# "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

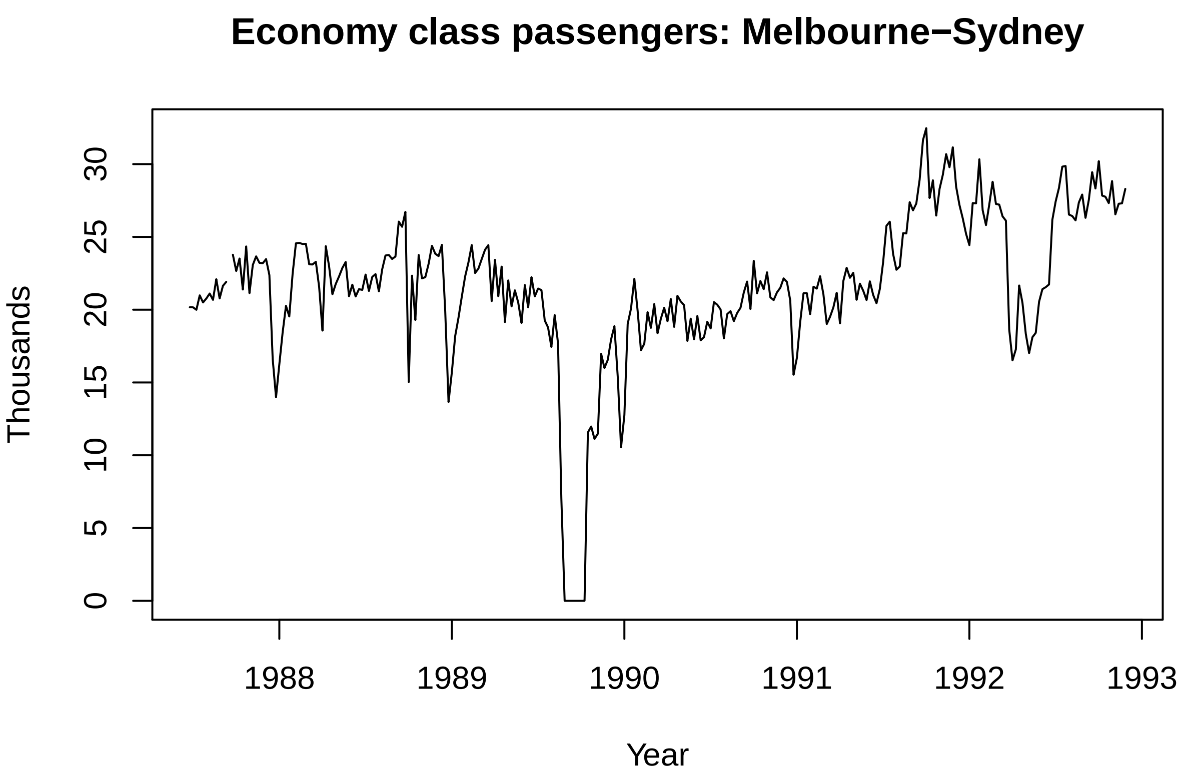
First thing to do for any forecasting exercise is to plot the data to look for patterns or any abnormalities.

### Time Plots

aka Line graphs.

#### Example 1

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

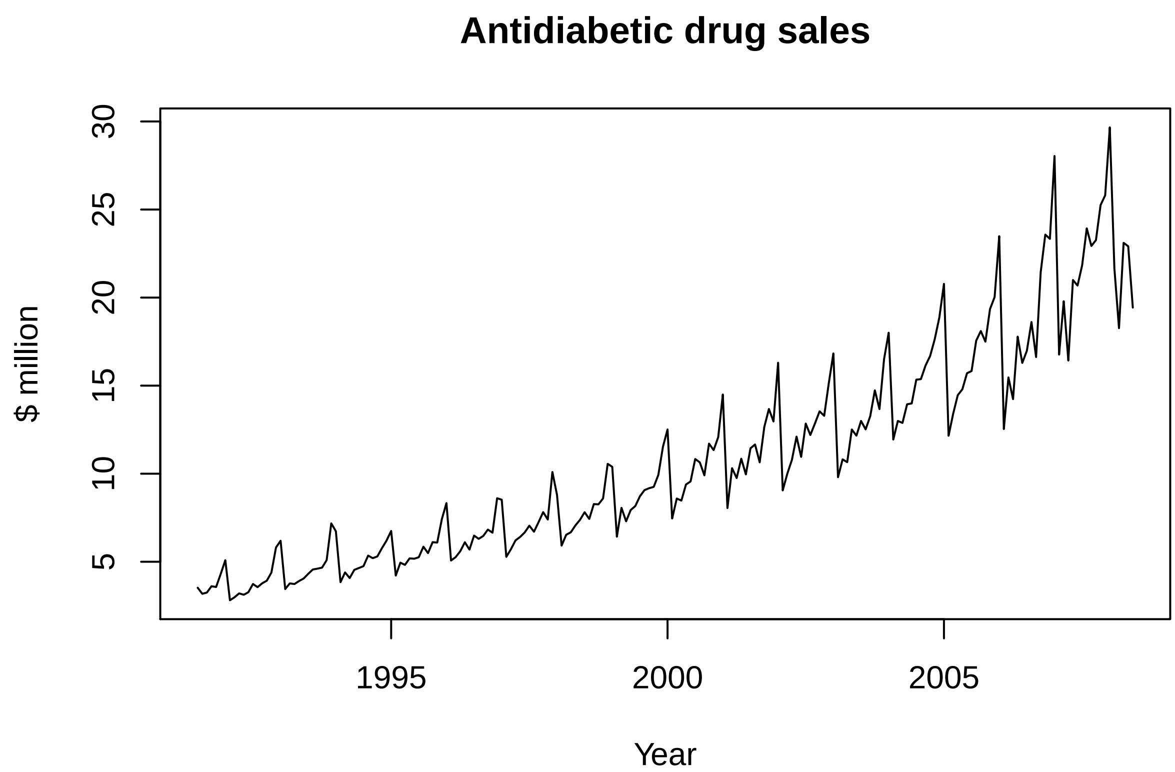



Notes:

- 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

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




Notes:

