SG1022 Seminar 1: R BootCamp (1)

Christopher Gandrud

2016

Objectives for the topic

- Motivation
- Getting around R and RStudio
- Basics of object-oriented programming in R
- Simple R data structures

Motivation: Government

Government agencies are increasingly adopting the technologies and methods of open data science.

Motivation: Government

- Public data is increasingly **accessible**.
  - e.g. World Bank Development Indicators, GovData Germany, data.gov.uk, New York City, data.gov
- Governments rely on data analysis for evidence based decision-making.
  - Tools of open data analysis enable better use of data **within** and **between** government actors.
  - Governments can take advantage of analyses done by **third parties**.

Motivation: Government

- They are also **sharing** and **collaboratively** developing code; **reducing development costs** and **improving applications**.
- Version control to **increase engagement with the legislative process**.
  - San Francisco laws are now forkable.

Motivation: NGO

NGO's are becoming increasingly data-oriented and need people with **skills** to **handle and analyse** this data.
Ex. One of my former students recently co-founded CorrelAid to assist NGOs with data analysis.

Motivation: Business

Data analysis and R programming skills in particular are **highly valued** in businesses such as finance and management.

AVERAGE SALARY FOR High Paying Skills and Experience

SKILL	2013	YR/YR CHANGE
R	$ 115,531	n/a
NoSQL	$ 114,796	1.6%
MapReduce	$ 114,396	n/a
PMBok	$ 112,382	1.3%
Cassandra	$ 112,382	n/a
Omnigraffle	$ 111,039	0.3%
Pig	$ 109,561	n/a
SOA (Service Oriented Architecture)	$ 108,997	-0.5%
Hadoop	$ 108,669	-5.6%
Mongo DB	$ 107,825	-0.4%

Source: Revolution Analytics (2014)

What is R?

Open source programming language, with a particular focus on statistical programming.
History: Originally (in 1993) an implementation of the S programming language (Bell Labs), by **R**oss Ihaka and **R**obert Gentleman (hence **R**) at University of Auckland.
Currently the R Foundation for Statistical Computing is based in Vienna. Development is distributed globally.

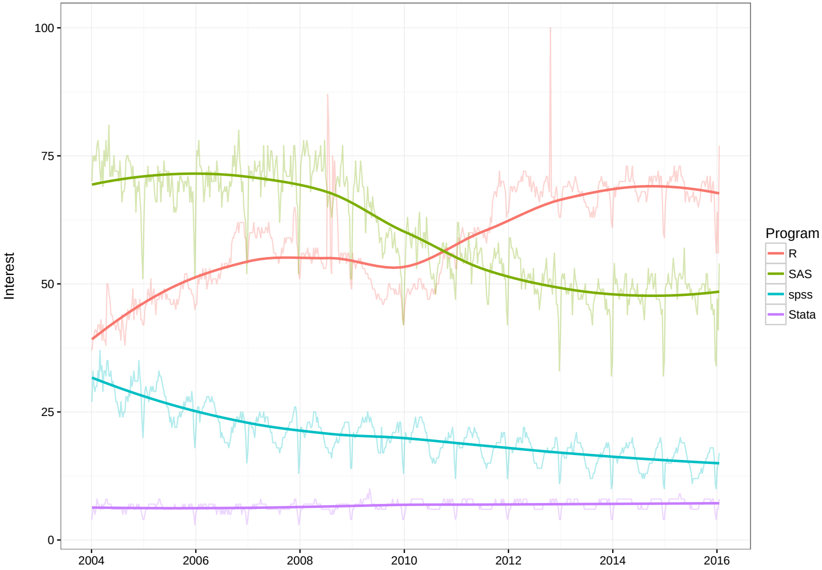
R and SPSS

SPSS is a **computer program** that can do statistics.
R is a **programming language** with strong statistical capabilities.

Implications

- Learning R means **learning computer programming** (applied statistics is programming!)
- R is much, much more **flexible** than SPSS. For example, can do text analysis, SPSS can't.
- R is open source = **FREE**. SPSS > **$1,000**.
- Increasingly popular in **high demand data science jobs**.

