I590 - Time Series Analysis - Code Portfolio

James Provost

April 21, 2019

Importing / Processing Time Series

Applied Statistical Time Series Analysis - has lots of data sets (astsa).

Reference: https://www.stat.pitt.edu/stoffer/tsa4/xChanges.htm for information about astsa.

```
library('astsa')
```

Read in data from a file.

Reference: Manipulating Time Series Data in R with xts & zoo - Chapter 1, Data Camp

```
library(xts)
# Convert either a time series object or zoo object to an xts object
# XTS - eXtensible Time Series - based on a zoo object
my.xts <- as.xts(chicken)

# Write zoo object to file and then read in a previously saved zoo object
write.zoo(my.xts, 'zoo-file.txt')
my.xts2 <- as.xts(read.zoo('zoo-file.txt', FUN = as.yearmon))

# Use regular file reading functions
# read.csv(), read.table(), read.delim() - see R help
```

Create a time series

Specify data, start (can be just a number, or a vector, with the second value referring to frequency), and frequency (number of units in a time period, like quarters or months)

Reference: Introduction to Time Series Analysis - Chapter 1, Data Camp.

```
# Example creates a 60 period time series, from Jan 1980 with 12 periods
# per year.  If frequency was 4, then it would start Q1 1980.
my.ts <- ts(seq(1:60), start = c(1980,1), frequency = 12)

my.ts
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1980   1   2   3   4   5   6   7   8   9  10  11  12
## 1981  13  14  15  16  17  18  19  20  21  22  23  24
## 1982  25  26  27  28  29  30  31  32  33  34  35  36
## 1983  37  38  39  40  41  42  43  44  45  46  47  48
## 1984  49  50  51  52  53  54  55  56  57  58  59  60
```

```
my.ts <- ts(seq(1:60), start = c(1980,1), frequency = 4)

my.ts
```

```
##      Qtr1 Qtr2 Qtr3 Qtr4
## 1980    1    2    3    4
## 1981    5    6    7    8
## 1982    9   10   11   12
```

```
## 1983   13   14   15   16
## 1984   17   18   19   20
## 1985   21   22   23   24
## 1986   25   26   27   28
## 1987   29   30   31   32
## 1988   33   34   35   36
## 1989   37   38   39   40
## 1990   41   42   43   44
## 1991   45   46   47   48
## 1992   49   50   51   52
## 1993   53   54   55   56
## 1994   57   58   59   60
```

Create an xts object.

Reference: Introduction to Time Series Analysis - Chapter 1, Data Camp.

```r
library(xts)
# Build a sample matrix and index vector of dates
my.matrix <- matrix(1:5, ncol = 1, nrow = 5)
my.index <- as.Date(c('2010-01-01','2011-01-01','2012-01-01','2013-01-01','2014-01-01'))
# Create an xts object
my.xts <- xts(my.matrix, order.by = my.index)
my.xts
```

```
##            [,1]
## 2010-01-01    1
## 2011-01-01    2
## 2012-01-01    3
## 2013-01-01    4
## 2014-01-01    5
```

Basic exploration

Basic functions to evaluate aspects of a time series object.

Reference: Introduction to Time Series Analysis, Data Camp.

```r
# Period the time series starts
start(my.ts)
```

```
## [1] 1980    1
```

```r
# Period it ends
end(my.ts)
```

```
## [1] 1994    4
```

```r
# Frequency
frequency(my.ts)
```

```
## [1] 4
```

```r
# The interval from one period to another in terms of time units (1/frequency)
deltat(my.ts)
```

```
## [1] 0.25
```

```r
# Whether it's a time series object
is.ts(my.ts)
```

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

```r
# Vector of indices
time(my.ts)
```

```
##         Qtr1    Qtr2    Qtr3    Qtr4
## 1980 1980.00 1980.25 1980.50 1980.75
## 1981 1981.00 1981.25 1981.50 1981.75
## 1982 1982.00 1982.25 1982.50 1982.75
## 1983 1983.00 1983.25 1983.50 1983.75
## 1984 1984.00 1984.25 1984.50 1984.75
## 1985 1985.00 1985.25 1985.50 1985.75
## 1986 1986.00 1986.25 1986.50 1986.75
## 1987 1987.00 1987.25 1987.50 1987.75
## 1988 1988.00 1988.25 1988.50 1988.75
## 1989 1989.00 1989.25 1989.50 1989.75
## 1990 1990.00 1990.25 1990.50 1990.75
## 1991 1991.00 1991.25 1991.50 1991.75
## 1992 1992.00 1992.25 1992.50 1992.75
## 1993 1993.00 1993.25 1993.50 1993.75
## 1994 1994.00 1994.25 1994.50 1994.75
```

```r
# Position in cycle of the observation
cycle(my.ts)
```

```
##      Qtr1 Qtr2 Qtr3 Qtr4
## 1980    1    2    3    4
## 1981    1    2    3    4
## 1982    1    2    3    4
## 1983    1    2    3    4
## 1984    1    2    3    4
## 1985    1    2    3    4
## 1986    1    2    3    4
## 1987    1    2    3    4
## 1988    1    2    3    4
## 1989    1    2    3    4
## 1990    1    2    3    4
## 1991    1    2    3    4
## 1992    1    2    3    4
## 1993    1    2    3    4
## 1994    1    2    3    4
```

```r
# Pulls out part of a TS between specified start and end periods
window(my.ts, start = c(1980,7), end = c(1980,12))
```

```
##      Qtr1 Qtr2 Qtr3 Qtr4
## 1981              7    8
## 1982    9   10   11   12
```

Data Manipulation

Aggregation nfrequency new number of observations per unit of time; must be a divisor of the frequency of x. FUN aggregation function

Reference: Metcalfe, A. and Cowpertwait, P. (2009). *Introductory Time Series with R*. New York, NY; Spring-Veriag, New York, p. 17

```r
library(astsa)
# Sum by quarter
aggregate(chicken, nfrequency = 4, FUN = sum)
```

