MEM52220 - Applied Econometrics

Nicolas Reigl

2018-10-25

Contents

Introduction

Welcome to MEM5220 - Applied Econometrics. This handout is designed for the use with MEM5220 - Applied Econometrics at Tallinn University of Technology. Note that this workbook is still under heavy development!

Prerequisites

A basic knowledge of the R [**?**] programming language is required.

In order to reproduce the examples in this script You need the statistical software package R. Additionally we recommend using the RStudio integrated developer environment (IDE) which will improve Your R working experience.

Installation of R and RStudio

Go to the following webpages and download the software adequate for Your operating system:

1. The Statistical Software Package R:
2. The GUI RStudio:

Install R first and then proceed with RStudio. Afterwards start RStudio. For accessing packages and datasets from github we need to install the **devtools** package. On Windows, download and install Rtools, and devtools takes care of the rest. On Mac, install the Xcode command line tools. On Linux, install the R development package, usually called r-devel or r-base-dev.

Resources

Our primary resource is **?** [1]. The book is available as free online version. For theoretical concepts I refer to **?**[2].

Standing on the shoulders of giants

This lecture material would not possible without many other open-source econometrics teaching materials and of course the R package developers. In addition to the main resources, examples and code of this workbook have been drawn from a number of free econometrics eBooks, blogs, R-vignette help pages and other researchers teaching materials.

[1] Heiss (2016) builds on the popular Introductory Econometrics by Wooldridge (2016) and demonstrates how to replicate the applications discussed therein using R.

[2] Introductory Econometrics: A Modern Approach.

Attribution

Chapter 1.1.2 Simulating SLR is based on **?** but adds parts of **?** and **?** teaching materials. Chapter 1.2.2 includes additonal material from **?**. Chapter 1.4 Heteroskedasticity has been amended with material from **?** blog.

- Using R for Introduction to Econometrics, **?**
- Applied Statistics with R, **?**
- Principles of Econometrics with R, **?**
- Introduction to Econometrics with R, **?**
- Broadening Your Statistical Horizons

Software information and conventions

The R session information when compiling this book is shown below:

```
sessionInfo()
```

```
## R version 3.5.1 (2018-07-02)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.6
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_3.5.1  backports_1.1.2 magrittr_1.5    bookdown_0.7.20
##  [5] rprojroot_1.3-2 tools_3.5.1     htmltools_0.3.6 rstudioapi_0.8
##  [9] yaml_2.2.0      Rcpp_0.12.19    stringi_1.2.4   rmarkdown_1.10
## [13] knitr_1.20      stringr_1.3.1   xfun_0.3        digest_0.6.18
## [17] evaluate_0.12
```

I do not add prompts (`>` and `+`) to R source code in this book, and I comment out the text output with two hashes `##` by default, as you can see from the R session information above. This is for your convenience when you want to copy and run the code (the text output will be ignored since it is commented out). Package names are in bold text (e.g., **rmarkdown**), and inline code and function arguments are formatted in a typewriter font (e.g., `knitr::knit('foo.Rmd')`). Function names are followed by parentheses (e.g., `bookdown::render_book()`). The double-colon operator `::` means accessing an object from a package. Content with **Note**, **Your turn** and **Overthinking** is surrounded by two horizontal lines. **Overthinking** sections serve as additional information for the interested reader but will not be covered in detail in class.

Acknowledgements

I thank Kadri Männasoo and Juan Carlos Cuestas for proofreading and their useful comments.

Chapter 1

Linear Regression

To load the dataset and necessary functions:

```
# This function 1. checks if the packages are installed. 2. It installs the packages if they were not i
# devtools::install_github("ccolonescu/PoEdata")
PACKAGES<-c("PoEdata", # R data sets for "Principles of Econometrics" by Hill, Griffiths, an d Lim, 4e,
            "wooldridge",  # Wooldrige Datasets
            "tidyverse",  # for data manipulation and ggplots
            "broom",  # Tidy regression output
            "ggpubr",  # Multiple ggplots on a page. Note that, the installation of ggpubr will automat
            "ggfortify", # Simple ggplot recipe for lm objects)
            "plot3D",  #  3D graphs
            "car", # Companion to applied regression
            "knitr", # knit functions
            # "kableExtra", # extended knit functions for objects exported from other packages
            "huxtable", #  Regression tables, broom compatible
            "mice",  # multiple imputation
            "VIM", # visualizing missing data
            "stargazer", # Regression tables
            "AER", #  Functions, data sets, examples, demos, and vignettes for the book Christian Kleib
            "MASS",  #  Functions and datasets to support Venables and Ripley,  "Modern Applied Statist
            "mvtnorm", # Multivariate Normal and t Distributions
            "summarytools", # Report regression summary tables
            "scales", # scale helper functions such as percent
            "OutliersO3", # Outlier comparison method
            "robustbase", # Basic robust statistics
            "quantreg", # Quantile regression
            "modelr", # model simulation/ boostraping the modern way
            "magrittr") #  pipes
inst<-match(PACKAGES, .packages(all=TRUE))
need<-which(is.na(inst))
if (length(need)>0) install.packages(PACKAGES[need])
lapply(PACKAGES, require, character.only=T)
```

1.1 Simple Linear Regression

We start off with a simple OLS Regression. We will work with multiple data sources:

Table 1.1: A table of the first eight columns and ten rows of the ceosal1 data.

salary	pcsalary	sales	roe	pcroe	ros	indus	finance
1095	20	27595.0	14.1	106.4	191	1	0
1001	32	9958.0	10.9	-30.6	13	1	0
1122	9	6125.9	23.5	-16.3	14	1	0
578	-9	16246.0	5.9	-25.7	-21	1	0
1368	7	21783.2	13.8	-3.0	56	1	0
1145	5	6021.4	20.0	1.0	55	1	0
1078	10	2266.7	16.4	-5.9	62	1	0
1094	7	2966.8	16.3	-1.6	44	1	0
1237	16	4570.2	10.5	-70.2	37	1	0
833	5	2830.0	26.3	-23.9	37	1	0

- Data from **?** : Introductory Econometrics: A Modern Approach.
- R data sets for "Principles of Econometrics" by **?**
- Build in examples such as the `airquality` dataset

Classic examples of quantities modeled with simple linear regression:

- College GPA SAT scores $\beta > 0$
- Change in GDP change in unemployment $\beta < 0$
- House price number of bedrooms $\beta > 0$
- Species heart weight species body weight $\beta > 0$
- Fatalities per year speed limit $\beta < 0$

Notice that these simple linear regressions are simplifications of more complex relationships between the variables in question.

In this exercise we use the dataset *ceosal1*. Let us analyse the dataset first

```
data("ceosal1")
help("ceosal1")
?ceosal1
```

As we see from the R documentation the *ceosal1* dataset contain of a random sample of data reported in the May 6, 1991 issue of Businessweek.

To get a first look at the data you can use the `View()` function inside R Studio.

```
View(ceosal1) # For the compilation we omit the full View()
```

We could also take a look at the variable names, the dimension of the data frame, and some sample observations with `str()`.

```
str(ceosal1)
```

```
## 'data.frame':    209 obs. of  12 variables:
##  $ salary  : int  1095 1001 1122 578 1368 1145 1078 1094 1237 833 ...
##  $ pcsalary: int  20 32 9 -9 7 5 10 7 16 5 ...
##  $ sales   : num  27595 9958 6126 16246 21783 ...
##  $ roe     : num  14.1 10.9 23.5 5.9 13.8 ...
##  $ pcroe   : num  106.4 -30.6 -16.3 -25.7 -3 ...
##  $ ros     : int  191 13 14 -21 56 55 62 44 37 37 ...
##  $ indus   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ finance : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ consprod: int  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ utility : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ lsalary : num  7 6.91 7.02 6.36 7.22 ...
##  $ lsales  : num  10.23 9.21 8.72 9.7 9.99 ...
##  - attr(*, "time.stamp")= chr "25 Jun 2011 23:03"
```

As we have seen before in the general R tutorial, there are a number of additional functions to access some of this information directly.

```
dim(ceosal1)
```

```
## [1] 209  12
```

```
nrow(ceosal1)
```

```
## [1] 209
```

```
ncol(ceosal1)
```

```
## [1] 12
```

```
summary(ceosal1)
```

```
##      salary         pcsalary          sales              roe
##  Min.   :  223   Min.   :-61.00   Min.   :  175.2   Min.   : 0.50
##  1st Qu.:  736   1st Qu.: -1.00   1st Qu.: 2210.3   1st Qu.:12.40
##  Median : 1039   Median :  9.00   Median : 3705.2   Median :15.50
##  Mean   : 1281   Mean   : 13.28   Mean   : 6923.8   Mean   :17.18
##  3rd Qu.: 1407   3rd Qu.: 20.00   3rd Qu.: 7177.0   3rd Qu.:20.00
##  Max.   :14822   Max.   :212.00   Max.   :97649.9   Max.   :56.30
##      pcroe             ros             indus           finance
##  Min.   :-98.9   Min.   :-58.0   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:-21.2   1st Qu.: 21.0   1st Qu.:0.0000   1st Qu.:0.0000
##  Median : -3.0   Median : 52.0   Median :0.0000   Median :0.0000
##  Mean   : 10.8   Mean   : 61.8   Mean   :0.3206   Mean   :0.2201
##  3rd Qu.: 19.5   3rd Qu.: 81.0   3rd Qu.:1.0000   3rd Qu.:0.0000
##  Max.   :977.0   Max.   :418.0   Max.   :1.0000   Max.   :1.0000
##     consprod         utility          lsalary          lsales
##  Min.   :0.0000   Min.   :0.0000   Min.   :5.407   Min.   : 5.166
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:6.601   1st Qu.: 7.701
##  Median :0.0000   Median :0.0000   Median :6.946   Median : 8.217
##  Mean   :0.2871   Mean   :0.1722   Mean   :6.950   Mean   : 8.292
##  3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:7.249   3rd Qu.: 8.879
##  Max.   :1.0000   Max.   :1.0000   Max.   :9.604   Max.   :11.489
```

The interesting task here is to determine how far a high the CEO salary is, for a given return on equity.

Your turn

What sign would be expect of β (the slope)?

A: Without seeing the data **my** prior is that $\beta > 0$.

Note

A simple linear model as assumes that the mean of each y_i conditioned on x_i is a linear function of x_i. But notice that simple linear regressions are simplifications of more complex relationships between the variables in question **?**.

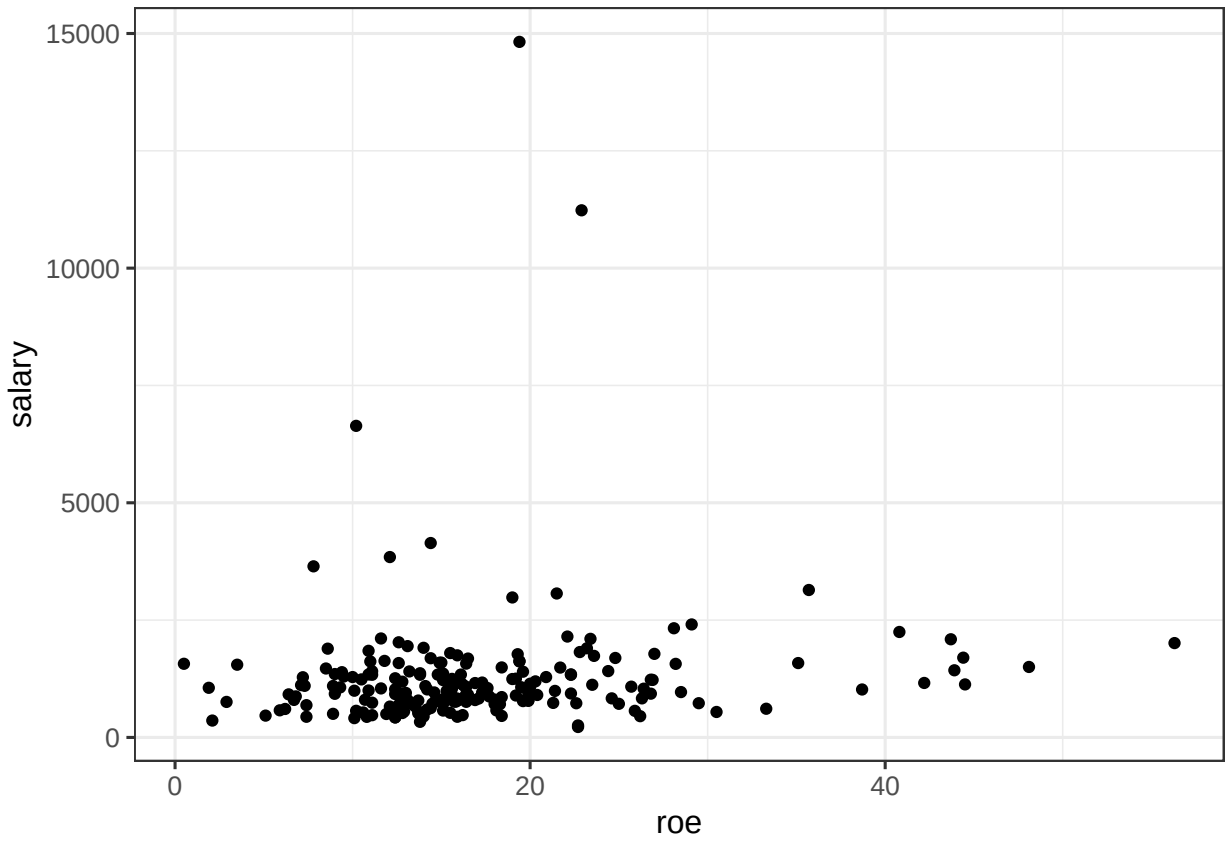


Figure 1.1: Relationship between ROE and Salary

```
# Use ggplot style
ggplot(ceosal1, aes(x = roe, y = salary)) +
  geom_point()
```