- Seasonality
- Upward trend

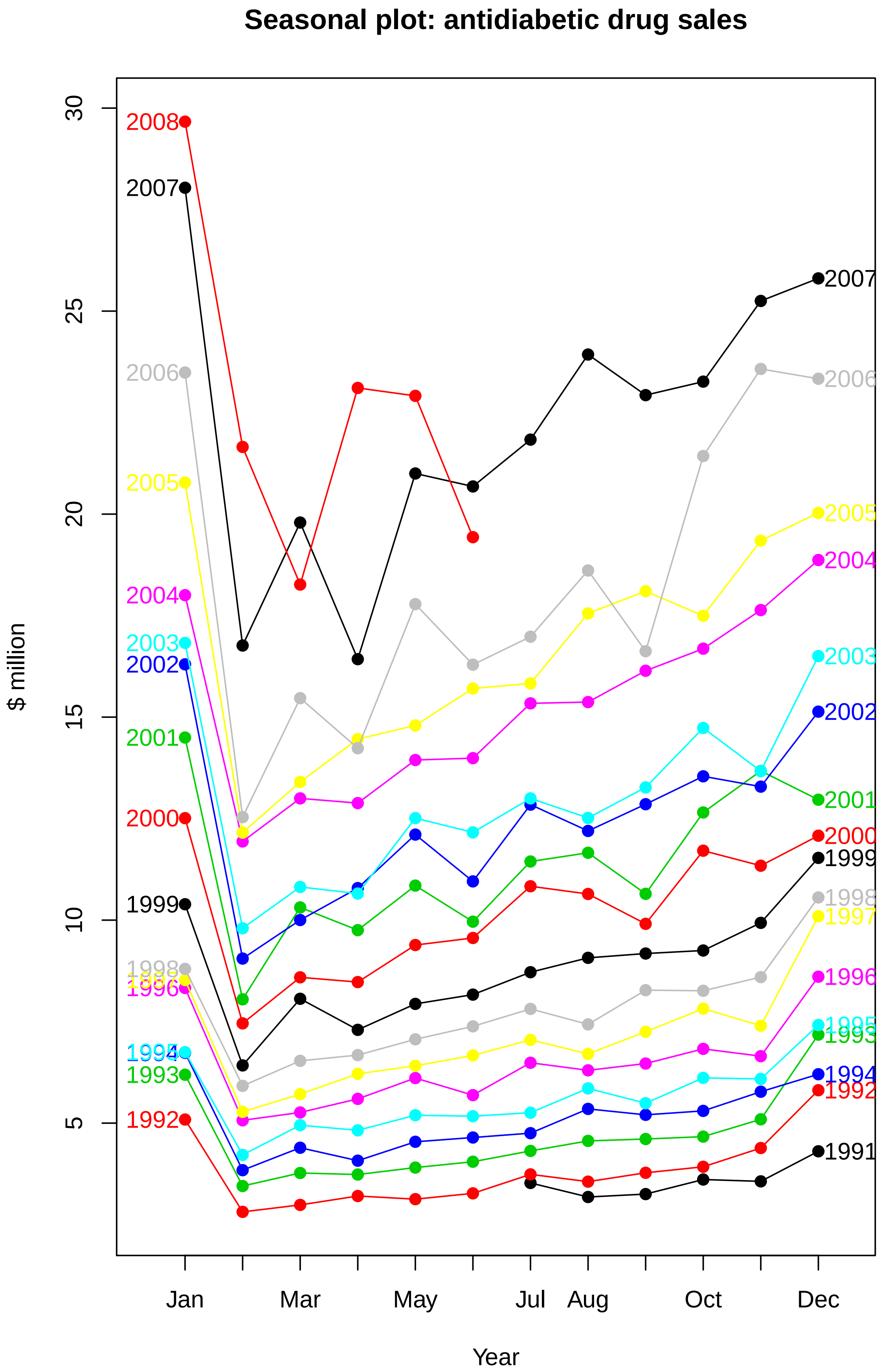
### Common Time Series Patterns

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

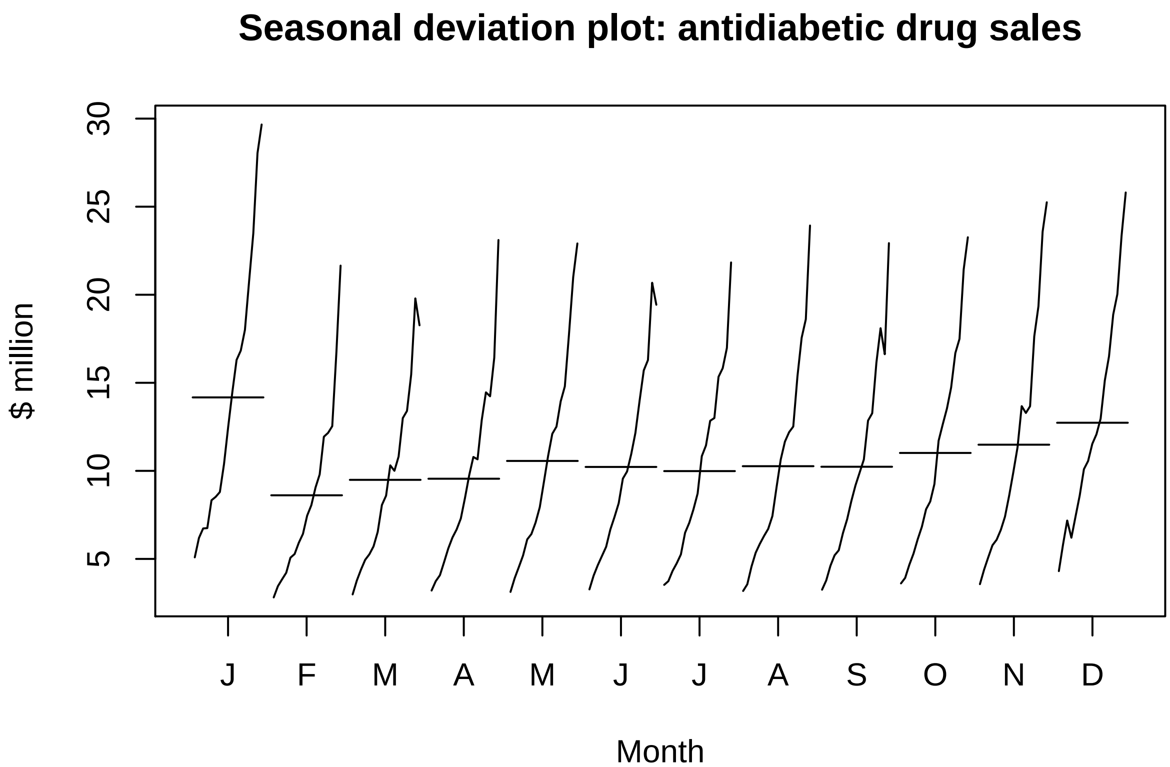
### Seasonal Plots

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
30
25
20
15
10
5
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**

