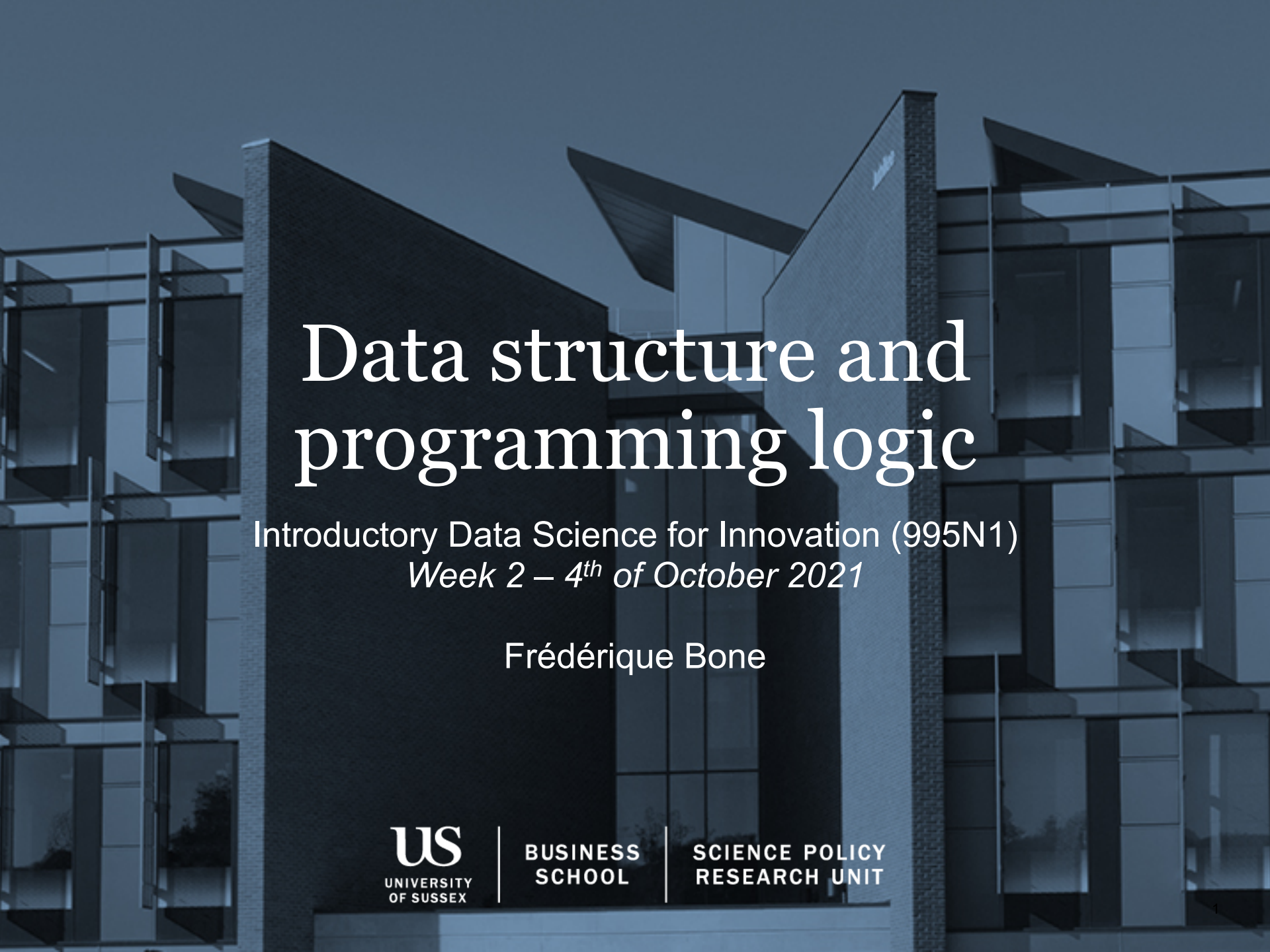


Data structure and programming logic

Introductory Data Science for Innovation (995N1)

Week 2 – 4th of October 2021

Frédérique Bone

UNIVERSITY OF SUSSEX | BUSINESS SCHOOL | SCIENCE POLICY RESEARCH UNIT

Group presentations

Don’t forget to add your groups in the google doc by next week (Week 3).

→ See assessment information page

Change of structure:

- Week 3 will be data visualisation

Session objectives

Bases of data science:

This session will present you with the **foundational toolbox** to get started with Data Science.

This session introduces many concepts; **it is aimed to be a resource** that you can build on for any of your future project.

I encourage you to open R or Rstudio to reproduce the code.

Practical objectives

Bases of data science:

1) Data types and structure
Is there a better data structure to analyse data?

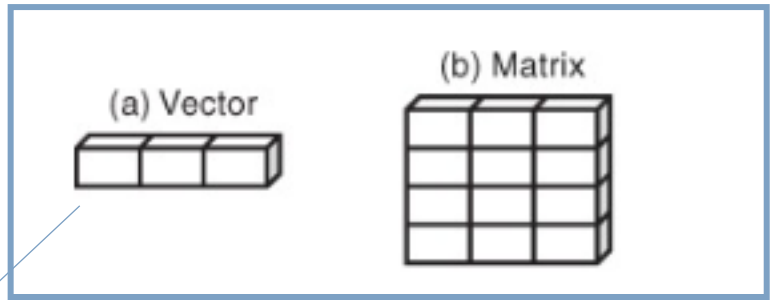
2) General programming logic
What are the general operators to give commands to a computer?

Data types and structure

UNIVERSITY OF SUSSEX | BUSINESS SCHOOL | SCIENCE POLICY RESEARCH UNIT

Data semantics

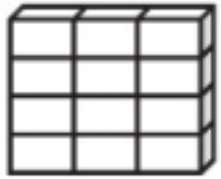
Most common R data structures that you may encounter are:



All the data have to be the same type of data

```
a <- c(1, 2, 3, 4, -1, 6)
b <- c("one", "two", "three")
```

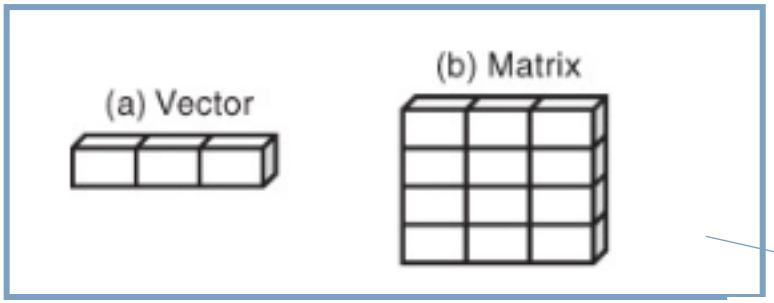
(d) Data frame



Columns can be different modes

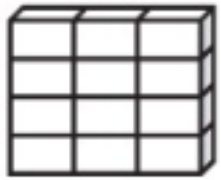
Data semantics

Most common R data structures that you may encounter are:



