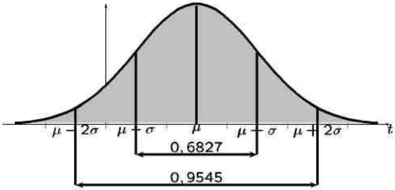
Statistics

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2}$$



Bachelor Studiengang Informatik

Prof. Dr. Egbert Falkenberg

Fachbereich Informatik & Ingenieurwissenschaften

Wintersemster 21/22

Section 1

Introduction to R

What is R? I

- A language and interactive environment for statistical calculations and Graphics.
- Open source project and runing on Unix (Linux), Windows and MacOS. [1].
- Offers a wide variety of standard statistical routines for data evaluation, and the possibility to display the data and results graphically.
- Many other routines are available it in freely available packages.
- Easy to extend R for your own, specialized applications and create your own modules.
- R is available for download on the Comprehensive R Archive Network (CRAN). There you can also find more information, FAQ's, instructions and a lot of packages for R.

[1] For more information and to download, please use http://www.r-project.org

What is R? II

Characteristics for R:

- efficient data management and data storage
- Operators for vector and matrix calculations
- a comprehensive collection of functions for statistical analysis
- Graphics to facilitate data analysis
- a simple but powerful programming language

Although R has a command line interface, there are several graphical front-ends, most notably RStudio a free and open-source integrated development environment for R. [2]

[2] You can get more information and download from https://www.rstudio.com/

Why R? I

Compare Sauer: Moderne Datenanalyse mit R, chapter 2.2

Advantages:

- R knows the modern methods of statistics. New methods are quickly made available to the general public.
- R is reproducible, i.e. the data and the processing steps are separated. In Excel, for example, this is not possible.
- R can be automated and parameterized.
- R is free, i.e. open source and free of charge.
- R offers a large number of diagrams with many design options.
- R is very widespread and is considered one of the most powerful statistics programs.

Why R? II

Disdvantages:

- Comparatively old (development beginning in 1976), a number of old parts are dragged along for compatibility reasons. Therefore, innovations of R do not take place in the core of R but in packages.
- R has been developed by statisticians and not programmers. Therefore it is not as efficient and widely usable as other programming languages.
- Phyton is more widespread in the field of computer science. The relative new programming language Julia is considered to be faster.
- The training in R is more complex than in other programs such as Excel. With increasing complexity of problems this has a decreasing meaning.
- Error messages in R are often difficult to understand. But there are many forums on the internet like stackoverflow.com for questions about R.

Questions

Which of the following statements are true or false?

t	f	
□	□	R is commercial software.
□	□	R is available for Windows, Mac OS and Linux.
□	□	Alternatives to R are Python, Julia and Excel.
□	□	There are about 1000 R packages on CRAN.
□	□	Excel mixes syntax and data; R does not.
□	□	Separation of syntax and data potentially improves the reproducibility of data analysis.

RStudio I

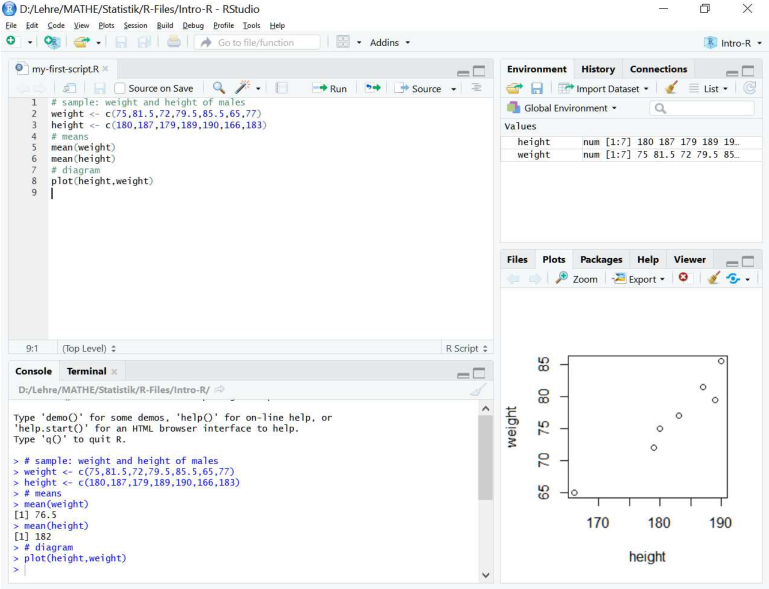
- An integrated development environment for R
- Oscar Torres - Reyna: Introduction to RStudio, https://dss.princeton.edu/training/RStudio101.pdf
- Some usefull settings for the work with RStudio could be find in Paul Hiemstra: RStudio Tutorial: working with RStudio http://stcorp.nl/R_course/rstudio_tutorial.html.

