

# Data Science with R - Introduction

Sonia Mazzi - Data Science Campus - ONS

19 June, 2019

# DATA SCIENCE WITH



Please, complete the online survey

<https://www.smartsurvey.co.uk/s/ITAG0/>

Download files from

[https://github.com/sonjam111/DSWR1\\_pub](https://github.com/sonjam111/DSWR1_pub)

# Objectives

- ▶ To understand the flow of scientific data analysis.
- ▶ To understand how R contributes to each stage of the flow.
- ▶ To become acquainted with Tidyverse packages to import different types of data into R.
- ▶ To become acquainted with the process of data tidying and manipulation using R.
- ▶ Be able to do basic visualisations of categorical and continuous data in order to explore a data set.

# Plan

## Day 1

- ▶ AM: the data analysis flow, importing data into R, tidying data sets
- ▶ PM: Case studies 1 and 2.

## Day 2

## Case study 3

# Why R?

- ▶ R was created specifically to support data analysis.
- ▶ R is an interactive environment for data analysis, not just a language.
- ▶ R allows for **reproducible research**.

# The data analysis process

*“There are no routine statistical questions, only questionable statistical routines.” — Sir David Cox*

Scientific data analysis can be summarised as a process which must contain the following 5 stages:

- ▶ Reflection on problem and resources.
- ▶ Data collection.
- ▶ Data preparation.
- ▶ Data analysis.
- ▶ Reporting of results.

It is NOT a non-stop, 5-step unidirectional process, where steps are followed in strict sequence.

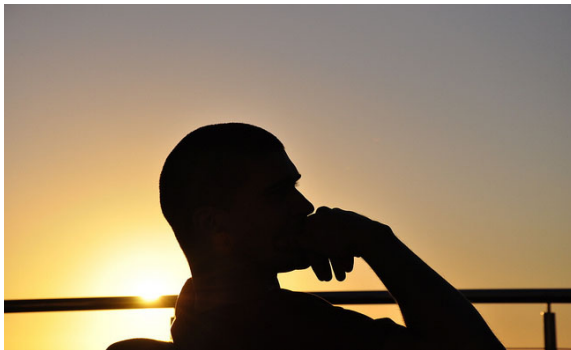
It is a process with 5 stages, which appear in the order above.

At any point it may be necessary to return to a previous stage and re-start from there.



# Reflection

*“Far better an approximate answer to the right question, which is often vague, than an exact answer to the wrong question, which can always be made precise.” — John Tukey*

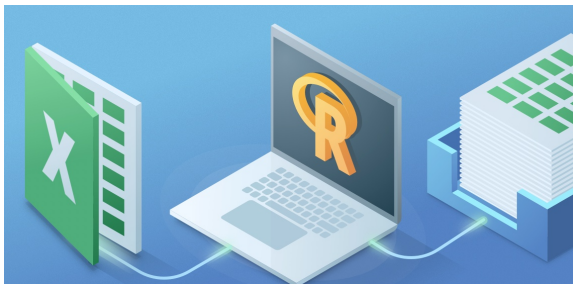


- ▶ What are the questions that need to be answered?
- ▶ What resources are needed to answer such questions?
- ▶ What resources are available?
- ▶ If the data is not yet available, then it is crucial to carefully plan how information will be gathered so that questions of interest can be answered.
- ▶ If data is available try to have a glimpse at it and anticipate challenges, tasks to be performed, etc.

Keep on coming to this stage as needed throughout the data analysis process.

# Data collection

- ▶ Data can arrive in a myriad of ways.
- ▶ How do you transfer the data to your chosen system for analysis?



# Data preparation

Once the data has been imported into the system

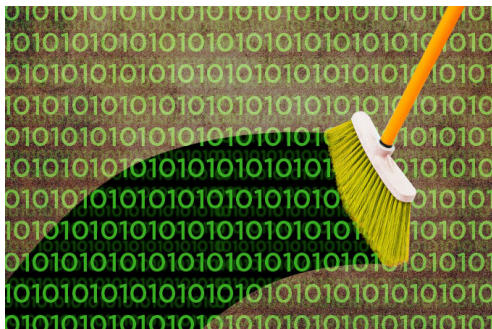
- ▶ Is the format of the data adequate for the system to carry out analyses?
- ▶ Data organised adequately, in a standardised format, is **tidy data**.
- ▶ Data usually arrives in a “messy” state in terms of the way it is organised.
- ▶ Once data is tidy other issues, that must be dealt with, may remain such as errors (mistyped entries), duplication of values, missing values, abnormal values, outliers, etc.