Comparative Google Search Interest



Source: `https://www.google.com/trends`

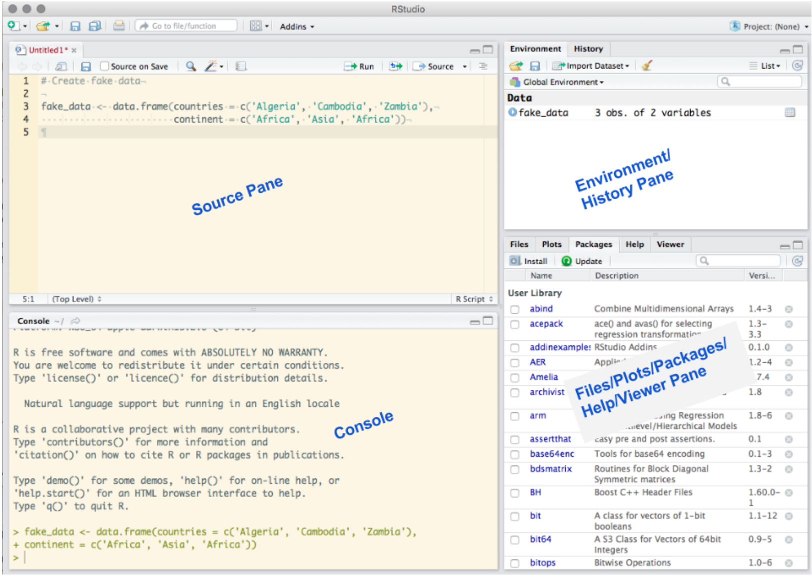
Growing popularity

R can be easily expanded by **user created packages** hosted on GitHub and/or CRAN.

RStudio:

RStudio is an Integrated Developer Environment (IDE) that makes using R and other reproducible research tools easier.

R is a programming language. So, you will have to change how you think about data/interacting with the computer.



Source Pane
Environment/
History Pane
Console
Files/Plots/Packages/
Help/Viewer Pane

Fundamentals of the R language

Like many other popular programming languages, R is **object-oriented**.
Objects are R's nouns. They include (not exhaustive):

- character strings (e.g. words)
- numbers
- vectors of numbers or character strings
- matrices
- data frames
- lists

Assignment

You use the **assignment operator** (<-) to assign character strings, numbers, vectors, etc. to object names

```
## Assign the number 10 to an object called number
number <- 10

number
```

```
## [1] 10
```

```
# Assign Hello World to an object called words
words <- "Hello World"

words
```

```
## [1] "Hello World"
```

Assignment

You can also use the equality sign (=):

```
number = 10

number
```

```
## [1] 10
```

Note: it has a slightly different meaning.
See StackOverflow discussion.

R is a Calculator

```
2 + 3
```

```
## [1] 5
```

```
2 - 3
```

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

```
2 * 3
```

```
## [1] 6
```

```
2 / 3
```

```
## [1] 0.6666667
```

Special values in R

- `NA`: not available, missing
- `NULL`: does not exist, is undefined
- `TRUE`, `T`: logical true. **Logical** is also an object class.
- `FALSE`, `F`: logical false

Finding special values

Function	Meaning
`is.na`	Is the value `NA`
`is.null`	Is the value `NULL`
`isTRUE`	Is the value `TRUE`
`!isTRUE`	Is the value `FALSE`

```r
absent <- NA
is.na(absent)
```

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

Operator	Meaning
`<`	less than
`>`	greater than
`==`	equal to
`<=`	less than or equal to
`>=`	greater than or equal to
`!=`	not equal to
`a | b`	a or b
`a & b`	a and b

Classes

Objects have distinct classes.

```
# Find the class of number
class(number)
```

```
## [1] "numeric"
```

```
# Find the class of absent
class(absent)
```

```
## [1] "logical"
```

Naming objects

- Object names **cannot have spaces**
  - Use `CamelCase`, `name_underscore`, or `name.period`
- Avoid creating an object with the same name as a function (e.g. `c` and `t`) or special value (`NA`, `NULL`, `TRUE`, `FALSE`).
- Use **descriptive object names**!
  - Not: `obj1`, `obj2`
- Each object name must be **unique** in an environment.
  - Assigning something to an object name that is already in use will **overwrite the object's previous contents**.

