Regression modelling in R

Nikolaos Kourentzes (nikolaos@kourentes.com)

Contents

1 Load dataset 1

2 Data exploration 2

3 Building a regression model 6

4 Forecast using regression models 10

5 Multiple regression 11

6 Variable selection 12

7 Exercises for basic regression building. 22

8 Advanced regression modelling 22

9 Construct lags 25

10 Seasonality with dummy variables 35

11 Modelling in differences (handling trends) 41

12 Exercises on advanced regression 50

1 Load dataset

First, we will load some data to get familiar with basic regression in R.

```
x <- read.csv("./Lab3data1.csv")
```

The principal command for this workshop is the function `lm()` which fits a regression line. I am not printing the output, but feel free to inspect the variables by simply typing their names or using the function `print()`. **Tip: you can see the help file for any function by typing ?function, for example `?print`**.

Regression models can be built quite easily if we store our data as *data.frame*. This is another way to store data in computer memory. It is very similar to an array that can have multiple columns, however, with a data.frame each column can be named and have different properties. For example, one column could contain numerical values, while another one could contain logical ones (i.e. TRUE or FALSE). We can do this with the function `as.data.frame()`

```
x <- as.data.frame(x)
```

We can check what is the nature of each variable in the memory using the function `class()`. For example:

```
class(x)
```

```
## [1] "data.frame"
```

we have 5 variables: week, price, sales, revenue, advertising.

2 Data exploration

First, we perform some initial data exploration. We start by looking at the correlation between the variables. To do that we use the function `cor()`.

```r
cor(x)
```

```
##                   week      price       sales    revenue advertising
## week        1.00000000  0.2365250  0.05310142  0.1316621   0.3056992
## price       0.23652496  1.0000000 -0.71842121  0.4687595   0.0000000
## sales       0.05310142 -0.7184212  1.00000000 -0.1403195   0.6632365
## revenue     0.13166208  0.4687595 -0.14031950  1.0000000   0.3462019
## advertising 0.30569920  0.0000000  0.66323653  0.3462019   1.0000000
```

We get the correlations between all variables because the data are stored in a data.frame. If we would like to get the correlations between two variables, assuming they are of comparable length, we could have used:

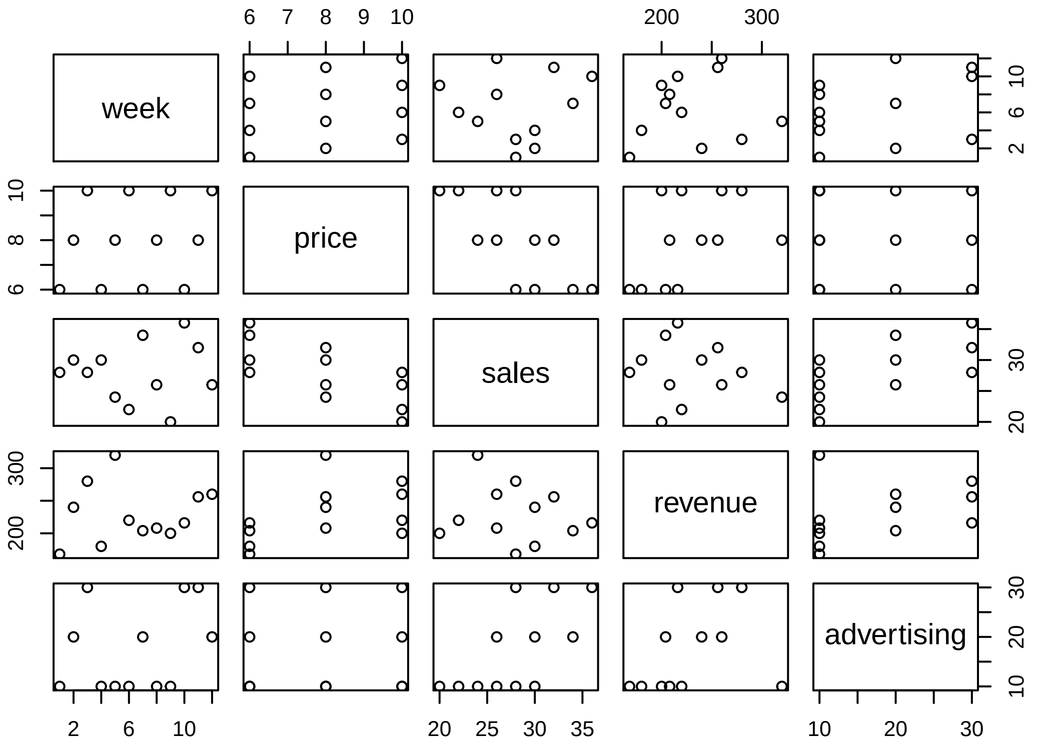
```r
a <- x[,3] # Store third column of x as variable a
b <- x[,4] # Store fourth column of x as variable b
cor(a,b)   # correlation between the 3rd and 4th column
```

```
## [1] -0.1403195
```

By default `cor()` gives us the Pearson correlation coefficient. We can get the non-parametric equivalents with the same function. Inspect the additional inputs by looking at the help documentation: `?cor`. For example, we could use `cor(x,method="spearman")`.

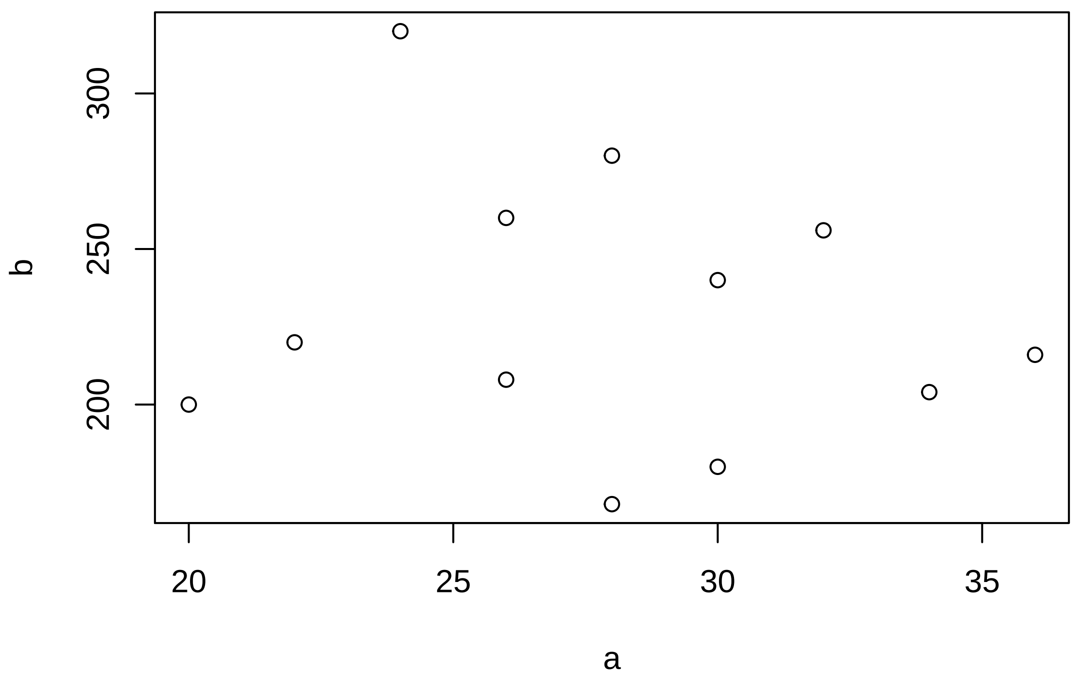
Correlations are not adequate and we need to visualise the data using scatter plots to ensure that we are modelling linear relationships. As long as the data are in a data.frame this is very easy to do:

```r
plot(x)
```



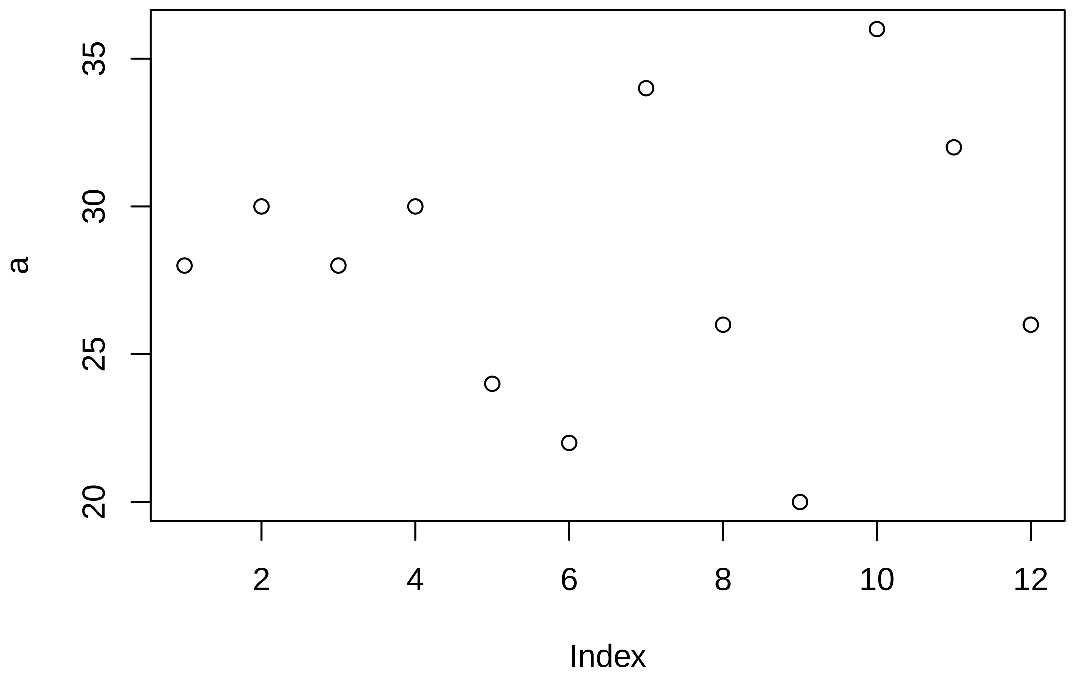
Alternatively, we can "manually" produce a scatter plot between any two variables:

```r
plot(a,b)
```

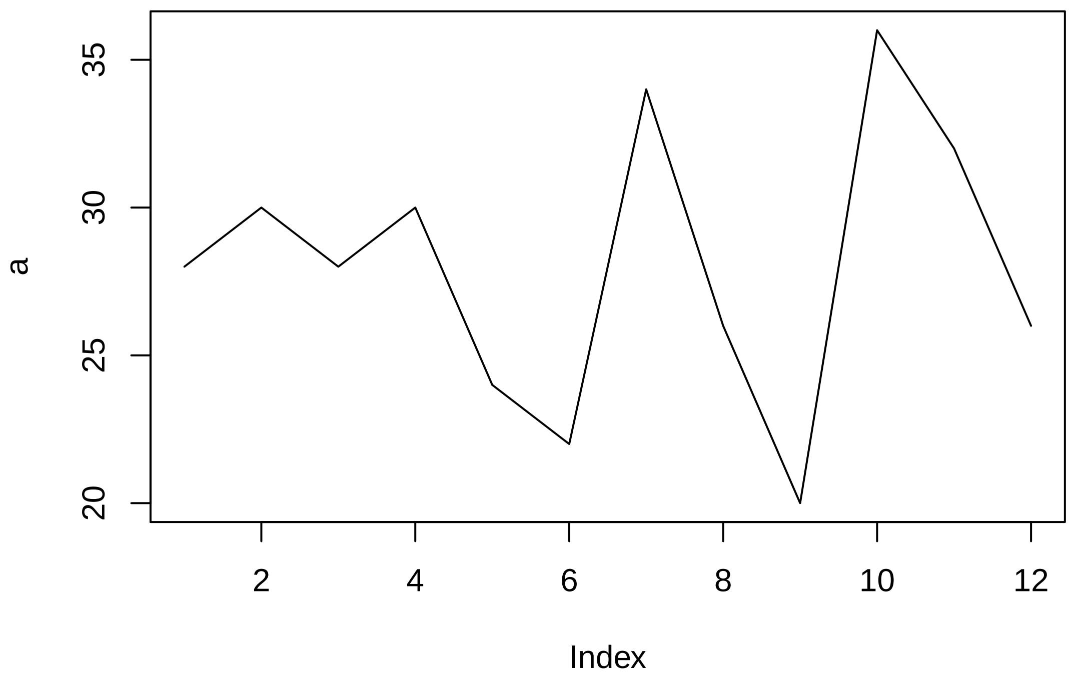


Given that we model time series data, it may be helpful to plot the time series. We still rely on the `plot()` function. In the previous workshop when we used plot, because the data were already in `ts` class, R automatically recognised the variables as time series and plotted them accordingly. Let us get some more understanding of the `plot()` function.

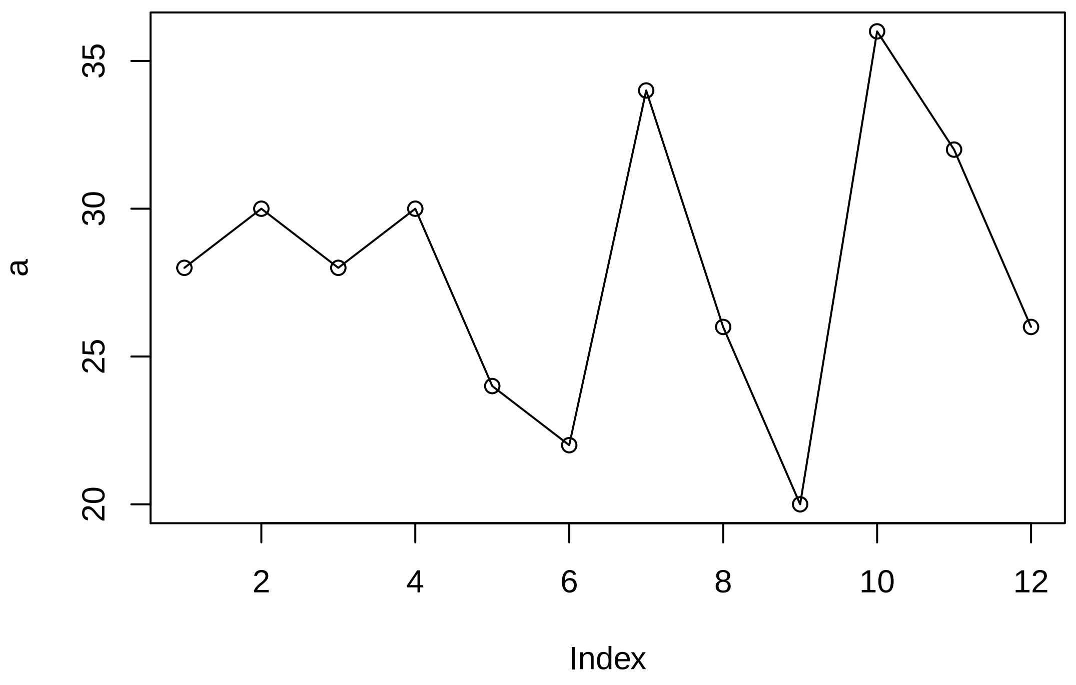
```r
plot(a,type="p") # This plots points
```



```r
plot(a,type="l") # This plots lines
```

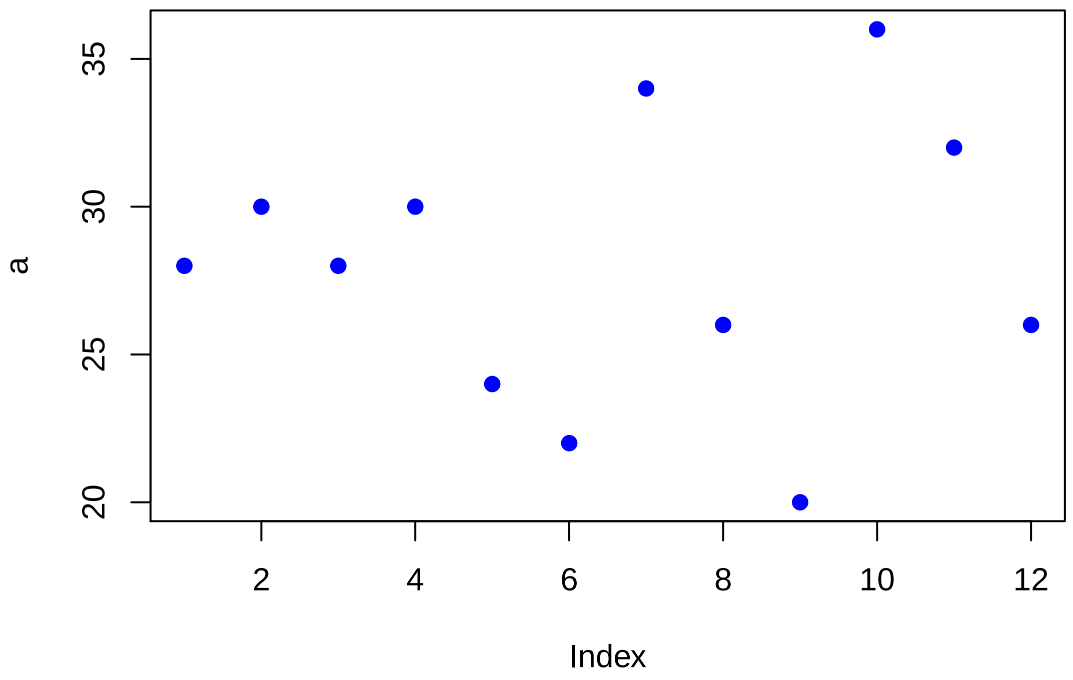


```r
plot(a,type="o") # This plots points, connected with lines
```



There are a lot of arguments we can use to alter the appearance of plots. For example, we can change the markers by using the argument `pch` which takes values from 0 to 25. You have already seen in the previous lab the `col` argument to control the colour used in plotting.

```r
plot(a,pch=19,col="blue")
```



3 Building a regression model

Now we build a regression using the `lm()` function. In this function, there are two crucial arguments: (i) the regression equation, and (ii) the dataset. For this task we want to regress *sales* on *price*, so we will write the equation `sales ~ price`. This is almost identical to how we would write it on a piece of paper. However, note that the intercept and the coefficients are not needed and that "=" is replaced by "~". For more details, you can access the help of `lm()` with: `?lm`.

```
# Regression sales on price
fit1 <- lm(sales ~ price, data=x)
fit1
```

```
##
## Call:
## lm(formula = sales ~ price, data = x)
##
## Coefficients:
## (Intercept)        price
##          44           -2
```

`fit1` is a variable name I chose. We could use any name we want. The basic output provides only the coefficients and the formula. We can get additional output using the function `summary()`.

```
summary(fit1)
```

```
##
## Call:
## lm(formula = sales ~ price, data = x)
##
## Residuals:
```

```
##    Min     1Q Median     3Q    Max
##   -4.0   -2.5    0.0    2.5    4.0
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  44.0000     5.0000   8.800 5.06e-06 ***
## price        -2.0000     0.6124  -3.266  0.00849 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.464 on 10 degrees of freedom
## Multiple R-squared:  0.5161, Adjusted R-squared:  0.4677
## F-statistic: 10.67 on 1 and 10 DF,  p-value: 0.008488
```

Now we get the coefficients, the standard error of the coefficients, the t-statistics, and the corresponding p-values. We also get model statistics, such as the coefficient of determination (R^2) and its adjusted form.

