Foundations of Data Science for Biologists

Exploring your data

BIO 724D

18-SEP-2023

Instructors: Jesse Granger, Paul Magwene, Greg Wray

Fundamental operations with data frames

Getting Started

- Make a new quarto document called “class 3”
- Load the Tidyverse package
- Import the “class_morphological_measurements_2023.csv” that we used last week. If you need to redownload it, it is under “class 02” on the wiki

Column Manipulations

- Column Manipulations refers to applying a dplyr verb (or other data transformation functions) to one column, or multiple columns simultaneously.
- For Example, creating NEW columns using mutate()
- Or, creating a new column using mutate() that applies to multiple columns

X
1
2
3
4
5
6
7

mutate(df, Y=X^2) →

X	Y
1	1
2	4
3	9
4	16
5	25
6	36
7	49

mutate(df, Z=X+Y) →

X	Y	Z
1	1	2
2	4	6
3	9	12
4	16	20
5	25	30
6	36	42
7	49	56

- By default, summarize() applies by column → Paul will expand on this later

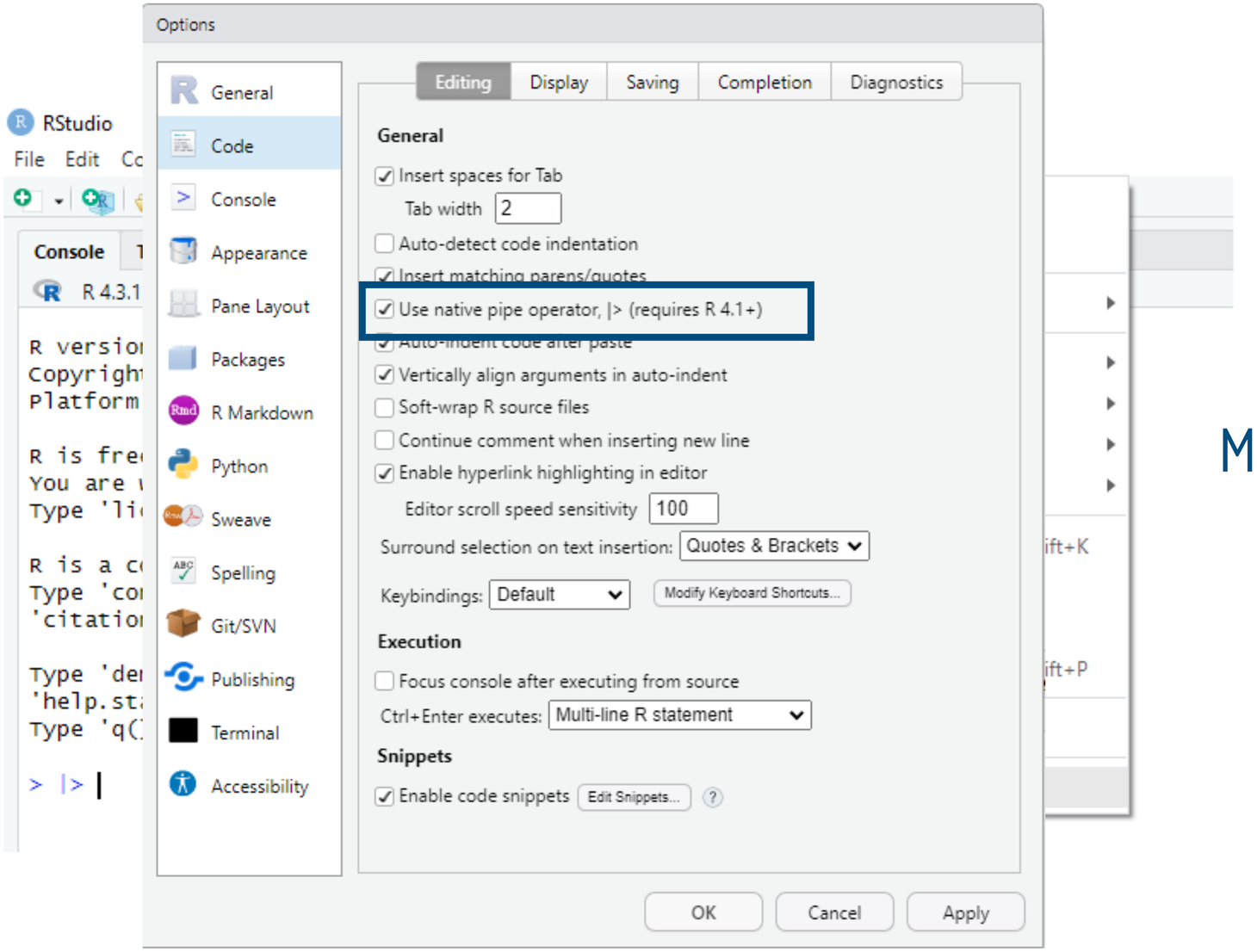
Group Exercise – In groups of 4

In your readings, you learned about a few of the functions in dplyr that are great for wrangling dataframes. In your groups of 4, use the morphological dataset to do the following:

- Use select() to remove the Palm.Length.cm column
- Use mutate() to convert the units in Left.Forearm.Length.cm to mm
- Use mutate() to add a column named "Sum.Finger.Forearm" that is the sum of Third.Finger.Length.mm and the Left.Forearm.Length column that you converted to mm. Use the + operator, not the sum() function.
- Use rename() to rename Third.Finger.Length.mm to "Finger.Length"
- Use filter() to reduce your dataframe to only those individuals where Finger.Length is greater than 75

Pipes

%>% OR |>



PC: ctrl + shift + m

Mac: command + shift + m

Pipes

df

X
1
2
3
4
5
6
7

mutate(df, Y=X^2)

df2

X	Y
1	1
2	4
3	9
4	16
5	25
6	36
7	49

mutate(df, Z=X+Y)

df3

X	Y	Z
1	1	2
2	4	6
3	9	12
4	16	20
5	25	30
6	36	42
7	49	56

```
