Week 3 Notes Part 2 Practicing Linear Models

Code

Author: Brendan Gongol

Last update: 02 January, 2023

1 Overview

In this Note we will use ALL dataset to run our first linear regression model. We will learn how to perform a linear fit in practice and how to draw and review the diagnostic plots; despite simplicity of its invocation in R, regression/model fitting is a complex problem, so the diagnostic plots and other information should always be inspected before deciding whether to accept the result of model fitting.

2 Evaluating Linear Model for the First Time

Let us prepare some data for our practical exercise in linear models. We are going to use ALL dataset, and specifically we will be searching for relationship between gene expression levels and patient's time-to-remission (which we will compute as the time passed from diagnosis to the start of remission).

Hide

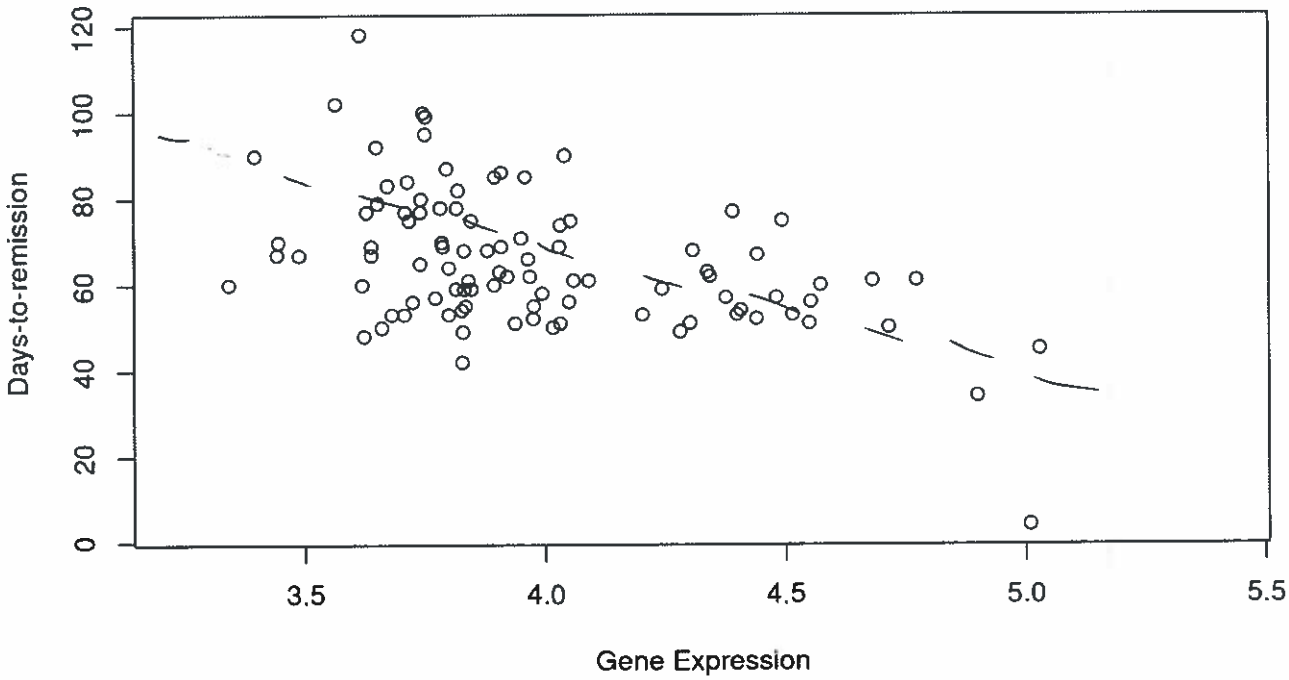
```
library(ALL); data(ALL)
ALL.pdat <- pData(ALL)
date.cr.chr <- as.character(ALL.pdat$date.cr)
diag.chr <- as.character(ALL.pdat$diagnosis)
date.cr.t <- strptime(date.cr.chr,"%m/%d/%Y")
diag.t <- strptime(diag.chr,"%m/%d/%Y")
days2remiss <- as.numeric(date.cr.t - diag.t)
plot(exprs(ALL)["34852_g_at",], as.numeric(days2remiss),
    xlab="Gene Expression", ylab="Days-to-remission")
```

clinical remission

convert into format that allows arithmetic

days to remission = date of remission − date of diagnosis

plot expression level vs # of days to reach remission



The code in the fragment above should be easily understandable by now: we load the patient information data columns for date of diagnosis and date of start of remission as character vectors into variables. In order to calculate the time interval we need to convert character strings representing dates (e.g. "5/4/1992") into some data type that allows subtracting one value from another. Fortunately, R provides just such data type for dates, so we convert character representation of dates into objects of class POSIX representing date objects (the strptime() character is such date conversion function and the only new piece here – since there

are so many different formats, we need to explicitly tell this function what format our character strings use to represent the date, see the docs). After that we compute the vector of days to remission (one value for each patient) as difference between the date in remission and date of diagnosis and finally draw a scatterplot of gene expression of the microarray probe "34852_g_at" across all patiens vs days-to-remission.

As one can see from the scatterplot, there might be some trend in the data indicating the inverse relation between expression of the specified gene and days to remission.