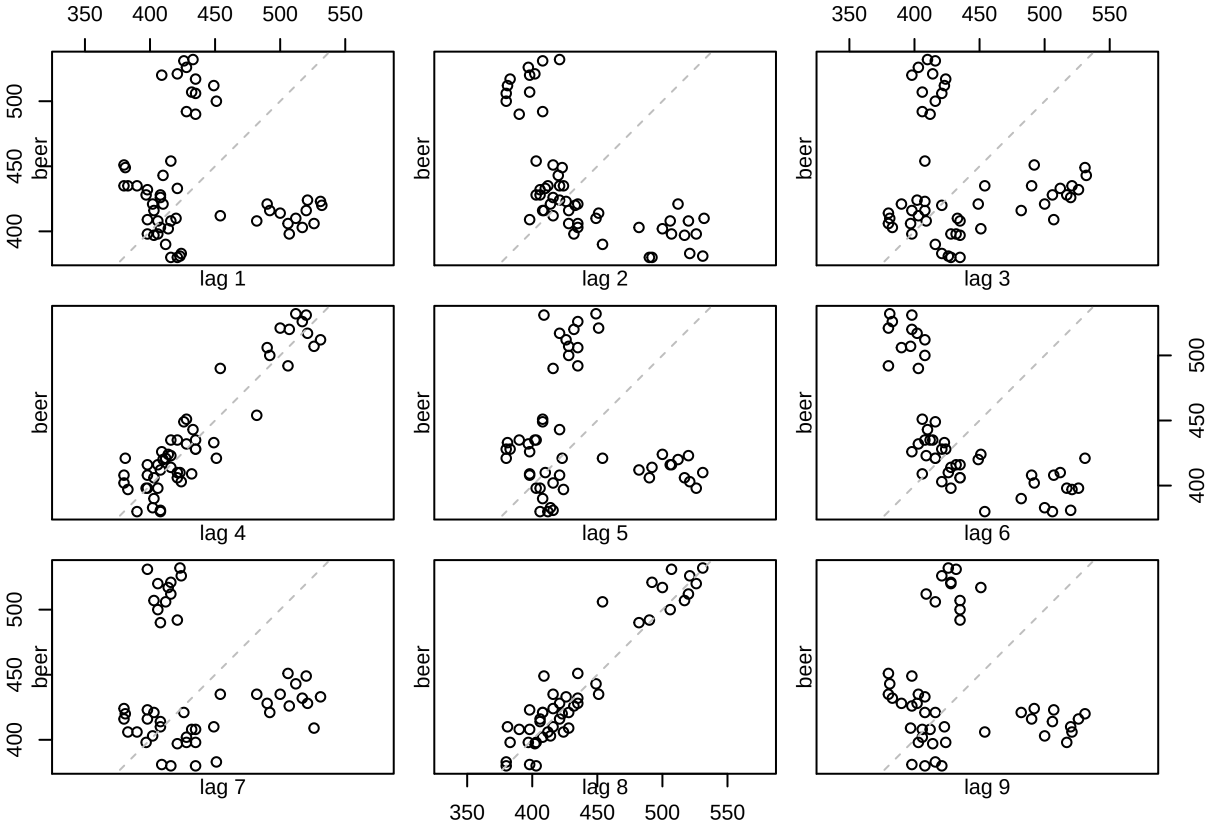
**Correlation coefficient**: $r$

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

**Autocorrelation**

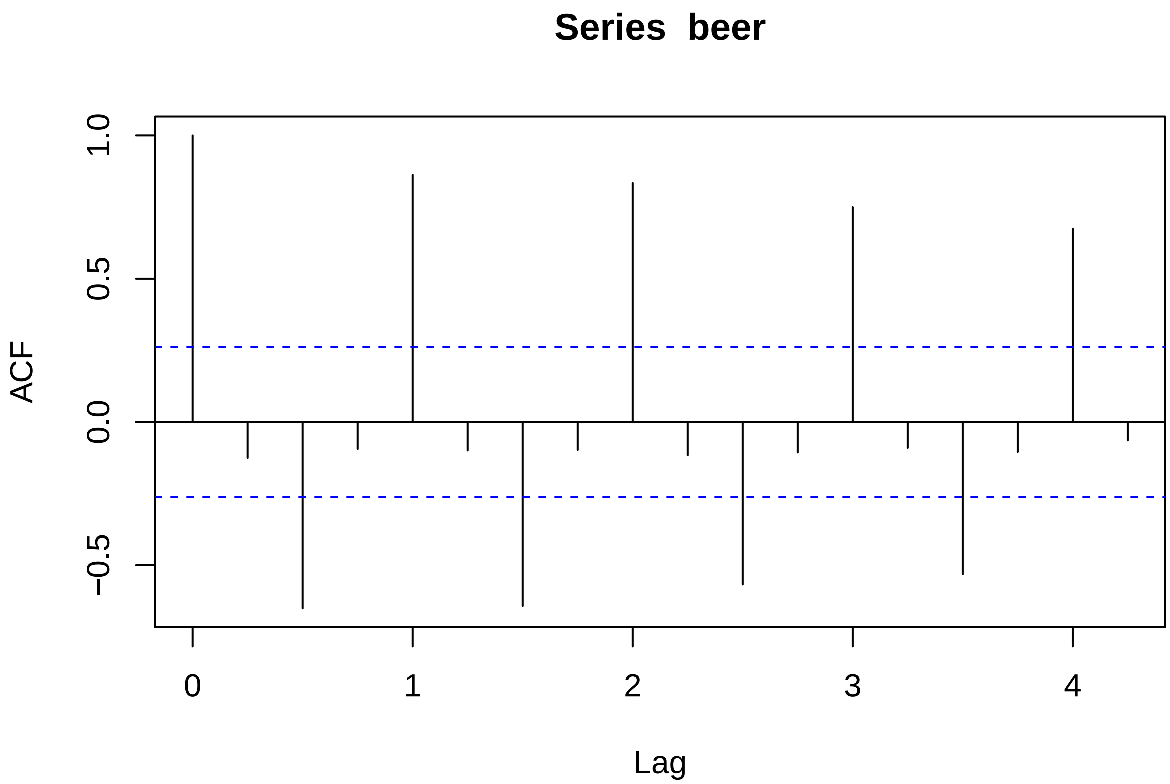
Used to test correlation on lag. $r_1$ tests correlation on lag 1, $r_2$ tests correlation on lag 2, etc.

```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.

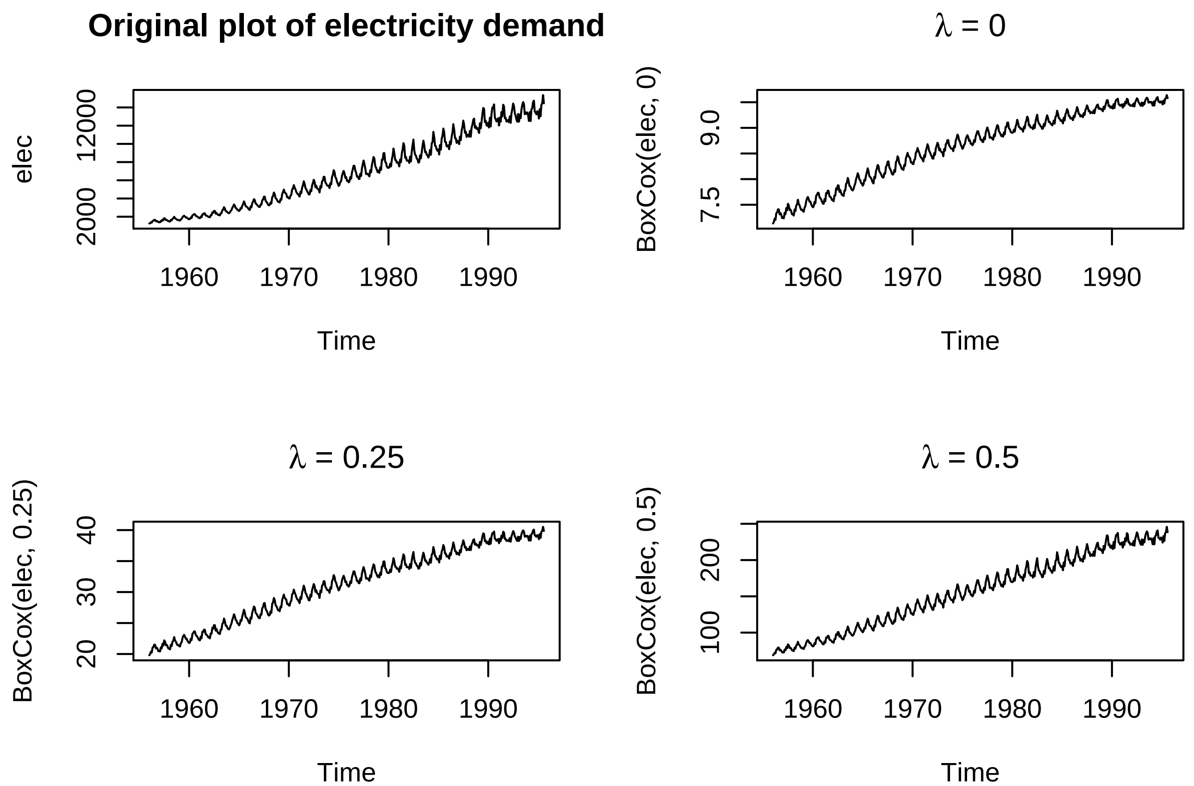
## Simple forecasting methods

- **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 $\lambda$ is 