fit2), lag.max=45, main="(6,2,12)")
```



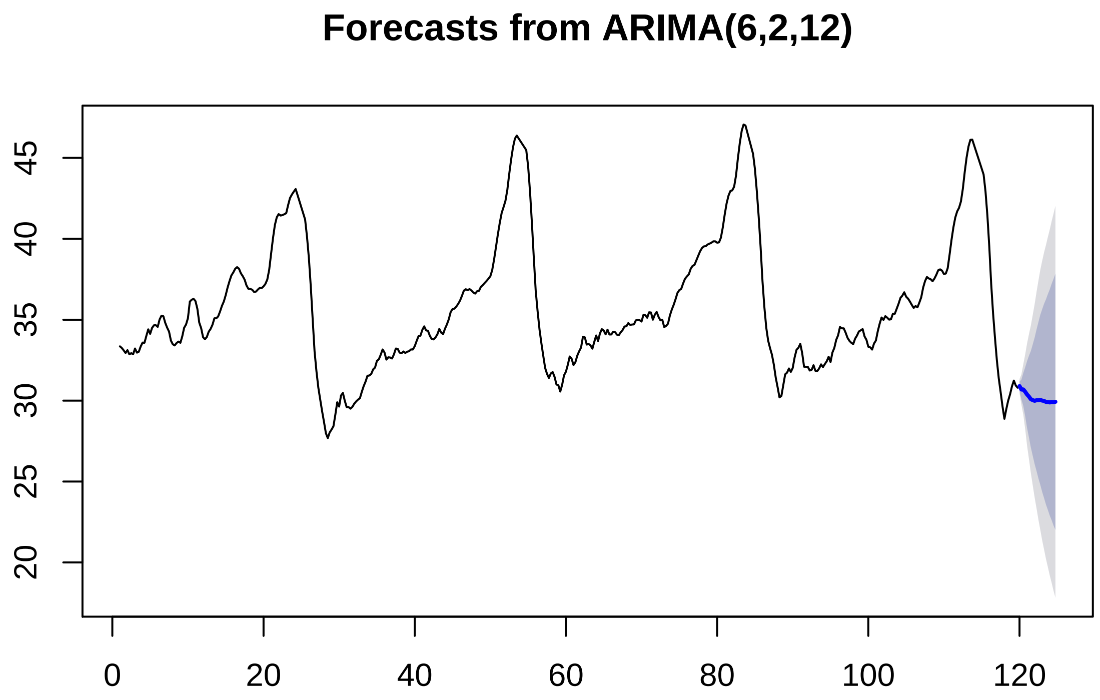
```
fit2
```

```
##
## Call:
## arima(x = CleanMA15Working, order = c(6, 2, 12))
##
## Coefficients:
##          ar1     ar2     ar3     ar4      ar5     ar6      ma1      ma2
##       0.6252  0.2596  -0.606  0.2389  -0.2431  0.2486  -0.7196  -0.1509
## s.e.  0.4798  0.5706   0.306  0.5411   0.4968  0.1947   0.4788   0.6189
##          ma3      ma4      ma5     ma6     ma7      ma8     ma9     ma10
##       0.7248  -0.4356  -0.5822  0.6034  0.1398  -0.7064  0.3344  -0.2811
## s.e.  0.3516   0.5604   0.5840  0.3805  0.6123   0.3464  0.5226   0.5336
##         ma11    ma12
##       0.0353  0.0382
## s.e.  0.2467  0.1005
##
## sigma^2 estimated as 0.04857:  log likelihood = 35.7,  aic = -33.4
```

Step 6) Use the model to forecast, check the results and calculate the rmse.

Use the forecast function to predict future values. Note that you must specify the horizon (i.e. how many samples in the future to forecast) in the forecast function call. Use 1 hour (which is 20 samples in the future) and plot the forecast to visually inspect the forecast.

```
# We can create a forecast using the ARIMA model built by using the forecast function.  To forecast a f
#  specify the forecast horizon (h periods ahead for predictions to be made)
# Let's do a forecast for 1 hour ahead (20 steps)
fcast1Hr <- forecast(fit2, h=20)

# and plot the forecast
# Note that the light blue line shows the forecast with the light gray line showing the 95% interval &
#  dark gray line showing the 80% interval
plot(fcast1Hr)
```