How can we quantify this trend and assess its significance? Here is all it takes:

Hide

```
exprs.34852 <- exprs(ALL)["34852_g_at",]
d2r.34852 <- data.frame(G=exprs.34852,D2R=days2remiss)
lm.34852.g.at <- lm(D2R~G,d2r.34852)
```

← vector of rows "3482...", all columns (patients)

← creates df with G + D2R as column names and the vectors fill in the data under each column

dependent variable ↑ ↑ independent variable

Our variables of interest X Y are gene expression levels of "34852_g_at" and days to remission, respectively and we want to fit a linear model Y ~ 1 + X (you should be familiar with this notation from Part 1 of the Notes!). All it takes is to get the data vectors and call the `lm()` command (l(inear) m(odel)), which does for us everything we need! Note

In this example based on ↑ X, Y defined above, Y = days to remission, ∴ Y = dependent variable, ∴ must put D2R~G (not G~D2R).

G	D2R
Elev1	Days1
ELV2	Days2
Elv3	Days3

- The first argument to the lm() command is the model we are fitting; formulas in R follow closely the notations adopted in statistics for specifying models. The variables that hold our data are named $D2R$ and G, so the formula is $D2R \sim G$
- We chose to pack the two variables we are fitting against each other into a data frame; this is purely for the convenience of keeping data together. When `lm()` command is given a formula $Y \sim X$ and a data frame as an additional argument, `lm()` will look for data in the columns named Y and X in that data frame. If data frame is not specified, or the columns with such names are not found in the provided data frame, `lm()` will simply look for variables (vectors) Y and X in the environment (i.e. in your session).
- In R, there is no need to explicitly specify an intercept in the formula. The fit is performed by default with an intercept, i.e. in R formula $Y \sim X$ is actually a shortcut for $Y \sim 1 + X$. If you want to perform a fit without an intercept (i.e. you require that your fitted line cross the origin), the formula must state it explicitly as $Y \sim 0 + X$

The function `lm()` returns quite a heavy-weight object of class "lm", which in addition to the fitted coefficients of the linear model contains lots of information, including full-length input data vectors, on which the model fit was performed (!). Try `names(lm)` to see the available data fields.

Let us examine the result of our first model fitting in more detail: (replace with the model you defined. in this case `names(lm.34852.g.at)`

Hide

```
coef(lm.34852.g.at)
```

```
## (Intercept)           G
##   152.59905   -21.87222
```

(Intercept) = y axis intercept; G = coefficient of our independent variable

∴ fitted linear model is

D2R = 152.59905 + -21.87... × G [+ residuals]

Hide

```
print(lm.34852.g.at)
```

```
##
## Call:
## lm(formula = D2R ~ G, data = d2r.34852)
##
## Coefficients:
## (Intercept)            G
##      152.60       -21.87
```

Hide

```
plot(exprs(ALL)["34852_g_at",],
as.numeric(days2remiss),
xlab="Gene Expression",
ylab="Days to remission")
points(d2r.34852$G[!is.na(d2r.34852$D2R)],
predict(lm.34852.g.at),
col="red",pch=20)
```

points takes 4 ~~patterns~~ arguements here, The first 2 arguments specify the x- and y- coordinates of the points being plotted!

①: x-coordinate: column G of df we created which equals vector of expression values, of that we exclude rows where D2R values are NA

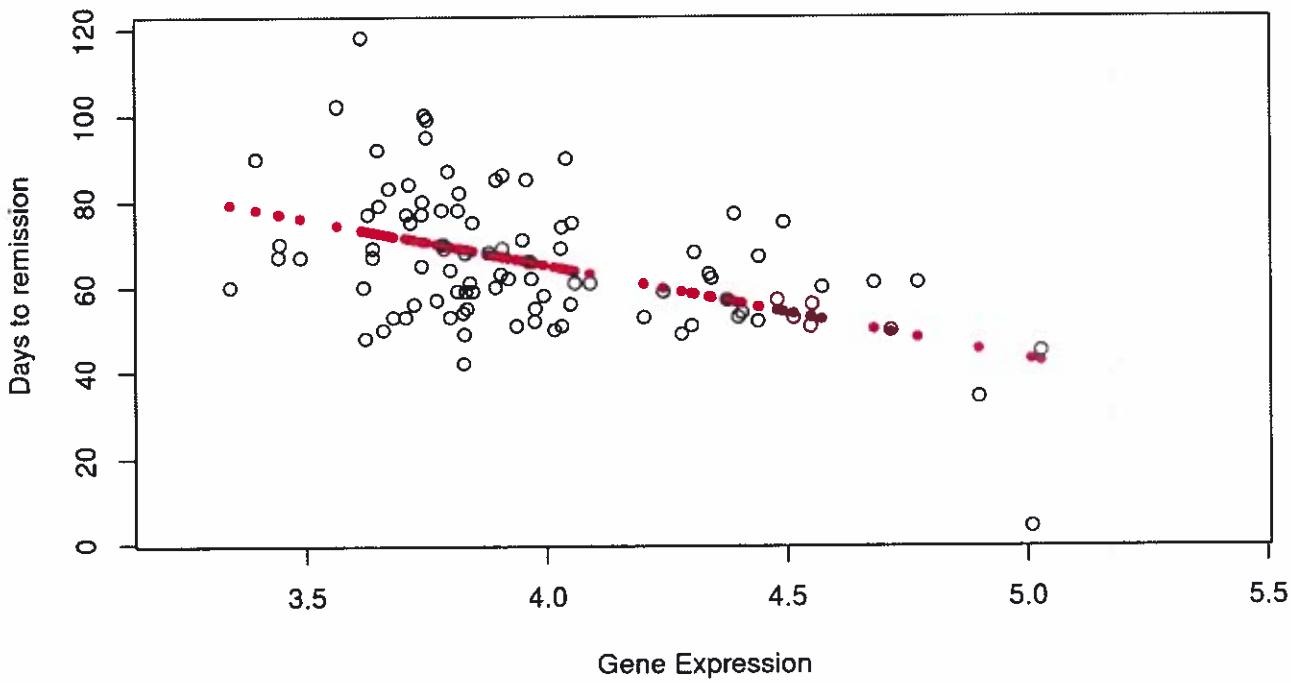
②: y-coordinate: use the linear model we created in a predict() function. ie: predict the dependent variable (D2R) based on the independent variable (G).