Consider a simple regression model

$salary = \beta_0 + \beta_1 roe + u$

In the general form the linear regression model can be written as:

$$y = \beta_0 + \beta_1 x + u \tag{1.1}$$

We are concerned with the population parameter β_0 and β_1. The ordinary least squares (OLS) estimators are:

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \tag{1.2}$$

The ordinary least squares (OLS) estimators are

$$\hat{\beta}_1 = \frac{Cov(x, y)}{Var(x)} \tag{1.3}$$

Ingredients for the OLS formulas

```
attach(ceosal1)
cov(roe, salary)
```

```
## [1] 1342.538
```

```
var(roe)
```

```
## [1] 72.56499
```

```
mean(salary)
```

```
## [1] 1281.12
```

Manual calculation of the OLS coefficients

```
b1hat <- cov(roe,salary)/var(roe)
```

```
b0hat <- mean(salary) - b1hat * mean(roe)
```

Or use the `lm()` function

```
lm(salary ~ roe, data=ceosal1)
```

```
##
## Call:
## lm(formula = salary ~ roe, data = ceosal1)
##
## Coefficients:
## (Intercept)          roe
##       963.2         18.5
```

```
lm1_ceosal1 <- lm(salary ~ roe, data=ceosal1)
summary(lm1_ceosal1)
```

```
##
## Call:
## lm(formula = salary ~ roe, data = ceosal1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1160.2  -526.0  -254.0   138.8 13499.9
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   963.19     213.24   4.517 1.05e-05 ***
## roe            18.50      11.12   1.663   0.0978 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1367 on 207 degrees of freedom
## Multiple R-squared:  0.01319,    Adjusted R-squared:  0.008421
## F-statistic: 2.767 on 1 and 207 DF,  p-value: 0.09777
```

Plot the linear regression fit the *base* r way.

```
plot(salary~ roe, data = ceosal1,
     xlab = "Return on equity",
     ylab = "Salary",
     main = "Salary vs return on equity",
```

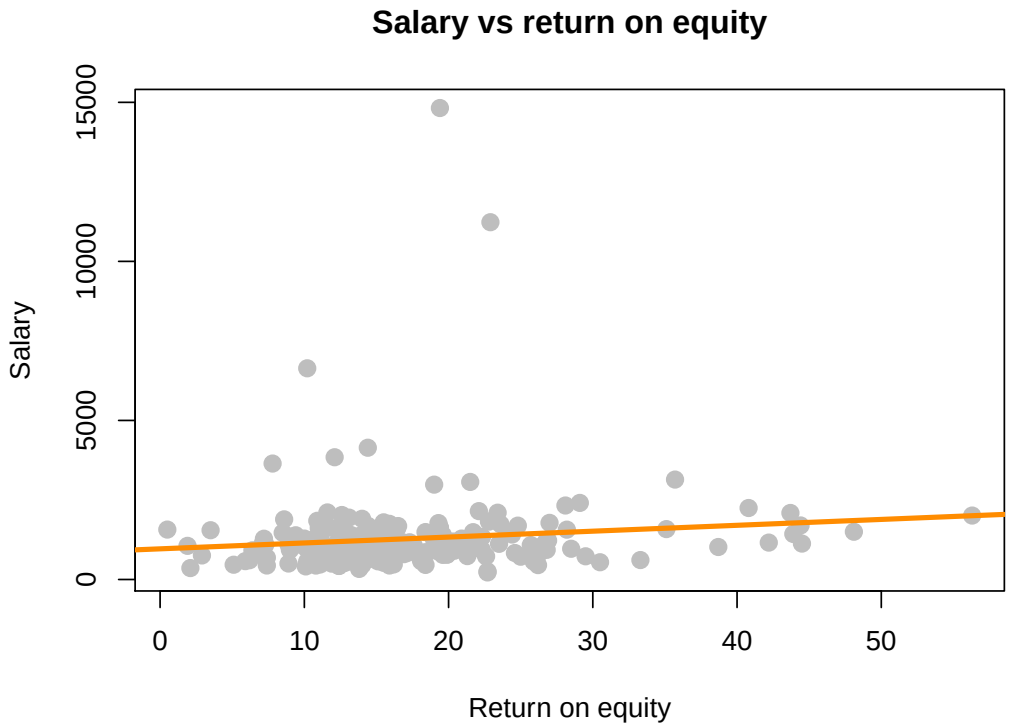


Figure 1.2: OLS regression base Rstyle

```
     pch  = 20,
     cex  = 2,
     col  = "grey")
abline(lm1_ceosal1, lwd = 3, col = "darkorange")
```

Or use ggplot

```
ggplot(ceosal1, aes(x = roe, y = salary)) +
  geom_point() +
  stat_smooth(method = "lm", col = "red")
```

Determine the names of the elements of the list using the `names()` command.

```
names(lm1_ceosal1)
```

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

Extract one element, for example the residuals from the list object

```
head(lm1_ceosal1$residuals) # head() just prints out the first 6 residual values
```

```
##         1         2         3         4         5         6
## -129.0581 -163.8543 -275.9692 -494.3483  149.4923 -188.2151
```

Another way to access stored information in *lm1_ceosal1* are the `coef()`, `resid()`, and `fitted()` functions. These return the coefficients, residuals, and fitted values, respectively.

```
coef(lm1_ceosal1)
```

```
## (Intercept)         roe
```

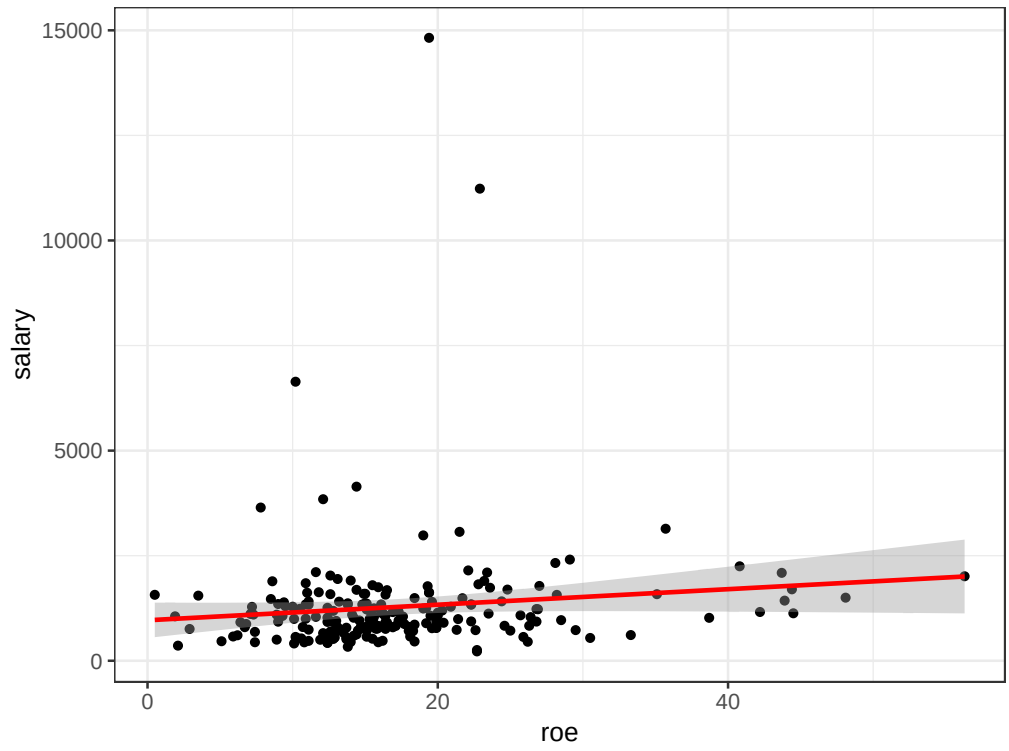


Figure 1.3: OLS regression ggplot2 style

```
##   963.19134    18.50119
```

The function `summary()` is useful in many situations. We see that when it is called on our model, it returns a good deal of information.

```
summary(lm1_ceosal1)
```

```
##
## Call:
## lm(formula = salary ~ roe, data = ceosal1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1160.2  -526.0  -254.0   138.8 13499.9
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   963.19     213.24   4.517 1.05e-05 ***
## roe            18.50      11.12   1.663   0.0978 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1367 on 207 degrees of freedom
## Multiple R-squared:  0.01319,    Adjusted R-squared:  0.008421
## F-statistic: 2.767 on 1 and 207 DF,  p-value: 0.09777
```

The `summary()` command also returns a list, and we can again use `names()` to learn what about the elements of this list.

```
names(summary(lm1_ceosal1))
```

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

So, for example, if we wanted to directly access the value of R^2, instead of copy and pasting it out of the printed statement from `summary()`, we could do so.

```
summary(lm1_ceosal1)$r.squared
```

```
## [1] 0.01318862
```

Your turn

Recall that the explained sum of squares (SSE) is

$$SSE = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2 = (n-1) \times Var(\hat{y}) \tag{1.4}$$

and the residual sum of squares (SSR) is

$$R^2 = \frac{Var(\hat{y})}{Var(y)} = 1 - \frac{Var(\hat{u})}{Var(y)} \tag{1.5}$$

One can see that the correlation between observed and fitted values is a square root of R^2.

Calculate R^2 manually:

```
var(fitted(lm1_ceosal1))/var(ceosal1$salary)
```

```
## [1] 0.01318862
```

```
1 - var(residuals(lm1_ceosal1))/var(ceosal1$salary)
```

```
## [1] 0.01318862
```

Another useful function is the `predict()` function.

```
set.seed(123)
# unique(ceosal1$roe)
roe_sample <-sample(ceosal1$roe, 1)
roe_sample
```

```
## [1] 20.3
```

Let's make a prediction for salary when the return on equity is 20.2999992.

```
b0hat_sample <- mean(salary) - b1hat * roe_sample
```

We are not restricted to observed values of the explanatory variable. Instead we can supply also our own predictor values

```
predict(lm1_ceosal1, newdata = data.frame(roe = 30))
```

```
##        1
## 1518.227
```

The above code reads "predict the salary when the return on equity is 30 using the *lm1_ceosal1* model."

Overthinking

1.1.1 Regression through the Origin and Regression on a Constant

Regression without intercept (through origin)

```
lm2 <- lm(salary ~  0 + roe, data = ceosal1)
```

Regression without slope

```
lm3 <- lm(salary ~ 1, data = ceosal1)
```

```
plot(salary~ roe, data = ceosal1,
     xlab = "Return on equity",
     ylab = "Salary",
     main = "Salary vs return on equity",
     pch  = 20,
     cex  = 2,
     col  = "grey")
abline(lm1_ceosal1, lwd = 3, lty = 1, col = "darkorange")
abline(lm2,lwd = 3,  lty = 2,   col = "darkblue")
abline(lm3, lwd = 3,  lty = 3,   col = "black")
legend("topleft",
       c("full",
         "through origin",
         "constant only"),
       lwd =2,
       lty = 1:3)
```

In models without the intercept the R^2 loses its interpretatation. The reason is that the R^2 is the ratio of explained variance to total variance **only** if the intercept is included.

Overthinking

1.1.2 Simulating SLR

1.1.2.0.1 Expected Values, Variance, and Standard Errors

The **Gauss–Markov theorem** tells us that when estimating the parameters of the simple linear regression model β_0 and β_1, the $\hat{\beta}_0$ and $\hat{\beta}_1$ which we derived are the best linear unbiased estimates, or BLUE for short. (The actual conditions for the Gauss–Markov theorem are more relaxed than the SLR model.)

In short those assumptions are:

- SLR.1 Linear population regression function $y = \beta_0 + \beta_1 \times x + u$
- SLR.2 Random sampling of x and y from the population

- SLR.3 Variation in the sample values: $x_1, \ldots, x_n$
- SLR.4 Zero conditional mean: $\mathbf{E}(u|x) = 0$
- SLR.5 Homeskedasticity: $Var(u|x) = \sigma^2$

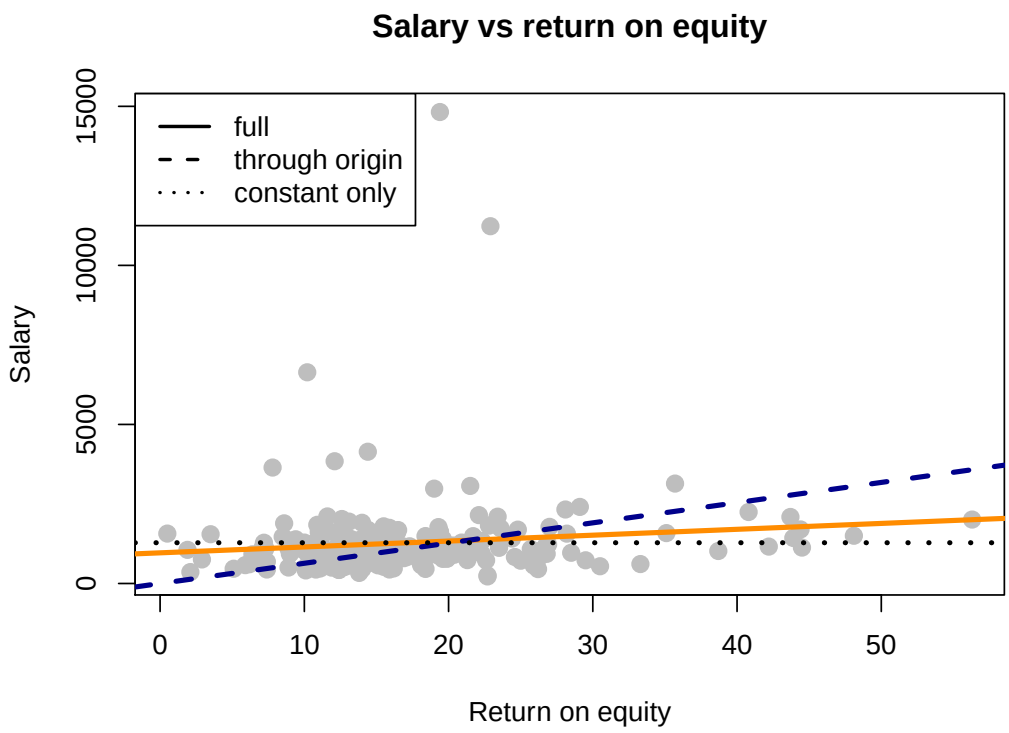


Figure 1.4: Regression through the Origin and on a Constant

Recall that under **SLR.1 - SLR.4** the OLS parameter estimators are unbiased. Under **SLR.1 - SLR.4** the OLS parameter estimators have a specific sampling variance.