```
# Note that you can also use the predict function to predict
#predict(fit2, n.head = 20)
```

Note that the forecast is shown as the blue line on the plot. The Dark Gray line is the 80% interval and the light gray line is the 95% interval.

Results

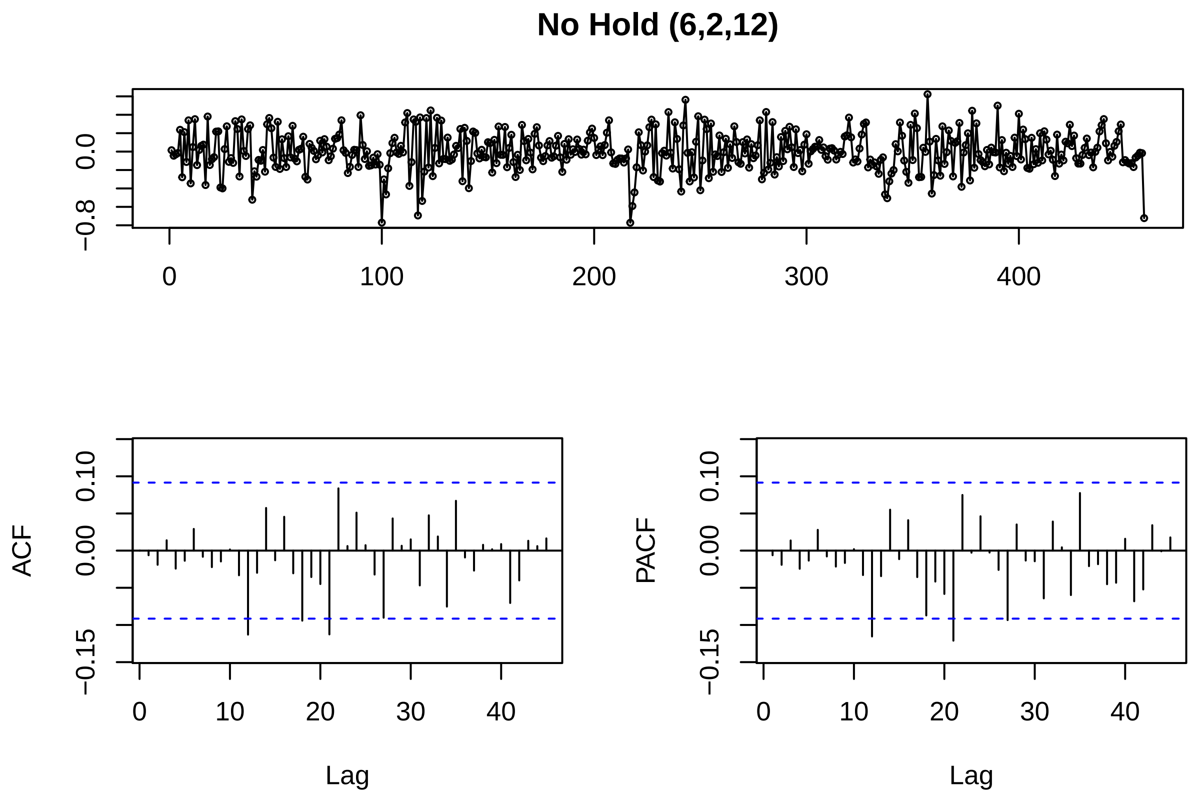
Let's look at how the forecast does when compared to the actual data. Do this by holding out some of the samples from the data that is used to create the model (i.e. the last 20 samples), then use the forecast function to predict those values. The predicted values are then compared to the actual values by overlay the actual & forecast value to we can see visually how the model performs. Note that we must create a new model without the holdout data and then perform the forecast.

```
# In order to see how the model might perform on future samples, we can reserve a portion of the data a
#  "hold out" set, fit the model and then compare the forecast to actual

