Regression in R

Intro to Stats, Spring 2017

Prof. Gaston Sanchez

Learning Objectives

- How to run a regression analysis in R
- Getting to know the function `lm()`
- How add the regression line to a scatter diagram
- How to graph the plot of residuals

About

In this tutorial I will show you how to:

- perform a Regression analysis in R.
- interpret the output and main results.
- use R to obtain the results the way they are calcuated in the book.

Introduction

To make sure we are all on the same page, I should start with a brief discussion about the term "Regression". This term was coined by Sir Francis Galton and introduced in his 1886 paper *Regression Towards Mediocrity in Hereditary Stature*. He used the word "regression" to describe a phenomenon that he observed when analyzing height data of parents and their adult children. What he noticed was that exceptionally tall parents had children who were, on average, less tall; and exceptionally short parents had children who were, on average, less short. Because of the connotations of the word "mediocrity", statisticians later modified the regression statement as: *regression towards the mean*.

For better or for worse, the term regression has evolved and become broader over the years. Nowadays people use the word regression in a more loosely way. The most common terms that you will probably find are "regression analysis" and "regression model" or "regression modeling".

The core idea behind virtually all regression tools is to predict a (quantitative) response variable in terms of one or more predictor variables. In its simplest version, the regression method is used for explaining or modeling the relationship between a single variable Y, called the *response*, *outcome*, *output* or *dependent* variable; and one *predictor*, *input*, *independent* or *explanatory* variable X. This version is commonly referred to as **simple linear regression**, and it is actually the method presented in the FPP book (chapters 10-12)—although it is not explicitly called like that. The reasons why statisticians call it *simple linear regression* are:

- "simple" because there is only one Y and one X
- "linear" because the mathematical model is expressed with a line equation
- "regression" because X is used to predict Y

Note: Pretty much the word "regression" has become synonym for prediction. The word "prediction" implying that the response variable is quantitative. If the response variable to be predicted is of categorical nature, then we talk about "classification". The type of regression we will discuss is the one in which both X and Y are quantitative variables.

Data

To illustrate the regression ideas, we are going to use a data set collected by English mathematician and biostatistician Karl Pearson (1857-1936). Among other things, Pearson was Galton's protégé, he founded the world's first university statistics department at University College London in 1911, and he is considered one of the founding fathers of modern-day Statistics.

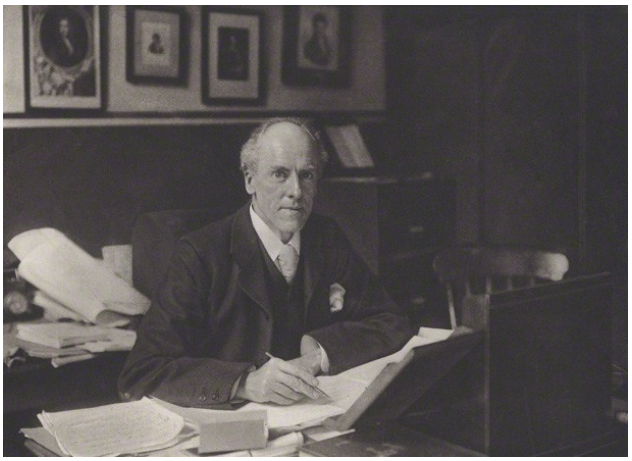


Figure 1: Karl Pearson (source: wikipedia)

The data set is in the csv file `pearson.csv` (in the github repository), and it contains the heights of 1078 fathers, and their adult sons.

```
# assembling the URL of the CSV file
# (otherwise it won't fit within the margins of this document)
repo = 'https://raw.githubusercontent.com/ucb-introstat/introstat-spring-2017/'
datafile = 'master/data/pearson.csv'
url = paste0(repo, datafile)

# read in data set
dat = read.csv(url)
```

If you are having problems importing the data from github, you can try this other source:

```
# another way to read in the data
dat = read.csv('http://www.math.uah.edu/stat/data/Pearson.csv')
```

Univariate Exploration

The data frame `dat` contains 1078 rows, and 2 columns.