It is often useful to store the result, so we can reuse it.

```
sum1 <- summary(fit1)
```

The output of `summary()` is a list. As in the previous workshop, you can get all the variables contained within using the function `names()`

```
names(sum1)
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

For future reference: within a list you can save any type of variables. These could be vectors, arrays, lists, data.frames, etc. For example, we extract the p-values and test them at 5% significance.

```
# Get p-values and evaluate with a = 0.05
sum1$coefficients
```

```
##             Estimate Std. Error   t value     Pr(>|t|)
## (Intercept)       44  5.0000000  8.800000 5.062315e-06
## price             -2  0.6123724 -3.265986 8.487732e-03
```

```
pval <- sum1$coefficients[,4]
pval <= 0.05
```

```
## (Intercept)       price
##        TRUE        TRUE
```

Observe how I compared the p-values with 0.05. I used the symbol `<` for less than and `=` for equal. The output is a logical variable, i.e. a result that can be either `TRUE` or `FALSE`. If I wanted to test for greater than a value, I would use the symbol `>`. If you want greater (less) or equal, the `=` symbol must follow the `>` (or `<`).

As you can see, in our example, everything is significant.

We also extract the R^2 of model `fit1`.

```
fit1r2 <- sum1$r.squared
fit1r2
```

```
## [1] 0.516129
```

We argued in the lecture that the coefficient of determination is not a great statistic for building forecasting models. Instead, we would like to have some information criterion, such as the Akaike Information Criterion (AIC). Remember that for the AIC, similarly to accuracy metrics, the smaller the better.

To calculate AIC we use the function `AIC()`.

```r
fit1aic <- AIC(fit1)
fit1aic
```

```
## [1] 67.68555
```

Now quickly repeat for the sales on revenue regression. Try to do it on your own, using the example above. Save the model in variable `fit2`, its summary in `sum2`, its R^2 in `fit2r2`, and its AIC in `fit2aic`.

Did you try? :)

The solution is in the next page.

```r
# Regress sales on revenue
fit2 <- lm(sales~revenue,data=x) # Fit the regression model
sum2 <- summary(fit2) # Get summary statistics

# Get R^2 and AIC
fit2r2 <- sum2$r.squared
fit2aic <- AIC(fit2)

# Test coefficients
sum2$coefficients[,4] <= 0.05
```

```
## (Intercept)     revenue
##        TRUE       FALSE
```

Observe that the revenue is insignificant at 5%. Let us see the details of the coefficients:

```r
sum2$coefficients
```

```
##                Estimate Std. Error    t value    Pr(>|t|)
## (Intercept) 31.49949135 7.93718825  3.9685957 0.002649352
## revenue     -0.01525941 0.03404878 -0.4481632 0.663588875
```

Revenue has a p-value of 0.66, so there is some evidence that this variable may need to be removed from the model.

Let us compare the models using R^2 and AIC. I merge both into a single vector, using the function `c()`. The first value will be the R^2 of the first model (`fit1`), and the second of `fit2`.

```r
r2 <- c(fit1r2,fit2r2)
r2
```

```
## [1] 0.51612903 0.01968956
```

The first model explains the data more, with a substantially higher coefficient of determination. (Remember, coefficient of determination is not a great statistic. We are doing this here, so that you know how to get the values, in case you need them for other tasks, rather than rely on them for building forecasting models.)

We often do not need to get so many decimal points. A handy function is `round()`, which rounds output to the desired number of decimals. For example:

```r
round(r2,2)
```

```
## [1] 0.52 0.02
```

The output is still not very nice, as we just get two numbers. We can name these. These are saved in a vector (the output of `c()`), so to name each element we use the function `names()`

```r
names(r2) <- c("Price","Revenue")
round(r2,3)
```

```
##   Price Revenue
##   0.516   0.020
```

We repeat the same for AIC.

```r
aic <- c(fit1aic,fit2aic)
names(aic) <- names(r2)
round(aic,3)
```

```
##   Price Revenue
##  67.686  76.158
```

Observe that instead of retyping the names, I copied them from variable `r2`. The first model has a lower AIC so it is preferable. As it happens, R^2 agreed.

Question: Why the sales~price has more explanatory power to sales~revenue? Remember a model needs to be reasonable!

4 Forecast using regression models

We will use a different dataset

```
x2 <- read.csv("./Lab3data2.csv")
x2 <- as.data.frame(x2)
x2
```

```
##   week spots price sales
## 1    1     8    11    25
## 2    2    12    11    34
## 3    3    16    12    39
## 4    4    10    10    32
## 5    5     8    12    22
## 6    6    12    12    30
## 7    7    16    10    43
## 8    8    10    10    31
```

This contains information for sales for 8 weeks, given some price and number of advertisement spots. We build the regression of sales on spots. Try it on your own. The solution follows.

```
# Construct the regression and get its statistics
fit3 <- lm(sales~spots,data=x2)
sum3 <- summary(fit3)
sum3
```

```
##
## Call:
## lm(formula = sales ~ spots, data = x2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0143 -2.3214  0.5429  1.9143  3.0429
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   8.6714     3.6619   2.368 0.055671 .
## spots         2.0286     0.3084   6.578 0.000592 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.58 on 6 degrees of freedom
## Multiple R-squared:  0.8782, Adjusted R-squared:  0.8579
## F-statistic: 43.27 on 1 and 6 DF,  p-value: 0.0005922
```

Question: What did I do wrong here?

I should have explored the data first, and crucially make sure that anything I am modelling is linear! For now, let us assume that things are fine and progress to build a forecast using this model. Please don't do this assumption if you ever have to do this in practice! :)

To generate forecasts we use the function `predict()`. There are three arguments that we are interested in:

1. The model;
2. The new data to use to construct the forecast, i.e. the explanatory variables;
3. An optional argument that is used to ask for prediction intervals.

Any inputs must be as a data.frame. Let's create one with the new spots value.

```
xnew <- data.frame("spots"=20)
xnew
```

```
##   spots
## 1    20
```

Note that I named the column that includes the new data point as `spots`, which is the same as the variable is named in our model. This is important, to make sure that R can identify what data to use.

We produce the forecast

```
frc <- predict(fit3,xnew,interval="prediction")
frc
```

```
##        fit      lwr      upr
## 1 49.24286 39.97025 58.51546
```

We get three results: `fit` is the forecast, and the other two values are the lower and upper prediction intervals. If I want to access a specific number, then I need to ask for the specific value I am interested in, using:

```
frc[1]
```

```
## [1] 49.24286
```

This will give us the forecast (named fit in the previous output).

5 Multiple regression

To fit a multiple regression we simply expand the formula to include more variables.

```
fit4 <- lm(sales~spots+price,data=x2)
sum4 <- summary(fit4)
sum4
```

```
##
## Call:
## lm(formula = sales ~ spots + price, data = x2)
##
## Residuals:
##       1       2       3       4       5       6       7       8
##  0.1000  0.9857  0.3714  0.5429 -0.4000 -0.5143 -0.6286 -0.4571
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 36.17143    3.29196  10.988 0.000109 ***
## spots        2.02857    0.08354  24.281 2.21e-06 ***
## price       -2.50000    0.28536  -8.761 0.000321 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.699 on 5 degrees of freedom
## Multiple R-squared:  0.9926, Adjusted R-squared:  0.9896
## F-statistic: 333.2 on 2 and 5 DF,  p-value: 4.787e-06
```

We can see from the output that all coefficients are significant. Let us compare the models (`fit3` and `fit4`). R^2 is useless for comparing models with a different number of variables. We could use its adjusted version (we can get that by typing `sum3$adj.r.squared` for `fit3` - we have saved its summary statistics in variable `sum3`), but this has its limitations as well, so we will directly use AIC.

```
c(AIC(fit3),AIC(fit4))
```

```
## [1] 41.56708 21.21283
```

As we can see the AIC of `fit4` (the second number) is better, so `fit4` should be used. To produce a forecast we need the data.frame to contain values for both variables.

```r
xnew <- data.frame("spots"=20, "price"=40)
predict(fit4,xnew) # I do not ask for prediction intervals, so I will get only the forecast.
```

```
##         1
## -23.25714
```

Note that I can use this expanded `xnew` variable with model `fit3` without any problems. R is smart enough to use only the variables that are relevant (i.e., named correctly!).

```r
predict(fit3,xnew)
```

```
##        1
## 49.24286
```

6 Variable selection

Let us revisit the first dataset.

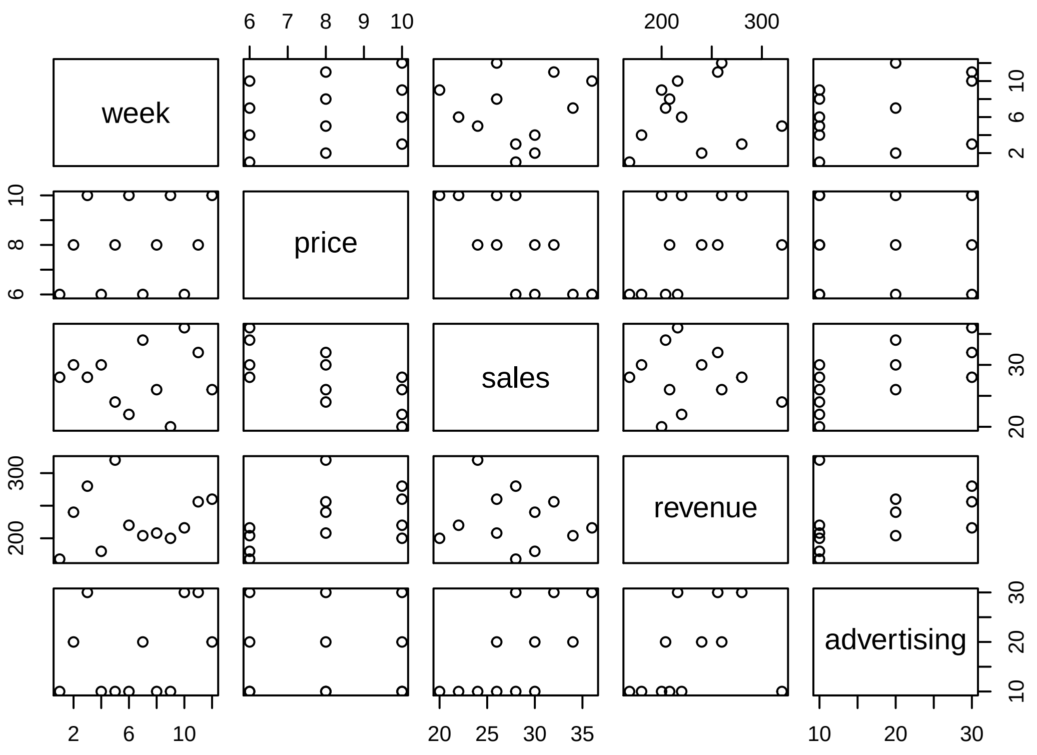
```r
x
```

```
##    week price sales revenue advertising
## 1     1     6    28     168          10
## 2     2     8    30     240          20
## 3     3    10    28     280          30
## 4     4     6    30     180          10
## 5     5     8    24     320          10
## 6     6    10    22     220          10
## 7     7     6    34     204          20
## 8     8     8    26     208          10
## 9     9    10    20     200          10
## 10   10     6    36     216          30
## 11   11     8    32     256          30
## 12   12    10    26     260          20
```

```r
# Correlation and scatterplots
cor(x)
```

```
##                   week      price       sales    revenue advertising
## week        1.00000000  0.2365250  0.05310142  0.1316621   0.3056992
## price       0.23652496  1.0000000 -0.71842121  0.4687595   0.0000000
## sales       0.05310142 -0.7184212  1.00000000 -0.1403195   0.6632365
## revenue     0.13166208  0.4687595 -0.14031950  1.0000000   0.3462019
## advertising 0.30569920  0.0000000  0.66323653  0.3462019   1.0000000
```

```r
plot(x)
```



To build a full model of sales we can write: `sales ~ .` The symbol "." means all remaining variables in the dataset.

```
fit5 <- lm(sales~.,data=x)
summary(fit5)
```

```
##
## Call:
## lm(formula = sales ~ ., data = x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.5434 -0.6873  0.1277  0.7575  1.2446
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 38.163449   2.193779  17.396  5.1e-07 ***
## week         0.025101   0.111346   0.225 0.828084
## price       -1.950216   0.258614  -7.541 0.000133 ***
## revenue     -0.005194   0.010435  -0.498 0.633884
## advertising  0.369508   0.048944   7.550 0.000132 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.22 on 7 degrees of freedom
## Multiple R-squared:  0.958,  Adjusted R-squared:  0.934
## F-statistic:  39.9 on 4 and 7 DF,  p-value: 6.619e-05
```

Observe which variables are significant. Let's see what the stepwise routines will give us. We can build the stepise using the function `step()`. Check its help: `?step`.

To do this we first need to define the smallest model to consider.

```r
fitmin <- lm(sales~1,data=x) # This means use only an intercept.
```

We use the argument `direction` to control how the stepwise evolves and the argument `scope` to define the full model. For (both directions) stepwise we use:

```r
fit6 <- step(fitmin,direction="both",scope=formula(sales~week+price+revenue+advertising))
```

```
## Start:  AIC=38.34
## sales ~ 1
##
##               Df Sum of Sq    RSS    AIC
## + price        1   128.000 120.00 31.631
## + advertising  1   109.091 138.91 33.387
## <none>                     248.00 38.342
## + revenue      1     4.883 243.12 40.104
## + week         1     0.699 247.30 40.308
##
## Step:  AIC=31.63
## sales ~ price
##
##               Df Sum of Sq     RSS    AIC
## + advertising  1   109.091  10.909  4.856
## <none>                     120.000 31.631
## + week         1    13.067 106.933 32.248
## + revenue      1    12.266 107.734 32.337
## - price        1   128.000 248.000 38.342
##
## Step:  AIC=4.86
## sales ~ price + advertising
##
##               Df Sum of Sq     RSS    AIC
## <none>                      10.909  4.856
## + revenue      1     0.413  10.496  6.393
## + week         1     0.120  10.789  6.724
## - advertising  1   109.091 120.000 31.631
## - price        1   128.000 138.909 33.387
```

The output provides 3 different models, as more variables are included in each step. The variable inclusion/exclusion criteria is the AIC. The final model is

```r
summary(fit6)
```

```
##
## Call:
## lm(formula = sales ~ price + advertising, data = x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.27273 -0.72727  0.09091  0.81818  1.09091
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 37.63636    1.72488  21.820 4.21e-09 ***
## price       -2.00000    0.19462 -10.276 2.85e-06 ***
## advertising  0.36364    0.03833   9.487 5.54e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.101 on 9 degrees of freedom
## Multiple R-squared:  0.956,  Adjusted R-squared:  0.9462
## F-statistic:  97.8 on 2 and 9 DF,  p-value: 7.853e-07
```

To produce forward regression we simply change the `direction` argument.