Simulating a model is an important concept. In practice you will almost never have a true model, and you will use data to attempt to recover information about the unknown true model. With simulation, we decide the true model and simulate data from it. Then, we apply a method to the data, in this case least squares. Now, since we know the true model, we can assess how well it did.

Simulation also helps to grasp the concepts of estimators, estimates, unbiasedness, the sampling variance of the estimators, and the consequences of violated assumptions.

Sample size

```
n <- 200
```

True parameters

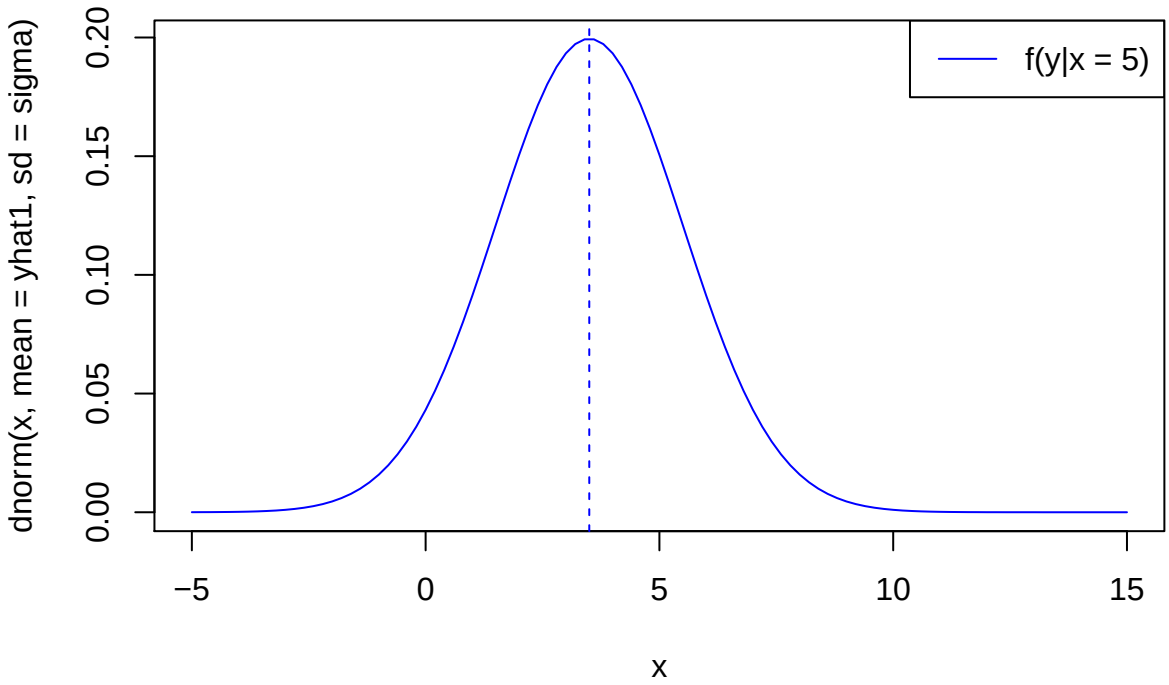
```
b0<- 1
b1 <- 0.5
sigma <- 2 # standard deviation of the error term u
x1 <- 5
```

Determine the distribution of the independent variable

```
yhat1 <- b0 + b1 * x1 #  Note that we do not include the error term
```

Plot a Gaussian distribution of the dependent variable based on the parameters

```
curve(dnorm(x, mean = yhat1, sd = sigma), -5, 15, col = "blue")
abline(v = yhat1, col = "blue", lty = 2)
legend("topright", legend = c("f(y|x = 5)"), lty = 1, col = c("blue"))
```



This represent the theoretical (true) probability distribution of y, given x

We can calculate the variance of b_1 and plot the corresponding density function.

$$var(b_2) = \frac{\sigma^2}{\sum (x_1 - \bar{x})^2} \tag{1.6}$$

Assume that x_2 represents a second possible predictor of y

```
x2 <- 18

x <- c(rep(x1, n/2), rep(x2, n/2))
xbar <- mean(x)

sumxbar <- sum((x-xbar)^2)
varb <- (sigma^2)/sumxbar
sdb <-sqrt(varb)
leftlim <- b1-3*sdb
rightlim <- b1+3*sdb
```

```
curve(dnorm(x, mean = b1, sd = sdb), leftlim, rightlim)
abline(v = b1, col = "blue", lty = 2)
```

Draw sample of size n

```
x <- rnorm(n, 4, sigma)
# Another way is to assume that the values for x are fixed and know
# x= seq(from = 0, to = 10, length.out = n)
```

```
u <- rnorm(n, 0, sigma)
```

```
y <- b0 + b1 * x + u
```

Estimate parameter by OLS

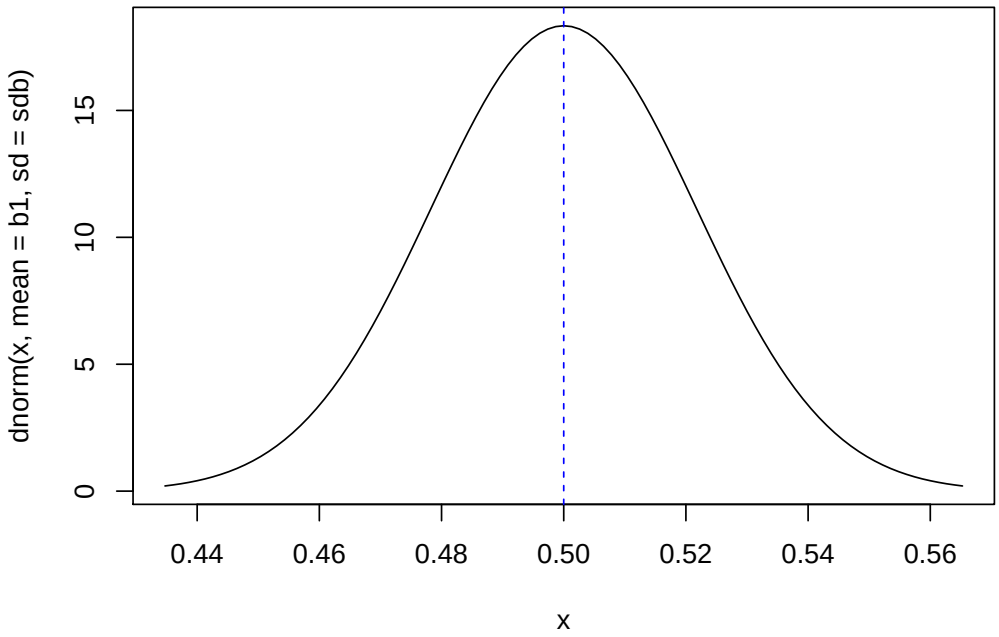


Figure 1.5: The theoretical (true) probability density function of b1

```
olsreg <- lm(y ~x )
```

```
simulation.df <- data.frame(x,y)
population.df <- data.frame(b0, b1)
```

```
plot(simulation.df,
     xlab = "x",
     ylab = "y",
     # main = "Simulate least squares regression",
     pch  = 20,
     cex  = 2,
     col  = "grey")
abline(olsreg, lwd = 3, lty = 1, col = "darkorange")
abline(b0, b1,  lwd = 3,  lty = 2,   col = "darkblue")
legend("topleft",
       c("OLS regression function",
         "Population regression function"),
       lwd =2,
       lty = 1:2)
```

```
lable1 <- "OLS regression function"
ggplot(simulation.df, aes(x = x,  y = y)) +
  geom_point() +
  geom_abline(aes(intercept=b0,slope=b1,colour="Population regression function"), linetype ="dashed", sl
  stat_smooth(aes(colour ="OLS regression function"), method = "lm",se=FALSE, show.legend =TRUE)+
  labs(colour = "Regression functions"
       # , title = "Simulate least squares regression"
  )
```

Since the expected values and variances of our estimators are defined over separate random samples from the same population, it makes sense to repeat our simulation exercise over many simulated samples.

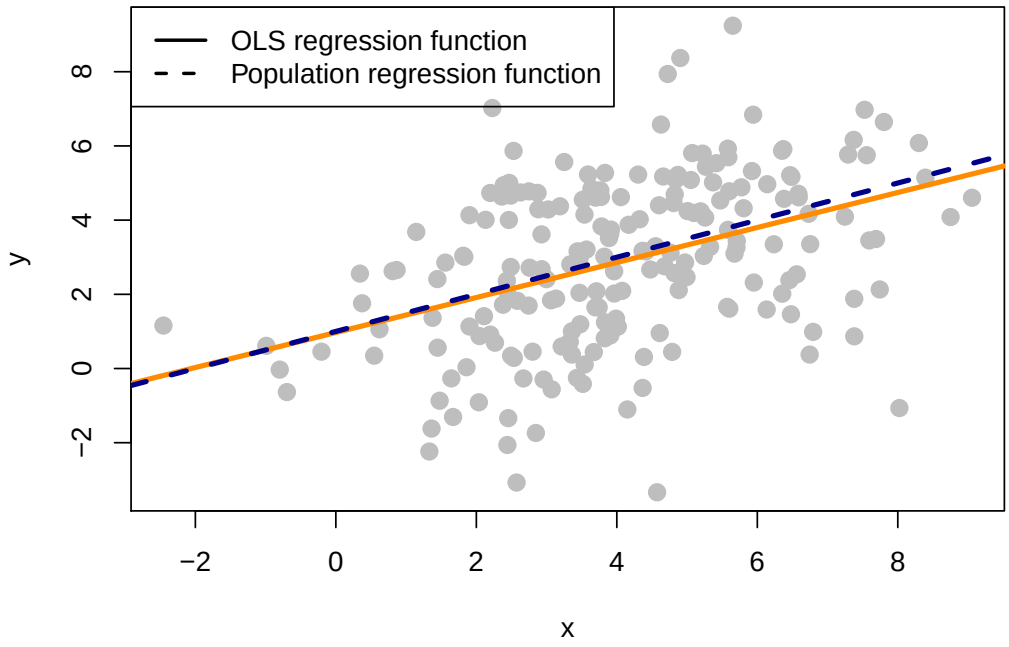


Figure 1.6: Simulated Sample and OLS Regression Line

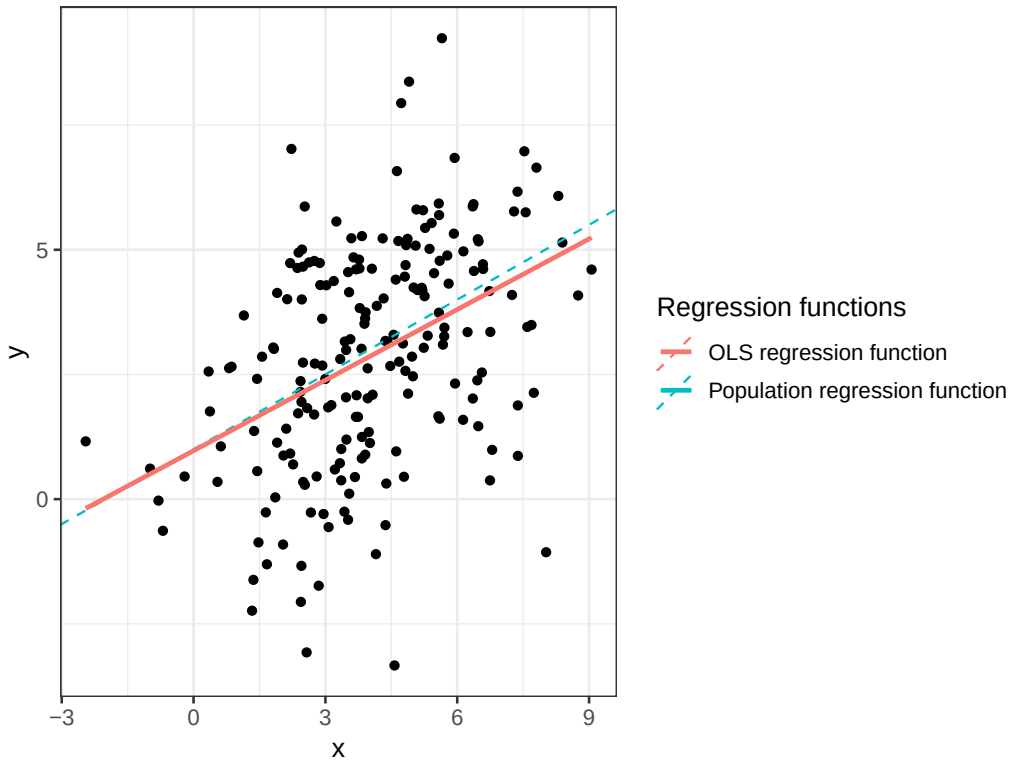


Figure 1.7: Simulated Sample and OLS Regression Line (gpplot Style)

```r
# Set the random seed
set.seed(1234567)

# set sample size and number of simulations
n<-1000; r<-10000

# set true parameters: betas and sd of u
b0<-1.0; b1<-0.5; sigma<-2

# initialize b0hat and b1hat to store results later:
b0hat <- numeric(r)
b1hat <- numeric(r)

# Draw a sample of x, fixed over replications:
x <- rnorm(n,4,1)

# repeat r times:
for(j in 1:r) {
  # Draw a sample of y:
  u <- rnorm(n,0,sigma)
  y <- b0 + b1*x + u

  # estimate parameters by OLS and store them in the vectors
  bhat <- coefficients( lm(y~x) )
  b0hat[j] <- bhat["(Intercept)"]
  b1hat[j] <- bhat["x"]
}
```

```r
# MC estimate of the expected values:
mean(b0hat)
```

```
## [1] 0.9985388
```

```r
mean(b1hat)
```

```
## [1] 0.5000466
```

```r
# MC estimate of the variances:
var(b0hat)
```

```
## [1] 0.0690833
```

```r
var(b1hat)
```

```
## [1] 0.004069063
```

```r
# Initialize empty plot
plot( NULL, xlim=c(0,8), ylim=c(0,6), xlab="x", ylab="y")
# add OLS regression lines
for (j in 1:10) abline(b0hat[j],b1hat[j],col="gray")
# add population regression line
abline(b0,b1,lwd=2)
# add legend
legend("topleft",c("Population","OLS regressions"),
       lwd=c(2,1),col=c("black","gray"))
```

Even though the loop solution is transparent, let us take a look at a different, more *modern* approach.

