

Foundations of Data Science for Biologists

Indexing, filtering, and grouping

BIO 724D

2024-SEP-17

Instructors: Greg Wray, Paul Magwene

Revisiting indexing

Revisiting indexing

Indexing = mechanism to refer to a specific subset of a data object

Square bracket `df[x]` or `df['x']`

Integer or string argument; returns an *object* (usually a list)

Double square bracket `df[[x]]`

Integer argument; returns *values* (atomic vector if indexing a data frame)

Dollar sign `df$x`

String argument; returns *values* (item from list or column from data frame)

Tidyverse `verb(df, x)`

String argument; typically returns an *object* (list or data frame)

Take-away lessons about indexing

All four methods of indexing are useful and an essential part of basic R proficiency

Make your code robust:

- Refer to columns by name

- Avoid using ranges (even when indexing with column names)

- Refer to rows by condition (or use unique IDs); avoid numerical indexing

Remember to input column names in different ways for different methods:

- Use quotation marks for `obj[x]` and `obj[[x]]`

- Do not use quotation marks for `obj$` and `verb[obj, x]`

Remember that different methods return data in different forms:

- `obj[x]` and `verb[obj, x]` return lists

- `obj[[x]]` and `obj$x` return atomic vectors

Passing information in pipes

Many common functions work well in pipes

Usually works if the first argument refers to the data

True for most Tidyverse functions: `select()`, `filter()`, `ggplot()`, etc.

True for many base R functions: `length()`, `mean()`, `min()`, etc.

However, not true for all common functions: `lm()`

Using functions in pipes:

Using a function by itself: `func(df, x)`

Using a function in a pipe: `funcA(df, x) |> funcB(y) |> funcC(z)`

The name of the data frame is only needed in the first step

Intermediate filtering

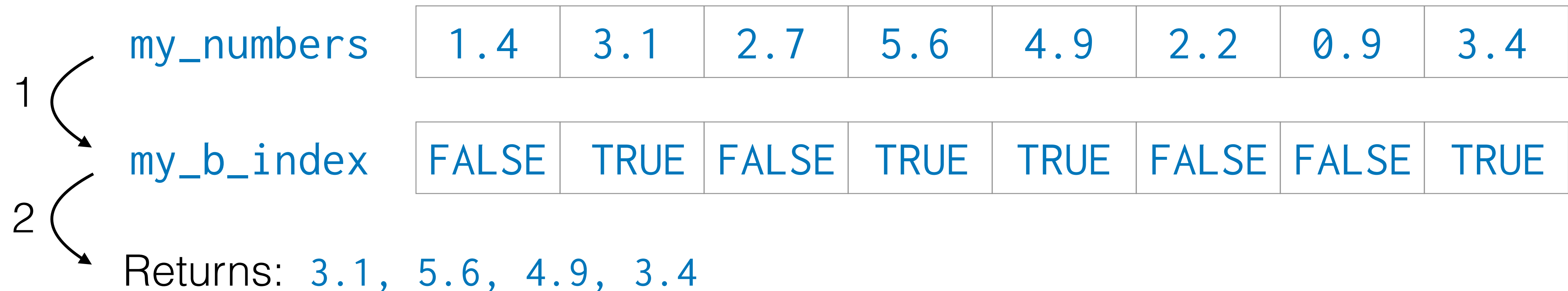
How does Boolean indexing work?

Step 1: create an index column based on a data column(s) and a condition(s)

```
my_df <- mutate(my_df, my_numbers > 3.0)
```

Step 2: filter based on TRUE/FALSE values

```
filter(my_df, my_b_index)
```



1	my_numbers	1.4	3.1	2.7	5.6	4.9	2.2	0.9	3.4
	my_b_index	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	TRUE
2	Returns:	3.1	5.6	4.9	3.4				

Why use Boolean indexing?

Code is more readable: apply the condition once and then use it repeatedly

Updating the condition only has to be done once in your code

You can create multiple Boolean index columns for different conditions

Filtering on a Boolean index is *much* faster than applying the condition again

Introducing factors

Factors are categorical data vectors (homogenous, but different from atomic vectors)

Categories are variables with a defined set of possible values, usually just a few

In R, categories are called **levels**

By default, levels are not ordered (e.g., herbivore, carnivore, omnivore)

Optionally, levels can be ordered (e.g., egg, larva, pupa, adult)

Why use factors?