# Data preparation is key to meaningful analysis but time consuming

In the New York Times article **For Big-Data scientists, 'Janitor work' is key hurdle to insights** (Aug 17, 2014) we read

*"Yet far too much handcrafted work — what data scientists call "data wrangling," "data munging" and "data janitor work" — is still required. Data scientists, according to interviews and expert estimates, spend from 50 percent to 80 percent of their time mired in this more mundane labor of collecting and preparing unruly digital data, before it can be explored for useful nuggets. . . . Data experts try to automate as many steps in the process as possible. "But practically, because of the diversity of data, you spend a lot of your time being a data janitor, before you can get to the cool, sexy things that got you into the field in the first place" "*

- ▶ Time spent in data preparation is well-spent time.
- ▶ Data preparation is key for reproducibility and streamlined analyses.
- ▶ Adopting common standards for what constitutes a tidy data set is essential for both correctness and reproducibility.



# Data analysis

Once our data is readable in the chosen system, has been prepared in the required format and the best effort to clean it has been made, it can be passed on to routines that will analyse the data and quantify the results using adequate paradigms.



# Reporting

- ▶ The process is reported for checking and dissemination.
- ▶ The latest trend in reporting is along the lines of reproducible analysis, using interactive digital platforms which can communicate with the data and software inline.





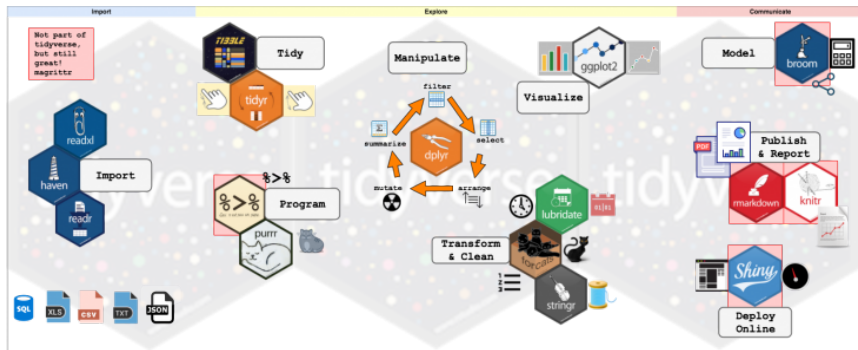
# Data science with R course

- ▶ In this intermediate course we will focus mainly on the first three steps of the data analysis process using the R software: reflexion, data collection and data preparation.
- ▶ We will see ways of importing data in several digital formats into R (excel, SAS, STATA, SPSS, .csv, .tsv, .txt, web data, etc.)
- ▶ We will define minimum qualities that a tidy data set must have and the most common features of messy data.
- ▶ We will learn how to fix messy data via manipulation of the data arrays and ways to detect common abnormalities via data summaries and simple data visualisation.

## R packages for data science

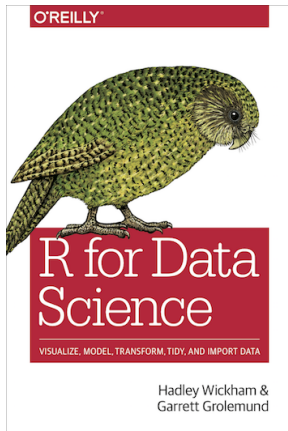
- ▶ The basic unit of communication with R is a **function**, written as a name followed by `()`, e.g. `summary()`. Inside the brackets we write arguments (options) of the function.
- ▶ For example: `summary(mydata)` provides a summary of the data in `mydata`.
- ▶ A package is a collection of R functions, bundled together as they have a common purpose.
- ▶ For example, `dplyr` is a package useful for data manipulation (wrangling).
- ▶ To use a package one must first install it using the function `install.packages()` and then, to use it, it must be invoked or loaded with the function `library()`.
- ▶ Installation of a package is done only once. Loading must be done on each new session.
- ▶ For example: `install.packages("dplyr")` to install and `library(dplyr)` to load the package `dplyr`.

# The Tidyverse (<https://www.Tidyverse.org>)



*“The Tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.”*

## Suggested literature:



<http://r4ds.had.co.nz>

# Cheatsheets

<https://www.rstudio.com/resources/cheatsheets/>



## Golden rules for working with RStudio

- ▶ Create a folder with a meaningful name for your project. Store all the files for the project in this folder.
- ▶ One folder per project.
- ▶ The first action after opening RStudio is to set the working directory, i.e. tell R where to find and store all the files needed or generated.

# Importing data into R

Data may come in many formats

- ▶ text files,
- ▶ excel files or workbooks,
- ▶ data prepared for SAS, SPSS, Stata, etc.,
- ▶ Data from the web.
- ▶ Data from a SQL server.

And more!

- ▶ R is a versatile software that allows data in almost any format to be imported and analysed.