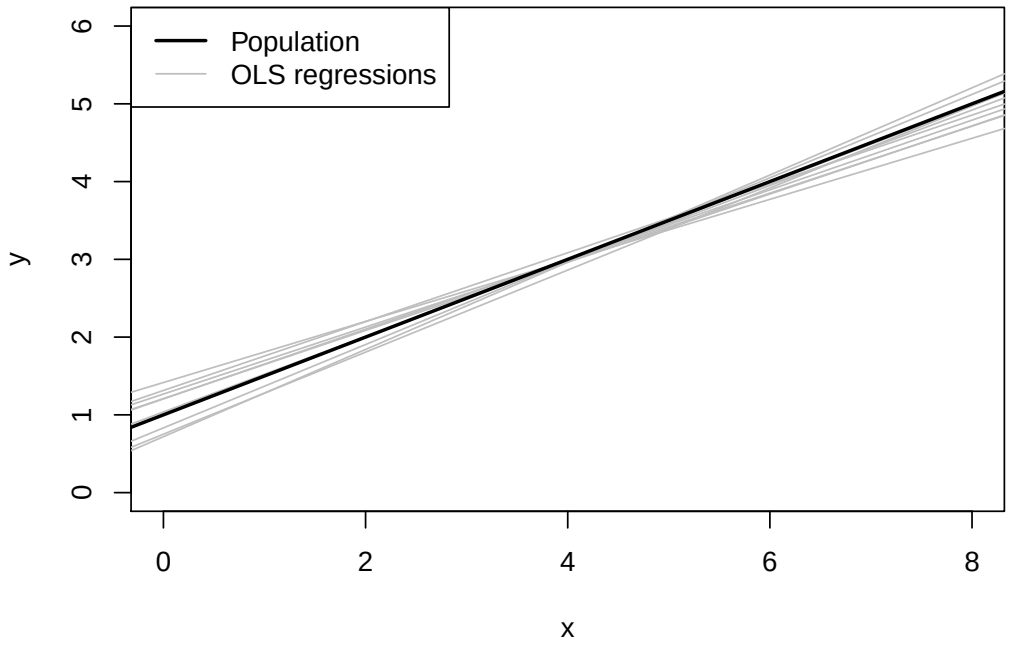


Figure 1.8: Population and Simulated OLS Regression Lines

```
# define a function the returns the alpha -- its point estimate, standard error, etc. -- from the OLS
x <- rnorm(n,4,1) # NOTE 1: Although a normal distribution is usually defined by its mean and variance,
# NOTE 2: We use the same values for x in all samples since we draw them outside of the loop.

iteration <- function() {
  u <- rnorm(n,0,sigma)
  y <- b0 + b1*x + u

  lm(y~x) %>%
    broom::tidy() # %>%
  # filter(term == 'x') # One could only extract the slope
}

# 1000 iterations of the above simulation
MC_coef<- map_df(1:1000, ~iteration())
str(MC_coef)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    2000 obs. of  5 variables:
##  $ term     : chr  "(Intercept)" "x" "(Intercept)" "x" ...
##  $ estimate : num  1.577 0.372 1.44 0.387 1.355 ...
##  $ std.error: num  0.2672 0.0639 0.2623 0.0628 0.2626 ...
##  $ statistic: num  5.9 5.82 5.49 6.17 5.16 ...
##  $ p.value  : num  4.94e-09 7.91e-09 5.13e-08 9.92e-10 2.99e-07 ...
```

Instead of plotting simulated and true parameter regression lines we can take a look at the kernel density of the simulated parameter estimates

Figure 1.9 shows the simulated distribution of β_0 and β_1 the theoretical one.

```
# plot the results
str(MC_coef)
```

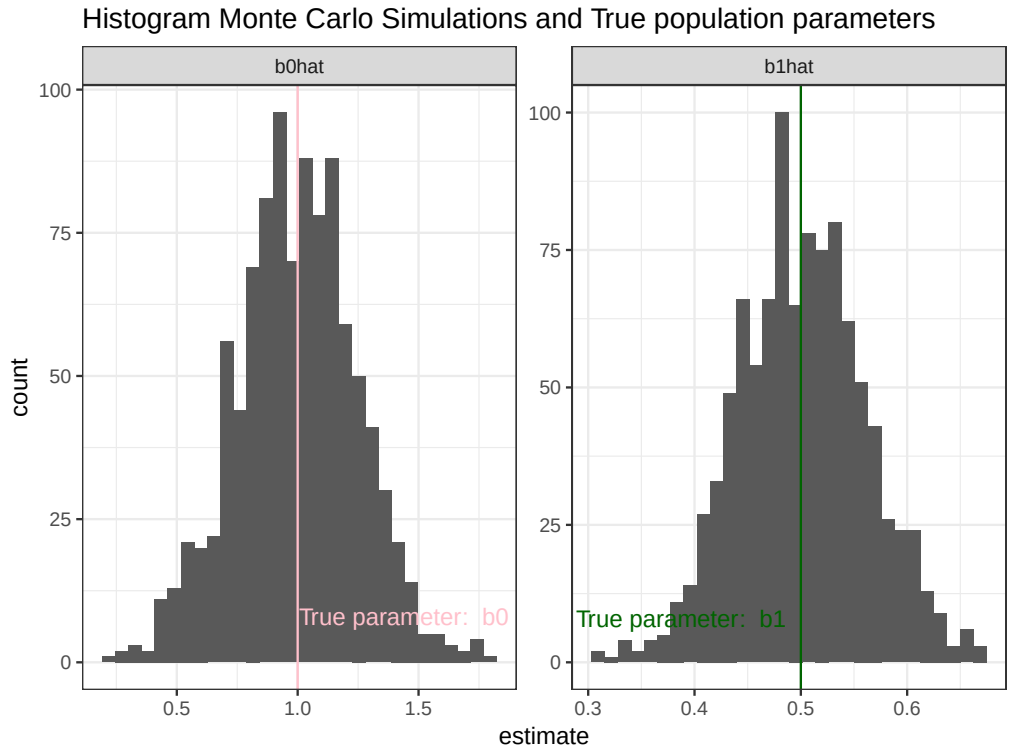


Figure 1.9: Histogram b0 and b1 and true parameter

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    2000 obs. of  5 variables:
##  $ term     : chr  "(Intercept)" "x" "(Intercept)" "x" ...
##  $ estimate : num  1.577 0.372 1.44 0.387 1.355 ...
##  $ std.error: num  0.2672 0.0639 0.2623 0.0628 0.2626 ...
##  $ statistic: num  5.9 5.82 5.49 6.17 5.16 ...
##  $ p.value  : num  4.94e-09 7.91e-09 5.13e-08 9.92e-10 2.99e-07 ...
```

```
MC_coef<- MC_coef %>%
  mutate(OLScoeff =  ifelse(term == "x", "b1hat", "b0hat")) %>%  # rename the x to b1hat and (Intercept
  mutate(Simulated = ifelse(term == "x", "b1", "b0")) #  %>%
```

```
ggplot(data= MC_coef, aes(estimate)) +
  geom_histogram() +
  geom_vline(data = filter(MC_coef, OLScoeff == "b0hat"), aes(xintercept=b0), colour="pink") +
  geom_vline(data = filter(MC_coef, OLScoeff == "b1hat"), aes(xintercept=b1), colour="darkgreen") +
  geom_text(data=MC_coef[3,], mapping=aes(x=estimate, y=8, label=paste("True parameter: ", MC_coef[3,7]
  geom_text(data=MC_coef[4,], mapping=aes(x=estimate, y=8, label=paste("True parameter: ", MC_coef[4,7]
  facet_wrap( ~ OLScoeff, scales = "free")   +
  labs(
    title = "Histogram Monte Carlo Simulations and True population parameters") +
  theme_bw()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
b1_sim <- MC_coef %>%
  filter(Simulated == "b1")

mean(b1_sim$estimate)
```

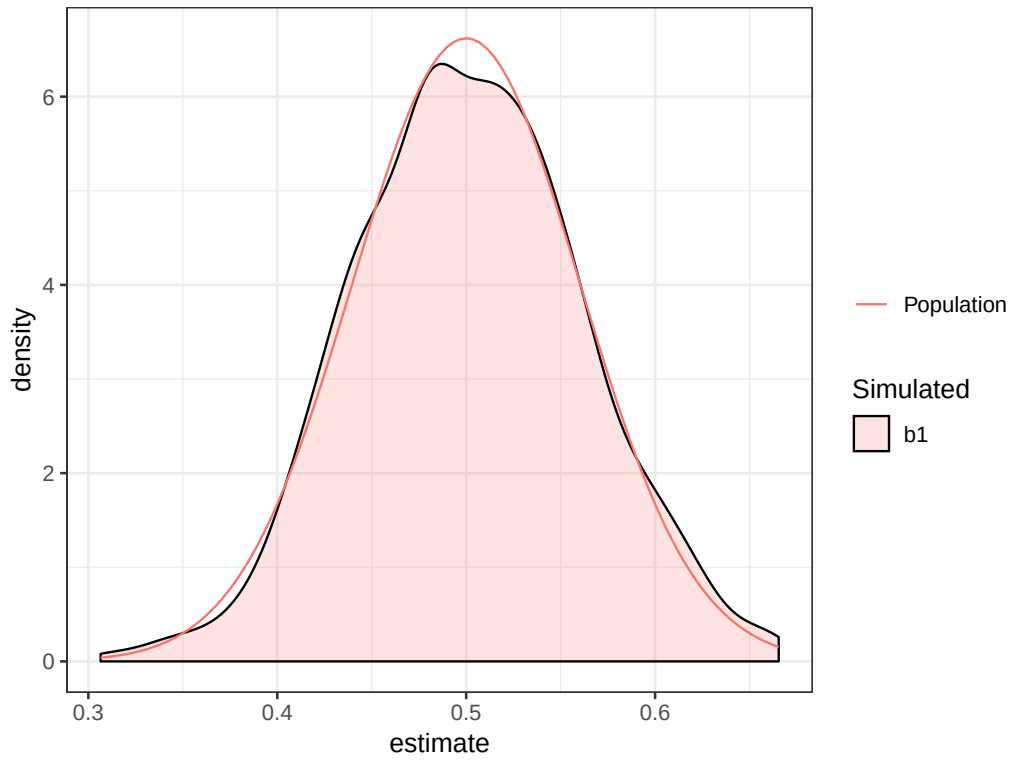


Figure 1.10: Kernel Density Monte Carlo Simulations vs. True population parameters of b1

```
## [1] 0.5011414
```

```
var(b1_sim$estimate) == (sd(b1_sim$estimate))^2
```

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

```
all.equal(var(b1_sim$estimate) , (sd(b1_sim$estimate))^2) # Floating point arithmetic!
```

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

```
ggplot(data= b1_sim, aes(estimate)) +
  geom_density(aes(fill = Simulated), alpha = 0.2) + # computes and draws the kernel density, which is
  # stat_function(fun = dnorm, args = list(mean = mean(b1_sim$estimate), sd = sd(b1_sim$estimate)), aes
  stat_function(fun = dnorm, args = list(mean = 0.5, sd = sd(b1_sim$estimate)), aes(colour = "Populatio
  # labs(
  #  title = "Kernel Density Monte Carlo Simulations vs. True population parameters"
  # ) +
  scale_color_discrete(name="")
```

1.1.2.0.2 Violation of SLR.4

To implement a violation of **SLR.4** (zero conditional mean) consider a case where in the population u is not mean independent of x, for example

$$\mathbf{E}(u|x) = \frac{x-4}{5}$$

```
# Set the random seed
set.seed(1234567)
```

```
# set sample size and number of simulations
n<-1000; r<-10000

# set true parameters: betas and sd of u
b0<-1; b1<-0.5; su<-2

# initialize b0hat and b1hat to store results later:
b0hat <- numeric(r)
b1hat <- numeric(r)

# Draw a sample of x, fixed over replications:
x <- rnorm(n,4,1)

# repeat r times:
for(j in 1:r) {
  # Draw a sample of y:
  u <- rnorm(n, (x-4)/5, su) # this is where manipulate the assumption of zero conditional mean
  y <- b0 + b1*x + u

  # estimate parameters by OLS and store them in the vectors
  bhat <- coefficients( lm(y~x) )
  b0hat[j] <- bhat["(Intercept)"]
  b1hat[j] <- bhat["x"]
}
```

OLS coefficients

```
# MC estimate of the expected values:
mean(b0hat)
```

```
## [1] 0.1985388
```

```
mean(b1hat)
```

```
## [1] 0.7000466
```

```
# MC estimate of the variances:
var(b0hat)
```

```
## [1] 0.0690833
```

```
var(b1hat)
```

```
## [1] 0.004069063
```

The average estimates are far from the population parameters $\beta_0 = 1$ and $\beta_1 = 0.5$!

1.1.2.0.3 Violation of SLR.5

Homoskedasticity is not required for unbiasedness but for it is a requirement for the theorem of sampling variance. Consider the following heteroskedastic behavior of u given x.

