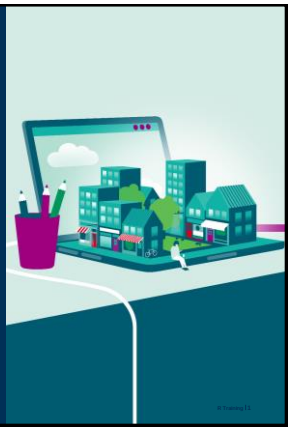
Pearson

R Training for TalentLens R&D

Morgan Strom

25 April 2016



Goals

1. **Automate boring data tasks**
   - Filtering data (creating norms)
   - Merging data (got more than 1 spreadsheet?)
2. **Apply statistics whenever**
   - Descriptive stats
   - Linear models (the usual stuff)
   - Significance testing
   - Predictive models
3. **Basics of R coding**
   - RStudio
   - Programming concepts
   - Useful functions
   - Getting help

Pearson

Day 1
Introduction to R

Day 2
Statistical computation

Introduction

Who am I?

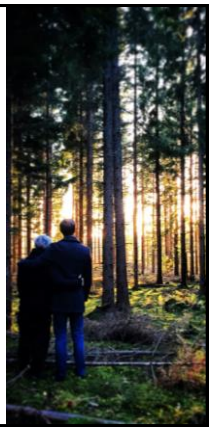
Senior Research Associate at TalentLens

Coding in R since 2012

Developing tests in Concerto since October 2015

Previous jobs:
- Consultant at Cubiks
- Clinical Psychologist at Habilitation Services
- Psych Intern at Pearson Assessments Sweden

Born and raised in the deep forests of Östergötland, Sweden!



Pearson

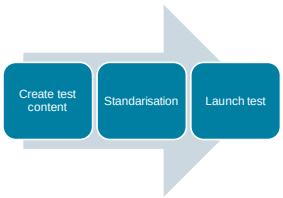
Concerto what?

Problems with previous research platforms:
- Data access
- Content management
- Costly
- Ugly

Advantages of Concerto:
- Flexibility
- Item banking
- Multiple language versions
- Supports adaptive testing (CAT)
- Automated data collection

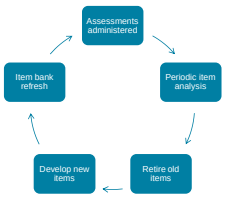
Pearson

Traditional/Print test development



Pearson

Continuous test development

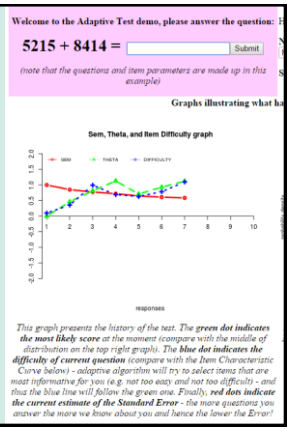


Pearson

Demo

Adaptive test created by Cambridge Psychometrics

http://concerto.e-psychometrics.com/demo/?tid=1116



Pearson

What is R?

R overview

- **Data analysis software**
  - Statistical analysis
  - Data visualization
  - Predictive modelling
- **Programming language**
  - Functional programming
  - (Object oriented programming)
  - #6 most popular language in the world 2015 (IEEE Spectrum)
- **Community**
  - 2 million users worldwide (www.inside-r.org)
  - 7885 packages that extends R functionality (CRAN February 2016)

Lingua Franca of Data Analysis

Pearson

"R is the most powerful and flexible statistical programming language in the world"

Norman Nie, co-founder of SPSS

Pearson

What do you need to know?

- Assignment
  - `<-`
- Data structures
  - Vectors
  - Matrices
  - Lists
  - Data frames
- Subsetting data
  - `[`
  - `$`
- Functions
  - `x <- function(y) {return(y^2)}`
- Logical statements
  - `if(a == b) { … }`
  - `else { … }`
- Loops
  - `for(i in 1:5) {print(i)}`
  - `while(TRUE) {break}`
- I/O
  - `library()`
  - `source()`
  - `read.csv()`
  - `write.csv()`
- Getting help
  - `?`
  - stackoverflow.com
- Libraries
  - dplyr
  - tidyr
  - ggplot2

Pearson 115

Install R & RStudio

R base package:
https://cran.r-project.org/

RStudio Desktop:
https://www.rstudio.com/products/rstudio/download/

Legal stuff

R is released under the GNU General Public License (GPL), version 2.