# Importing text files



```
library(readr)
```

The most common type of file is a “flat file”, a text file with a bi-dimensional array of entries or cells. The entries can be separated by either a tab (or blank space), or a comma.

Useful packages to import text files

- ▶ `readr` (in the Tidyverse, the one we will use)
- ▶ `utils` (loads automatically when R is started, always useful)
- ▶ `data.table` (not in the Tidyverse, useful to read large data sets)

utils	readr	use
<code>read.table()</code>	<code>read_delim()</code>	read any flat file with any delimiter
<code>read.csv()</code>	<code>read_csv()</code>	read comma separated values
<code>read.delim()</code>	<code>read_tsv()</code>	read tab delimited files

- ▶ Also useful are `read_table()` and `read_table2()`.
- ▶ Use to read data which has columns separated by white space.

## EXAMPLE.

"EXER\_age\_sex\_race.csv": file containing demographic information about people in a study to assess the health benefits of an exercise plan.

```
data1 <- read_csv("DataFiles/EXER_age_sex_race.csv")
```

```
## Parsed with column specification:  
## cols(  
##   subject_ID = col_integer(),  
##   SexAge_Race = col_character()  
## )
```

To inspect what type of object data1 is, we use the command `class()`

```
class(data1)
```

```
## [1] "tbl_df"      "tbl"        "data.frame"
```

```
#We can see the top 6 rows of `data1`
```

```
head(data1)
```

```
## # A tibble: 6 x 2
##   subject_ID SexAge_Race
##       <int> <chr>
## 1         1 MALE41.2_White
## 2         2 FEMALE42.9_White
## 3         3 FEMALE38.5_White
## 4         4 FEMALE35.6_Hispanic
## 5         5 FEMALE48.5_White
## 6         6 FEMALE36.9_NA
```

```
#and the last 3 rows of `data1`
```

```
tail(data1, n = 3)
```

```
## # A tibble: 3 x 2
##   subject_ID SexAge_Race
##       <int> <chr>
## 1      4998 FEMALE44.1_Black
## 2      4999 FEMALE46.4_Black
## 3      5000 FEMALE49.9_Black
```

Note:

```
class(read.csv("DataFiles/EXER_age_sex_race.csv"))
```

```
## [1] "data.frame"
```

and

```
class(read_csv("DataFiles/EXER_age_sex_race.csv"))
```

```
## Parsed with column specification:
```

```
## cols(
```

```
##   subject_ID = col_integer(),
```

```
##   SexAge_Race = col_character()
```

```
## )
```

```
## [1] "tbl_df"      "tbl"        "data.frame"
```

# Importing Excel files

- ▶ Use the package `readxl` from the Tidyverse.
- ▶ Use the function `read_excel()`.
- ▶ Also useful is `excel_sheets()` : lists different sheets in an excel workbook.



```
library(readxl)
```

## EXAMPLE. Importing an excel file sourced from Gapminder

Let us consider geographical information of countries in the world from <https://www.gapminder.org/data/geo/>.

- ▶ “DataGeographiesGapminder.xlsx”: file containing geographical information about countries.
- ▶ Workbook with many sheets.
- ▶ The second sheet is the one of interest.



```
#names of sheets in workbook
```

```
excel_sheets("DataFiles/DataGeographiesGapminder.xlsx")
```

```
## [1] "ABOUT" "List of countries" "List of regions"
```

```
## [4] "World"
```

```
# import only the second sheet
```

```
continent <-
```

```
  read_excel("DataFiles/DataGeographiesGapminder.xlsx", sheet = 2)
```

```
# first six rows of continent
```

```
head(continent)
```

```
## # A tibble: 6 x 11
```

```
##   geo   name four_regions eight_regions six_regions members_oecd_g77
```

```
##   <chr> <chr> <chr>         <chr>         <chr>         <chr>
```

```
## 1 afg   Afgh~ asia       asia_west     south_asia   g77
```

```
## 2 alb   Alba~ europe     europe_east   europe_cen~ others
```

```
## 3 dza   Alge~ africa     africa_north middle_eas~ g77
```

```
## 4 and   Ando~ europe     europe_west   europe_cen~ others
```

```
## 5 ago   Ango~ africa     africa_sub_s~ sub_sahara~ g77
```

```
## 6 atg   Anti~ americas   america_north america      g77
```

```
## # ... with 5 more variables: Latitude <dbl>, Longitude <dbl>, `UN member
```

```
## #   since` <dtm>, `World bank region` <chr>, `World bank income group
```

```
## #   2017` <chr>
```

If we want to retrieve all the sheets in an Excel workbook we can use the function `map()` from the package `purrr` in the Tidyverse.



```
library(purrr)
```