③ col="red"

④ pch=20

PTO

∴ when points() is called: x-axis - plotting gene expression values (without missing D2R data), Y-axis - plotting the predicted days to remission values based on the linear model.



The first command is simply a "getter" function that retrieves fitted coefficient values from the "lm" object. The values tell us that the best fit has intercept ~152.6, and that the slope (coefficient at the independent variable, i.e. the G term in our formula) equals to ~-21.9. In other words, the fitted linear dependence between D2R and G is

$$D2R = 152.6 - 21.9 * G \text{ [+ residuals]}$$

The last two commands generate a plot shown below: first we draw a scatter plot of actual data points, D2R vs G (same as above); then for each available value g of variable G we draw a red point at the value of D2R predicted for g by our fitted linear model. Note the use of

function `predict()`. When given only a fitted linear model object (object of class "lm") as an argument, this function returns the values of the response variable predicted by the fitted formula for each value of the independent (predictor) variable the model was fitted on. Remember that "lm" object stores all the input data vectors, so it is sufficient to just pass the linear model object to predict().

3 Quality Control for Model Fitting: Diagnostic Plots

As simple as model fitting can be in R, the data in real life are usually complex and imperfect, can contain outliers (that will significantly distort linear fit just like they distort estimates of mean or standard deviation), etc. It is also very important that we were fitting a predefined (parameterized) class of models. We could not possibly try fitting to our data all possible functional dependencies in the world. Instead we were specifically looking for the straight line that gives the best possible fit to our data: only the intercept and slope were our fitting parameters. It is always possible to find such a line as our previous discussion demonstrated, but the "best possible line" we could find can be still a terrible fit if the actual trend in the data is not a straight line at

all. In order to address all the above issues you are always strongly advised to (1) if possible, examine the fit visually (but if you are fitting an outcome Y on 15 independent variables $X_1, X_2, \ldots, X_15$ this might become a problem), and (2) examine quality metrics for the obtained model fit.

A very useful way for quickly checking the density distribution of a data sample visually is Quantile-Quantile Plot. This plot is built as follows. Suppose we have samples from the two distributions, f(x) and g(x), and we want to compare these distributions. We compute a large number of quantiles for both functions (hence quantile-quantile plot). Let's suppose that we use 20 quantiles for both (5%percentile, 10% percentile, 15% percentile, ...). Let the calculated quantile locations for the first function, f(x), be located at x_1 (i.e. 5% of the mass of the distribution is below x_1), x_2 (10% of the mass is below x_2 - that also means that 5% of the mass is between x_1 and x_2), etc. Let the quantile locations for the second function be x'_1, x'_2, etc. Now we plot locations of respective quantiles vs each other, e.g. we put points into a plot at $(x_1, x'_1), (x_2, x'_2), \ldots$ Clearly if the two distributions are the same, their quantiles are also the same, so the points in the QQ plot will be on a diagonal. As the differences between the distributions arise, the points in QQ-plot start deviating from diagonal, more so in the regions where differences are more pronounced.

"prime" symbol

in this case prime is the modeled or predicted value of x. so we are comparing observed vs predicted values.

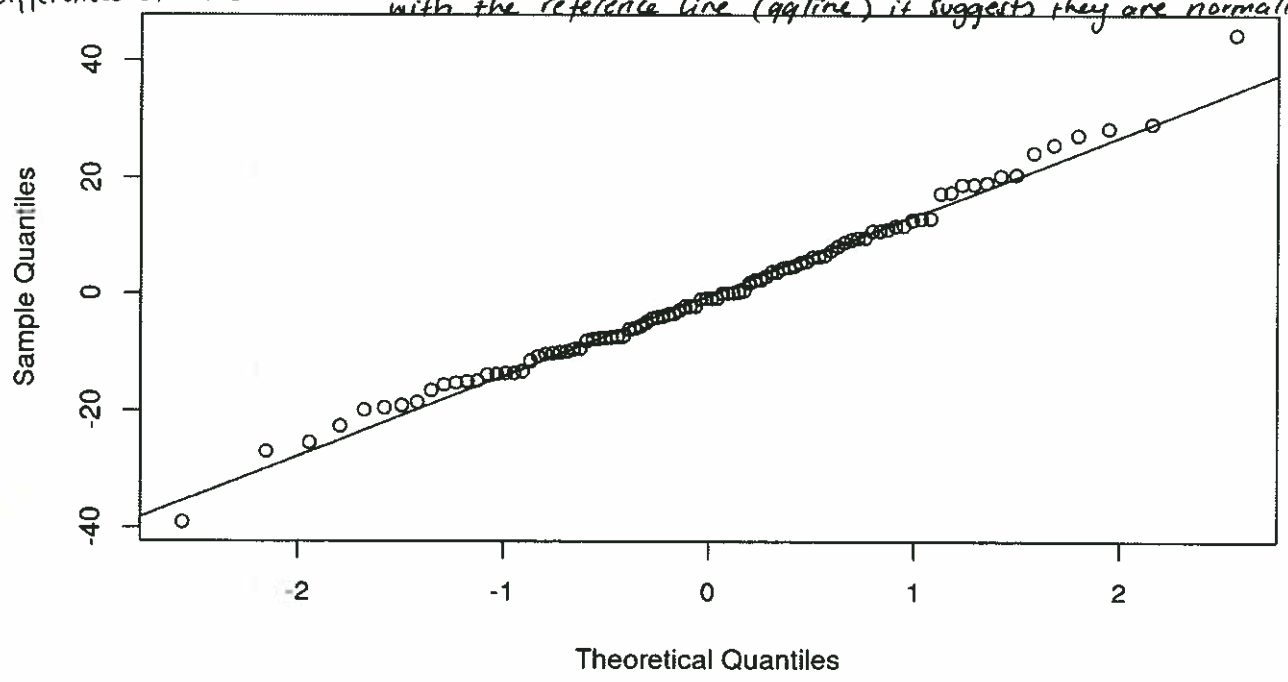
In R, a quantile-quantile plot for two samples is generated by function `qqplot()`. A special version of the function, `qqnorm()` takes only one sample as an argument and draws its QQ-plot against the normal distribution with the same mean and variance as the submitted sample. Hence the latter function provides a quick way to (visually) check a sample for normality.

