“Forecasting: Principles and Practice” Notes

Francis Tan

2016-08-31

Contents

Chapter 1: Getting Started

Types of Quantitative Forecasts

- Cross-sectional Data
  - Given a set of parameters, try to *predict* an outcome based on data. For example, predict the house price based on number of bedrooms, bathrooms, etc.
- Time series Data
  - Forecast future outcome based on historical data

Basic Steps of Forecasting

1. Problem Definition
2. Gathering Information
3. Exploratory Analysis
4. Choosing and Fitting Models
5. Using and Evaluating Model

Chapter 2: Forecaster's Toolbox

Graphs

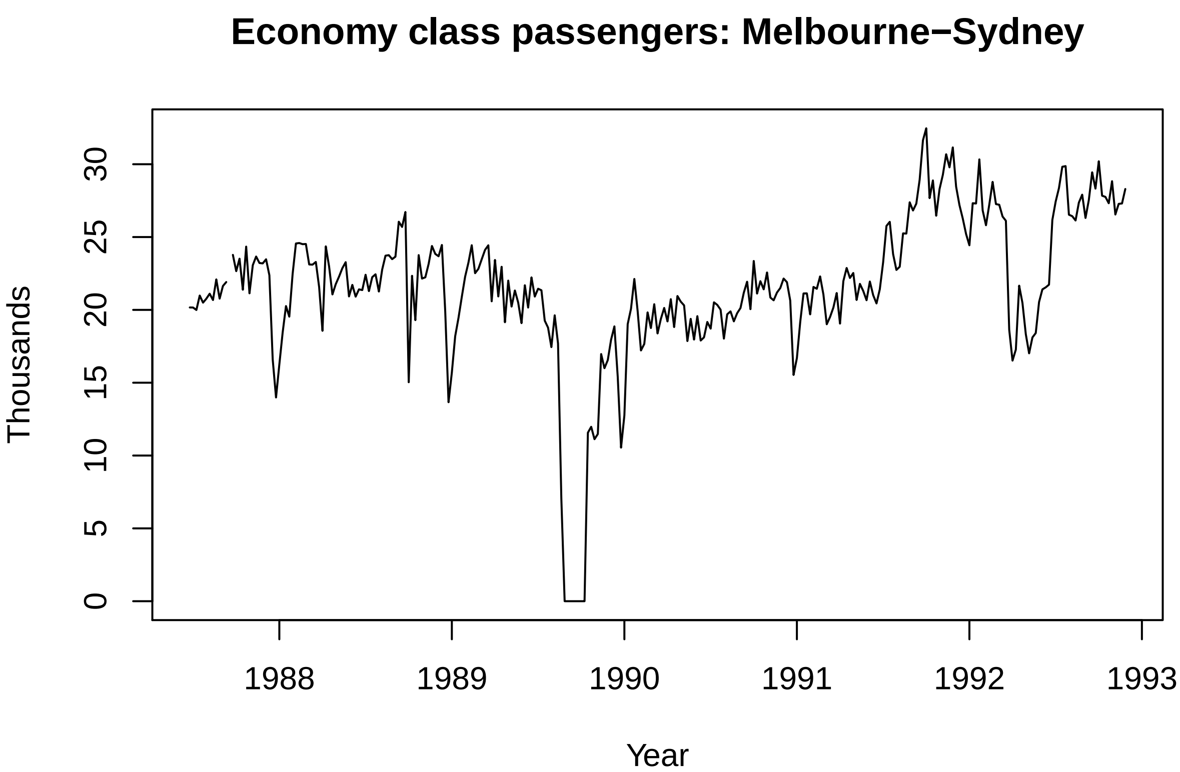
First thing to do for any forecasting exercise is to plot the data to look for patterns or any abnormalities.

Time Plots

aka Line graphs.

Example 1

```
data(melsyd)
plot(melsyd[,"Economy.Class"],
  main="Economy class passengers: Melbourne-Sydney",
  xlab="Year",ylab="Thousands")
```

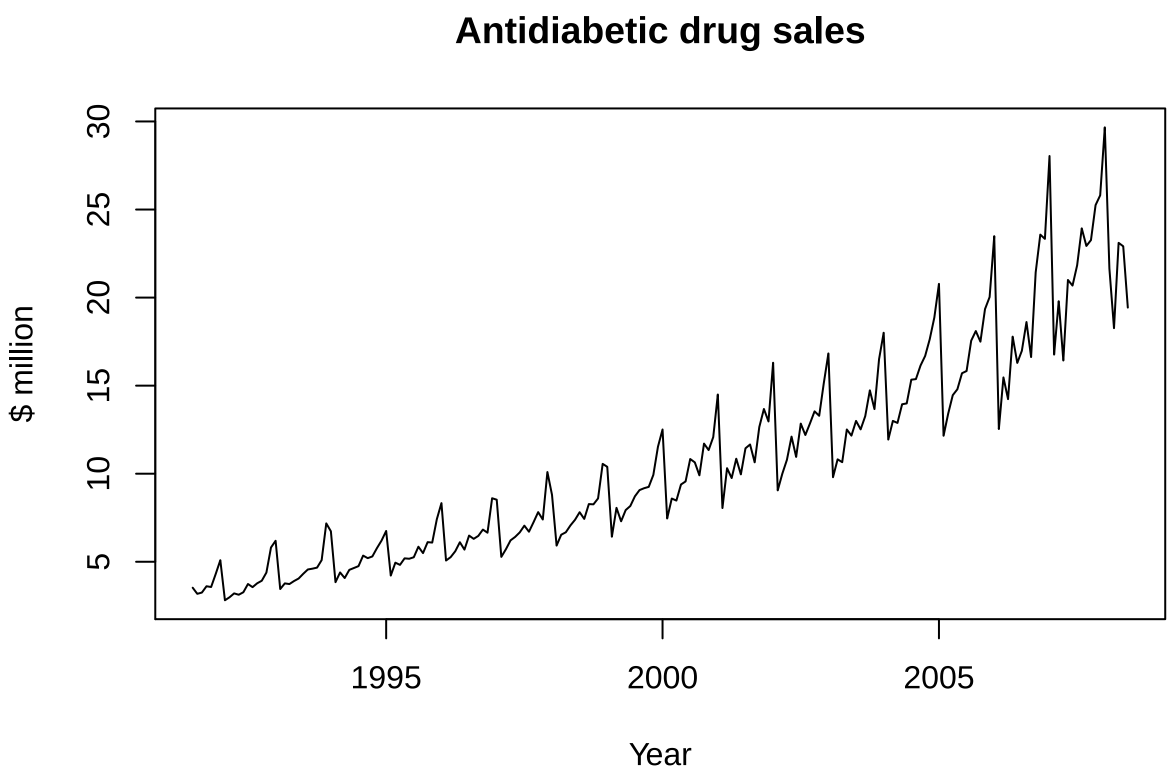


Notes:

- Missing data in 1989 – industrial dispute
- Dip in 1992 – trial which replaced some economy class seats with business class
- Large increase in 1991
- etc

Example 2

```
data(a10)
plot(a10, ylab="$ million", xlab="Year", main="Antidiabetic drug sales")
```



Notes:

- Seasonality
- Upward trend

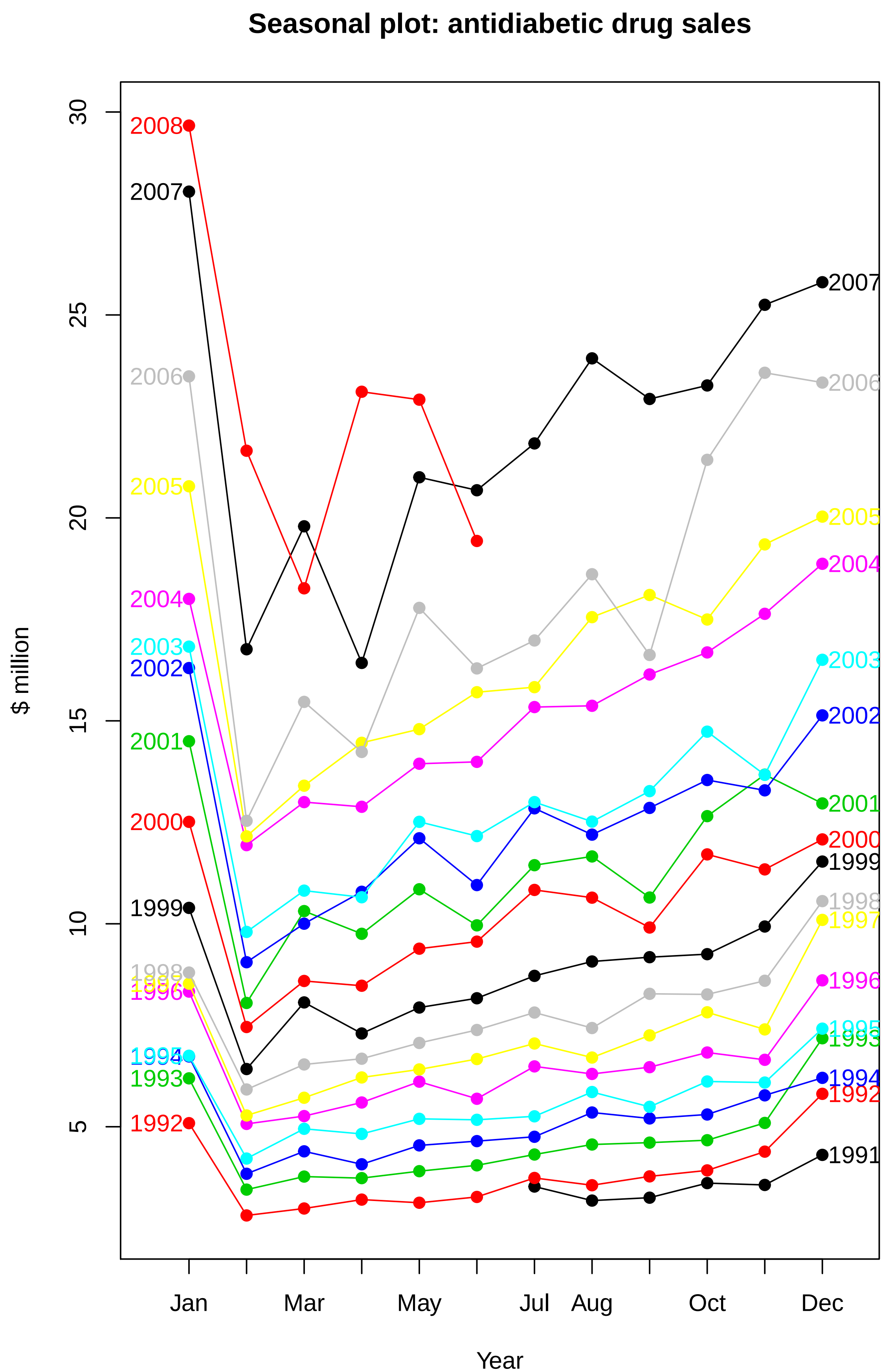
Common Time Series Patterns

- **Trend**
- **Seasonality**
- **Cycles** – rises and falls that are not of a fixed period