The function `map()` applies a function to each of the elements of a vector and outputs a list with as many entries as the length of the vector.

```
sheets <- excel_sheets("DataFiles/DataGeoGapm.xlsx")
table_list <-
  map(sheets, ~read_excel("DataFiles/DataGeoGapm.xlsx", sheet = .x))
```

```
head(table_list[[2]])
```

```
## # A tibble: 6 x 11
##   geo   name four_regions eight_regions six_regions members_oecd_g77
##   <chr> <chr> <chr>         <chr>         <chr>         <chr>
## 1 afg   Afgh~ asia         asia_west      south_asia     g77
## 2 alb   Alba~ europe       europe_east    europe_cen~    others
## 3 dza   Alge~ africa       africa_north   middle_eas~    g77
## 4 and   Ando~ europe       europe_west    europe_cen~    others
## 5 ago   Ango~ africa       africa_sub_s~  sub_sahara~    g77
## 6 atg   Anti~ americas     america_north america         g77
## # ... with 5 more variables: Latitude <dbl>, Longitude <dbl>, `UN member
## #   since` <dtm>, `World bank region` <chr>, `World bank income group
## #   2017` <chr>
```

The same action, using the %>% operator

```
library(purrr)
sheets <- excel_sheets("DataFiles/DataGeoGapm.xlsx")
table_list <- sheets %>%
  map(~read_excel("DataFiles/DataGeoGapm.xlsx", sheet = .x))
```

```
head(table_list[[2]])
```

```
## # A tibble: 6 x 11
##   geo   name four_regions eight_regions six_regions members_oecd_g77
##   <chr> <chr> <chr>          <chr>          <chr>          <chr>
## 1 afg   Afgh~ asia         asia_west      south_asia    g77
## 2 alb   Alba~ europe       europe_east    europe_cen~  others
## 3 dza   Alge~ africa       africa_north   middle_eas~  g77
## 4 and   Ando~ europe       europe_west    europe_cen~  others
## 5 ago   Ango~ africa       africa_sub_s~ sub_sahara~  g77
## 6 atg   Anti~ americas     america_north america        g77
## # ... with 5 more variables: Latitude <dbl>, Longitude <dbl>, `UN member
## #   since` <dtm>, `World bank region` <chr>, `World bank income group
## #   2017` <chr>
```

# Importing data from Statistical software (SAS, SPSS, STATA)

The package `haven`, of the Tidyverse, provides easy to use commands to import data from SAS, STATA and SPSS



```
library(haven)
```

Software	Function
SAS	<code>read_sas()</code>
STATA	<code>read_dta()</code> or <code>read_stata()</code> (identical commands)
SPSS	<code>read_sav()</code> or <code>read_por()</code> (depending on the file type to be imported)

# Importing a SAS file

We use the function `read_sas()` in the package `haven` of the Tidyverse. We will import the file “tax.sas7bdat” which contains information about the income and tax paid of 30 US firms in 1988, 1989.

```
tax <- read_sas("DataFiles/tax.sas7bdat")
```

```
glimpse(tax)
```

```
## Observations: 30
## Variables: 4
## $ INC88 <dbl> 9.215, 2.047, 9.989, 8.321, 4.588, 4.736, 3.596, 4.830, ...
## $ TAX88 <dbl> 1.643, 0.413, 1.752, 1.408, 0.838, 0.748, 0.577, 0.752, ...
## $ INC89 <dbl> 9.518, 2.068, 9.992, 8.515, 4.389, 5.015, 3.811, 4.939, ...
## $ TAX89 <dbl> 2.125, 0.565, 2.221, 1.905, 0.943, 1.051, 0.819, 1.015, ...
```

```
head(tax, n = 5)
```

```
## # A tibble: 5 x 4
##   INC88 TAX88 INC89 TAX89
##   <dbl> <dbl> <dbl> <dbl>
## 1  9.22  1.64   9.52  2.12
## 2  2.05  0.413  2.07  0.565
## 3  9.99  1.75   9.99  2.22
## 4  8.32  1.41   8.52  1.90
## 5  4.59  0.838  4.39  0.943
```

```
tail(tax, n = 5)
```

```
## # A tibble: 5 x 4
##   INC88 TAX88 INC89 TAX89
##   <dbl> <dbl> <dbl> <dbl>
## 1  7.26  1.14   7.64  1.72
## 2  2.13  0.414  2.17  0.433
## 3  7.53  1.33   7.86  1.46
## 4  9.58  1.66  10.00  2.17
## 5  2.02  0.351  2.26  0.447
```



# Importing a SPSS file

- ▶ Use the function `read_sav()` (for .sav files)

or

- ▶ use `read_por()` (for .por files),
- ▶ both in the package `haven`.