In the following code fragment we extract residuals from the fitted linear model object and draw their QQ-plot against the theoretical normal (also known a "Normal QQ plot") as well as their empirical distribution.

Hide

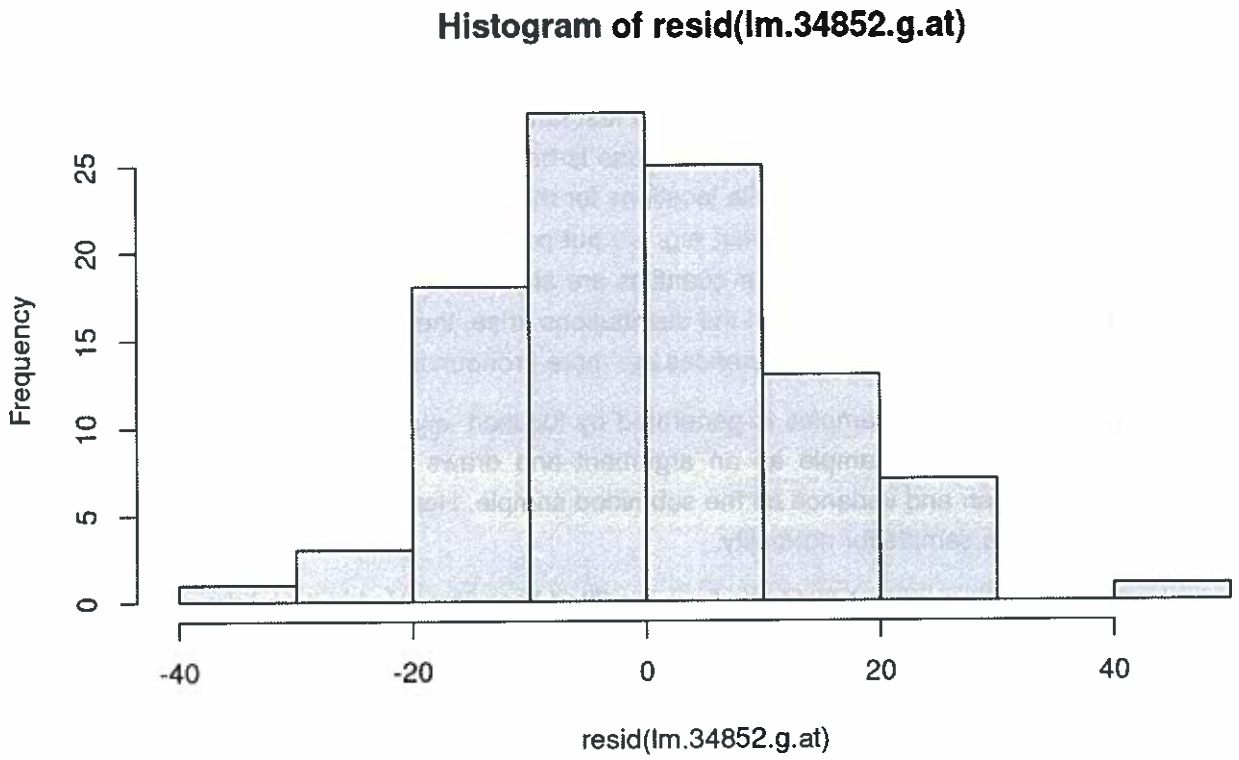
```
qqnorm(resid(lm.34852.g.at))
qqline(resid(lm.34852.g.at))
```

Usual here ↑ is to check quality of the linear model fit, specifically whether the residuals follow a normal distribution (which is a key assumption of linear regression). We look at the residuals because by definition they are the ~~observed~~ differences b/w the observed data + the values predicted by the model. - So if the residuals align with the reference line (qqline) it suggests they are normally distributed.