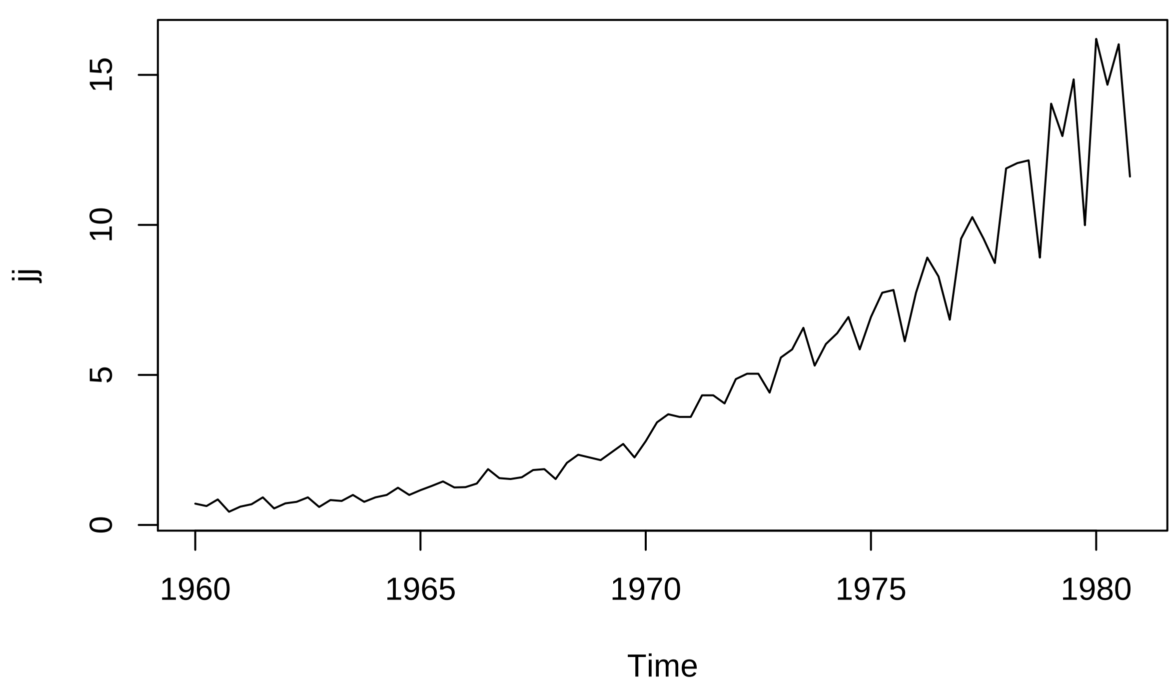
```
## Time Series:
## Start = 2001.58333333333
## End = 2016.33333333333
## Frequency = 4
##  [1] 197.76 190.50 188.15 191.36 191.15 185.66 190.52 198.09 206.30 206.92
## [11] 219.43 237.16 234.26 221.75 221.63 223.52 224.34 213.76 204.99 206.75
## [21] 210.76 209.68 228.24 241.44 242.57 231.87 242.42 257.42 264.36 261.20
## [31] 257.81 263.69 254.50 247.32 252.58 261.46 262.27 255.81 257.55 260.82
## [41] 266.11 269.40 277.21 283.29 286.51 293.41 304.19 316.05 317.76 313.34
## [51] 317.02 332.27 340.20 341.80 342.92 347.86 344.73 338.94 335.21 335.28
```

Stationarity - Stable - Mean remains constant - there is no trend - Correlation from period to period remains constant

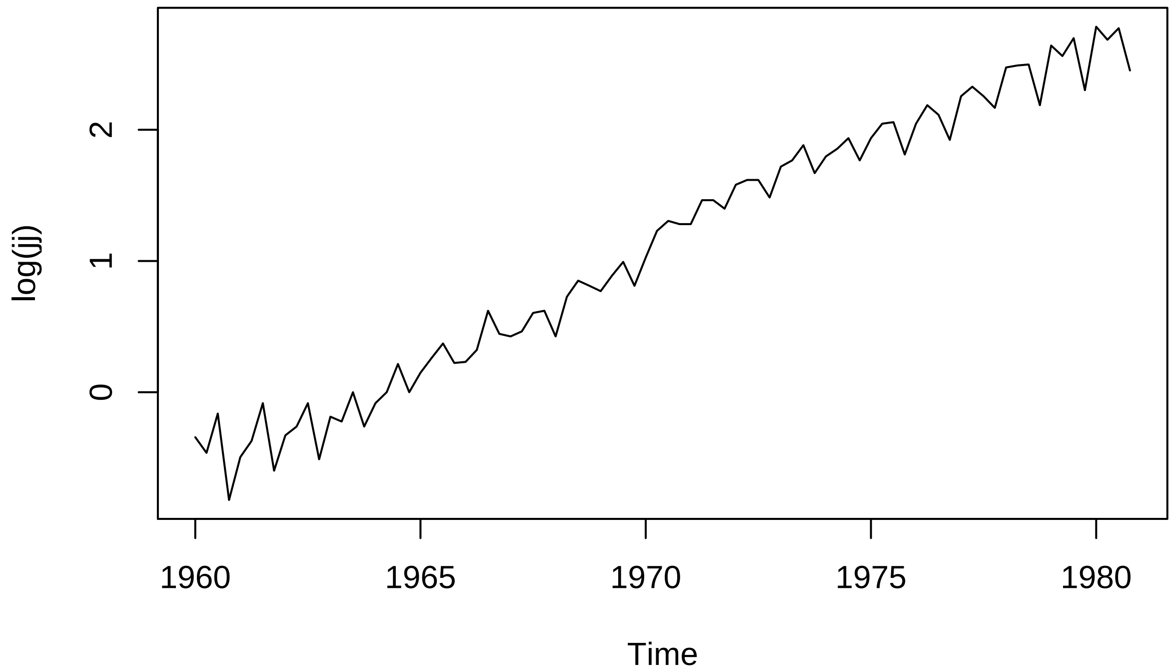
Data Transformation. The diff() function shows the difference from period to period in the time series. It is a way to remove the trend (including Random Walk) from a time series. The log() function will help remove a growth in variability over time (like a multiplicative trend). If you diff(log()), you could make this type of data stationary.

Reference: ARIMA Modeling with R - Chapter 1, Data Camp.

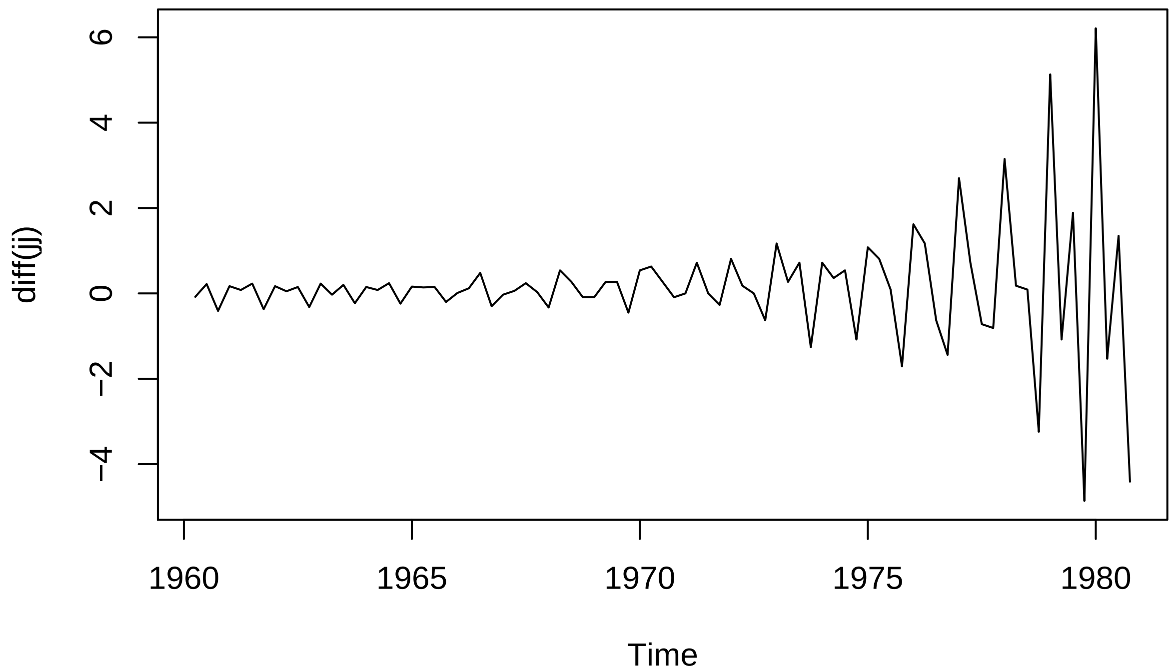
```r
library(astsa)
# Using Johnson & Johnson quarterly earnings, because it has a growing variability in the trend
ts.plot(jj)
```