RStudio II

Example: Sample of heights and weights of males
$weight_i : 75, 81.5, 72, 79.5, 85.5, 65, 77$
$height_i : 180, 187, 179, 189, 190, 166, 183$

- Averages of weight and height
- Diagram of the pairs (height,weight)?

RStudio II


D:/Lehre/MATHE/Statistik/R-Files/Intro-R - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Intro-R
my-first-script.R
Source on Save
Run
Source
9:1
(Top Level)
R Script
Environment History Connections
Import Dataset
List
Global Environment
Files Plots Packages Help Viewer
Zoom
Export
Console Terminal
D:/Lehre/MATHE/Statistik/R-Files/Intro-R/
weight
65
70
75
80
85
170
180
190
height

```r
# sample: weight and height of males
weight <- c(75,81.5,72,79.5,85.5,65,77)
height <- c(180,187,179,189,190,166,183)
# means
mean(weight)
mean(height)
# diagram
plot(height,weight)
```

```
Values
  height    num [1:7] 180 187 179 189 19…
  weight    num [1:7] 75 81.5 72 79.5 85…
```

```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> # sample: weight and height of males
> weight <- c(75,81.5,72,79.5,85.5,65,77)
> height <- c(180,187,179,189,190,166,183)
> # means
> mean(weight)
[1] 76.5
> mean(height)
[1] 182
> # diagram
> plot(height,weight)
> 
```

Questions

Which of the following statements are true or false?

t	f	
□	□	RStudio is a development environment for R.
□	□	The RStudio script window displays R output.
□	□	Only objects displayed in the Environment window exist in the current session.
□	□	CRAN provides R packages for a fee.
□	□	Packages must be installed each time R is started.
□	□	Additional required R-packages have to be reloaded when opening R again.
□	□	Upper and lower case in R is irrelevant.

Variables and Assignments I

Source: compare https://homepage.univie.ac.at/david.wozabal/SS2008/Session1.pdf

Basic (atomic) types of variables

- numeric: Integer and double data type (64 bit, double precision)
- complex: complex numbers
- logical: logical data type, possible values: TRUE and FALSE
- character: characters
  possible values: letters, numbers, punctuation marks,. . .

Variables and Assignments II

Remark:

- Assignments are made using $< -$.
- The content of a variable can be checked by entering the variable name.
- To display the structure of a R object, use the function str().
- The generated variables remain in memory until R is terminated. To get an overview of the variables in the memory, you can use the function ls() use. To delete variables from memory, use the command rm(variableName).

Data Structures Vectors

Compare:
https://homepage.univie.ac.at/david.wozabal/SS2008/Session2.pdf

- Individual variables can be combined to vectors using the function c(). Numerical vectors can be calculated in the same way as numbers. Operations are performed point by point.
- For logical vectors, the logical operators & (and), | (or) and ! (not) is defined.
- Others methods to create vectors:
  - from:to generates a sequence
  - seq(from,to) generates a sequence by= specifies increment; length= specifies desired length
  - rep(x,times) replicate x times; use each= to repeat “each” element of x each times

Data Structures Matrices I

matrix(data = NA, nrow= 1, ncol = 1, byrow = FALSE, dimnames = NULL)

- The argument data should be a vector containing the entries of the matrix. The arguments nrow and ncol determine the shape of the matrix.
- The logical argument byrow can be used to determine whether matrices should be read column by column or row by row (default: column by column). Matrices can be joined column by column or line using the cbind and rbind functions.
- Matrices can be used as usual. The operators +, - function point by point. Multiplication can be done pointwise by $*$ or by matrix multiplication by $\% * \%$.

Data Structures Matrices II

- Vectors are one-dimensional arrays and matrices are two-dimensional arrays. R offers the possibility to define arrays of higher dimensions using the function array(data = NA, dim = length(data), dimnames = NULL) It should be noted here that the dimensions of the array are determined using the vector dim be specified.
- Access elements of a matrix:
  - square brackets: specify the positions of the values you want to view or change.
  - logical indexing: a data structure (vector, matrix, array) of logical variables is passed that has the same dimension as the structure to be indexed. The logical variable selects the elements at whose position it itself has the value TRUE.

Data Structures Factors

- Factors are used to represent nominal data.
- Factors are stored as integers, and have labels associated with these unique integers. While factors look (and often behave) like character vectors, they are actually integers under the hood, and you need to be careful when treating them like strings.
- Once created, factors can only contain a pre-defined set values, known as levels.
- factor() is used to create factors in R.

