An Introduction to Data Wrangling and Analysis with R

Rod Alence
International Relations and e-Science
University of the Witwatersrand

Workshop
Introduction to Emerging Methodologies in Social Science Research (IEMeSSR)
10–14 May 2021

Aims of my sessions

My two sessions aim to introduce R as software for “data science” – understood to include:

1. Data “wrangling”
    - Importing (possibly “messy”) data into R – from text files, spreadsheets, or other statistical programs (SPSS, Stata, SAS, etc.); and
    - “Tidying” data for analysis – getting the data into a rectangular (one-observation-per-row, one-variable-per-column) format.
2. Data analysis
    - Transforming data (mathematical calculations and recoding);
    - Visualizing data (graphics); and
    - Modeling data (statistics).

Approach of my sessions

“Learning by doing” (as much as possible)

- Start with “traditional” slides/lecture format;
- Shift to “live coding” in R/RStudio as soon as possible (technology permitting);
- Recommend options for offline self-study.

Programming within R

- Give “non-exclusive emphasis” to Base R over Tidyverse.

  (If you have no idea what that means, don’t worry!)

Expectations for my sessions (beyond attendance!)

What I do not expect

- Prior experience programming in R (a bonus if you have it!);
- Mathematical expertise beyond high-school algebra.

What I do expect

- Some prior experience using some statistical software (e.g., SPSS or Stata);
- Basic familiarity with descriptive statistics and statistical models (not beyond least-squares linear regression).

Outline

On learning R

Some R basics

Data “wrangling” with R: the UNDP Human Development Index

Model and visualize the data

Self-study in R/RStudio with `swirl`

Wrapping up

Outline

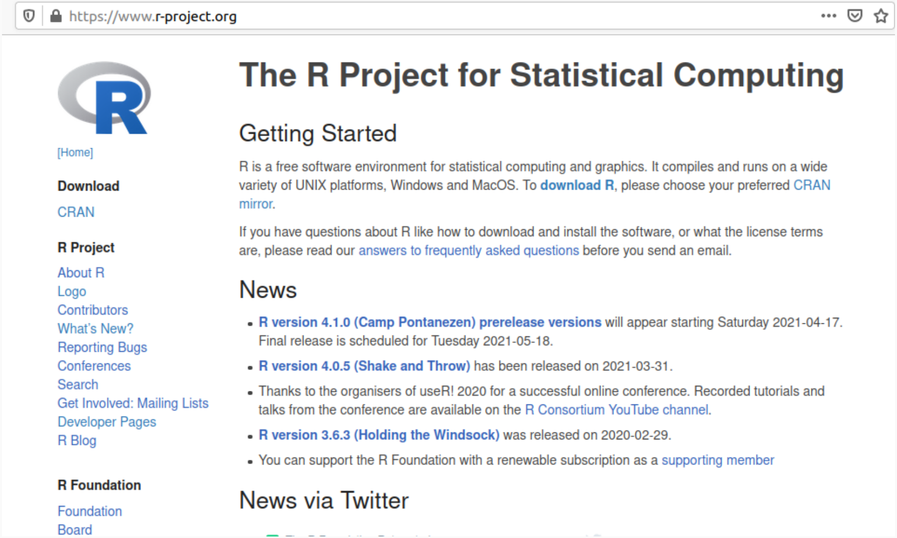
What is R?

- Free, open-source statistical software that runs on all major operating systems;
- Created in the mid-1990s at the University of Auckland, New Zealand, by Ross Ihaka and Robert Gentleman as an implementation of the S programming language;
- Now maintained by a volunteer Core Development Team, which releases an updated version about twice a year;
- New and updated add-on “packages” appear weekly – more than 17,000 now available;
- For more information: http://www.r-project.org

Why R?

- Is probably the most powerful software for statistical analysis;
- Has the best graphics capabilities;
- Its package system is “going viral” (in a good way);
- Is “free” – as intellectual property and in price.

The R Project website



R package “task views”

https://cran.r-project.org/web/views/

CRAN Task Views

CRAN task views aim to provide some guidance which packages on CRAN are relevant for tasks related to
automatically installed using the ctv package. The views are intended to have a sharp focus so that it is sui
not meant to endorse the "best" packages for a given task.

- To automatically install the views, the ctv package needs to be installed, e.g., via
  `install.packages("ctv")`
  and then the views can be installed via `install.views` or `update.views` (where the latter only installs those
  `ctv::install.views("Econometrics")`
  `ctv::update.views("Econometrics")`
- The task views are maintained by volunteers. You can help them by suggesting packages that should l
  individual task view pages.
- For general concerns regarding task views contact the ctv package maintainer.

Topics

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
Databases	Databases with R
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Econometrics
Environmetrics	Analysis of Ecological and Environmental Data

https://twitter.com/statsepi/status/795574223439876100

Does R have a “steep learning curve”?

The two most challenging things about R

1. It is entirely command (“expression”) based – you type commands, and R executes them (no “point-and-click” menus).
2. It allows multiple (unlimited) data “objects” in a session simultaneously.