Finding objects

```r
# Find objects in your workspace
ls()
```

```
## [1] "absent"   "number"   "searches" "words"
```

Or the *Environment* tab in RStudio

Style Guides

As with natural language writing, it is a good idea to stick to one style guide with your R code:

- Google's R Style Guide
- Hadely Wickham's R Style Guide

Vectors

A vector is an **ordered collection** of numbers, characters, etc. of the **same type**.
Vectors can be created with the c (**combine**) function.

```
# Create numeric vector
numeric_vector <- c(1, 2, 3)

# Create character vector
character_vector <- c('Albania', 'Botswana', 'Cambodia')
```

Factor class vector

Categorical variables are called **factors** in R.

```
# Create numeric vector
fruits <- c(1, 1, 2)

# Create character vector for factor labels
fruit_names <- c('apples', 'mangos')

# Convert to labelled factor
fruits_factor <- factor(fruits, labels = fruit_names)

summary(fruits_factor)
```

```
## apples mangos
##      2      1
```

Matrices

Matrices are collections of vectors **with the same length and class**.

```
# Combine numeric_vector and character_vector into a matri
combined <- cbind(numeric_vector, character_vector)

combined
```

```
##      numeric_vector character_vector
## [1,] "1"            "Albania"
## [2,] "2"            "Botswana"
## [3,] "3"            "Cambodia"
```

Note (1): R *coerced* `numeric_vector` into a character vector.
Note (2): You can `rbind` new rows onto a matrix.

Data frames

Data frames are collections of vectors with the same length.
Each column (vector) can be of a **different class**.

```
# Combine numeric_vector and character_vector into a data f
combined_df <- data.frame(numeric_vector, character_vector,
                          stringsAsFactors = FALSE)


combined_df
```

```
##   numeric_vector character_vector
## 1              1          Albania
## 2              2         Botswana
## 3              3         Cambodia
```

Lists

A list is an object containing other objects that can have **different** lengths and classes.

```
# Create a list with three objects of different lengths
test_list <- list(countries = character_vector, not_there =
                  more_numbers = 1:10)
test_list
```

```
## $countries
## [1] "Albania"  "Botswana" "Cambodia"
##
## $not_there
## [1] NA NA
##
## $more_numbers
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Functions

Functions do things to/with objects. Functions are like **R's verbs**. When using functions to do things to objects, they are always followed by parentheses (). The parentheses contain the **arguments**. Arguments are separated by commas.

```
# Summarise combined_df
summary(combined_df, digits = 2)
```

```
##  numeric_vector character_vector
##  Min.   :1.0    Length:3
##  1st Qu.:1.5    Class :character
##  Median :2.0    Mode  :character
##  Mean   :2.0
##  3rd Qu.:2.5
##  Max.   :3.0
```

Functions help

Use `?` to find out what arguments a function can take.

```
?summary
```

The help page will also show the function's **default argument values**.

Component selection ($)

The $ is known as the component selector. It selects a component of an object.

```
combined_df$character_vector
```

```
## [1] "Albania"  "Botswana" "Cambodia"
```

Subscripts []

You can use subscripts [] to also select components.
For data frames they have a [row, column] pattern.

```
# Select the second row and first column of combined_df
combined_df[2, 1]
```

```
## [1] 2
```

```
# Select the first two rows
combined_df[c(1, 2), ]
```

```
##   numeric_vector character_vector
## 1              1          Albania
## 2              2         Botswana
```

Subscripts []

```
# Select the character_vector column
combined_df[, 'character_vector']
```

```
## [1] "Albania"  "Botswana" "Cambodia"
```

Assignment with elements of objects

You can use assignment with parts of objects. For example:

```
combined_df$character_vector[3] <- 'China'
combined_df$character_vector
```

```
## [1] "Albania"  "Botswana" "China"
```

You can even add new variables:

```
combined_df$new_var <- 1:3
combined_df
```

```
##   numeric_vector character_vector new_var
## 1              1          Albania       1
## 2              2         Botswana       2
## 3              3            China       3
```

Packages

You can greatly expand the number of functions by installing and loading user-created packages.

```
# Install dplyr package
install.packages('dplyr')

# Load dplyr package
library(dplyr)
```

You can also call a function directly from a specific package with the double colon operator (`::`).

```
Grouped <- dplyr::group_by(combined_df, character_vector)
```

R's build-in data sets

List internal data sets:

```
data()
```

Load **swiss** data set:

```
data(swiss)
```

Find data description:

```
?swiss
```

R's build-in data sets

Find variable names:

```
names(swiss)
```

```
## [1] "Fertility"        "Agriculture"      "Examination"
## [4] "Education"        "Catholic"         "Infant.Mortal
```

See the first three rows and four columns

```
head(swiss[1:3, 1:4])
```

```
##              Fertility Agriculture Examination Educatio
## Courtelary        80.2        17.0          15        1
## Delemont          83.1        45.1           6
## Franches-Mnt      92.5        39.7           5
```

Creating Functions

You can create a function to find the sample mean ($\bar{x} = \frac{\sum x}{n}$) of a vector.