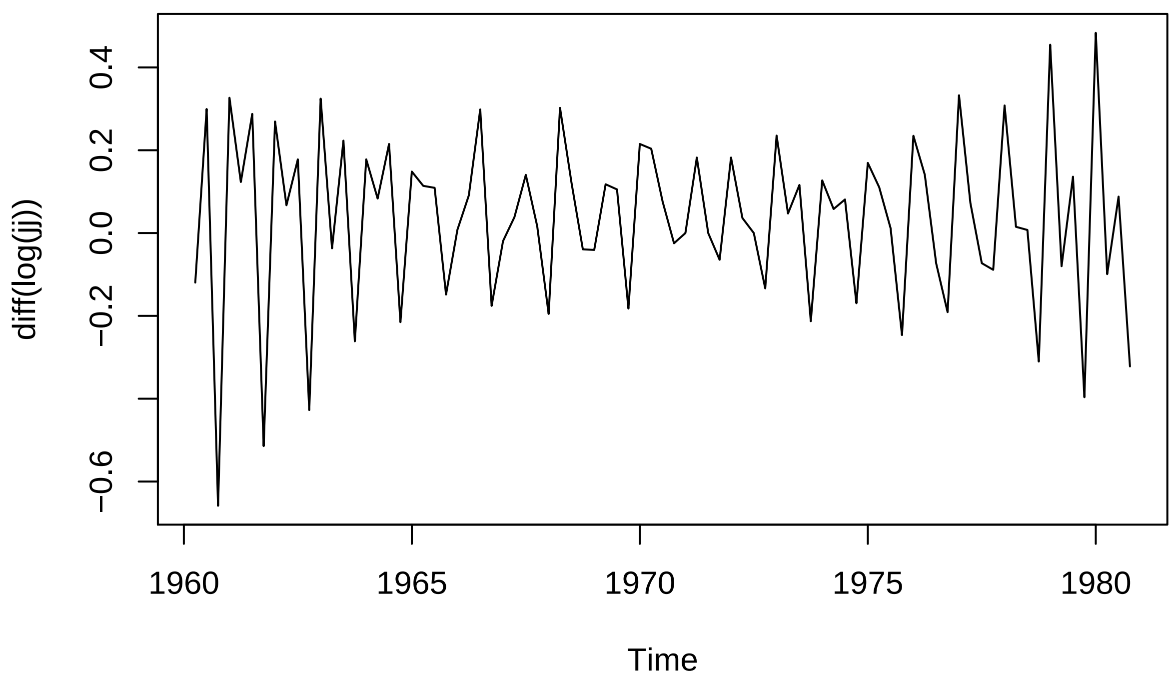
```r
# Take out the upward curve, make it a more linear trend
ts.plot(log(jj))
```




```r
# Take out the trend alone
ts.plot(diff(jj))
```




```
# Take out the curve then take out the trend
ts.plot(diff(log(jj)))
```



Box-Cox Transformations. A transformation for stabilizing variance, usually from -1 (inverse transformation) to 1 (no transformation), with things like natural log and square root in between. BoxCox.lambda() will determine the best lambda value.

Reference: Forecasting Using R - Chapter 4, Data Camp.

```
library(forecast)
BoxCox.lambda(chicken)
```

```
## [1] 1.686301
```

Exploratory Visualization of Time Series

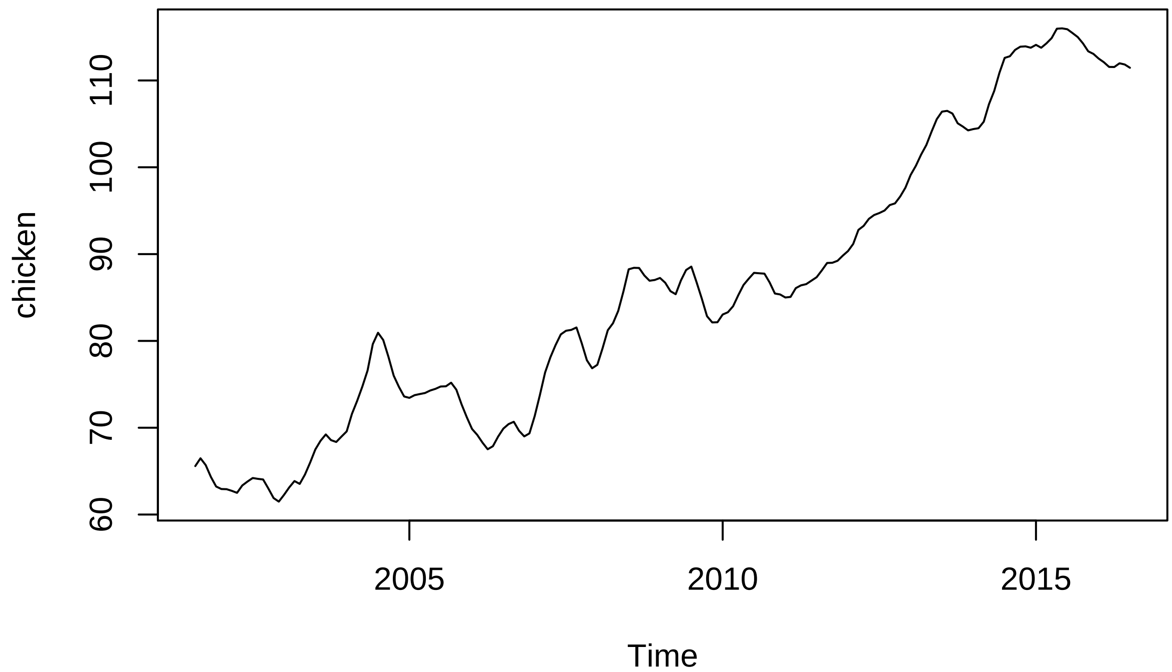
Basic plotting

References: Forecasting Using R, Chapter 1, Data Camp. https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf https://cran.r-project.org/web/packages/ggfortify/vignettes/plot_ts.html