```
fit7 <- step(fitmin,direction="forward",scope=formula(sales~week+price+revenue+advertising))
```

```
## Start:  AIC=38.34
## sales ~ 1
##
##                Df Sum of Sq    RSS    AIC
## + price         1   128.000 120.00 31.631
## + advertising   1   109.091 138.91 33.387
## <none>                      248.00 38.342
## + revenue       1     4.883 243.12 40.104
## + week          1     0.699 247.30 40.308
##
## Step:  AIC=31.63
## sales ~ price
##
##                Df Sum of Sq     RSS    AIC
## + advertising   1   109.091  10.909  4.856
## <none>                      120.000 31.631
## + week          1    13.067 106.933 32.248
## + revenue       1    12.266 107.734 32.337
##
## Step:  AIC=4.86
## sales ~ price + advertising
##
##            Df Sum of Sq    RSS    AIC
## <none>                  10.909 4.8563
## + revenue   1   0.41308 10.496 6.3931
## + week      1   0.11985 10.789 6.7237
```

```
summary(fit7)
```

```
##
## Call:
## lm(formula = sales ~ price + advertising, data = x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.27273 -0.72727  0.09091  0.81818  1.09091
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 37.63636    1.72488  21.820 4.21e-09 ***
## price       -2.00000    0.19462 -10.276 2.85e-06 ***
## advertising  0.36364    0.03833   9.487 5.54e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.101 on 9 degrees of freedom
## Multiple R-squared:  0.956,  Adjusted R-squared:  0.9462
## F-statistic:  97.8 on 2 and 9 DF,  p-value: 7.853e-07
```

But remember, forward regression is not considered a great idea. In brief, the reason is that it does not search between alternative models effectively.

The backward result is somewhat different in the syntax. Now we start from the full model (`fit5`) and ask it to eliminate variables, according to the progression of AIC.

```r
fit8 <- step(fit5,direction="backward",scope=formula(sales~week+price+revenue+advertising))
```

```
## Start:  AIC=8.31
## sales ~ week + price + revenue + advertising
##
##               Df Sum of Sq    RSS    AIC
## - week         1     0.076 10.496  6.393
## - revenue      1     0.369 10.789  6.724
## <none>                     10.420  8.306
## - price        1    84.654 95.074 32.837
## - advertising  1    84.846 95.266 32.861
##
## Step:  AIC=6.39
## sales ~ price + revenue + advertising
##
##               Df Sum of Sq     RSS    AIC
## - revenue      1     0.413  10.909  4.856
## <none>                      10.496  6.393
## - price        1    89.852 100.348 31.485
## - advertising  1    97.238 107.734 32.337
##
## Step:  AIC=4.86
## sales ~ price + advertising
##
##               Df Sum of Sq     RSS    AIC
## <none>                      10.909  4.856
## - advertising  1    109.09 120.000 31.631
## - price        1    128.00 138.909 33.387
```

The final backward model is:

```r
summary(fit8)
```

```
##
## Call:
## lm(formula = sales ~ price + advertising, data = x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.27273 -0.72727  0.09091  0.81818  1.09091
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 37.63636    1.72488  21.820 4.21e-09 ***
## price       -2.00000    0.19462 -10.276 2.85e-06 ***
## advertising  0.36364    0.03833   9.487 5.54e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.101 on 9 degrees of freedom
## Multiple R-squared:  0.956,  Adjusted R-squared:  0.9462
## F-statistic:  97.8 on 2 and 9 DF,  p-value: 7.853e-07
```

Observe that in setting the `scope` argument we used the function `formula()`. This is to tell R that this is not a simple text and that it should look for variables named like that. We could have avoided writing this manually by using:

```r
formula(fit5)
```

```
## sales ~ week + price + revenue + advertising
```

So in a single like we could have (here I encompass several functions in a single line, so we have two `lm()`, the `formula()` and `step()`)

```r
step(lm(sales~1,data=x),direction="both",scope=formula(lm(sales~.,data=x)))
```

```
## Start:  AIC=38.34
## sales ~ 1
##
##                 Df Sum of Sq    RSS    AIC
## + price          1   128.000 120.00 31.631
## + advertising    1   109.091 138.91 33.387
## <none>                       248.00 38.342
## + revenue        1     4.883 243.12 40.104
## + week           1     0.699 247.30 40.308
##
## Step:  AIC=31.63
## sales ~ price
##
##                 Df Sum of Sq     RSS    AIC
## + advertising    1   109.091  10.909  4.856
## <none>                       120.000 31.631
## + week           1    13.067 106.933 32.248
## + revenue        1    12.266 107.734 32.337
## - price          1   128.000 248.000 38.342
##
## Step:  AIC=4.86
## sales ~ price + advertising
##
##                 Df Sum of Sq     RSS    AIC
## <none>                        10.909  4.856
## + revenue        1     0.413  10.496  6.393
## + week           1     0.120  10.789  6.724
## - advertising    1   109.091 120.000 31.631
## - price          1   128.000 138.909 33.387

##
## Call:
## lm(formula = sales ~ price + advertising, data = x)
##
## Coefficients:
## (Intercept)        price  advertising
##     37.6364      -2.0000       0.3636
```

Let us compare the solutions from earlier on (`fit1` and `fit2`) with the full model `fit5` and a stepwise result `fit6`. We use AIC for that purpose.

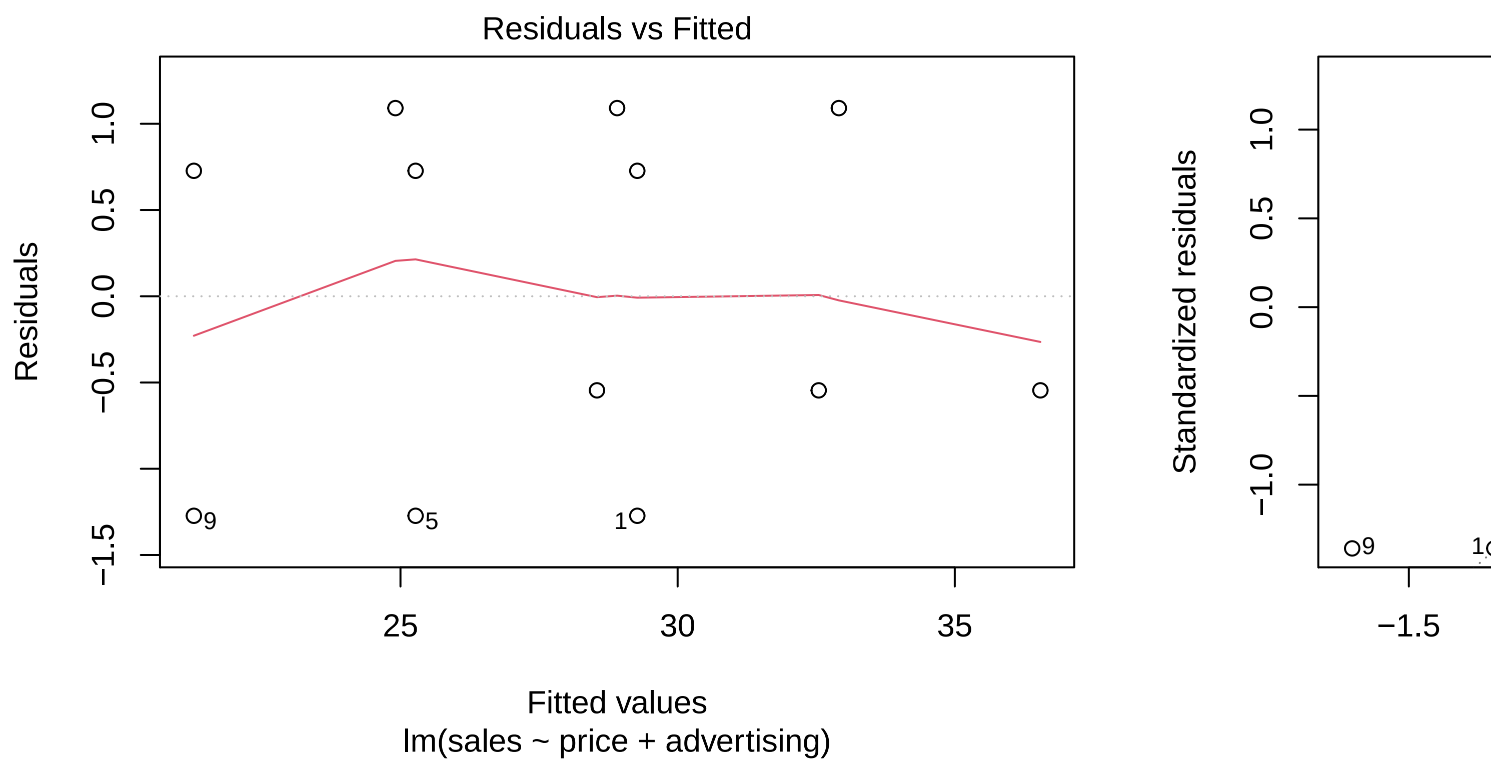
```r
aic <- c(AIC(fit1),AIC(fit2),AIC(fit5),AIC(fit6))
names(aic) <- c(formula(fit1),formula(fit2),"Full model","Stepwise")
round(aic,4)
```

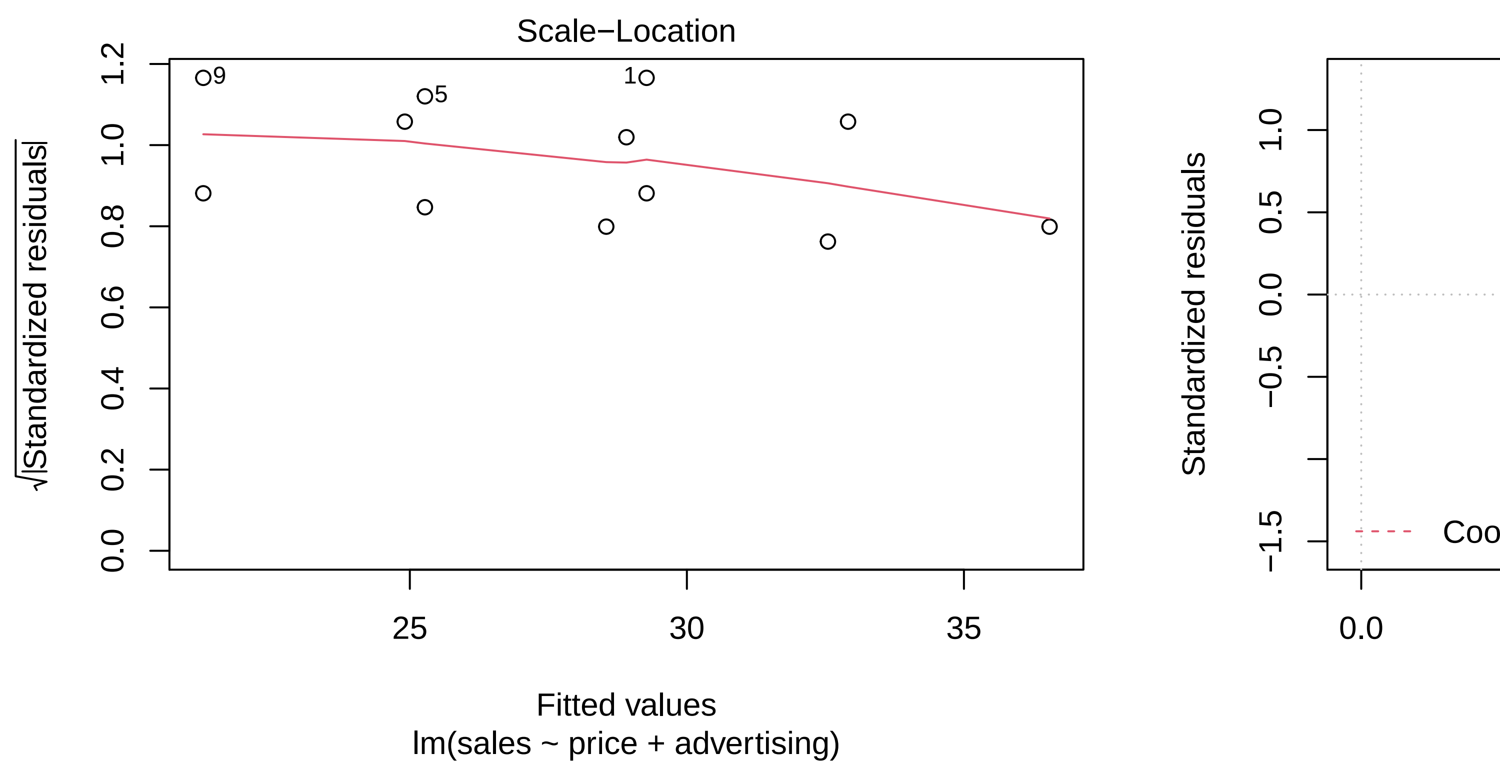
```
##   sales ~ price sales ~ revenue      Full model        Stepwise
##         67.6855         76.1582         44.3608         40.9108
```

As you can see, the stepwise provided the best solution. Note that the stepwise is done on AIC, so we would expect it to perform well on that criterion!

We find `fit6` as being the best model so far. Is the model valid? To diagnose the model we can get useful plots using the `plot()` function on the model fit. R looks at the class of the `fit6` variable and forces the `plot()` function to behave accordingly.

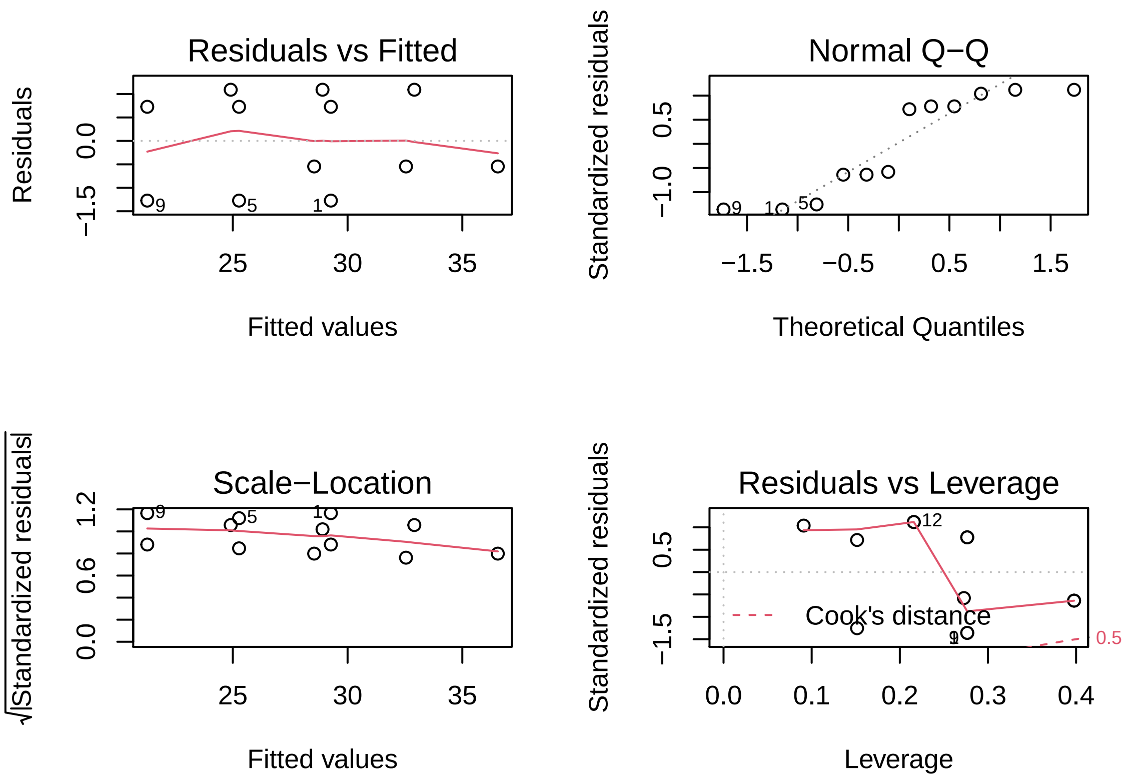
```r
plot(fit6)
```





Some of the statistics you may be familiar with from your prior regression experience. To get all 4 plots in one screen we ask R to split the plot in a 2 by 2 matrix:

```
par(mfrow=c(2,2)) # Split into 2 by 2
plot(fit6) # Plot
```



```r
par(mfrow=c(1,1)) # Revert to a single plot. Otherwise it will keep on plotting on a 2 by 2 matrix.
```

The `par(mfrow=c(*,*))` splits the plotting screen into that many rows x columns.

What are we looking for in these plots? We would like the residuals vs. fitted to look random. We would like the points on the Q-Q plot to look as much on the diagonal line as possible. Values that are too far away hint at the presence of outliers. As for the two plots in the bottom row, they are also diagnostics for outliers. Before we happily explain away the outliers with binary dummy variables, remember that outliers contain information: what about that strange customer (outlier)? Is that an emerging new trend, or just a strange customer? Sometimes in observing how (and why) outliers differ from the rest of the data, we may understand our problem better.

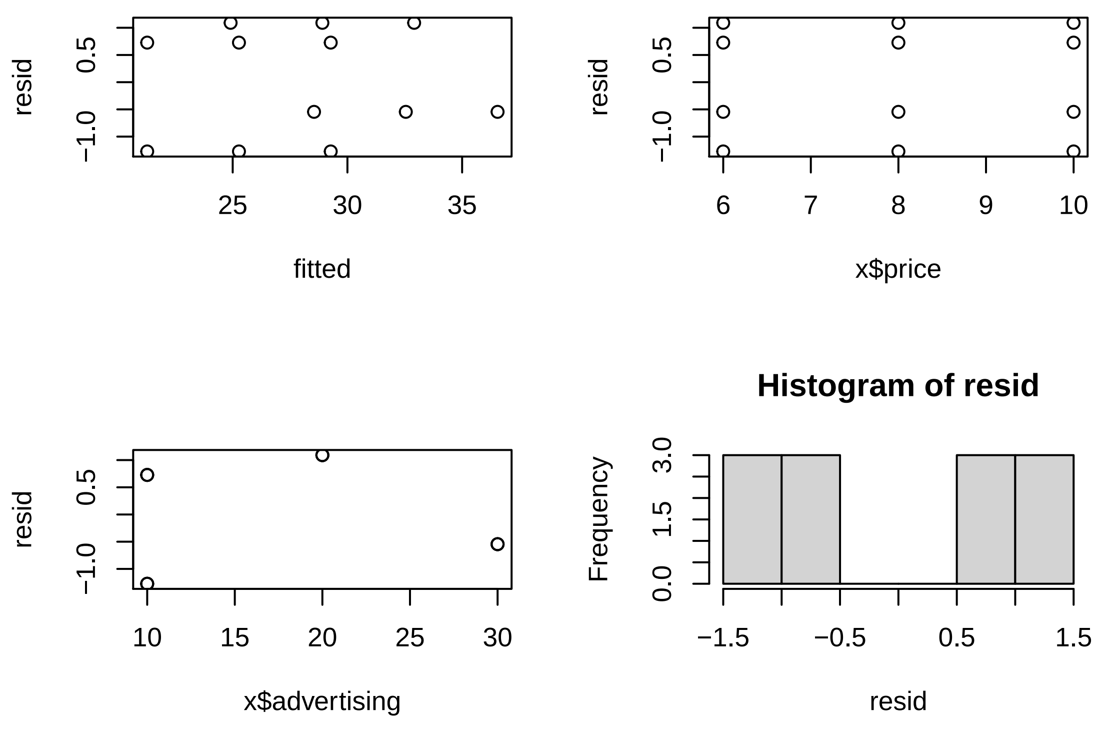
Also, remember that with small samples these diagnostics will never look very clean. The more data we have, the better we will be able to distinguish outliers vs. randomness and potential patterns in the residuals.

Alternatively, we can extract the residuals and produce any plot we want manually.

```r
resid <- fit6$residuals
fitted <- fit6$fitted.values
```

And produce the plots

```r
par(mfrow=c(2,2))
plot(fitted,resid)          # Scatter plot fitted vs. residuals
plot(x$price,resid)         # Scatter plot price vs. residuals
plot(x$advertising,resid) # Scatter advertising vs. residuals
hist(resid)                 # Historgram of residuals
```



```r
par(mfrow=c(1,1))
```

Again, we want all scatter plots to exhibit no patterns. If there are patterns there, they may indicate the presence of unused information (either because our model has omitted terms, or because of nonlinearities). As for the histogram, we would like this to look like a normal distribution. Here it does not, but keep in mind that we have too little data to expect it to look anything like a normal distribution. If anything, it does not have any outliers and looks fairly symmetric, so as good as it would get with only 8 data points.

With the limited sample size we have, there is not too much we can do with these plots. Statistical tests would not be much more helpful, as their results would be very uncertain with so little data.

> **Question**: How can we test against an exponential smoothing forecast? Remember that exponential smoothing needs the input data to be a time series using the function `ts()`.

A final note! If you want to build a model without a constant you would do it as (note the 0):

```r
fit9 <- lm(sales ~ 0 + ., data=x)
summary(fit9)
```

```
##
## Call:
## lm(formula = sales ~ 0 + ., data = x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.0297 -3.9997  0.5795  4.8267 10.6046
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## week         0.15937    0.69104   0.231    0.823
## price       -0.13471    1.47207  -0.092    0.929
## revenue      0.08385    0.05657   1.482    0.177
## advertising  0.44740    0.30322   1.476    0.178
##
## Residual standard error: 7.59 on 8 degrees of freedom
## Multiple R-squared:  0.9523, Adjusted R-squared:  0.9284
## F-statistic:  39.9 on 4 and 8 DF,  p-value: 2.497e-05
```

As you can see this includes all the variables, but no intercept.

7 Exercises for basic regression building.

Using the second dataset `x2`

1. Build stepwise, forward, and backward models;
2. Compare all options, including the manual models we built.

8 Advanced regression modelling

We will learn how to build dynamic regression models (with lags) and special aspects of regression model building. At this point, for convenience, we will erase all variables in memory (in R nomenclature we call this environment), using the function `rm()`.

```
rm(list=ls()) # The argument within rm() lists all names of variables.
```

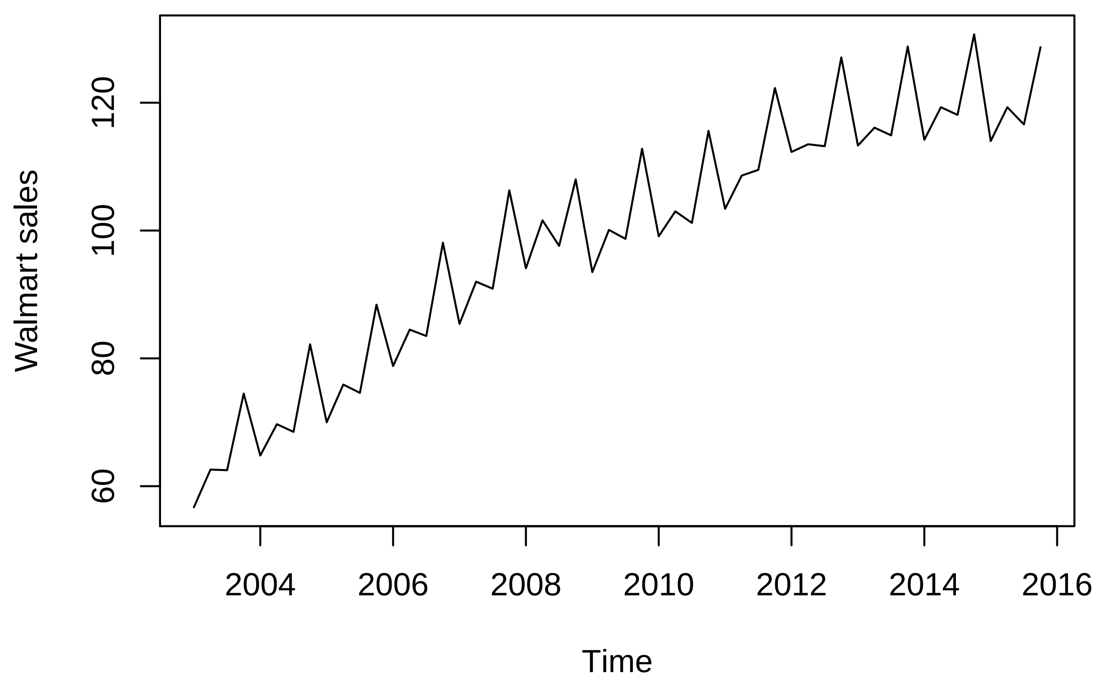
First, we will see how to use lags in regression models and how to build forecasts using these. We load some relevant data and use a conveniently named variable.

```
x <- ts(read.csv("./Lab3data3.csv"),frequency=4,start=c(2003,1))
# Print the first 10 rows
x[1:10,]
```

```
##       sales     gdp
##  [1,]  56.7 11230.1
##  [2,]  62.6 11370.7
##  [3,]  62.5 11625.1
##  [4,]  74.5 11816.8
##  [5,]  64.8 11988.4
##  [6,]  69.7 12181.4
##  [7,]  68.5 12367.7
##  [8,]  82.2 12562.2
##  [9,]  70.0 12813.7
## [10,]  75.9 12974.1
```

The dataset contains the quarterly Walmart sales together with the US GDP data. Let us visually explore the target time series.

```
plot(x[,1],ylab="Walmart sales")
```

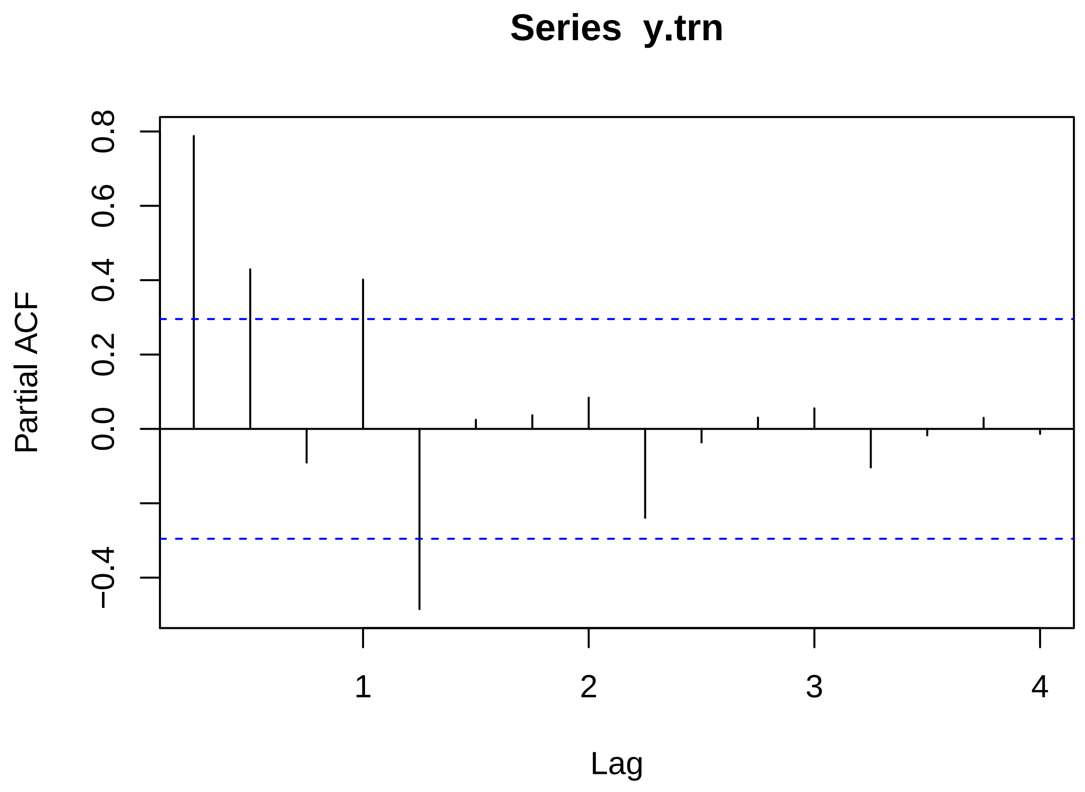


For this task, we will retain the last 2 years as a test set and use the rest to build the models.

```
y.trn <- window(x[,1],end=c(2013,4))
y.tst <- window(x[,1],start=c(2014,1))
```

Let us explore the potential lag-structure of the time series. We will do that by consulting the PACF (Partial AutoCorrelation Function) and to do this we use the function `pacf()`.

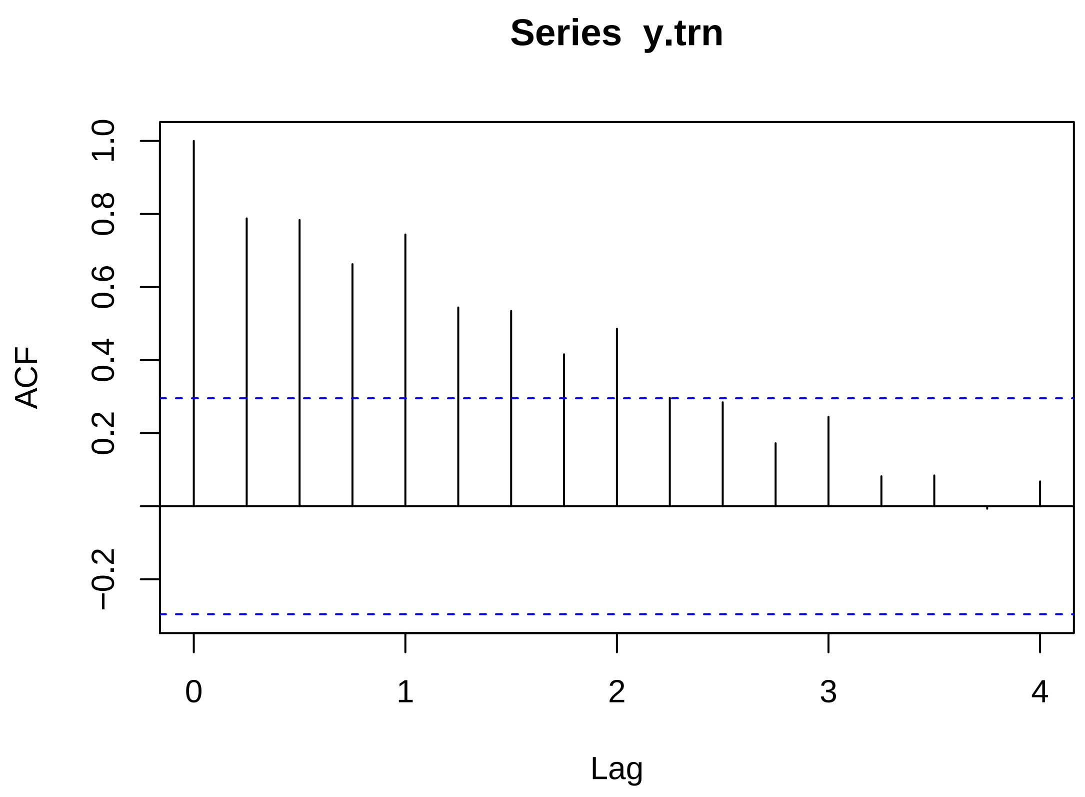
```
pacf(y.trn)
```



The plot tells us that lag 1, lag 2, lag 4 and lag 5 are significant. This may be helpful inputs for a regression model. We will construct all lagged inputs 1-5 and test their usefulness. The significant lag 4 suggests there might be seasonality, which we already saw in the time series plot. A seasonal plot would confirm this further.

If we want to see the ACF (AutoCorrelation Function), we can use the function `acf()`. However, remember that in building standard regression models it is the PACF that carries the useful information, and not the ACF.

```
acf(y.trn)
```



9 Construct lags

First, let us see how many observations we have.

```
n <- length(y.trn)
n
```

```
## [1] 44
```

We will construct an array X to build the lags (and the target variable). We will create an array with n rows and 6 columns, one for the target and five for the lags. By default, the array will be populated with NA (Non-Arithmetic) values.

```
X <- array(NA,c(n,6))
```

We will construct the lags with a *loop*, observe the syntax of the loop, this is quite useful.

```
# We start a loop, which will iterate for all values of i = 1, 2, 3, 4, 5, 6
for (i in 1:6){
  # We tell it to place the data in the i th column, from observation i till the end.
  # We place the data from the beginning towards as much as we can fit to the array (the n-i+1 bit).
  X[i:n,i] <- y.trn[1:(n-i+1)]
}
# Name the columns
# paste0("lag",1:5) creates names lag1, lag2, lag3, lag4, lag5
colnames(X) <- c("y",paste0("lag",1:5))
# Let us see how the resulting array looks like (the first 10 observations)
X[1:10,]
```

```
##          y lag1 lag2 lag3 lag4 lag5
##  [1,] 56.7   NA   NA   NA   NA   NA
##  [2,] 62.6 56.7   NA   NA   NA   NA
##  [3,] 62.5 62.6 56.7   NA   NA   NA
##  [4,] 74.5 62.5 62.6 56.7   NA   NA
##  [5,] 64.8 74.5 62.5 62.6 56.7   NA
##  [6,] 69.7 64.8 74.5 62.5 62.6 56.7
##  [7,] 68.5 69.7 64.8 74.5 62.5 62.6
##  [8,] 82.2 68.5 69.7 64.8 74.5 62.5
##  [9,] 70.0 82.2 68.5 69.7 64.8 74.5
## [10,] 75.9 70.0 82.2 68.5 69.7 64.8
```

Notice that observations are lagged as needed, and therefore at the beginning of the array we miss some data (they remained NA) - these observations will be ignored in a regression. Keep in mind that each observation, in a regression context, must be a complete row of data.

Let us look at the last 10 rows of X.