Seasonal Plots

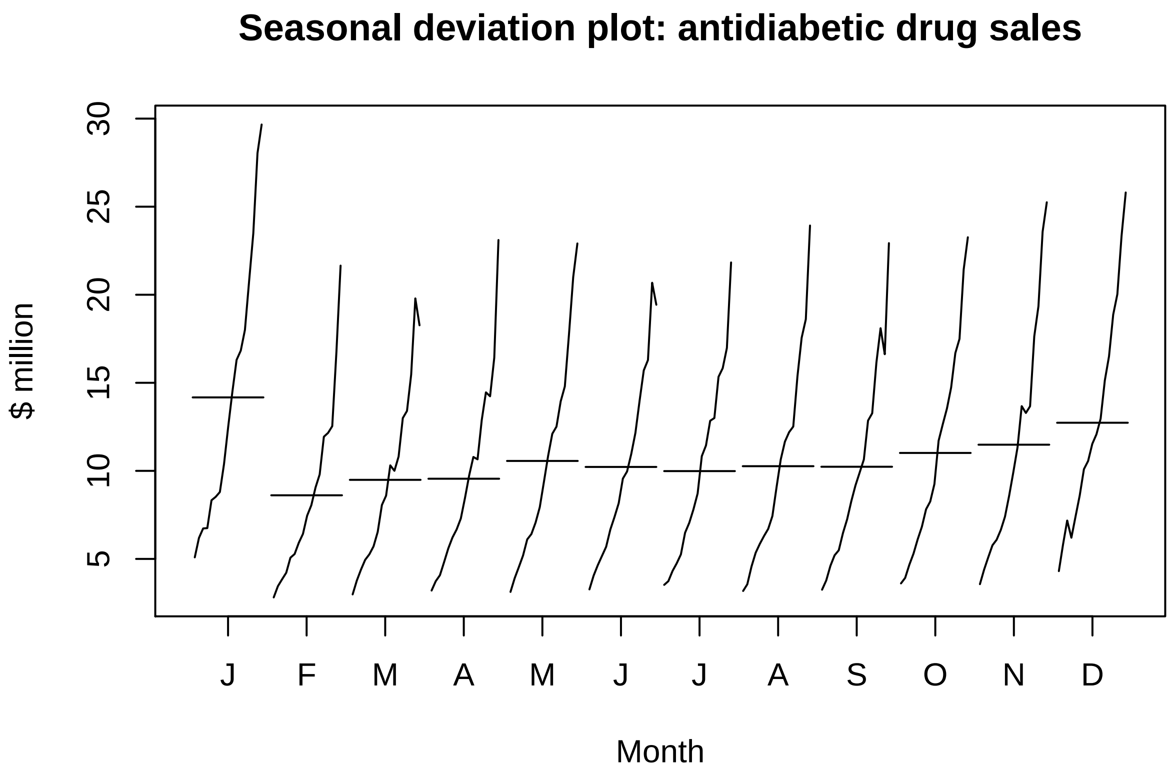
Line plots comparing each *season*.

```
seasonplot(a10,
           ylab='$ million',
           xlab='Year',
           main='Seasonal plot: antidiabetic drug sales',
           year.labels=TRUE,
           year.labels.left=TRUE,
           col=1:20,
           pch=19)
```



Seasonal plot: antidiabetic drug sales
$ million
Year
Jan
Mar
May
Jul
Aug
Oct
Dec
5
10
15
20
25
30
2008
2007
2006
2005
2004
2003
2002
2001
2000
1999
1998
1997
1996
1995
1994
1993
1992
1991

```r
monthplot(a10,
          ylab='$ million',
          xlab='Month',
          main='Seasonal deviation plot: antidiabetic drug sales')
```



Scatterplots

Useful for analyzing cross-sectional data

Summary Statistics

Univariate statistics

Can simply use *summary* function on the data.

Bivariate statistics

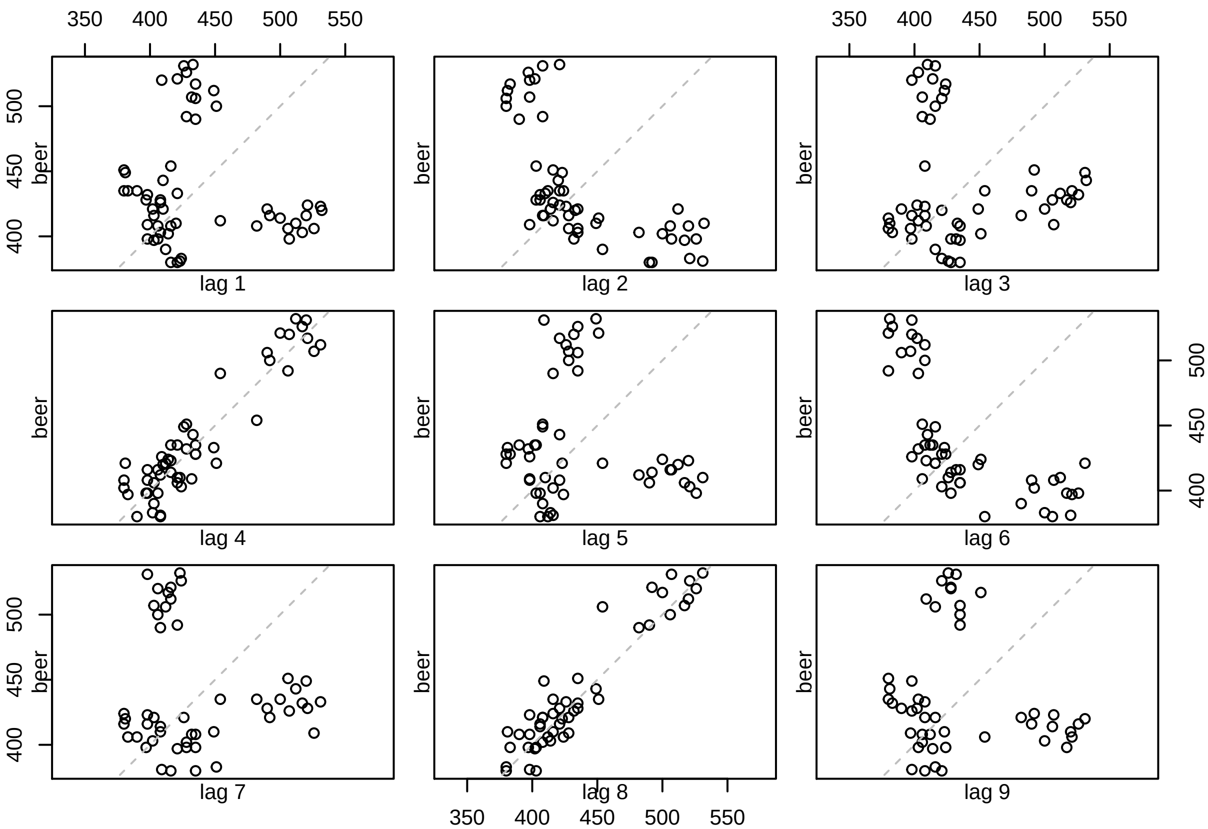
Correlation coefficient: r

$$r = \frac{\sum (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum (x_i - \overline{x})^2}\sqrt{\sum (y_i - \overline{y})^2}}$$

Autocorrelation

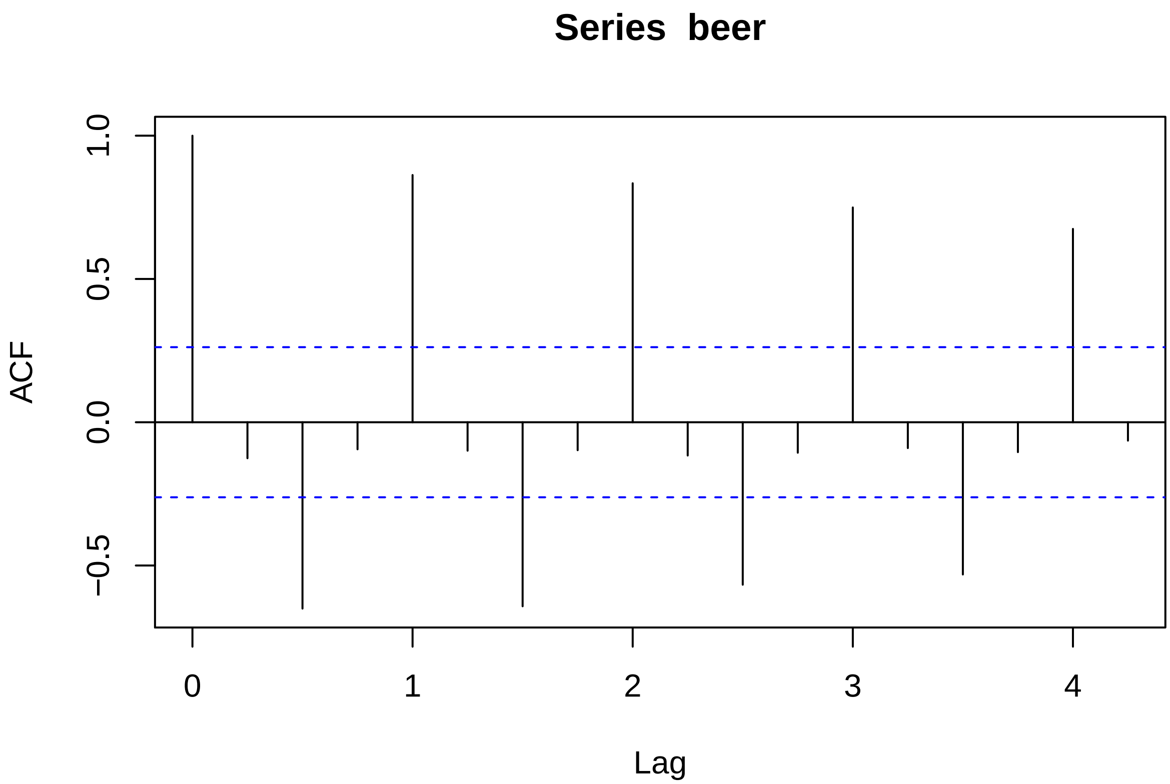
Used to test correlation on lag. r_1 tests correlation on lag 1, r_2 tests correlation on lag 2, etc.

```r
data(ausbeer)
beer <- window(ausbeer, start=1992, end=2006-0.1)
lag.plot(beer, lags=9, do.lines=FALSE)
```



Each lag has a corresponding correlation value r. These correlation values are plotted to form an *autocorrelation function* or *ACF*. The plot is known as a *correlogram*.

```r
acf(beer)
```



Notes:

- r_4 at lag 4 has the highest correlation because seasonal patterns happen every four quarters
- Negative correlations happen two quarters after peaks

Time series that show no autocorrelation are called *white noise.*Acf plot will show no significant correlations for any lag periods.

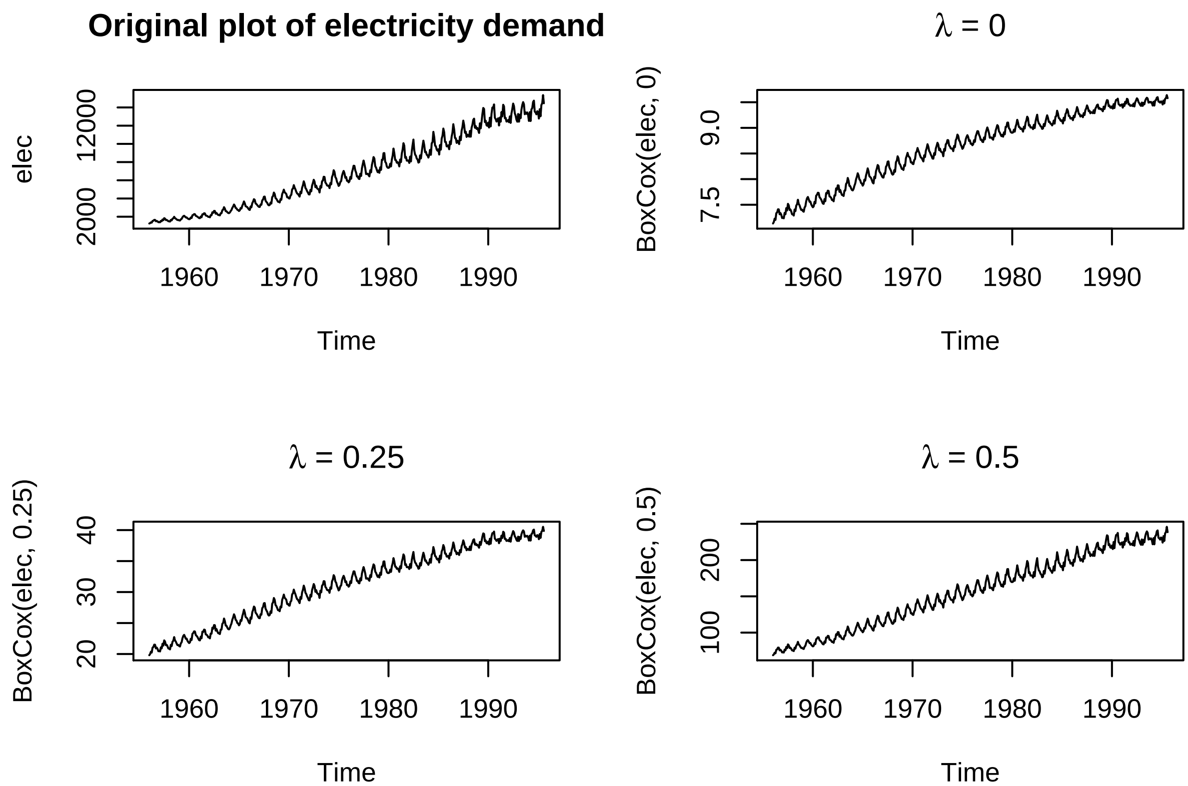
Simple forecasting methods

- **Mean** – just the mean of historical values for all forecasted values.
- **Naive** – just the last actual value for all forecasted values.
- **Seasonal Naive** – variation of naive. Uses the last value of some period last season.
- **Drift** – variation of naive. Allows forecast to increase or decrease over time (the *drift*) based on average change.