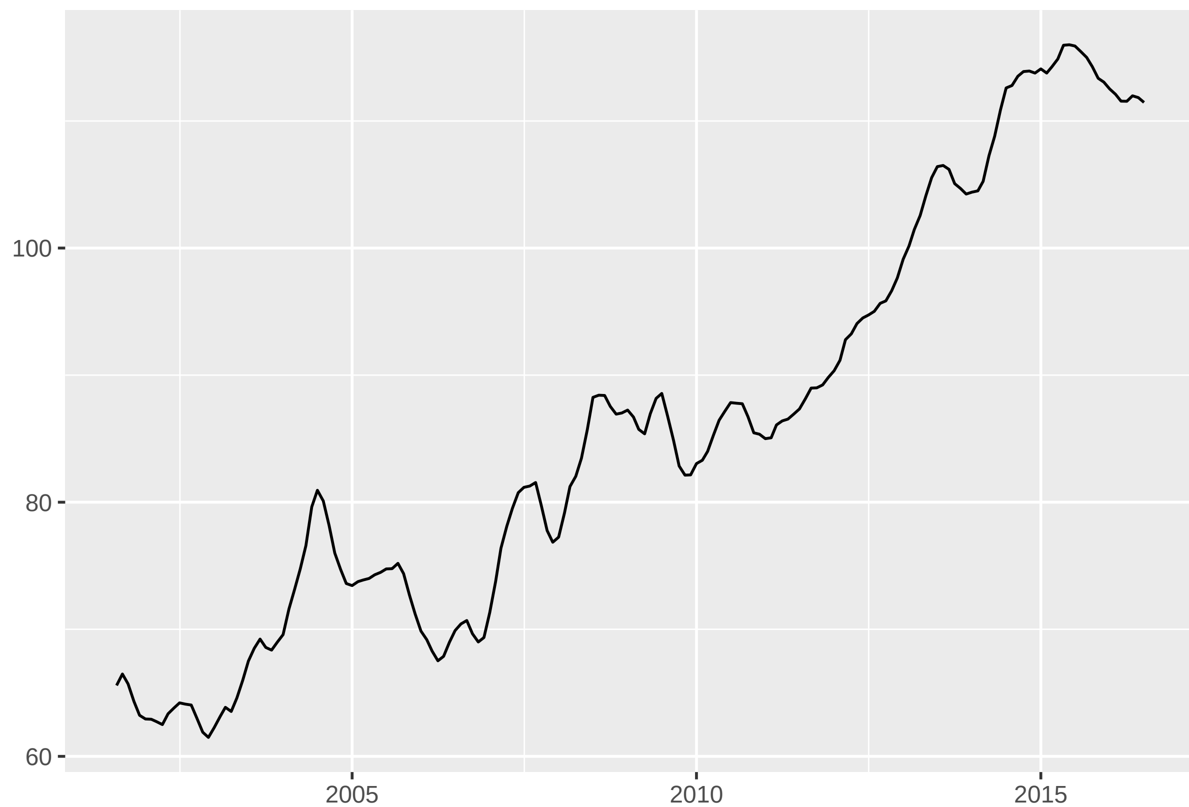
```
library(astsa)
library(ggplot2)

plot(chicken)

ts.plot(chicken)
```




```r
# Library ggfortify is needed for fortify to handle time series objects
library(ggfortify)
autoplot(chicken)
```



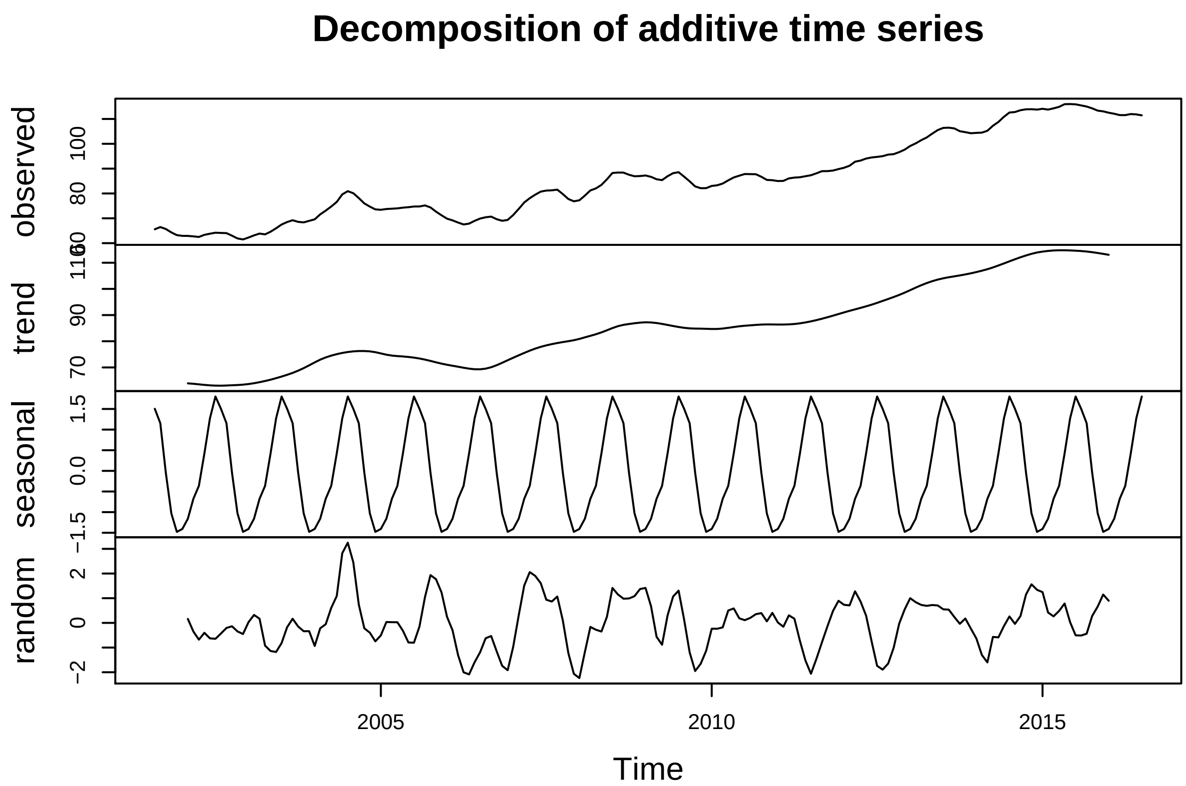
Also refer to decompose(), sarima(), checkresiduals(), acf(), pacf() and acf2() functions in the Time Series Analysis section.

Time Series Analysis

Decompose - Split a TS into components for Trend, Seasonal and Random (Residual)

Reference: Metcalfe, A. and Cowpertwait, P. (2009). *Introductory Time Series with R*. New York, NY; Spring-Veriag, New York, p. 22

```
plot(decompose(chicken))
```