0, it does a natural log, otherwise it does a power transformation. $\lambda$ 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 $\lambda$ 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

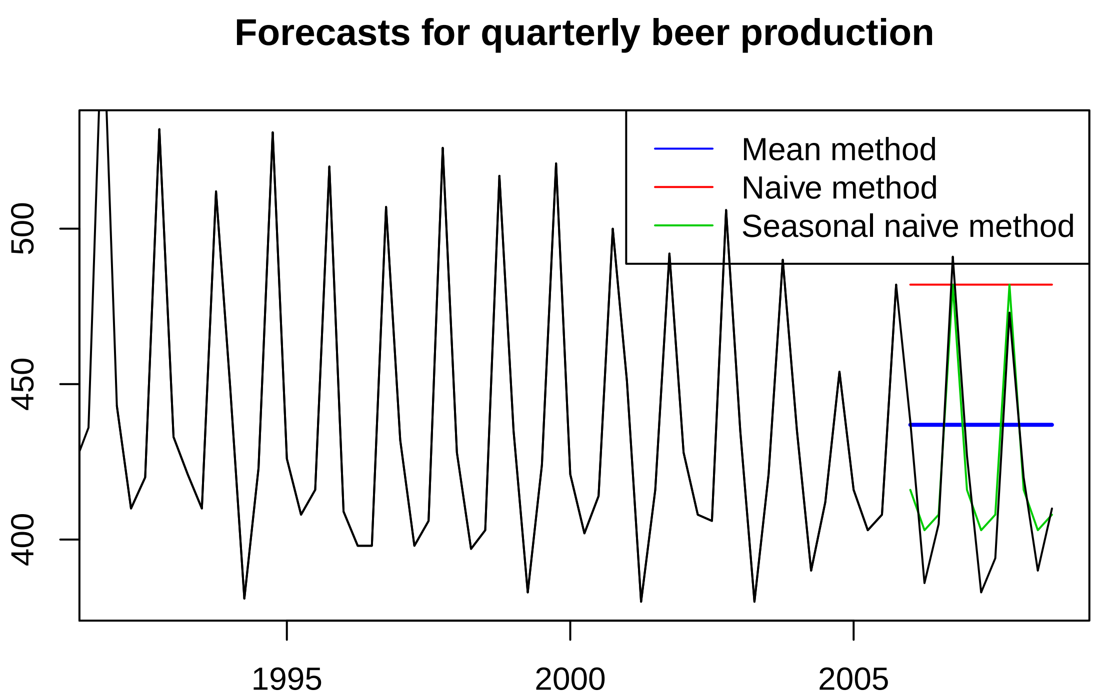
Scaled errors are used as an alternative to percentage errors. The *mean absolute scaled error* or *MASE* is a commonly used one (alternatively *mean squared scaled error* or *MSSE* is used).

### 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**.

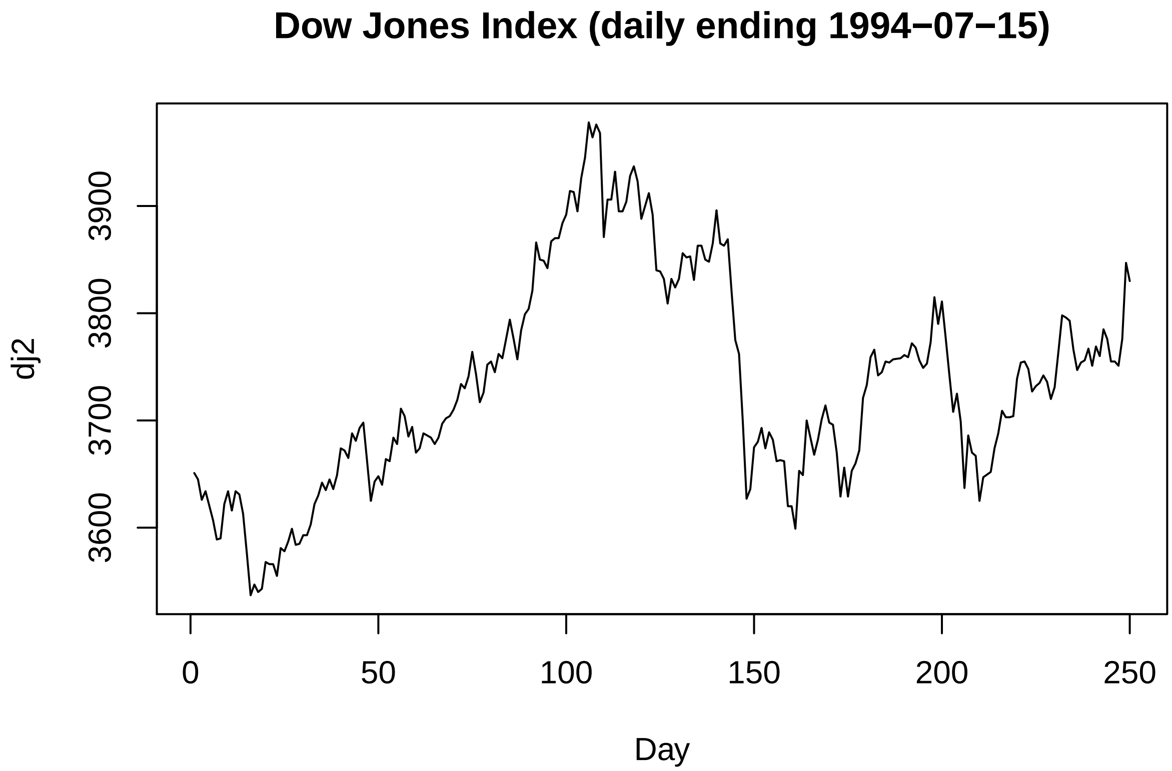