Transformations

Log and power transformations are common. *Box-Cox Transformations* is a useful family of log and power transformations. If the coefficient λ is 0, it does a natural log, otherwise it does a power transformation. λ can be between 0 and 1. A good lambda will transform the data such that each seasonal swing is roughly equal. Running the function *BoxCox.lambda(data)* will choose a λ for you. In this case, it will choose 0.27.

```
data(elec)
par(mfrow=c(2, 2))
plot(elec, main='Original plot of electricity demand')
plot(BoxCox(elec, 0), main=expression(paste(lambda, ' = 0')))
plot(BoxCox(elec, 0.25), main=expression(paste(lambda, ' = 0.25')))
plot(BoxCox(elec, 0.5), main=expression(paste(lambda, ' = 0.5')))
```



After tranforming, we need to make a forecast on the transformed data. Then we need to *back transform* to obtain the forecast in the original scale.

Evaluating forecast accuracy

Scale-dependent errors

Forecast error is simply $e_i = y_i - \hat{y}_i$ where y_i is actual and $\hat{y}_i$ is forecast. Two common measures are:

$$\text{Mean absolute error: MAE} = mean(|e_i|)$$

$$\text{Root mean squared error: RMSE} = \sqrt{mean(e_i^2)}$$

MAE is most common, however can only be compared to values on the same scale, or on the same data set.

Percentage errors

Scale independent so can compare errors from different data sets. This can be calculated as $p_i = 100e_i/y_i$. The most commonly used measure is:

$$\text{Mean absolute percentage error: MAPE} = mean(|p_i|)$$

This can present the problem is any value y_i is 0 or close to 0.

Scaled errors

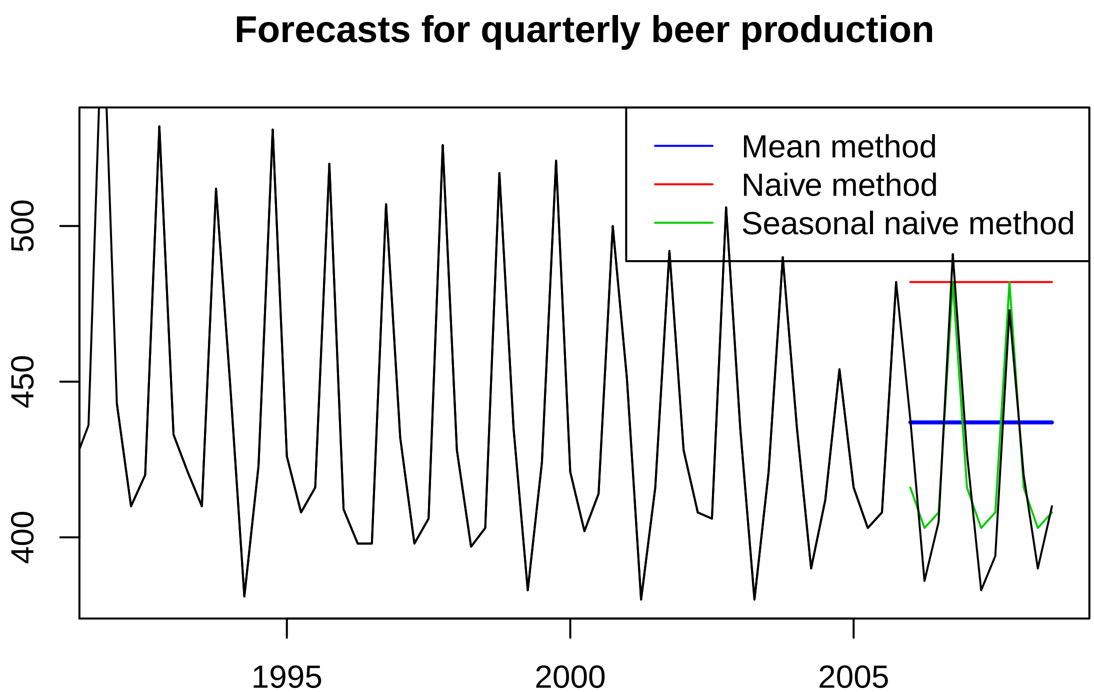
Scaled errors are used as an alternative to percentage errors. The *mean absolute scaled error* or *MASE* is a commonly used one (alternatively *mean squared scaled error* or *MSSE* is used).

Example

```
beer2 <- window(ausbeer,start=1992,end=2006-.1)

beerfit1 <- meanf(beer2,h=11)
beerfit2 <- rwf(beer2,h=11)
beerfit3 <- snaive(beer2,h=11)

plot(beerfit1, plot.conf=FALSE,
  main="Forecasts for quarterly beer production")
lines(beerfit2$mean,col=2)
lines(beerfit3$mean,col=3)
lines(ausbeer)
legend("topright", lty=1, col=c(4,2,3),
  legend=c("Mean method","Naive method","Seasonal naive method"))
```



The following shows the various error tests:

```
beer3 <- window(ausbeer, start=2006)
accuracy(beerfit1, beer3)
```

```
##                         ME     RMSE      MAE        MPE     MAPE     MASE
## Training set  8.121418e-15 44.17630 35.91135 -0.9510944 7.995509 2.444228
## Test set     -1.718344e+01 38.01454 33.77760 -4.7345524 8.169955 2.298999
##                     ACF1 Theil's U
## Training set -0.12566970        NA
## Test set     -0.08286364 0.7901651
```

```r
accuracy(beerfit2, beer3)
```

```
##                      ME     RMSE      MAE         MPE     MAPE     MASE
## Training set   0.7090909 66.60207 55.43636  -0.8987351 12.26632 3.773156
## Test set     -62.2727273 70.90647 63.90909 -15.5431822 15.87645 4.349833
##                     ACF1 Theil's U
## Training set -0.25475212        NA
## Test set     -0.08286364  1.428524
```

```r
accuracy(beerfit3, beer3)
```

```
##                     ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -1.846154 17.24261 14.69231 -0.4803931 3.401224 1.0000000
## Test set     -2.545455 12.96849 11.27273 -0.7530978 2.729847 0.7672537
##                    ACF1 Theil's U
## Training set -0.3408329        NA
## Test set     -0.1786912   0.22573
```

Here we see that the seasonal naive method is best.

Residual diagnostics

Residuals are simply the difference between the forecast and actual value $e_i = y_i - \hat{y}_i$. A good forecast will yield residuals with the following properties:

- Residuals are uncorrelated. If you find correlations then there was something not included in the forecasting model.
- Residuals have zero mean. A non-zero mean means forecast is biased.

If the above properties are not satisfied, then forecast can be improved. If residuals have a mean m, then simply adding m to all forecasts will solve the problem. Fixing the correlation problem will be explained in **Chapter 8**.

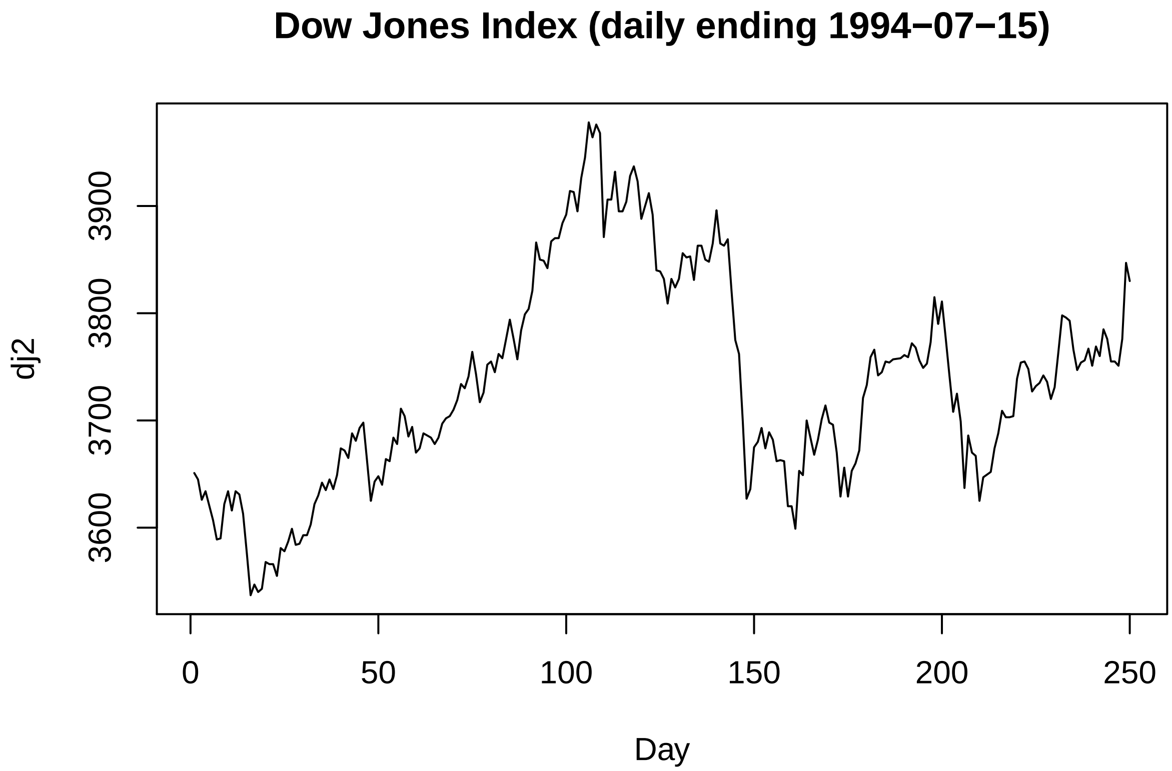
The following two properties are not required, but useful:

- Residuals have constant variance.
- Residuals are normally distributed.

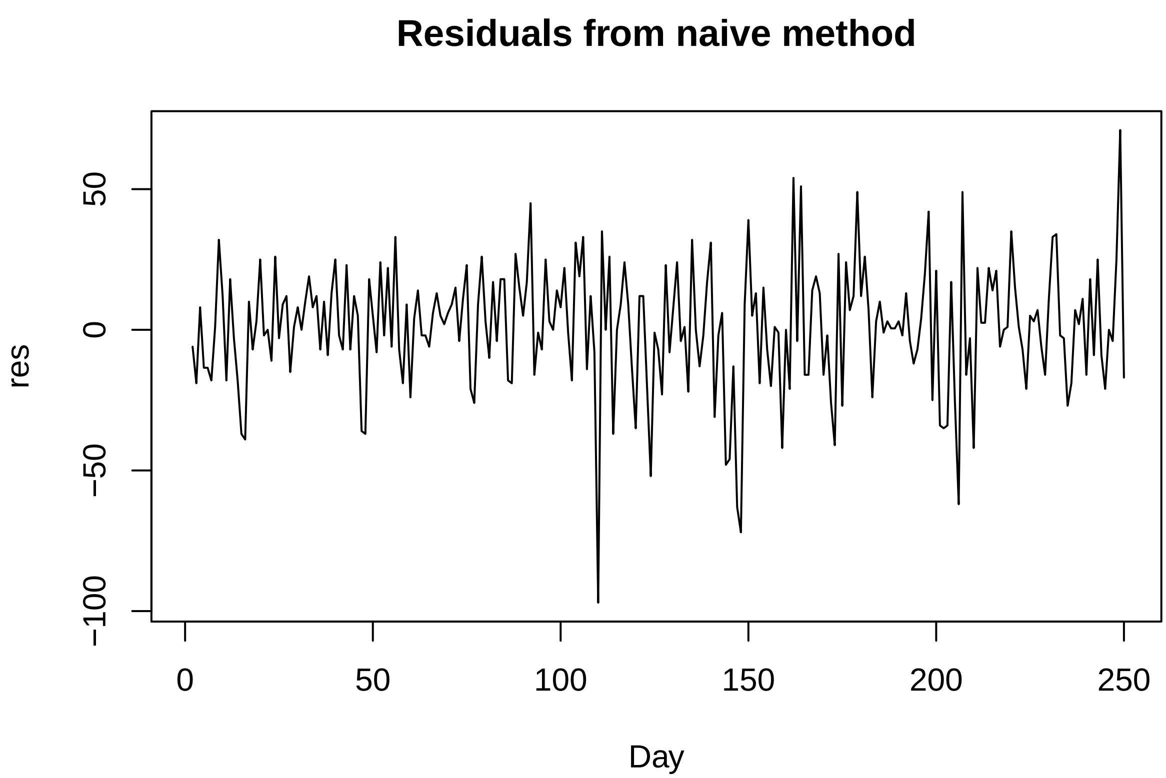
Example: Dow Jones

Naive method is usually best for stocks. Therefore, residuals are simply the difference between consecutive observations.