All the data have to be the same type of data

$$A = \begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$



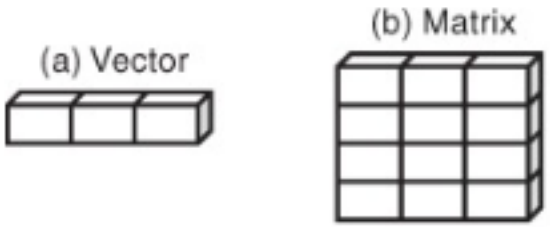
Columns can be different m

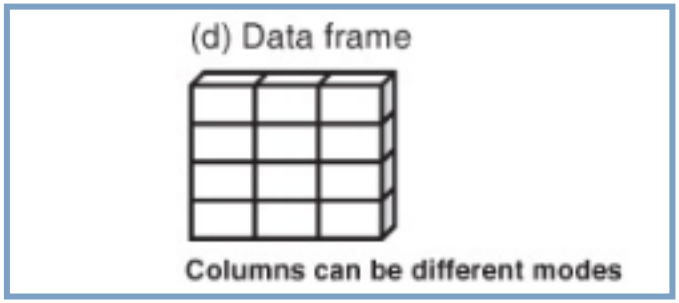
```
> A = matrix(
+   c(2, 4, 3, 1, 5, 7), # the data elements
+   nrow=2,              # number of rows
+   ncol=3,              # number of columns
+   byrow = TRUE)        # fill matrix by rows

> A                      # print the matrix
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

http://www.r-tutor.com/r-introduction/matrix

Data semantics

Most common R data structures that you may encounter are:



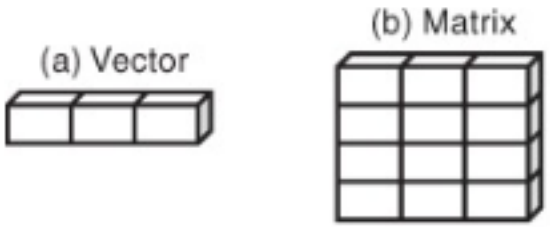


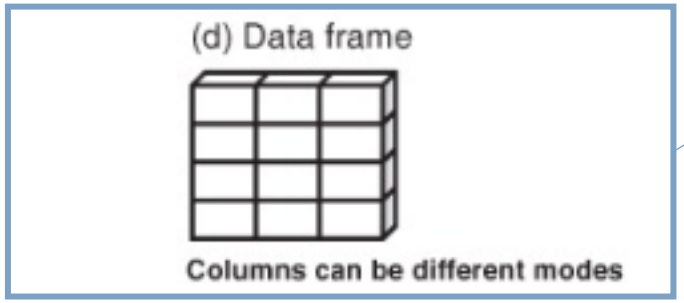
Data within columns have to be of the same type

Columns can be of different types.

Data semantics

Most common R data structures that you may encounter





Data from seminar 1: Titanic and Iris

Seminar1 – Script.R | Titanic

Filter

	Class	Sex	Age	Survived	Freq
1	1st	Male	Child	No	0
2	2nd	Male	Child	No	0
3	3rd	Male	Child	No	35
4	Crew	Male	Child	No	0
5	1st	Female	Child	No	0
6	2nd	Female	Child	No	0
7	3rd	Female	Child	No	17
8	Crew	Female	Child	No	0
9	1st	Male	Adult	No	118
10	2nd	Male	Adult	No	154
11	3rd	Male	Adult	No	387
12	Crew	Male	Adult	No	670
13	1st	Female	Adult	No	4
14	2nd	Female	Adult	No	13
15	3rd	Female	Adult	No	89
16	Crew	Female	Adult	No	3
17	1st	Male	Child	Yes	5
18	2nd	Male	Child	Yes	11
19	3rd	Male	Child	Yes	13
20	Crew	Male	Child	Yes	0

What are the main data types that we will encounter?

UNIVERSITY OF SUSSEX | BUSINESS SCHOOL

Data types

What types of variables are present?

Seminar1 – Script.R × Titanic ×

Filter

	Class	Sex	Age	Survived	Freq
1	1st	Male	Child	No	0
2	2nd	Male	Child	No	0
3	3rd	Male	Child	No	35
4	Crew	Male	Child	No	0
5	1st	Female	Child	No	0
6	2nd	Female	Child	No	0
7	3rd	Female	Child	No	17
8	Crew	Female	Child	No	0
9	1st	Male	Adult	No	118
10	2nd	Male	Adult	No	154
11	3rd	Male	Adult	No	387
12	Crew	Male	Adult	No	670
13	1st	Female	Adult	No	4
14	2nd	Female	Adult	No	13
15	3rd	Female	Adult	No	89
16	Crew	Female	Adult	No	3
17	1st	Male	Child	Yes	5
18	2nd	Male	Child	Yes	11
19	3rd	Male	Child	Yes	13
20	Crew	Male	Child	Yes	0

What are the main types of data?

Most common R data structures that you may encounter are:

- **Logical** → only two possible values

```
TRUE   FALSE
```

- **Numeric** → this is any type of number (two main types integer: whole numbers, double: decimal number)

```
1   2   3   16   257
```

- **Character** → text data, defined in "quotation marks".

```
"Hello" "3 or three" "anything you want to write"
```

- **Missing data** have also a representation:
  NA (not available) → is there where there may be missing values
  NaN (not a number) → impossible number like when divided by 0

What are the main types of data?

Most common R data structures that you may encounter:

- **Factor** → Looks like textual data,
  but we want the data to be grouped by value.

```
"Hello"  "Bonjour" "Ciao" "Bonjour"
```

Recognised as the same value

- **Date** → Looks like numeric data,
  but we want the data to be recognised as date format.

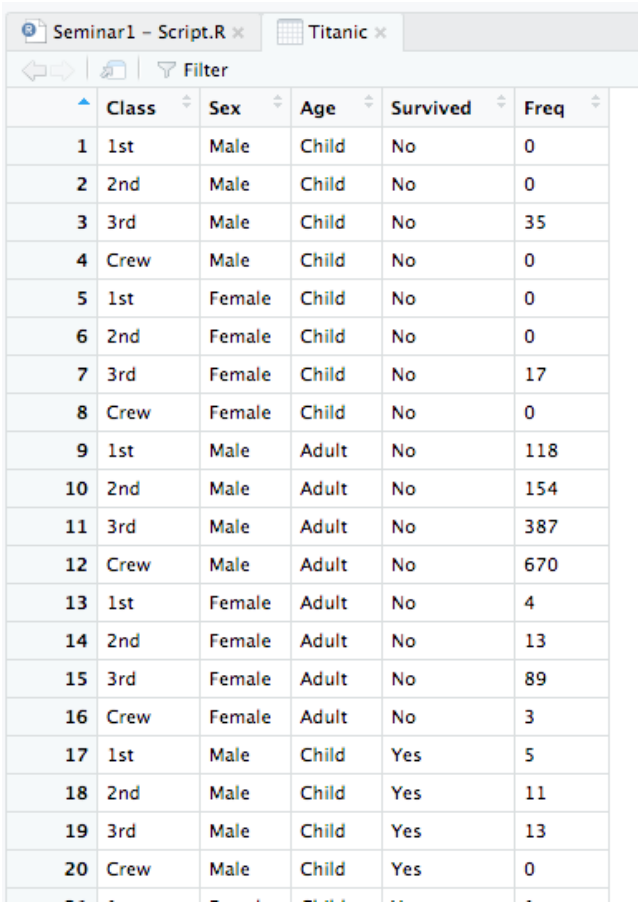
R default format for dates is **yyyy-mm-dd**