```
# Set the random seed
set.seed(1234567)

# set sample size and number of simulations
n<-1000; r<-10000
```

```
# set true parameters: betas and sd of u
b0<-1; b1<-0.5; su<-2

# initialize b0hat and b1hat to store results later:
b0hat <- numeric(r)
b1hat <- numeric(r)

# Draw a sample of x, fixed over replications:
x <- rnorm(n,4,1)

# repeat r times:
for(j in 1:r) {
  # Draw a sample of y:
  varu <- 4/exp(4.5) * exp(x)
  u <- rnorm(n, 0, sqrt(varu) )
  y <- b0 + b1*x + u

  # estimate parameters by OLS and store them in the vectors
  lm_heterosced <- lm(y~x)

  bhat <- coefficients( lm(y~x) )
  b0hat[j] <- bhat["(Intercept)"]
  b1hat[j] <- bhat["x"]
}
```

```
summary(lm_heterosced) # just the last sample of the MC-simulation
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -23.6742  -0.9033   0.0052   1.0012   9.3411
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.24088    0.27158   4.569 5.51e-06 ***
## x            0.44561    0.06593   6.759 2.37e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.075 on 998 degrees of freedom
## Multiple R-squared:  0.04377,    Adjusted R-squared:  0.04281
## F-statistic: 45.68 on 1 and 998 DF,  p-value: 2.367e-11
```

Plot the residual against the regressor suspected of creating heteroskedasticity, or more generally, the fitted values of the regression.

```
res <- residuals(lm_heterosced)
yhat <- fitted(lm_heterosced)
```

```
par(mfrow = c(1,2))
plot(x, res, ylab = "residuals")
plot(yhat, res, xlab = "fitted values", ylab = "residuals")
```

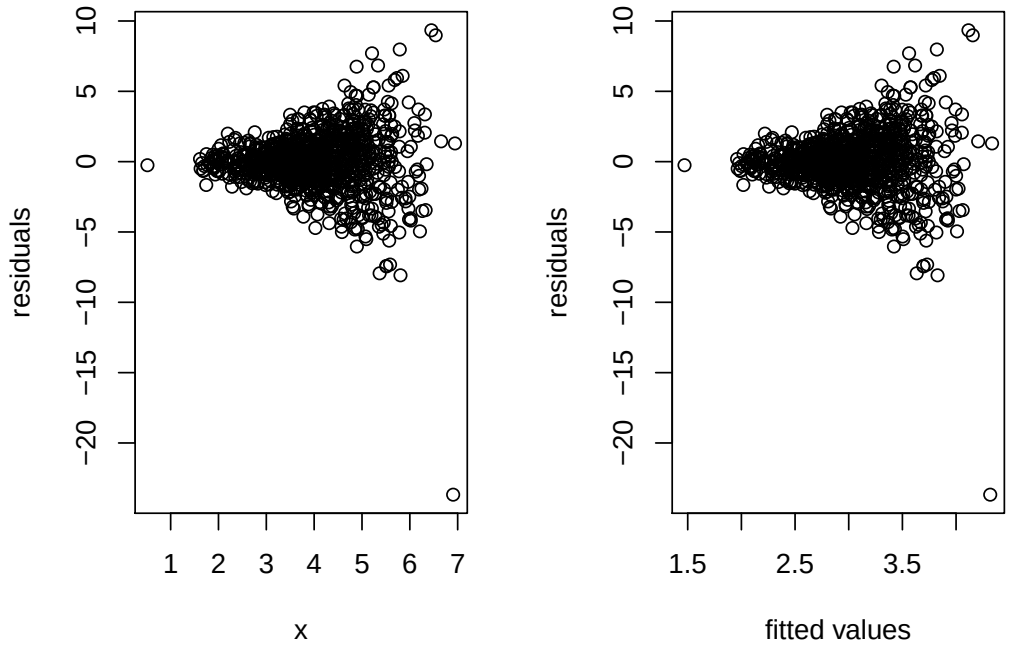


Figure 1.11: Heteroskedasticity in the simulated data

```
# MC estimate of the expected values:
mean(b0hat)
```

```
## [1] 1.0019
```

```
mean(b1hat)
```

```
## [1] 0.4992376
```

```
# MC estimate of the variances:
var(b0hat)
```

```
## [1] 0.08967037
```

```
var(b1hat)
```

```
## [1] 0.007264373
```

Unbiasedness is provided but sampling variance is incorrect (compared to the results provided above).

1.1.3 Nonlinearities

Sometimes the scatter plot diagram or some theoretical considerations suggest a non-linear relationship. The most popular non-linear transformation involve logarithms of the dependent or independent variables and polynomial functions.

We will use a new dataset, *wage1*, for this section. A detailed exploratory analysis of the dataset is left to the reader.

```
data("wage1")
```

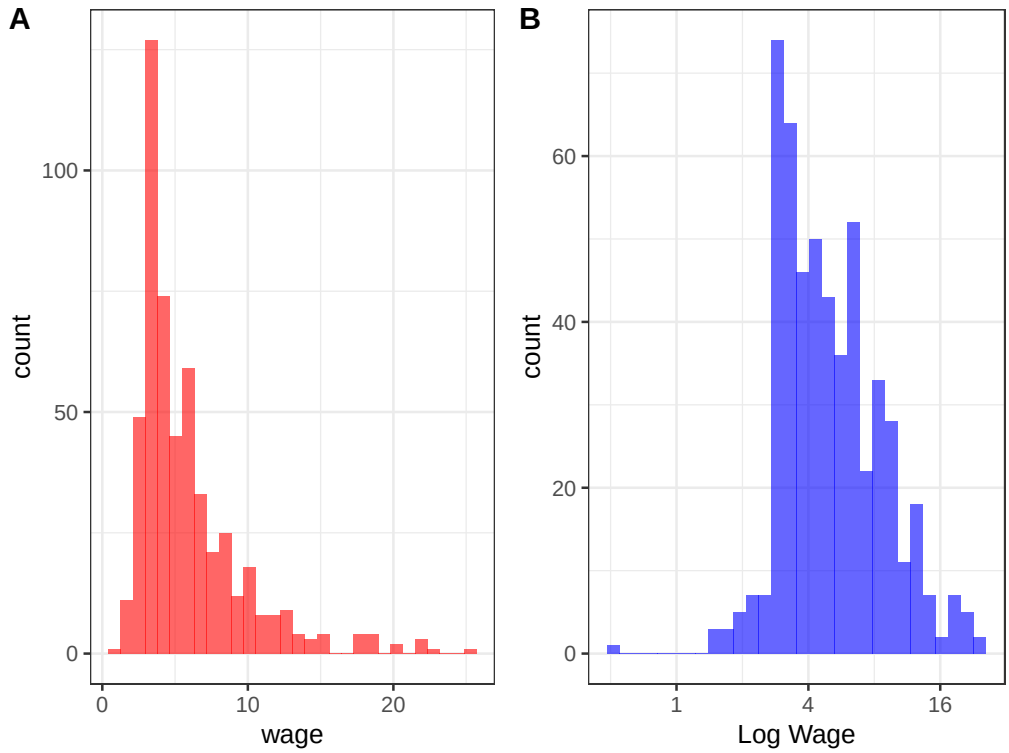


Figure 1.12: Histogram of wage and log(wage)

1.1.3.1 Predictor variable transformation

A common variance stabilizing transformation (VST) is necessary when we see increasing variance in a fitted versus residuals plot.

To use the *log* of an independent variable is to make its distribution closer to the normal distribution.

```
# wage1$logwage <- log(wage1$wage) # one could also create a new variable

p1_wagehisto <- ggplot(wage1)  +
  geom_histogram(aes(x = wage), fill = "red", alpha = 0.6)

p2_wagehisto <- ggplot(wage1)  +
  geom_histogram(aes(x = wage),  fill = "blue", alpha = 0.6) +
  scale_x_continuous(trans='log2', "Log Wage")  # instead of creating a new variable with simply define
```

```
ggarrange(p1_wagehisto, p2_wagehisto,
          labels = c("A", "B"),
          ncol = 2, nrow = 1)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

A model with a log transformed response:

$$log(Y_i) = \beta_0 + \beta_1 \times x_i + \epsilon_i \tag{1.7}$$

```
lm_wage <- lm(wage ~ educ, data = wage1)
lm_wage1 <- lm(log(wage) ~ educ, data =  wage1)
summary(lm_wage)
```

```
##
## Call:
## lm(formula = wage ~ educ, data = wage1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.3396 -2.1501 -0.9674  1.1921 16.6085
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.90485    0.68497  -1.321    0.187
## educ         0.54136    0.05325  10.167   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.378 on 524 degrees of freedom
## Multiple R-squared:  0.1648, Adjusted R-squared:  0.1632
## F-statistic: 103.4 on 1 and 524 DF,  p-value: < 2.2e-16
```

```
summary(lm_wage1)
```

```
##
## Call:
## lm(formula = log(wage) ~ educ, data = wage1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.21158 -0.36393 -0.07263  0.29712  1.52339
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.583773   0.097336   5.998 3.74e-09 ***
## educ        0.082744   0.007567  10.935  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4801 on 524 degrees of freedom
## Multiple R-squared:  0.1858, Adjusted R-squared:  0.1843
## F-statistic: 119.6 on 1 and 524 DF,  p-value: < 2.2e-16
```

Plotting Diagnostics for Linear Models

```
plot(lm_wage)
```

```
autoplot(lm_wage, which = 1:6, colour = 'dodgerblue3',
         smooth.colour = 'red', smooth.linetype = 'dashed',
         ad.colour = 'blue',
         label = FALSE,
         label.size = 3, label.n = 5, label.colour = 'blue',
         ncol = 3) +
  theme_bw()
```