But – these features are essential to R’s strengths

- No menu system could ever keep up with software as powerful and dynamic as R.
- Allowing multiple objects is essential to a programming language in which the output of nearly any command can be the input of another.

And… RStudio makes learning and executing R command syntax

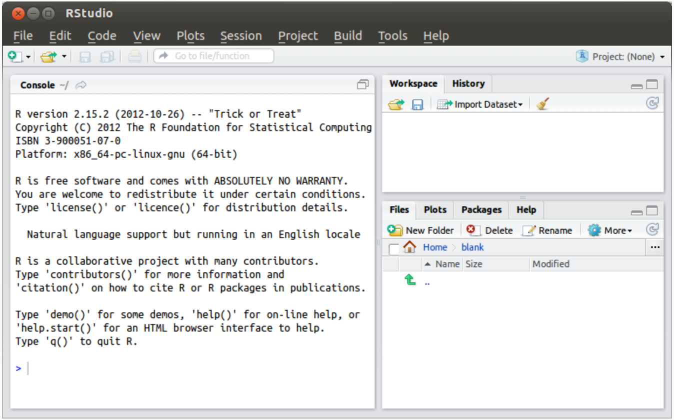
What is RStudio?

- An “integrated development environment” (IDE) for R (but not a “point-and-click” interface to R commands);
- Launched in 2011;
- Free and open source;
- Available for all major operating systems (Windows, MacOS, and Linux);
- For more information: http://www.rstudio.org

RStudio at first start-up

Three windows

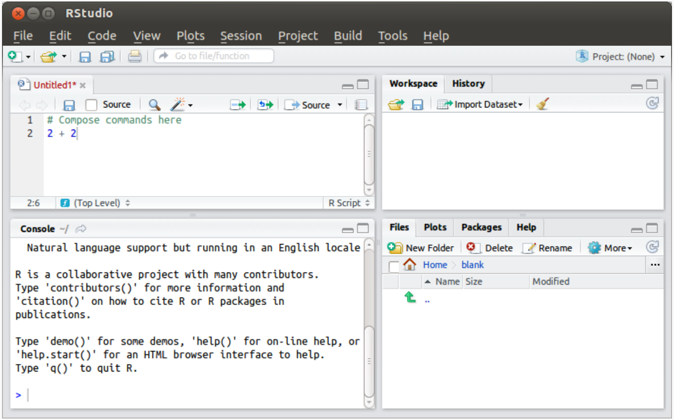
R console occupies full left side



RStudio with editor window open (the usual way)

Four windows

Left side split between editor (top) and R console (bottom)



Outline

Data structures: vectors

The most basic data structure in R is the vector – which is just a fancy word for "a list of things in a particular order."

- Even a single number is a vector to R – it is a vector that happens to contain only one thing.
- When a vector holds data about units, like a column in a spreadsheet, a vector is synonymous with what is often called a "variable."
- Each vector has two intrinsic structural attributes:

  Length

  How many things (elements) does it contain?

  Mode

  What kinds of things does it contain – e.g., numeric values, typographic characters?

Data structures: data frames

A data frame is a rectangular, spreadsheet-like data structure – typically organized with “observations” in rows and “variables” in columns.

Data frames

- May contain column vectors of any class; but
- Must contain column vectors of the same length.

The collection of packages known as the “Tidyverse” often use a special type of data frame called the tibble, which

- Has the same basic structure as the “traditional” data frame, with a few distinct features; and
- Can easily be converted to the “traditional” data frame.

Assigning names to data

Any non-trivial "data" is assigned a name for further use, using the "backward-arrow" operator.

For example, we can "stick together" some numbers as a vector using c (for combine) and assign it a name, like `some_numbers`:

```
some_numbers <- c(1, 2, 3, 4, 5)
```

And we can do the same with some letters (in quotation marks):

```
some_letters <- c("e", "d", "c", "b", "a")
```

Assignment is "silent" – but we can check the objects' contents by typing its name and pressing return.

```
some_numbers
# [1] 1 2 3 4 5

some_letters
# [1] "e" "d" "c" "b" "a"
```

Combining vectors in a data frame

Because the two vectors assigned in the previous slide are the same length, we can stick them together side-by-side in a data frame using `data.frame`.

```
boring_df <- data.frame(some_numbers, some_letters)
```

And to view the data frame, enter its name.

```
boring_df
#   some_numbers some_letters
# 1            1            e
# 2            2            d
# 3            3            c
# 4            4            b
# 5            5            a
```

Notes on naming

R's rules about names

- May contain lower-case and upper-case letters, numbers, dots (`.`), and underscores (`_`).
- May not start with numbers – and they should almost always start with letters.
- Are case-sensitive – lower-case and upper-case versions of the same letter are treated as entirely different characters.
- Overwrite any existing object with the same name.

Common sense about names

- Should be concise (to avoid too much typing).
- Should be Informative (to clarify content).

Numeric indexing

Elements of data structures can be accessed by position using numeric square-bracket indexes.