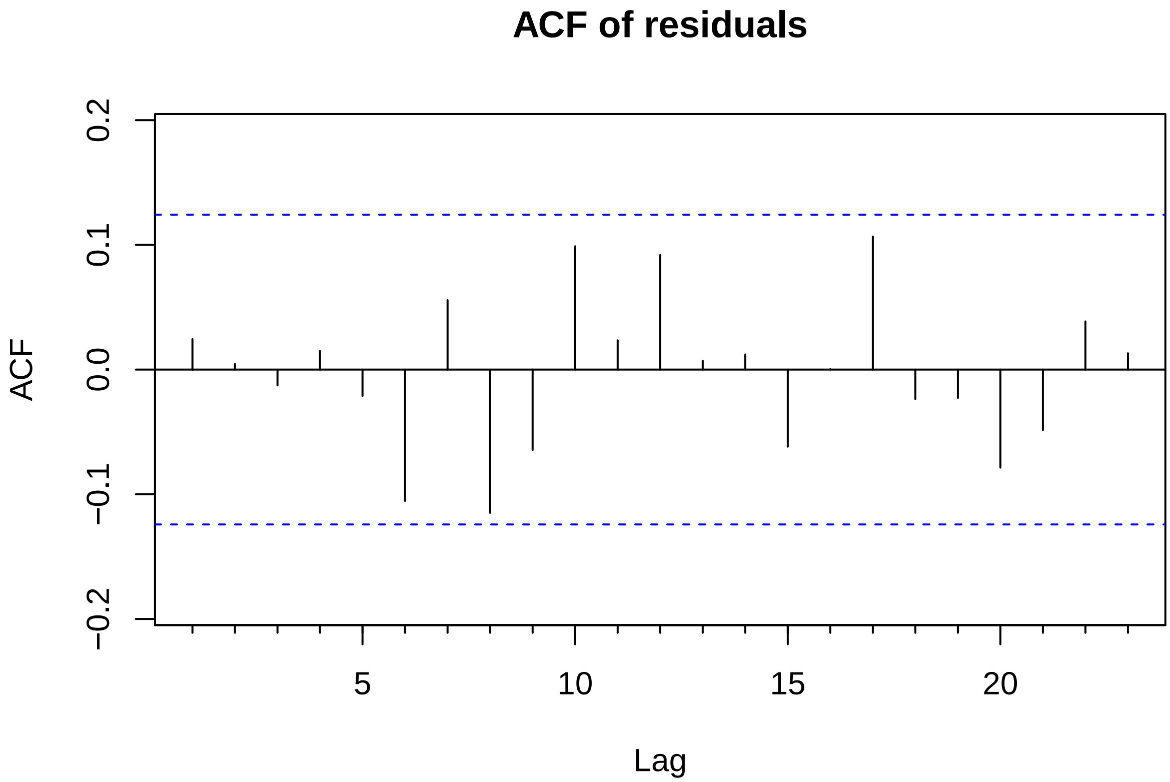
```r
data(dj)
# par(mfrow=c(4, 1))
dj2 <- window(dj, end=250)
plot(dj2,
     main='Dow Jones Index (daily ending 1994-07-15)',
     xlab='Day')
```



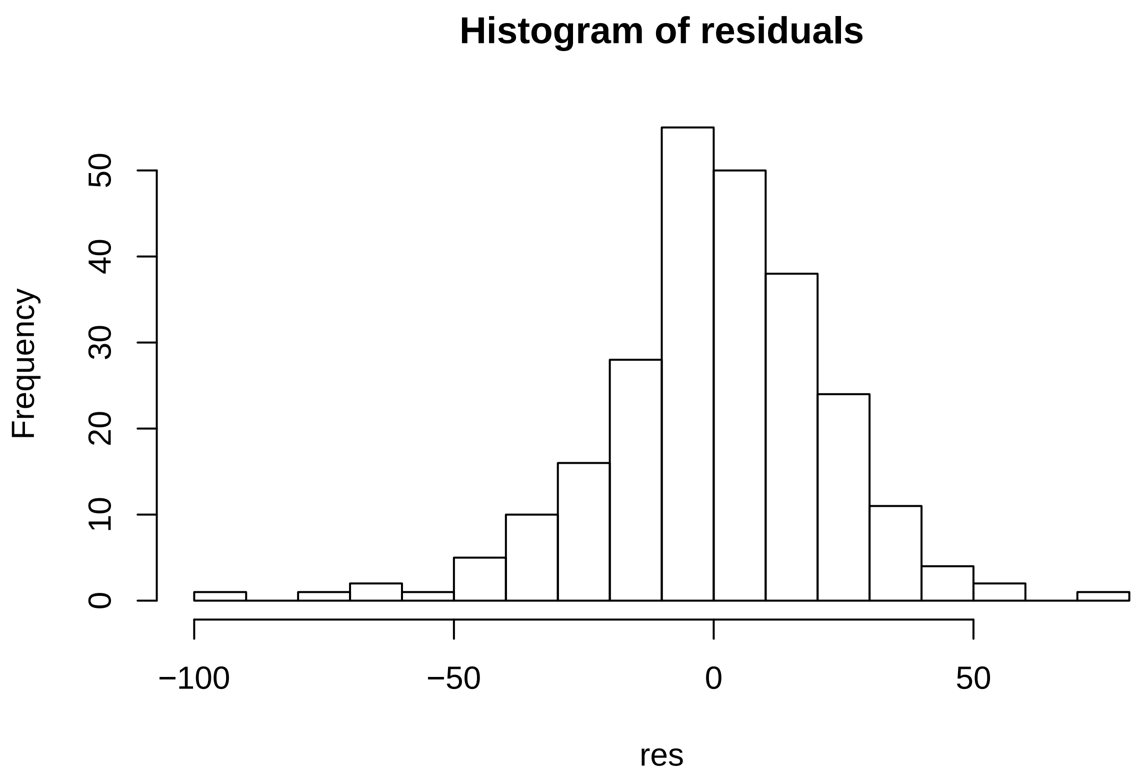

```r
res <- residuals(naive(dj2))
plot(res,
     main='Residuals from naive method',
     xlab='Day')
```



```r
Acf(res,
    main='ACF of residuals')
```




```r
hist(res,
     nclass='FD',
     main='Histogram of residuals')
```



Notes:

- [x] Residuals are not correlated.
- [x] Residuals are close to zero.
- [x] Residuals have constant variance.
- [ ] Not quite normally distributed so prediction intervals may be inaccurate.

Portmanteau tests for autocorrelation

Test if autocorrelation is the result of white noise. Portmanteau test

Chapter 4: Simple regression

Fit a line over observations where it minimizes the sum of square errors:

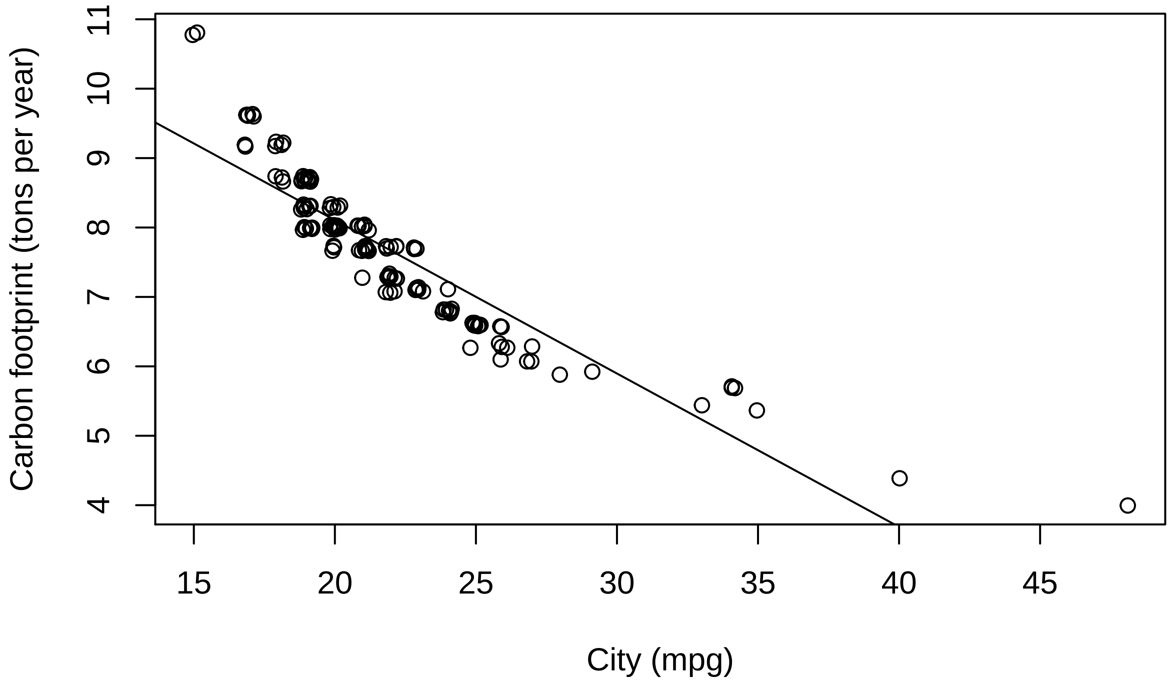
$$\sum_{i=1}^{N} \epsilon_i^2$$

The correlation coefficient r shows how much x predicts y.

Example: Car emission

```
data(fuel)
plot(jitter(Carbon) ~ jitter(City),
     xlab='City (mpg)',
     ylab='Carbon footprint (tons per year)',
```

```r
      data=fuel)
fit <- lm(Carbon ~ City, data=fuel)
abline(fit)
```



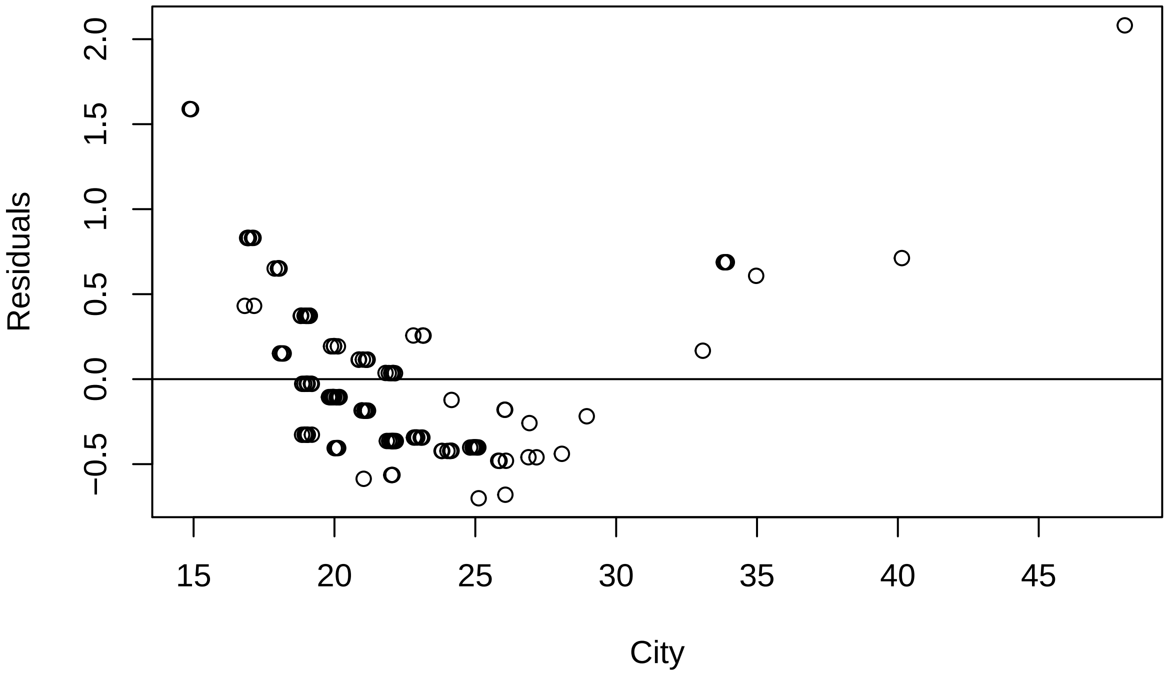

```r
summary(fit)
```

```
##
## Call:
## lm(formula = Carbon ~ City, data = fuel)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.7014 -0.3643 -0.1062  0.1938  2.0809
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 12.525647   0.199232   62.87   <2e-16 ***
## City        -0.220970   0.008878  -24.89   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4703 on 132 degrees of freedom
## Multiple R-squared:  0.8244, Adjusted R-squared:  0.823
## F-statistic: 619.5 on 1 and 132 DF,  p-value: < 2.2e-16
```

Evaluating regression models

```r
res <- residuals(fit)
plot(jitter(res) ~ jitter(City),
     ylab='Residuals',
     xlab='City',
```

```
    data=fuel)
abline(0, 0)
```



We see that there is a U-shaped pattern and therefore a simple linear model may not be appropriate.