The following two properties are not required, but useful:

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

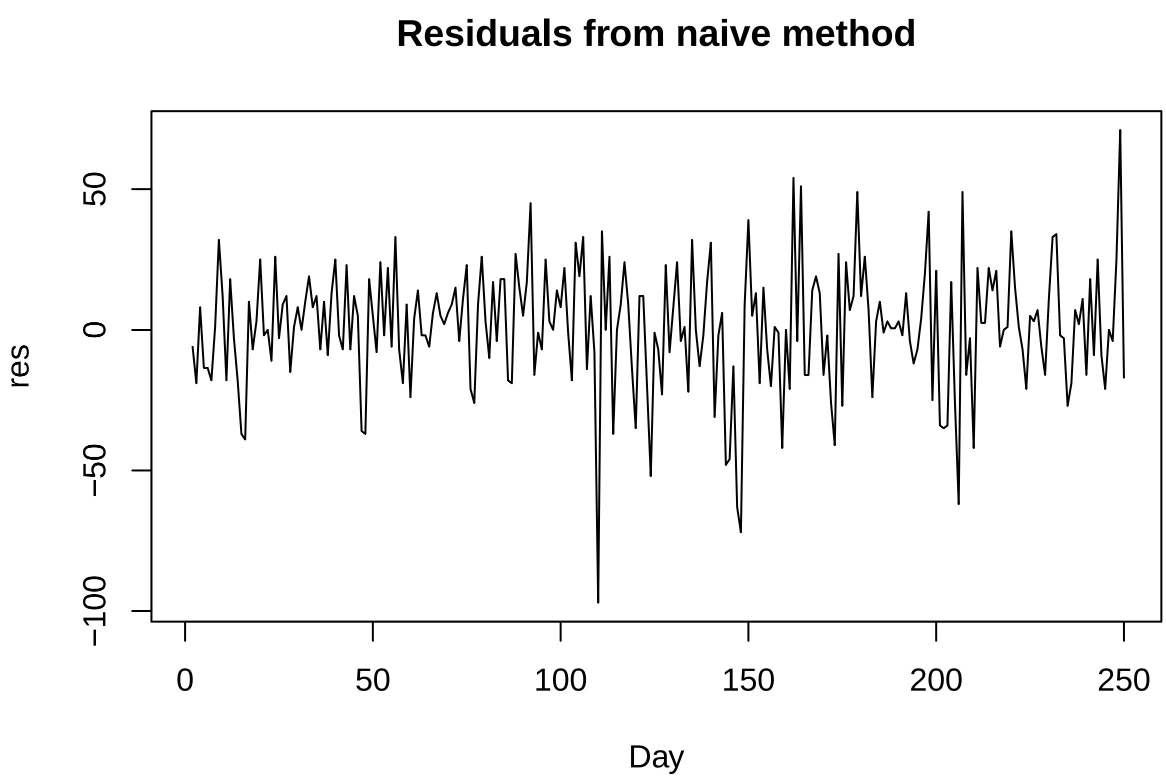
### Example: Dow Jones

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

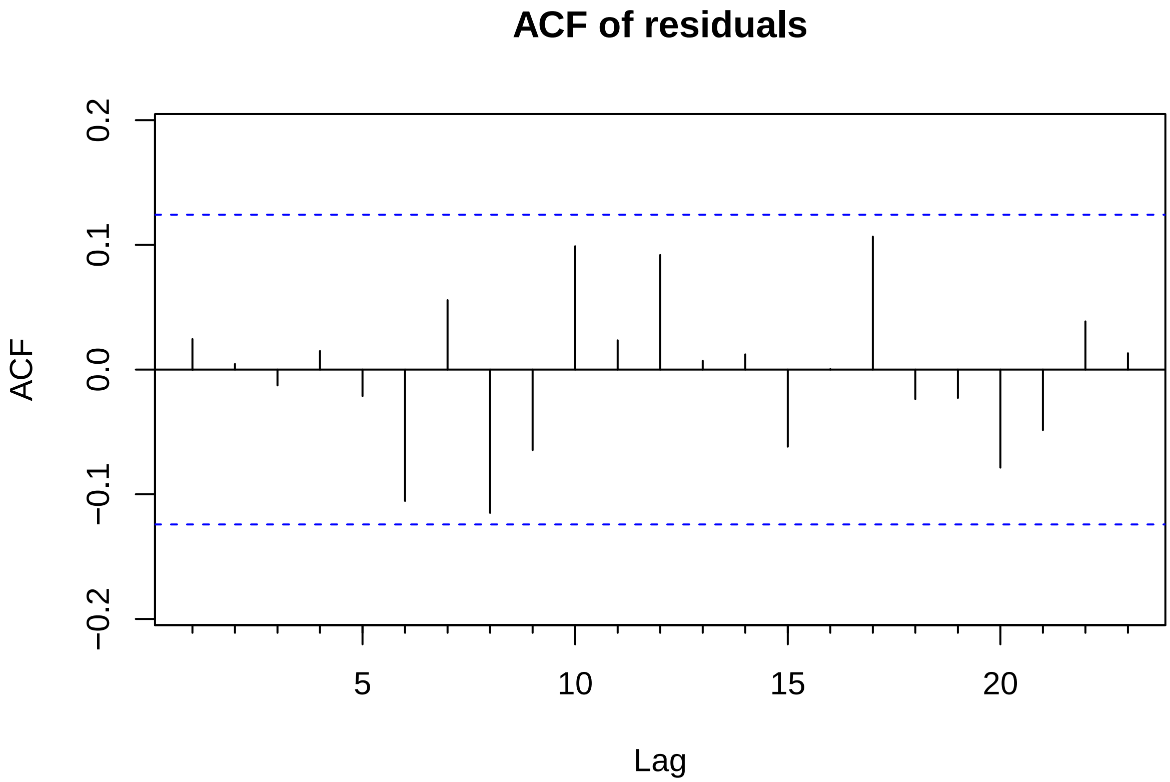
```r
data(dj)
# par(mfrow=c(4, 1))
dj2 <- window(dj, end=250)
plot(dj2,
     main='Dow Jones Index (daily ending 1994-07-15)',
     xlab='Day')
```



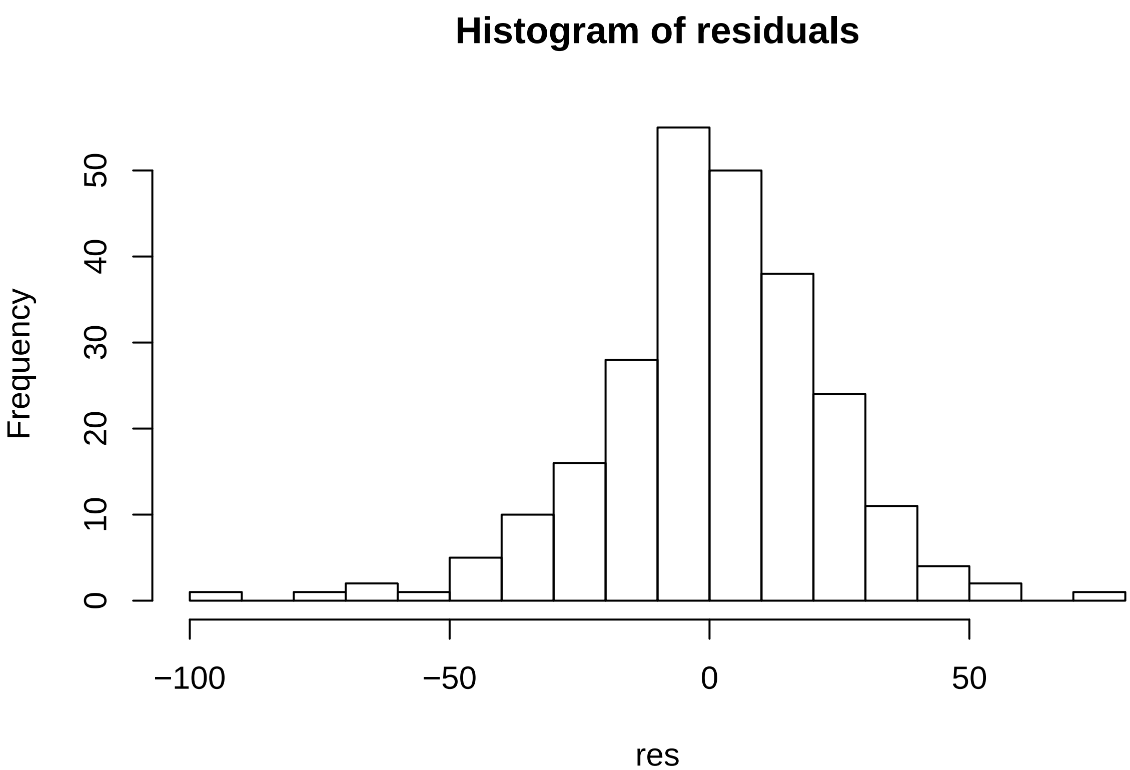