Vectors

```
some_letters
# [1] "e" "d" "c" "b" "a"

some_letters[4]  # get the fourth element
# [1] "b"
```

Data frames

```
boring_df
#   some_numbers some_letters
# 1            1            e
# 2            2            d
# 3            3            c
# 4            4            b
# 5            5            a

boring_df[3, 2]  # row-by-col (third row, second col)
# [1] "c"
```

Indexing columns (variables) in data frames

Three common ways to select a column (variable) in a data frame:

1. By numeric position

```
boring_df[ , 2]  # get the second col (all rows)
# [1] "e" "d" "c" "b" "a"
```

2. By column name

```
boring_df[ , "some_letters"]  # get the col called "some_letters"
# [1] "e" "d" "c" "b" "a"
```

3. Dollar-sign (list) notation

```
boring_df$some_letters
# [1] "e" "d" "c" "b" "a"
```

Indexing rows (observations) in data frames

Two common ways to select rows in a data frame:

1. By numeric position

```
boring_df[c(1, 3), ]  # get the first and third rows (all cols)
#   some_numbers some_letters
# 1            1            e
# 3            3            c
```

2. By logical expression

```
## Get rows in which the logical expression holds
boring_df[boring_df$some_numbers > 3, ]
#   some_numbers some_letters
# 4            4            b
# 5            5            a
```

Functions in R

Functions are what "do things" in R – if data objects are like nouns, functions are the verbs.

Function syntax

To use a function:

1. Type its name (exactly, remember case-sensitivity),
2. Followed immediately by parentheses (curved brackets),
3. Insert any inputs ("arguments") inside the parentheses, separated by commas.

Often the reason for using a function is to generate output which is immediately assigned to an object.

An example using functions

Which two functions are used here?

```
marks <- c(78, 56, 91, 88, NA, 62, 67)  # one student was absent
class_ave <- mean(marks, na.rm=TRUE)
class_ave
# [1] 73.66667
```

Each function has a help page, which explains what the function does and what inputs it takes.

Typing a question mark followed by a function name calls up the help page. To find out what the `na.rm=TRUE` is about, try entering `?mean` in the R console.

Reading data into R

R has functions for reading in data in various formats, for example:

- Comma- or tab-delimited text: `read.csv` and `read.delim` in Base R, or `read_csv` and `read_tsv` in the readr package
- Spreadsheets (.xls, .xlsx): `read_excel` in the readxl package;
- Stata (.dta): `read.dta` (Stata 5-12) in the foreign package, `read.dta13` (Stata 13 onwards) in the readstata13 package, or `read_dta` (all versions) in the haven package;
- SPSS (.sav): `read.spss` in the foreign package, or `read_spss` in the haven package.

Outline

Setting up

Install two packages

The installation only needs to be run once – I use the console.

```
install.packages(c("readxl", "tidyverse"))
```

Download the data spreadsheet

The download only needs to be run once (if the data set is static).

```
## Break up the url for convenience, because it is long
site_url <- "http://hdr.undp.org/"
path_url <- "sites/default/files/"
fn_url   <- "2020_statistical_annex_table_1.xlsx"

## Paste the parts together
link_url <- paste0(site_url, path_url, fn_url)

## Download using the full url
download.file(url=link_url,
              method="curl",
              destfile="hdi2020.xlsx")
```

Read in the spreadsheet (and have a quick look)

```
library(readxl)
HDI <- read_excel("hdi2020.xlsx",
                  range = "B8:K200",   # cells to read
                  col_names = FALSE,   # column names not read
                  na = c("", ".."))    # missing value strings
dim(HDI)
# [1] 193  10
```

```
head(HDI)
# # A tibble: 6 x 10
#   ...1        ...2 ...3   ...4 ...5   ...6 ...7   ...8 ...9
#   <chr>      <dbl> <lgl> <dbl> <chr> <dbl> <chr> <dbl> <chr>
# 1 VERY HIG~ NA     NA     NA   <NA>   NA   <NA>   NA   <NA>
# 2 Norway     0.957 NA     82.4 <NA>   18.1 b      12.9 <NA>
# 3 Ireland    0.955 NA     82.3 <NA>   18.7 b      12.7 <NA>
# 4 Switzerl~  0.955 NA     83.8 <NA>   16.3 <NA>   13.4 <NA>
# 5 Hong Kon~  0.949 NA     84.9 <NA>   16.9 <NA>   12.3 <NA>
# 6 Iceland    0.949 NA     83.0 <NA>   19.1 b      12.8 c
# # ... with 1 more variable: ...10 <dbl>
```

The structure of the data frame

The data frame is still a bit “messy.”

```
str(HDI)
# tibble[,10] [193 x 10] (S3: tbl_df/tbl/data.frame)
#  $ ...1 : chr [1:193] "VERY HIGH HUMAN DEVELOPMENT" "Norway" "Ireland
#  $ ...2 : num [1:193] NA 0.957 0.955 0.955 0.949 0.949 0.947 0.945 0.
#  $ ...3 : logi [1:193] NA NA NA NA NA NA ...
#  $ ...4 : num [1:193] NA 82.4 82.3 83.8 84.9 ...
#  $ ...5 : chr [1:193] NA NA NA NA ...
#  $ ...6 : num [1:193] NA 18.1 18.7 16.3 16.9 ...
#  $ ...7 : chr [1:193] NA "b" "b" NA ...
#  $ ...8 : num [1:193] NA 12.9 12.7 13.4 12.3 ...
#  $ ...9 : chr [1:193] NA NA NA NA ...
#  $ ...10: num [1:193] NA 66494 68371 69394 62985 ...
```