```
X[(n-9):n,] # Observe the use of parenthesis when I calculate locations in an array
```

```
##           y  lag1  lag2  lag3  lag4  lag5
##  [1,] 109.5 108.6 103.4 115.6 101.2 103.0
##  [2,] 122.3 109.5 108.6 103.4 115.6 101.2
##  [3,] 112.3 122.3 109.5 108.6 103.4 115.6
##  [4,] 113.5 112.3 122.3 109.5 108.6 103.4
##  [5,] 113.2 113.5 112.3 122.3 109.5 108.6
##  [6,] 127.1 113.2 113.5 112.3 122.3 109.5
##  [7,] 113.3 127.1 113.2 113.5 112.3 122.3
##  [8,] 116.1 113.3 127.1 113.2 113.5 112.3
##  [9,] 114.9 116.1 113.3 127.1 113.2 113.5
## [10,] 128.8 114.9 116.1 113.3 127.1 113.2
```

Note that is the end of the series, the lagged variables (as expected) do not contain all values of the target. By calculating lag1 I lose one value (128.8), lag2 two values (114.9 and 128.8), and so on.

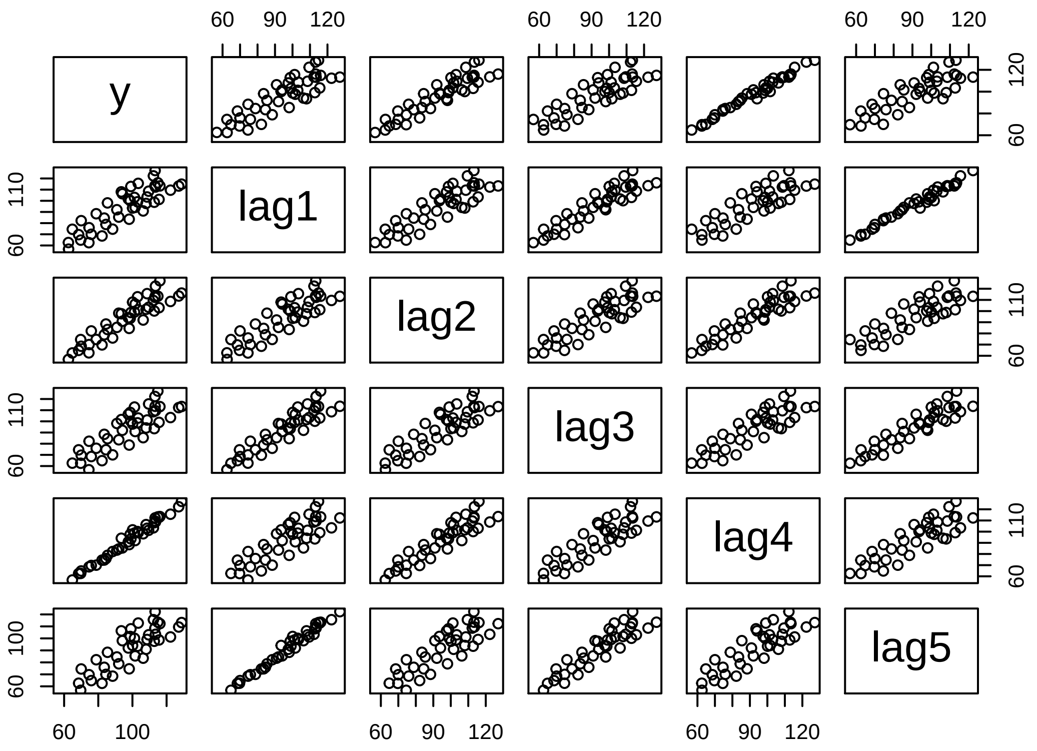
I do not need to check for correlations, as this is guaranteed for lags 1, 2, 4, and 5 from the PACF. However, since I am building a *linear* regression, I should consult scatter plots to make sure that the connection is linear. To make our lives easy, first, we transform `X` into a data.frame. This is done using the function `as.data.frame()`.

```
X <- as.data.frame(X)
```

Now that `X` is a data.frame I can use `plot()` and `lm()` in the same way as before.

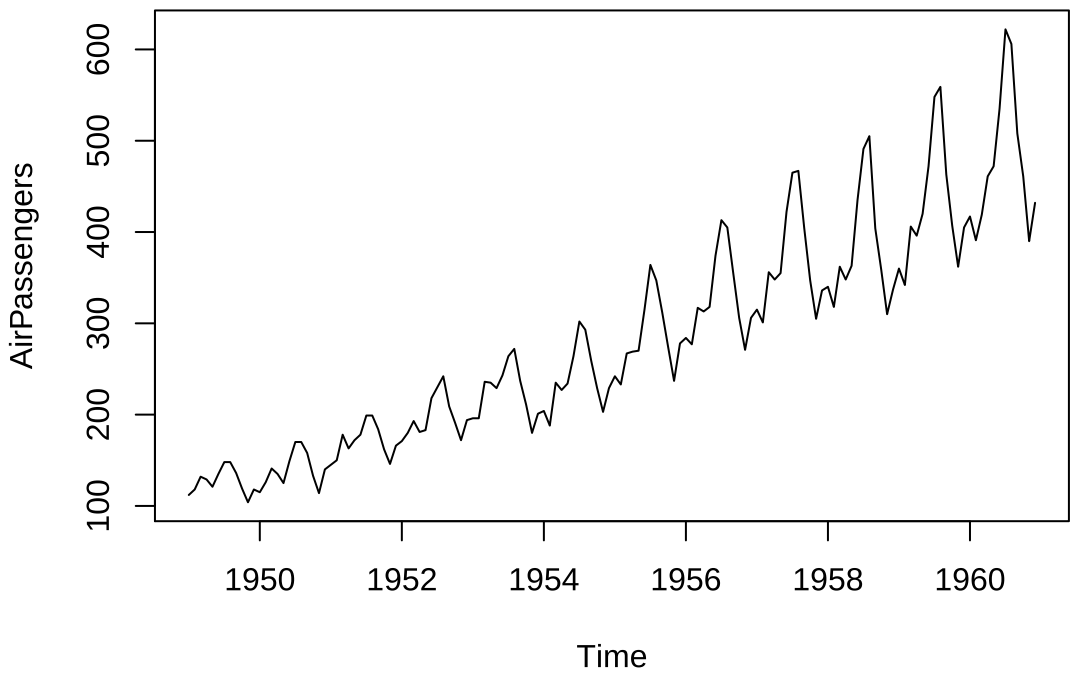
First, look at the scatter plots:

```
plot(X)
```



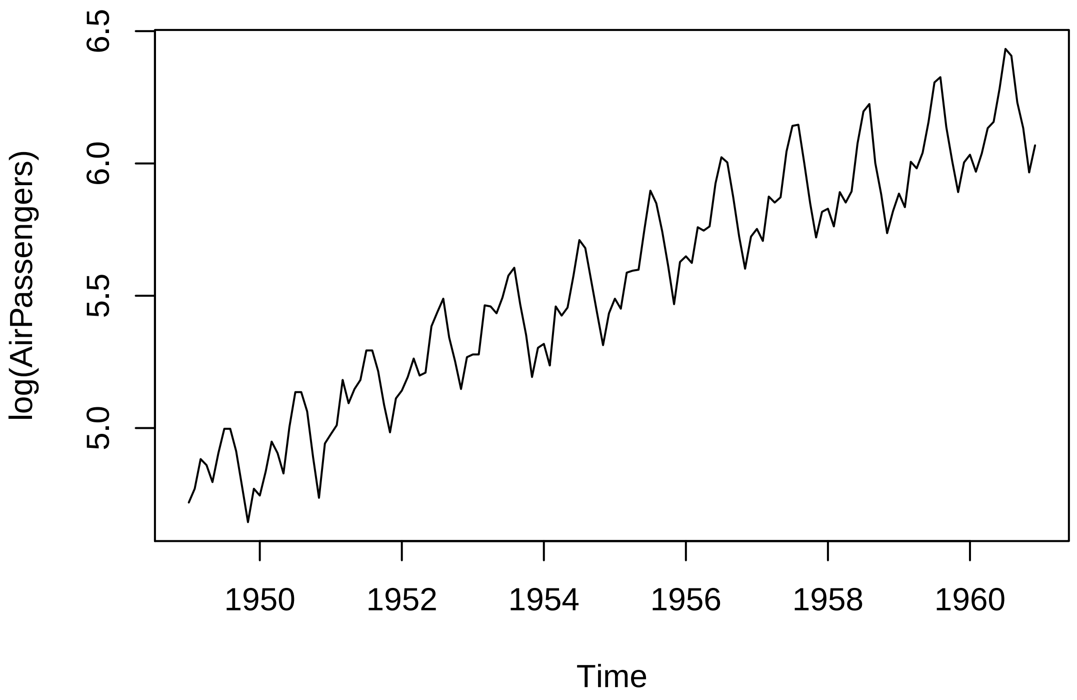
Everything is fairly linear. The seasonality that is present in the time series complicates the scatter plots, as the remaining structure can interact linearly with `y`, but the seasonality non-linearly. To illustrate the point, let us plot the `AirPassengers` time series we used before with exponential smoothing.

```r
plot(AirPassengers)
```



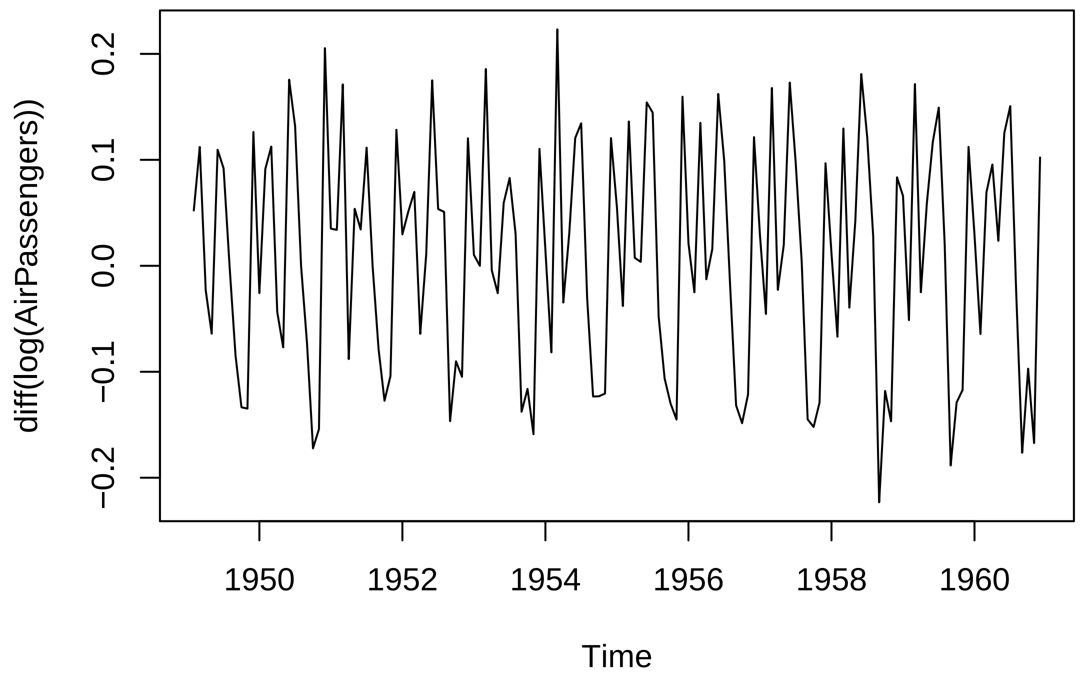
This is a classic case of multiplicative seasonality. Observe that the seasonality becomes wider, as the time series slopes upwards. Therefore, this is a particularly interesting case as it is linear on trend, but non-linear on seasonality (multiplicative). Therefore, exponential smoothing models work on such time series seamlessly, while for regression it is more complicated, as it can only do linear stuff. One way to help regression is to model the variables in logarithms. Logarithms have the property of transforming multiplicative interactions to additive:

```
plot(log(AirPassengers))
```



The seasonality is now additive, and therefore linear. Observe that the width of the seasonality remains fixed, irrespectively of the time series slopping upwards. However, we have distorted the long-term trend. Before it seemed more like a straight line, while now it appears somewhat damped. That complicates modelling in different ways. Remember that trends can be quite nasty on their own, as they can give us the impression of false-correlations. Any two trended time series will exhibit some correlation. In that case, it is good practice to model the differences, which can be done using the function `diff()`.

```
# Logs are calculated first, and then differences
plot(diff(log(AirPassengers)))
```



We will return to the discussion of differences later on. Now let us focus again on modelling the Walmart time series. We use `lm()` to build regression models. We will consider two alternatives:

- The complete model (all lags)
- The stepwise selection

```
# The complete model
fit1 <- lm(y~.,data=X)
summary(fit1)
```

```
##
## Call:
## lm(formula = y ~ ., data = X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.3828 -1.0817  0.3289  1.2419  3.4923
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.94606    2.55986   1.932    0.062 .
## lag1         0.68880    0.12896   5.341 6.74e-06 ***
## lag2        -0.01486    0.04917  -0.302    0.764
## lag3        -0.02849    0.04952  -0.575    0.569
## lag4         0.99860    0.04920  20.297  < 2e-16 ***
## lag5        -0.67931    0.13025  -5.215 9.77e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.965 on 33 degrees of freedom
##   (5 observations deleted due to missingness)
## Multiple R-squared:  0.987,  Adjusted R-squared:  0.985
## F-statistic: 499.8 on 5 and 33 DF,  p-value: < 2.2e-16
```

```
# The stepwise model
fit2 <- step(fit1)
```

```
## Start:  AIC=58.18
## y ~ lag1 + lag2 + lag3 + lag4 + lag5
##
##        Df Sum of Sq     RSS     AIC
## - lag2  1      0.35  127.79  56.286
## - lag3  1      1.28  128.71  56.567
## <none>               127.44  58.178
## - lag5  1    105.04  232.48  79.624
## - lag1  1    110.17  237.61  80.475
## - lag4  1   1590.97 1718.40 157.638
##
## Step:  AIC=56.29
## y ~ lag1 + lag3 + lag4 + lag5
##
##        Df Sum of Sq     RSS     AIC
## - lag3  1      1.51  129.29  54.743
## <none>               127.79  56.286
## - lag5  1    104.99  232.78  77.674
## - lag1  1    111.14  238.92  78.691
## - lag4  1   2717.34 2845.12 175.302
##
## Step:  AIC=54.74
## y ~ lag1 + lag4 + lag5
##
##        Df Sum of Sq     RSS     AIC
## <none>               129.29  54.743
## - lag1  1    110.09  239.39  76.766
## - lag5  1    116.20  245.50  77.749
## - lag4  1   2910.88 3040.17 175.888
```

```
summary(fit2)
```

```
##
## Call:
## lm(formula = y ~ lag1 + lag4 + lag5, data = X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2420 -1.2261  0.2523  1.3036  3.2640
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.5873     2.4497   1.873   0.0695 .
## lag1          0.6783     0.1242   5.459 3.99e-06 ***
## lag4          0.9824     0.0350  28.071  < 2e-16 ***
## lag5         -0.6927     0.1235  -5.609 2.53e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.922 on 35 degrees of freedom
##   (5 observations deleted due to missingness)
## Multiple R-squared:  0.9868, Adjusted R-squared:  0.9856
## F-statistic: 870.6 on 3 and 35 DF,  p-value: < 2.2e-16
```

This suggests removing lags 2 and 3. We already anticipated lag 2 to be removed, as it was insignificant in the PACF before. AIC suggests that lag 3 was redundant as well (note that `step()` is using AIC as the default selection criterion).

Look at the coefficients and keep in mind that the series is trending upwards. The net effect from the autoregressive coefficients is less than 1 (sum of coefficients for lags 1, 4, and 5). This means that you always *copy* a bit less, so the forecast will tend to go back to the intercept, and not to higher numbers. Therefore, this model will not be able to predict a trending time series. For now, we ignore this issue.

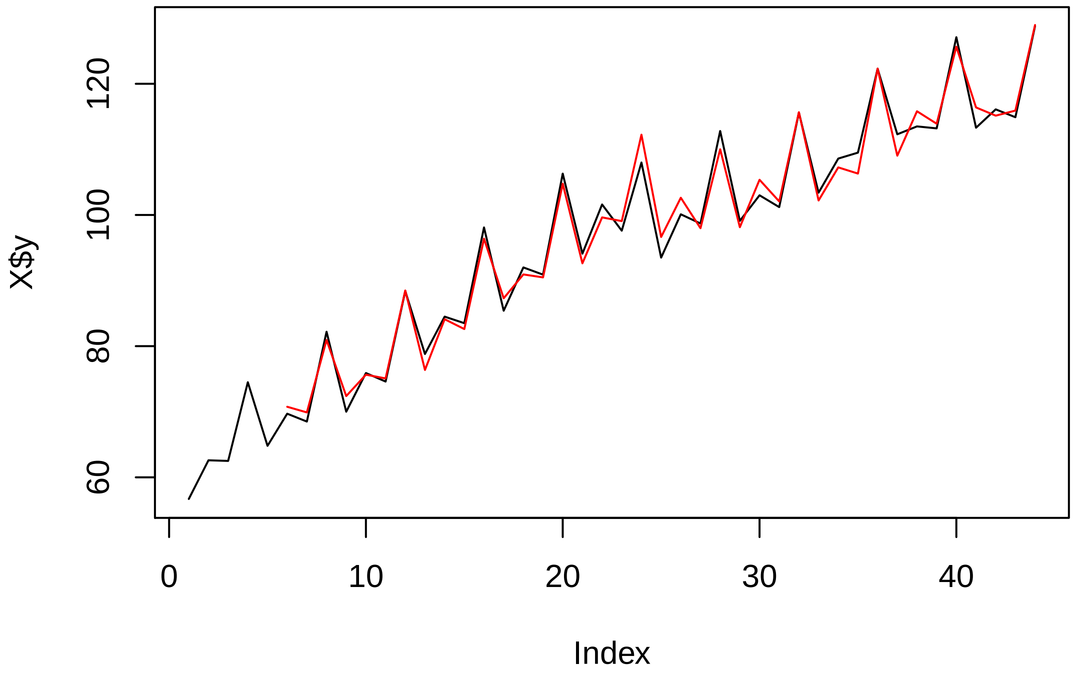
Let us compare the two models and select the most plausible one

```
c(AIC(fit1),AIC(fit2))
```

```
## [1] 170.8552 167.4197
```

Model `fit2` has lower AIC, so it is preferable. Let us explore visually how well the model fits to the data.