```
fun_mean <- function(x){
    sum(x) / length(x)
}

## Find the mean
fun_mean(x = swiss$Examination)
```

```
## [1] 16.48936
```

Why create functions?

Functions:

- Simplify your code if you do repeated tasks.
- Lead to fewer mistakes.
- Are easier to understand.
- Save time over the long run–a general solution to problems in different contexts.

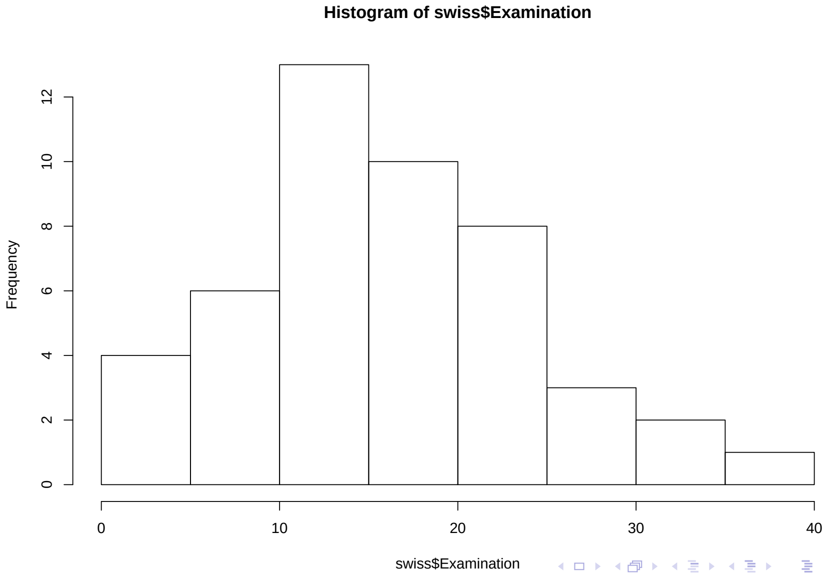
Descriptive statistics: review

Descriptive Statistics: describe samples
Stats 101: describe sample **distributions** with appropriate measure of:

- **central tendancy**
- **variability**

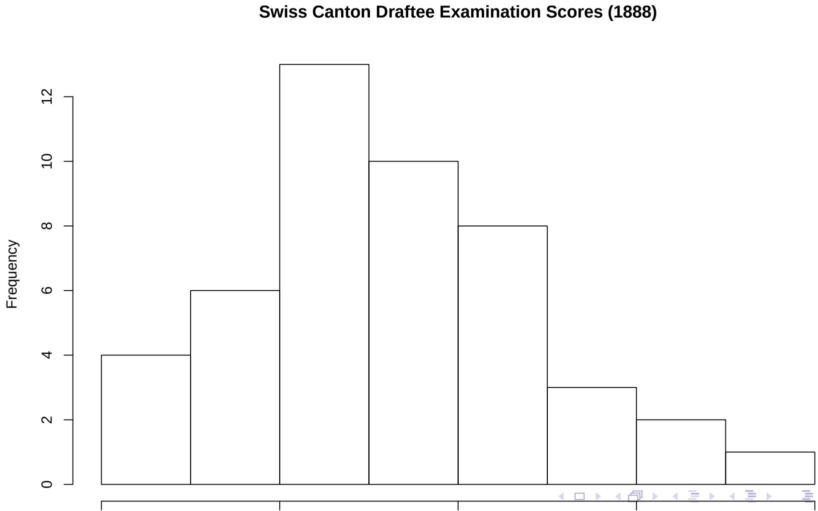
Histograms

```r
hist(swiss$Examination)
```



Histograms: styling

```
hist(swiss$Examination,
     main = 'Swiss Canton Draftee Examination Scores (1888)
     xlab = '% receiving highest mark on army exam')
```



Finding means

(or use the mean function in base R)

```
mean(swiss$Examination)
```

```
## [1] 16.48936
```

If you have missing values (`NA`):

```
mean(swiss$Examination, na.rm = TRUE)
```

Digression: Loops

You can 'loop' through the data set to find the mean for each column

```
for (i in 1:length(names(swiss))) {
    swiss[, i] %>%
    mean() %>%
    round(digits = 1) %>%
    paste(names(swiss)[i], ., '\n') %>%  # the . directs th
    cat()
}
```

```
## Fertility 70.1
## Agriculture 50.7
## Examination 16.5
## Education 11
## Catholic 41.1
## Infant.Mortality 19.9
```

Other functions for central tendency

Median