- `Father`: height of the father (in inches)
- `Son`: height of the son (in inches)

As it is customary, the first thing when analyzing data (one variable at a time), is to obtain summary statistics and look at the distributions:

```
# basic summary statistics
summary(dat)
```

```
##      Father           Son
##  Min.   :59.00   Min.   :58.50
##  1st Qu.:65.80   1st Qu.:66.90
##  Median :67.80   Median :68.60
##  Mean   :67.69   Mean   :68.68
##  3rd Qu.:69.60   3rd Qu.:70.50
##  Max.   :75.40   Max.   :78.40
```

```
# number of rows
n = nrow(dat)

# SD of Father
sqrt((n-1)/n) * sd(dat$Father)
```

```
## [1] 2.744553
```

```
# SD of Son
sqrt((n-1)/n) * sd(dat$Son)
```

```
## [1] 2.814888
```

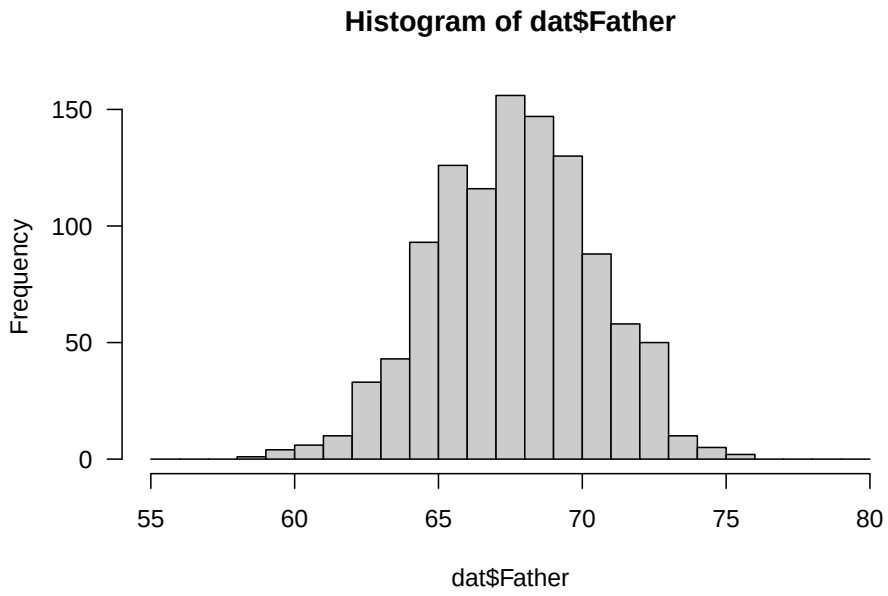
Histograms

For convenience purposes, to plot histograms for the `Father` and `Son` heights, we can define a vector of class intervals or bins for the argument `breaks` inside the `hist()` function. Looking at the output of `summary()`, we can take a minimum of 55, and a maximum of 80 to create a vector of `bins` with a numeric sequence like this:

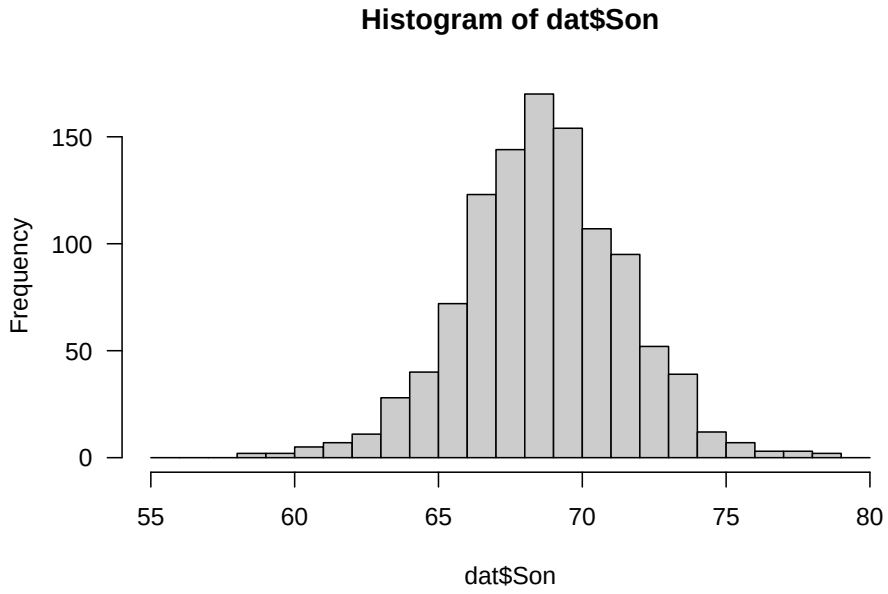
```
bins = seq(from = 55, to = 80, by = 1)
```