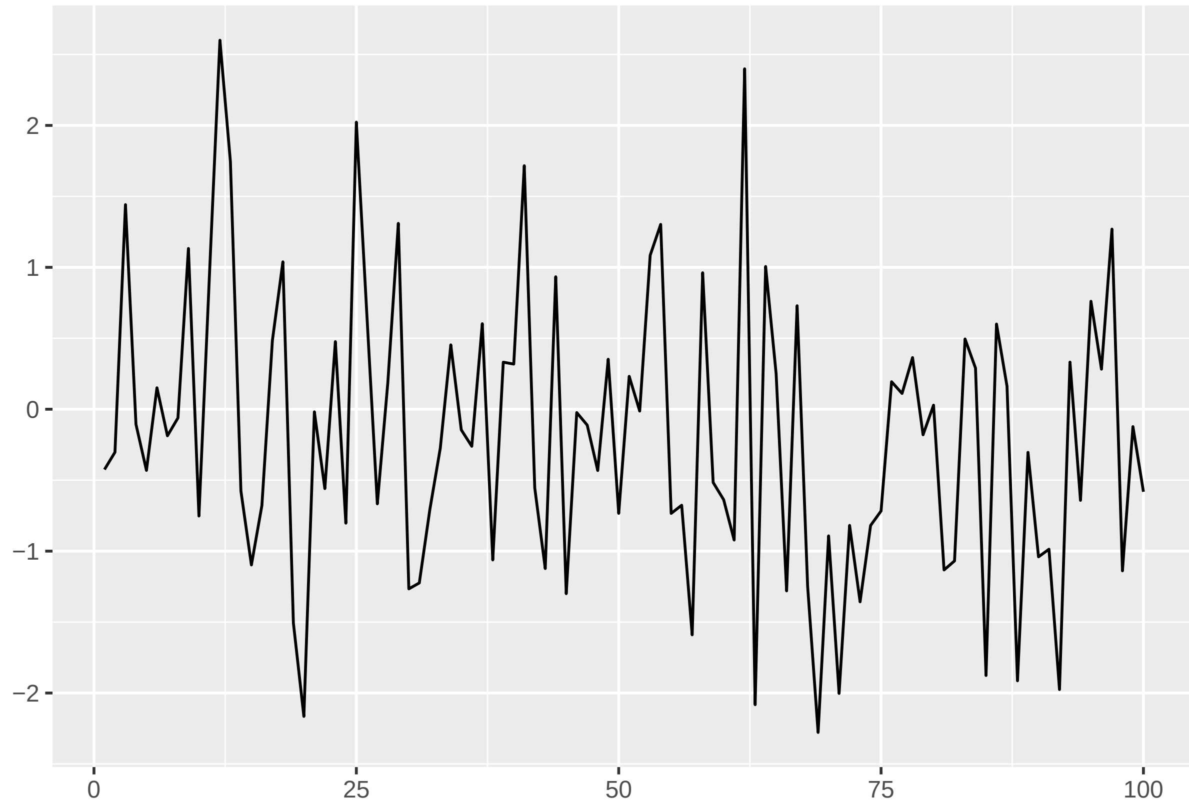


ARIMA - Auto Regressive Integrated Moving Average

To build an ARIMA time series use arima.sim(). Specify the level of each of AR, Differencing and MA. Optionally, specify the coefficients in additional parameters in the list, like ma or ar.

Reference: ARIMA Modeling with R - Chapter 1, Data Camp.

```
# Example of a White Noise model:  0 AR, 0 Diff, 0 MA
my.wn <- arima.sim(model = list(order = c(0,0,0)), n = 100)
autoplot(my.wn)
```



To estimate an ARIMA model, use the arima() function to build such as model and then evaluate.

This is some code to cycle through a bunch of models to see which one has the best AIC (you could use a different criterion, like BIC)

Reference: Metcalfe, A. and Cowpertwait, P. (2009). *Introductory Time Series with R*. New York, NY; Spring-Veriag, New York, p. 131

```
# Set the default - white noise
best.order <- c(0, 0, 0)
# Initialize the AIC score
best.aic <- Inf
# This is just looping through a few AR and MA models, may want to use
# acf() and pacf() to get a sense for the size of these loops
for (i in 0:2) for (j in 0:2) {
  # Calculate the AIC for the next type of ARIMA model
  fit.aic <- AIC(arima(diff(log(chicken)), order = c(i, 0, j)))
  # If the new AIC is lowest, keep it, the type of model and the model
  if (fit.aic < best.aic) {
    best.order <- c(i, 0, j)
    best.arma <- arima(diff(log(chicken)), order = best.order)
    best.aic <- fit.aic
  }
}

# Display the best type of model
best.order
```

```
## [1] 2 0 1
```

Can also build models using the sarima() function, supplying the AR, Differencing and MA parameters. This function will also build several visualizations of the residuals of the model to help evaluate if this is a good model or not.

Reference: ARIMA Modeling with R - Chapter 2, Data Camp.