```
> # print today's date
> today <- Sys.Date()
> print(today)
[1] "2020-01-30"
>
> # Change the format of the day
> format(today, format="%d of %B %Y")
[1] "30 of January 2020"
```

Symbol	Meaning	Example
%d	day as a number (0-31)	01-31
%a %A	abbreviated weekday unabbreviated weekday	Mon Monday
%m	month (00-12)	00-12
%b %B	abbreviated month unabbreviated month	Jan January
%y %Y	2-digit year 4-digit year	07 2007

Data types

What types could we alter?


Seminar1 – Script.R × Titanic ×

Filter

	Class	Sex	Age	Survived	Freq
1	1st	Male	Child	No	0
2	2nd	Male	Child	No	0
3	3rd	Male	Child	No	35
4	Crew	Male	Child	No	0
5	1st	Female	Child	No	0
6	2nd	Female	Child	No	0
7	3rd	Female	Child	No	17
8	Crew	Female	Child	No	0
9	1st	Male	Adult	No	118
10	2nd	Male	Adult	No	154
11	3rd	Male	Adult	No	387
12	Crew	Male	Adult	No	670
13	1st	Female	Adult	No	4
14	2nd	Female	Adult	No	13
15	3rd	Female	Adult	No	89
16	Crew	Female	Adult	No	3
17	1st	Male	Child	Yes	5
18	2nd	Male	Child	Yes	11
19	3rd	Male	Child	Yes	13
20	Crew	Male	Child	Yes	0

Data semantics

Converting types:

It may be desirable to transform one type of data into another:

→ to be able to order them

→ to allow certain types of operation

Use operator to change the type of variable:

as.numeric(x), as.character(x), factor(x)…

```
> # create a vector
> x <- c(1, 2, 3, 4)
>
> # check its data type
> class(x)
[1] "numeric"
> print(x)
[1] 1 2 3 4
>
> # change x to text
> x <- as.character(x)
> print(x)
[1] "1" "2" "3" "4"
```

Data semantics

What would the result of this be?

```
x <- c("1", "2", "3", "4")

x+x
```

```
> x <- c("1", "2", "3", "4")
>
> x+x
Error in x + x : non-numeric argument to binary operator
```

Data semantics

What would the result of this be?