Same licence as Linux and MySQL, which are both industry standard and used by companies all over the world.

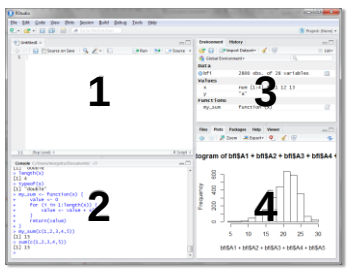
Hornik (2015) "The R FAQ":
https://CRAN.R-project.org/doc/FAQ/R-FAQ.html

"It is the opinion of the R Core Team that one can use R for commercial purposes (e.g., in business or in consulting). The GPL, like all Open Source licenses, permits all and any use of the package. It only restricts distribution of R or of other programs containing code from R."

RStudio

First – this is what RStudio looks like

1. Code editor
2. Console
3. Environment
4. Help/Plots etc.



Data types and assignment

- Vector: one dimension, all values of the same type
  - Numeric: `x <- 5`
  - Character: `y <- c("a", "b")`
  - Factor: `f <- factor(y)`
  - Logical: `i <- c(TRUE, FALSE)`
  - Etc.
- Matrix: two dimensions, all values of the same type
  - Ex: `X <- matrix(1:4, ncol = 2, nrow = 2)`
- List: very flexible structure, can be used to group data in one object
- Data Frame: OUR FAVOURITE! Pretty much a spreadsheet.
  - Ex:
```
df <- data.frame(col1 = 5:8,
                 col2 = rep(f, 2))
```


Export spreadsheets

1. Point R to a folder where you want to save the spreadsheet:
```
setwd("C:/Users/morgstro/Desktop/R Training")
```
2. Use function to write the csv file:
```
write.csv(df, file = "df.csv",
          na = "", row.names = FALSE)
```
3. Open the folder and view the spreadsheet using Excel for example

Import spreadsheets

1. Save spreadsheet as .csv in a folder you like
2. Start R and point to that folder:
```
setwd("C:/Users/morgstro/Desktop/R Training")
```
3. Use function to read the csv file:
```
df2 <- read.csv("df.csv")
```
4. The spreadsheet is now imported and you can start playing around with the data!

Extremely useful functions

- Set working directory: `setwd()`
- See current directory: `getwd()`
- See files in directory: `dir()`
- Get variable names (dataframe): `names(df)`
- See structure of dataframe: `str(df)`
- See class of object: `class(x)`
- See length of a vector: `length(x)`
- See number of rows (dataframe): `nrow(df)`
- See number of columns (dataframe): `ncol(df)`
- Peek at dataframe: `head(df); tail(df)`
- Get help: `?`

Subsetting

- First row: `df[1, ]`
- First column: `df[ ,1]`
- First row of first column: `df[1,1]`
- Column “col2” by name: `df[, “col2”]`
- Shorthand: `df$col2`
- All rows where “col2” is “b”: `df[df$col2 == “b”, ]`

Logical operations

Comparison:

- Equal to: `==`
- Not equal to: `!=`
- Less than: `<`
- Less or equal: `<=`
- Greater than: `>`
- Greater or equal: `>=`
- Group membership: `%in%`
- Is missing: `is.na()`
- Is not missing: `!is.na()`

Booleans:

- Logical AND: `&`
- Logical OR: `|`
- Any TRUE?: `any`
- All TRUE?: `all`

Exercise

- Find row in df where col1 is either 7 or 8, and col2 is not “b”!

Regular expressions

What if you want to "search" for a word in a vector, rather than use exact matches?

For example, looking for "manager" in a vector of job titles `x`.