# EXAMPLE.

- ▶ SPSS file “sleep.sav”
- ▶ Data concerns the prevalence and impact of sleep problems on people's lives.
- ▶ Measured variables are sleep behaviour (e.g. hours slept per night), sleep problems (e.g. difficulty getting to sleep) and the impact that these problems have on aspects of their lives (work, driving, relationships).
- ▶ The sample consists of 271 individuals.

```
sleep <- read_sav("DataFiles/sleep.sav")
```

```
head(sleep)
```

```
## # A tibble: 6 x 55
##       id sex    age marital edlevel weight height healthrate fitrate
##   <dbl> <dbl> <dbl> <dbl+1> <dbl+1>  <dbl>  <dbl> <dbl+1> <dbl+1>
## 1    83 0      42 2        2      52   162 10      7
## 2   294 0      54 2        5      65   174 " 8"    7
## 3   425 1      NA 2        2      89   170 " 6"    5
## 4    64 0      41 2        5      66   178 " 9"    7
## 5   536 0      39 2        5      62   160 " 9"    5
## 6    57 0      66 2        4      62   165 " 8"    8
## # ... with 46 more variables: weightrate <dbl+1>, smoke <dbl+1>,
## # smokenum <dbl>, alchcohol <dbl>, caffeine <dbl>, hourwnit <dbl>,
## # hourwend <dbl>, hourneed <dbl>, trubslep <dbl+1>,
## # trubstay <dbl+1>, wakenite <dbl+1>, niteshft <dbl+1>,
## # liteslp <dbl+1>, refreshhd <dbl+1>, satsleep <dbl+1>,
## # qualslp <dbl+1>, stressmo <dbl+1>, medhelp <dbl+1>,
## # problem <dbl+1>, impact1 <dbl+1>, impact2 <dbl+1>,
## # impact3 <dbl+1>, impact4 <dbl+1>, impact5 <dbl+1>,
## # impact6 <dbl+1>, impact7 <dbl+1>, stopb <dbl+1>,
## # restlss <dbl+1>, drvsleep <dbl+1>, drvresul <dbl+1>, ess <dbl>,
## # anxiety <dbl>, depress <dbl>, fatigue <dbl>, lethargy <dbl>,
## # tired <dbl>, sleepy <dbl>, energy <dbl>, stayslprec <dbl+1>,
## # getsleprec <dbl+1>, qualsleeprec <dbl+1>, totsas <dbl+1>,
## # cigsgp3 <dbl+1>, agegp3 <dbl+1>, probsleeprec <dbl+1>
```



Factor columns have numeric entries, and each number has a corresponding name or label.

Use this function to get the labels of a factor

```
print_labels(sleep$marital)
```

```
##  
## Labels:  
##   value      label  
##     1      single  
##     2 married/defacto  
##     3      divorced  
##     4      widowed
```

The variable `marital` is a factor with four levels: level 1 corresponds to single people, level 2 to married people, level 3 to divorced and level 4 to widowed people.

# Importing a STATA file

“trade.dta”: US yearly import and export numbers of sugar, in USD and lbs.

```
sugar <- read_dta("DataFiles/trade.dta")
```

```
glimpse(sugar)
```

```
## Observations: 10  
## Variables: 5  
## $ Date      <dbl+lbl> 10, 9, 8, 7, 6, 5, 4, 3, 2, 1  
## $ Import    <dbl> 37664782, 16316512, 11082246, 35677943, 9879878, 1539...  
## $ Weight_I  <dbl> 54029106, 21584365, 14526089, 55034932, 14806865, 174...  
## $ Export    <dbl> 54505513, 102700010, 37935000, 48515008, 71486545, 12...  
## $ Weight_E  <dbl> 93350013, 158000010, 88000000, 112000005, 131800000, ...
```

- ▶ The values in the column Date are labelled and the labels are dates.
- ▶ The other four columns contain numerical values.

```
print_labels(sugar$Date)
```

```
##  
## Labels:  
##   value      label  
##      1 2004-12-31  
##      2 2005-12-31  
##      3 2006-12-31  
##      4 2007-12-31  
##      5 2008-12-31  
##      6 2009-12-31  
##      7 2010-12-31  
##      8 2011-12-31  
##      9 2012-12-31  
##     10 2013-12-31
```

# Importing data from the web

- ▶ Data from the web can be read directly into R if it is a text file (comma or tab separated), sas, stata or spss file.
- ▶ Instead of a filename, we specify a url.
- ▶ Excel files cannot be retrieved directly from the web. First the file needs to be downloaded using the command `download.file()`.