```
x <- c("1", "2", "3", "4")
x <- as.numeric(x)
x+x
```

```
[1] 2 4 6 8
```

Data types

It is important to understand the type of the data:

- To know the kind of operation that can be done with them
- Avoid generating errors or NaNs when transforming data

→ Check the data type with class() to see whether some operations are possible.

→ If the data is the wrong type, tell R how you want your data to be considered.

Datasets (i.e. data frames)

Is there a best way to organise them?

UNIVERSITY OF SUSSEX | BUSINESS SCHOOL | SCIENCE POLICY RESEARCH UNIT

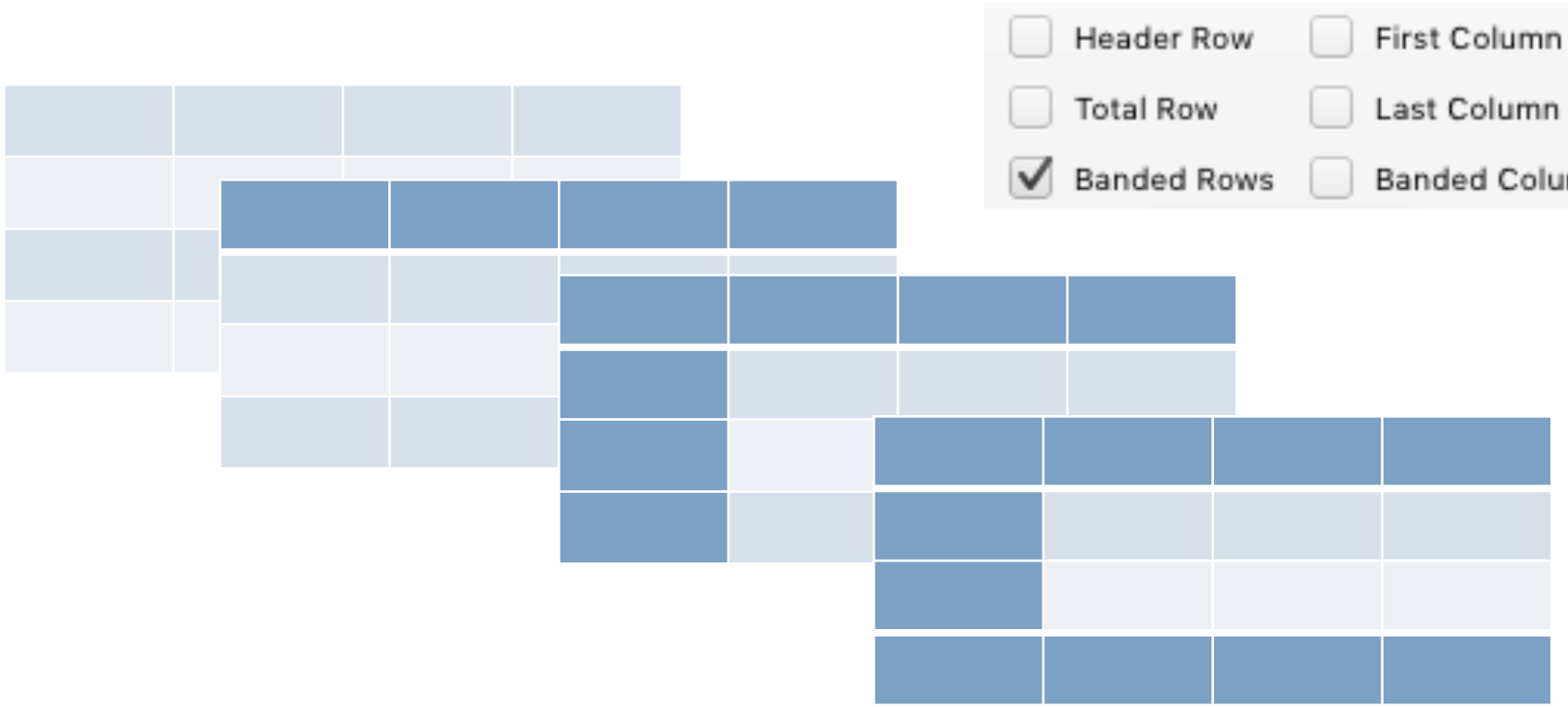
What makes a dataset tidy?

Wickham (2014), “Tidy data”

UNIVERSITY OF SUSSEX | BUSINESS SCHOOL

Data structure

Datasets in most cases are represented in rectangular tables,
With a number of n rows and m columns



Data structure

Datasets represents:

- a set of values
- from a specific observation
- looking at certain characteristics (i.e. variables)

Data structure

Datasets represents:

- a set of values
- from a specific observation
- looking at certain characteristics (i.e. variables)

Tidy data:

- Each observation → represented in a single row
- Each variable → represented in a column
- Values of each variable for each observation is contained in the remainder of the table

→ These are **data frames** (see above)

Data structure

Seminar1 – Script.R × | Titanic ×

Filter

	Class	Sex	Age	Survived	Freq
1	1st	Male	Child	No	0
2	2nd	Male	Child	No	0
3	3rd	Male	Child	No	35
4	Crew	Male	Child	No	0
5	1st	Female	Child	No	0
6	2nd	Female	Child	No	0
7	3rd	Female	Child	No	17
8	Crew	Female	Child	No	0
9	1st	Male	Adult	No	118
10	2nd	Male	Adult	No	154
11	3rd	Male	Adult	No	387
12	Crew	Male	Adult	No	670
13	1st	Female	Adult	No	4
14	2nd	Female	Adult	No	13
15	3rd	Female	Adult	No	89
16	Crew	Female	Adult	No	3
17	1st	Male	Child	Yes	5
18	2nd	Male	Child	Yes	11
19	3rd	Male	Child	Yes	13
20	Crew	Male	Child	Yes	0

Data structure

Observations

Seminar1 – Script.R | Titanic

Filter

	Class	Sex	Age	Survived	Freq
1	1st	Male	Child	No	0
2	2nd	Male	Child	No	0
3	3rd	Male	Child	No	35
4	Crew	Male	Child	No	0
5	1st	Female	Child	No	0
6	2nd	Female	Child	No	0
7	3rd	Female	Child	No	17
8	Crew	Female	Child	No	0
9	1st	Male	Adult	No	118
10	2nd	Male	Adult	No	154
11	3rd	Male	Adult	No	387
12	Crew	Male	Adult	No	670
13	1st	Female	Adult	No	4
14	2nd	Female	Adult	No	13
15	3rd	Female	Adult	No	89
16	Crew	Female	Adult	No	3
17	1st	Male	Child	Yes	5
18	2nd	Male	Child	Yes	11
19	3rd	Male	Child	Yes	13
20	Crew	Male	Child	Yes	0

Data structure

Seminar1 – Script.R | Titanic

Filter

	Class	Sex	Age	Survived	Freq
1	1st	Male	Child	No	0
2	2nd	Male	Child	No	0
3	3rd	Male	Child	No	35
4	Crew	Male	Child	No	0
5	1st	Female	Child	No	0
6	2nd	Female	Child	No	0
7	3rd	Female	Child	No	17
8	Crew	Female	Child	No	0
9	1st	Male	Adult	No	118
10	2nd	Male	Adult	No	154
11	3rd	Male	Adult	No	387
12	Crew	Male	Adult	No	670
13	1st	Female	Adult	No	4
14	2nd	Female	Adult	No	13
15	3rd	Female	Adult	No	89
16	Crew	Female	Adult	No	3
17	1st	Male	Child	Yes	5
18	2nd	Male	Child	Yes	11
19	3rd	Male	Child	Yes	13
20	Crew	Male	Child	Yes	0

Variables : Dimensions of interest

Data structure

Seminar1 – Script.R | Titanic

Filter

	Class	Sex	Age	Survived	Freq
1	1st	Male	Child	No	0
2	2nd	Male	Child	No	0
3	3rd	Male	Child	No	35
4	Crew	Male	Child	No	0
5	1st	Female	Child	No	0
6	2nd	Female	Child	No	0
7	3rd	Female	Child	No	17
8	Crew	Female	Child	No	0
9	1st	Male	Adult	No	118
10	2nd	Male	Adult	No	154
11	3rd	Male	Adult	No	387
12	Crew	Male	Adult	No	670
13	1st	Female	Adult	No	4
14	2nd	Female	Adult	No	13
15	3rd	Female	Adult	No	89
16	Crew	Female	Adult	No	3
17	1st	Male	Child	Yes	5
18	2nd	Male	Child	Yes	11
19	3rd	Male	Child	Yes	13
20	Crew	Male	Child	Yes	0

Values

Data structure

Seminar1 – Script.R | Titanic

Filter

	Class	Sex	Age	Survived	Freq
1	1st	Male	Child	No	0
2	2nd	Male	Child	No	0
3	3rd	Male	Child	No	35
4	Crew	Male	Child	No	0
5	1st	Female	Child	No	0
6	2nd	Female	Child	No	0
7	3rd	Female	Child	No	17
8	Crew	Female	Child	No	0
9	1st	Male	Adult	No	118
10	2nd	Male	Adult	No	154
11	3rd	Male	Adult	No	387
12	Crew	Male	Adult	No	670
13	1st	Female	Adult	No	4
14	2nd	Female	Adult	No	13
15	3rd	Female	Adult	No	89
16	Crew	Female	Adult	No	3
17	1st	Male	Child	Yes	5
18	2nd	Male	Child	Yes	11
19	3rd	Male	Child	Yes	13
20	Crew	Male	Child	Yes	0

There are no (Freq)
first class (Class)
male (Sex)
child (Age)
who has died (Survived).

Let's Practice!

UNIVERSITY OF SUSSEX | BUSINESS SCHOOL

Let's start!

Load the Script of this week:

Understand the dataset

REPLACE _ _ by the appropriate value/function in the code

The exercise / questions are available in the comments (e.g. after the # in the Rscript)

#----------------------------------
 # 1. Explore the given dataset
#----------------------------------

Datasets can be untidy in many different ways...

Wickham (2014), "Tidy data"

UNIVERSITY OF SUSSEX | BUSINESS SCHOOL

Data structure

(1) Column headers are values, not variables

religion	<$10k	$10–20k	$20–30k	$30–40k	$40–50k	$50–75k
Agnostic	27	34	60	81	76	137
Atheist	12	27	37	52	35	70
Buddhist	27	21	30	34	33	58
Catholic	418	617	732	670	638	1116
Don't know/refused	15	14	15	11	10	35
Evangelical Prot	575	869	1064	982	881	1486
Hindu	1	9	7	9	11	34
Historically Black Prot	228	244	236	238	197	223
Jehovah's Witness	20	27	24	24	21	30
Jewish	19	19	25	25	30	95

Source: Wickham (2014), "Tidy data", Journal of Statistical Software, 59(10)

Data structure

(1) Column headers are values, not variables (tidy)

row	a	b	c
A	1	4	7
B	2	5	8
C	3	6	9

(a) Raw data

row	column	value
A	a	1
B	a	2
C	a	3
A	b	4
B	b	5
C	b	6
A	c	7
B	c	8
C	c	9

(b) Molten data

Source: Wickham (2014), “Tidy data”, Journal of Statistical Software, 59(10)

Data structure

(2) Multiple variables are stored in one column

country	year	column	cases
AD	2000	m014	0
AD	2000	m1524	0
AD	2000	m2534	1
AD	2000	m3544	0
AD	2000	m4554	0
AD	2000	m5564	0
AD	2000	m65	0
AE	2000	m014	2
AE	2000	m1524	4
AE	2000	m2534	4
AE	2000	m3544	6
AE	2000	m4554	5
AE	2000	m5564	12
AE	2000	m65	10
AE	2000	f014	3

Source: Wickham (2014), “Tidy data”, Journal of Statistical Software, 59(10)

Data structure

(2) Multiple variables are stored in one column

country	year	column	cases
AD	2000	m014	0
AD	2000	m1524	0
AD	2000	m2534	1
AD	2000	m3544	0
AD	2000	m4554	0
AD	2000	m5564	0
AD	2000	m65	0
AE	2000	m014	2
AE	2000	m1524	4
AE	2000	m2534	4
AE	2000	m3544	6
AE	2000	m4554	5
AE	2000	m5564	12