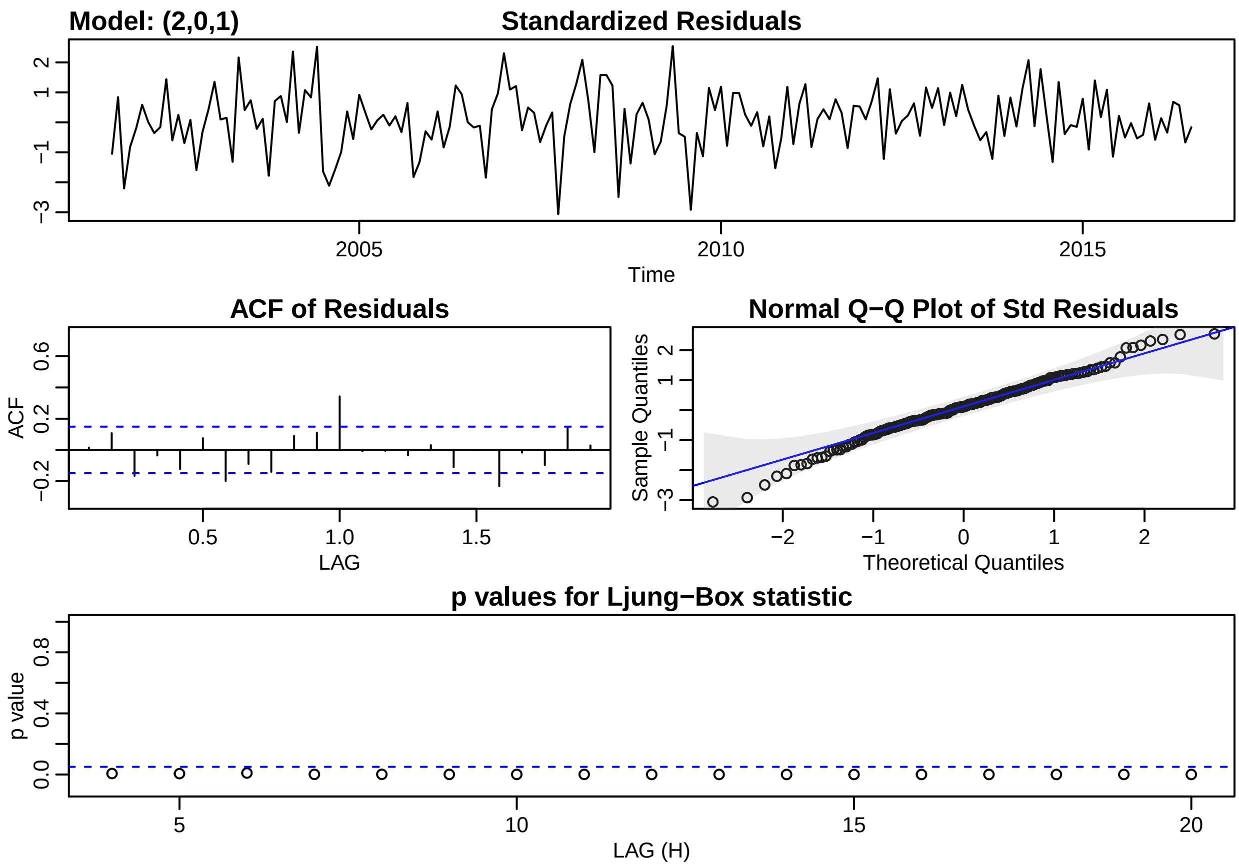
```r
library(astsa)
my.arima <- sarima(chicken, p = 2, d = 0, q = 1)
```

```
## initial  value 2.784868
## iter   2 value 2.197149
## iter   3 value 2.180068
## iter   4 value 1.403829
## iter   5 value 0.907290
## iter   6 value -0.070224
## iter   7 value -0.095722
## iter   8 value -0.108832
## iter   9 value -0.213591
## iter  10 value -0.213600
## iter  11 value -0.271682
## iter  12 value -0.333618
## iter  13 value -0.383427
## iter  14 value -0.399481
## iter  15 value -0.400894
## iter  16 value -0.400975
## iter  17 value -0.401010
## iter  18 value -0.401031
## iter  19 value -0.401032
## iter  20 value -0.401045
## iter  21 value -0.401956
## iter  22 value -0.402457
## iter  23 value -0.403025
## iter  24 value -0.403329
## iter  25 value -0.404528
## iter  26 value -0.405032
## iter  27 value -0.405785
## iter  28 value -0.405808
## iter  29 value -0.405816
## iter  30 value -0.405824
## iter  31 value -0.405825
## iter  32 value -0.405827
## iter  33 value -0.405829
## iter  34 value -0.405838
## iter  35 value -0.405855
## iter  36 value -0.405897
## iter  37 value -0.405898
## iter  38 value -0.405930
## iter  39 value -0.405943
## iter  40 value -0.405944
## iter  41 value -0.405944
## iter  42 value -0.405946
## iter  43 value -0.405951
## iter  44 value -0.405963
## iter  45 value -0.405990
```

```
## iter  46 value -0.405991
## iter  47 value -0.406014
## iter  48 value -0.406015
## iter  49 value -0.406016
## iter  50 value -0.406016
## iter  51 value -0.406017
## iter  52 value -0.406020
## iter  53 value -0.406026
## iter  54 value -0.406039
## iter  55 value -0.406040
## iter  56 value -0.406051
## iter  57 value -0.406055
## iter  58 value -0.406056
## iter  59 value -0.406056
## iter  60 value -0.406058
## iter  61 value -0.406063
## iter  62 value -0.406075
## iter  63 value -0.406098
## iter  64 value -0.406098
## iter  65 value -0.406116
## iter  66 value -0.406118
## iter  67 value -0.406118
## iter  68 value -0.406118
## iter  69 value -0.406121
## iter  70 value -0.406124
## iter  71 value -0.406133
## iter  72 value -0.406149
## iter  73 value -0.406149
## iter  74 value -0.406153
## iter  75 value -0.406161
## iter  76 value -0.406162
## iter  77 value -0.406162
## iter  78 value -0.406162
## iter  79 value -0.406162
## iter  80 value -0.406162
## iter  81 value -0.406163
## iter  81 value -0.406163
## final  value -0.406163
## converged
## initial  value -0.366038
## iter   2 value -0.367487
## iter   3 value -0.369377
## iter   4 value -0.373230
## iter   5 value -0.374092
## iter   6 value -0.374742
## iter   7 value -0.374976
## iter   8 value -0.375007
## iter   9 value -0.375011
## iter  10 value -0.375022
## iter  11 value -0.375027
## iter  12 value -0.375042
## iter  13 value -0.375050
## iter  14 value -0.375121
## iter  15 value -0.375276
```

```
## iter  16 value -0.375750
## iter  17 value -0.376215
## iter  18 value -0.376781
## iter  19 value -0.377474
## iter  20 value -0.377820
## iter  21 value -0.377997
## iter  22 value -0.378079
## iter  23 value -0.378087
## iter  24 value -0.378323
## iter  25 value -0.378397
## iter  26 value -0.378424
## iter  27 value -0.378432
## iter  28 value -0.378456
## iter  29 value -0.378466
## iter  30 value -0.378476
## iter  31 value -0.378600
## iter  32 value -0.378659
## iter  33 value -0.378758
## iter  34 value -0.378798
## iter  35 value -0.378815
## iter  36 value -0.378819
## iter  37 value -0.378843
## iter  38 value -0.378852
## iter  39 value -0.378863
## iter  40 value -0.378869
## iter  41 value -0.378878
## iter  42 value -0.378902
## iter  43 value -0.378943
## iter  44 value -0.378994
## iter  45 value -0.379022
## iter  46 value -0.379029
## iter  47 value -0.379032
## iter  48 value -0.379125
## iter  49 value -0.379127
## iter  50 value -0.379128
## iter  51 value -0.379130
## iter  52 value -0.379132
## iter  53 value -0.379136
## iter  54 value -0.379144
## iter  55 value -0.379159
## iter  56 value -0.379167
## iter  57 value -0.379173
## iter  58 value -0.379174
## iter  59 value -0.379174
## iter  60 value -0.379176
## iter  61 value -0.379178
## iter  62 value -0.379181
## iter  63 value -0.379182
## iter  64 value -0.379182
## iter  65 value -0.379182
## iter  66 value -0.379186
## iter  67 value -0.379186
## iter  68 value -0.379186
## iter  69 value -0.379187
```

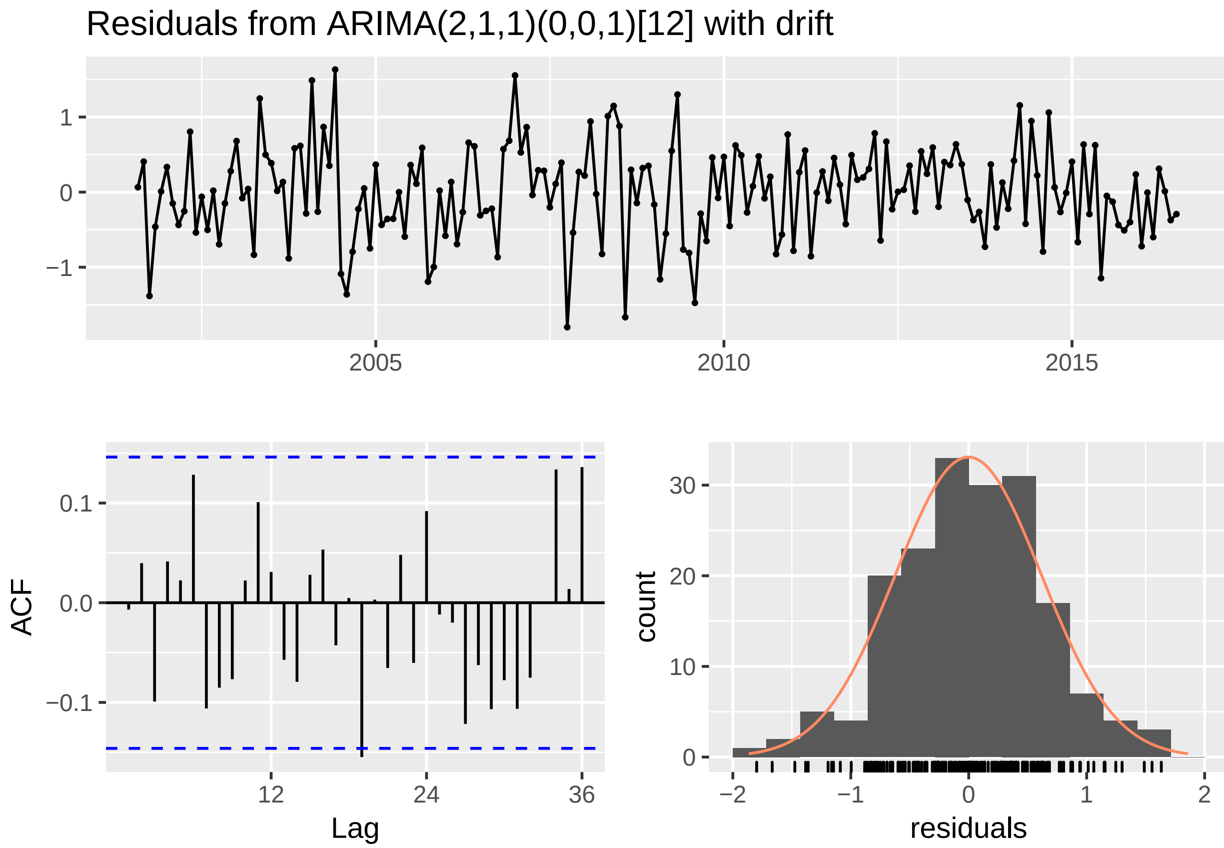
```
## iter  70 value -0.379187
## iter  71 value -0.379188
## iter  72 value -0.379189
## iter  73 value -0.379189
## iter  74 value -0.379190
## iter  75 value -0.379190
## iter  76 value -0.379190
## iter  76 value -0.379190
## iter  76 value -0.379190
## final  value -0.379190
## converged
```