```r
res <- residuals(naive(dj2))
plot(res,
     main='Residuals from naive method',
     xlab='Day')
```



```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

### Simple regression

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

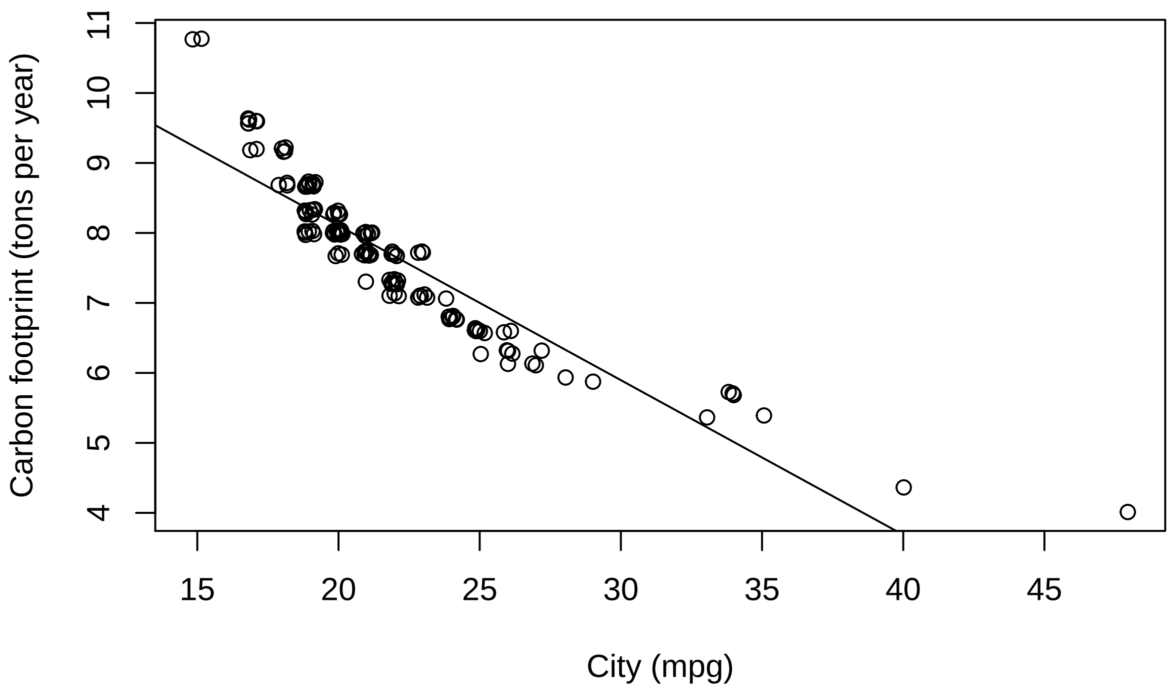
$$\sum_{i=1}^{N} \epsilon_i^2$$

The correlation coefficient $r$ shows how much $x$ predicts $y$.