AE	2000	m65	10
AE	2000	f014	3

Sex and Age are included in this column.

Source: Wickham (2014), “Tidy data”, Journal of Statistical Software, 59(10)

Data structure

(2) Multiple variables are stored in one column (tidy)

country	year	sex	age	cases
AD	2000	m	0–14	0
AD	2000	m	15–24	0
AD	2000	m	25–34	1
AD	2000	m	35–44	0
AD	2000	m	45–54	0
AD	2000	m	55–64	0
AD	2000	m	65+	0
AE	2000	m	0–14	2
AE	2000	m	15–24	4
AE	2000	m	25–34	4
AE	2000	m	35–44	6
AE	2000	m	45–54	5
AE	2000	m	55–64	12
AE	2000	m	65+	10
AE	2000	f	0-14	3

Source: Wickham (2014), “Tidy data”, Journal of Statistical Software, 59(10)

Data structure

(3) Variables are stored in both rows and columns

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
MX17004	2010	1	tmax	—	—	—	—	—	—	—	—
MX17004	2010	1	tmin	—	—	—	—	—	—	—	—
MX17004	2010	2	tmax	—	27.3	24.1	—	—	—	—	—
MX17004	2010	2	tmin	—	14.4	14.4	—	—	—	—	—
MX17004	2010	3	tmax	—	—	—	—	32.1	—	—	—
MX17004	2010	3	tmin	—	—	—	—	14.2	—	—	—
MX17004	2010	4	tmax	—	—	—	—	—	—	—	—
MX17004	2010	4	tmin	—	—	—	—	—	—	—	—
MX17004	2010	5	tmax	—	—	—	—	—	—	—	—
MX17004	2010	5	tmin	—	—	—	—	—	—	—	—

Source: Wickham (2014), “Tidy data”, Journal of Statistical Software, 59(10)

Data structure

(3) Variables are stored in both rows and columns

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
MX17004	2010	1	tmax								
MX17004	2010	1	tmin	—	—	—	—	—	—	—	—
MX17004	2010	2	tmax	—	27.3	24.1	—	—	—	—	—
MX17004	2010	2	tmin	—	14.4	14.4	—	—	—	—	—
MX17004	2010	3	tmax	—	—	—	—	32.1	—	—	—
MX17004	2010	3	tmin	—	—	—	—	14.2	—	—	—
MX17004	2010	4	tmax	—	—	—	—	—	—	—	—
MX17004	2010	4	tmin	—	—	—	—	—	—	—	—
MX17004	2010	5	tmax	—	—	—	—	—	—	—	—
MX17004	2010	5	tmin	—	—	—	—	—	—	—	—

These could be considered as values.

Source: Wickham (2014), “Tidy data”, Journal of Statistical Software, 59(10)

Data structure

(3) Variables are stored in both rows and columns

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
MX17004	2010	1	tmax	—	—	—	—	—	—	—	—
MX17004	2010	1	tmin	—	—	—	—	—	—	—	—
MX17004	2010	2	tmax	—	27.3	24.1	—	—	—	—	—
MX17004	2010	2	tmin	—	14.4	14.4	—	—	—	—	—
MX17004	2010	3	tmax	—	—	—	—	32.1	—	—	—
MX17004	2010	3	tmin	—	—	—	—	14.2	—	—	—
MX17004	2010	4	tmax	—	—	—	—	—	—	—	—
MX17004	2010	4	tmin	—	—	—	—	—	—	—	—
MX17004	2010	5	tmax	—	—	—	—	—	—	—	—
MX17004	2010	5	tmin	—	—	—	—	—	—	—	—

These could be considered as variable.

Source: Wickham (2014), “Tidy data”, Journal of Statistical Software, 59(10)

Data structure

(3) Variables are stored in both rows and columns (tidy)

id	date	tmax	tmin
MX17004	2010-01-30	27.8	14.5
MX17004	2010-02-02	27.3	14.4
MX17004	2010-02-03	24.1	14.4
MX17004	2010-02-11	29.7	13.4
MX17004	2010-02-23	29.9	10.7
MX17004	2010-03-05	32.1	14.2
MX17004	2010-03-10	34.5	16.8
MX17004	2010-03-16	31.1	17.6
MX17004	2010-04-27	36.3	16.7
MX17004	2010-05-27	33.2	18.2

(b) Tidy data

Source: Wickham (2014), “Tidy data”, Journal of Statistical Software, 59(10)

Data structure

(4) Multiple types in one table

year	artist	time	track	date	week	rank
2000	2 Pac	4:22	Baby Don't Cry	2000-02-26	1	87
2000	2 Pac	4:22	Baby Don't Cry	2000-03-04	2	82
2000	2 Pac	4:22	Baby Don't Cry	2000-03-11	3	72
2000	2 Pac	4:22	Baby Don't Cry	2000-03-18	4	77
2000	2 Pac	4:22	Baby Don't Cry	2000-03-25	5	87
2000	2 Pac	4:22	Baby Don't Cry	2000-04-01	6	94
2000	2 Pac	4:22	Baby Don't Cry	2000-04-08	7	99
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-02	1	91
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-09	2	87
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-16	3	92
2000	3 Doors Down	3:53	Kryptonite	2000-04-08	1	81
2000	3 Doors Down	3:53	Kryptonite	2000-04-15	2	70
2000	3 Doors Down	3:53	Kryptonite	2000-04-22	3	68
2000	3 Doors Down	3:53	Kryptonite	2000-04-29	4	67
2000	3 Doors Down	3:53	Kryptonite	2000-05-06	5	66

Source: Wickham (2014), "Tidy data", Journal of Statistical Software, 59(10)

Data structure

Multiple types in one table

Table 1

year	artist	time	track	date	week	rank
2000	2 Pac	4:22	Baby Don't Cry	2000-02-26	1	87
2000	2 Pac	4:22	Baby Don't Cry	2000-03-04	2	82
2000	2 Pac	4:22	Baby Don't Cry	2000-03-11	3	72
2000	2 Pac	4:22	Baby Don't Cry	2000-03-18	4	77
2000	2 Pac	4:22	Baby Don't Cry	2000-03-25	5	87
2000	2 Pac	4:22	Baby Don't Cry	2000-04-01	6	94
2000	2 Pac	4:22	Baby Don't Cry	2000-04-08	7	99
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-02	1	91
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-09	2	87
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-16	3	92
2000	3 Doors Down	3:53	Kryptonite	2000-04-08	1	81
2000	3 Doors Down	3:53	Kryptonite	2000-04-15	2	70
2000	3 Doors Down	3:53	Kryptonite	2000-04-22	3	68
2000	3 Doors Down	3:53	Kryptonite	2000-04-29	4	67
2000	3 Doors Down	3:53	Kryptonite	2000-05-06	5	66

Table 2

Source: Wickham (2014), "Tidy data", Journal of Statistical Software, 59(10)

Data structure

Multiple types in one table (tidy)

id	artist	track	time
1	2 Pac	Baby Don't Cry	4:22
2	2Ge+her	The Hardest Part Of ...	3:15
3	3 Doors Down	Kryptonite	3:53
4	3 Doors Down	Loser	4:24
5	504 Boyz	Wobble Wobble	3:35
6	98ˆ0	Give Me Just One Nig...	3:24
7	A*Teens	Dancing Queen	3:44
8	Aaliyah	I Don't Wanna	4:15
9	Aaliyah	Try Again	4:03
10	Adams, Yolanda	Open My Heart	5:30
11	Adkins, Trace	More	3:05
12	Aguilera, Christina	Come On Over Baby	3:38
13	Aguilera, Christina	I Turn To You	4:00
14	Aguilera, Christina	What A Girl Wants	3:18
15	Alice Deejay	Better Off Alone	6:50

id	date	rank
1	2000-02-26	87
1	2000-03-04	82
1	2000-03-11	72
1	2000-03-18	77
1	2000-03-25	87
1	2000-04-01	94
1	2000-04-08	99
2	2000-09-02	91
2	2000-09-09	87
2	2000-09-16	92
3	2000-04-08	81
3	2000-04-15	70
3	2000-04-22	68
3	2000-04-29	67
3	2000-05-06	66

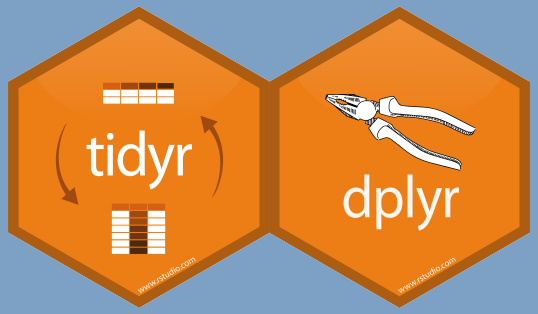
Source: Wickham (2014), "Tidy data", Journal of Statistical Software, 59(10)

Data structure

To sum up the most frequent problems with untidy datasets:

- Column headers are values, not variables
- Multiple variables are stored in one column
- Variables are stored in both rows and columns
- Multiple types of observational units are stored in the same table
- A single observational unit is stored in multiple tables

Tools to make a database tidy:

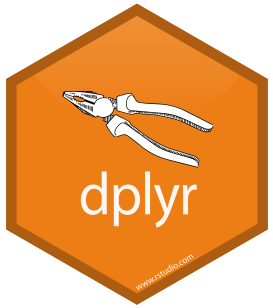


UNIVERSITY OF SUSSEX | BUSINESS SCHOOL

Dplyr: functions for tidying

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

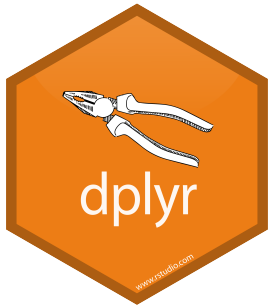
- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.



Dplyr: functions for tidying