```
# In-sample fit:
plot(X$y,type="l")
frc <- predict(fit2,X)
lines(frc,col="red")
```



This looks pretty good. However, we are doing forecasting, so a good in-sample fit is not the complete picture. Let us produce some forecasts.

We will do it iteratively. The inputs are lag1-5. so we will make a vector for the inputs.

```
# I will take the last 5 values (remember: up to lag 5)
Xnew <- array(tail(y.trn,5),c(1,5))
colnames(Xnew) <- paste0("lag",5:1) # Note that I invert the order.
# I do that as the last value is lag1 and 5 values ago is lag 5.
```

```r
# R is smart enough to pick the right element, just by looking at the names.
Xnew <- as.data.frame(Xnew)
Xnew
```

```
##    lag5  lag4  lag3  lag2  lag1
## 1 127.1 113.3 116.1 114.9 128.8
```

And the forecast is:

```r
predict(fit2,Xnew)
```

```
##        1
## 115.2038
```

This is the t+1 forecast. Let us get multiple step-ahead forecasts for the complete test set. I don't want to do that manually, so I will make use of a for-loop. Also, as I will eventually move away from the last observed data point, I will need to switch to using forecasted values as inputs.

Let me first explain the logic of how this will work. First, we create an array to save the forecasts

```r
frc1 <- array(NA,c(8,1)) # 8 because the test set is 8 periods
```

Now we want to create the inputs. Initially, these are the last 5 observations, just like before. Now I will invert the order of the observation directly:

```r
Xnew <- tail(y.trn,5)
Xnew <- Xnew[5:1]
Xnew
```

```
## [1] 128.8 114.9 116.1 113.3 127.1
```

Let us remember what our model form is:

```r
formula(fit2)
```

```
## y ~ lag1 + lag4 + lag5
```

To construct the t+1 forecast I have everything in Xnew. However, to produce the t+2 forecast, I will need lag1, which I will not have. Let me exemplify with dates. Suppose period t is November and that period is the last data point we have observed:

- the t+1 forecast is for December. In December lag1 is the previous month, so that would be November, which I have.
- the t+2 forecast is for January. In January the lag1 refers to December. For December I do not have an actual observation, but I can instead use the t+1 forecast that refers to December. This makes it possible to produce an `iterative` forecast. Note that the errors of t+1 forecast will contaminate the t+2 forecast and so on. Eventually, for long-term predictions these errors accumulate, making long-term forecasts fairly inaccurate.

To do this logic in code, I will do the following trick. I will append to Xnew the forecasted values:

```r
Xnew <- c(Xnew, frc1)
Xnew
```

```
##  [1] 128.8 114.9 116.1 113.3 127.1    NA    NA    NA    NA    NA    NA    NA
## [13]    NA
```

These forecasts are for now NA (Non-Arithmetic), but as I generate them these will be replaced by the appropriate values. All I need then is to slide across that vector, to always take the relevant last 5 observations. For example, to produce:

- the t+1 I need the first 5 values, `Xnew[1:5]`
- the t+2 I need values 2-6, dropping the oldest first value, `Xnew[2:6]`, where `Xnew[6]` will be the first forecast, and so on.

I do this below with a for-loop.

```r
frc1 <- array(NA,c(8,1))

for (i in 1:8){
  # For the Xnew we use the last five observations as before
  Xnew <- tail(y.trn,5)
  # Add to that the forecasted values
  Xnew <- c(Xnew,frc1)
  # Take the relevant 5 values. The index i helps us to get the right ones
  Xnew <- Xnew[i:(4+i)]
  # If i = 1 then this becomes Xnew[1:5].
  # If i = 2 then this becomes Xnew[2:6] - just as the example above.
  # Reverse the order
  Xnew <- Xnew[5:1]
  # Make Xnew an array and name the inputs
  Xnew <- array(Xnew, c(1,5)) # c(1,5) are the dimensions of the array
  colnames(Xnew) <- paste0("lag",1:5) # I have already reversed the order
  # Convert to data.frame
  Xnew <- as.data.frame(Xnew)
  # Forecast
  frc1[i] <- predict(fit2,Xnew)
}
frc1
```

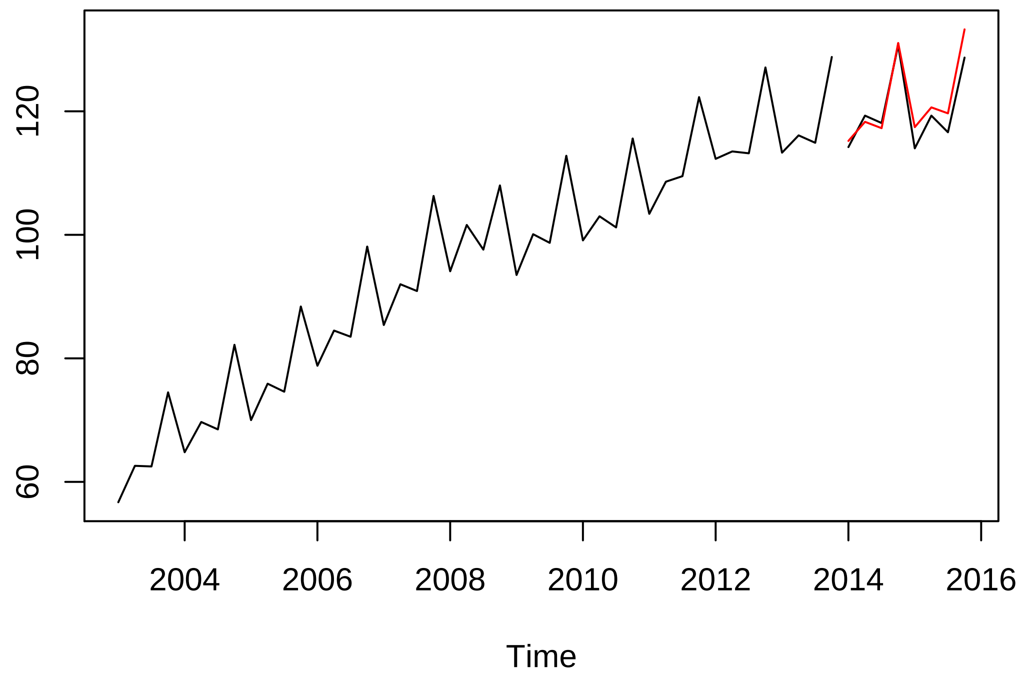
```
##          [,1]
## [1,] 115.2038
## [2,] 118.2922
## [3,] 117.2685
## [4,] 131.0602
## [5,] 117.4294
## [6,] 120.6364
## [7,] 119.6665
## [8,] 133.2663
```

Let us plot the result. To do this conveniently I need to tell `frc1` to copy the time series properties from my data. As it refers to the same periods as the test set `y.tst`, I will copy from that one.

```r
# Transform to time series, by copying the information from y.tst
frc1 <- ts(frc1,frequency=frequency(y.tst),start=start(y.tst))
```

And produce a plot with the actual data and the forecast

```r
ts.plot(y.trn,y.tst,frc1,col=c("black","black","red"))
```



That looks reasonable. The seasonality is modelled using autoregressive lags. Below we consider the case of dummy variables.

10 Seasonality with dummy variables

We will use dummies to encode seasonality. To do this we will create a *factor* variable. This will code seasonality from 1 to 4, and R internally will translate it into dummies. We tell R that something is a factor using the function `factor()`

```
D <- rep(1:4,11) # Replicate 1:4 11 times
D <- factor(D)
D
```

```
##  [1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2
## [39] 3 4 1 2 3 4
## Levels: 1 2 3 4
```

Observe that when I print the result, now it adds another line “Levels: 1 2 3 4” that shows what are the categories. These could be numbers or text. So instead, I could have used:

```
factor(rep(c("Q1","Q2","Q3","Q4"),11))
```

```
##  [1] Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1
## [26] Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4
## Levels: Q1 Q2 Q3 Q4
```

We bind the dummies with the lags in `X` we constructed before.

```
X2 <- cbind(X,D)
colnames(X2) <- c(colnames(X2)[1:6],"D")
X2
```

```
##        y  lag1  lag2  lag3  lag4  lag5 D
## 1   56.7    NA    NA    NA    NA    NA 1
## 2   62.6  56.7    NA    NA    NA    NA 2
## 3   62.5  62.6  56.7    NA    NA    NA 3
## 4   74.5  62.5  62.6  56.7    NA    NA 4
## 5   64.8  74.5  62.5  62.6  56.7    NA 1
## 6   69.7  64.8  74.5  62.5  62.6  56.7 2
## 7   68.5  69.7  64.8  74.5  62.5  62.6 3
## 8   82.2  68.5  69.7  64.8  74.5  62.5 4
## 9   70.0  82.2  68.5  69.7  64.8  74.5 1
## 10  75.9  70.0  82.2  68.5  69.7  64.8 2
## 11  74.6  75.9  70.0  82.2  68.5  69.7 3
## 12  88.4  74.6  75.9  70.0  82.2  68.5 4
## 13  78.8  88.4  74.6  75.9  70.0  82.2 1
## 14  84.5  78.8  88.4  74.6  75.9  70.0 2
## 15  83.5  84.5  78.8  88.4  74.6  75.9 3
## 16  98.1  83.5  84.5  78.8  88.4  74.6 4
## 17  85.4  98.1  83.5  84.5  78.8  88.4 1
## 18  92.0  85.4  98.1  83.5  84.5  78.8 2
## 19  90.9  92.0  85.4  98.1  83.5  84.5 3
## 20 106.3  90.9  92.0  85.4  98.1  83.5 4
## 21  94.1 106.3  90.9  92.0  85.4  98.1 1
## 22 101.6  94.1 106.3  90.9  92.0  85.4 2
## 23  97.6 101.6  94.1 106.3  90.9  92.0 3
## 24 108.0  97.6 101.6  94.1 106.3  90.9 4
## 25  93.5 108.0  97.6 101.6  94.1 106.3 1
## 26 100.1  93.5 108.0  97.6 101.6  94.1 2
## 27  98.7 100.1  93.5 108.0  97.6 101.6 3
## 28 112.8  98.7 100.1  93.5 108.0  97.6 4
## 29  99.1 112.8  98.7 100.1  93.5 108.0 1
## 30 103.0  99.1 112.8  98.7 100.1  93.5 2
## 31 101.2 103.0  99.1 112.8  98.7 100.1 3
## 32 115.6 101.2 103.0  99.1 112.8  98.7 4
## 33 103.4 115.6 101.2 103.0  99.1 112.8 1
## 34 108.6 103.4 115.6 101.2 103.0  99.1 2
## 35 109.5 108.6 103.4 115.6 101.2 103.0 3
## 36 122.3 109.5 108.6 103.4 115.6 101.2 4
## 37 112.3 122.3 109.5 108.6 103.4 115.6 1
## 38 113.5 112.3 122.3 109.5 108.6 103.4 2
## 39 113.2 113.5 112.3 122.3 109.5 108.6 3
## 40 127.1 113.2 113.5 112.3 122.3 109.5 4
## 41 113.3 127.1 113.2 113.5 112.3 122.3 1
## 42 116.1 113.3 127.1 113.2 113.5 112.3 2
## 43 114.9 116.1 113.3 127.1 113.2 113.5 3
## 44 128.8 114.9 116.1 113.3 127.1 113.2 4
```

Build the regression model as usual:

```
fit3 <- lm(y~.,data=X2)
summary(fit3)
```

```
##
## Call:
## lm(formula = y ~ ., data = X2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.5499 -0.6431 -0.0694  0.7327  2.7217
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6.84018    3.64761  -1.875 0.070522 .
## lag1         0.89964    0.18055   4.983 2.45e-05 ***
## lag2         0.09947    0.22994   0.433 0.668390
## lag3        -0.25396    0.22740  -1.117 0.272946
## lag4         0.23654    0.22898   1.033 0.309838
## lag5        -0.01125    0.17798  -0.063 0.950009
## D2          13.01788    5.16637   2.520 0.017302 *
## D3          12.21078    3.51534   3.474 0.001584 **
## D4          20.25475    5.12283   3.954 0.000433 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.567 on 30 degrees of freedom
##   (5 observations deleted due to missingness)
## Multiple R-squared:  0.9925, Adjusted R-squared:  0.9905
## F-statistic: 494.3 on 8 and 30 DF,  p-value: < 2.2e-16
```

Observe that D was broken into 3 dummy variables automatically. Note that lag4 is no longer significant, as seasonality is captured by the dummies. We use stepwise to refine the model. Remember: stepwise is not a perfect answer, but it will give us a good starting point to manually enhance the model further.

Normally we would write `step(fit3)`. This will not work here, due to the NA values in X2. R does not like this when factors are involved. So we need to manually take care of that.

```r
# Find NA in X2
idx <- is.na(X2)
# The result is logical TRUE/FALSE values
idx[1:10,]
```

```
##           y  lag1  lag2  lag3  lag4  lag5     D
##  [1,] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
##  [2,] FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
##  [3,] FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE
##  [4,] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE
##  [5,] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
##  [6,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [7,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [8,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [9,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [10,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

When you make calculations with logicals then TRUE = 1 and FALSE = 0. If we sum across columns, any value greater than zero will have contained at least one TRUE, i.e. at least one NA.

```r
idx <- rowSums(idx)
idx
```

```
##  [1] 5 4 3 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [39] 0 0 0 0 0 0
```

```r
idx <- idx == 0
idx
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [37]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Now `idx` will be TRUE if there are no NA in that row. Now we refit the regression, manually removing the redundant data points. These were excluded in `fit3` automatically.