Data Structures Data Frames and Tibbles I

- **Data Frame:**
  - two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.
  - data can be of numeric, factor or character type
  - each column should contain same number of data items.
- **Tibbles:**
  - Modern form of data.frame: tibble() or data_frame()
  - Tibbles provide the good and effective parts of data.frame but do less (i.e. they do not change variable names or types, and do not do partial matching) and complain more (e.g. when a variable does not exist).
  - Tibble commands are part of the tidyverse package.

Data Structures Data Frames and Tibbles II

Properties of tibbles:

- It never changes an input's type, i.e. no more converting string vectors to factors.
- Strict about subsetting: [ ] always returns another tibble.
- No partial matching with $ referring a column.
- Printing: When you print a tibble, it only shows the first ten rows and all the columns that fit on one screen. It also prints an abbreviated description of the column type. This is an advantage in case of large datasets containing complex objects.

Questions

Which of the following statements are true or false?

t	f	
□	□	Assigments are made using “<-”.
□	□	The structure of a R object is displayed by entering the variable name.
□	□	The result of “ 2 c(3,4,6)” is the vector (6,8,12).
□	□	matrix(data=5:10,ncol=3,nrow=2,byrow=TRUE) creates a 2x3 matrix with first row (5,6,7) and second row (8,9,10).
□	□	The first column of a matrix X can be accessed by $X[1,]$.
□	□	Data frames are two dimensional array-like structures in which columns may have different types but must have the same length.
□	□	Data frames resp. tibbles have the property that subsetting returns always another data frame resp tibble.

Functions I

Compare
https://homepage.univie.ac.at/david.wozabal/SS2008/Session2.pdf

- R has a large numbers of functions built in.
- User can create their own functions to avoid to enter calculations that occur again and again in a similar form completely again and again in the R Console.
- To write a function in R, use the keyword function.

```
myfunction <- function(arg1, arg2, ... ){
        statements
        return(object)
}
```

Functions II

If Statements

-
  ```
  if (condition) {
          statements }
  ```

  If the condition is TRUE, the statements gets executed. But if it is FALSE, nothing happens.

-
  ```
  if (condition) {
          statements
  } else {
          statements
  }
  ```

  The else part is optional and is only evaluated if condition is FALSE. Mention else must be in the same line as the closing braces of the if statement.

Functions III

Loops:

-
  ```
  while (condition)
  {
          statements
  }
  ```

  Here, condition is evaluated and the body of the loop is entered if the result is TRUE. The statements inside the loop are executed and the flow returns to evaluate the condition again.

-
  ```
  for (val in sequence)
  {
          statements
  }
  ```

  Here, iterate over a vector: sequence is a vector and val takes on each of its value during the loop. In each iteration, the statements are evaluated.

Functions III

Example: converting km to miles and vice versa

```
ch_unit <- function(unit,dat){
  if (unit == "km") {
    for (i in 1:length(dat)) {
      dat[i] <- dat[i]*1.60934
    }
    return(dat)
  }
  if (unit == "mile") {
    # instead of a loop use
    return(dat*0.621371)
  }
  return("error")
}
```

The built-in function length() returns the length of a vector.

Functions IV

Two ways to load a completed function into the R-System

- enter the code into the console or
- save the function in a file with the extension .R and load it with the function source().
- For example, if you have saved the above function in the file ch_unit.R and saved it in the directory c:/myPrograms, then you would proceed as follows.

```
setwd(”c:/myPrograms”)
source(”ch_unit”)
```

Scripts

- textfile containing R commands you can enter in the R console, too.
- It is recommended to write and edit all of your R code in a script before you run it in the console. So you produce reproducible code and you can share it with others.
- To get better readable R scripts there exists several style guides like Google's R Style Guide: https://google.github.io/styleguide/Rguide.xml

Questions

Which of the following statements are true or false?

t	f	
□	□	The argument of an R function is always written in square brackets.
□	□	fun(x=y) assigns the value of the global variable y to the argument x within the function fun().
□	□	If (dat <- c(1,2234,23,2,3)) {med(dat)} returns 3.
□	□	If (dat <- c(1,2234,2,3)) {med(dat)} returns 3.
□	□	A script is a textfile containing executable R statements.

```
med <- function(x) {
sorted.x <- sort(x,decreasing=FALSE)
return(sorted.x[floor((length(x)+1)/2)])
}
```

Tidyverse I

compare https://www.tiq-solutions.de/tidyverse-ein-moderner-ansatz-fuer-datenanalysen-in-r/

- A modern approach to data analysis in R
- The large number of freely available libraries (called packages) offer the possibility to extend the functionality of Basis-R without having to implement complex algorithms and data structures.
- Many thousands of packages have been made available (and used intensively) to the community.
- In the field of data science, the packages **tidyverse** have emerged here.