Outliers

An outlier is considered an *influential observation* if it has a large impact on the regression model

Example: Predicting weight from height

Red line is a fit if the outlier is included while the black line is the fit if outlier is excluded.

Goodness of fit

R^2 or the square of the correlation coefficient r measures how well the model fits the data. This can be found in the section called *Multiple R-squared* in the output of the *summary* function. The car example shows the R^2 value to be 0.8244. This means that 82% of the variation is captured by the model. Be careful as this is often used incorrectly. As in the example above, we see that the residuals have a U-shaped pattern.

Standard error of regression

Another measure of how well a model fits is the *Standard error of regression* which is the same as the standard deviation of residuals. This can be found in the section called *Residual standard error* in the output of the *summary* function. This is also used in calculating the prediction interval.

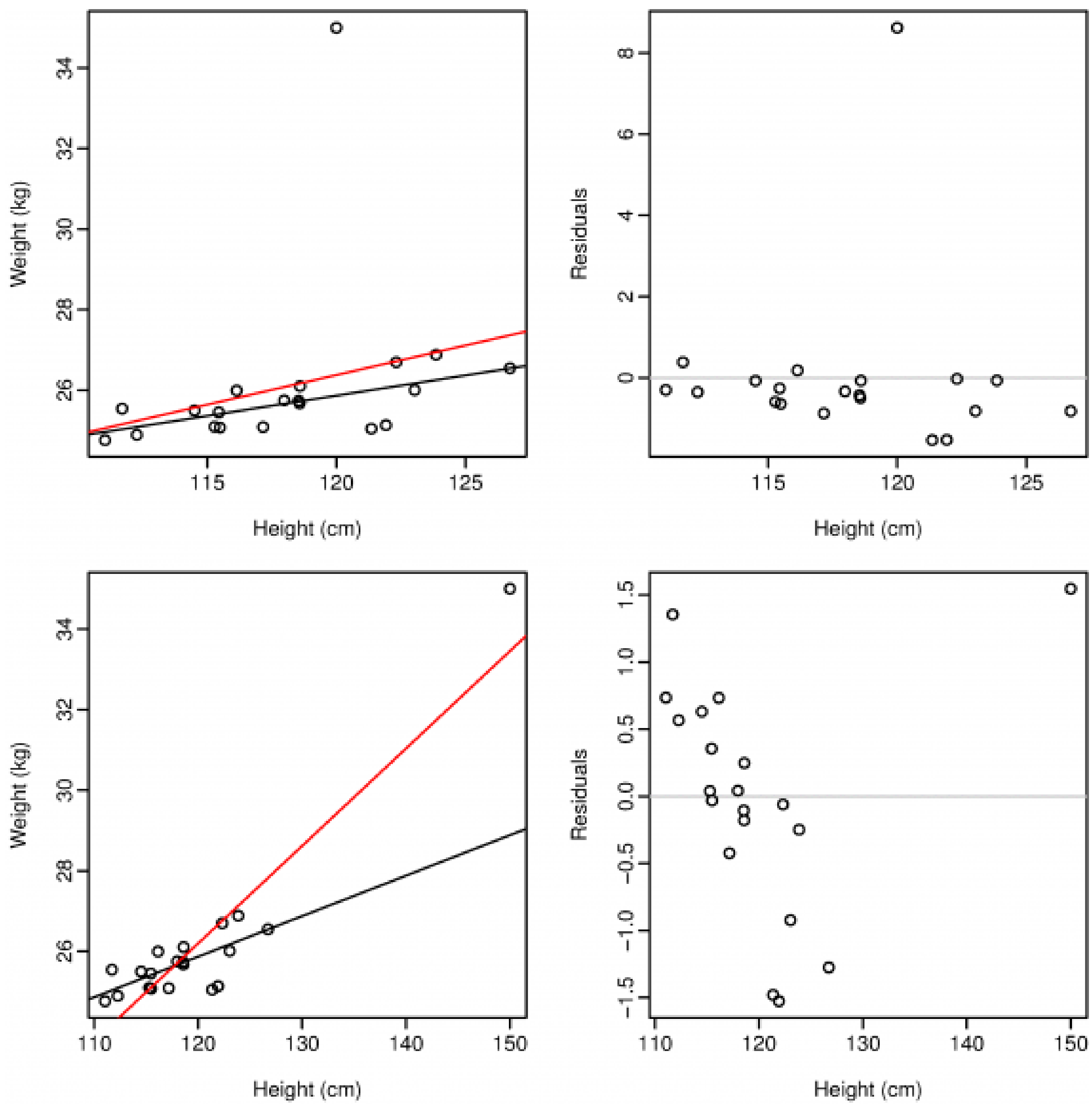
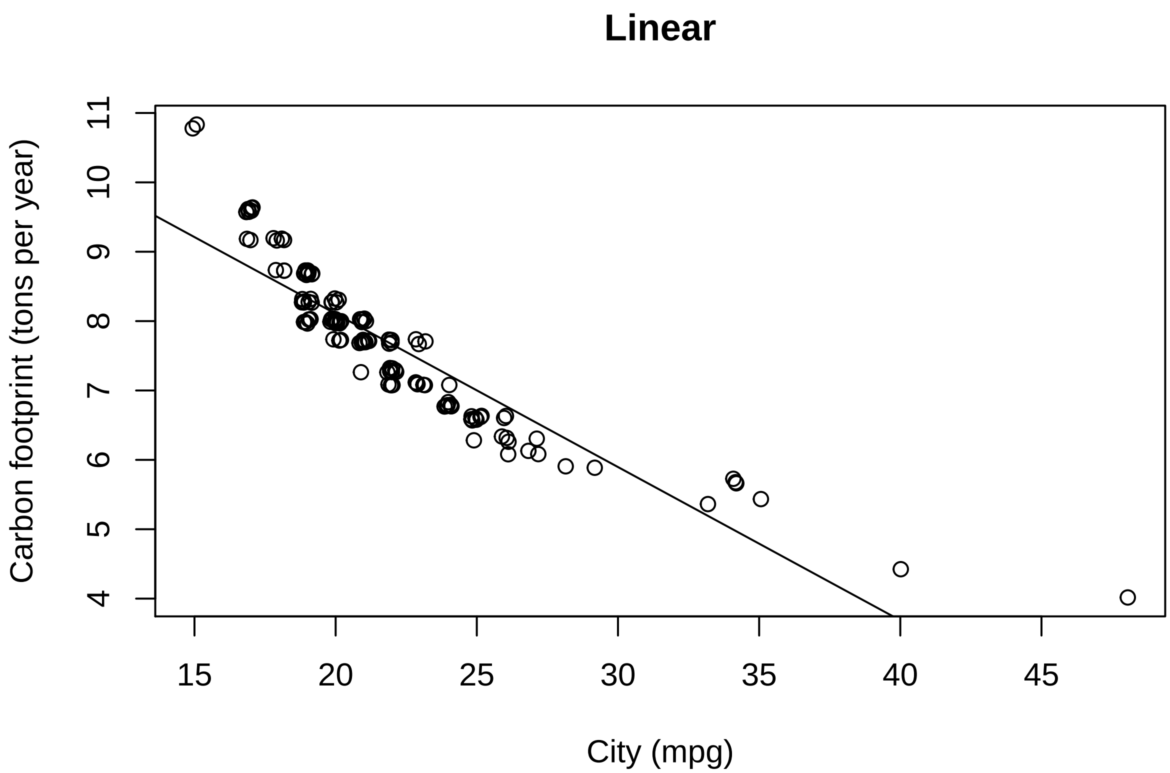


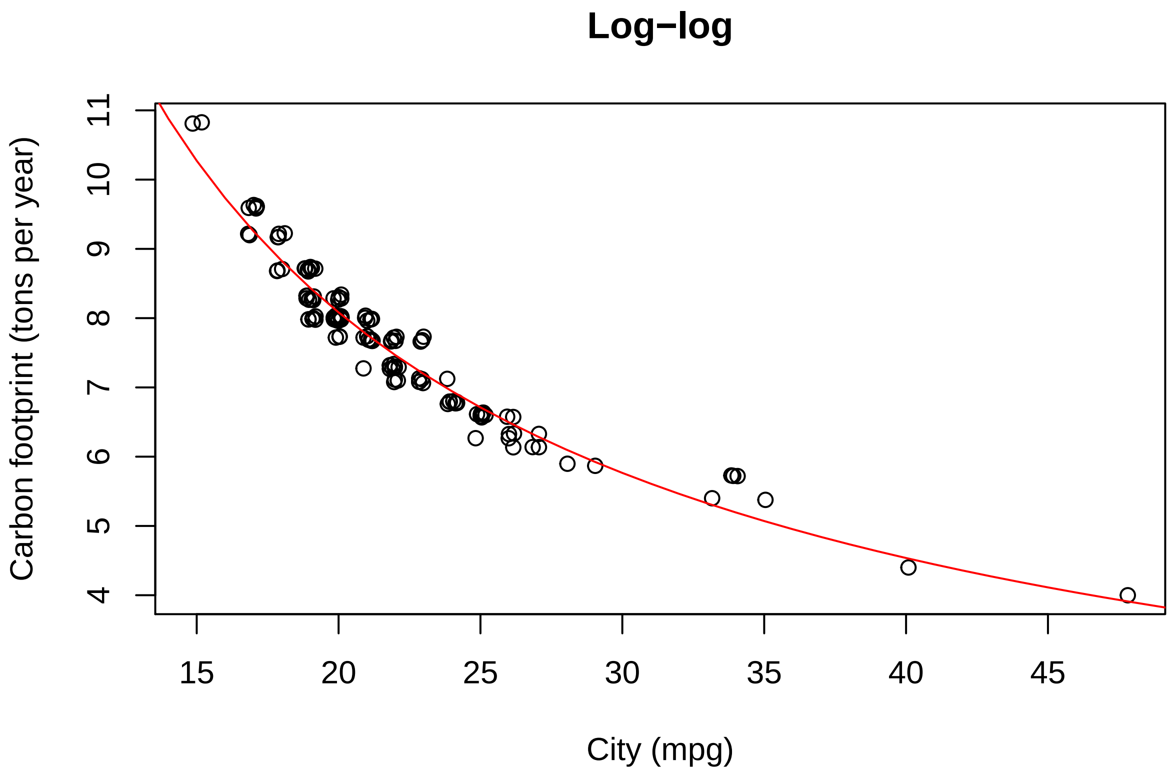
Figure 1: Weight vs height

Non-linear regression

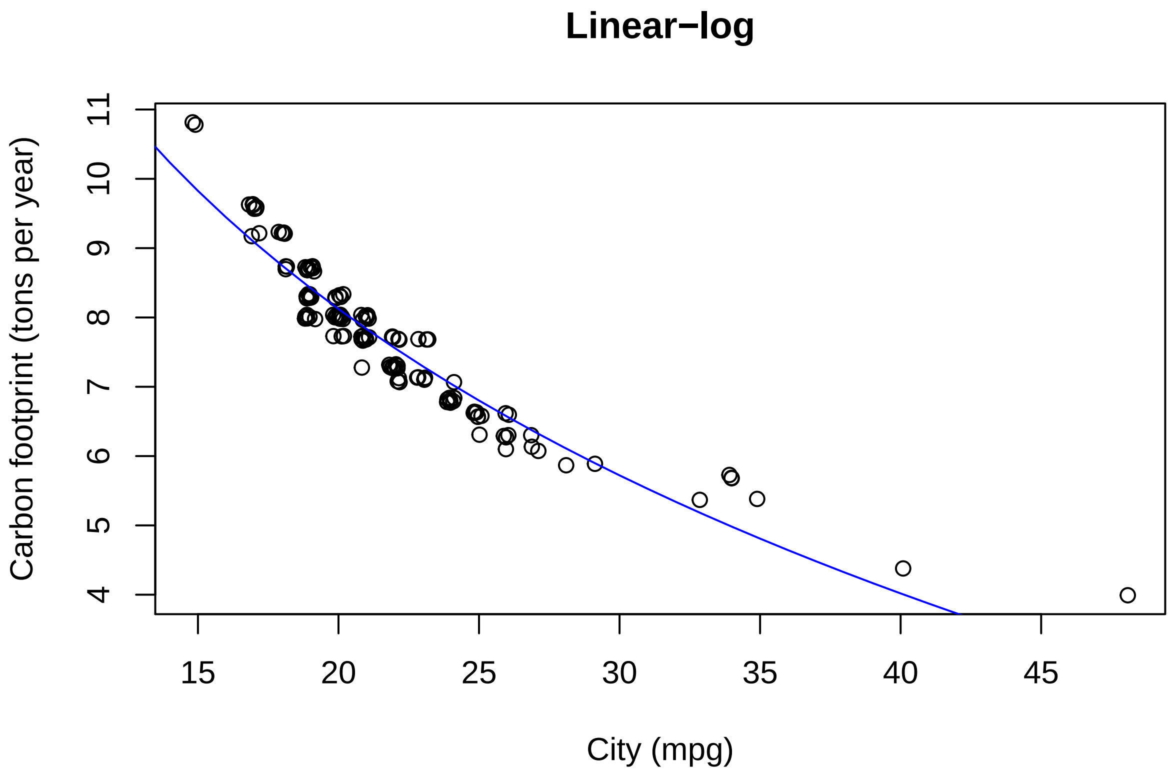
```
fit <- lm(Carbon ~ City, data=fuel)
fit2 <- lm(log(Carbon) ~ log(City), data=fuel)
fit3 <- lm(Carbon ~ log(City), data=fuel)
fit4 <- lm(log(Carbon) ~ City, data=fuel)
plot(jitter(Carbon) ~ jitter(City),
     xlab='City (mpg)',
     ylab='Carbon footprint (tons per year)',
     main='Linear',
     data=fuel)
lines(1:50, fit$coef[1] + fit$coef[2] * (1:50))
```