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.



These functions enable to prepare datasets for analysis looking at **variables**
Making operation on **columns**:

→ Only keep columns needed for the analysis (select)
→ Make new variables (columns) from previous variables (Mutate)
→ Arrange columns in a specific way (select – again)

→ Change variable name to be more understandable (rename).

Name columns and variables

Variable names is very important to make code understandable for others but for you future self! (coding standard)

→ Use descriptive name (descriptive strings)
→ You can use multiple words using camel**C**ase, snake_case, kebab**-**case, **P**ascal**C**ase

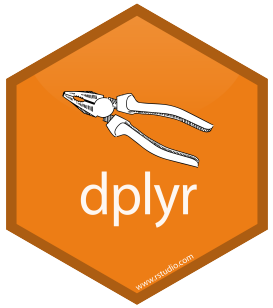
In naming variables you cannot use spaces between words.

Artwork by @allison_horst

Dplyr: functions for tidying

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.



These functions enable to clean datasets for analysis making operation on **entries**
Making operation on **rows**:

→ Only keep entries of interest (filter)

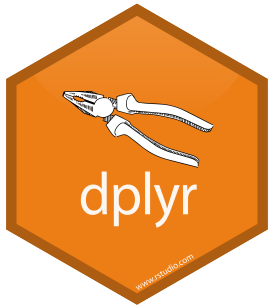
two main uses: dealing entries with incomplete or erroneous data
analyse a subset of the data wit specific characteristics (e.g. variables)

→ Arrange columns in a specific way (arrange)

Dplyr: functions for tidying

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.



These functions enable to prepare datasets for analysis looking at variables Simplifying the dataset:

→ Summarise: enables to reduce the size of the existing data frame, with summary statistics.

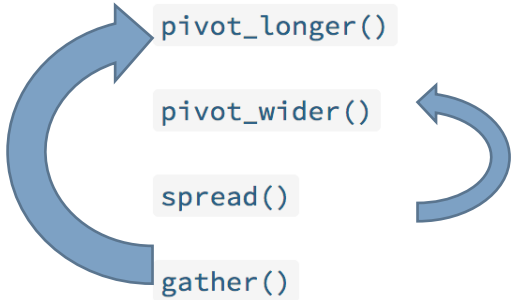
→ With group_by(), we can generate such summary for observations sharing the same characteristics.

TidyR: functions for tidying

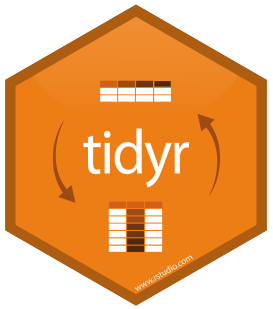
Pivoting

Pivoting changes the representation of a rectangular dataset, without changing the data inside of it.

See `vignette("pivot")` for more details and examples.



`pivot_longer()`	Pivot data from wide to long
`pivot_wider()`	Pivot data from long to wide
`spread()`	Spread a key-value pair across multiple columns
`gather()`	Gather columns into key-value pairs



In September 2019 the package tidyr has been updated, the functions for gather and spread have been updated.

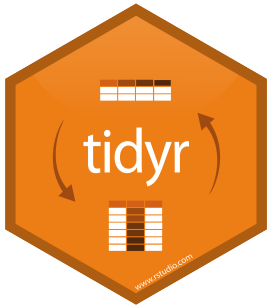
TidyR: functions for tidying

Pivoting

Pivoting changes the representation of a rectangular dataset, without changing the data inside of it.

See `vignette("pivot")` for more details and examples.

`pivot_longer()`	Pivot data from wide to long
`pivot_wider()`	Pivot data from long to wide
`spread()`	Spread a key-value pair across multiple columns
`gather()`	Gather columns into key-value pairs



For pivot longer → pb of column headers are values

Country %>%
pivot_longer(-country, names_to="year", values_to="cases")

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

key value

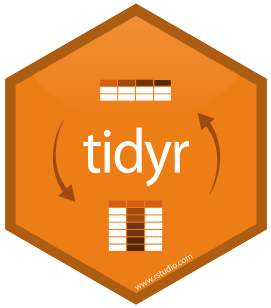
TidyR: functions for tidying

Pivoting

Pivoting changes the representation of a rectangular dataset, without changing the data inside of it.

See `vignette("pivot")` for more details and examples.

`pivot_longer()`	Pivot data from wide to long
`pivot_wider()`	Pivot data from long to wide
`spread()`	Spread a key-value pair across multiple columns
`gather()`	Gather columns into key-value pairs



For pivot wider → pb of values being variables

Country %>%
pivot_wider(names_from="type", values_from="count")

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

key value

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

TidyR: functions for tidying

Transform columns

Character vectors

Multiple variables are sometimes pasted together into a single column, and these tools help you separate back out into individual columns.

`extract()`	Extract a character column into multiple columns using regular expression groups
`separate()`	Separate a character column into multiple columns using a regular expression separator
`separate_rows()`	Separate a collapsed column into multiple rows
`unite()`	Unite multiple columns into one by pasting strings together

→ Pb of multiple variables are stored in one column

separate()

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174
C	1999	212K	1T
C	2000	213K	1T

separate(table3, rate, sep = "/", into = c("cases", "pop"))

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	rate
A	1999	0.7K
A	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K
B	1999	172M
B	2000	80K
B	2000	174M
C	1999	212K
C	1999	1T
C	2000	213K
C	2000	1T

separate_rows()

separate_rows(table3, rate, sep = "/")

TidyR: functions for tidying

Transform columns

Character vectors

Multiple variables are sometimes pasted together into a single column, and these tools help you separate back out into individual columns.

`extract()`	Extract a character column into multiple columns using regular expression groups
`separate()`	Separate a character column into multiple columns using a regular expression separator
`separate_rows()`	Separate a collapsed column into multiple rows
`unite()`	Unite multiple columns into one by pasting strings together

→ Merge variable onto one (example dates)

unite()

country	century	year
Afghan	19	99
Afghan	20	00
Brazil	19	99
Brazil	20	00
China	19	99
China	20	00

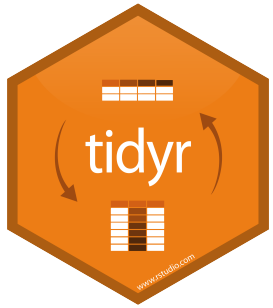
→

country	year
Afghan	1999
Afghan	2000
Brazil	1999
Brazil	2000
China	1999
China	2000

unite(table5, century, year, col = "year", sep = "")

TidyR: functions for tidying

Deal with empty or missing values:



Missing values

Tools for converting between implicit (absent rows) and explicit (`NA`) missing values, and for handling explicit `NA` s.

`complete()`	Complete a data frame with missing combinations of data
`drop_na()`	Drop rows containing missing values
`expand()` `crossing()` `nesting()`	Expand data frame to include all combinations of values
`expand_grid()`	Create a tibble from all combinations of inputs
`fill()`	Fill in missing values with previous or next value
`full_seq()`	Create the full sequence of values in a vector
`replace_na()`	Replace missing values

Let's Practice!

UNIVERSITY OF SUSSEX | BUSINESS SCHOOL

Let's code!

Continue with the second step:

Reshape the dataset

REPLACE _ _ by the appropriate value/function in the code

The exercise / questions are available in the comments (e.g. after the # in the Rscript)