Remove unneeded columns

Which columns are not needed?

```
head(HDI)
# # A tibble: 6 x 10
#   ...1         ...2  ...3   ...4  ...5   ...6  ...7   ...8  ...9
#   <chr>       <dbl> <lgl> <dbl> <chr> <dbl> <chr> <dbl> <chr>
# 1 VERY HIG~ NA      NA      NA    <NA>   NA    <NA>   NA    <NA>
# 2 Norway      0.957 NA      82.4 <NA>   18.1 b       12.9 <NA>
# 3 Ireland     0.955 NA      82.3 <NA>   18.7 b       12.7 <NA>
# 4 Switzerl~   0.955 NA      83.8 <NA>   16.3 <NA>    13.4 <NA>
# 5 Hong Kon~   0.949 NA      84.9 <NA>   16.9 <NA>    12.3 <NA>
# 6 Iceland     0.949 NA      83.0 <NA>   19.1 b       12.8 c
# # ... with 1 more variable: ...10 <dbl>
```

Remove the “skinny” footnote columns.

```
## Use negative column indexes to remove columns
HDI <- HDI[ , -c(3, 5, 7, 9)]  # negative column index to remove

## Tidyverse alternative (dplyr package) (NOT RUN)
## HDI <- dplyr::select(HDI, -c(3, 5, 7, 9))
```

Add meaningful column names

Add meaningful (informative and concise) names by “assigning into” the data frame’s `colnames`:

```
## Old column names
colnames(HDI)
# [1] "...1"  "...2"  "...4"  "...6"  "...8"  "...10"

## Assign new column names
colnames(HDI) <- c("country", "hdi", "life_exp",
                   "school_exp", "school_mean", "gni_pc")
head(HDI)
# # A tibble: 6 x 6
#   country          hdi life_exp school_exp school_mean gni_pc
#   <chr>          <dbl>    <dbl>      <dbl>       <dbl>  <dbl>
# 1 VERY HIGH H~ NA          NA         NA          NA       NA
# 2 Norway         0.957     82.4       18.1        12.9 66494.
# 3 Ireland        0.955     82.3       18.7        12.7 68371.
# 4 Switzerland    0.955     83.8       16.3        13.4 69394.
# 5 Hong Kong, ~   0.949     84.9       16.9        12.3 62985.
# 6 Iceland        0.949     83.0       19.1        12.8 54682.
```

Remove unneeded rows

Which rows are not needed?

```
head(HDI, n=2)
# # A tibble: 2 x 6
#   country          hdi life_exp school_exp school_mean gni_pc
#   <chr>          <dbl>    <dbl>      <dbl>       <dbl>  <dbl>
# 1 VERY HIGH H~ NA          NA         NA          NA       NA
# 2 Norway         0.957     82.4       18.1        12.9 66494.

HDI[HDI$country == "MEDIUM HUMAN DEVELOPMENT", ]
# # A tibble: 1 x 6
#   country          hdi life_exp school_exp school_mean gni_pc
#   <chr>          <dbl>    <dbl>      <dbl>       <dbl>  <dbl>
# 1 MEDIUM HUMAN~     NA       NA         NA          NA     NA
```

Remove rows with no numeric data (e.g., in the `hdi` column).

```
HDI <- HDI[! is.na(HDI$hdi), ]

## Tidyverse alternative (NOT RUN)
## HDI <- dplyr::filter(HDI, ! is.na(HDI$hdi))
```

Check the data structure

Things to check:

- Dimensions (rows by columns);
- Column names;
- Object "classes" (e.g., character vs. numeric).

```
str(HDI)
# tibble[,6] [189 x 6] (S3: tbl_df/tbl/data.frame)
#  $ country    : chr [1:189] "Norway" "Ireland" "Switzerland" "Hong Ko
#  $ hdi        : num [1:189] 0.957 0.955 0.955 0.949 0.949 0.947 0.945
#  $ life_exp   : num [1:189] 82.4 82.3 83.8 84.9 83  ...
#  $ school_exp : num [1:189] 18.1 18.7 16.3 16.9 19.1  ...
#  $ school_mean: num [1:189] 12.9 12.7 13.4 12.3 12.8  ...
#  $ gni_pc     : num [1:189] 66494 68371 69394 62985 54682  ...
```

Check the data summary

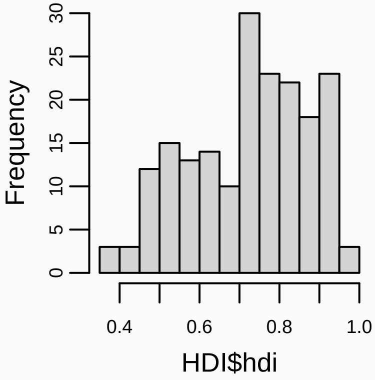
Things to check:

- Descriptive statistics;
- Missing values (NA frequencies are reported for each variable).