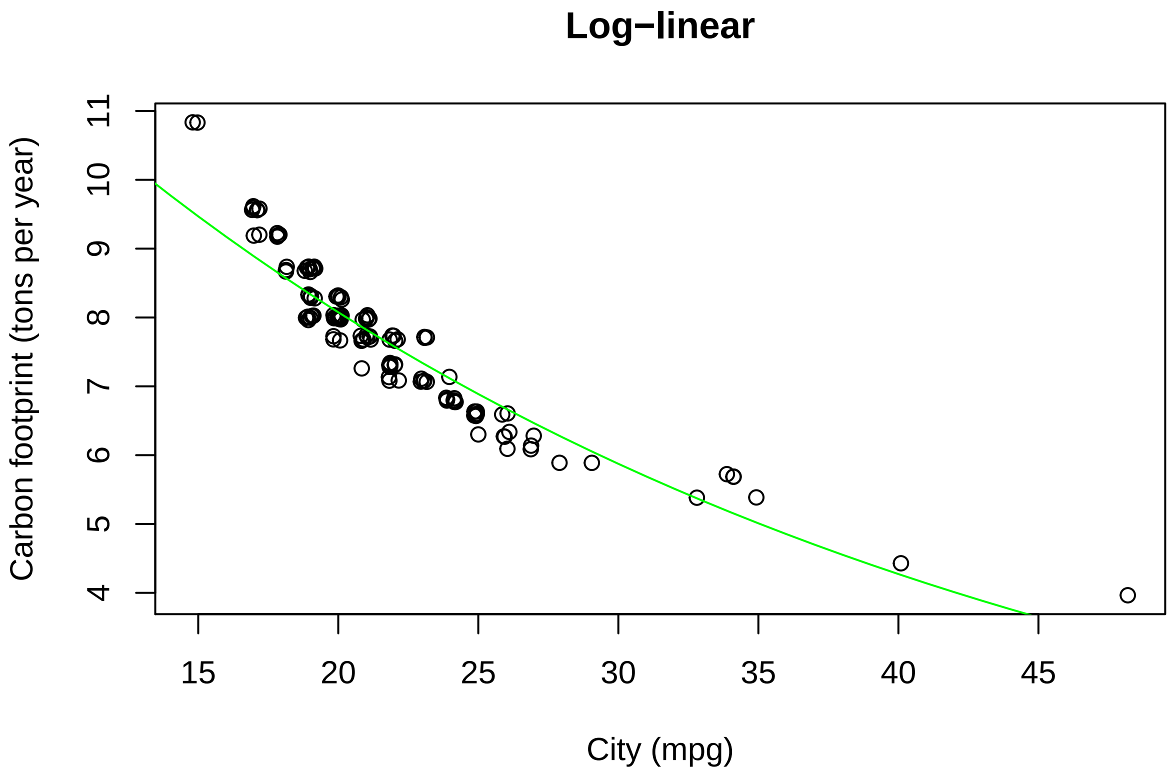

```
plot(jitter(Carbon) ~ jitter(City),
     xlab='City (mpg)',
     ylab='Carbon footprint (tons per year)',
     main='Log-log',
     data=fuel)
lines(1:50, exp(fit2$coef[1] + fit2$coef[2] * log(1:50)), col='red')
```




```
plot(jitter(Carbon) ~ jitter(City),
     xlab='City (mpg)',
     ylab='Carbon footprint (tons per year)',
     main='Linear-log',
     data=fuel)
lines(1:50, fit3$coef[1] + fit3$coef[2] * log(1:50), col='blue')
```




```r
plot(jitter(Carbon) ~ jitter(City),
     xlab='City (mpg)',
     ylab='Carbon footprint (tons per year)',
     main='Log-linear',
     data=fuel)
lines(1:50, exp(fit4$coef[1] + fit4$coef[2] * (1:50)), col='green')
```



Model	Functional Form
linear	$y = \beta_0 + \beta_1 x$
log-log	$\log y = \beta_0 + \beta_1 \log x$
linear-log	$y = \beta_0 + \beta_1 \log x$
log-linear	$\log y = \beta_0 + \beta_1 x$

Chapter 8: ARIMA

Stationarity and differencing

Needs to be stationary so need to difference first. Differencing can be one of two type:

1. First order (or second) differencing.
2. Seasonal differencing.

When to difference can be somewhat subjective

Unit root tests

One way to see if differencing is require is to use a *unit root test*. These are basically unit tests. One popular one is *Augmented Dickey-Fuller (ADF) test*.

```
adf.test(x, alternative='stationary')
```

Null hypothesis is that the data is not stationary. So large p-value (using 5% threshold, larger than 0.05 is large) indicates that differencing is required.

A useful R function is `ndiffs()` to find out how many times the data needs to be differenced and `nsdiffs()` to find out how many times it needs seasonal differencing.

Example:

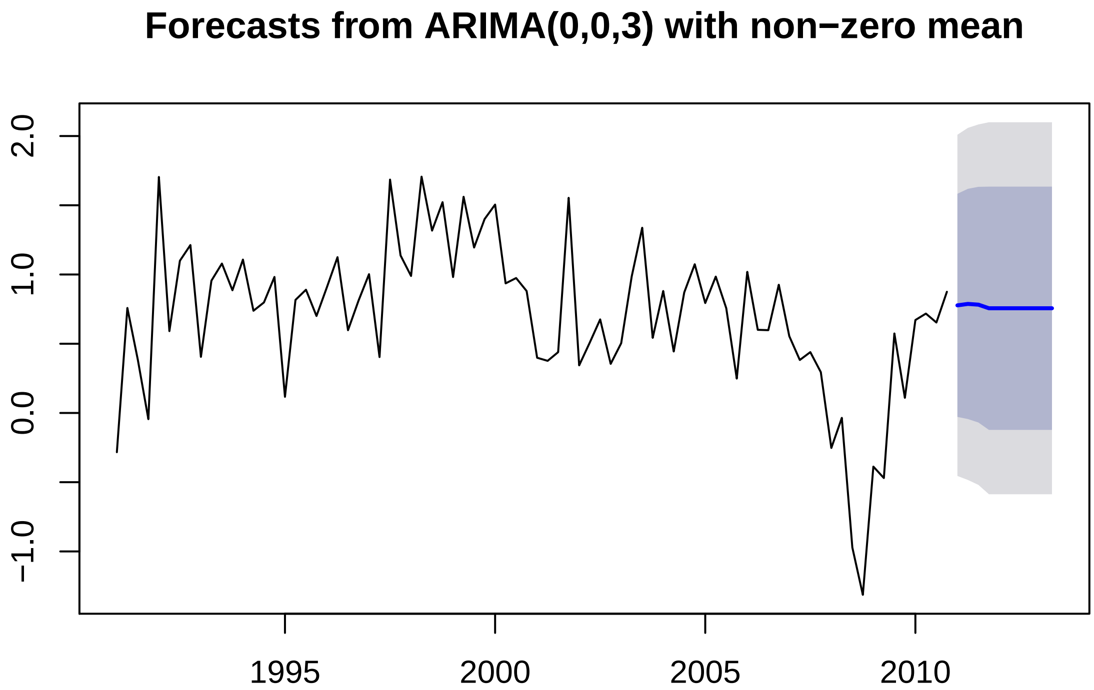
```
ns <- nsdiffs(x)
if(ns > 0) {
  xstar <- diff(x,lag=frequency(x),differences=ns)
} else {
  xstar <- x
}
nd <- ndiffs(xstar)
if(nd > 0) {
  xstar <- diff(xstar,differences=nd)
}
```

Non-seasonal ARIMA models

```
fit <- auto.arima(usconsumption[,1],seasonal=FALSE)
fit
```

```
## Series: usconsumption[, 1]
## ARIMA(0,0,3) with non-zero mean
##
## Coefficients:
##          ma1     ma2     ma3  intercept
##       0.2542  0.2260  0.2695     0.7562
## s.e.  0.0767  0.0779  0.0692     0.0844
##
## sigma^2 estimated as 0.3953:  log likelihood=-154.73
## AIC=319.46   AICc=319.84   BIC=334.96
```

```
plot(forecast(fit,h=10),include=80)
```



Auto ARIMA

The *auto.arima()* function can be useful but also dangerous. Be wary of the following:

- If c=0 and d=0, long-term forecasts will go to zero.
- If c=0 and d=1, will go to non-zero constant.
- If c=0 and d=2, will follow straight line.
- If c!=0 and d=0, will go to mean of data.
- If c!=0 and d=1, will follow a straight line.
- If c!=0 and d=2, will follow quadratic trend.

ACF and PACF

The ACF and PACF graphs may be helpful in finding the p and q values. If data are from a *ARIMA(p, d, 0)* or *ARIMA(0, d, q)*, then the ACF or PACF can be useful.

If the data follows *ARIMA(p, d, 0)* then the ACF and PACF plots of the differenced data will have the following patterns:

- the ACF is exponentially decaying or sinusoidal;
- there is a significant spike at lag p in PACF but not beyond that

if the data follows *ARIMA(0, d, q)* then the ACF and PACF plots of the differenced data will have the following patterns:

- the PACF is exponentially decaying or sinusoidal;
- there is a significant spike at lag q in ACF but not beyond that

ARIMA example (manual method)

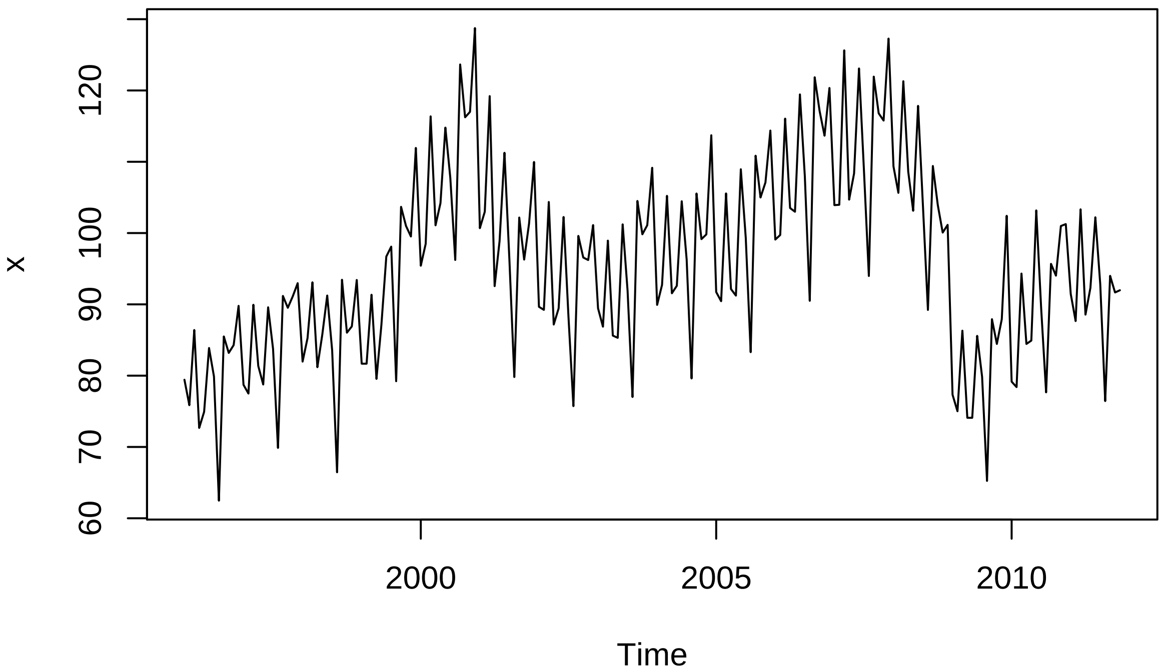
Process:

1. Plot data and identify unusual observations and patterns.
2. If necessary, use Box-Cox transformation to stabilize variance.
3. Difference the data until stationary. Use unit root test.
4. Plot ACF/PACF of differenced data to find order.
5. Try your model along with variations. Minimize the AICc.
6. Check residuals by plotting the ACF of residuals and conducting a portmanteau test.
7. If residuals look like white noise, then forecast using model.