```
#-------------------------------------
# 2. Reshape the given dataset
#-------------------------------------
```

Programming 1.0.1

UNIVERSITY OF SUSSEX | BUSINESS SCHOOL | SCIENCE POLICY RESEARCH UNIT

Programming 1.0.1

Logic and automation:

In order to automate a series of action with a computer, one needs to communicate using a set of **symbols**, specifying rules.

There are a set of tools to give those instructions:

→ Refer to a specific subset of your data (Boolean logic)
→ Apply an action to these subsets (conditions)
→ The ability to repeat the statements to apply to all observations (iterations)

Boolean logic

Referring to a specific subset of the data:

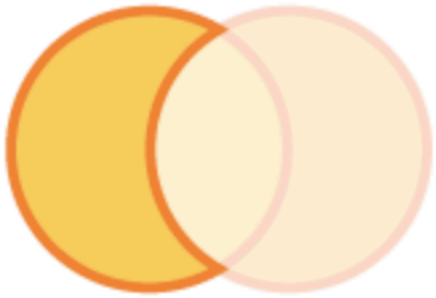
OR

black OR white

AND

black AND white

AND NOT



black AND NOT white

Boolean logic

Logical operators in R:

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y

Boolean logic

Logical operators in R:

Operators for numeric variables

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y

General operators

Conditions

Boolean logic enables and conditions:

→ Making rules for observation of different sets

→ Filter the set you want to work with (Research Questions)

→ Working with outliers or missing values

→ Apply corrections for observation does not conform to a variable specification (e.g. negative age)

Conditions

For applying a specific action for a targeted subset we use conditions:

The general syntax is as follows

AND

If an object is:

black AND white

Then: it is grey

Conditions

Syntax in R:

```
if ("black" %in% x & "white" %in% x){
  print("x is grey")
} else {
  print("x is not grey")
}
```

```
x <- c("white", "black")
```

```
x <- c("white", "blue")
```

Conditions

Syntax in R:

```
if ("black" %in% x & "white" %in% x){
  print("x is grey")
} else {
  print("x is not grey")
}
```

```
x <- c("white", "black")
```

```
[1] "x is grey"
```

```
x <- c("white", "blue")
```

```
[1] "x is not grey"
```

Other possible syntax:

```
ifelse(("black" %in% x & "white" %in% x), "x is grey", "x is not grey")
```

Iterators

Conditions and iterators:

Iterator → repeating an action a certain number of times

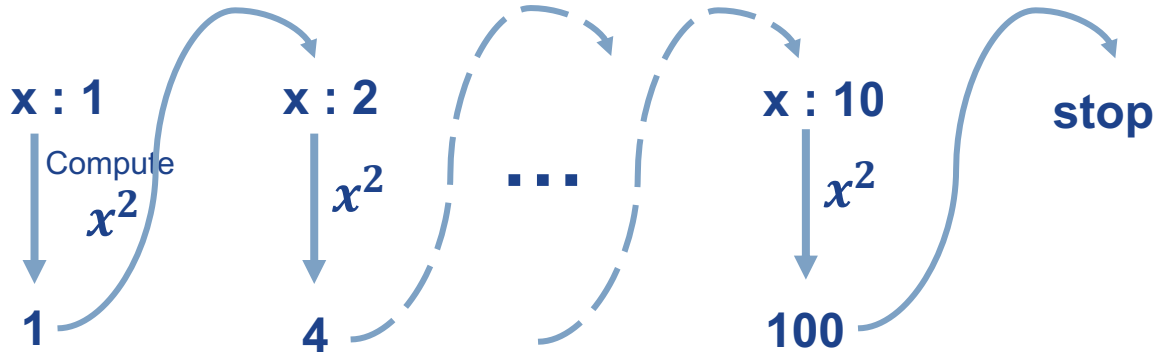
You may want to adapt the action you are taking depending on the characteristic of the variable.

Iterators

For some cases, one wishes to repeat a certain action a number of times,
or until a certain condition is met.

For example: **for** the number 1 to 10, I would like to know the square of these number.

Take a number
x of value which will
take the **value 1 to 10**



Iterators

For some cases, one wishes to repeat a certain action a number of times,
or until a certain condition is met.

For example: **for** the number 1 to 10, I would like to know the square of these number.

```
for (x in 1:10) {
  print(x^2)
}
```



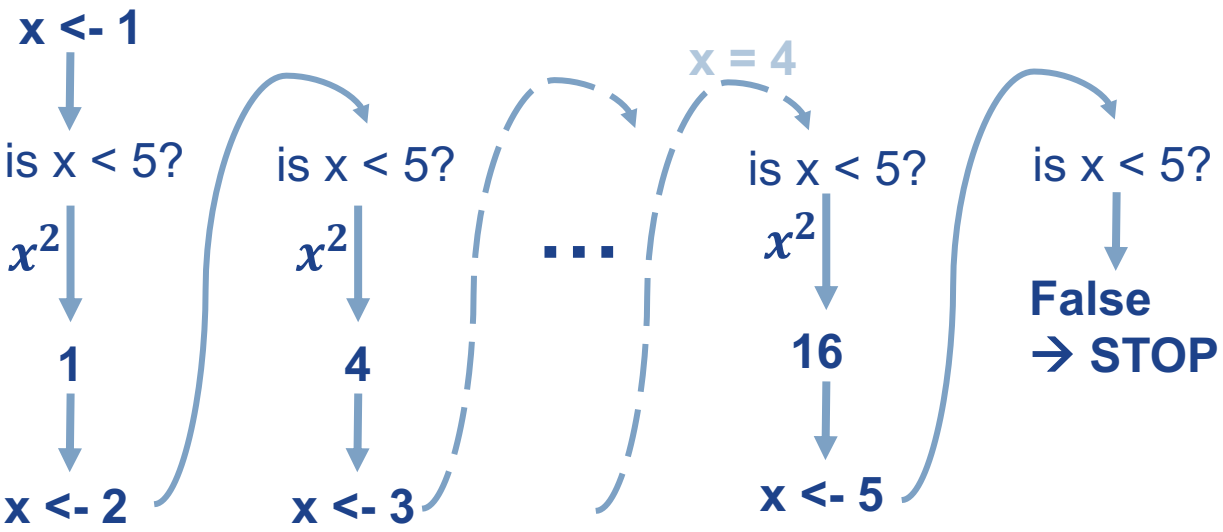
```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
[1] 49
[1] 64
[1] 81
[1] 100
```

Iterators

For some cases, one wishes to repeat a certain action a number of times,
or until a certain condition is met.

For example: I would like to know the square of these number, **while** the number is inferior to 5.

Take a number
x assign it to a number
do something with x
Check if the condition is still true…



Iterators

For some cases, one wishes to repeat a certain action a number of times,
or until a certain condition is met.

For example: I would like to know the square of these number, **while** the number is inferior to 5.

```
x <- -5
while (x < 5) {
  print(x^2)
  x<-x+1
}
```

```
[1] 25
[1] 16
[1] 9
[1] 4
[1] 1
[1] 0
[1] 1
[1] 4
[1] 9
[1] 16
```

Iterators

For some cases, one wishes to repeat a certain action a number of times,
or until a certain condition is met.

For example: I would like to know the square of these number, **while** the number is inferior to 5.

```
x <- -5
while (x < 5) {
  print(x^2)
  x<-x+1
}
```

Don't forget to iterate,
If not the code will get stuck!

```
[1] 25
[1] 16
[1] 9
[1] 4
[1] 1
[1] 0
[1] 1
[1] 4
[1] 9
[1] 16
```

Let's Practice!

UNIVERSITY OF SUSSEX | BUSINESS SCHOOL

Let's code!

Continue with the third step:

Remove the missing values

REPLACE _ _ by the appropriate value/function in the code

The exercise / questions are available in the comments (e.g. after the # in the Rscript)

#--
 # 3. Using logic in you code
#--

Have a go at finishing 4. 5. for next week.

UNIVERSITY OF SUSSEX | BUSINESS SCHOOL

Thank you.

Contact:

Dr Frédérique Bone
(f.bone@sussex.ac.uk)

Research Fellow at SPRU

UNIVERSITY OF SUSSEX | BUSINESS SCHOOL | SCIENCE POLICY RESEARCH UNIT