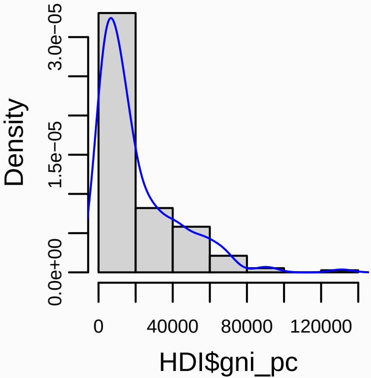
```
summary(HDI)
#    country               hdi              life_exp
#  Length:189         Min.   :0.3940   Min.   :53.28
#  Class :character   1st Qu.:0.6020   1st Qu.:67.44
#  Mode  :character   Median :0.7400   Median :74.05
#                     Mean   :0.7224   Mean   :72.71
#                     3rd Qu.:0.8290   3rd Qu.:77.91
#                     Max.   :0.9570   Max.   :84.86
#    school_exp       school_mean         gni_pc
#  Min.   : 5.005   Min.   : 1.644   Min.   :   753.9
#  1st Qu.:11.431   1st Qu.: 6.437   1st Qu.:  4910.2
#  Median :13.188   Median : 9.032   Median : 12707.4
#  Mean   :13.325   Mean   : 8.728   Mean   : 20219.7
#  3rd Qu.:15.227   3rd Qu.:11.326   3rd Qu.: 29497.2
#  Max.   :21.954   Max.   :14.152   Max.   :131031.6
```

Run a few histograms?

```r
hist(HDI$hdi,
     main="")
```




```r
hist(HDI$gni_pc,
     freq=FALSE,
     main="")
lines(density(HDI$gni_pc),
      col="blue")
```



Step 1: Calcuate the dimension indexes

The HDI is based on three “dimension” indexes:

1. Health

   Based on life expectancy;

2. Education

   Based on expected *and* mean years of schooling;

3. Income

   Based on gross national income per capita.

Calculating the dimension indexes

The dimension indexes are normalized using *modified min-max scales* with "goalpost" values:

$$\text{Dimension index} = \frac{\text{actual value} - \text{lower goalpost}}{\text{upper goalpost} - \text{lower goalpost}}$$

If the actual value falls between the goalposts, the dimension index value will fall between zero and one.

The goalposts

Dimension	Indicator	Lower	Upper
Health	Life expectancy (yrs)	20	85
Education	Expected schooling (yrs)	0	18
	Mean schooling (yrs)	0	15
Income	GNI per capita (2017 PPP$)	100	75,000

Calculate the health index

Calculate the index using the "goalpost" formula, and assign it to the variable `health_index` in the HDI data frame.

```
HDI$health_index <- (HDI$life_exp - 20) / (85 - 20)
```

Check that the values are consistent with expectations.

```
summary(HDI$health_index)
#    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#  0.5120  0.7298  0.8315  0.8109  0.8909  0.9978

summary(HDI$life_exp)
#    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#   53.28   67.44   74.05   72.71   77.91   84.86
```

Calculate the education index

There are two education indicators – the dimension index is the arithmetic mean of the two "goalposted" schooling indicators.

```
HDI$school_exp_index  <- (HDI$school_exp - 0) / (18 - 0)
HDI$school_mean_index <- (HDI$school_mean - 0) / (15 - 0)
HDI$educ_index <- (HDI$school_exp_index + HDI$school_mean_index) / 2

## Check summary
summary(HDI[, c("school_exp_index", "school_mean_index", "educ_index")]
#  school_exp_index school_mean_index   educ_index
#  Min.   :0.2781   Min.   :0.1096    Min.   :0.2491
#  1st Qu.:0.6351   1st Qu.:0.4291    1st Qu.:0.5295
#  Median :0.7327   Median :0.6021    Median :0.6823
#  Mean   :0.7403   Mean   :0.5819    Mean   :0.6611
#  3rd Qu.:0.8459   3rd Qu.:0.7551    3rd Qu.:0.7929
#  Max.   :1.2197   Max.   :0.9434    Max.   :1.0340
```

OOPS – what is the problem!

“Enforce” the upper goalpost for `school_exp_index`

Index values must not exceed 1.0, even if the indicator exceeds the upper goalpost.

One way to fix this is to replace values of `school_exp_index` greater than 1.0 with values of exactly 1.0.

```
## Square brackets identify values to "reassign"
HDI[HDI$school_exp_index > 1, "school_exp_index"] <- 1

## Recalculate the dimension index
HDI$educ_index <- (HDI$school_exp_index + HDI$school_mean_index) / 2

## Check summary
summary(HDI[, c("school_exp_index", "school_mean_index", "educ_index")])
#  school_exp_index school_mean_index   educ_index
#  Min.   :0.2781   Min.   :0.1096    Min.   :0.2491
#  1st Qu.:0.6351   1st Qu.:0.4291    1st Qu.:0.5295
#  Median :0.7327   Median :0.6021    Median :0.6823
#  Mean   :0.7366   Mean   :0.5819    Mean   :0.6592
#  3rd Qu.:0.8459   3rd Qu.:0.7551    3rd Qu.:0.7929
#  Max.   :1.0000   Max.   :0.9434    Max.   :0.9433
```

Calculate the income index

Calculate the income index, noting that it is based on the natural logarithm of GNI per capita.