Tidyverse Package II

- A collection of R-packages that unites a common view of the individual steps of data analysis and the central data structures.
- Extremely important is the so-called pipe operator %>%. By providing this simple (and actually not new) construct in R, a new coding style developed.
- Tidyverse is maintained and extended by a large number of developers. But Hadley Wickham, professor of statistics and meanwhile Chief Scientist at RStudio is the central figure in past und ongoing development of the package

Principles of Tidyverse Philosophy IV

- Reuse existing data structures
- Compose simple functions with the pipe
- Embrace functional programming
- Design for humans

Tidyverse Advantages and Benefits V

- Increased consistency between different packages
- Easy transferability: for newcomers the way to first successes in R was often a rocky one, because some “special features” of the programming language unnecessarily dampened the initial motivation. Tidyverse enables to implement complete data analyses quickly and intuitively.
- Tidyverse has lowered the entry threshold for own data analysis in R enormously.

Section 2

Datacleaning and -preparation in R

Importing a file I

- Existing data, which is stored in files with many different formats, can be read in directly.
- here: csv files (comma separated values)
- Reading in many other formats works in the same way.

Example: Import the file

```
Group;Name;Age
A;Schmidt;22
A;Olsen;18
B;Johansson;21
B;Mayer;22
```

Importing a file II

- base R command: read.csv()
- first argument is mandatory: name of the file
- second argument shows whether the first line contains variable names.
- third argument defines the separator between the values
- read.csv() reads the input file and creates a data frame.

Remark: Directly create a tibble by

- read_csv() for comma separated values
- read_csv2() for “;” separated values

The Pipe Operator $\% > \%$ I

- Pipe operator $\% > \%$ part of tidyverse package
- Advantage: ability to string multiple functions together by incorporating $\% > \%$
- Forwards a value, or the result of an expression, into the next function call/expression.
- If many functions are involved using the $\% > \%$ operator code becomes more efficient and more legible.

The Pipe Operator $\% > \%$ II

Example: Apply several function to filter some data, summarize it, and then order the summarized results.

- Nested Option:

```
arrange(
        summarize(
            filter(data, variable == numeric_value),
            Total = sum(variable)
        ),
    desc(Total)
)
```

- Multiple Object Option:

```
a <- filter(data, variable == numeric_value)
b <- summarise(a, Total = sum(variable))
c <- arrange(b, desc(Total))
```

- $\% > \%$ Option:

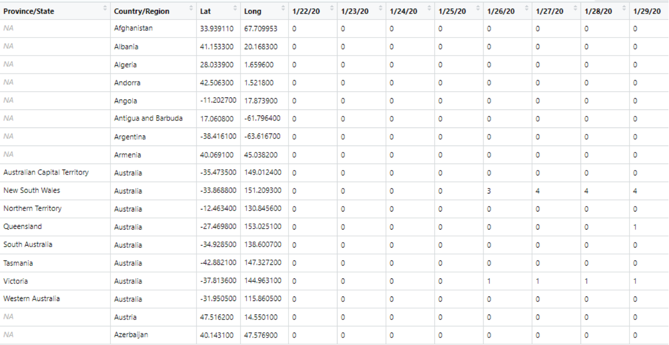
```
data %>%
       filter(variable == "value") %>%
       summarise(Total = sum(variable)) %>%
       arrange(desc(Total))
```

Tidy Data I

- Most time consuming challenge in doing statistical analysis: cleaning and preparing the data
- Raw data in many times: incorrect, redundant, inconsistent or incorrectly formatted data
- But even if the data is cleaned in many times it is not ready for analysis.

Tidy Data II

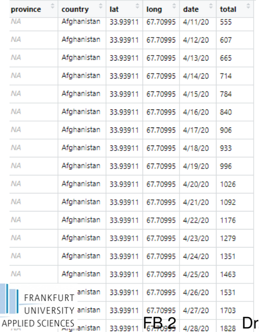
Example: COVID-19 data from John Hopkins University that consists of numbers of cases, ranging from confirmed, death, and recovered from countries and regions around the world.



Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	1/28/20	1/29/20
NA	Afghanistan	33.939110	67.709953	0	0	0	0	0	0	0	0
NA	Albania	41.153300	20.168300	0	0	0	0	0	0	0	0
NA	Algeria	28.033900	1.659600	0	0	0	0	0	0	0	0
NA	Andorra	42.506300	1.521800	0	0	0	0	0	0	0	0
NA	Angola	-11.202700	17.873900	0	0	0	0	0	0	0	0
NA	Antigua and Barbuda	17.060800	-61.796400	0	0	0	0	0	0	0	0
NA	Argentina	-38.416100	-63.616700	0	0	0	0	0	0	0	0
NA	Armenia	40.069100	45.038200	0	0	0	0	0	0	0	0
Australian Capital Territory	Australia	-35.473500	149.012400	0	0	0	0	0	0	0	0
New South Wales	Australia	-33.868800	151.209300	0	0	0	0	3	4	4	4
Northern Territory	Australia	-12.463400	130.845600	0	0	0	0	0	0	0	0
Queensland	Australia	-27.469800	153.025100	0	0	0	0	0	0	0	1
South Australia	Australia	-34.928500	138.600700	0	0	0	0	0	0	0	0
Tasmania	Australia	-42.882100	147.327200	0	0	0	0	0	0	0	0
Victoria	Australia	-37.813600	144.963100	0	0	0	0	1	1	1	1
Western Australia	Australia	-31.950500	115.860500	0	0	0	0	0	0	0	0
NA	Austria	47.516200	14.550100	0	0	0	0	0	0	0	0
NA	Azerbaijan	40.143100	47.576900	0	0	0	0	0	0	0	0

Tidy Data III

- Objective: Line plot for some nations
- x-axis = date, y-axis = value
- Here: every date is a column name and the column consists of the values of each country on that date
- To create the diagram, the values in these columns must be filtered by country for all date columns and the corresponding pairs must be plotted.
- This is much easier, if both the dates and the values are own columns, i.e. the data is tidy.



province	country	lat	long	date	total
NA	Afghanistan	33.93911	67.70995	4/11/20	555
NA	Afghanistan	33.93911	67.70995	4/12/20	607
NA	Afghanistan	33.93911	67.70995	4/13/20	665
NA	Afghanistan	33.93911	67.70995	4/14/20	714
NA	Afghanistan	33.93911	67.70995	4/15/20	784
NA	Afghanistan	33.93911	67.70995	4/16/20	840
NA	Afghanistan	33.93911	67.70995	4/17/20	906
NA	Afghanistan	33.93911	67.70995	4/18/20	933
NA	Afghanistan	33.93911	67.70995	4/19/20	996
NA	Afghanistan	33.93911	67.70995	4/20/20	1026
NA	Afghanistan	33.93911	67.70995	4/21/20	1092
NA	Afghanistan	33.93911	67.70995	4/22/20	1176
NA	Afghanistan	33.93911	67.70995	4/23/20	1279
NA	Afghanistan	33.93911	67.70995	4/24/20	1351
NA	Afghanistan	33.93911	67.70995	4/25/20	1463
[illegible]	anistan	33.93911	67.70995	4/26/20	1531
[illegible]	anistan	33.93911	67.70995	4/27/20	1703
[illegible]	Afghanistan	33.93911	67.70995	4/28/20	1828

Tidy Data IV

Hadley Wickham:

- Write down some conventions for the clean display/storage of data.
- Concept of “well-structured data” (“tidy data”) which provides a standardized way to link the structure of a dataset with its meaning. [3]
- Develops the package tidyverse: collection of commands for data tidying, data manipulation, data transformation and data visualisation.
- Grolemund and Wickham “R for Data Science” [4] offers extensive support on this topic.

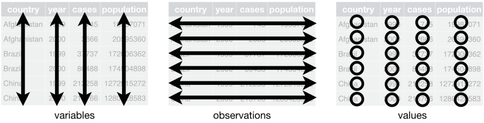
[3] Hadley Wickham: Tidy Data - RStudio: Journal of Statistical Software August 2014, Volume 59, Issue 10. http://www.jstatsoft.org/

[4] Hadley Wickham, Garrett Grolemund: R for Data Science, O’Reilly 2017, http://r4ds.had.co.nz

Tidy Data V

- Wickham : “A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types.”
- Tidy data:
  1. Each variable forms a column.
  2. Each observation forms a row.
  3. Each type of observational unit forms a table.
- Messy data is any other arrangement of the data.
- Wickham:
  - dataset a collection of values, usually either numbers (if quantitative) or strings (if qualitative)
  - values organised in two ways: Every value belongs to a variable and an observation. A variable contains all values that measure the same underlying attribute across units.
  - observation: all values measured on the same unit across attributes

Tidy Data VI



Wickham Grolemund, Figure 12.1

- Rules of a tidy dataset: variables are in columns, observations are in rows, and values are in cells
- At the beginning of data analysis: identify what are the variables are and what are the observations.
- typical problems
  - One variable might be spread across multiple columns.
  - One observation might be scattered across multiple rows.
- functions part of the tidyverse package for solving this problems

Tidy Data VII

Why ensure that data is tidy?