#### Example: Car emission

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

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



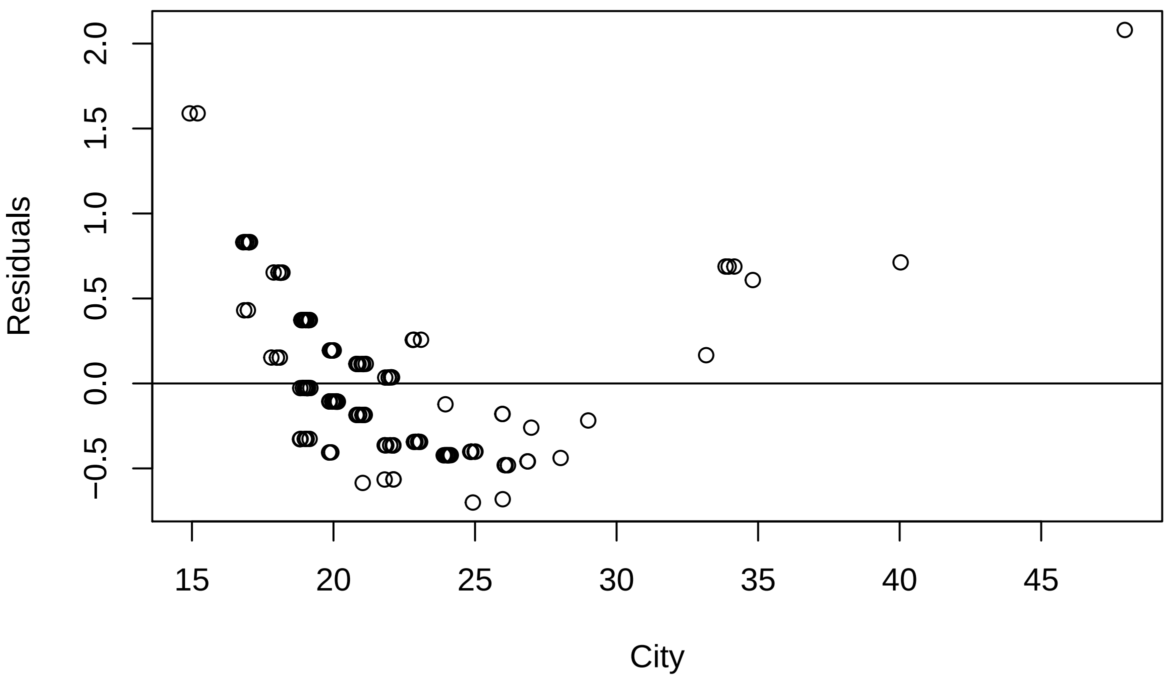

```r
summary(fit)
```

```
##
## 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)
```

```r
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.

### Non-linear regression

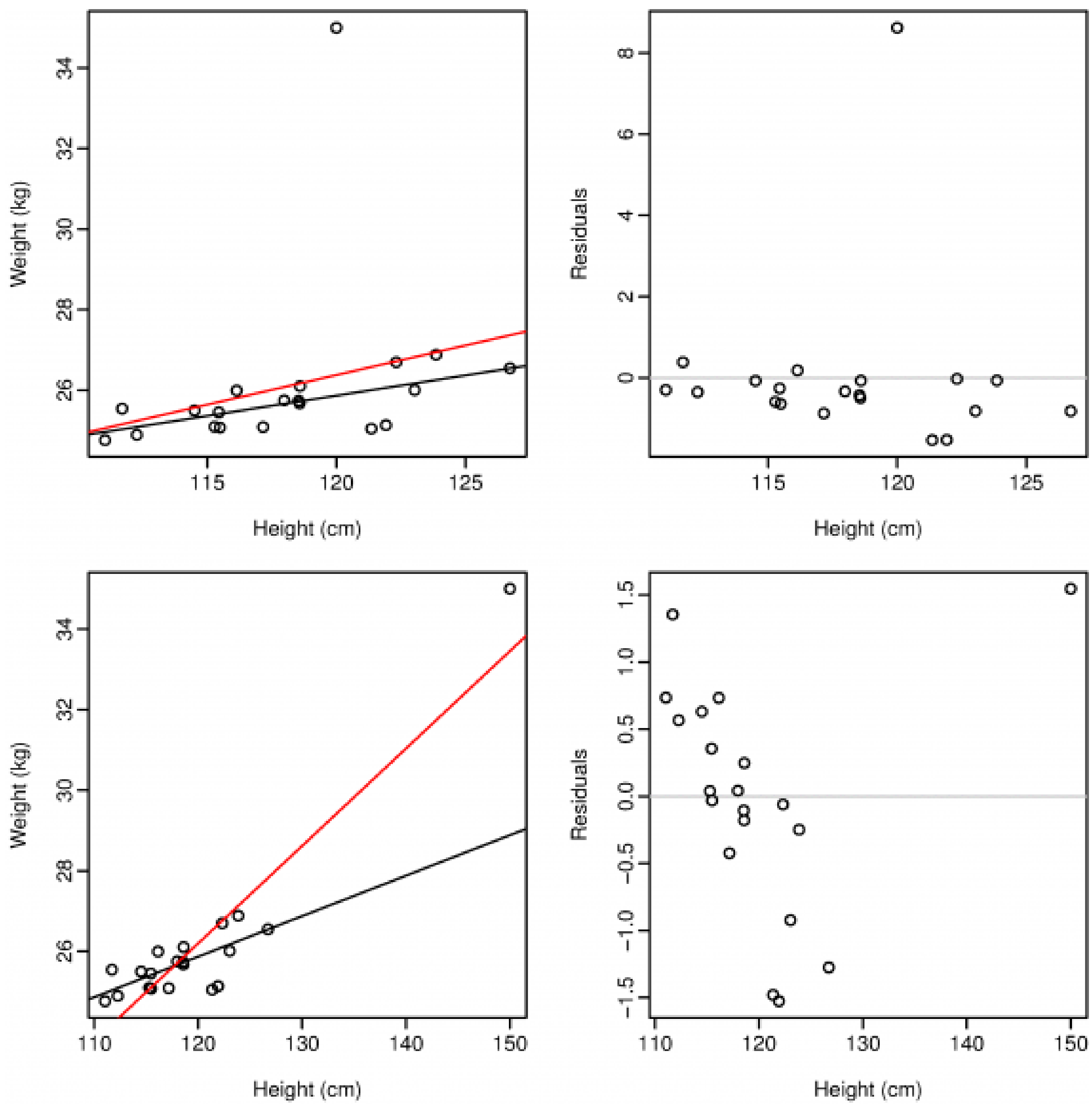
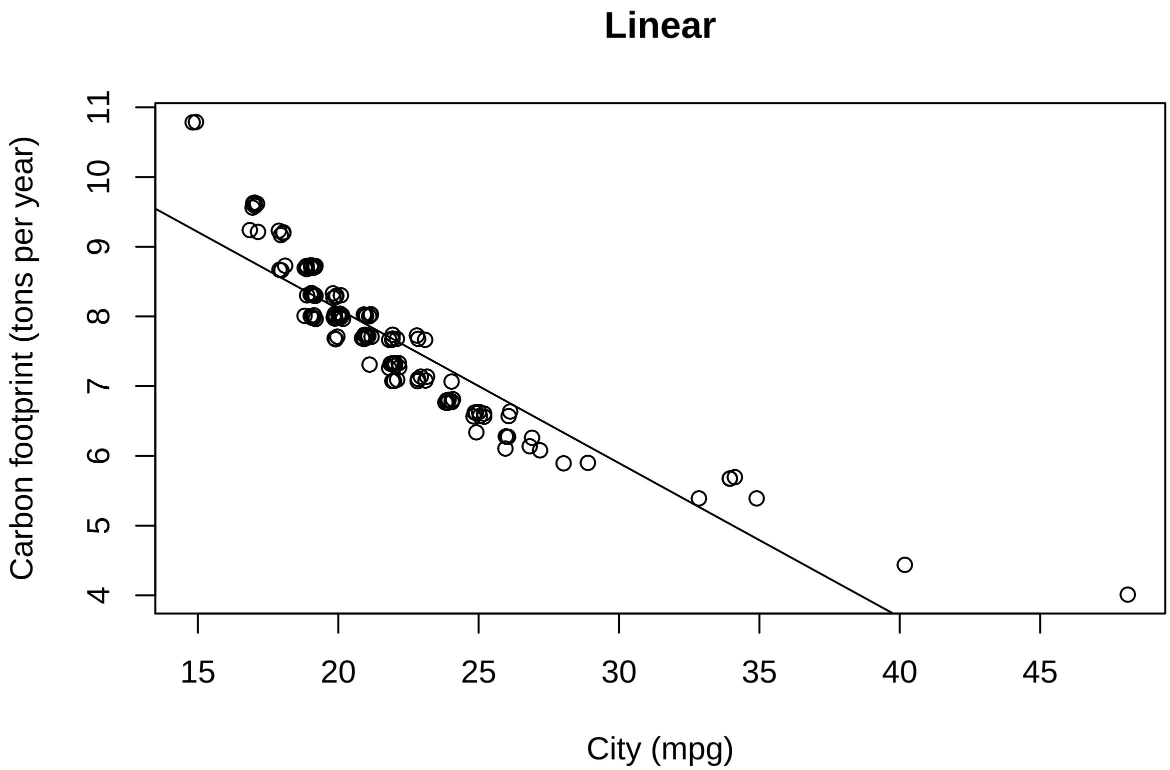
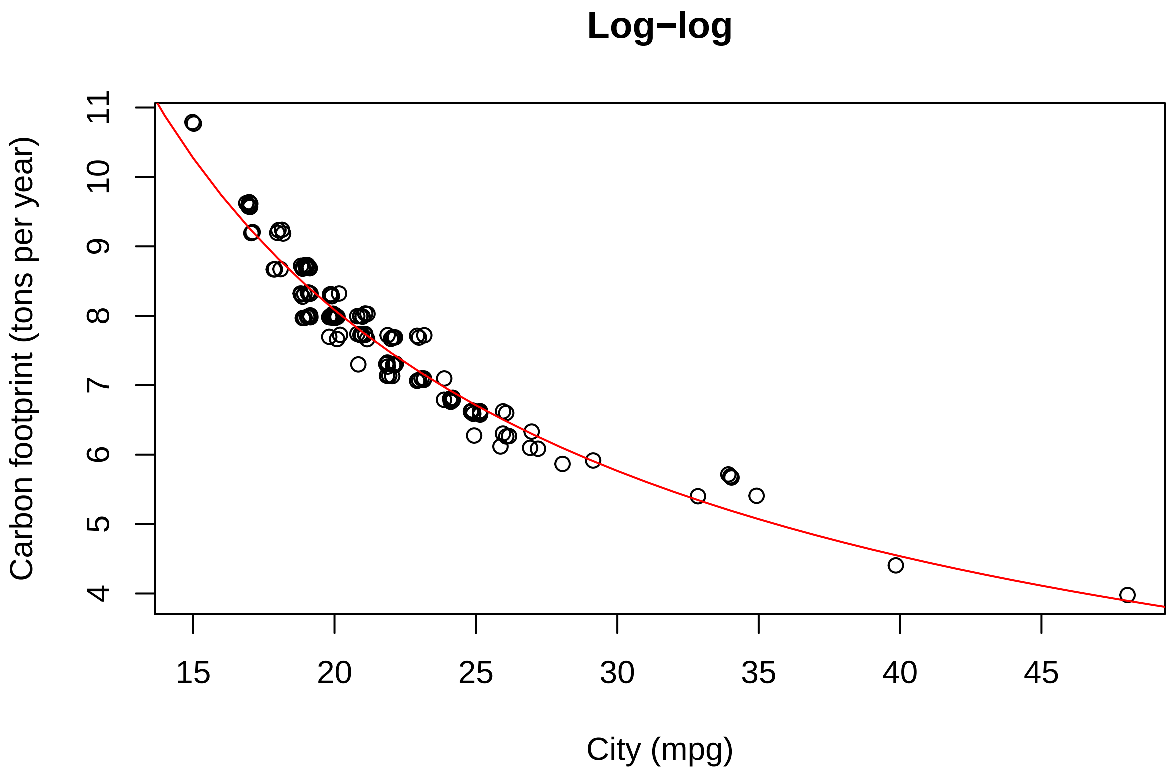



Figure 1: Weight vs height

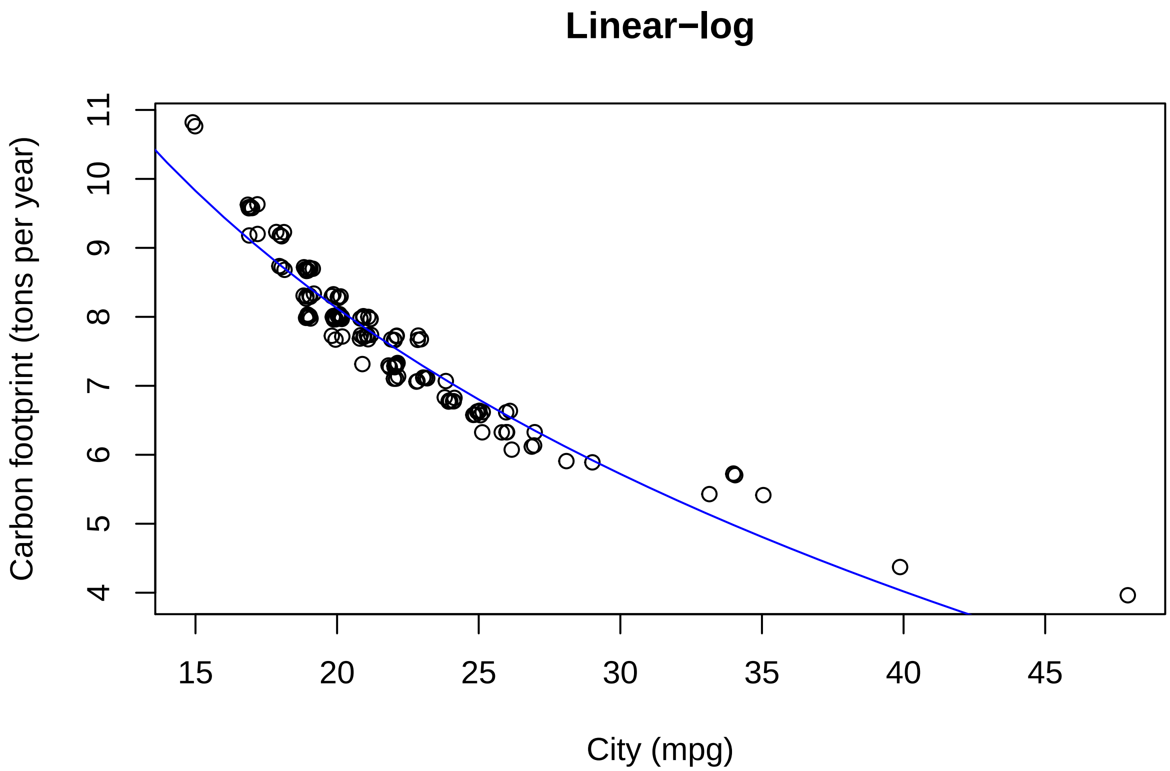
```r
fit <- lm(Carbon ~ City, data=fuel)
fit2 <- lm(log(Carbon) ~ log(City), data=fuel)
fit3 <- lm(Carbon ~ log(City), data=fuel)
fit4 <- lm(log(Carbon) ~ City, data=fuel)
plot(jitter(Carbon) ~ jitter(City),
     xlab='City (mpg)',
     ylab='Carbon footprint (tons per year)',
     main='Linear',
  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