Hide

```
hist(resid(lm.34852.g.at))
```



Hide

```
shapiro.test(resid(lm.34852.g.at))$p.value
```

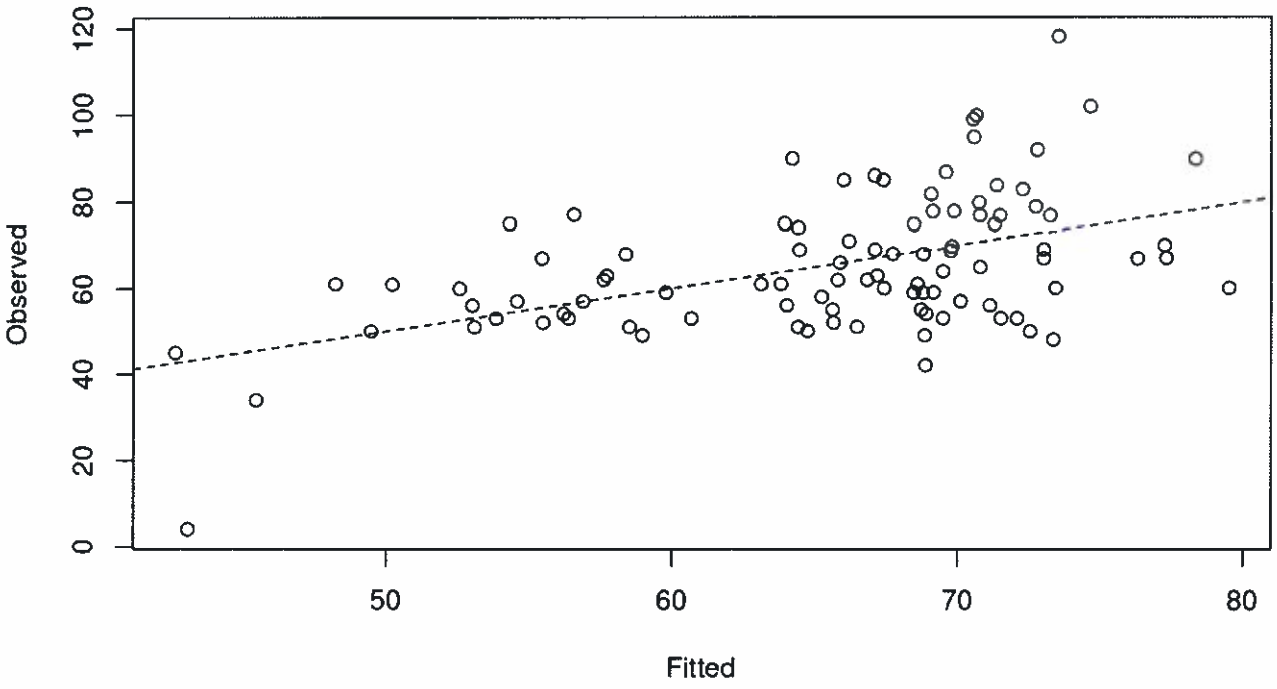
```
## [1] 0.656474
```

As it can be seen from the figures, the distribution of residuals seems to be very close to normal, and this is confirmed by Shapiro test (last command in the code fragment above). The purpose of the latter test is to check the normality of the distribution of the sample passed to it as an argument (as compared to the t-test that as we know only checks for the location of the sample in fact). Normality of the distribution from which the sample at hand was drawn is the null hypothesis for this test, so the large p-value means that there was no evidence in the sample to reject that null. On the contrary, small p-value would mean that the sample is too extreme and it is unlikely that it follows/was drawn from a normal distribution. In the context of model fitting, remember that optimizing parameters by minimizing sum of squares of residuals (this is what lm() does) is based on the assumption of the normality of the noise distribution. Strongly non-normal distribution of the residuals would suggest some problem with the model fitting, and/or would indicate that fitted coefficients do not necessarily provide the best possible fit, because the model was evaluated under the wrong assumptions.

So far we looked at the overall distribution of (all) the residuals. Next, let us see how the residuals are distributed across the range of values of the response variable:

Hide

```
plot(fitted(lm.34852.g.at), d2r.34852[!is.na(d2r.34852$D2R),"D2R"],
     xlab="Fitted",ylab="Observed")
abline(0,1,lty=2)
```