## EXAMPLE. Excel file from Gapminder

- ▶ Data: Adults with HIV (estimated prevalence of HIV in percentage, ages 15-49) from Gapminder,
- ▶ url: <https://docs.google.com/spreadsheet/pub?key=pyj6tScZqmEfbZyl0qjbiRQ&output=xlsx>
- ▶ The function `read_excel()` cannot download excel files directly from the web.
- ▶ First use the function `download.file()` to download the file into a directory,
- ▶ then `read_excel()` to read it into R.



```
url <- "https://docs.google.com/spreadsheets/pub?key=pyj6tScZqmEfbZyl0qjbiRQ&outp
```

```
download.file(url, "DataFiles/HIV.xlsx")
```

```
HIV <- read_excel("DataFiles/HIV.xlsx")
```

```
head(HIV)
```

```
## # A tibble: 6 x 34
##   `Estimated HIV ~` `1979.0` `1980.0` `1981.0` `1982.0` `1983.0` `1984.0`
##   <chr>             <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 Abkhazia          NA      NA      NA      NA      NA      NA
## 2 Afghanistan        NA      NA      NA      NA      NA      NA
## 3 Akrotiri and Dh~    NA      NA      NA      NA      NA      NA
## 4 Albania            NA      NA      NA      NA      NA      NA
## 5 Algeria            NA      NA      NA      NA      NA      NA
## 6 American Samoa     NA      NA      NA      NA      NA      NA
## # ... with 27 more variables: `1985.0` <dbl>, `1986.0` <dbl>,
## #   `1987.0` <dbl>, `1988.0` <lgl>, `1989.0` <lgl>, `1990.0` <dbl>,
## #   `1991.0` <dbl>, `1992.0` <dbl>, `1993.0` <dbl>, `1994.0` <dbl>,
## #   `1995.0` <dbl>, `1996.0` <dbl>, `1997.0` <dbl>, `1998.0` <dbl>,
## #   `1999.0` <dbl>, `2000.0` <dbl>, `2001.0` <dbl>, `2002.0` <dbl>,
## #   `2003.0` <dbl>, `2004.0` <dbl>, `2005.0` <dbl>, `2006.0` <dbl>,
## #   `2007.0` <dbl>, `2008.0` <dbl>, `2009` <chr>, `2010` <chr>,
## #   `2011` <chr>
```

# Tidy data

*"Happy families are all alike; every unhappy family is unhappy in its own way." Leo Tolstoy*

- ▶ Hadley Wickham in “Tidy Data” defines the three qualities of tidy data which standardise the process of dealing with any data set:

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

- ▶ These three qualities are what the end product of the data tidying process must possess.
- ▶ *Messy data* is data which is not tidy.
- ▶ Applying the tidy data criteria standardises the structure of a data set, making exploration and analysis of data easier and less error-prone.

# Tidying messy data sets

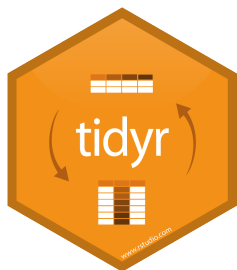
Common problems with messy data sets are:

1. Column headers are values, not variable names.
2. Multiple variables are stored in one column.
3. Variables are stored in both rows and columns.
4. Multiple types of observational units are stored in the same table.
5. A single observational unit is stored in multiple tables.

The first three features of a messy data set are the most common.

- ▶ A tidy data set is not a correct data set.
- ▶ Tidy data refers only to the format of data.
- ▶ Other problems with data sets are errors, unusual observations, duplicated entries, missing data, and many more.
- ▶ These are dealt with once the data is tidy.

# Tidying data



```
library(tidyr); library(dplyr); library(stringr)
```

## tidyr functions to tidy data

**EXAMPLE.** Number of text messages that Sue and Peter sent in 2016 and 2017.

Name	2016	2017
Sue	300	500
Peter	400	600

- ▶ Observational units are Sue and Peter
- ▶ Variables are year and number of text messages sent.
- ▶ Column headers are values of variable year, not variable names.
- ▶ A tidy version of this data set has columns Name, Year, and Number of text messages.

## How to transform the messy data

Name	2016	2017
Sue	300	500
Peter	400	600

into tidy data?

Name	Year	NrSMS
Sue	2016	300
Peter	2016	400
Sue	2017	500
Peter	2017	600



Use the function 'gather()', in 'tidyr', to create a column with column names values.