Step 1: Plot data

Here is what the original data looks like.

```r
data(elecequip)
x <- elecequip
plot(x)
```



Step 2: Box-Cox transformation

Variance looks stable so no need for a Box-Cox transformation.

Step 3: Differencing

We'll use unit root tests to see if we need to difference.

```r
nsdiffs(x)
```

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

Function states we need to seasonaly difference once.

```r
m <- frequency(x)
x1 <- seasadj(stl(x, s.window='periodic'))
plot(x1)
```

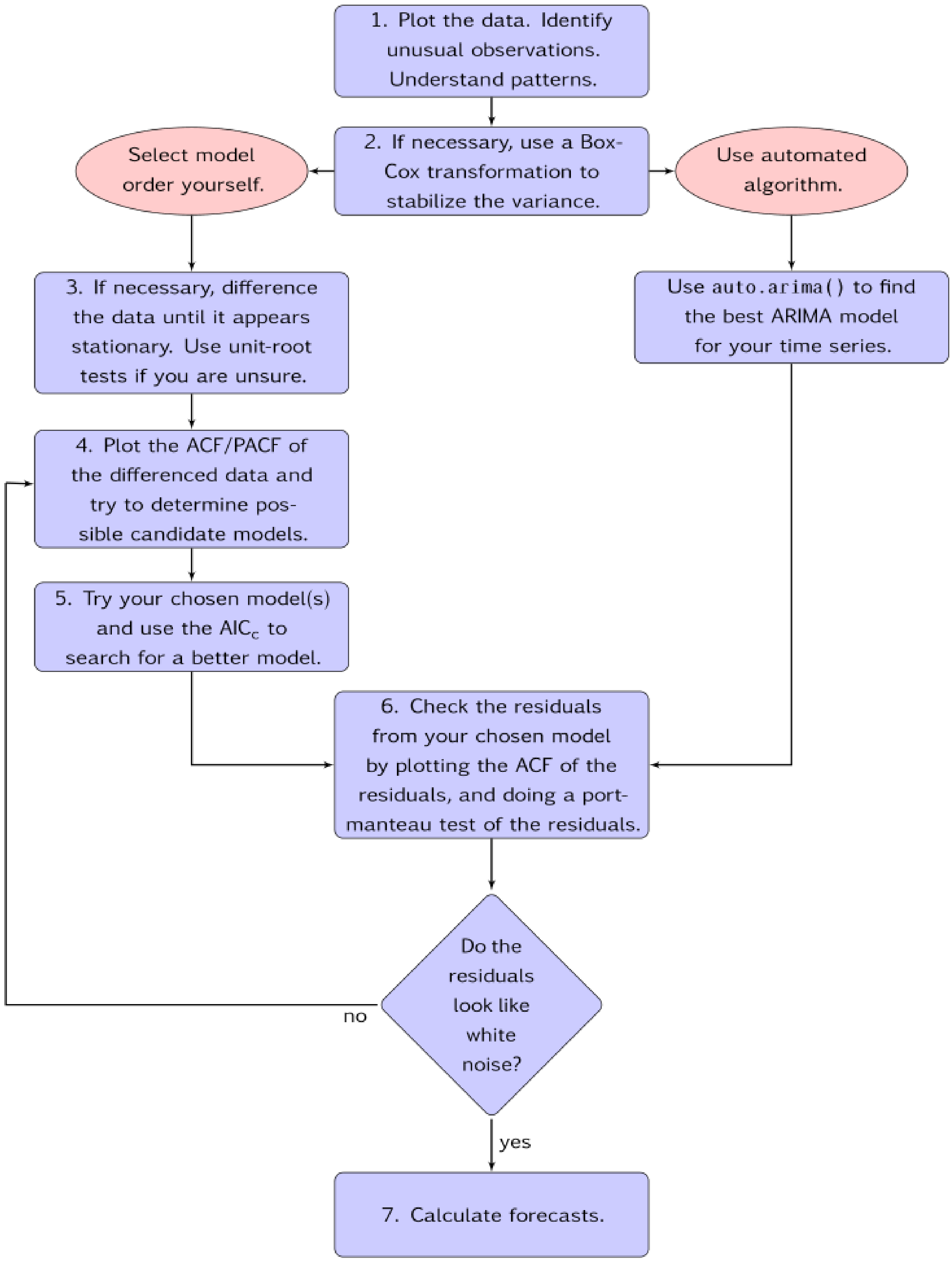
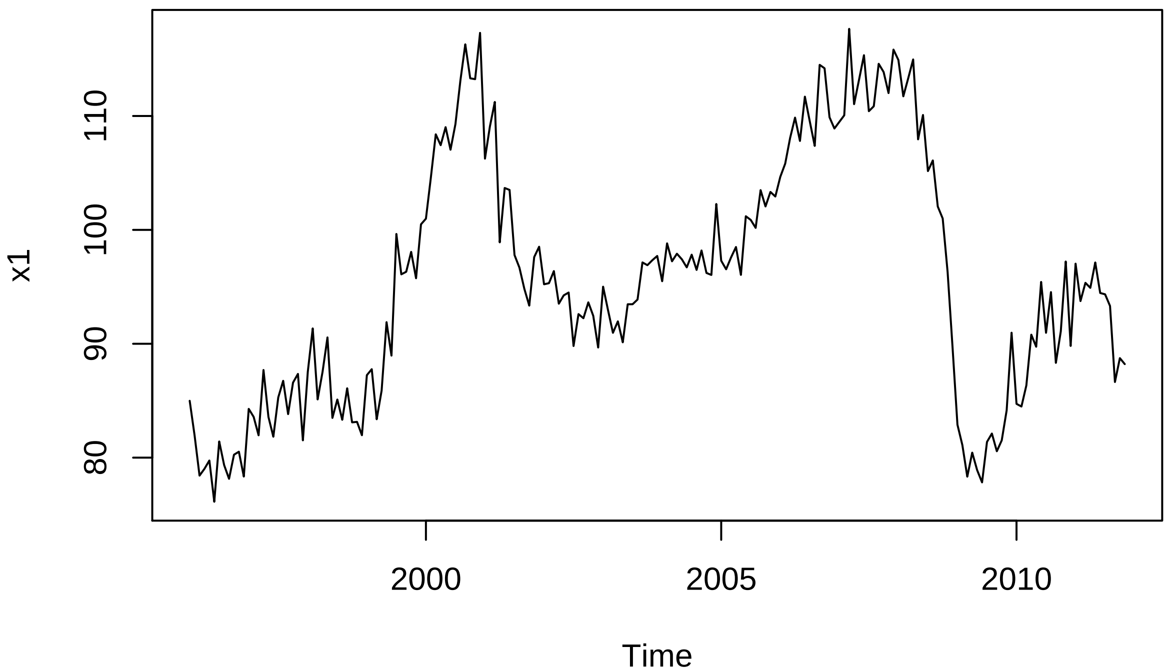
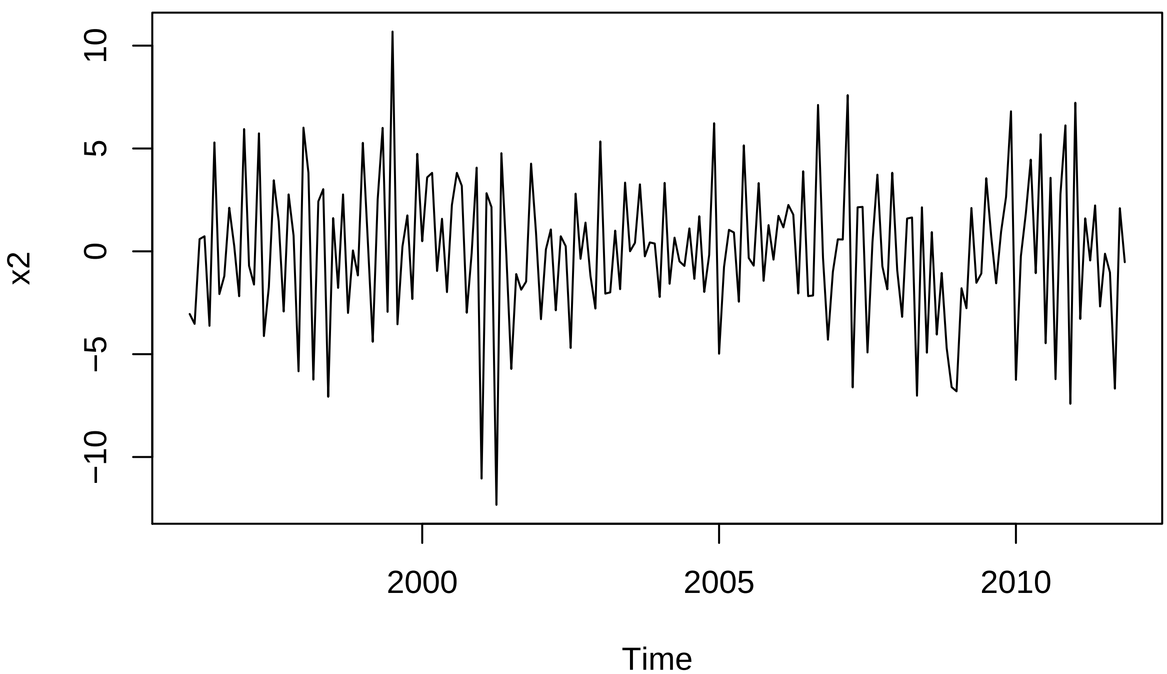

1. Plot the data. Identify unusual observations. Understand patterns.
2. If necessary, use a Box-Cox transformation to stabilize the variance.
Select model order yourself.
Use automated algorithm.
3. If necessary, difference the data until it appears stationary. Use unit-root tests if you are unsure.
Use auto.arima() to find the best ARIMA model for your time series.
4. Plot the ACF/PACF of the differenced data and try to determine possible candidate models.
5. Try your chosen model(s) and use the AICc to search for a better model.
6. Check the residuals from your chosen model by plotting the ACF of the residuals, and doing a portmanteau test of the residuals.
Do the residuals look like white noise?
no
yes
7. Calculate forecasts.