```
fit_temp <- lm(y~.,data=X2[idx,])
# fit_temp is the same as fit3, without the first NA part
fit4 <- step(fit_temp)
```

```
## Start:  AIC=42.78
## y ~ lag1 + lag2 + lag3 + lag4 + lag5 + D
##
##         Df Sum of Sq     RSS    AIC
## - lag5  1     0.010  73.634 40.786
## - lag2  1     0.459  74.083 41.024
## - lag4  1     2.619  76.243 42.144
## - lag3  1     3.061  76.685 42.370
## <none>               73.624 42.781
## - D     3    53.812 127.436 58.178
## - lag1  1    60.931 134.555 64.298
##
## Step:  AIC=40.79
## y ~ lag1 + lag2 + lag3 + lag4 + D
##
##         Df Sum of Sq     RSS    AIC
## - lag2  1     0.507  74.141 39.054
## - lag3  1     3.206  76.840 40.449
## <none>               73.634 40.786
## - lag4  1     4.338  77.972 41.019
## - lag1  1    63.371 137.005 63.002
## - D     3   158.844 232.478 79.624
##
## Step:  AIC=39.05
## y ~ lag1 + lag3 + lag4 + D
##
##         Df Sum of Sq     RSS    AIC
## - lag3  1     2.704  76.845 38.451
## <none>               74.141 39.054
## - lag4  1     4.999  79.140 39.599
## - lag1  1   124.312 198.453 75.453
## - D     3   158.634 232.776 77.674
##
## Step:  AIC=38.45
## y ~ lag1 + lag4 + D
##
##         Df Sum of Sq     RSS    AIC
## - lag4  1     2.343  79.188 37.622
## <none>               76.845 38.451
## - D     3   168.652 245.498 77.749
## - lag1  1   155.276 232.121 79.564
##
## Step:  AIC=37.62
## y ~ lag1 + D
##
##         Df Sum of Sq    RSS     AIC
## <none>                 79.2  37.622
## - D     3    3076.3 3155.5 175.340
## - lag1  1    8317.2 8396.4 217.508
```

```r
summary(fit4)
```

```
##
## Call:
## lm(formula = y ~ lag1 + D, data = X2[idx, ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.3091 -0.6497  0.0275  0.6699  2.7110
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -9.65240    1.81455  -5.319 6.61e-06 ***
## lag1         0.97499    0.01632  59.758  < 2e-16 ***
## D2          16.96995    0.74424  22.802  < 2e-16 ***
## D3          10.82574    0.72090  15.017  < 2e-16 ***
## D4          25.73473    0.72586  35.454  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.526 on 34 degrees of freedom
## Multiple R-squared:  0.9919, Adjusted R-squared:  0.9909
## F-statistic:  1041 on 4 and 34 DF,  p-value: < 2.2e-16
```

The resulting model has kept only lag1 and the dummies. Note that stepwise tested the dummies together as a group. This is why we used a single factor, instead of adding manually 3 binary dummy variables.

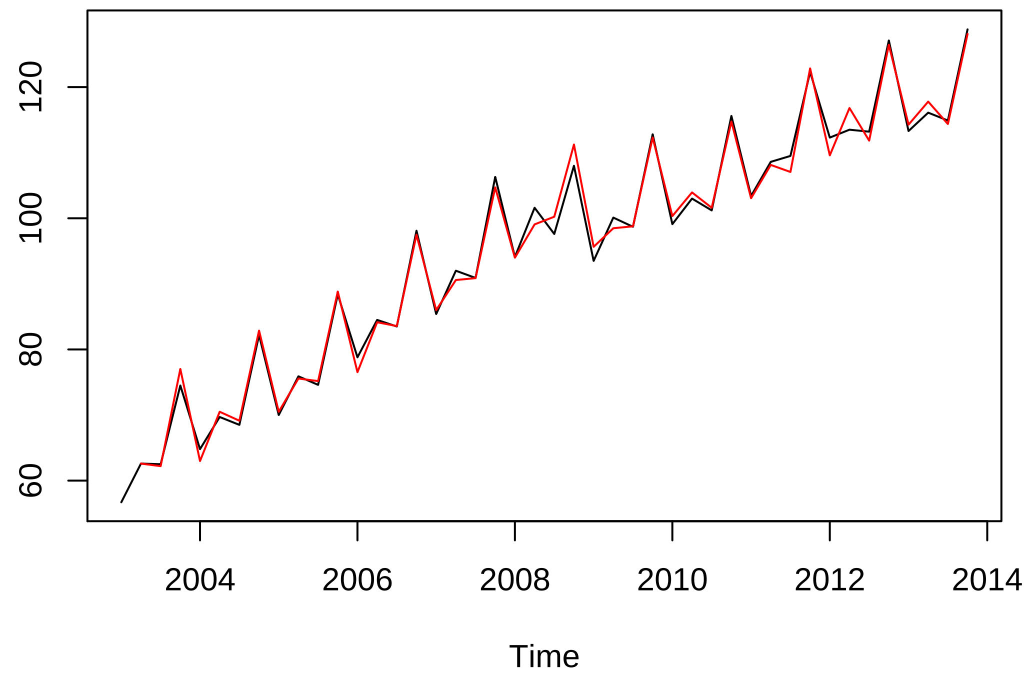
Let us compare with the stochastic seasonality model, from before:

```r
c(AIC(fit2),AIC(fit4))
```

```
## [1] 167.4197 150.2997
```

The new model has lower (better) AIC, let us produce its in-sample fit

```r
frc <- predict(fit4,X2)
ts.plot(y.trn,frc,col=c("black","red"))
```



Looks fine, but we already know that in-sample fit can be deceiving. Let us produce its forecast. We will adjust the code slightly so that we get the dummies right.

```
# Initialise frc2 to store the forecasts
frc2 <- array(NA,c(8,1))
for (i in 1:8){
  # Create lags - same as before
  Xnew <- tail(y.trn,5)
  Xnew <- c(Xnew,frc2)
  Xnew <- Xnew[i:(4+i)]
  Xnew <- Xnew[5:1]
  Xnew <- array(Xnew, c(1,5))
  colnames(Xnew) <- paste0("lag",1:5)
  Xnew <- as.data.frame(Xnew)
  # Xnew contains all the lags
  # Create the value of the dummy
  D <- as.factor(rep(1:4,2)[i])
  # The logic is that I create the dummy for all 8
  # periods and I pick the i th value. I start the
  # dummy from 1 because I know that the first period
  # is quarter 1. I should ammend this otherwise.
  Xnew <- cbind(Xnew,D)
  # Forecast
  frc2[i] <- predict(fit4,Xnew)
}
```

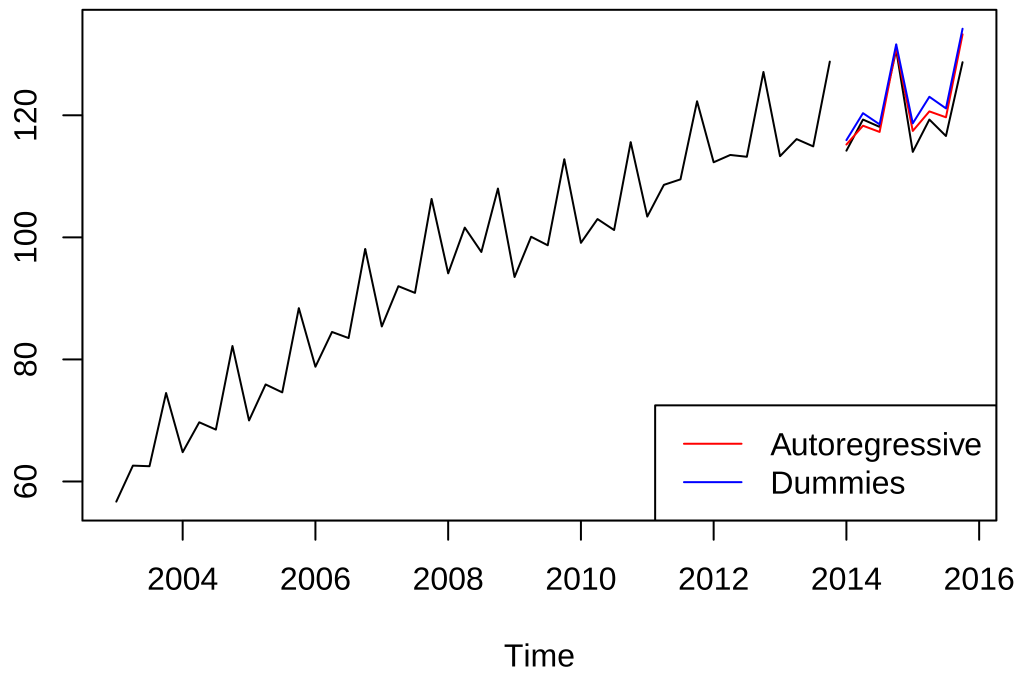
Let us compare the values of the previous forecast and the new one

```
cbind(frc1, frc2)
```

```
##              frc1      frc2
## 2014 Q1 115.2038 115.9265
## 2014 Q2 118.2922 120.3448
## 2014 Q3 117.2685 118.5085
## 2014 Q4 131.0602 131.6271
## 2015 Q1 117.4294 118.6829
## 2015 Q2 120.6364 123.0323
## 2015 Q3 119.6665 121.1288
## 2015 Q4 133.2663 134.1818
```

Numerically they look somewhat different. We plot the new forecast

```r
# Transform to time series
frc2 <- ts(frc2,frequency=frequency(y.tst),start=start(y.tst))
# Plot
ts.plot(y.trn,y.tst,frc1,frc2,col=c("black","black","red","blue"))
legend("bottomright",c("Autoregressive","Dummies"),col=c("red","blue"),lty=1)
```



The new forecast seems to be somewhat biased compared to the old one. Trend could also be an issue, so we deal with that one next.

11 Modelling in differences (handling trends)

We first store the inputs (without the dummy) in a new variable

```r
X3 <- X
```

To calculate differences we can use the function `diff()` for each column.

```r
# The function ncol() counts how many columns
for (i in 1:ncol(X3)){
  X3[,i] <- c(NA,diff(X3[,i]))
}
print(X3)
```

```
##        y  lag1  lag2  lag3  lag4  lag5
## 1     NA    NA    NA    NA    NA    NA
## 2    5.9    NA    NA    NA    NA    NA
## 3   -0.1   5.9    NA    NA    NA    NA
## 4   12.0  -0.1   5.9    NA    NA    NA
## 5   -9.7  12.0  -0.1   5.9    NA    NA
## 6    4.9  -9.7  12.0  -0.1   5.9    NA
## 7   -1.2   4.9  -9.7  12.0  -0.1   5.9
## 8   13.7  -1.2   4.9  -9.7  12.0  -0.1
## 9  -12.2  13.7  -1.2   4.9  -9.7  12.0
## 10   5.9 -12.2  13.7  -1.2   4.9  -9.7
## 11  -1.3   5.9 -12.2  13.7  -1.2   4.9
## 12  13.8  -1.3   5.9 -12.2  13.7  -1.2
## 13  -9.6  13.8  -1.3   5.9 -12.2  13.7
## 14   5.7  -9.6  13.8  -1.3   5.9 -12.2
## 15  -1.0   5.7  -9.6  13.8  -1.3   5.9
## 16  14.6  -1.0   5.7  -9.6  13.8  -1.3
## 17 -12.7  14.6  -1.0   5.7  -9.6  13.8
## 18   6.6 -12.7  14.6  -1.0   5.7  -9.6
## 19  -1.1   6.6 -12.7  14.6  -1.0   5.7
## 20  15.4  -1.1   6.6 -12.7  14.6  -1.0
## 21 -12.2  15.4  -1.1   6.6 -12.7  14.6
## 22   7.5 -12.2  15.4  -1.1   6.6 -12.7
## 23  -4.0   7.5 -12.2  15.4  -1.1   6.6
## 24  10.4  -4.0   7.5 -12.2  15.4  -1.1
## 25 -14.5  10.4  -4.0   7.5 -12.2  15.4
## 26   6.6 -14.5  10.4  -4.0   7.5 -12.2
## 27  -1.4   6.6 -14.5  10.4  -4.0   7.5
## 28  14.1  -1.4   6.6 -14.5  10.4  -4.0
## 29 -13.7  14.1  -1.4   6.6 -14.5  10.4
## 30   3.9 -13.7  14.1  -1.4   6.6 -14.5
## 31  -1.8   3.9 -13.7  14.1  -1.4   6.6
## 32  14.4  -1.8   3.9 -13.7  14.1  -1.4
## 33 -12.2  14.4  -1.8   3.9 -13.7  14.1
## 34   5.2 -12.2  14.4  -1.8   3.9 -13.7
## 35   0.9   5.2 -12.2  14.4  -1.8   3.9
## 36  12.8   0.9   5.2 -12.2  14.4  -1.8
## 37 -10.0  12.8   0.9   5.2 -12.2  14.4
## 38   1.2 -10.0  12.8   0.9   5.2 -12.2
## 39  -0.3   1.2 -10.0  12.8   0.9   5.2
## 40  13.9  -0.3   1.2 -10.0  12.8   0.9
## 41 -13.8  13.9  -0.3   1.2 -10.0  12.8
## 42   2.8 -13.8  13.9  -0.3   1.2 -10.0
## 43  -1.2   2.8 -13.8  13.9  -0.3   1.2
## 44  13.9  -1.2   2.8 -13.8  13.9  -0.3
```

We build the full regression

```r
summary(lm(y~.,X3))
```

```
##
## Call:
## lm(formula = y ~ ., data = X3)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -4.1629 -1.5089  0.3572  1.3891  2.8476
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.3341     0.7653   1.743   0.0909 .
## lag1         -0.1118     0.1754  -0.638   0.5282
## lag2         -0.2588     0.1271  -2.036   0.0501 .
## lag3         -0.2716     0.1269  -2.141   0.0400 *
## lag4          0.7300     0.1313   5.560 3.89e-06 ***
## lag5         -0.1508     0.1818  -0.829   0.4130
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.045 on 32 degrees of freedom
##   (6 observations deleted due to missingness)
## Multiple R-squared:  0.9615, Adjusted R-squared:  0.9555
## F-statistic:   160 on 5 and 32 DF,  p-value: < 2.2e-16
```

We can see that there is little evidence for autoregressive lags to help now, apart from the seasonal one. We let stepwise deal with the selection of variables.

```
fit5 <- step(lm(y~.,X3))
```

```
## Start:  AIC=59.85
## y ~ lag1 + lag2 + lag3 + lag4 + lag5
##
##        Df Sum of Sq    RSS    AIC
## - lag1  1     1.702 135.57 58.332
## - lag5  1     2.878 136.75 58.660
## <none>              133.87 59.852
## - lag2  1    17.344 151.21 62.481
## - lag3  1    19.175 153.04 62.939
## - lag4  1   129.322 263.19 83.541
##
## Step:  AIC=58.33
## y ~ lag2 + lag3 + lag4 + lag5
##
##        Df Sum of Sq    RSS    AIC
## <none>              135.57 58.332
## - lag5  1    15.080 150.65 60.340
## - lag2  1    15.642 151.21 60.481
## - lag3  1    17.488 153.06 60.942
## - lag4  1   158.014 293.58 85.694
```

```
summary(fit5)
```

```
##
## Call:
## lm(formula = y ~ lag2 + lag3 + lag4 + lag5, data = X3)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.1763 -1.6582  0.1921  1.4694  2.9309
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.2013     0.7297   1.646   0.1092
## lag2         -0.2334     0.1196  -1.951   0.0596 .
## lag3         -0.2453     0.1189  -2.063   0.0470 *
## lag4          0.7586     0.1223   6.202 5.33e-07 ***
## lag5         -0.2355     0.1229  -1.916   0.0641 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.027 on 33 degrees of freedom
##   (6 observations deleted due to missingness)
## Multiple R-squared:  0.961,  Adjusted R-squared:  0.9563
## F-statistic: 203.6 on 4 and 33 DF,  p-value: < 2.2e-16
```

Stepwise keeps quite a few lags in the model.

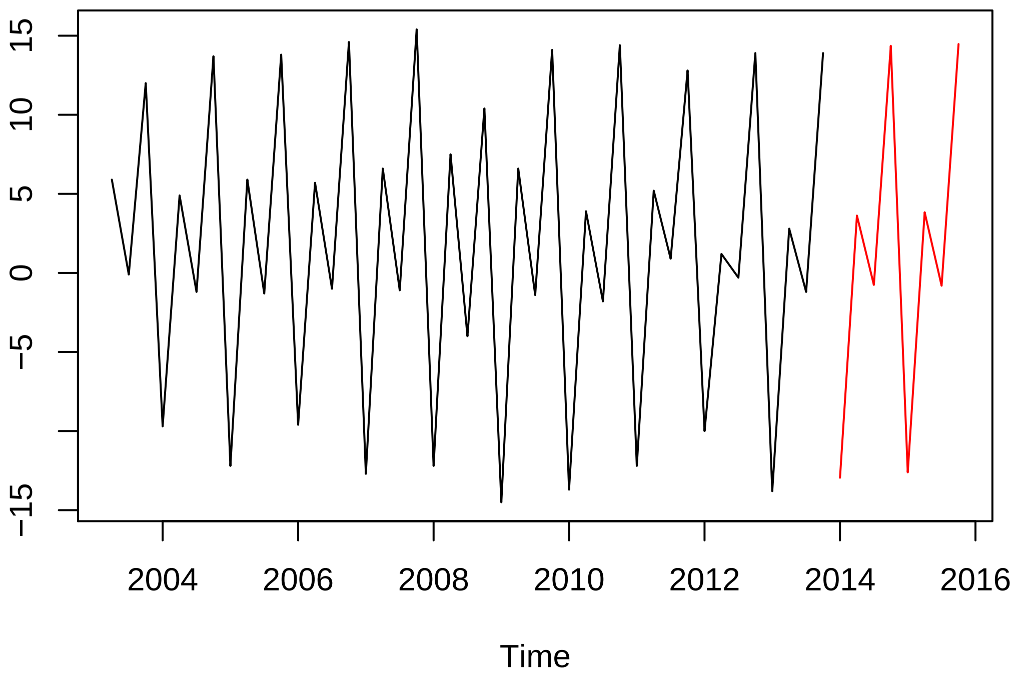
Important: I cannot use AIC to compare `fit5` with the previous models. The reason is that this model is calculated on the differenced data. AIC is only applicable when the data stay the same. In fact, we also need the data to have the same observations, so if we were to use more lags, and therefore trim some more observations from our data, we could no longer use AIC.

We produce the forecasts

```r
frc3 <- array(NA,c(8,1))
for (i in 1:8){
  # Calculate the differences of the in-sample data
  y.diff <- diff(y.trn)
  # Create lags - same as before
  Xnew <- tail(y.diff,5)
  Xnew <- c(Xnew,frc3)
  Xnew <- Xnew[i:(4+i)]
  Xnew <- Xnew[5:1]
  Xnew <- array(Xnew, c(1,5))
  colnames(Xnew) <- paste0("lag",1:5)
  Xnew <- as.data.frame(Xnew)
  # Forecast
  frc3[i] <- predict(fit5,Xnew)
}
```

Plot the result

```r
# Transform to time series
frc3 <- ts(frc3,frequency=frequency(y.tst),start=start(y.tst))
# Plot
ts.plot(diff(y.trn),frc3,col=c("black","red"))
```



The data look very different, as they are in differences. The forecast looks reasonable, but to compare it with the others we need to reverse the differences. To do that we need to add them up from the last undifferenced point (that is the forecast origin).