The `log` function in R gives the natural logarithm by default.

```
HDI$income_index <- (log(HDI$gni_pc) - log(100)) /
                    (log(75000) - log(100))
summary(HDI[, c("gni_pc", "income_index")])
#      gni_pc          income_index
#  Min.   :   753.9   Min.   :0.3051
#  1st Qu.:  4910.2   1st Qu.:0.5882
#  Median : 12707.4   Median :0.7318
#  Mean   : 20219.7   Mean   :0.7152
#  3rd Qu.: 29497.2   3rd Qu.:0.8590
#  Max.   :131031.6   Max.   :1.0843

## Enforce the upper goalpost
HDI[HDI$income_index > 1, "income_index"] <- 1
summary(HDI$income_index)
#    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#  0.3051  0.5882  0.7318  0.7145  0.8590  1.0000
```

Step 2: Combine the dimension indexes into "our" HDI

The HDI is the geometric mean of the three dimension indexes:

$$\text{HDI} = (\text{health} \times \text{education} \times \text{income})^{1/3}$$

(Raising to the one-third power is the same as taking a cube root.)

```
HDI$our_hdi <- (HDI$health_index * HDI$educ_index *
                HDI$income_index) ^ (1/3)
summary(HDI$our_hdi)
#    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#  0.3937  0.6019  0.7397  0.7225  0.8289  0.9570
```

Compare “our” HDI with the UNDP’s

```
summary(HDI[, c("hdi", "our_hdi")])
#       hdi              our_hdi
#  Min.   :0.3940   Min.   :0.3937
#  1st Qu.:0.6020   1st Qu.:0.6019
#  Median :0.7400   Median :0.7397
#  Mean   :0.7224   Mean   :0.7225
#  3rd Qu.:0.8290   3rd Qu.:0.8289
#  Max.   :0.9570   Max.   :0.9570

## Differences between HDI values
HDI$hdi_diff <- HDI$hdi - HDI$our_hdi
summary(HDI$hdi_diff)
#       Min.    1st Qu.     Median       Mean    3rd Qu.
# -4.992e-04 -3.275e-04 -6.067e-05 -4.642e-05  2.258e-04
#       Max.
#  4.827e-04
```

Do we have a problem?

The UNDP rounds HDI to the third decimal place!

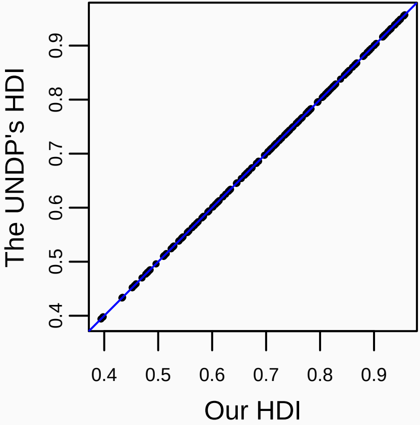
What if we do the same?

```
HDI$our_hdi <- round(HDI$our_hdi, digits=3)

## Recompare
HDI$hdi_diff <- HDI$hdi - HDI$our_hdi
summary(HDI$hdi_diff)
#    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#       0       0       0       0       0       0
```

We have successfully wrangled the UNDP data in R and reproduced the HDI from the source data!

Plotting our success

```
plot(x=HDI$our_hdi, y=HDI$hdi,
     xlab="Our HDI",
     ylab="The UNDP's HDI",
     pch=16, cex=0.5)
abline(a=0, b=1, col="blue")  # x=y line
```



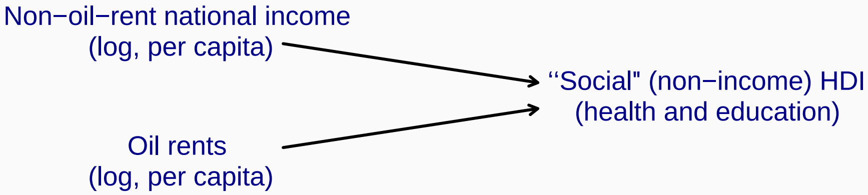
Outline

A regression model: income vs. non-income HDI

HDI was developed as an alternative to using national income (per capita) as a “proxy” measure of human development.

A concern beginning in the 1970s was that oil-exporting countries could have high income per capita but low human development.