Having defined `bins`, we can then graph the histograms:

```
hist(dat$Father, breaks = bins, las = 1, col = 'gray80')
```




```
hist(dat$Son, breaks = bins, las = 1, col = 'gray80')
```

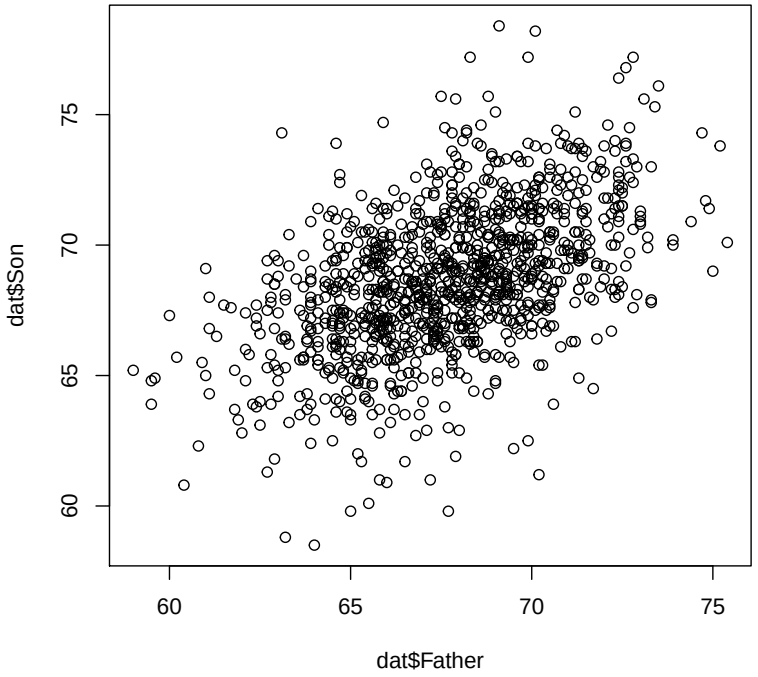


Both histograms have nice symmetric bell-shaped distributions. Keep in mind that these are classic textbook examples of variables that follow the normal curve.

Scatter Plot

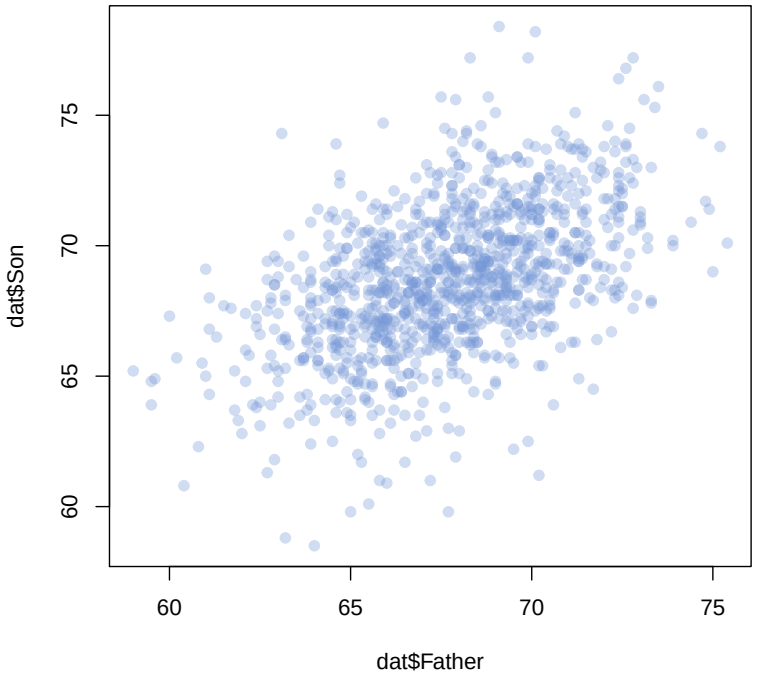
The next step is to get a picture of the relationship between the `Father` and `Son` heights. The quickest option is to create a scatter diagram with the function `plot()`:

```
plot(dat$Father, dat$Son)
```