```
frc3ud <- cumsum(c(tail(y.trn,1),frc3))
# The function cumsum() is the cumulative sum.
```

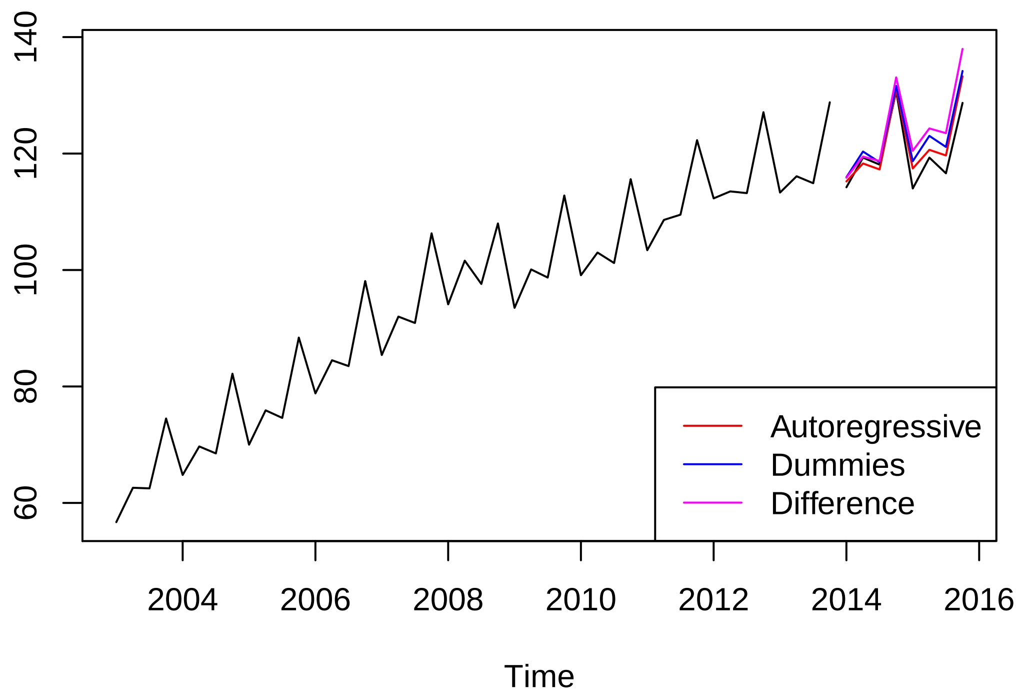
We take the forecast origin and we add the first forecast in differences (that is how much it should change by). On the resulting value (forecast t+1), we add the next forecasted change to get the t+2 forecast. That is a cumulative sum. We do that to get all t+1 to t+8 forecasts.

We do not need the first value (the forecast origin) that was added to help us with the undifferencing:

```
frc3ud <- frc3ud[-1]
```

Plot against the previous forecasts:

```
frc3ud <- ts(frc3ud,frequency=frequency(y.tst),start=start(y.tst))
ts.plot(y.trn,y.tst,frc1,frc2,frc3ud,col=c("black","black","red","blue","magenta"))
legend("bottomright",c("Autoregressive","Dummies","Difference"),col=c("red","blue","magenta"),lty=1)
```



That did not seem to improve things! We compare the three forecasts using out-of-sample metrics:

```
# Create an array with the actuals replicated three times
# to compare with the three forecasts in one go
actual <- matrix(rep(y.tst,3),ncol=3)
actual
```

```
##       [,1]  [,2]  [,3]
## [1,] 114.2 114.2 114.2
## [2,] 119.3 119.3 119.3
## [3,] 118.1 118.1 118.1
## [4,] 130.7 130.7 130.7
## [5,] 114.0 114.0 114.0
## [6,] 119.3 119.3 119.3
## [7,] 116.6 116.6 116.6
## [8,] 128.7 128.7 128.7
```

and calculate the error

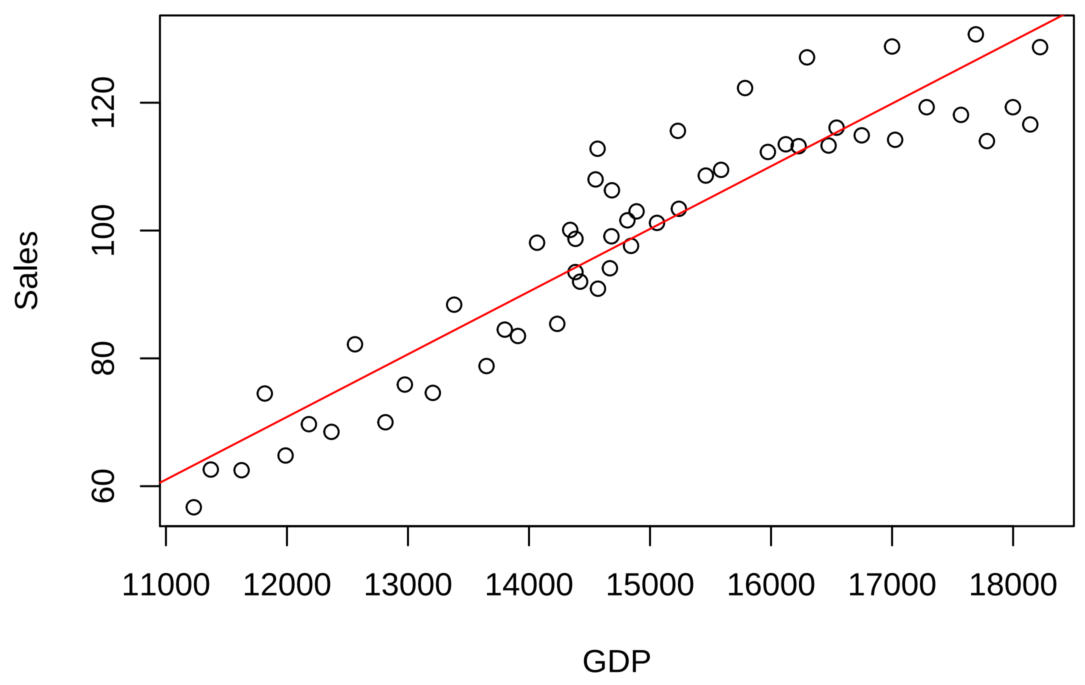
```
error <- abs(actual - cbind(frc1,frc2,frc3ud))
MAE <- colMeans(error)
MAE
```

```
##     frc1     frc2   frc3ud
## 1.950239 2.816589 4.060461
```

The first forecast (with the autoregressions) is the most accurate, as we have already seen visually.

However, maybe the change in trend is better captured by the changes in GDP. Let us explore.

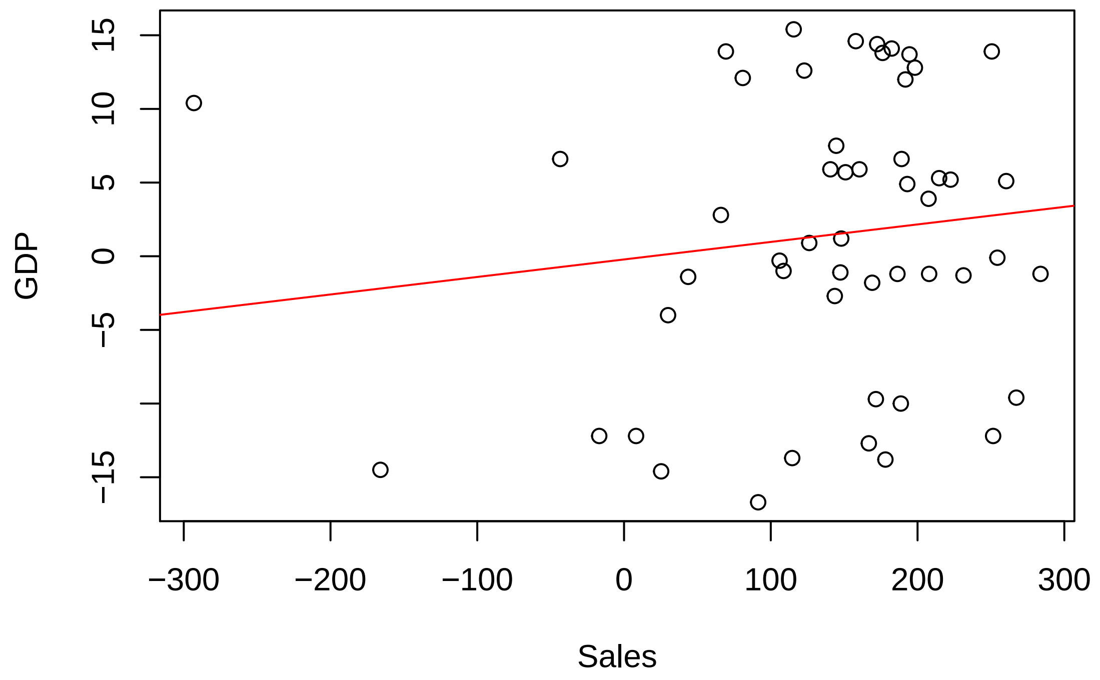
```
plot(as.vector(x[,2]),as.vector(x[,1]),ylab="Sales",xlab="GDP")
abline(lm(x[,1]~x[,2]),col="red")
```



```
# The function abline plots a function, here the regression fit
```

However, when time series are trended, the connection can be spurious. Let us do that in differences

```
plot(as.vector(diff(x[,2])),as.vector(diff(x[,1])),xlab="Sales",ylab="GDP")
abline(lm(diff(x[,1])~diff(x[,2])),col="red")
```



There may be some relevant information there

```
# Get gdp in differences after the test set is removed
gdp <- c(NA,diff(x[1:(length(x[,2])-8),2]))
# Construct inputs for regression
X4 <- cbind(X3,gdp)
fit6 <- step(lm(y~.,X4[-(1:6),])) # Remove NA
```

```
## Start:  AIC=56.83
## y ~ lag1 + lag2 + lag3 + lag4 + lag5 + gdp
##
##          Df Sum of Sq    RSS    AIC
## - lag5   1     0.042 117.35 54.848
## <none>               117.31 56.835
## - lag1   1     8.527 125.84 57.501
## - gdp    1    16.558 133.87 59.852
## - lag2   1    17.762 135.07 60.192
## - lag3   1    20.926 138.24 61.072
## - lag4   1   125.653 242.96 82.502
##
## Step:  AIC=54.85
## y ~ lag1 + lag2 + lag3 + lag4 + gdp
##
##          Df Sum of Sq    RSS    AIC
## <none>               117.35 54.848
## - gdp    1    19.393 136.75 58.660
## - lag1   1    19.458 136.81 58.678
## - lag2   1    20.413 137.76 58.942
## - lag3   1    22.923 140.27 59.629
## - lag4   1   135.254 252.60 81.981
```

```r
summary(fit6)
```

```
##
## Call:
## lm(formula = y ~ lag1 + lag2 + lag3 + lag4 + gdp, data = X4[-(1:6),
##     ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.4271 -1.2216  0.5818  1.4958  3.0880
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.532350   0.712472   0.747   0.4604
## lag1        -0.261779   0.113647  -2.303   0.0279 *
## lag2        -0.265991   0.112742  -2.359   0.0246 *
## lag3        -0.287376   0.114944  -2.500   0.0177 *
## lag4         0.716369   0.117959   6.073 8.79e-07 ***
## gdp          0.006526   0.002838   2.300   0.0281 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.915 on 32 degrees of freedom
## Multiple R-squared:  0.9663, Adjusted R-squared:  0.961
## F-statistic: 183.4 on 5 and 32 DF,  p-value: < 2.2e-16
```

Create forecasts with this model

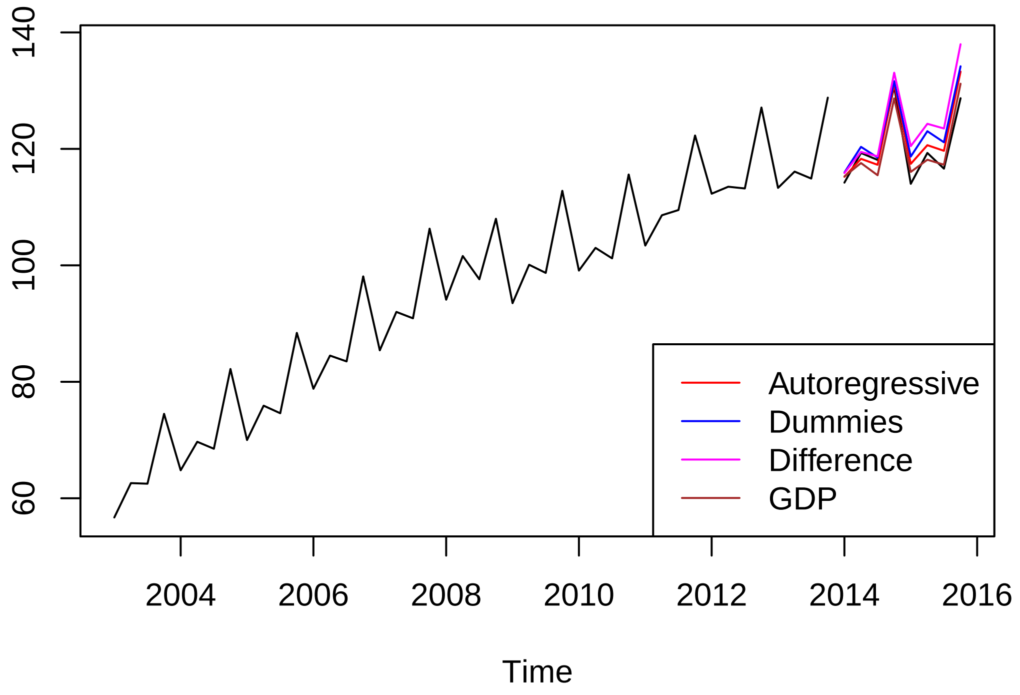
```r
frc4 <- array(NA,c(8,1))
for (i in 1:8){
  # -- Autoregressions are same as before --
  # Calculate the differences of the in-sample data
  y.diff <- diff(y.trn)
  # Create lags - same as before
  Xnew <- tail(y.diff,5)
  Xnew <- c(Xnew,frc3)
  Xnew <- Xnew[i:(4+i)]
  Xnew <- Xnew[5:1]
  # Add differenced gdp information
  # We take the last 9 values, that is test set + 1
  Xgdp <- tail(gdp,9)
  # and calculate differences - this is why we needed the
  # one extra value, which is now removed from the differencing
  Xgdp <- diff(Xgdp)
  # Use only the i th value
  Xgdp <- Xgdp[i]
  # Bind to Xnew
  Xnew <- c(Xnew,Xgdp)
  # Name things
  Xnew <- array(Xnew, c(1,6))
  colnames(Xnew) <- c(paste0("lag",1:5),"gdp")
  Xnew <- as.data.frame(Xnew)
  # Forecast
  frc4[i] <- predict(fit6,Xnew)
}
```

Reverse differencing

```
frc4ud <- cumsum(frc4) + as.vector(tail(y.trn,1))
```

Plot everything together

```
frc4ud <- ts(frc4ud,frequency=frequency(y.tst),start=start(y.tst))
ts.plot(y.trn,y.tst,frc1,frc2,frc3ud,frc4ud,col=c("black","black","red","blue","magenta","brown"))
legend("bottomright",c("Autoregressive","Dummies","Difference","GDP"),col=c("red","blue","magenta","brown")
```



GDP seems to help in tracking the trend better. Let us check the MAE

```
c(MAE, mean(abs(y.tst-frc4ud)))
```

```
##     frc1     frc2   frc3ud
## 1.950239 2.816589 4.060461 1.726872
```

Indeed, the new forecast has the lowest MAE.

12 Exercises on advanced regression

1. Develop a regression using lagged only values of GDP and forecast the next 8 quarters. Attempt the model in differences and on the original data.
2. Develop an exponential smoothing benchmark. Which model is better? OLS or ETS.

Happy forecasting!