df2=mutate(df, Y=X^2)
df3=mutate(df2,Z=X+Y)
mutate(mutate(df, Y=X^2), Z=X+Y)
```

```
df |>
   mutate(Y=X^2) |>
   mutate(Z=X+Y)
```

Group Exercise – In groups of 4

In your groups of 4, use the morphological dataset to re-do the code you wrote before in ONE PIPELINE:

- Use select() to remove the Palm.Length.cm column AND THEN
- Use mutate() to convert the units in Left.Forearm.Length.cm to mm AND THEN
- Use rename() to rename Third.Finger.Length.mm to “Finger.Length” and the Left.Forearm.Length column that you converted to mm to “Forearm.Length” AND THEN
- Use filter() to reduce your dataframe to only those individuals where Finger is greater than 75 AND THEN
- Use mutate() to add a column named “Sum.Finger.Forearm” that is the sum of Finger.Length and Forearm.Length. Use the + operator, not the sum() function.

Row Manipulations

Sometimes you want to apply dplyr verbs across rows instead of by columns. If you first use rowwise() at the beginning of your pipeline, it lets you run operations one row at a time.

(This is similar to group_by(), but groups the data by row instead of by group → Paul will discuss this more later)

Try using mutate() and mean() to add a row to your morphological dataset that is the mean of Finger.Length and Forearm.Length. Hint: you will need to use c(). Remember to add na.rm=T inside of your mean function.
Did it give you the correct answer?

```
df |>
  rowwise() |>
  mutate(c(Finger.Length, Forearm.Length), na.rm=T)
```

  mutate(c(Finger.Length, Forearm.Length), na.rm=T)

Now add rowwise() to the beginning of your pipeline and see what you get

Long versus Wide data

In your readings you learned about wide versus long data—let's identify some examples

X	Y
1	2
2	3
3	4
4	5
5	6
6	7
7	8

Variable	Value1	Value2	Value3	Value4	Value5	Value6	Value7
X	1	2	3	4	5	6	7
Y	2	3	4	5	6	7	8

Long versus Wide data

In your readings you learned about wide versus long data—let's identify some examples