Memory-efficient: occupies less space than a character vector

Computationally efficient: sort, group_by, and other operations are *much* faster

Minimizes data entry errors: reports any non-standard values

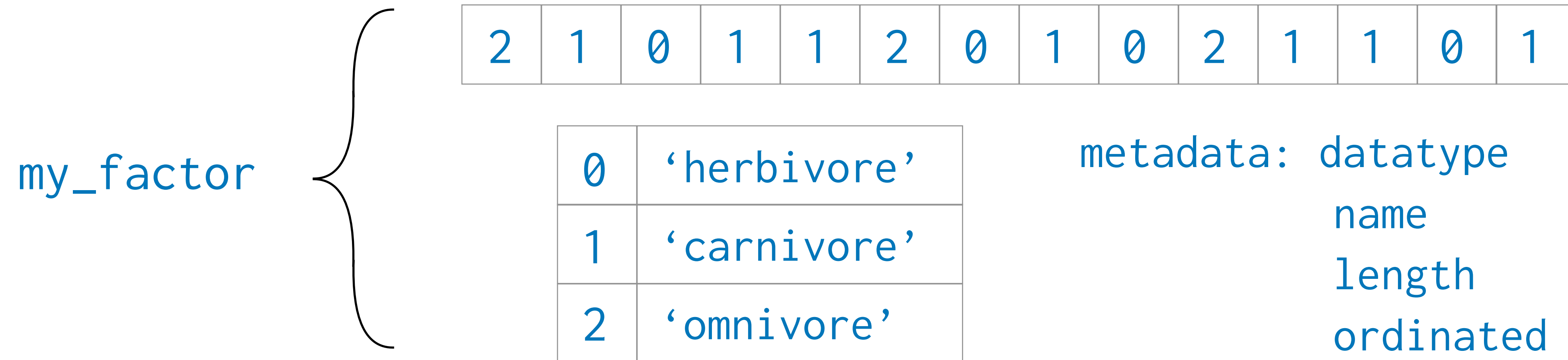
Required for some functions and 3rd-party packages

How do factors work?

Factors are structures with two data components, plus metadata

Integer vector that stores the data

Look-up table that maps integers to strings (optionally ordinated)



Factors provide the best of both worlds

For humans: displayed as strings, thus simple to understand

For computer: manipulated as integers, thus very memory- and processor-efficient

Data set: Yeast colony morphology

Reference: Granek, J. A., D. Murray, Ö. Kayıkçı, and P. M. Magwene. 2013. The genetic architecture of biofilm formation in a clinical isolate of *Saccharomyces cerevisiae*. *Genetics* 193(2):587-600. <https://doi.org/10.1534/genetics.112.142067>

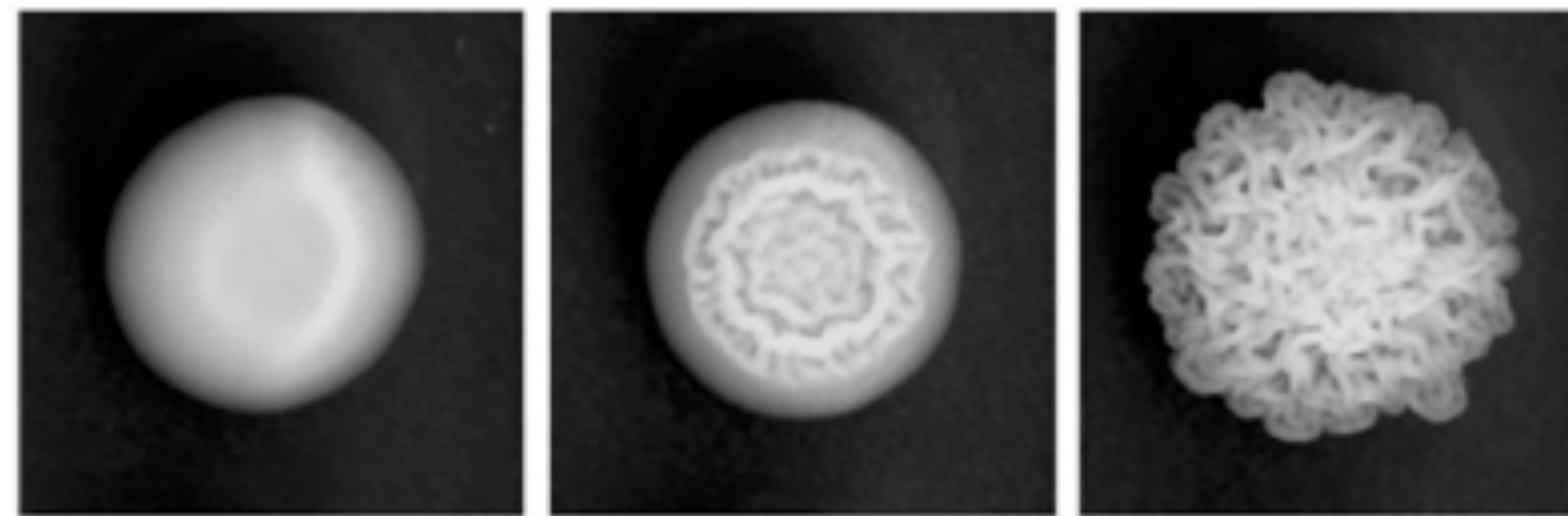
Data on Dryad: <https://doi.org/10.5061/dryad.mn71g>