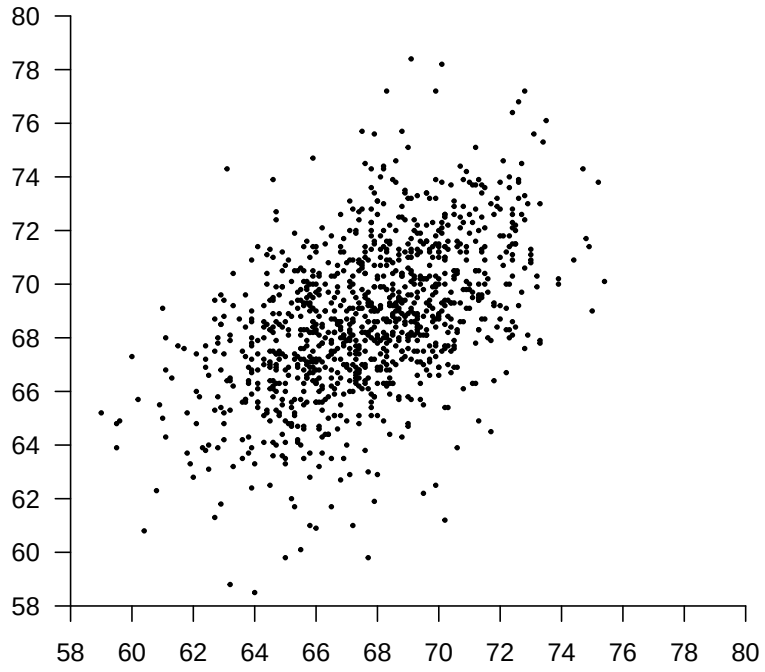
To obtain a better visual display, you can modify the point character `pch` value, and add a color with some transparency using hexadecimal notation:

```
plot(dat$Father, dat$Son, pch = 19, col = '#7396D655')
```



If you want to get a scatter diagram like the one displayed in the textbook (see page 120), then run the following lines of code:

```
plot.new()
plot.window(xlim = c(58, 80), ylim = c(58, 80))
points(dat$Father, dat$Son, pch = 20, cex = 0.5)
axis(side = 1, pos = 58, at = seq(58, 80, 2))
axis(side = 2, las = 1, pos = 58, at = seq(58, 80, 2))
```



The way in which this plot is constructed is a bit special. This approach to create graphs in R uses only low-level functions that give you total control of the appearance of graphical elements:

- `plot.new()` starts a new plot frame
- `plot.window()` is used to set up the coordinates of the axes
- `points()` is used to actually plot the dots
- `axis()` is used to plot both the x-axis and the y-axis.

The most important thing to pay attention to when inspecting the scatter diagram is to check whether the cloud of points follows a linear pattern. When this is the case, it makes sense to use of a line to summarize the relationship between the analyzed variables.

Regression Method

The goal is to find a line that fits the data well. In other words, we want to find a line that is as close as possible to all the data points. The typical algebraic equation of a line is $y = mx + b$, where m represents the slope, and b represents the intercept. However, when working within a

regression framework, we use a slightly different notation for the equation of a line. The most standard notation is:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

where β_0 and β_1 are two unknown constants that represent the intercept and the slope of a line, respectively. In turn, ϵ is the error term that accounts for the spread in the cloud of points that is not captured by the model. Together, β_0 and β_1 are known as the model *coefficients* or *parameters*.