Classmate	Palm.Length.cm	Third.Finger.Length.mm	Left.Forearm.Length.cm
Madison	16.0	80	25.5
Amanda	18.5	90	26.0
Hannah	17.0	77	23.0
Jimmy	18.0	84	26.0
Emma	17.5	NA	24.0
Emily	16.0	68	24.0
Bertram	17.0	93	NA
Litong	18.0	90	25.0
Miao	16.0	75	23.0
Desiree	17.0	83	24.0
Anabelle	NA	72	25.0

Long versus Wide data

Classmate	Measurement	Value
Amanda	Palm.Length.cm	18.5
Amanda	Third.Finger.Length.mm	90.0
Amanda	Left.Forearm.Length.cm	26.0
Anabelle	Palm.Length.cm	NA
Anabelle	Third.Finger.Length.mm	72.0
Anabelle	Left.Forearm.Length.cm	25.0
Bertram	Palm.Length.cm	17.0
Bertram	Third.Finger.Length.mm	93.0
Bertram	Left.Forearm.Length.cm	NA
Desiree	Palm.Length.cm	17.0
Desiree	Third.Finger.Length.mm	83.0
Desiree	Left.Forearm.Length.cm	24.0
Emily	Palm.Length.cm	16.0
Emily	Third.Finger.Length.mm	68.0
Emily	Left.Forearm.Length.cm	24.0
Emma	Palm.Length.cm	17.5
Emma	Third.Finger.Length.mm	NA
Emma	Left.Forearm.Length.cm	24.0
Hannah	Palm.Length.cm	17.0
Hannah	Third.Finger.Length.mm	77.0
Hannah	Left.Forearm.Length.cm	23.0
Jimmy	Palm.Length.cm	18.0
Jimmy	Third.Finger.Length.mm	84.0
Jimmy	Left.Forearm.Length.cm	26.0
Litong	Palm.Length.cm	18.0
Litong	Third.Finger.Length.mm	90.0
Litong	Left.Forearm.Length.cm	25.0
Madison	Palm.Length.cm	16.0
Madison	Third.Finger.Length.mm	80.0
Madison	Left.Forearm.Length.cm	25.5
Miao	Palm.Length.cm	16.0
Miao	Third.Finger.Length.mm	75.0
Miao	Left.Forearm.Length.cm	23.0

In your groups of 4, use the morphological dataset to do the following in ONE PIPELINE:

- Use pivot_longer() to go from wide to long format as shown to the left (you could also use gather() if you wanted)
- Use arrange() to sort your data alphabetically by classmate
- Use pivot_wider() to go from long to wide format, returning to the original form (you could also use spread() if you wanted)

Grouping and Boolean indexing

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 availability on Dryad

Dryad link: 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]

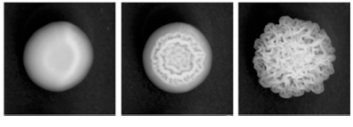


Figure 1: Yeast colonies

Data set: Yeast colony morphology **seg_strain_table.csv**

- See `README_for_seg_strain_table.csv` on Dryad for explanation of columns

Load the data

```
ccm <- read_csv("seg_strain_table.csv")
```

	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	PMY1278	s44	C	C	S	4.3	3.9	3.9	221.34	1816	226922	0.4055	0.4576	0.4564
2	PMY1330	s67	C	C	C	3.6	3.6	3.5	276.70	*NA*	*NA*	0.4989	0.9868	1.0785
3	PMY1331	s68	C	C	C	3.2	3.2	3.5	122.06	*NA*	*NA*	0.5176	1.2117	1.0973
4	PMY1254	s20	S	S	S	1.0	1.0	1.0	164.80	1418	22337	0.0576	0.3082	0.2970
5	PMY1289	s55	C	S	C	3.1	3.0	3.6	195.59	2159	184194	0.3301	0.4025	0.4269
6	PMY1255	s21	S	S	S	1.0	1.0	1.0	252.33	1555	156586	0.1703	0.1821	0.1847
7	PMY1294	s60	C	C	C	3.3	3.6	3.7	247.58	1864	136993	0.7280	1.5931	1.4825
8	PMY1288	s54	C	C	C	3.6	3.7	3.5	197.00	1140	136704	0.4053	0.6482	0.6894
9	PMY1260	s26	S	S	S	1.0	1.0	1.0	194.97	1444	2113	0.0887	0.1066	0.1055
10	PMY1264	s30	S	S	S	1.0	1.0	1.0	176.29	1265	82581	0.1414	0.2100	0.1957