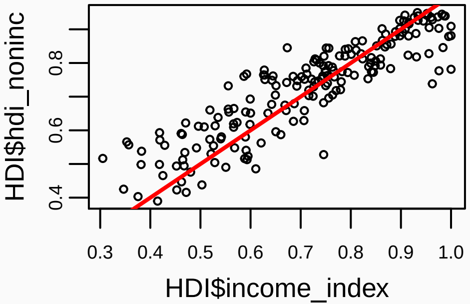
A (too?) simple model



Calculate "non-income" HDI

Defining "non-income" HDI as the geometric mean of the health and education indexes, we can use our existing data to calculate it:

```
HDI$hdi_noninc <- sqrt(HDI$health_index * HDI$educ_index)
summary(HDI$hdi_noninc)
#    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#  0.3897  0.6176  0.7585  0.7282  0.8382  0.9497

plot(x=HDI$income_index, y=HDI$hdi_noninc, cex=0.5)
abline(a=0, b=1, col="red", lwd=2)  # x=y line
```



Load data on oil rents

The HDI data set does not have oil rents. I downloaded some from the World Bank's *World Development Indicators*:

https://databank.worldbank.org/source/world-development-indicators

To save time, I put the data in an R data file, so that we only need to `load` the data (from the project folder).

```
load("oil_rents.RData")
summary(oil_rents)
#    isocode          oil_rents_pct
#  Length:217         Min.   : 0.0000
#  Class :character   1st Qu.: 0.0000
#  Mode  :character   Median : 0.0015
#                     Mean   : 2.9722
#                     3rd Qu.: 0.8795
#                     Max.   :44.7903
#                     NA's   :23
```

Prepare to merge the oil data with the HDI data

Countries are identified in the oil-rent data using three-letter ISO country codes:

```
head(oil_rents$isocode)
# [1] "AFG" "ALB" "DZA" "ASM" "AND" "AGO"
```

To merge with the HDI data, we need the same three-letter country codes. Fortunately, the `countrycode` converts country names to country codes pretty well:

```
library(countrycode)
HDI$isocode <- countrycode(HDI$country,
                           origin="country.name",
                           destination="iso3c")
```

```r
HDI[1:15, c("country", "isocode", "hdi")]
# # A tibble: 15 x 3
#    country                    isocode   hdi
#    <chr>                      <chr>   <dbl>
#  1 Norway                     NOR     0.957
#  2 Ireland                    IRL     0.955
#  3 Switzerland                CHE     0.955
#  4 Hong Kong, China (SAR)     HKG     0.949
#  5 Iceland                    ISL     0.949
#  6 Germany                    DEU     0.947
#  7 Sweden                     SWE     0.945
#  8 Australia                  AUS     0.944
#  9 Netherlands                NLD     0.944
# 10 Denmark                    DNK     0.94
# 11 Finland                    FIN     0.938
# 12 Singapore                  SGP     0.938
# 13 United Kingdom             GBR     0.932
# 14 Belgium                    BEL     0.931
# 15 New Zealand                NZL     0.931
```

Run the merge

Now we can merge the HDI and oil data.

```
dim(HDI)  # dimensions before merging
# [1] 189  15

HDI <- merge(x=HDI, y=oil_rents,
             by="isocode",
             all.x=TRUE)  # do not drop any "x" (HDI) data
```

And do a few quick checks on the merged data.

```
dim(HDI)  # dimensions after merging (added one column)
# [1] 189  16

table(is.na(HDI$oil_rents_pct))  # "missing" oil-rent data
#
# FALSE  TRUE
#   181     8

rownames(HDI) <- HDI$isocode  # this will be useful later
```

Calculate oil-rent and non-oil-rent income

The oil-rent data is as a percentage of GNI.

```
HDI$gni_oil_pc    <- HDI$gni_pc * (HDI$oil_rents_pct / 100)
HDI$gni_nonoil_pc <- HDI$gni_pc - HDI$gni_oil_pc
summary(HDI[ , c("gni_oil_pc", "gni_nonoil_pc")])
#    gni_oil_pc         gni_nonoil_pc
#  Min.   :    0.000   Min.   :  753.9
#  1st Qu.:    0.000   1st Qu.: 4857.8
#  Median :    1.246   Median :12212.1
#  Mean   :  739.236   Mean   :19120.9
#  3rd Qu.:  128.568   3rd Qu.:29368.7
#  Max.   :25811.942   Max.   :88155.2
#  NA's   :8           NA's   :8
```

Many countries have zero oil rents. Before taking logarithms, add one dollar to avoid undefined values.

```
HDI$gni_oil_pc_log    <- log(HDI$gni_oil_pc + 1)
HDI$gni_nonoil_pc_log <- log(HDI$gni_nonoil_pc)
```

R's "formula" syntax for models

Models are specified in R using its "formula" syntax:

```
y ~ x1 + x2, data=df
```

- The response (dependent variable) is to the left of ~;
- Predictors (independent variables) are to the right of the ~;
- Predictors that enter "additively" are separated with plus signs (they are not literally added!);

  Predictors with multiplicative interactions are separated by times signs (asterisks).
- A `data` argument allows to specify the data frame, so that you do not need to retype it for each variable in the formula.

Running the linear model regression

The function for running a linear (OLS) regression in R is `lm`, which stands for “linear m.”

So we can run our model of the “non-income” part of HDI as a function of the (the log of) non-oil-rent GNI and oil rents as:

```
lm(hdi_noninc ~ gni_oil_pc_log + gni_nonoil_pc_log, data=HDI)
#
# Call:
# lm(formula = hdi_noninc ~ gni_oil_pc_log + gni_nonoil_pc_log,
#     data = HDI)
#
# Coefficients:
#       (Intercept)     gni_oil_pc_log  gni_nonoil_pc_log
#         -0.312802          -0.002105           0.112596
```

But the output is not very satisfying – just a bare printout of coefficients!