First let us create the tibble (enhanced data frame) sms

```
c1 <- c("Sue", "Peter")
c2 <- c(300, 400)
c3 <- c(500, 600)
sms <- tibble("Name" = c1, "2016" = c2, "2017" = c3)
sms
```

```
## # A tibble: 2 x 3
##   Name   `2016` `2017`
##   <chr>   <dbl>   <dbl>
## 1 Sue      300     500
## 2 Peter    400     600
```

Tidy the data by gathering column names under a new variable and creating a column with the corresponding values.

```
#"Year": the "key", new column with the column names being gathered  
#"NrSMS": the "value", new column that will have the values which used  
# to be under 2016 and 2017  
#the -1 stands for don't gather the first column
```

```
sms_tidy <- gather(sms, key = "Year", value = "NrSMS", -1)
```

```
sms_tidy
```

```
## # A tibble: 4 x 3  
##   Name   Year  NrSMS  
##   <chr> <chr> <dbl>  
## 1 Sue    2016    300  
## 2 Peter  2016    400  
## 3 Sue    2017    500  
## 4 Peter  2017    600
```

## EXAMPLE

Number of text messages and tweets Alice, Ann and John made in 2016.

Name	Number	Type
Alice	100	sms
Alice	200	tw
Ann	300	sms
Ann	400	tw
John	500	sms
John	600	tw

- ▶ Observational units are Alice, Ann and John
- ▶ Variables measured are number of sms sent and number of tweets made.
- ▶ Type is not a measured variable. It stores the names of two variables: sms and tweet.

Messy data: variables are stored in both rows and columns.

## How to transform the messy data

Name	Number	Type
Alice	100	sms
Alice	200	tw
Ann	300	sms
Ann	400	tw
John	500	sms
John	600	tw

into tidy data?

Name	sms	tw
Alice	100	200
Ann	300	400
John	500	600

Use the function 'spread()', in 'tidyr', to create new columns from the values of another column.

To do this in R let us first create the tibble smstwt

```
Name <- c("Alice", "Alice", "Ann", "Ann", "John", "John")
Number <- c(100, 200, 300, 400, 500, 600)
Type <- c("sms", "twit", "sms", "twit", "sms", "twit")
```

```
smstwt <- tibble(Name, Number, Type)
smstwt
```

```
## # A tibble: 6 x 3
##   Name    Number Type
##   <chr>   <dbl> <chr>
## 1 Alice     100 sms
## 2 Alice     200 twt
## 3 Ann       300 sms
## 4 Ann       400 twt
## 5 John      500 sms
## 6 John      600 twt
```

In order to tidy the data set we must “spread” the values of Type into two new columns, “sms” and “tw”, and distribute the values in Number accordingly.

```
#Type "key" : create new columns with names from values of Type.  
#Number "value": the values to spread under the new columns.  
#
```

```
smstwt_tidy <- spread(smstwt, key = Type, value = Number)
```

```
smstwt_tidy
```

```
## # A tibble: 3 x 3  
##   Name      sms    twt  
##   <chr> <dbl> <dbl>  
## 1 Alice   100    200  
## 2 Ann     300    400  
## 3 John    500    600
```

**EXAMPLE.** Extra data about the number of sms sent and tweets made by Alice, Ann and John in 2017.

Name	sms_2016	twit_2016	sms_2017	twit_2017
Alice	100	200	10	20
Ann	300	400	30	40
John	500	600	50	60

- ▶ The data without the last two columns was tidy; this data set is messy.
- ▶ Observational units are Alice, Ann and John and measured variables are year, number of tweets and number of sms.

Let us create a tibble, `smstwt2`, with the data.

```
Name <- c("Alice", "Ann", "John")
sms_2016 <- c(100, 300, 500)
tw_2016 <- c(200, 400, 600)
sms_2017 <- c(10, 30, 50)
tw_2017 <- c(20, 40, 60)
```

```
smstwt2 <- tibble(Name, sms_2016, tw_2016, sms_2017, tw_2017)
```

```
smstwt2
```

```
## # A tibble: 3 x 5
##   Name sms_2016 tw_2016 sms_2017 tw_2017
##   <chr>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 Alice     100     200      10      20
## 2 Ann       300     400      30      40
## 3 John      500     600      50      60
```



- ▶ Names of columns, except Name, are values of variables.
- ▶ Gather column names into one new column named “year-type” and put the corresponding values in the new column “Number”.

```
smstwt2_g <- gather(smstwt2, "type_year", "Number",-1)
```

```
head(smstwt2_g)
```

```
## # A tibble: 6 x 3
##   Name  type_year Number
##   <chr> <chr>      <dbl>
## 1 Alice sms_2016    100
## 2 Ann   sms_2016    300
## 3 John  sms_2016    500
## 4 Alice twt_2016    200
## 5 Ann   twt_2016    400
## 6 John  twt_2016    600
```

- ▶ Not tidy: multiple variables are stored in column type

- ▶ Separate column `type_year` into two new columns: “type” and “year”.