Figure 2: A sample of rows from `seg_strain_table.csv`

Summarizing using `dplyr::summarize`

- When used on an un-grouped data frame (see below) `dplyr::summarize` applies a function to one or more columns in a data frame, returning a new data frame with a single row
- Data are "collapsed" across rows

```
ccm |>
  summarize(mean_cAMP = mean(cAMP, na.rm=TRUE))
```

mean_cAMP
216.2056

summarize can calculate multiple summaries simultaneously

Example

```
ccm |>
  summarize(mean_cAMP = mean(cAMP, na.rm=TRUE),
            mean_log2_Cyr1 = sd(log2(Cyr1.expr), na.rm=TRUE))
```

mean_cAMP	mean_log2_Cyr1
216.2056	0.4665786

Group assignment: Summarizing

Tukey's Five Number Summary

John Tukey, an important 20th-century statistician and mathematician who is often consider one of the founders of Data Science, recommended that exploratory analyses of data always start with calculation of "five number" summary of key variables:

1. Minimum
2. First quartile (25th percentile)
3. Median (50th percentile)
4. Third quartile (75th percentile)
5. Maximum

Assignment

- Use the `dplyr::summary` to produce a 5-number summary of the `CM.a` variable from the yeast colony morphology data set
- Hint: The function `quantile` may be useful here, among others.

Grouping using **dplyr::group_by**

The function `group_by` “decorates” a data frame with grouping information that you specify

- `group_by` by itself doesn’t do any further calculations

Example

```
grouped_ccm <-
  ccm |>
  group_by(Pool)
```

Grouping and Summarizing

Once a data frame is grouped, `summarize` then applies it's functions in a group-wise manner

Example

```
grouped_ccm |>
  summarize(nobs = n(),
            mean_cAMP = mean(cAMP, na.rm=TRUE),
            sd_cAMP = sd(cAMP, na.rm=TRUE))
```

Pool	nobs	mean_cAMP	sd_cAMP
C	35	232.4509	48.33254
S	35	200.8886	43.02055

Grouping: Multiple grouping variables can be used simultaneously

```
ccm |>
  group_by(Pool, Cyr1.geno) |>
  summarize(nobs = n(),
            mean_cAMP = mean(cAMP, na.rm=TRUE),
            sd_cAMP = sd(cAMP, na.rm=TRUE))
```

Pool	Cyr1.geno	nobs	mean_cAMP	sd_cAMP
C	C	22	233.7377	49.01749
C	H	2	231.7150	25.20836
C	S	11	229.4689	54.24839
S	S	35	200.8886	43.02055

Group Assignment: Grouping and summarizing

- Calculate a five-number summary of the `CM.a` variable, grouped by `Cyr1.geno` and `Pool`

Grouping within rows using **dplyr::rowwise**

Sometimes you need to apply a grouping of variables row-wise, such as when you have replicate measures of the same variable and you want to average those replicates. `rowwise` applies grouping on a per-row basis.