Assign the model output!

The the `lm` output can be assigned as an object and summarized:

```
lm_out1 <- lm(hdi_noninc ~ gni_oil_pc_log + gni_nonoil_pc_log, data=HDI)
summary(lm_out1)
#
# Call:
# lm(formula = hdi_noninc ~ gni_oil_pc_log + gni_nonoil_pc_log,
#     data = HDI)
#
# Residuals:
#       Min        1Q    Median        3Q       Max
# -0.187955 -0.039392  0.008389  0.040400  0.147095
#
# Coefficients:
#                    Estimate Std. Error t value Pr(>|t|)
# (Intercept)       -0.312802   0.037234  -8.401 1.37e-14 ***
# gni_oil_pc_log    -0.002105   0.001541  -1.366    0.174
# gni_nonoil_pc_log  0.112596   0.004036  27.901  < 2e-16 ***
# ---
# Signif. codes:
# 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 0.06058 on 178 degrees of freedom
#   (8 observations deleted due to missingness)
# Multiple R-squared:  0.8177,^^IAdjusted R-squared:  0.8156
# F-statistic: 399.2 on 2 and 178 DF,  p-value: < 2.2e-16
```

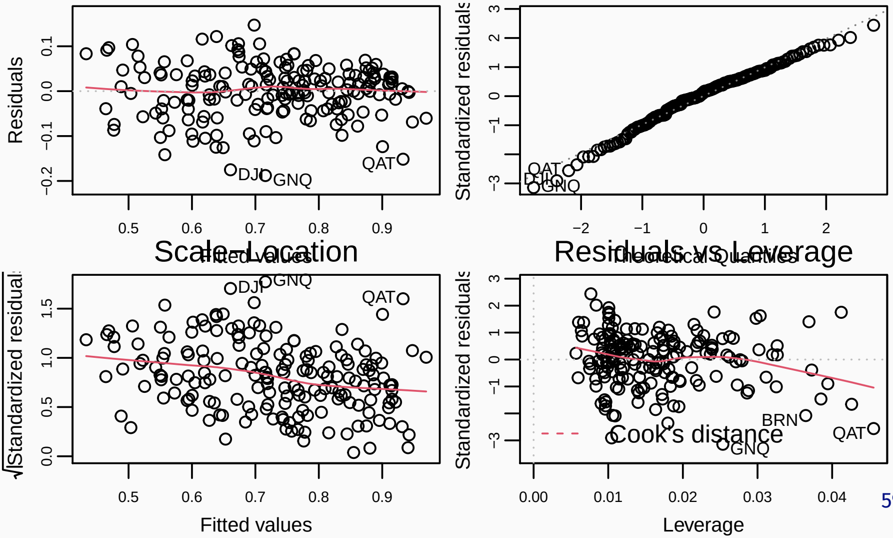
Extracting model estimates

The `lm` output contains model estimates that can be extracted:

```
names(lm_out1)
#  [1] "coefficients"  "residuals"     "effects"
#  [4] "rank"          "fitted.values" "assign"
#  [7] "qr"            "df.residual"   "na.action"
# [10] "xlevels"       "call"          "terms"
# [13] "model"

coef(lm_out1)  # extract coefficients
#      (Intercept)    gni_oil_pc_log gni_nonoil_pc_log
#      -0.31280185       -0.00210509        0.11259562

resid(lm_out1)[1:5]  # extract and view the first five residuals
#         AFG         AGO         ALB         ARE         ARG
# -0.02101448 -0.05647314  0.07106905 -0.05332631  0.06824457

fitted.values(lm_out1)[1:5]  # same for the fitted values
#       AFG       AGO       ALB       ARE       ARG
# 0.5551595 0.6188717 0.7490323 0.8991386 0.7951784
```

Running “regression diagnostics”

```
par(mfrow=c(2, 2))  # set graphics to print "2 x 2"
plot(lm_out1)
```



Outline

What is `swirl`

`swirl`

An R package that provides the infrastructure to run interactive self-study lessons, right in the R console.

(For more information: https://swirlstats.com/.)

"Introduction to R Programming"

A foundational `swirl` course consisting of 15 short lessons (work out your own pace, but most take about 15–20 minutes each).

Install and run `swirl` courses

Install (once-off)

```
## Install the swirl package -- the "infrastructure"
install.packages("swirl")

## Install the R programming course -- the content
swirl::install_course("R Programming")
```

Run the course

```
## "Load" (attach) the swirl package
library(swirl)

## Run swirl
swirl()  # follow the prompts to choose a course, lessons
```

(Mostly do the courses in order, except you may want to skip the second one on "Workspace and Files" and come back to it later.)

Outline

Wrapping up

Learning R

- Push ahead with the `swirl` lessons on “R Programming”;
- For further self-study, with a Tidyverse focus, try:

  Hadley Wickham and Garrett Grolemund, *R for Data Science* (O’Reilly Media, 2017) – available for free online at https://r4ds.had.co.nz/.

For tomorrow

- Data *analysis* in R.