The function `'separate()'` from `'tidyr'` separates one column into several.

```
smstwt2_gs <-  
  separate(smstwt2_g, "type_year", c("type", "year"), sep = "_")
```

```
head(smstwt2_gs)
```

```
## # A tibble: 6 x 4  
##   Name  type  year  Number  
##   <chr> <chr> <chr>   <dbl>  
## 1 Alice sms   2016     100  
## 2 Ann  sms   2016     300  
## 3 John sms   2016     500  
## 4 Alice twt   2016     200  
## 5 Ann  twt   2016     400  
## 6 John twt   2016     600
```

- ▶ “spread” the values of type into “sms” and “twit”, with the corresponding values of Number.

```
smstwt2_tidy <- spread(smstwt2_gs, type, Number)
```

```
smstwt2_tidy
```

```
## # A tibble: 6 x 4
##   Name  year    sms    twt
##   <chr> <chr> <dbl> <dbl>
## 1 Alice 2016    100   200
## 2 Alice 2017     10    20
## 3 Ann   2016    300   400
## 4 Ann   2017     30    40
## 5 John  2016    500   600
## 6 John  2017     50    60
```

From messy data

Name	sms_2016	twit_2016	sms_2017	twit_2017
Alice	100	200	10	20
Ann	300	400	30	40
John	500	600	50	60

to tidy data

Name	year	sms	twit
Alice	2016	100	200
Alice	2017	10	20
Ann	2016	300	400
Ann	2017	30	40
John	2016	500	600
John	2017	50	60

We can use `%>%` to do the entire tidying in one go, without creating the intermediate objects.

```
smstwt2_tidy <- smstwt2 %>%  
  gather("type_year", "Number", -1) %>%  
  separate("type_year", c("type", "year"), sep = "_") %>%  
  mutate(year = as.numeric(year)) %>%  
  spread(type, Number)
```

```
smstwt2_tidy
```

```
## # A tibble: 6 x 4  
##   Name   year   sms   twt  
##   <chr> <dbl> <dbl> <dbl>  
## 1 Alice  2016   100   200  
## 2 Alice  2017    10    20  
## 3 Ann    2016   300   400  
## 4 Ann    2017    30    40  
## 5 John   2016   500   600  
## 6 John   2017    50    60
```

**EXAMPLE.** Add information on the ages of Alice, Ann, John, Peter and Sue

Name	Age
Alice	15
Ann	25
John	35
Peter	45
Sue	55

Now, we have feature 5 of messy data sets: a single observational unit is stored in multiple tables.

Let us enter the new information into R.

```
dem1 <- c("Alice", "Ann", "John", "Peter", "Sue")  
dem2 <- c(15, 25, 35, 45, 55)
```

```
dem <- tibble("Name" = dem1, "Age" = dem2)
```

```
dem
```

```
## # A tibble: 5 x 2  
##   Name    Age  
##   <chr> <dbl>  
## 1 Alice    15  
## 2 Ann      25  
## 3 John     35  
## 4 Peter    45  
## 5 Sue      55
```

- Add this information to our data sets `sms_tidy` and `smstwt2_tidy`.

The function 'inner\_join()' of the 'dplyr' package will join data sets by matching only the entries of all common columns, or by a specific column, in both data sets.

```
inner_join(sms_tidy,dem)
```

```
## Joining, by = "Name"
```

```
## # A tibble: 4 x 4
##   Name   Year  NrSMS   Age
##   <chr> <chr> <dbl> <dbl>
## 1 Sue   2016    300    55
## 2 Peter 2016    400    45
## 3 Sue   2017    500    55
## 4 Peter 2017    600    45
```



```
inner_join(smstwt2_tidy,dem)
```

```
## Joining, by = "Name"
```

```
## # A tibble: 6 x 5
```

```
##   Name   year   sms   twt   Age
##   <chr> <dbl> <dbl> <dbl> <dbl>
## 1 Alice  2016   100   200   15
## 2 Alice  2017    10    20   15
## 3 Ann    2016   300   400   25
## 4 Ann    2017    30    40   25
## 5 John   2016   500   600   35
## 6 John   2017    50    60   35
```

- ▶ Age is added twice for each observational unit.
- ▶ This in itself can be considered a messy feature: multiple types in one table.
- ▶ May need to keep, for example, demographic information about observational units in a separate data sets,
- ▶ or may need to merge both data sets into one for the purpose of analysis.

- ▶ Messy data is not necessarily incorrect data.
- ▶ Messy features are introduced when data for presentations is confused with data for analysis.
- ▶ Elegantly presented data may constitute a very messy data set and a tidy data set is usually unsuitable for presentation.