- Get logical index vector (can be combined with other logicals):
  ```
  grepl("manager", x, ignore.case = TRUE)
  ```
- Get numerical indices:
  ```
  grep("manager", x, ignore.case = TRUE)
  ```

Can be combined using Booleans:

- Get indices for manager OR director:
  ```
  grepl("manager|director", x, ignore.case = TRUE)
  ```
- Get indices for manager AND senior:
  ```
  grepl("manager&senior", x, ignore.case = TRUE)
  ```


Control functions

If-else statements are used in programming to do "branching", where the program adapts to the user's choices.

It basically asks a logical question, where the response can be either TRUE or FALSE, and selects an operation based on the response.

The basic syntax works like this:

```
if ("a" == 1) {
        #This code will not be executed
} else {
        #This code will be executed
}
```


Control functions

If-else statements are also used in statistics for "step functions":

$$f(x|c) = \begin{cases} 1 & \text{if } x \geq c \\ 0 & \text{if } x < c \end{cases}$$

This statement below returns 1 if x is larger than or equal to a cut-off value c, and 0 otherwise:

```
f <- function(x, c) {
    if (x >= c) {
        return(1)
    } else {
        return(0)
    }
}
```

For loop

"Loop over a vector" = Do the same operation once for every value in a vector

Very useful for simple repetitive tasks!

Syntax:

```
for (i in 1:n) {
    # i starts at 1
    # code here is executed
    # i is increased by 1
    # and the code is executed again
    # until i >= n
}
```


For loop

Example: Summation

Mathematical notation:

$$\sum_{i=1}^{n} x_i$$

R code with for loop:

```
for (i in 1:n) {
    s <- s + x[i]
}
```


Function

Simple function to calculate the sum of a vector of values, x

```
my_sum <- function(x) {

        s <- 0
        n <- length(x)

        for (i in 1:n) {
                s <- s + x[i]
        }

        return(s)

}
```

Quick exercises

- Write the function `my_sum()` in your text editor, mark all the code and press CTRL + Enter on your keyboard. This will execute the code in the console.
- Concatenate the numbers 4, 5, 8 and 11 to a numeric vector using `c()` and calculate their sum using the function `my_sum()`.
- Compare the results to using the built-in function `sum()`

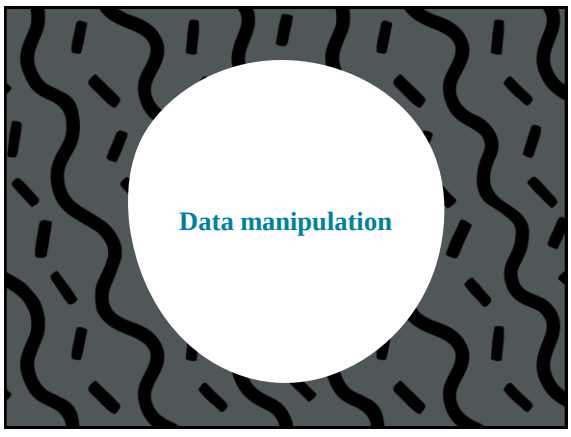
More on R

Very short introduction to R:
https://cran.r-project.org/doc/contrib/Torfs+Brauer-Short-R-Intro.pdf

R Programming online course:
https://www.coursera.org/learn/r-programming

Swirl interactive learning library:
install.packages("swirl")

Excellent books:
Discovering Statistics using R, Andy Field
The Art of R Programming, Norman Matloff
Advanced R, Hadley Wickham



Tidy data

One row per observation
One column per feature/variable

Long vs Wide format

Use tidyr to reshape data
- From wide to long, (when you want to do group summaries or make grouped plots): `gather()`
- From long to wide: `spread()`

Cheat sheet:
https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

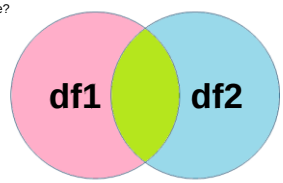
Case: merging spreadsheets

We have two spreadsheets with results from Watson-Glaser.

```
df1 <- read.csv("wg1.csv", stringsAsFactors = FALSE)
df2 <- read.csv("wg2.csv", stringsAsFactors = FALSE)
```

The columns are identical.