Often, you will find an alternative represention of the model as:

$$Y \approx \beta_0 + \beta_1 X$$

you migth read "$\approx$" as *"is approximately modeled as."* This is a very important detail: we are assuming that there is approximately a linear relationship between X and Y.

We will sometimes describe the equation $Y \approx \beta_0 + \beta_1 X$ by saying that we are *regressing* Y *on* X (or Y *onto* X).

In our working example, X represents the height of parents, `Father`, and Y represents height of sons, `Son`. Then we can regress `Son` onto `Father` by fitting the model:

$$\texttt{Son} \approx \beta_0 + \beta_1 \texttt{Father}$$

Once we have used the data to produce estimates b_0 and b_1 for the model coefficients, we can predict the heights of sons based on a particular value of a father's height by computing:

$$\hat{y} = b_0 + b_1 x$$

Here we use a *hat* symbol (i.e. the caret), `^`, to denote the predicted value. That is, $\hat{y}$ indicates a prediction of Y on the basis of $X = x$.

Estimating the Coefficients with `lm()`

In practice, β_0 and β_1 are unknown. To find estimates b_0 and b_1, we must use the data in X and Y. In other words, we want to find an intercept b_0 and a slope b_1 such that the resulting line is as close as possible to all the data points. There are a number of ways of measuring *closeness*. However, by far the most common approach involves minimizing the *least squares* criterion.

In R, the function that allows us to find the intercept b_0 and the slope b_1 is `lm()` (i.e. linear model). I should say that this function is a general function that works for various types of linear models, not just simple linear regressions.

Because the data is already in a data frame, you can use `lm()` as follows:

```
# run regression analysis
reg = lm(Son ~ Father, data = dat)
```

The first argument of `lm()` consists of an R formula: `Son ~ Father`. The tilde, `~`, is used to indicate that `Son` *depends* or is *described* by `Father`. The second argument, `data =`, is used to indicate the name of the data frame that contains the variables `Son` and `Father`, which in this case is the object `dat`.

In this example we are storing the output of `lm()` in the object `reg`. Technically, `reg` is an object of class `"lm"`. Let's take a look at `reg`:

```
# default output
reg
```

```
##
## Call:
## lm(formula = Son ~ Father, data = dat)
##
## Coefficients:
## (Intercept)       Father
##      33.893        0.514
```

The first part of the output simply tells you the command used to run the analysis, in this case: `lm(formula = Son ~ Father, data = dat)`.

The second part of the output shows information about the regression coefficients. The intercept is 33.893 and the slope is 0.514. Observe the names used by R to display the intercept b_0, and the slope b_1. While the intercept has the same name `(Intercept)`, the slope is displayed with the name of the associated variable, `Father`.

As you can tell, the printed output of `reg` is kind of minimalist. However, `reg` contains more information. To see a list of the different components in `reg`, use the function `names()`:

```
names(reg)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

As you can tell, `reg` contains many more things than just the `coefficients`. Here's a short description of each of the output elements:

- `coefficients`: a named vector of coefficients.
- `residuals`: the residuals, that is, response minus fitted values.
- `fitted.values`: the fitted mean values.
- `rank`: the numeric rank of the fitted linear model.
- `df.residual`: the residual degrees of freedom.
- `call`: the matched call.
- `terms`: the terms object used.
- `model`: if requested (the default), the model frame used.

To inspect what's in each component, type the name of the regression object, `reg`, followed by the `$` dollar operator, followed by the name of the component. For example, to inspect the `coefficients` run this:

```
# regression coefficients
reg$coefficients
```

```
## (Intercept)      Father
##  33.8928005   0.5140059
```

For the purposes and scope of this course, the important output of an `"lm"` object are the elements `coefficients`, `residuals`, and `fitted.values`.

Summary output

Let me repeat it: the object `reg` is a special type of object. More precisely, `reg` is an object of class `"lm"`—linear model. For this type of R object, you can use the `summary()` function to get additional information and diagnostics:

```
# summarized linear model
sum_reg = summary(reg)
sum_reg
```

```
##
## Call:
## lm(formula = Son ~ Father, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.8910 -1.5361 -0.0092  1.6359  8.9894
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 33.89280    1.83289   18.49   <2e-16 ***
## Father       0.51401    0.02706   19.00   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.438 on 1076 degrees of freedom
## Multiple R-squared:  0.2512, Adjusted R-squared:  0.2505
## F-statistic: 360.9 on 1 and 1076 DF,  p-value: < 2.2e-16
```

The information displayed by `summary()` is the typical output that most statistical programs provide about a simple linear regression model. There are four major parts:

- `Call`: the command used when invoking `lm()`.

- `Residuals`: summary indicators of the residuals.
- `Coefficients`: table of regression coefficients.
- Additional statistics: more diagnostics toosl.

In the same way that `lm()` produces `"lm"` objects, `summary()` of `"lm"` objects produce `"summary.lm"` objects. This type of objects also contain more information than what is displayed by default. To see the list of all the components in `sum_reg`, you can use again the function `names()`:

```
names(sum_reg)
```

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

Computing Regression Coefficients

Let $\hat{y}_i = b_0 + b_1 x_i$ be the prediction for Y based on the ith value of X. Then $e_i = y_i - \hat{y}_i$ represents the ith *residual*. We define the **residual sum of squares** (RSS) as

$$RSS = e_1^2 + e_2^2 + \cdots + e_n^2$$

The **least squares** approach chooses b_0 and b_1 to minimize the RSS. The solution is given by:

$$b_1 = r\frac{SD_y}{SD_x}$$

and

$$b_0 = avg_y - b_1 \times avg_x$$

where:

- r is the correlation coefficient.
- avg_x is the average of X.
- avg_y is the average of Y.
- SD_x is the standard deviation of X.
- SD_y is the standard deviation of Y.

You can compare the coefficients given by `lm()` with our own calculated b_1 and b_0 according to the previous formulas. First let's get the main ingredients:

```
# number of values (to be used for correcting SD+)
n = nrow(dat)

# averages
```

```r
avg_x = mean(dat$Father)
avg_y = mean(dat$Son)

# SD (corrected SD+)
sd_x = sqrt((n-1)/n) * sd(dat$Father)
sd_y = sqrt((n-1)/n) * sd(dat$Son)

# correlation coefficient
r = cor(dat$Father, dat$Son)
```

Now let's compute the slope and intercept, and compare them with `reg$coefficients`

```r
# slope
b1 = r * (sd_y / sd_x)
b1
```

```
## [1] 0.5140059
```

```r
# intercept
b0 = avg_y - (b1 * avg_x)
b0
```

```
## [1] 33.8928
```

```r
# compared with coeffs
reg$coefficients
```

```
## (Intercept)      Father
##  33.8928005   0.5140059
```

Are residuals homoscedastic?

As you know, the main assumption in a simple regression analysis is that X and Y are approximately linear related. This means that we can use a line as a good summary for the cloud of points. For a line to able to do a good summarizing job, the amount of spread around the line should be fairly the same (i.e. constant). This requirement has a very specific—and rather ugly—name: **homoscedasticity**; which simply means "same scatter". Visually, homoscedascity comes in the form of the so-called football-shaped cloud of points. Or in a more geometric sense, cloud of points with a chiefly elliptical shape.

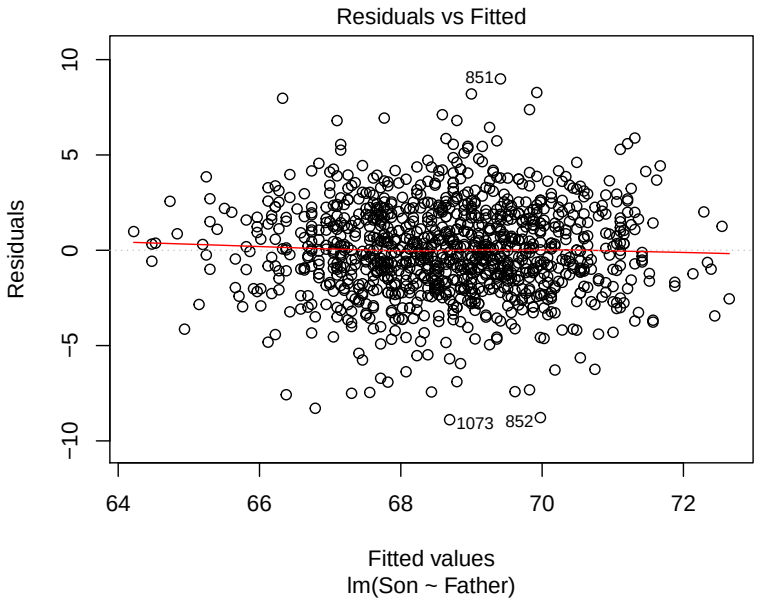
The `"lm"` object `reg` contains the vector of redisuals (see `reg$residuals`). The residuals from the regression line must average out to 0. To confirm this, let's get their average:

```r
mean(reg$residuals)
```

```
## [1] 2.050079e-16
```

You can take a look at the *residual plot* by running this command:

```
# residuals plot
plot(reg, which = 1)
```

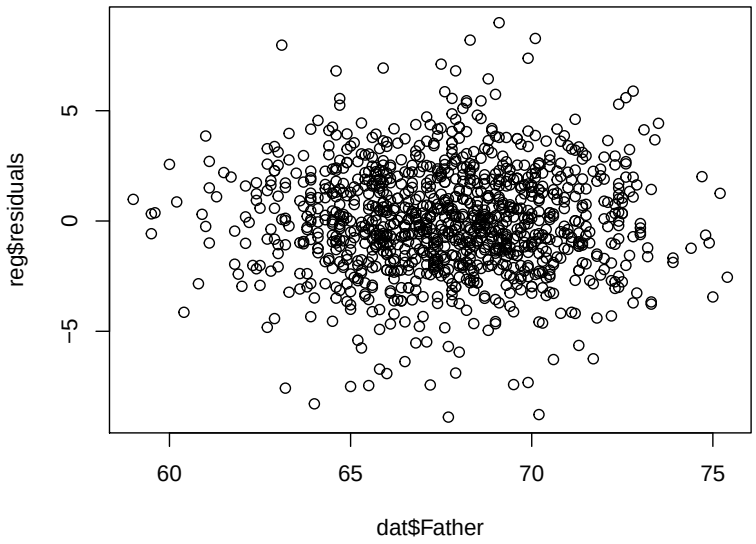


which is equivalent to this other command:

```
# equivalently
plot(reg$fitted.values, reg$residuals)
```

This residual plot is not exactly the same that the book describes (pages 187-188). To plot the residuals like the book does, you would need to use the `Father` in the x-axis:

```
# residuals plot
plot(dat$Father, reg$residuals)
```



The difference is only in the scale of the horizontal axis. But the important part in both plots is the shape of the cloud. As you look across the residual plot, there is no systematic tendency for the points to drift up or down. The red line displayed by `plot(reg, which = 1)`, is a regression line for the residuals. When residuals are homoscedastic, this line is basically a horizontal line.

Predicting Values

As I mentioned in the introduction of this tutorial, regression tools are mainly used for prediction purposes. This means that we can use the estimated regerssion line $\texttt{Son} \approx b_0 + b_1\texttt{Father}$, to predict the height of son given a particular Father's height.

For example, if a father has a height of 71 inches, what is the predicted son's height? One way to answer this question using R is like this:

```
# predict height of son with a 71 in. tall father
b0 + b1 * 71
```

```
## [1] 70.38722
```

But you can also use the `predict()` function. The first argument must be an `"lm"` object; the second argument must be a data frame containing the values for `Fater`:

```
# new data (must be a data frame)
newdata = data.frame(Father = 71)

# predict son's height
predict(reg, newdata)
```

```
##        1
## 70.38722
```

If you want to know the predicted values based on several `Father`'s heights, then do something like this:

```
more_data = data.frame(Father = c(65, 66.7, 67, 68.5, 70.5, 71.3))

predict(reg, more_data)
```

```
##        1        2        3        4        5        6
## 67.30318 68.17699 68.33120 69.10221 70.13022 70.54142
```