Example

```
ccm |>
  rowwise() |>
  mutate(AvgAdhes = mean(c(Adhes.a, Adhes.b, Adhes.c), na.rm=TRUE)) |>
  ungroup() |>  # remove grouping info so slice_sample does what we want
  slice_sample(n=5) |>
  select(Strain, Segregant, AvgAdhes)
```

Strain	Segregant	AvgAdhes
PMY1324	s61	0.1986333
PMY1236	s2	0.1092333
PMY1263	s29	0.2513000
PMY1283	s49	0.5421000
PMY1252	s18	0.1803333

Group Assignment: Using `rowwise`

Assignment

- What does the call to `slice_sample()` do in the code above?
- Repeat the calculation on the previous slide but leave out the call to `rowwise()` and compare the output. What happens?
- Use `rowwise` and `mutate` to create a new column `AvgCM` representing the average `CM` of each strain in the mapping population.
- Calculate five number summaries of `AvgCM` for the data grouped by `Cyr1.geno` and `Pool`

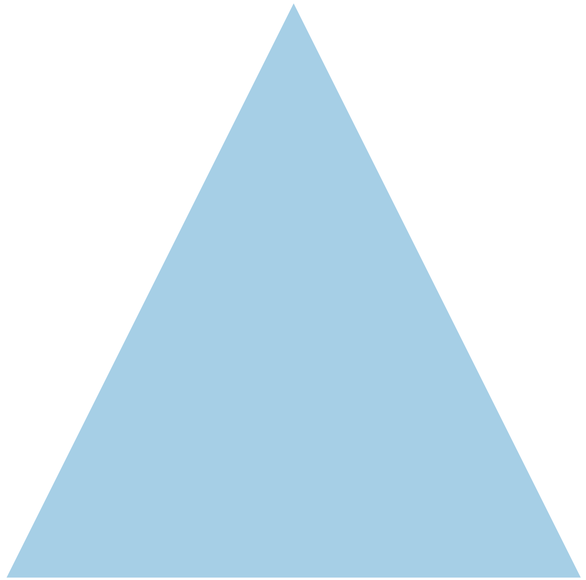
Organizing data and files for computability

Think about your data holistically

individual data types

data files

whole project



Representing data in a computable format

"Since computer programs process real-world information, the first step of computational problem solving is to encode real-world information as data that can be processed by a computer. Real-world information comes from many sources and in a variety of forms. Converting information into data that a computer can store and understand presents many challenges. For example, how can an audio recording, or a seventeenth-century Dutch oil painting or a full-color page from a textbook or even a fingerprint be stored inside of a computer?"

Riley, D and K Hunt (2014) *How Real-World Information Becomes Computable Data*. Chapman & Hall, London.

Encoding data to be maximally computable

Often, there is one obvious best way to represent data — but this is not always the case!

As early as feasible in a new project try to anticipate how you will:

Process: Will you need to encode, normalize, or transform?

Analyze: Which statistical tests or models? Merging with external data?

Interpret: What hypotheses? What relationships or patterns?

Based on these considerations, proactively encode your data in a thoughtful way

Exercise: how best to encode data?

Example 1: behavior observed in the field

Predation by Cheetahs

Example 2: genotype:phenotype association

Mimicry in *Heliconius* butterflies

In both cases:

What information do you want to record?

How can the data be best represented in a computable format?

How can this representation reveal the relationships / hypotheses of interest?

Will these data be useful to others, and if so, how will they work with them?

Some common pitfalls when encoding data

Using inconsistent units (e.g., ˚F and ˚C, DMS and DD for geolocations)

Using multiple representations of missing data (e.g., `999`, `NA`, `0`, `-`, empty cell)

Storing date/time information without timezone (inability to compute time differences)

Scaling or normalizing (inability to work with original range or distribution information)

Transforming with outliers or skewness (can alter linearity or normality of data)

Rounding too much (loss of meaningful precision)

Discretizing / binning a continuous variable (inability to access original values)

Using inconsistent, incompatible, or arbitrary encoding schemes for categorical data