Figure 2:



Still not stationary. Let's run a first order difference.

```
x2 <- diff(x1)
plot(x2)
```



Better. Final check using unit root test.

```
nsdiffs(x2)
```

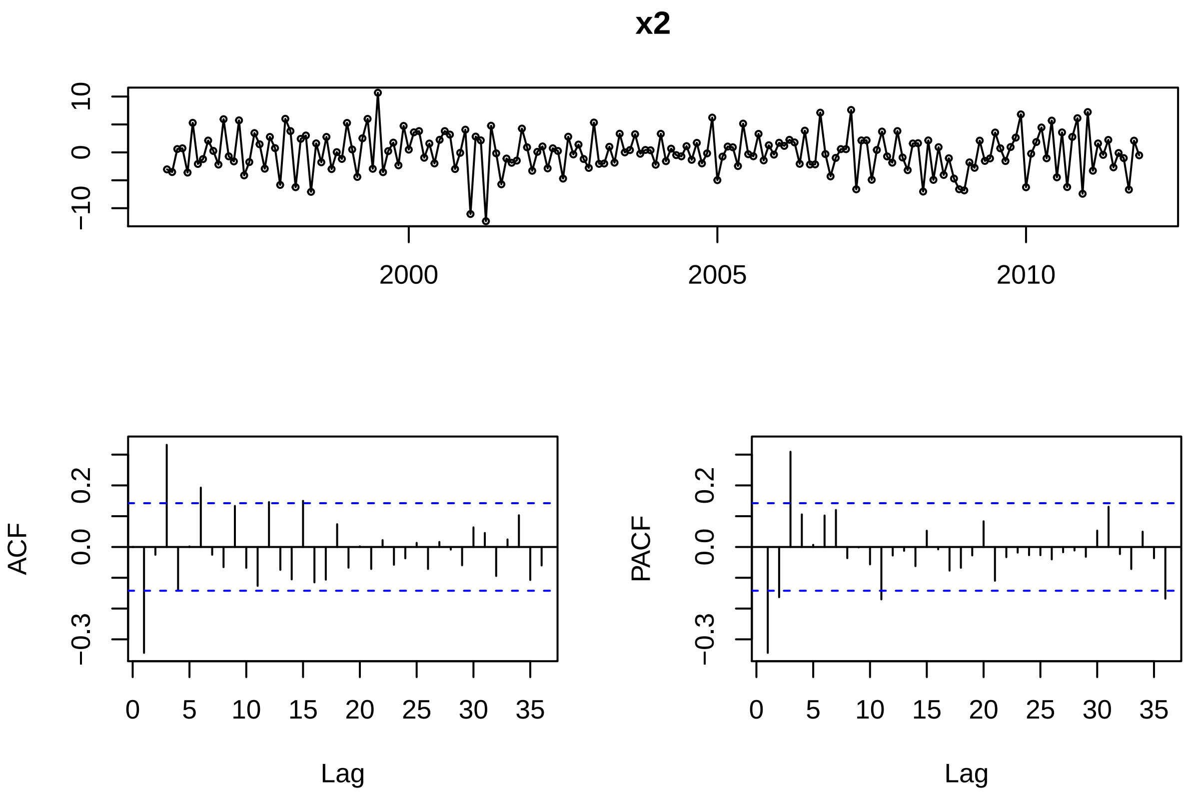
```
## [1] 0
```

```r
ndiffs(x2)
```

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

Step 4: ACF/PACF

```r
tsdisplay(x2)
```



ACF is sinusoidal. PACF shows spike up to lag 3. Potential order of c(3, 1, 0). Let's try that along with other variations such as:

- c(4, 1, 0)
- c(2, 1, 0)
- c(3, 1, 1)
- c(4, 1, 1)
- c(2, 1, 1)

Step 5: Minimize AICc

```r
summary(Arima(x1, order=c(3, 1, 0)))
```

```
## Series: x1
## ARIMA(3,1,0)
##
## Coefficients:
##           ar1      ar2     ar3
##       -0.3488  -0.0386  0.3139
## s.e.   0.0690   0.0736  0.0694
```

```
##
## sigma^2 estimated as 9.853:  log likelihood=-485.67
## AIC=979.33   AICc=979.55   BIC=992.32
##
## Training set error measures:
##                      ME     RMSE      MAE         MPE     MAPE      MASE
## Training set 0.01170679 3.105828 2.430723 -0.04353974 2.560168 0.2964478
##                     ACF1
## Training set -0.03463506
```

```
summary(Arima(x1, order=c(4, 1, 0)))
```

```
## Series: x1
## ARIMA(4,1,0)
##
## Coefficients:
##           ar1      ar2     ar3     ar4
##       -0.3847  -0.0341  0.3551  0.1138
## s.e.   0.0723   0.0731  0.0737  0.0728
##
## sigma^2 estimated as 9.777:  log likelihood=-484.45
## AIC=978.9   AICc=979.23   BIC=995.14
##
## Training set error measures:
##                       ME     RMSE      MAE         MPE     MAPE      MASE
## Training set 0.005571975 3.085604 2.401538 -0.04310566 2.528205 0.2928884
##                       ACF1
## Training set -0.0002297178
```

```
summary(Arima(x1, order=c(2, 1, 0)))
```

```
## Series: x1
## ARIMA(2,1,0)
##
## Coefficients:
##           ar1      ar2
##       -0.4019  -0.1660
## s.e.   0.0717   0.0717
##
## sigma^2 estimated as 10.87:  log likelihood=-495.34
## AIC=996.68   AICc=996.81   BIC=1006.42
##
## Training set error measures:
##                      ME     RMSE      MAE         MPE     MAPE      MASE
## Training set 0.02904185 3.270719 2.472795 -0.04743905 2.610552 0.3015789
##                    ACF1
## Training set 0.05216481
```

```
summary(Arima(x1, order=c(3, 1, 1)))
```

```
## Series: x1
## ARIMA(3,1,1)
##
## Coefficients:
##          ar1     ar2     ar3      ma1
##       0.0519  0.1191  0.3730  -0.4542
```

```
## s.e.  0.1840  0.0888  0.0679   0.1993
##
## sigma^2 estimated as 9.737:  log likelihood=-484.08
## AIC=978.17   AICc=978.49   BIC=994.4
##
## Training set error measures:
##                        ME     RMSE      MAE         MPE     MAPE      MASE
## Training set -0.001227744 3.079373 2.389267 -0.04290849 2.517748 0.2913919
##                     ACF1
## Training set 0.008928479
```

```
summary(Arima(x1, order=c(4, 1, 1)))
```

```
## Series: x1
## ARIMA(4,1,1)
##
## Coefficients:
##          ar1     ar2     ar3      ar4      ma1
##       0.1805  0.1626  0.3761  -0.0657  -0.5752
## s.e.  0.2360  0.1015  0.0687   0.1155   0.2224
##
## sigma^2 estimated as 9.775:  log likelihood=-483.93
## AIC=979.87   AICc=980.33   BIC=999.35
##
## Training set error measures:
##                        ME     RMSE     MAE         MPE     MAPE     MASE
## Training set -0.001948234 3.076947 2.39058 -0.04216591 2.519967 0.291552
##                      ACF1
## Training set 0.0004688917
```

```
summary(Arima(x1, order=c(2, 1, 1)))
```

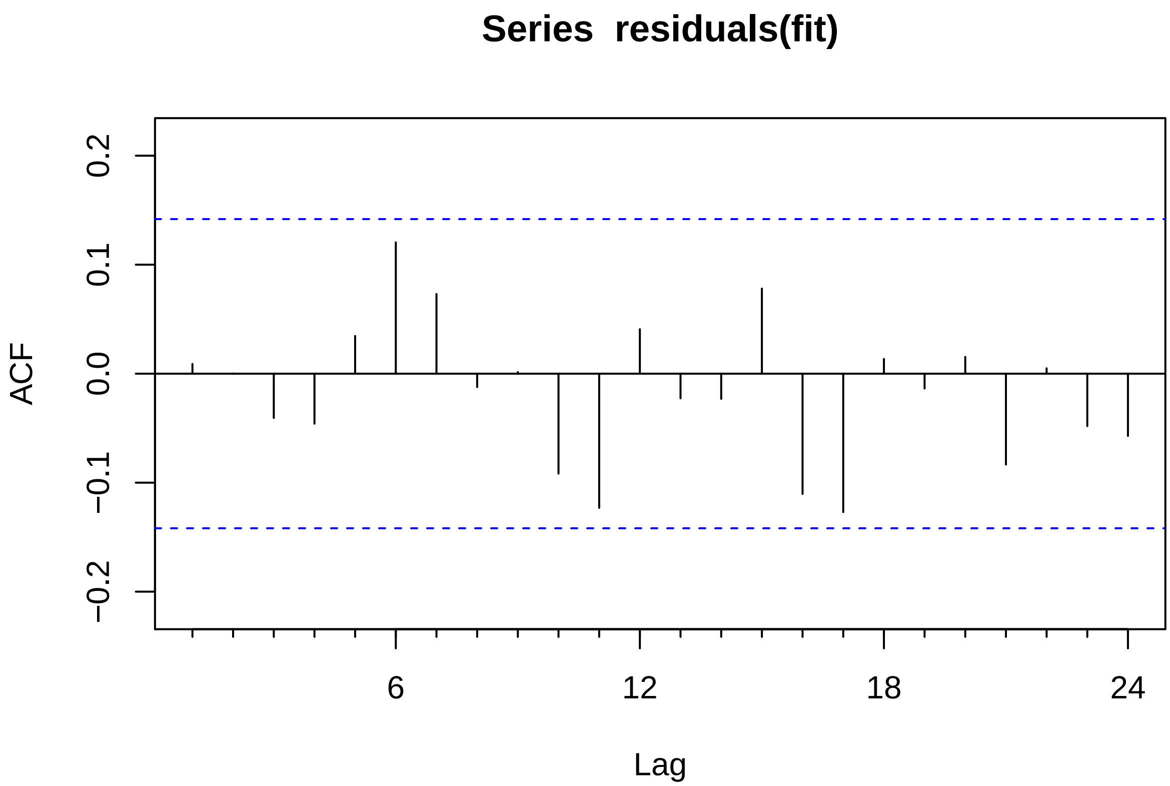
```
## Series: x1
## ARIMA(2,1,1)
##
## Coefficients:
##           ar1      ar2     ma1
##       -0.9015  -0.3842  0.5147
## s.e.   0.1544   0.0726  0.1573
##
## sigma^2 estimated as 10.52:  log likelihood=-491.76
## AIC=991.52   AICc=991.74   BIC=1004.51
##
## Training set error measures:
##                      ME     RMSE      MAE         MPE     MAPE      MASE
## Training set 0.02630035 3.208853 2.445192 -0.04627073 2.580595 0.2982124
##                    ACF1
## Training set 0.04285319
```

Looks like order (3, 1, 1) has the smallest AICc of 978.49.

```
fit <- Arima(x1, order=c(3, 1, 1))
```

Step 6: Check residuals

```r
Acf(residuals(fit))
```



ACF looks good. Let's try the Portmanteau test.

```r
Box.test(residuals(fit), lag=24, fitdf=4, type='Ljung')
```

```
##
##  Box-Ljung test
##
## data:  residuals(fit)
## X-squared = 20.496, df = 20, p-value = 0.4273
```

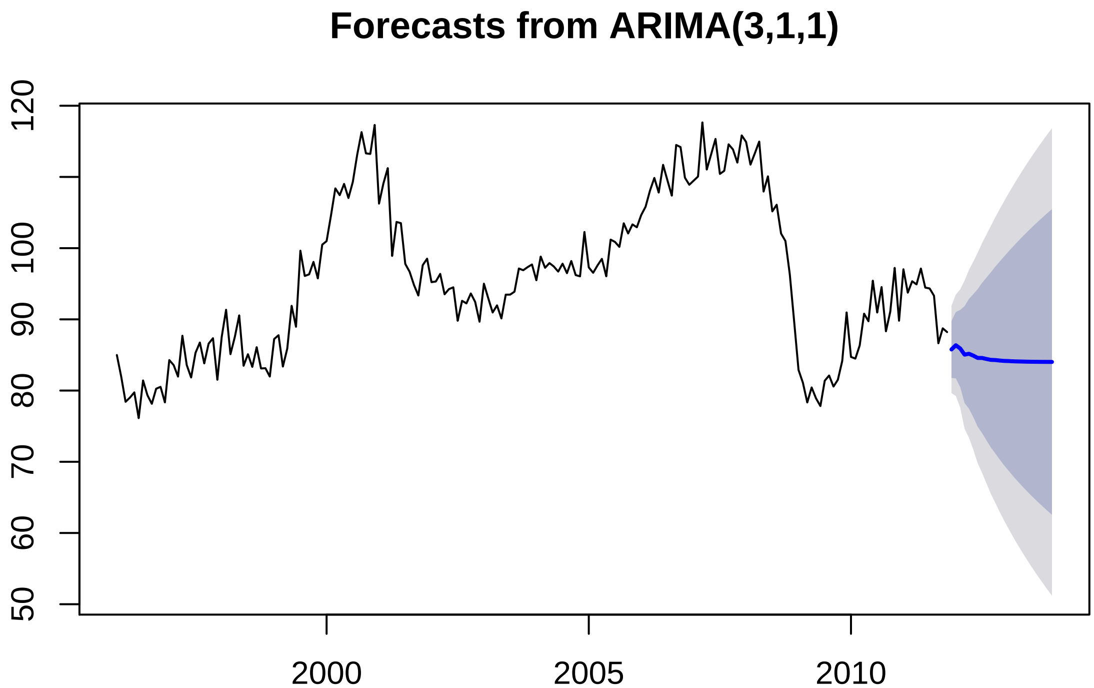
p-value is large so just white noise.

Recommended lag is 10 for non-seasonal data and $2m$ for seasonal where m is a period of seasonality. This can't be too large, so if larger than $T/5$ where T is the number of observations, the just use $T/5$.

`fitdf` is just the sum of p and q.

Step 7: Forecast

```r
plot(forecast(fit))
```



`auto.arima()` would have returned the same thing.

```
auto.arima(x1, seasonal=FALSE)
```

```
## Series: x1
## ARIMA(3,1,1)
##
## Coefficients:
##          ar1     ar2     ar3      ma1
##       0.0519  0.1191  0.3730  -0.4542
## s.e.  0.1840  0.0888  0.0679   0.1993
##
## sigma^2 estimated as 9.737:  log likelihood=-484.08
## AIC=978.17   AICc=978.49   BIC=994.4
```