```
median(swiss$Examination)
```

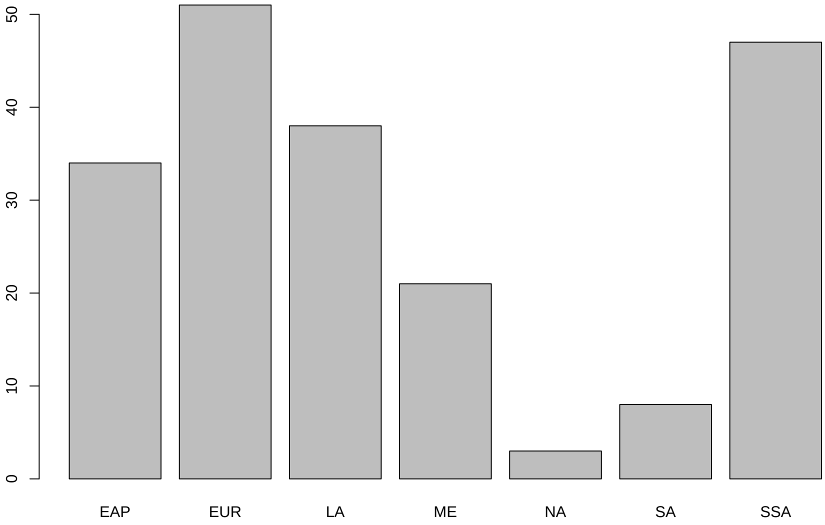
```
## [1] 16
```

Mode

`mode` is not an R function to find the statistical mode.
Instead use `summary` for factor nominal variables or make a bar chart.

Simple bar chart for nominal

```
devtools::source_url('http://bit.ly/OTWEGS')
plot(MortalityGDP$region, xlab = 'Region')
```



Variation

Variation is "perhaps the **most important quantity** in statistical analysis. The greater the variability in the data, the greater will be our **uncertainty** in the values of the parameters estimated . . . and the **lower our ability to distinguish between competing hypotheses**" (Crawley 2005, 33)

Variation

Range:

```r
range(swiss$Examination)
```

```
## [1]  3 37
```

Quartiles:

```r
summary(swiss$Examination)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    3.00   12.00   16.00   16.49   22.00   37.00
```

Variation

Interquartile Range ($IQR = Q_3 - Q_1$):

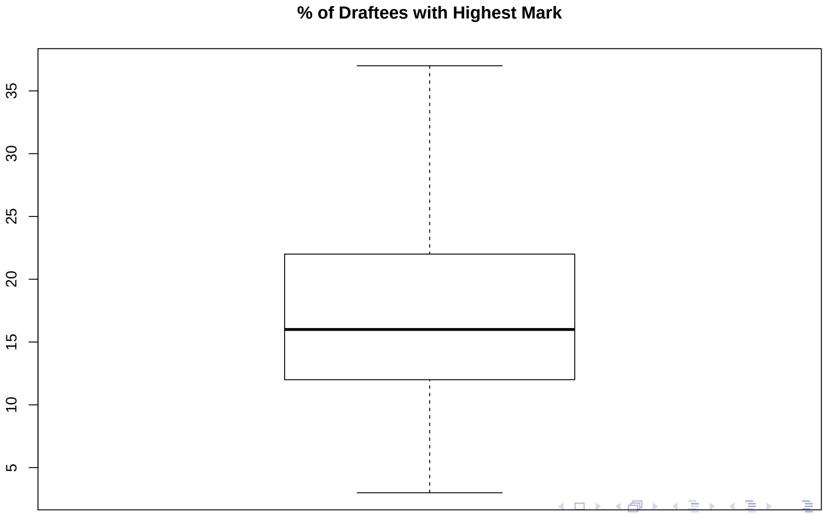
```
IQR(swiss$Examination)
```

```
## [1] 10
```

Variation

Boxplots:

```
boxplot(swiss$Examination, main = '% of Draftees with Highe
```



Variation: Sum of Squares

Sum of squares (summing deviations from the mean):

$$\text{Sum of Squares} = \sum (x - \bar{x})^2$$

- But sum of squares always gets bigger with a larger sample size.
  - Unless the new values exactly equal the mean.

Variation: Degrees of Freedom

Degrees of freedom (number of values that are free to vary):
For the mean:

$$\mathrm{df} = n - 1$$

Why?
For a given mean and sample size, $n - 1$ values can vary, but the nth value must always be the same. See Crawley (2005, 36-37).

Variation: Variance

We can use degrees of freedom to create an "unbiased" measure of variability that is not dependent on the sample size.

Variance (s^2):

$$s^2 = \frac{\text{Sum of Squares}}{\text{Degrees of Freedom}} = \frac{\sum(x - \bar{x})^2}{n - 1}$$

But this is not in the same units as the mean, so it can be confusing to interpret.

Variation: Standard Deviation

Use standard deviation **(s)** to put variance in terms of the mean:

$$s = \sqrt{s^2}$$

Variation: Standard Error

The **standard error** of the mean:
If we think of the variation around a central tendency as a measure of the **unreliability** of an estimate (mean) in a population, then we want the measure to **decrease as the sample size goes up**.

$$\mathrm{SE}_{\bar{x}} = \sqrt{\frac{s^2}{n}}$$

Note: $\sqrt{}$ so that the dimensions of the measure of unreliability and the parameter whose variability is being measured are the same.
Good overview of variance, degrees of freedom, and standard errors in Crawley (2005, Ch. 4).

Variation: Variance and Standard Deviation

Variance:

```r
var(swiss$Examination)
```

```
## [1] 63.64662
```

Standard Deviation:

```r
sd(swiss$Examination)
```

```
## [1] 7.977883
```

Variation: Standard Error

Standard Error:

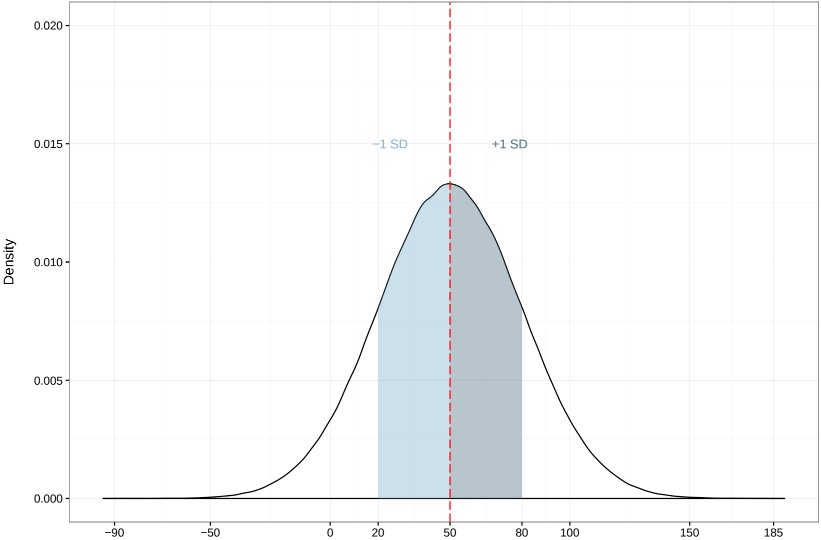
```
sd_error <- function(x) {
    sd(x) / sqrt(length(x))
}

sd_error(swiss$Examination)
```

```
## [1] 1.163694
```

Playing with distributions

Simulated normally distributed data with SD of 30 and mean 50

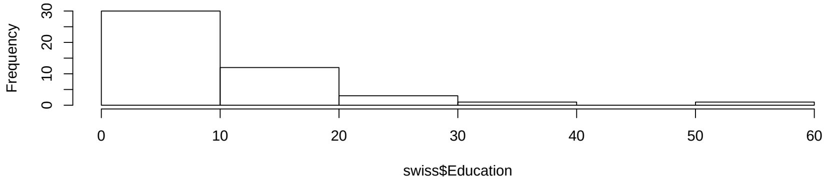
```
Normal30 <- rnorm(1e+6, mean = 50, sd = 30)
```



Transform skewed data

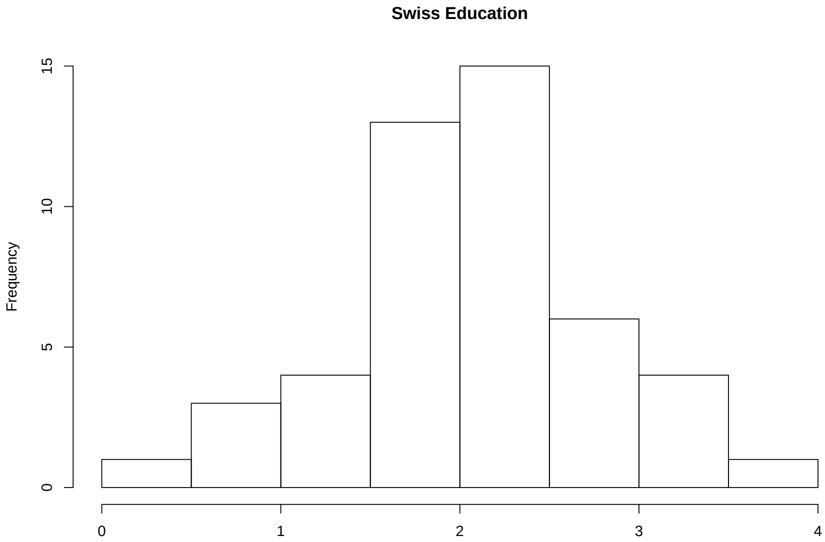
Highly skewed data can be transformed to have a normal distribution.
Helps correct two violations of key assumptions: (a) non-linearity and (b) heteroskedasticity.