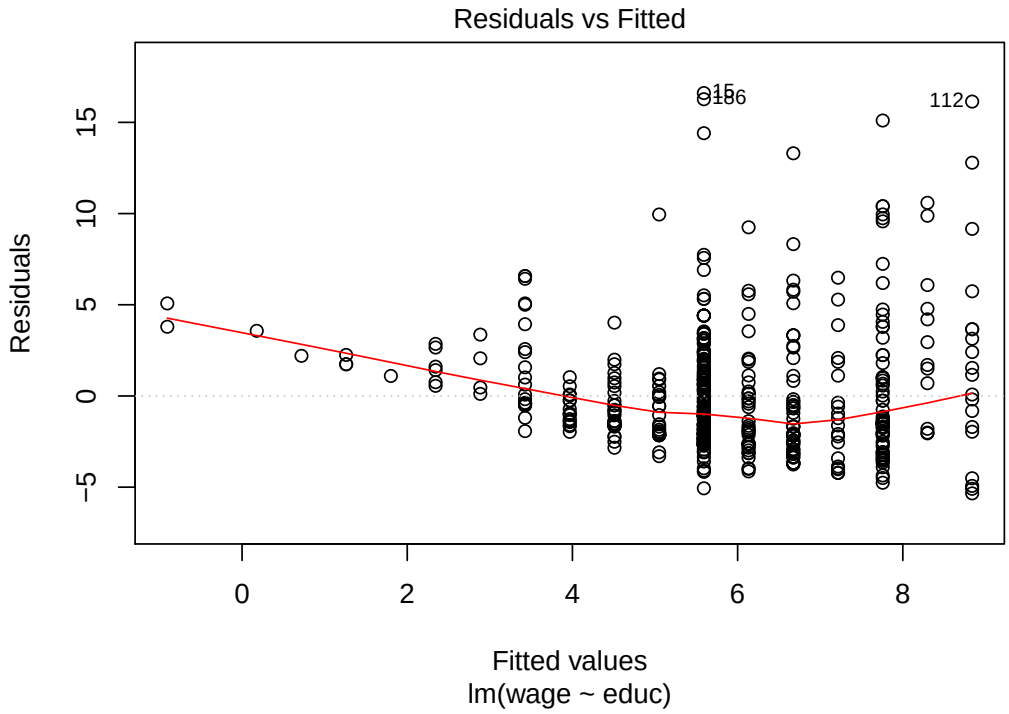


Figure 1.13: Regression diagnostics plot base R - Linear Relationship

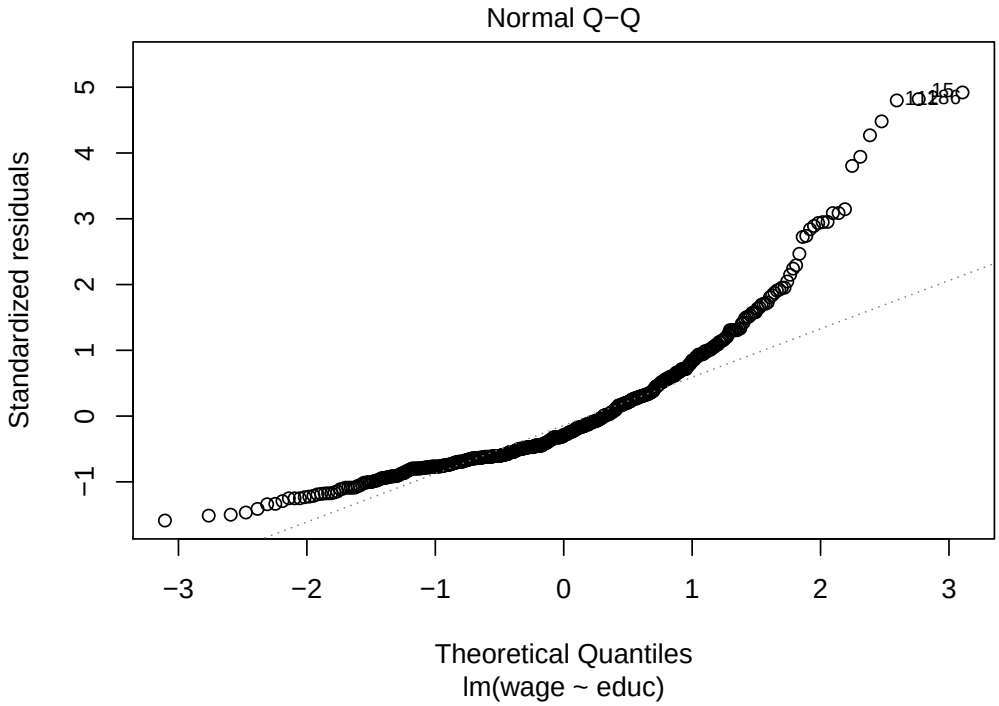


Figure 1.14: Regression diagnostics plot base R - Linear Relationship

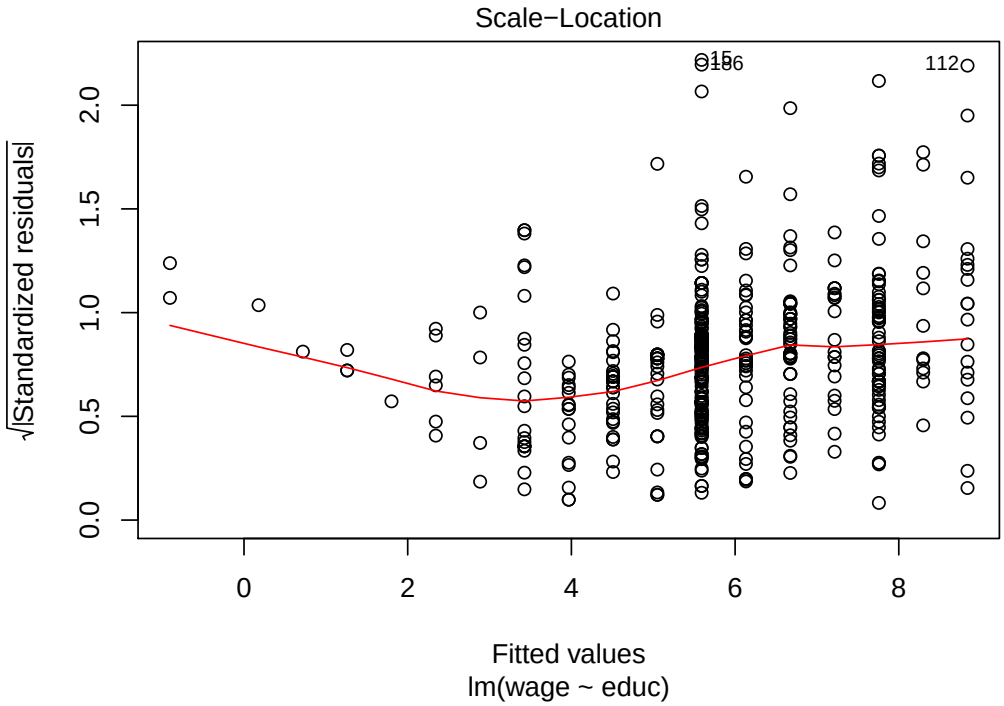


Figure 1.15: Regression diagnostics plot base R - Linear Relationship

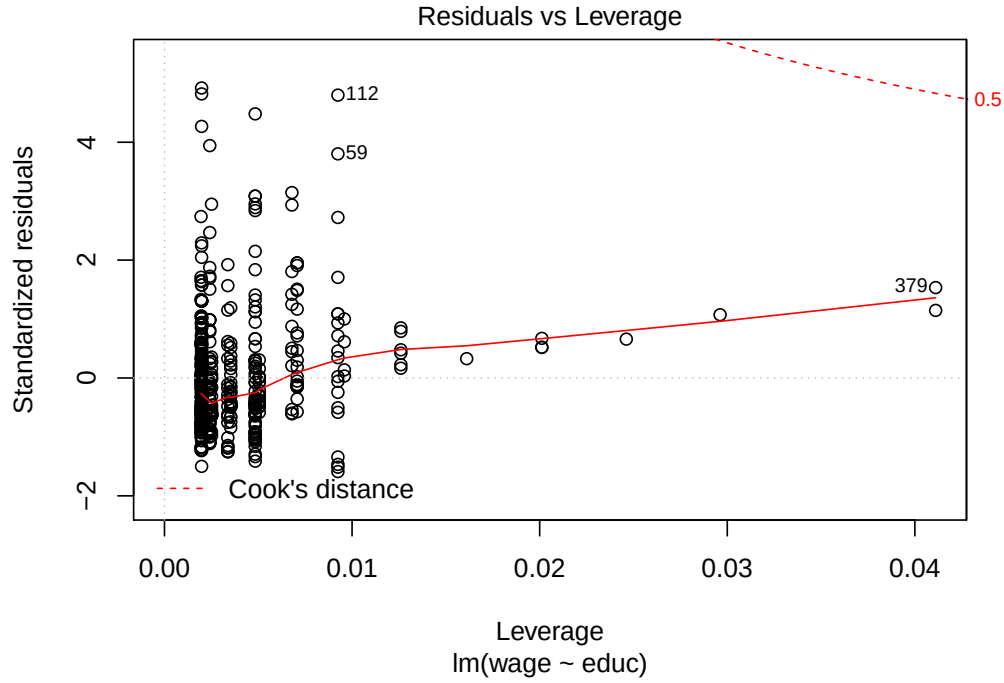


Figure 1.16: Regression diagnostics plot base R - Linear Relationship

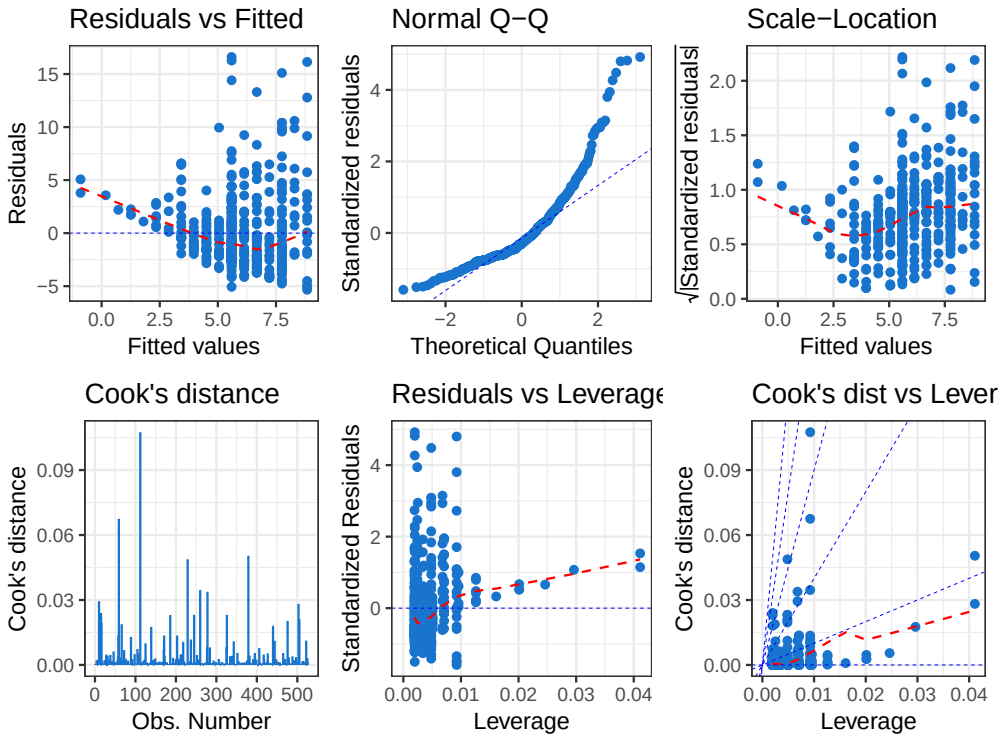


Figure 1.17: Regression diagnostics autoplot(ggplot) - Linear Relationship

```
autoplot(lm_wage1, which = 1:6, colour = 'dodgerblue3',
         smooth.colour = 'red', smooth.linetype = 'dashed',
         ad.colour = 'blue',
         label = FALSE,
         label.size = 3, label.n = 5, label.colour = 'blue',
         ncol = 3) +
  theme_bw()
```

```
p1_nonlinearities <- ggplot(wage1, aes(x = educ, y = wage )) +
  geom_point()   +
  scale_y_continuous(trans='log2', "Log Wage") +
  stat_smooth(aes(fill="Linear Model"),size=1,method = "lm" ,span =0.3, se=F) +
  guides(fill = guide_legend("Model Type")) +
  theme_bw()
p1_nonlinearities
```

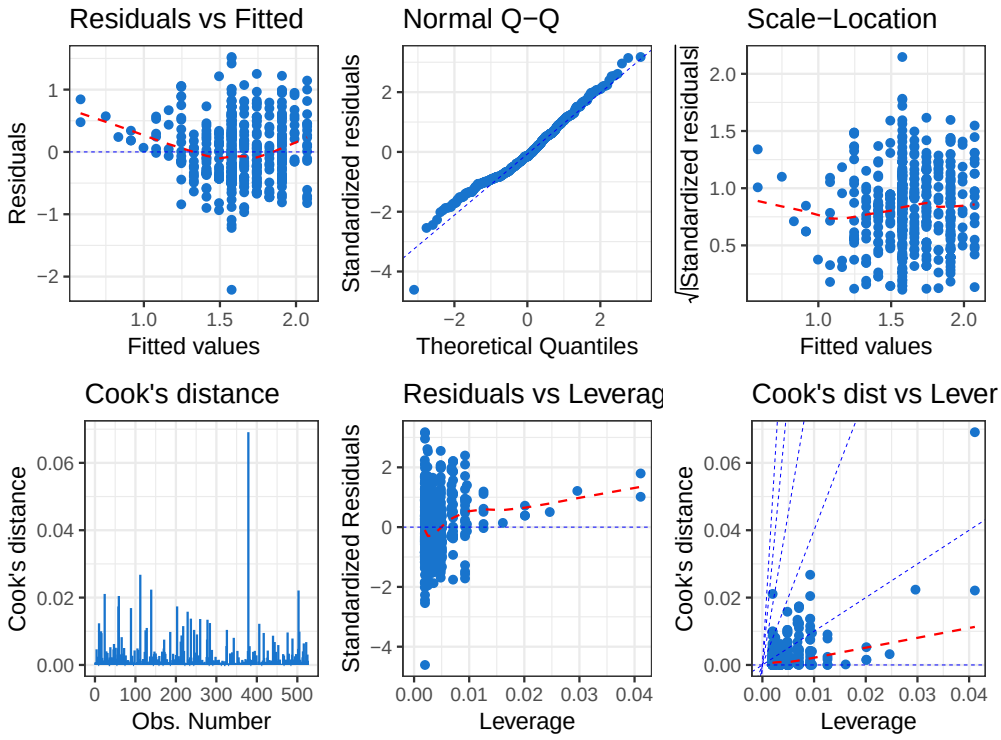
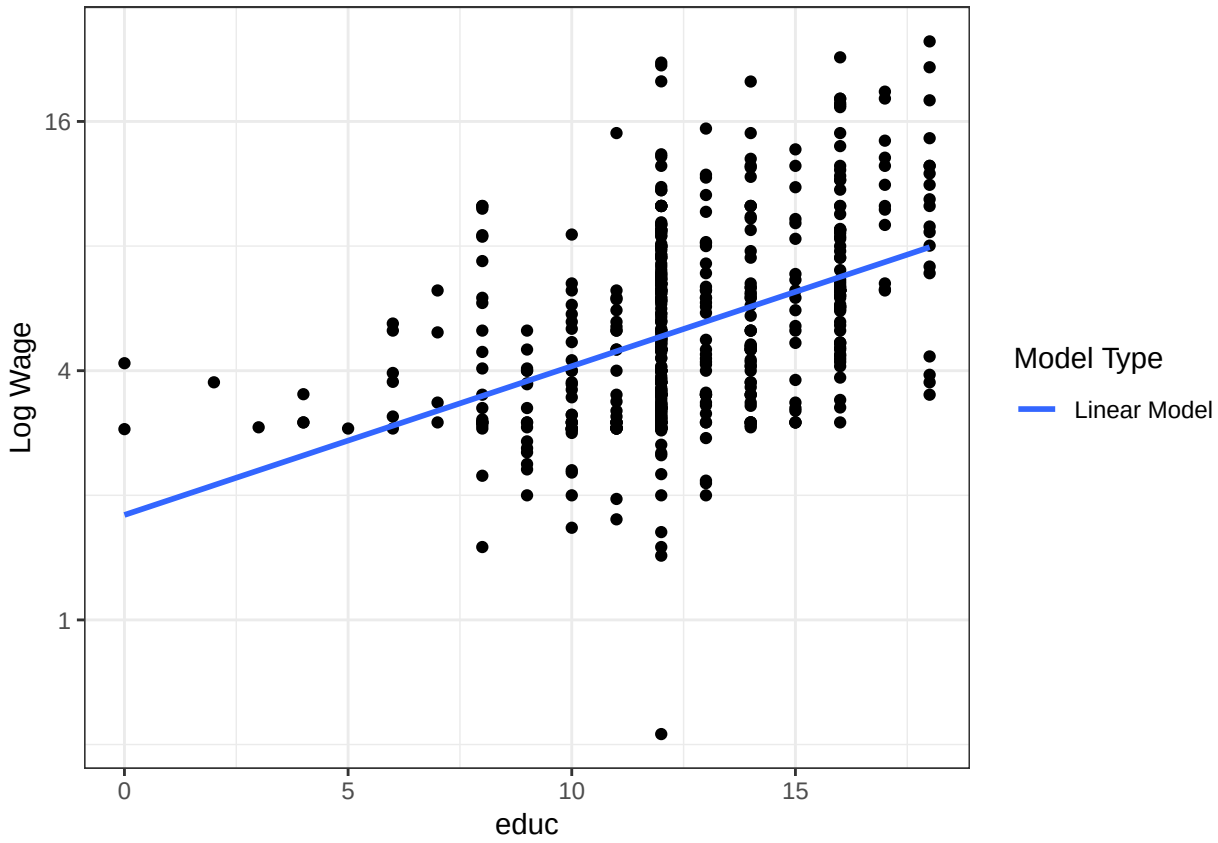