- In many times it is easier to describe functional relationships between columns (variables) than between rows, and it is easier to make comparisons between groups of rows (observations) than between groups of columns.
- There's a general advantage to picking one consistent way of storing data. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity. [5]
- There's a specific advantage to placing variables in columns because it allows R's vectorised nature to shine. ...most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural. [6]

[5] Wickham, Hadley, and Garrett Grolemund. 2016. R for Data Science. O'Reilly Media. https://r4ds.had.co.nz/.

[6] Wickham, Hadley, and Garrett Grolemund. 2016. R for Data Science. O'Reilly Media. https://r4ds.had.co.nz/.

Questions

Which of the following statements are true or false?

t	f	
□	□	A dataset is a collection of values
□	□	A variable contains all values that measure the same underlying attribute.
□	□	An observation consists of all values measured on the same unit across attributes
□	□	The dataset tab1 is tidy.
□	□	The dataset tab2 is messy since observations are spread across two rows.

```
tab1:
   country        1999          2000          2001
   chr            int           int           int
   Afghanistan    19987071      20595360      21347782
   ...
tab2:
   country        year     count         type
   chr            int      int           chr
   Afghanistan    1997     19021226      population
   Afghanistan    1997     128           cases ...
```

Tidying data

compare Wickham Grolemund, chapter 12

Most common problems in tidying data:

- One variable is spread across multiple columns. → gather()
- One observation is scattered across multiple rows. → spread()
- Values of more than one variable are contained in one column. → separate()

Typically a dataset will only suffer from one of these problems.

Tidying data gather()

Column names are not names of variables, but values of a variable

```
> head(table4a)
# A tibble: 6 x 4
  country      `1999`   `2000`   `2001`
  <chr>         <int>    <int>    <int>
1 Afghanistan 19987071 20595360 21347782
2 Albania      3317941  3304948  3286084
...
```

- Columns representing values, not variables: 1999, 2000 and 2001 are values of a variable year.
- Name of the variable (key) whose values form the column names: year.
- Name of the variable (value) whose values are spread over the cells: cases

```
> table4a %>% gather('1999','2000','2001',key="year",value="cases") %>%
+   arrange(country, year) %>% head
# A tibble: 6 x 3
  country     year     cases
  <chr>       <chr>    <int>
1 Afghanistan 1999  19987071
2 Afghanistan 2000  20595360
3 Afghanistan 2001  21347782
4 Albania     1999   3317941
5 Albania     2000   3304948
6 Albania     2001   3286084
```

Tidying data spread()

An observation is scattered across multiple rows.

```
> head(table2)
# A tibble: 6 x 4
  country      year    count type
  <chr>       <int>    <int> <chr>
1 Afghanistan  1997 19021226 population
2 Afghanistan  1997      128 cases
3 Afghanistan  1998 19496836 population
4 Afghanistan  1998     1778 cases
5 Afghanistan  1999 19987071 population
6 Afghanistan  1999      745 cases
```

- Column containing variable names (key column): type.
- Column containing values from multiple variables: count.

```
> table2 %>% spread(key = type, value = count) %>% head
# A tibble: 6 x 4
  country      year cases population
  <chr>       <int> <int>      <int>
1 Afghanistan  1997   128   19021226
2 Afghanistan  1998  1778   19496836
3 Afghanistan  1999   745   19987071
4 Afghanistan  2000  2666   20595360
5 Afghanistan  2001  4639   21347782
6 Afghanistan  2002  6509   22202806
```

Tidying data seperate()

One column containing values of two variables

```
> head(table3)
# A tibble: 6 x 3
  country      year rate
  <chr>       <int> <chr>
1 Afghanistan  1997 128/19021226
2 Afghanistan  1998 1778/19496836
3 Afghanistan  1999 745/19987071
4 Afghanistan  2000 2666/20595360
5 Afghanistan  2001 4639/21347782
6 Afghanistan  2002 6509/22202806
```

The column rate contains values of the variables cases and population.

```
> table3 %>% separate(rate,into=c("cases","population"),sep ="/") %>%
+   head
# A tibble: 6 x 4
  country      year cases population
  <chr>       <int> <chr> <chr>
1 Afghanistan  1997 128   19021226
2 Afghanistan  1998 1778  19496836
3 Afghanistan  1999 745   19987071
4 Afghanistan  2000 2666  20595360
5 Afghanistan  2001 4639  21347782
6 Afghanistan  2002 6509  22202806
```

Datamanipulations

compare Wickham Grolemund, chapter 5

- The package dplyr (part of the tidyverse package) contains several functions to manipulate data.