Brief description:

70 offspring from a genetic cross used to carry out QTL mapping

Genotype information at two major QTLs [grouping variable]

Organismal and molecular phenotypes such as colony complexity score, gene expression, concentration of cyclic AMP [discrete and continuous traits]



Data set: Yeast colony morphology

See [README_for_seg_strain_table.csv](#) on Dryad for explanation of columns

↩

↪

📄

🔍 Filter

🔍

	Strain	Segregant	Pool	Cyr1.geno	Flo11.geno	CM.a	CM.b	CM.c	cAMP	Cyr1.expr	Flo11.expr	Adhes.a	Adhes.b	Adhes.c
1	PMY1235	s1	S	S	S	1.0	1.0	1.0	194.38	1438	2035	0.0987	0.1035	0.1087
2	PMY1236	s2	S	S	S	1.0	1.0	1.0	238.86	1410	696	0.0903	0.1352	0.1022
3	PMY1237	s3	S	S	S	1.0	1.0	1.0	245.32	1292	143961	0.0870	0.2564	0.2591
4	PMY1238	s4	S	S	S	1.0	1.0	1.0	222.58	1814	10077	0.0470	0.1497	0.1119
5	PMY1239	s5	S	S	S	1.0	1.0	1.0	168.99	1204	11281	0.1522	0.1735	0.1511
6	PMY1240	s6	S	S	C	1.0	1.0	1.0	266.80	2274	11729	0.1004	0.1964	0.1740
7	PMY1241	s7	S	S	S	1.0	1.1	1.0	194.64	1334	112014	0.0707	0.1941	0.2002
8	PMY1242	s8	S	S	S	1.1	1.0	1.0	302.43	1374	160390	0.0639	0.1530	0.1540
9	PMY1243	s9	S	S	S	1.0	1.0	1.0	214.02	1722	35037	0.0516	0.1429	0.1543
10	PMY1244	s10	S	S	S	1.0	1.0	1.0	188.81	1674	16110	0.0771	0.2006	0.2185
11	PMY1245	s11	S	S	S	1.0	1.0	1.0	243.36	1816	21117	0.1625	0.1621	0.1866
12	PMY1246	s12	S	S	S	1.0	1.0	1.0	160.04	1769	69004	0.1402	0.1196	0.1159
13	PMY1247	s13	S	S	S	1.0	1.0	1.0	161.37	1746	1246	0.0541	0.1181	0.0946
14	PMY1248	s14	S	S	C	1.0	1.0	1.0	263.64	1100	11573	0.0497	0.1311	0.1523
15	PMY1249	s15	S	S	S	1.0	1.0	1.0	182.38	1522	134342	0.0687	0.1985	0.2600
16	PMY1250	s16	S	S	S	1.0	1.0	1.0	218.14	1604	49086	0.0530	0.1693	0.1791

Showing 1 to 17 of 70 entries, 14 total columns