How do we merge these into one?



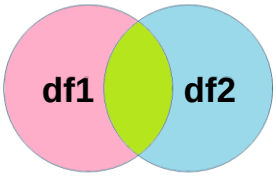
Case: merging spreadsheets

Rows unique to df1: Pink
Rows unique to df2: Blue
Rows shared by df1 & df2: Green

Rows in df1: Pink & Green
Rows in df2: Green & Blue
Everything: Pink & Green & Blue

Use dplyr package!!

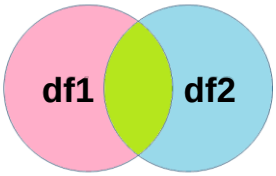
```
install.packages("dplyr")
library(dplyr)
```



Case: merging spreadsheets

Pink: `df3 <- dplyr::setdiff(df1, df2)`
Blue: `df3 <- dplyr::setdiff(df2, df1)`
Green: `df3 <- dplyr::intersect(df1, df2)`

Pink & Green: `df3 <- df1`
Green & Blue: `df3 <- df2`
Pink & Green & Blue: `df3 <- dplyr::union(df1, df2)`



Case: joining spreadsheets

If you want to join 2 spreadsheet with DIFFERENT columns, you should instead use JOIN functions.

Make sure that there is a unique ID to identify each case in both spreadsheets!

Example: Athena validity study

```
athena <- read.csv("athena.csv")
cubiks <- read.csv("cubiks.csv")
```


Case: joining spreadsheets

Athena has more cases than Cubiks.

We can join the data in three ways

1. Full join: keep all cases, matched by the ID column
   `full_join(athena, cubiks, by = c("ID" = "ID")`
2. Inner join: keep only cases that appear in both `athena` and `cubiks`
   `inner_join(athena, cubiks, by = c("ID" = "ID")`
3. Left join: keep all cases in `cubiks`, add matching columns from `athena`. In this case, removes missing data.
   ```
   left_join(cubiks, athena,
             by = c("ID" = "ID", "Group" = "Group")
   ```

Case: data cleaning

My process for data preparation

1. Import data
2. Play around & explore!
3. Get serious, you're being paid for this work
4. Clean up variable names, data classes & factor level names
5. Exploration loop:
   1. Summarise data
   2. Make plots
   3. Amend obvious errors

Outliers & data wierdness

Graphs!

- Histogram: `hist(x)`
- Box plot: `boxplot(x)` or `boxplot(x ~ f, df)`
- Scatterplot: `plot(x,y)`
- Pairwise scatterplots: `psych::pairs.panels(X)`

Summaries!

- `summary(df)`
- `psych::describe(x)`
- `dplyr::summarize(x, ...)`

Case: creating norms

My process for norm creation

1. Make summaries of
   - Occupation
   - Education level
   - Position Type/Level
2. Use `filter` to create norms

Case: creating norms

Example of filtering for graduate norm (rough but efficient):

```
graduates <- df3 %>%
  filter(!is.na(wgcta.theta.score)) %>% #Filter out NA scores
  filter(TimeTaken > 0 & TimeTaken <= 2000) %>% #Filter out wierdness
  filter(Education %in% c("Doctoral", "Master", "Bachelor"))
```

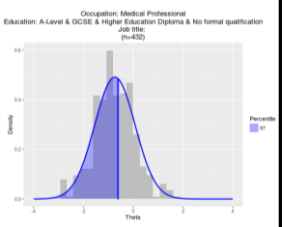
Pearson

143

Demo: shiny web app

Interactive norms for Watson-Glaser UK

https://morgan.shinyapps.io/wg_norm



Pearson

44

Exercises

1. Store all integers between 1 and 100 in a vector `x`. Write a for-loop with if-statements that replaces all numbers divisible by 3 with missing values, `NA`. (Hint: use the modulus operator `%%`)
2. Optional: Can you find a faster way to perform the same task? i.e. without a for-loop
3. Make a function that takes a character string as an argument and prints the letters in a randomised order to the console. (Hint: use `strsplit()` to split the string into letters)

Pearson

145