```
hist(swiss$Education, main = '')
```



Natural log transformed skewed data

```
log(swiss$Education) %>% hist(main = "Swiss Education")
```



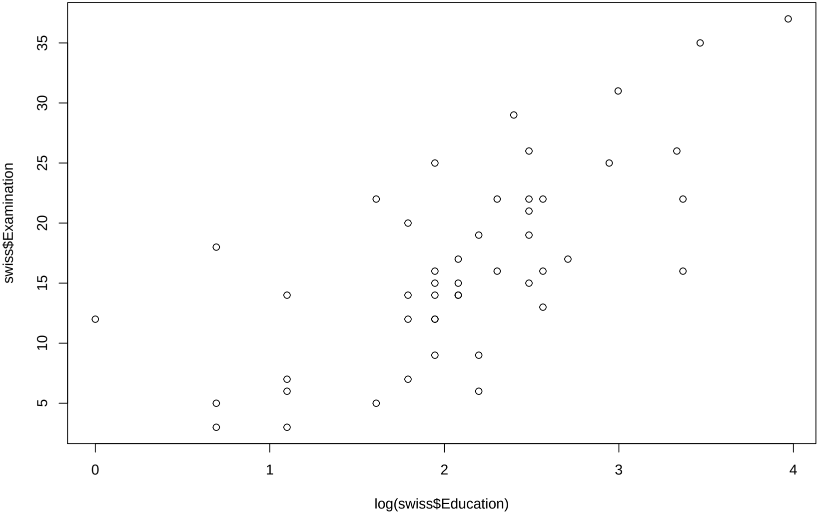
Transformations

The natural log transformation is only useful for data that **does not contain zeros**.

See `http://robjhyndman.com/hyndsight/transformations/` for suggestions on other transformations such as Box-Cox and Inverse Hyperbolic Sine.

Joint distributions (continuous variables)

```
plot(log(swiss$Education), swiss$Examination)
```



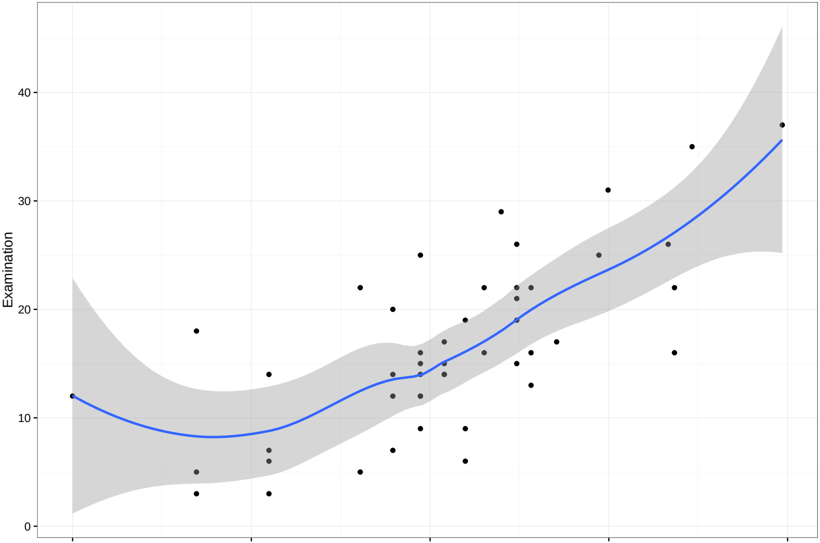
Summarise with correlation coefficients

```
cor.test(log(swiss$Education), swiss$Examination)
```

```
##
##  Pearson's product-moment correlation
##
## data:  log(swiss$Education) and swiss$Examination
## t = 6.4313, df = 45, p-value = 7.133e-08
## alternative hypothesis: true correlation is not equal to
## 95 percent confidence interval:
##  0.5053087 0.8168779
## sample estimates:
##       cor
## 0.6920531
```

Summarise with loess

