MPP-E1180 Lecture 2: Introduction to the R Programming Language

Christopher Gandrud

25 September 2014

Objectives for the week

- Reminder: **Pair Assignment 1**
- Basics of object oriented programming in R
- Simple R data structures
- Simple descriptive statistics and plotting with Base R

Pair Assignment 1

- **Due:** Midnight 26 September.
- Learning objectives: develop your understanding of
  - **file structures**,
  - **version control**,
  - **basic R data structures** and **descriptive statistics**.

Pair Assignment 1

Each pair will create a **new public GitHub repository**

- Must be **fully documented**, including with a **descriptive README.md** file. Your code must be human readable and clearly commented.
- Include **R source** code files that:
  - Access at least **two** core R data sets
  - Illustrate the datas' distributions using a variety of **relevant descriptive statistics**
  - Two files must be **dynamically linked**
- **Another pair** makes a **pull request**. And this is discussed/merged.

What is R?

Open source programming language, with a particular focus on statistical programming.

History: Originally 1993 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.

Growing popularity

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

How to Cite R

```r
citation()
```

```
##
## To cite R in publications use:
##
##   R Core Team (2014). R: A language and environment for
##   statistical computing. R Foundation for Statistical Co
##   Vienna, Austria. URL http://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistic
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Comput
##     address = {Vienna, Austria},
##     year = {2014},
##     url = {http://www.R-project.org/},
```

Fundamentals of the R language

R is **object-oriented**.

Objects are R's nouns. They include (but are more):

- a character string (e.g. word)
- a number
- a vector of numbers or character strings
- a matrix
- a data frame
- a list

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 =, but it has a slightly different meaning.

See StackOverflow discussion.

Special values in R

- `NA`: not available, missing
  - Find with `is.na`
- `NULL`: does not exist, is undefined
  - Find with `is.null`
- `TRUE`, `T`: logical true. ‘‘logical’’ is also an object class
  - Find with `isTRUE`
- `FALSE`, `F`: logical false
  - Find with `!isTRUE`

<table>
<tr><th>Operator</th><th>Meaning</th></tr>
<tr><td><code><</code></td><td>less than</td></tr>
<tr><td><code>></code></td><td>greater than</td></tr>
<tr><td><code>==</code></td><td>equal to</td></tr>
<tr><td><code><=</code></td><td>less than or equal to</td></tr>
<tr><td><code>>=</code></td><td>greater than or equal to</td></tr>
<tr><td><code>!=</code></td><td>not equal to</td></tr>
<tr><td><code>a | b</code></td><td>a or b</td></tr>
<tr><td><code>a & b</code></td><td>a and b</td></tr>
</table>

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**!
- Each object name must be **unique** in a workspace.
  - 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] "number" "words"
```

Classes

Objects have distinct classes.

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

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

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**

```
# 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: In addition to `cbind` 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 a vector containing other objects.

They the objects can have different lengths and classes.

```
# Create a list with three objects of different lengths
test_list <- list(countries = character_vector,
                  not_there = c(NA, NA),
                  more_numbers = 1:100)

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  11  12 
```

Functions

Functions do things to objects. Functions are like **R's verbs**.

When using them to do things to objects, they are always followed by parentheses `()`. The parentheses contain the **arguments**. Arguments are separated by commas.

```r
# 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

To find out what arguments a function can take use `?`.

```
?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.

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

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

```r
# 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"
```

Packages

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

```r
# 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 (`::`).

```r
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        12
## Delemont          83.1        45.1           6         9
## Franches-Mnt      92.5        39.7           5         5
```

What all the cool kids are doing: piping

Pipe: pass a value forward to a function call.

Why?

- Faster compilation.
- Enhanced code readability.

In R use %>% from the magrittr package.

%>% passes a value to the **first argument** of the next function call.

Simple piping example

Not piped:

```
values <- rnorm(1000, mean = 10)
value_mean <- mean(values)
round(value_mean, digits = 2)
```

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

Piped:

```
library(magrittr)