Figure 1.18: Regression diagnostics - Non-Linear Relationship



Note that if we re-scale the model from a log scale back to the original scale of the data, we now have

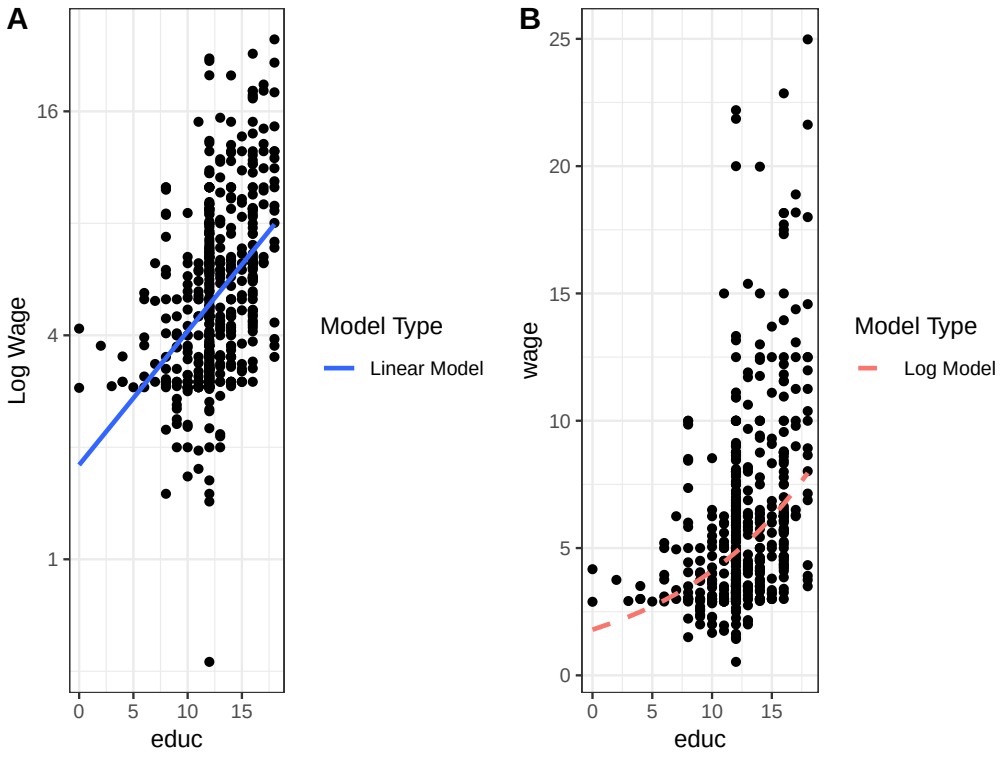


Figure 1.19: Wages by Education - Different transformations

$$Y_i = exp(\beta_0 + \beta_1 \times x_i) \times exp(\epsilon_i) \tag{1.8}$$

which has errors entering in a multiplicative fashion.

```
log.model.df <- data.frame(x = wage1$educ,
                           y = exp(fitted(lm_wage1))) # This is essentially exp(b0_wage1 + b1_wage1 * w
```

```
p2_nonlinearities <- ggplot(wage1, aes(x = educ, y = wage))  +
  geom_point()   +
  geom_line(data = log.model.df, aes(x, y, color = "Log Model"), size = 1, linetype = 2)  +
  guides(color = guide_legend("Model Type")) +
  theme_bw()
```

```
ggarrange(p1_nonlinearities, p2_nonlinearities,
          labels = c("A", "B"),
          ncol = 2, nrow = 1)
```

A: Plotting the data on the transformed log scale and adding the fitted line, the relationship again appears linear, and the variation about the fitted line looks more constant.

B: By plotting the data on the original scale, and adding the fitted regression, we see an exponential relationship. However, this is still a *linear* model, since the new transformed response, $log(Y_i$, is still a *linear* combination of the predictors. In other words, only β needs to be linear, not the x values.

Quadratic Model

$$Y_i = \beta_0 + \beta_1 \times x_i \beta_2 \times x_i^2 + \epsilon_i \tag{1.9}$$

New dataset from Wooldrige: Collected from the real estate pages of the Boston Globe during 1990. These are homes that sold in the Boston, MA area.

```
data("hprice1", package = "wooldridge")
```

In R, independent variables involving mathematical operators can be included in regression equation with the function `I()`

We estimate the following regression:

$$hprice\ \beta_0 + \beta_1 sqrft + u \tag{1.10}$$

```
lm_hprice <- lm(price ~ sqrft, data  = hprice1)
```

and a regression model that includes a squared term

$$hprice\ \beta_0 + \beta_1 sqrft + \beta_2 sqrft^2 + u \tag{1.11}$$

```
lm_hprice1 <- lm(price ~ sqrft + I(sqrft^2), data  = hprice1)
```

Alternatively use the `poly()` function. Be careful of the additional argument `raw`.

```
lm_hprice2 <- lm(price ~ poly(sqrft, degree = 2),  data  = hprice1)
lm_hprice3 <- lm(price ~ poly(sqrft, degree = 2, raw = TRUE),  data  = hprice1) # if true, use raw and
```

```
unname(coef(lm_hprice1))
```

```
## [1]  1.849453e+02 -1.710855e-02  3.262809e-05
```

```
unname(coef(lm_hprice2))
```

```
## [1] 293.5460 754.8517 135.6051
```

```
unname(coef(lm_hprice3))
```

```
## [1]  1.849453e+02 -1.710855e-02  3.262809e-05
```

```
all.equal(unname(coef(lm_hprice1)), unname(coef(lm_hprice2)))
```

```
## [1] "Mean relative difference: 5.401501"
```

```
all.equal(unname(coef(lm_hprice1)), unname(coef(lm_hprice3)))
```

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

```
all.equal(fitted(lm_hprice1), fitted(lm_hprice2))
```

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

```
all.equal(fitted(lm_hprice1), fitted(lm_hprice3))
```

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

With the function `all.equal()` we can test if all elements in two vectors are the "nearly" the same. "Nearly" refers to the case when *tolerance* values are exceeded.

Why are those values not the same depending on the `poly()` function argument `raw = True`? In the case of `raw = True` R uses raw and not orthogonal polynomials.

This can have importan implications in the case when multiple polynimoals are used in the regression. The linear regressar $sqrft$ is uncorrelated linearly with the squared explanatory variable $sqrft^2$. However, if

we add a cubic term $sqrft^3$, multicollinearty between $sqrft^2$ and $sqrft^3$ might become and issues of the polynomials are *NOT* orthogonalized.

We can also extract the standard error of our estimated coefficents manually.

Starting with the variance-covariance matrix of the regression model

```
vcov_lm_hprice2 <- vcov(lm_hprice2)
```

Extracting only the diagonal element of the matrix and taking the square root we obtain the standard errors as reported in `summary()` function call.

```
sqrt(diag(vcov_lm_hprice2))
```

```
##              (Intercept) poly(sqrft, degree = 2)1 poly(sqrft, degree = 2)2
##                 6.638736               62.276866               62.276866
```

```
summary(lm_hprice2)
```

```
##
## Call:
## lm(formula = price ~ poly(sqrft, degree = 2), data = hprice1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -158.261  -35.158   -7.924   24.262  223.516
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 293.546      6.639  44.217   <2e-16 ***
## poly(sqrft, degree = 2)1  754.852     62.277  12.121   <2e-16 ***
## poly(sqrft, degree = 2)2  135.605     62.277   2.177   0.0322 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 62.28 on 85 degrees of freedom
## Multiple R-squared:  0.6408, Adjusted R-squared:  0.6324
## F-statistic: 75.83 on 2 and 85 DF,  p-value: < 2.2e-16
```

1.2 Multiple Linear Regression

Note

A **(general) linear model** is similar to the simple variant, but with a multivariate $x \epsilon R^{\rho}$ and a mean given by a hyperplane in place of a single line.

- General principles are the same as the simple case
- Math is more difficult because we need to use matrices
- Interpretation is more difficult because the β_j are effects conditional on the other variables

Many would retain the same signs as the simple linear regression, but the magnitudes would be smaller. In some cases, it is possible for the relationship to flip directions when a second (highly correlated) variable is added **?**.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + u \tag{1.12}$$

The next example from Wooldrige relates the college GPA ("cloGPA") to the high school GPA ("hsGPA") and achievement test score ("ACT") for a sample of 141 students.

```
data("gpa1", package = "wooldridge")
attach(gpa1)
```

```
## The following object is masked from package:robustbase:
##
##     alcohol
```

```
## The following objects are masked from package:wooldridge:
##
##     alcohol, campus
```

```
?gpa1
```

Obtain parameter estimates

```
GPAres <- lm(colGPA ~ hsGPA + ACT, data = gpa1)
summary(GPAres)
```

```
##
## Call:
## lm(formula = colGPA ~ hsGPA + ACT, data = gpa1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.85442 -0.24666 -0.02614  0.28127  0.85357
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.286328   0.340822   3.774 0.000238 ***
## hsGPA       0.453456   0.095813   4.733 5.42e-06 ***
## ACT         0.009426   0.010777   0.875 0.383297
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3403 on 138 degrees of freedom
## Multiple R-squared:  0.1764, Adjusted R-squared:  0.1645
## F-statistic: 14.78 on 2 and 138 DF,  p-value: 1.526e-06
```

```
coef(GPAres)[[1]]
```

```
## [1] 1.286328
```

In the multiple linear regression setting, some of the interpretations of the coefficients change slightly. Here, $\hat{\beta}_0 = 1.2863278$ is our estimate for β_0 when all of the predictors are 0. In this example this makes sense but think of the following example:

Your turn

Assume the following model:

```
mpg_model = lm(hp ~ wt + cyl, data = mtcars)
coef(mpg_model)
```

```
## (Intercept)          wt         cyl
##  -51.805567    1.330463   31.387901
```

How do you interpret the intercept coefficient estimate?

A: Here, $\hat{\beta}_0$ = -51.8055669 is our estimate for β_0, the mean gross horsepower for a car that weights 0 pounds and has 0 cylinders. We see our estimate here is negative, which is a physical impossibility. However, this isn't unexpected, as we shouldn't expect our model to be accurate for cars which weight 0 pounds and have no cylinders to propel the engine.

```r
with (gpa1, {
  # find min-max seq for grid construction
  min_hsGPA <- min(gpa1$hsGPA)
  max_hsGPA <- max(gpa1$hsGPA)
  min_ACT <- min(gpa1$ACT)
  max_ACT <- max(gpa1$ACT)


  # linear regression
  fit <- lm(colGPA ~ hsGPA + ACT)

  # predict values on regular xy grid
  hsGPA.pred <- seq(min_hsGPA, max_hsGPA, length.out = 30)
  ACT.pred <- seq(min_ACT, max_ACT, length.out = 30)
  xy <- expand.grid(hsGPA = hsGPA.pred,
                    ACT = ACT.pred)

  colGPA.pred <- matrix (nrow = 30, ncol = 30,
                         data = predict(fit, newdata = data.frame(xy),
                                        interval = "prediction"))

  # fitted points for droplines to surface
  fitpoints <- predict(fit)

  scatter3D(z = colGPA, x = hsGPA, y = ACT, pch = 18, cex = 2,
            theta = 20, phi = 20, ticktype = "detailed",
            xlab = "hsGPA", ylab = "ACT", zlab = "colGPA",
            surf = list(x = hsGPA.pred, y = ACT.pred, z = colGPA.pred,
                        facets = NA, fit = fitpoints),
            main = "colGPA")

})
```

The data points (x_{i1},x_{i2},y_i) now exist in 3-dimensional space, so instead of fitting a line to the data, we will fit a plane.