```
ggplot2::ggplot(swiss, aes(log(Education), Examination)) +
    geom_point() + geom_smooth() + theme_bw()
```



Joint distributions (categorical variables)

Contingency table

```r
library(MASS)
```

```
##
## Attaching package: 'MASS'
## The following object is masked from 'package:dplyr':
##
##     select
```

```r
smoking <- table(survey$Smoke, survey$Exer)

smoking
```

```
##
##          Freq None Some
##   Heavy     7    1    3
##   Never    87   18   84
##   Occas    12    3    4
```

Cross-tabs with proportions

```r
prop.table(smoking, margin = 1) # row proportions
```

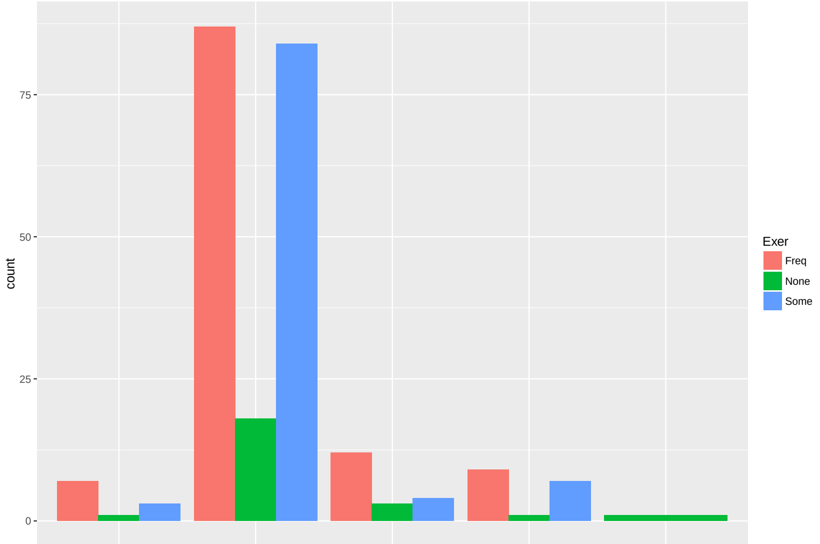
```
##
##              Freq       None       Some
##   Heavy 0.63636364 0.09090909 0.27272727
##   Never 0.46031746 0.09523810 0.44444444
##   Occas 0.63157895 0.15789474 0.21052632
##   Regul 0.52941176 0.05882353 0.41176471
```

```r
prop.table(smoking, margin = 2) # column proportions
```

```
##
##              Freq       None       Some
##   Heavy 0.06086957 0.04347826 0.03061224
##   Never 0.75652174 0.78260870 0.85714286
##   Occas 0.10434783 0.13043478 0.04081633
##   Regul 0.07826087 0.04347826 0.07142857
```

Plotting

```
ggplot2::ggplot(survey, aes(Smoke, fill = Exer)) +
    geom_bar(position = 'dodge')
```



Joint distributions (categorical variables)

χ^2 Test

```r
chisq.test(smoking)
```

```
##
##  Pearson's Chi-squared test
##
## data:  smoking
## X-squared = 5.4885, df = 6, p-value = 0.4828
```

Programming Hint (1)

Google is your friend!
If you have a question, someone else has probably asked it before.

Programming Hint (2)

Always close!
In R this means closing:

- `()`
- `[]`
- `{}`
- `''`
- `""`

Programming Hint (3)

There are usually **many ways to acheive the same goal**, but . . make your code as **simple as possible**.

- Easier to read.
- Easier to write (ultimately).
- Easier to find mistakes.
- Often computationally more efficient.

One way to do this is to **define things once**–e.g. use variables to contain values and custom functions to contain multiple sequential function calls.

Programming Hint (3)

Bad

```r
mean(rnorm(1000))
```

```
## [1] -0.02922025
```

```r
sd(rnorm(1000))
```

```
## [1] 1.00674
```

Programming Hint (3)

Good

```r
rand_sample <- rnorm(1000)

mean(rand_sample)
```

```
## [1] 0.02082105
```

```r
sd(rand_sample)
```

```
## [1] 0.9814993
```

Seminar: Start using R!

- Code School Interactive Tutorial: `http://tryr.codeschool.com/`
- Open RStudio and create a new source code file. Copy in and run some of the Code School Examples.