rnorm(1000, mean = 10) %>% mean() %>% round(digits = 2)
```

```
## [1] 10.04
```

Descriptive statistics: review

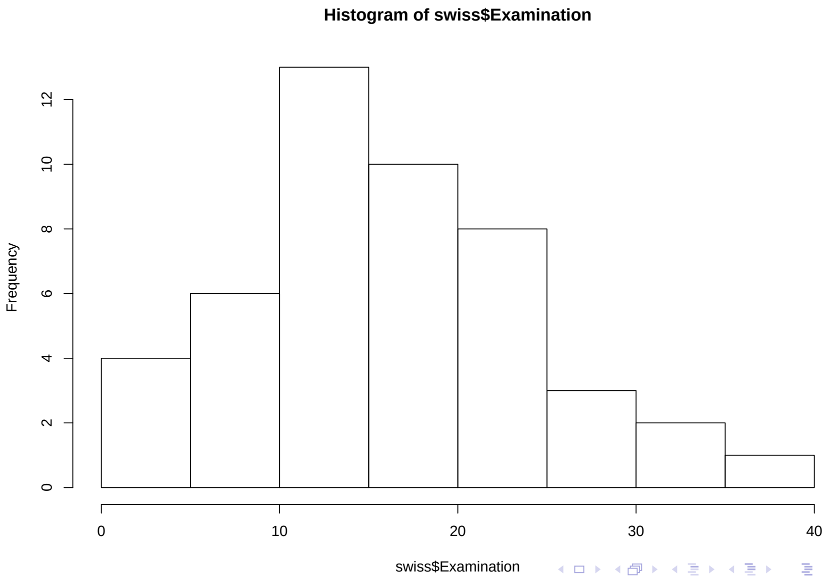
Descriptive Statistics: describe samples

Stats 101: describe samples **distributions** with appropriate measure of

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

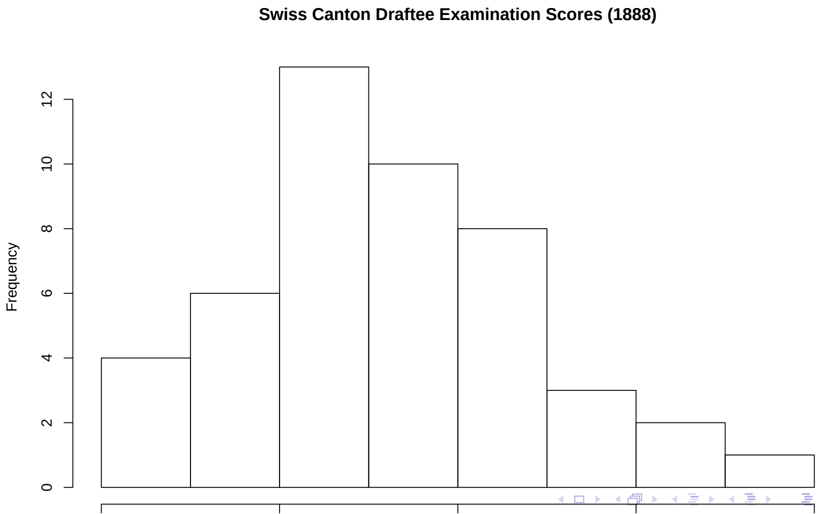
Histograms

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



Histograms: styling

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



Digression: 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] 16.49
```

Finding means

(or use the mean function in base R)

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

```
## [1] 16.49
```

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') %>%
    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

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

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

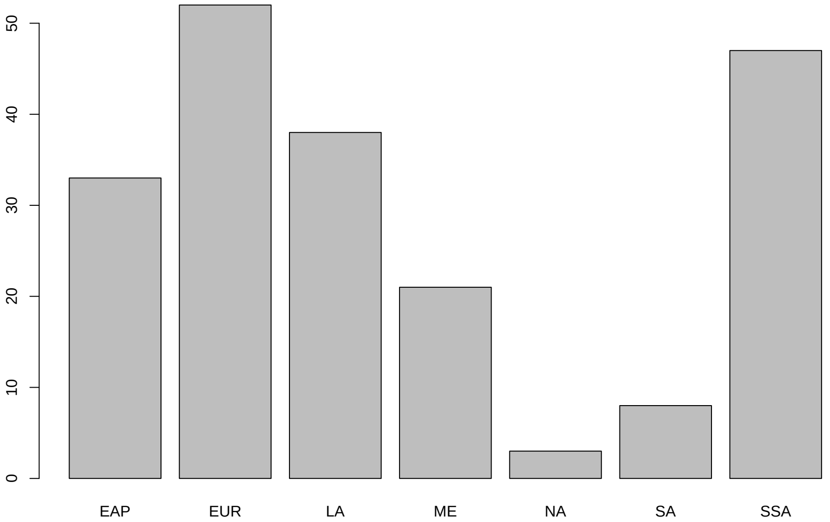
Mode

`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

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



Variation

Range:

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

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

Quartiles:

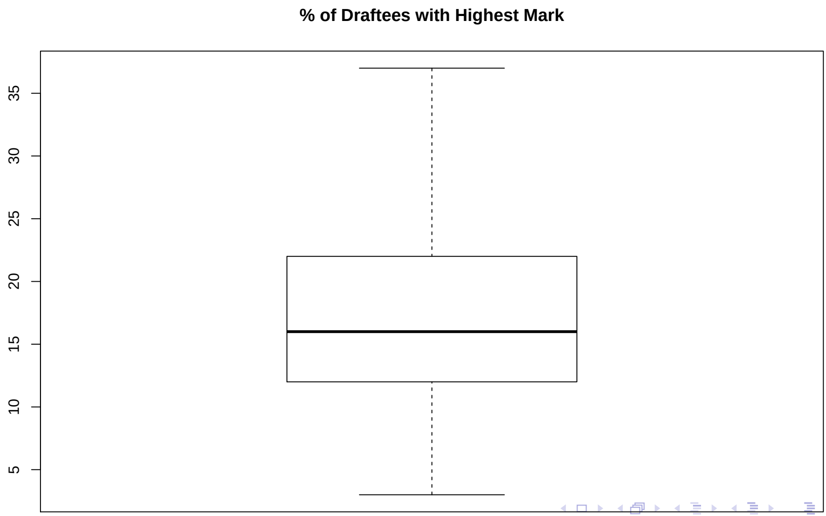
```r
summary(swiss$Examination)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     3.0    12.0    16.0    16.5    22.0    37.0
```

Variation

Boxplots:

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



Variation

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

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

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

Variation: standard deviation

Sum of squared deviations:

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

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

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

Variance (s^2):

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

Standard deviation (s) (in terms of the mean):

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

Variation: Standard Error

The **standard error** of the mean:

If we think of the variation as around a central tendancy as a measure of **unreliability** then we want the measure to **decrease as the sample size goes up**.

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

Variation: Variance and Standard Deviation

Variance:

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

```
## [1] 63.65
```

Standard Deviation:

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

```
## [1] 7.978
```

Variation: Standard Error

Standard Error:

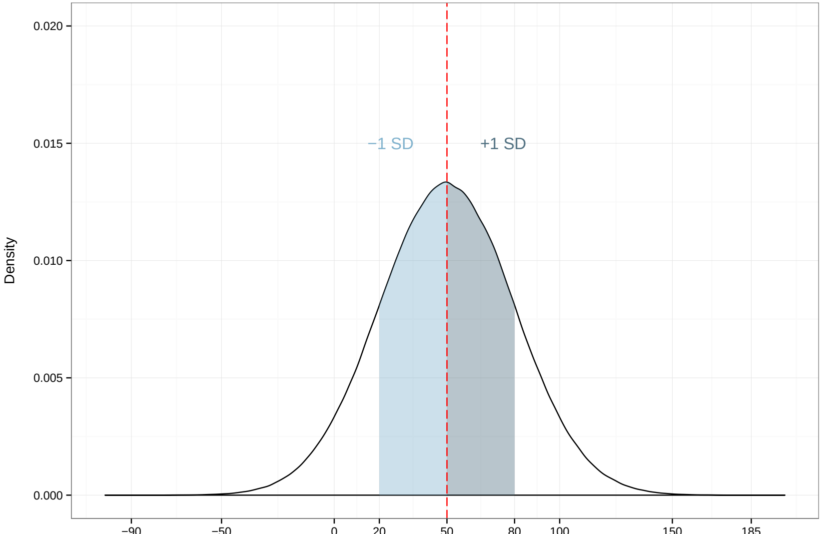
```
sd_error <- function(x) {
    sd(x) / sqrt(length(x))
}

sd_error(swiss$Examination)
```

```
## [1] 1.164
```