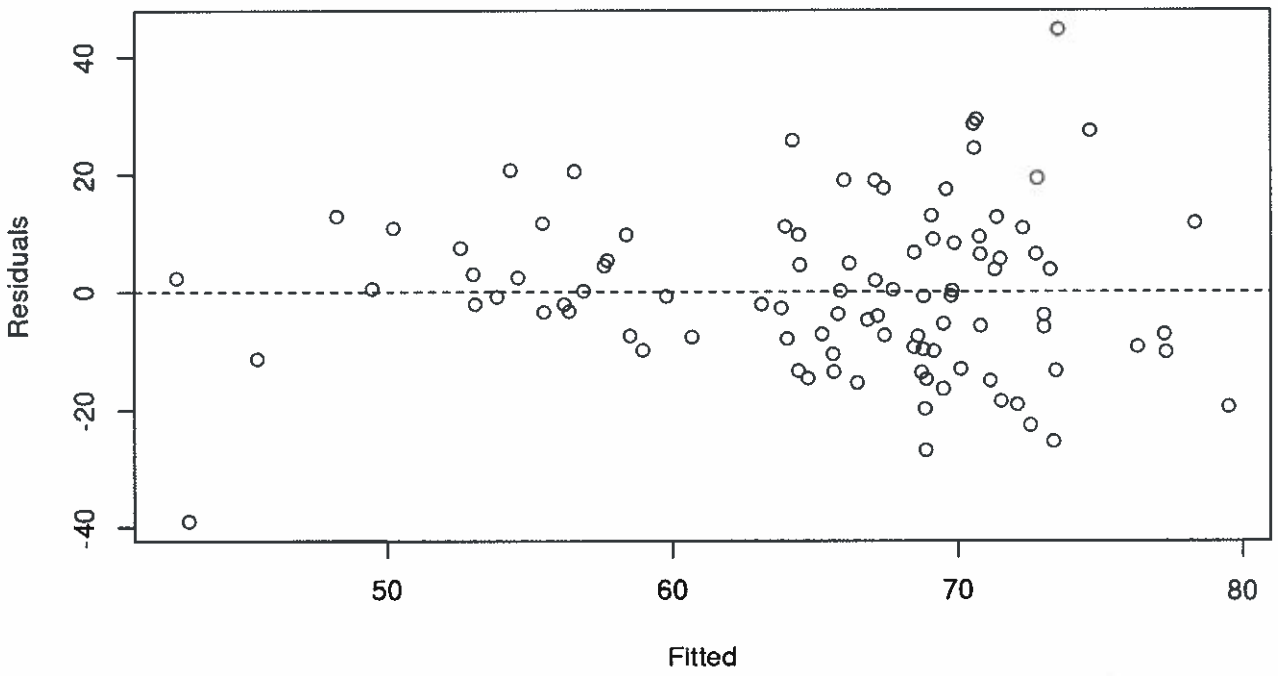
In the code fragment above, we use `fitted()` function in order to retrieve the fitted (predicted) value of $D2R$ at each value in G (the "lm" object carries fitted values around as well, so we can simply retrieve them). In this context the functions `fitted()` and predict() (which we used before) are interchangeable. `predict()` is more powerful, since it can also use the fitted model object in order to apply the formula and predict values of the response variable for new values of the predictor variable, which are not part of the sample data originally used to fit the model. For each fitted, or predicted, value (i.e. $D2R = a * G + b$, without the error term), we plot it against the $D2R$ value actually measured in that patient (Observed value). If our predictions were perfect, there were no noise and all residuals were equal to 0, the fitted values would be equal to observed, so that all the points would lie on the y=x diagonal. In the last command (`abline()`) we add this line to the plot in order to see how far the observed values are from the fitted ones. In this plot, residuals ($y_observed - y_fitted$) are equal to the distances of the points from the $y = x$ line in vertical direction.

As the figure above shows, there are some potential problems with the fit. While the overall distribution of residuals is normal, the subset of residuals at higher D2R values tends to have much higher variance than the subset of residuals at low D2R values. In the case of a good homogeneous fit, the residuals should have the same/similar variance across the whole range of values. What our plot shows is that there is clear difference between low and high D2R values: we can predict the former's much more accurately (smaller residuals) than the latter's. Among other reasons, this can suggest that there might be another variable (another gene? Subtype of the disease? We do not know!) that also contributes to $D2R$, so that when we leave it out of scope the predictive power of the remaining variable G varies across the range of D2R values. Or it may suggest that data transformation is required (e.g. to log $D2R$, since log stabilizes variance). In any case non-homogeneity of the fit is something we do not want to see in general.

Another type of the plot that conveys the same information (probably in slightly more convenient and comprehensible way) is Residuals-vs-Fitted. As the name suggests, instead of observed vs fitted, we simply plot residual values vs corresponding fitted values:

Hide

```
plot(fitted(lm.34852.g.at),
resid(lm(D2R~G,d2r.34852)),
xlab="Fitted",ylab="Residuals")
abline(h=0,lty=2)
```

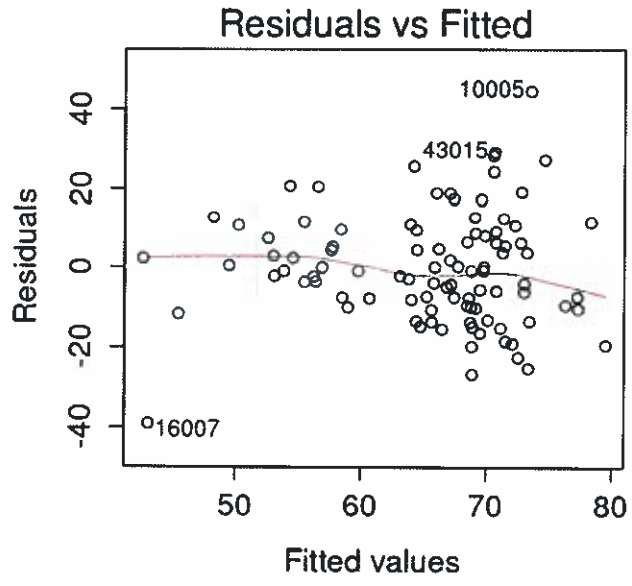


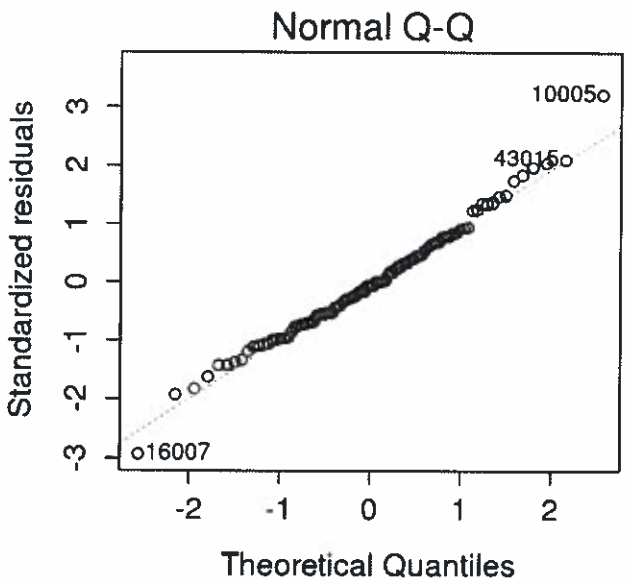
The above plot conveys the same effect as we have already observed in the previous plot: the "funnel" shape of point distribution in the residuals vs fitted plot suggests strong non-uniformity of the fit (note that the red lines are just the guide for the eye and were not plotted in R).