Using ordinated levels for categorical data when no natural order exists

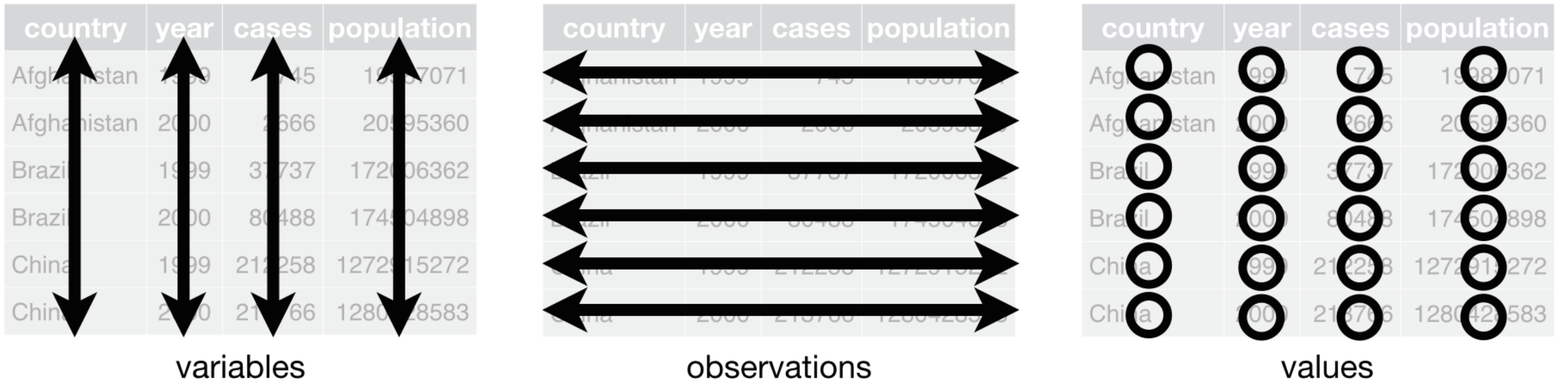
Encoding in a way that is incompatible with a model (e.g., string for machine learning)

Encoding or transforming in a way that violates assumptions of a statistical test

Tidy format

Three characteristics

- Each variable is a column; each column is a variable
- Each row is an observation; each observation is a row
- Each value is a cell; each cell is a single value



Or, put another way

- The values for a single observation are in their own row
- The values for a single variable are in their own column
- The observations are all of the same data type and format

Wickham, H (2014) Tidy Data. *JStatistical Software* 59
Figure from: Groelmund et al, *R For Data Science*

Advantages of tidy data

Two high-level advantages:

- Using a consistent data structure simplifies every downstream step
- Placing variables in columns leverages R's vectorized operations