1.2.1 Ceteris Paribus Interpretation and Omitted Variable bias

Consider a regression with two explanatory variables

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 \tag{1.13}$$

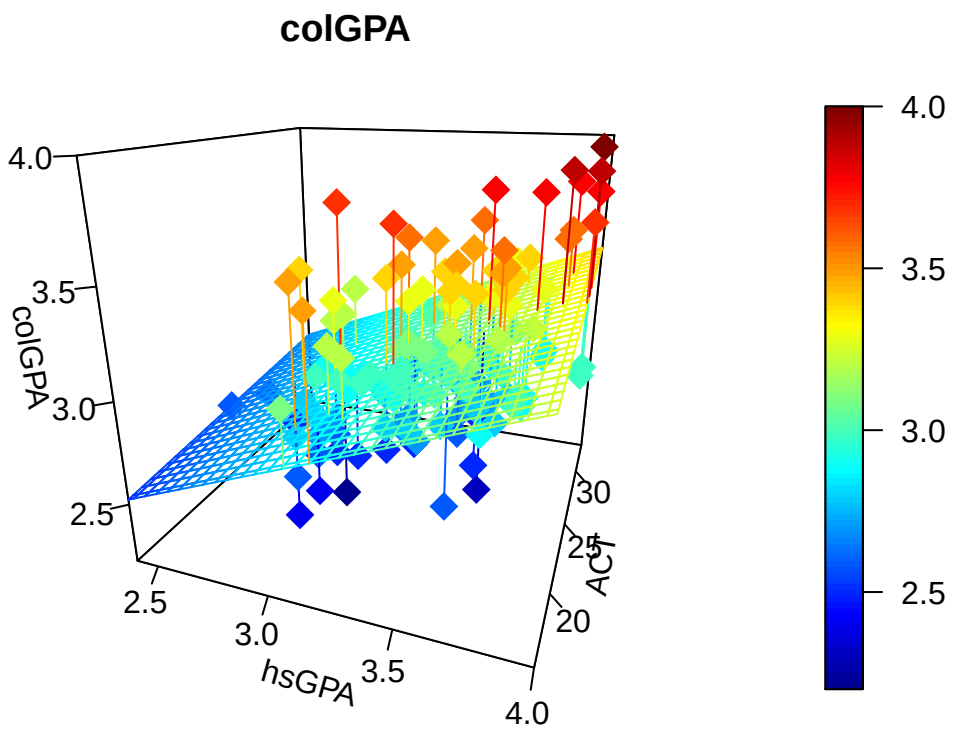


Figure 1.20: College GPA High School GPA + Achievment test score

```
# Parameter estimates for full and simple model:
beta.hat <- coef( lm(colGPA ~ ACT+hsGPA, data=gpa1))
beta.hat
```

```
## (Intercept)         ACT       hsGPA
## 1.286327767 0.009426012 0.453455885
```

Now, lets omit one variable in the regression

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 \tag{1.14}$$

```
# Relation between regressors:
delta.tilde <- coef( lm(hsGPA ~ ACT, data=gpa1) )
delta.tilde
```

```
## (Intercept)         ACT
##  2.46253658  0.03889675
```

The parameter $\hat{\beta}_1$ is the estimated effect of increasing x_1 by one unit (and **NOT** keeping x_2 fixed). It can be related to $\hat{\beta}_1$ using the formula

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 \tilde{\delta}_1 \tag{1.15}$$

where $\tilde{\delta}_1$ is the slope parameters of the linear regression of x_2 on x_1

$$\hat{x}_2 = \tilde{\delta}_0 + \tilde{\delta}_1 x_1 \tag{1.16}$$

```
# Omitted variables formula for beta1.tilde:
beta.hat["ACT"] + beta.hat["hsGPA"]*delta.tilde["ACT"]
```

```
##        ACT
## 0.02706397
```

```
# Actual regression with hsGPA omitted:
summary(lm(colGPA ~ ACT, data=gpa1))
```

```
##
## Call:
## lm(formula = colGPA ~ ACT, data = gpa1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.85251 -0.25251 -0.04426  0.26400  0.89336
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.40298    0.26420   9.095  8.8e-16 ***
## ACT          0.02706    0.01086   2.491   0.0139 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3656 on 139 degrees of freedom
## Multiple R-squared:  0.04275,    Adjusted R-squared:  0.03586
## F-statistic: 6.207 on 1 and 139 DF,  p-value: 0.0139
```

In this example, the indirect effect is actually stronger than the direct effect. *ACT* predicts *colGPA* mainly because it is related to *hsGPA* which in turn is strongly related to *colGPA*.

1.2.2 Standard errors, Multicollinearity and VIF

Multicollinearity means that two or more regressors in a multiple regression model are strongly correlated. If the correlation between two or more regressors is perfect, that is, one regressor can be written as a linear combination of the other(s), we have perfect multicollinearity. While strong multicollinearity in general is unpleasant as it causes the variance of the OLS estimator to be large (we will discuss this in more detail later), the presence of perfect multicollinearity makes it impossible to solve for the OLS estimator, i.e., the model cannot be estimated in the first place.

1.2.2.1 Perfect multicollinearity

We will work first with the *CAschools* data from the **AER** package to simulate an example of perfect multicollinearity

```
data("CASchools", package = "AER")
?CASchools
```

```
# define the fraction of English learners
CASchools$FracEL <- CASchools$english / 100

# check the correlation between CASchools$FracEL and CASchools$english
cor(CASchools$FracEL, CASchools$english)
```

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

```
# estimate the model
mult.mod <- lm(read ~ students + english + FracEL, data = CASchools)

# obtain a summary of the model
summary(mult.mod)
```

```
##
## Call:
## lm(formula = read ~ students + english + FracEL, data = CASchools)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -51.077  -9.767  -0.695   9.097  40.005
##
## Coefficients: (1 not defined because of singularities)
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.665e+02  9.771e-01 682.054   <2e-16 ***
## students     3.326e-04  1.941e-04   1.714   0.0873 .
## english     -7.843e-01  4.153e-02 -18.886   <2e-16 ***
## FracEL              NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.53 on 417 degrees of freedom
## Multiple R-squared:  0.4802, Adjusted R-squared:  0.4777
## F-statistic: 192.6 on 2 and 417 DF,  p-value: < 2.2e-16
```

The row *FracEL* in the coefficients section of the output consists of NA entries since FracEL was excluded from the model.

Another example of perfect multicollinearity is known as the dummy variable trap. This may occur when multiple dummy variables are used as regressors. A common case for this is when dummies are used to sort the data into mutually exclusive categories. For example, suppose we have spatial information that indicates whether a school is located in the North, West, South or East of California.

```
# set seed for reproducibility
set.seed(1)

# generate artificial data on location
CASchools$direction <- sample(c("West", "North", "South", "East"),
                              420,
                              replace = T)

# estimate the model
mult.mod <- lm(read ~ students + english + direction, data = CASchools)

# obtain a model summary
summary(mult.mod)
```

```
##
## Call:
## lm(formula = read ~ students + english + direction, data = CASchools)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -50.518  -9.690  -1.146   8.926  39.698
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)     6.659e+02  1.547e+00 430.575   <2e-16 ***
## students        3.177e-04  1.945e-04   1.634    0.103
## english        -7.844e-01  4.154e-02 -18.882   <2e-16 ***
## directionNorth  9.185e-01  1.934e+00   0.475    0.635
## directionSouth -1.119e+00  2.081e+00  -0.537    0.591
## directionWest   2.436e+00  2.057e+00   1.184    0.237
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.53 on 414 degrees of freedom
## Multiple R-squared:  0.484,  Adjusted R-squared:  0.4778
## F-statistic: 77.67 on 5 and 414 DF,  p-value: < 2.2e-16
```

Notice that R solves the problem on its own by generating and including the dummies directionNorth, directionSouth and directionWest but omitting directionEast. Of course, the omission of every other dummy instead would achieve the same. Another solution would be to exclude the constant and to include all dummies instead.

A last example considers the case where a perfect linear relationship arises from redundant regressors. Suppose we have a regressor
Spanish speakers, *spanish*, , the percentage of English speakers in the school where

$$spanish = 100 - english \tag{1.17}$$

and both *spanish* and *english* are included in a regression model.

```
# Percentage of english speakers
CASchools$spanish <- 100 - CASchools$english

# estimate the model
mult.mod <- lm(read ~ students + english + spanish, data = CASchools)

# obtain a model summary
summary(mult.mod)
```

```
##
## Call:
## lm(formula = read ~ students + english + spanish, data = CASchools)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -51.077  -9.767  -0.695   9.097  40.005
##
## Coefficients: (1 not defined because of singularities)
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.665e+02  9.771e-01 682.054   <2e-16 ***
## students     3.326e-04  1.941e-04   1.714   0.0873 .
## english     -7.843e-01  4.153e-02 -18.886   <2e-16 ***
## spanish             NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 14.53 on 417 degrees of freedom
## Multiple R-squared:  0.4802, Adjusted R-squared:  0.4777
## F-statistic: 192.6 on 2 and 417 DF,  p-value: < 2.2e-16
```

Once more, `lm()` refuses to estimate the full model using OLS and excludes spanish.

1.2.2.2 Imperfect multicollinearity

As opposed to perfect multicollinearity, imperfect multicollinearity is — to a certain extent — less of a problem. In fact, imperfect multicollinearity is the reason why we are interested in estimating multiple regression models in the first place: the OLS estimator allows us to isolate influences of correlated regressors on the dependent variable. If it was not for these dependencies, there would not be a reason to resort to a multiple regression approach and we could simply work with a single-regressor model. However, this is rarely the case in applications. We already know that ignoring dependencies among regressors which influence the outcome variable has an adverse effect on estimation results.

Simulation study: imperfect multicollinearity

```
# set number of observations
n <- 50

# initialize vectors of coefficients
coefs1 <- cbind("hat_beta_1" = numeric(10000), "hat_beta_2" = numeric(10000))
coefs2 <- coefs1

# set seed
set.seed(1)

# loop sampling and estimation
for (i in 1:1000) {

  # for cov(X_1,X_2) = 0.25
  X <- rmvnorm(n, c(50, 100), sigma = cbind(c(10, 2.5), c(2.5, 10))) # function from the mvtnorm packag
  u <- rnorm(n, sd = 5)
  Y <- 5 + 2.5 * X[, 1] + 3 * X[, 2] + u
  coefs1[i, ] <- lm(Y ~ X[, 1] + X[, 2])$coefficients[-1]

  # for cov(X_1,X_2) = 0.85
  X <- rmvnorm(n, c(50, 100), sigma = cbind(c(10, 8.5), c(8.5, 10)))
  Y <- 5 + 2.5 * X[, 1] + 3 * X[, 2] + u
  imperf_multicol <- lm(Y ~ X[, 1] + X[, 2])
  coefs2[i, ] <- lm(Y ~ X[, 1] + X[, 2])$coefficients[-1]

}

# obtain variance estimates
diag(var(coefs1))
```

```
## hat_beta_1 hat_beta_2
##  0.5698281  0.8163287
```

```
diag(var(coefs2))
```

```
## hat_beta_1 hat_beta_2
##  0.5834243  0.8402187
```

We are interested in the variances which are the diagonal elements. We see that due to the high collinearity, the variances of $\hat{\beta}_1$ and $\hat{\beta}_2$ and have increased, meaning it is more difficult to precisely estimate the true coefficients.

The variance inflation factor, *VIF*, accounts for (imperfect) multicollinearity.If x_t is highly related to the other regressors, R_j^2 and therefore also VIF_j and the variance of $\hat{\beta}_j$ are large.

$$\frac{1}{1-R_j^2} \tag{1.18}$$

```
GPAres <- lm(colGPA ~ hsGPA + ACT, data = gpa1)
SER<-summary(GPAres)$sigma
```

```
# regressing hsGPA on ACT for calculation of R2 & VIF
( R2.hsGPA  <- summary( lm(hsGPA~ACT, data=gpa1) )$r.squared )
```

```
## [1] 0.1195815
```

```
( VIF.hsGPA <- 1/(1-R2.hsGPA) )
```

```
## [1] 1.135823
```

The **car** package implements the command `vif()` for each regressor

```
vif(GPAres)
```

```
##    hsGPA      ACT
## 1.135823 1.135823
```

```
vif(imperf_multicol) # from the simulated data
```

```
##   X[, 1]   X[, 2]
## 4.932864 4.932864
```

1.2.3 Reporting Regression Results

As we start moving towards the comparing different regression models this section provides a discussion on how to report regression reports in R. Depending on your script (R scripts, R Markdown, bookdown) and what your desired output format is (LaTeX, word, html) the exact approach might differ. There are multiple packages to format regression or table output, most notably **stargazer**[1], **huxtable**, **Hmisc** and **xtable**. One can also tidy the the regression output as well as tables with **broom** or **summarytool**. The wrapper `knitr::kable()` is a support function that renders the table in an R Markdown in a pretty way.

1.2.3.1 Table

```
knitr::kable(
  head(gpa1[,1:8], 10), booktabs = TRUE,
  caption = "A table of the first eight columns and ten rows of the gpa1 data."
)
```

Reporting summary statistics (transposed)

[1]Stargazer supports ton of options, including theming the LaTex output to journal styles. However, stargazer was written before R Markdown was really a thing, so it has excellent support for HTML and LaTeX output, but that's it. Including stargazer tables in an R Markdown document is a hassle. **huxtable** on the the contrary plays really nice with **broom** and the **tidyverse**. For more info see Andrew Heiss blog.