To show the use of these functions we will use the dataset flights of nycflights13 package. On-time data for all flights that departed NYC in 2013.

```
> flights
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay
   <int> <int> <int>    <int>          <int>     <dbl>
 1  2013     1     1      517            515         2
 2  2013     1     1      533            529         4
 3  2013     1     1      542            540         2
 4  2013     1     1      544            545        -1
 5  2013     1     1      554            600        -6
 6  2013     1     1      554            558        -4
 7  2013     1     1      555            600        -5
 8  2013     1     1      557            600        -3
 9  2013     1     1      557            600        -3
10  2013     1     1      558            600        -2
# ... with 336,766 more rows, and 13 more variables:
#   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

Datamanipulations filter()

Pick observations by their values: filter()

- Allows you to subset observations based on their values.
- First argument; name of the data frame
- Second argument: subsequent arguments are the expressions that filter the data frame.
  all flights with dayy = 10 and month = 5

  ```
  flights %>% filter(day == 10, month == 5)
  ```
- Using comparisons with $>, >=, <, <=, ! =, ==$ one can construct more complex subsets.

  ```
  flights %>% filter(month == 5 | month == 6, dep_time == 9)
  ```

  filters all flights in month 5 or month 6 with dep_time = 9

  ```
  flights %>% filter( month %in% c(3,4,10), dep_time == 9)
  ```

  filters all flights in the month 3, 4 and 10 with dep_time = 9

Datamanipulations arrange()

Reorder rows of the data by value of given columns: arrange()

```
arrange(flights, desc(dep_time), dep_delay)
```

Sort the flights according to dep_time and dep_delay in descending order with dep_time.

Datamanipulations select()

Pick variables by their names: select()

- select the columns carrier and flight

  ```
  flights %>%
      select(carrier,flight) %>% unique
  ```

  unique() remove all duplicate entries.

- Select all columns except those from dep_time to air_time (inclusive)

  ```
  flights %>% select(-(dep_time:air_time))
  ```

  Without “-” you will get all columns between dep_time and air_time (inclusive).

- A number of helper functions can be used within select(). Use ?select to get more informations.

Datamanipulations mutate()

Create new variables with functions of existing variables: mutate()

```
flights %>%
  select(year:day, distance, air_time) %>%
  mutate(hours = air_time / 60,
         speed = distance / hours)
```

Selects the columns year to day, distance and air_time and the columns hours and speed and adds the columns hours and speed

Datamanipulations summarise()

Collapse many values down to a single summary: summarise()

-
  ```
  flights %>%
    summarise(mean_delay=mean(dep_delay,na.rm=TRUE),
              med_dely=median(dep_delay,na.rm=TRUE))
  # A tibble: 1 x 2
    mean_delay med_dely
         <dbl>    <dbl>
  1       12.6       -2
  ```

  Computes the mean and the median of dep_delay.
- group_by() command groups the calculation by values of given columns.