# 480 total samples; hold out the last 20 samples of data
hold<- window(ts(CleanMA15Working), start=460)
```

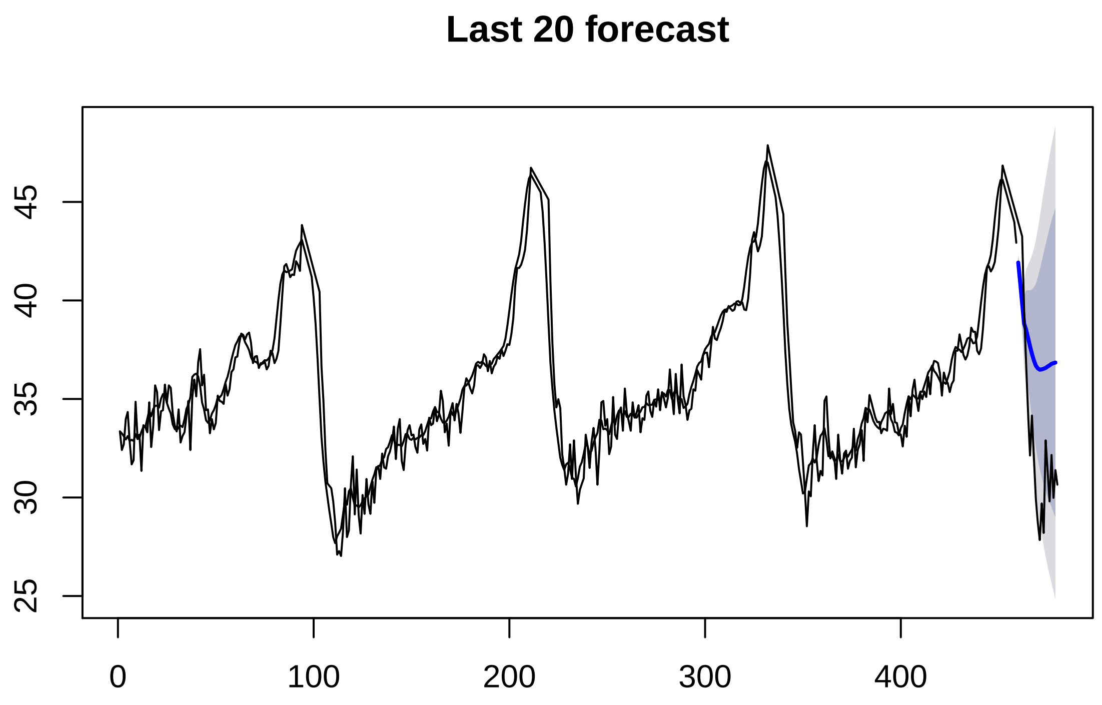
```r
# And pickup the last 20 actual values for later comparison
Last20Act <- as.vector(CaseTempData$CleanTemp[460:479])
# And refit the model based on all samples except what is being held out.  Note the same order is used
fit2NoHoldout <- arima(CleanMA15Working[1:459], order=c(6,2,12))
```

```
## Warning in arima(CleanMA15Working[1:459], order = c(6, 2, 12)): possible
## convergence problem: optim gave code = 1
```

```r
# And look at the results
tsdisplay(residuals(fit2NoHoldout), lag.max=45, main="No Hold (6,2,12)")
```



```r
#fit2NoHoldout

# And forecast the last 20 samples
fcastLast20 <- forecast(fit2NoHoldout, h=20)

plot(fcastLast20, main="Last 20 forecast")

# And add the original "hold out" data to the plot

lines(CaseTempData$CleanTemp)
```



Looking at the plot we can see that a) The actual samples are all within the 95% interval and mostly within the 80% interval and b) The model does a good job predicting a few samples in the future but the accuracy falls off for samples farther into the future.

Finally, let's calculate the RMSE and display the results.

```r
# Let's see what the RMSE is for the next 5, 10 and 20 samples
# Extract the mean value of the forecast
FCast20Mean <- as.vector(fcastLast20$mean)
#Last20Act <- as.vector(CleanMA15Working[460:479])
#Last20Act <- as.vector(CaseTempData$CleanTemp[460:479])
# Error of last 20 look ahead samples
Last20Error <- Last20Act-FCast20Mean
# Error of first 10 look ahead samples
Last10Error <- Last20Act[1:10]- FCast20Mean[1:10]
# Error of first 5 look ahead samples
Last5Error <- Last20Act[1:5] - FCast20Mean[1:5]

RMSE20 <- sqrt(mean(Last20Error^2))
RMSE10 <- sqrt(mean(Last10Error^2))
RMSE5 <- sqrt(mean(Last5Error^2))

# And finally display the results
RMSEResults <- data_frame(Forecast = "All 20 samples", RMSE=RMSE20)
RMSEResults <-bind_rows(RMSEResults, data_frame(Forecast = "First 10", RMSE=RMSE10))
RMSEResults <-bind_rows(RMSEResults, data_frame(Forecast = "First 5", RMSE=RMSE5))
RMSEResults %>% knitr::kable()
```

Forecast	RMSE
All 20 samples	5.434274
First 10	3.893084
First 5	2.296411

Conclusions

ARIMA is an important tool that can be used for time series analysis and prediction. A model was created that can successfully predict the expected values a few samples (in this case 15 minutes +/-) in the future. The initial results show some promise that an ARIMA algorithm can be developed and used as part of a refrigeration system to compare actual to predicted temperature values and detect problems in a more optimized manner.

The following additional steps could be taken in the future to attempt to improve on the model:

1) Use a longer time series (more than 1 day of data) to see if more data can improve the forecasting accuracy.
2) Revisit the order components (p,q,d) specified in the model to attempt to better match the forecast versus actual values 3 Strip the regular "Defrost" cycles from the data before exploration and modeling
3) Revisit an LSTM RNN model

Report by John Wallace