checkresiduals() does analysis of a model's residuals, similar to sarima() earlier.

Reference: Forecasting Using R - Chapter 2, Data Camp.

```
library(forecast)
checkresiduals(auto.arima(chicken))
```



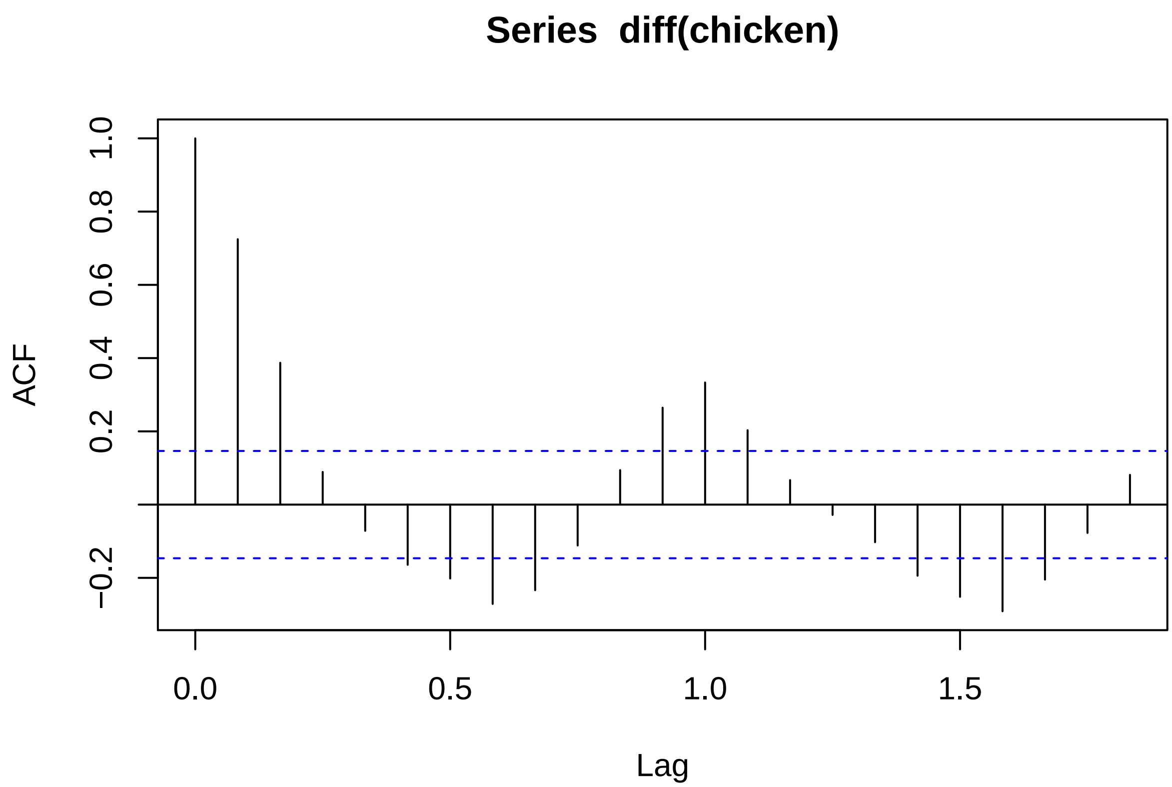

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(2,1,1)(0,0,1)[12] with drift
## Q* = 24.287, df = 19, p-value = 0.1854
##
## Model df: 5.   Total lags used: 24
```

To evaluate the model you might use, use the auto correlation function acf() and the partial auto correlation function pacf().

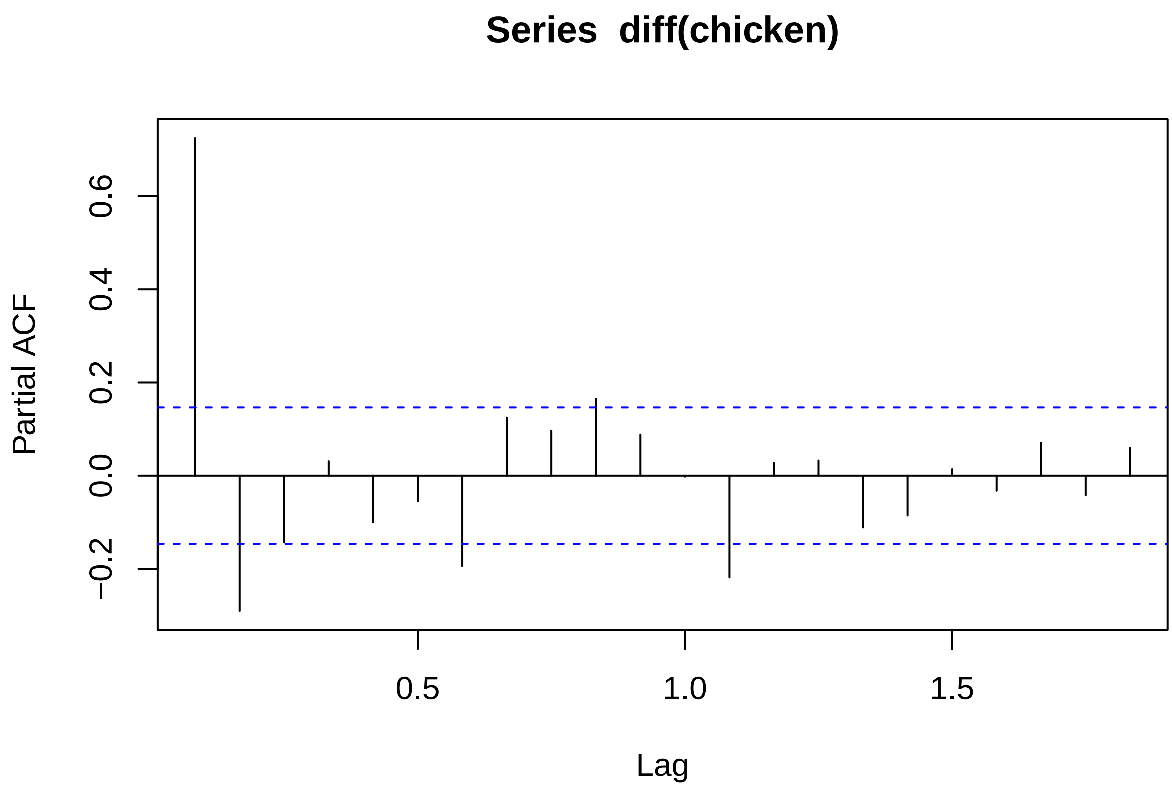
AR model: ACF tails off, PACF cuts off at lag MA model: ACF cuts off at lag, PACF tails off ARMA model: both tail off

Reference: ARIMA Modeling with R - Chapter 2 and Chapter 3, Data Camp.

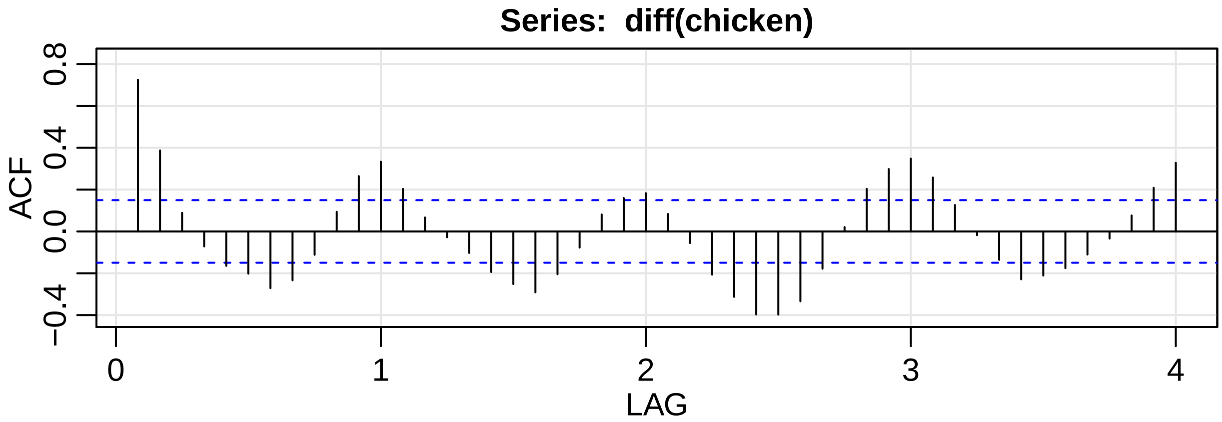
```
acf(diff(chicken))
```

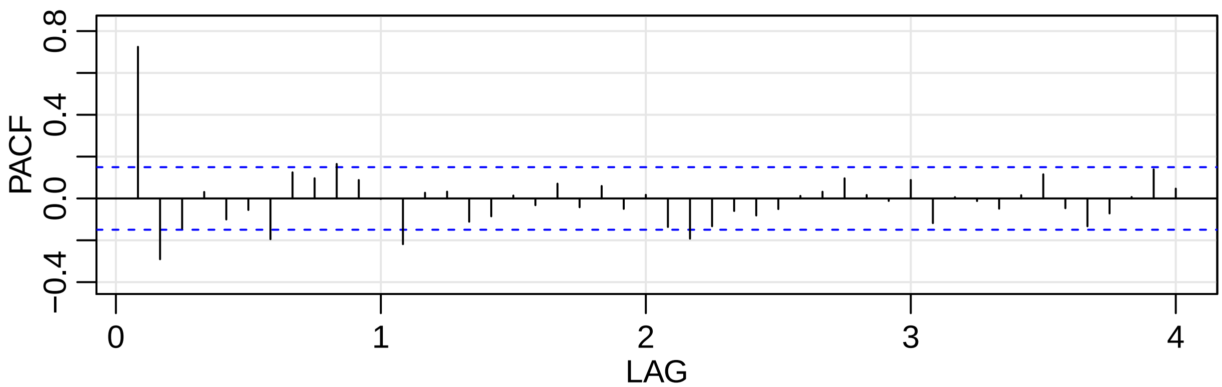



```
pacf(diff(chicken))
```




```
# Alternatively
acf2(diff(chicken))
```






```
##         ACF  PACF
##  [1,]  0.72  0.72
##  [2,]  0.39 -0.29
##  [3,]  0.09 -0.14
##  [4,] -0.07  0.03
##  [5,] -0.16 -0.10
##  [6,] -0.20 -0.06
##  [7,] -0.27 -0.19
##  [8,] -0.23  0.12
##  [9,] -0.11  0.10
## [10,]  0.09  0.16
## [11,]  0.26  0.09
## [12,]  0.33  0.00
## [13,]  0.20 -0.22
## [14,]  0.07  0.03
## [15,] -0.03  0.03
## [16,] -0.10 -0.11
## [17,] -0.19 -0.09
## [18,] -0.25  0.01
## [19,] -0.29 -0.03
## [20,] -0.20  0.07
## [21,] -0.08 -0.04
## [22,]  0.08  0.06
## [23,]  0.16 -0.05
## [24,]  0.18  0.02
## [25,]  0.08 -0.14
```

```