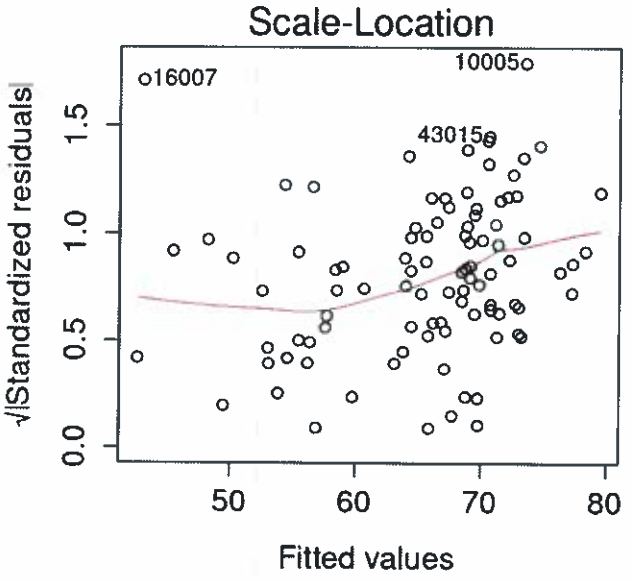
Finally, there exists an overloaded version of the function `plot()` for the linear model objects. If you simply execute plot() command with "lm" object as an argument, a series of diagnostic plots will be generated as shown below:

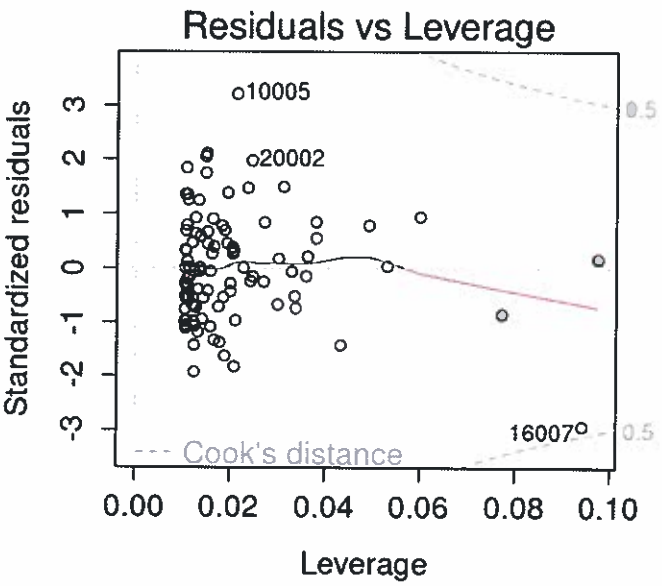
Hide

```
old.par <- par(mfrow=c(2,2),ps=16)
plot(lm.34852.g.at)
```









Hide

```
par(old.par)
```

The figure generated by these commands is shown below. The top two plots are residual-vs-fitted and Normal-QQ-plot, which we are already familiar with and were able to reproduce manually. The next two plots are also interesting. The interpretation of the scale-location plot is similar to residuals vs fitted: it is a related plot except that on y axis we use square root of standardized residuals instead of residuals. Square root suppresses skewness in the data, so this plot provides a slightly different view at the same residuals vs fitted relation. This plot should also exhibit no patterns for a good fit. Finally, residuals vs leverage plot shows by how much

the fit is affected by individual outlier points. The situation is very similar to the population mean: if there are a few outliers in the data, they strongly and disproportionately affect the mean and can pull it quite far away from the true value. Linear regression fits are similar in that respect: a single outlier may have undue effect

on the whole fit (in terms of fitted parameters of the model). In an attempt to characterize the effect of outliers, leverage is introduced. The leverage values indicate whether or not X values for a given observation are outlying (far from the main body of the data). A high leverage value indicates that the particular observation is distant from the center of the X observations. The fact that X data point is an outlier in X variable (high leverage) means that it can have a strong and undue influence on the whole model fit, but it does not mean that it necessarily does. This is why residuals are plotted vs leverage in the last diagnostic plot: a high leverage point that also has high residual may be a cause of problems for the whole fit (i.e. we failed to fit it well and it is still pulling the fit towards itself). In order to provide better metric for comparison of different suspicious points in the data (is a datapoint with leverage 1, residual 0.7 better than the one with leverage 0.7 and residual 1?) the Cook's distance contours are plotted in the same graph with red and provide a useful guide. Cook's distance measures the effect of deleting a single observation: what would happen to parameters estimated for our model then? Data points with high Cook's distance have large effect on the estimated parameter values. All points on the same contour have the same "distance" so their effect must be comparable. We can see, for instance, from the leverage plot that probably the most problematic point in the dataset is the one labeled 16007: it has second largest leverage, large residual, and also the largest Cook's distance among all other points!