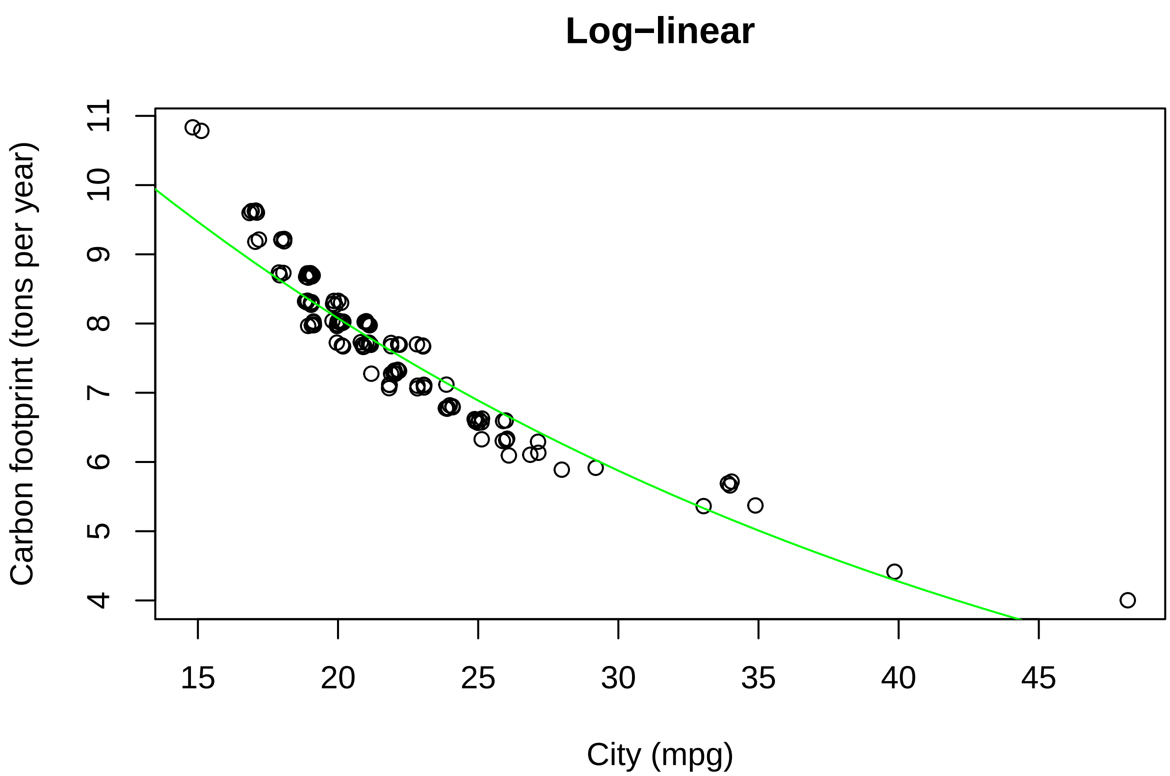

```r
plot(jitter(Carbon) ~ jitter(City),
     xlab='City (mpg)',
     ylab='Carbon footprint (tons per year)',
     main='Log-log',
     data=fuel)
lines(1:50, exp(fit2$coef[1] + fit2$coef[2] * log(1:50)), col='red')
```




```
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')
```




```
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 | $\log y = \beta_0 + \beta_1 x$ |