## [26,] -0.06 -0.19
## [27,] -0.21 -0.13
## [28,] -0.31 -0.06
## [29,] -0.40 -0.08
## [30,] -0.40 -0.05
## [31,] -0.33  0.01
## [32,] -0.18  0.03
## [33,]  0.02  0.10
## [34,]  0.20  0.02
## [35,]  0.30 -0.01
## [36,]  0.35  0.09
## [37,]  0.26 -0.12
## [38,]  0.13  0.01
## [39,] -0.02 -0.01
## [40,] -0.14 -0.05
## [41,] -0.23  0.02
## [42,] -0.21  0.12
## [43,] -0.18 -0.05
## [44,] -0.11 -0.13
## [45,] -0.03 -0.07
## [46,]  0.08  0.01
## [47,]  0.21  0.14
## [48,]  0.33  0.05
```

Error Trend and Seasonality models. ets() function picks the best model using AICc. Get the lambda from teh BoxCox.lambda() function (mentioned earlier).

Reference: Forecasting Using R - Chapter 4, Data Camp.

```
ets(chicken, lambda = 1.686)
```

```
## ETS(A,Ad,N)
##
## Call:
##  ets(y = chicken, lambda = 1.686)
##
##   Box-Cox transformation: lambda= 1.686
##
##   Smoothing parameters:
##     alpha = 0.9999
##     beta  = 0.9997
##     phi   = 0.8
##
##   Initial states:
##     l = 696.4935
##     b = -3.8061
##
##   sigma:  14.6426
##
##      AIC     AICc      BIC
## 1907.877 1908.363 1927.035
```

auto.arima() can also be used to pick a model and can handle seasonality.

Reference: Forecasting Using R - Chapter 4, Data Camp.

```r
auto.arima(chicken)
```

```
## Series: chicken
## ARIMA(2,1,1)(0,0,1)[12] with drift
##
## Coefficients:
##          ar1      ar2      ma1    sma1   drift
##       1.2933  -0.5375  -0.4019  0.2756  0.2518
## s.e.  0.2220   0.1542   0.2569  0.0692  0.1428
##
## sigma^2 estimated as 0.396:  log likelihood=-169.51
## AIC=351.01   AICc=351.5   BIC=370.14
```