We have generated so far a number of useful plots, but we did not offer yet much beyond the visual inspection of the diagnostics. There is of course a quantitative way to characterize the significance of the effect of the independent (predictor) variable on the response variable in a linear model. We will discuss the mechanics behind this calculation next week; for now it suffices to mention as an introduction and transition into the next week material, that anova() function applied to a fitted linear model will calculate significance levels for the fit. In the code fragment below we demonstrate that despite some problems with the fit we have observed in the diagnostic plots, the effect of our selected gene G on the D2R is highly significant (p-value of 2.9*10-7 !!). The meaning of p-value in this context is as follows: if in reality there were no association between G and D2R (this is our null hypothesis), what are the chances to sample randomly from G and D2R and observe the effect of the magnitude we observe or even greater? While we leave the question of how we define "magnitude of the effect" in this case until next week, we note that probability 10-7 to observe the effect by chance is pretty impressive. Note that the summary printed by the command also shows the sum of squares of the residuals. If we were to compare different fits (e.g. with different models), we would want to look both at significance of the fit and at how little unexplained variance (sum of squares of the residuals) is left after we fitted the model.

The last command in the fragment below shows that while anova() prints nicely formatted report into the screen, it still returns an object which can be programmatically addressed (try `names(anova( lm.34852.g.at))`), and from which required data can be pulled out.

Hide

```
anova(lm.34852.g.at)
```

```
## Analysis of Variance Table
##
## Response: D2R
##           Df  Sum Sq Mean Sq F value    Pr(>F)
## G          1  6000.1  6000.1  30.498 2.959e-07 ***
## Residuals 94 18493.6   196.7
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Hide

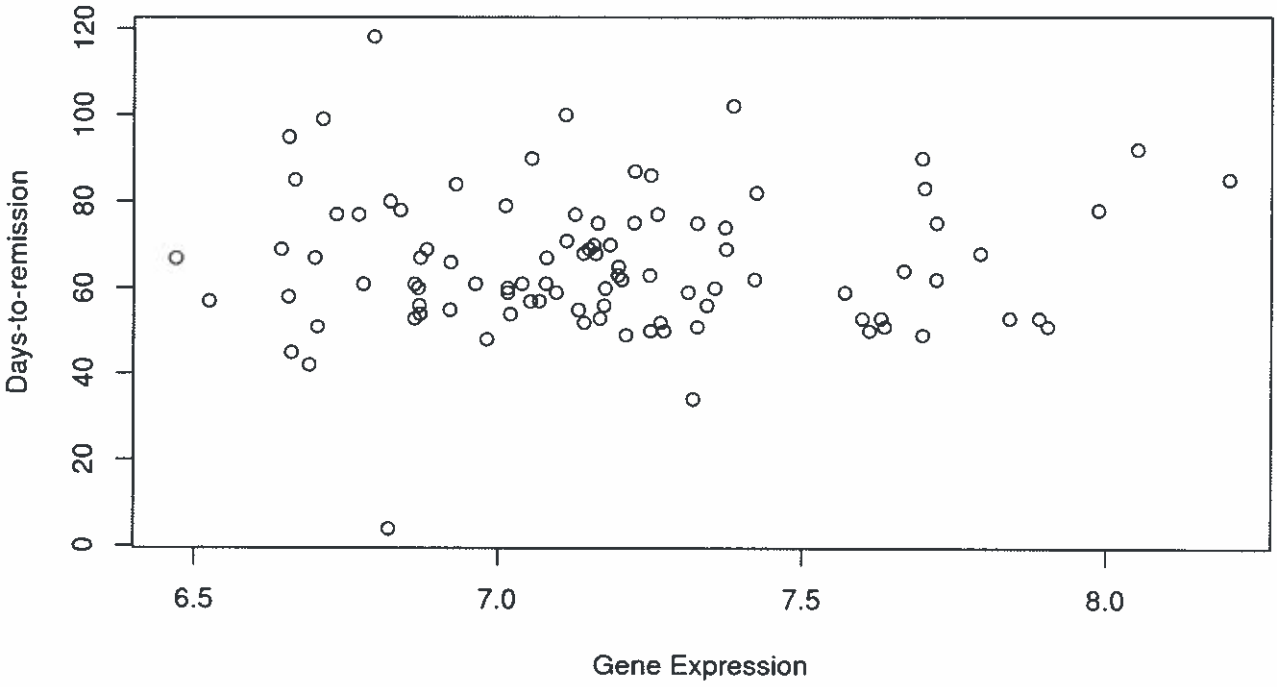
```
anova(lm.34852.g.at)$"Pr(>F)"[1] # we can retrieve just the p-value
```

```
## [1] 2.958899e-07
```

Note that not all effects are necessarily significant (if that's what you started thinking after the previous example with its tremendously low p-value). Let us choose a different gene and try fitting the same model $D2R \sim G$ for that gene. The result shows no effect whatsoever:

Hide

```
plot(exprs(ALL)["37138_at",],
 as.numeric(days2remiss),
 xlab="Gene Expression",
 ylab="Days-to-remission")
```



Hide

```
anova(lm(D2R~G,data.frame(
 G=exprs(ALL)["37138_at",],
 D2R=as.numeric(days2remiss))))
```

```
## Analysis of Variance Table
##
## Response: D2R
##            Df  Sum Sq Mean Sq F value Pr(>F)
## G           1    11.1   11.09  0.0426  0.837
## Residuals 94 24482.6  260.45
```