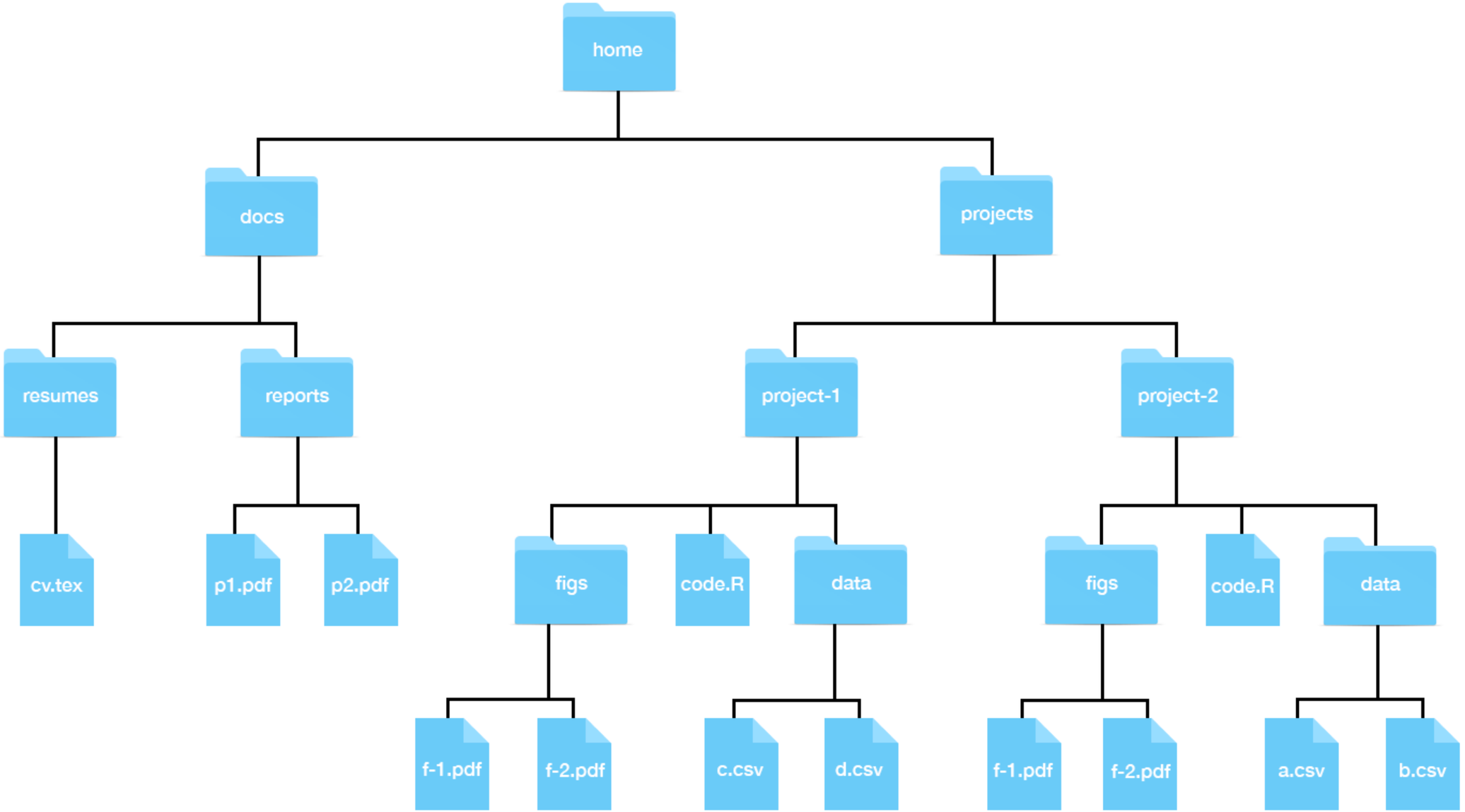
More specific advantages:

- Usually easier to understand many rows than many columns
- Fewer confusing variable names
- Fewer issues with missing values and imbalanced repeated measures data
- Observations are all of the same data type and format
- Required for many statistical procedures (e.g., hierarchical or mixed effects models)
- Simplifies making plots using ggplot2

Wickham, H (2014) Tidy Data. *Journal of Statistical Software* issue 59

Sample project organization: bad

Sample project organization: good



Izzary, R (2023) *Introduction to Data Science: Data Analysis and Prediction with R*. CRC Press.

Considerations for project file organization

File organization poses several challenges

- **Quantity**: even a simple project may involve 10s of files; 100s or 1000s is common
- **Diversity**: a typical project involves many different kinds of files
- **Versions**: code is updated, working data files are often sequentially processed, etc.

Being organized and consistent with your files will save you time and pain

- Rational directory organization and file naming conventions are *essential*

There is no single perfect system — find a way that **fits the project** and **works for you**!

Best practices for naming files and directories

Do not:

- Use spaces (creates major hassles with CLI); instead use underscore or dash
- Depend on case to distinguish (confusing / error-prone); instead use all lower case
- Avoid using symbols if possible (creates major hassles with RegEx)

Do:

- Think about your naming conventions in advance of starting project
- Stick with your naming convention within a project (but learn and adapt for later ones!)
- Use descriptive names rather than generic ones (e.g., thesis_ch_01.txt, not 1.txt)
- For number-based names, pad with leading zeros and anticipate the maximum value
- For date-based names, use YYYY-MM-DD or YYYYMMDD formats (ISO 8601)

Best practices for working with project files

Keep original data files in a separate folder and only work with copies

Make clear how metadata are associated with data (text, code, schema, etc.)

Add a readme.txt file to most directories: even brief descriptions are very useful

Use code for all file manipulations, analyses, and generating figures/tables

Organize code into workflows: initial wrangling, statistical tests, figures, etc.

If working on code as a team, use a git repository to track/merge versions

Back up everything securely (3:2:1 rule)

At the end of a project, share data, metadata, code, and readme files