  ```
  flights %>%
  +   group_by(carrier, flight) %>%
  +   summarise(mean_delay=mean(dep_delay,na.rm=TRUE),
  +             med_dely=median(dep_delay,na.rm=TRUE))
  ```

  gives the mean and median delay for every carrier and flight.
- By na.rm = TRUE na.rm the NA values are stripped before the computation.

Joining Data

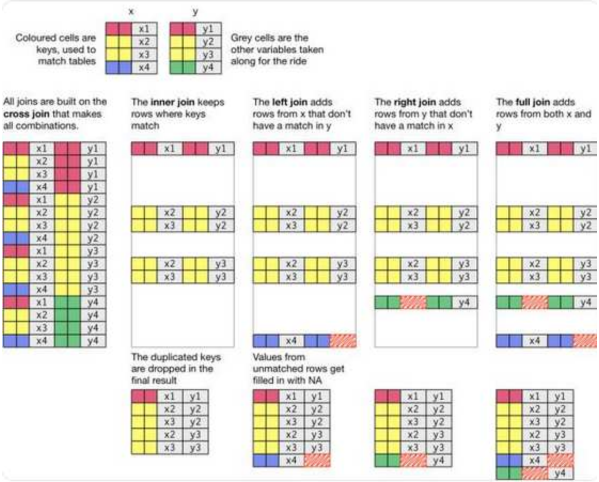
compare Wickham Grolemund, chapter 13

- Combine data spread over several tables
- Corresponding to SQL joins different possibilities to combine tables via joins.
  - inner_join(x, y, by = NULL, ...)
  - left_join(x, y, by = NULL, ...)
  - right_join(x, y, by = NULL, ...)
  - full_join(x, y, by = NULL, ...)

  Arguments:

  x, y: tbls to join
  - by: a character vector of variables to join by. If NULL, the default, _join() will do a natural join, using all variables with common names across the two tables. To join by different variables on x and y use a named vector. For example, by = c(“a” = “b”) will match x.a to y.b.
  - ...: other parameters

Joining Data Overview



Coloured cells are keys, used to match tables

Grey cells are the other variables taken along for the ride

All joins are built on the **cross join** that makes all combinations.

The **inner join** keeps rows where keys match

The **left join** adds rows from x that don't have a match in y

The **right join** adds rows from y that don't have a match in x

The **full join** adds rows from both x and y

The duplicated keys are dropped in the final result

Values from unmatched rows get filled in with NA

Source: Mara Averick https://twitter.com/dataandme/status/723221339704799233

Joining Data Example I

```
# A tibble: 3 x 2
    key val_x
  <dbl> <chr>
1     1 x1
2     2 x2
3     3 x3
```

```
y
# A tibble: 3 x 2
    key val_y
  <dbl> <chr>
1     1 y1
2     2 y2
3     4 y3
```

inner_join: only rows with matching keys in both x and y

```
> x %>% inner_join(y,by = "key")
# A tibble: 2 x 3
    key val_x val_y
  <dbl> <chr> <chr>
1     1 x1    y1
2     2 x2    y2
```

left_join: all rows in x, adding matching columns from y

```
> x %>% left_join(y,by = "key")
# A tibble: 3 x 3
    key val_x val_y
  <dbl> <chr> <chr>
1     1 x1    y1
2     2 x2    y2
3     3 x3    NA
```

right_join: all rows in y, adding matching columns from x

```
> x %>% right_join(y,by = "key")
# A tibble: 3 x 3
    key val_x val_y
  <dbl> <chr> <chr>
1     1 x1    y1
2     2 x2    y2
3     4 NA    y4
```

full_join: all rows in x with matching columns in y, then the rows of y that don’t match x

```
> x %>% full_join(y,by = "key")
# A tibble: 4 x 3
    key val_x val_y
  <dbl> <chr> <chr>
1     1 x1    y1
2     2 x2    y2
3     3 x3    NA
4     4 NA    y4
```

Joining Data Example II

If the values in the key column are not unique a join of the table results in a cartesian product of all the values in the key columns.

```
> x
# A tibble: 4 x 2
    key val_x
  <dbl> <chr>
1     1 x1
2     2 x2
3     2 x3
4     4 x4
```

```
> y
# A tibble: 4 x 2
    key val_y
  <dbl> <chr>
1     1 y1
2     2 y2
3     2 y3
4     3 y4
```

```
> x %>% left_join(y,by = "key")
# A tibble: 6 x 3
    key val_x val_y
  <dbl> <chr> <chr>
1     1 x1    y1
2     2 x2    y2
3     2 x2    y3
4     2 x3    y2
5     2 x3    y3
6     3 x4    y4
```

Questions

Which of the following statements are true or false?

t	f	
□	□	filter(profiles, sex == “f”) filters the records from the data set profile where the variable sex has the value f.
□	□	If you want to select the columns sp1 and sp2 from the data set df, you can do this with “select(df, sp1, sp2)”.
□	□	filter() allows only one filter criterion.
□	□	summarise() can only be used if one column should summarised to one value.
□	□	With left_join(), only rows that are in the second data frame are retained, provided they have a match in the first record.
□	□	The statement below create a data frame with the columns passed, score and top. top has only the value false and true, where true denotes the students with score $>$ 80.

```
results %>%
  select(passed, score) %>%
  mutate(top = score > 80)
```

Content

- Introduction to R
  - What is R?
  - Why R?
  - RStudio
  - R as a Programming Language
    - Variables and Assignments
    - Data Structures
    - Functions
    - Scripts
  - Tidyverse

- Datacleaning and -preparation in R
  - Importing a file
  - The Pipe Operator % > %
  - Tidy Data
  - Tidying data
  - Datamanipulations
  - Joining Data