Playing with distributions

Simulated normally distributed data with SD of 30 and mean 50

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

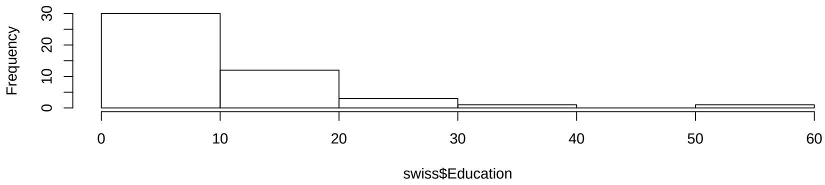


Transform skewed data

Highly skewed data can be transformed to have a normal distribution

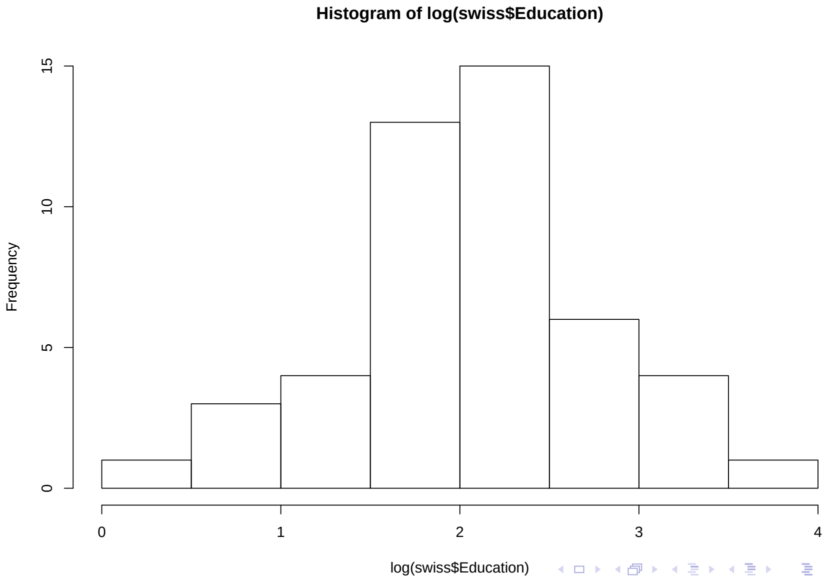
Helps correct two violations of key assumptions: (a) non-linearity and (b) heteroskedasticity.

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



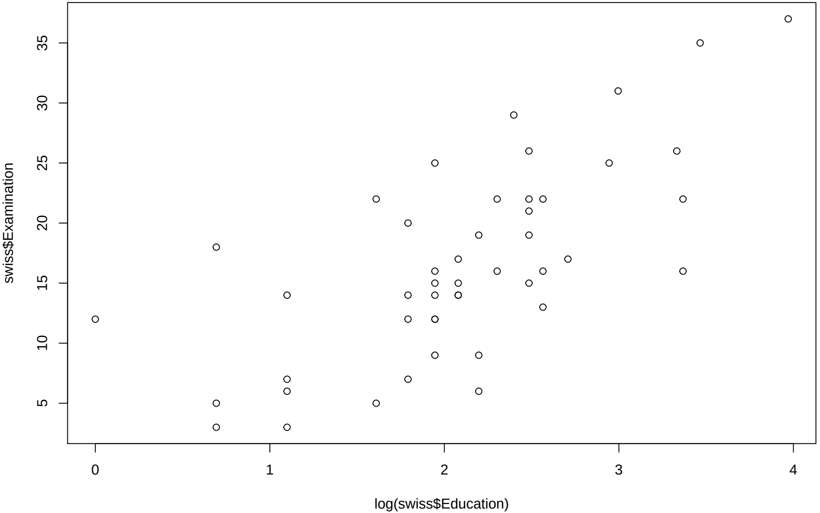
Natural log transformed skewed data

```
hist(log(swiss$Education))
```



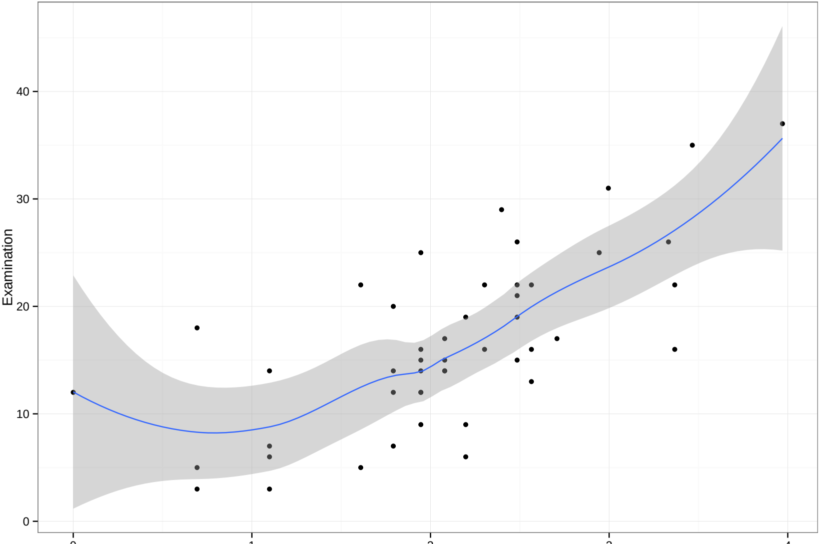
Joint distributions

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



Summarise with loess

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



Programming Hint (1)

Always close!

In R this means closing:

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

Programming Hint (2)

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**.

Programming Hint (2)

Bad

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

```
## [1] 0.05052
```

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

```
## [1] 1.004
```

Programming Hint (2)

Good

```r
rand_sample <- rnorm(1000)

mean(rand_sample)
```

```
## [1] 0.0322
```

```r
sd(rand_sample)
```

```
## [1] 0.954
```

Seminar: Start using R!

- **Access** R data sets
- Explore the data and find ways to **numerically/graphically** describe it
- Find and use R functions that were **not covered** in the lecture for exploring and transforming your data.
- Create **your own function** (what it does is open to you).