








Week 3: Data Wrangling

Willa & Elena
9/14/2021

Welcome to Week 3!

1. Make sure you have the session 3 practice materials downloaded from the webpage
2. Open up `week3_starter_code.R` in Rstudio and get started with the warm-up

Today's agenda

-  Warm-up
-  What is tidy data
-  Demo - Introducing tidyverse and the pipe operator
-  Practice - Data wrangling
-  Discussion

What is “tidy data”?

“**TIDY DATA** is a standard way of mapping the meaning of a dataset to its structure.”

—HADLEY WICKHAM

Organizing a dataset this way makes it easy to interpret

In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement

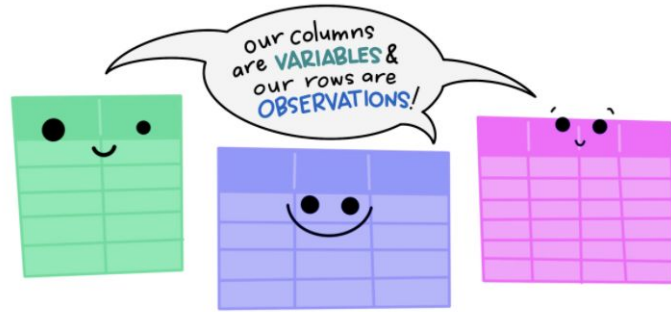
each column a variable

id	name	color
1	floof	gray
2	max	black
3	cat	orange
4	donut	gray
5	merlin	black
6	panda	calico

each row an observation

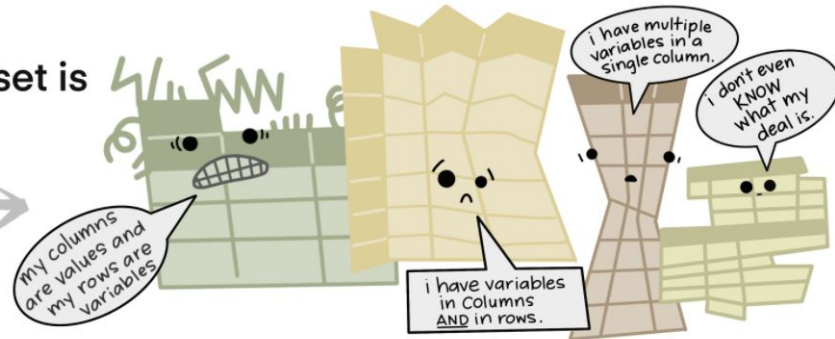
Wickham, H. (2014). Tidy Data. Journal of Statistical Software 59 (10). DOI: 10.18637/jss.v059.i10

The standard structure of tidy data means that "tidy datasets are all alike..."



"...but every messy dataset is messy in its own way."

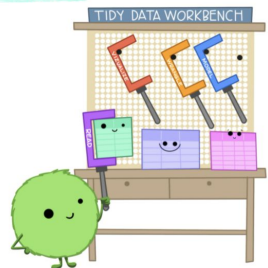
-HADLEY WICKHAM



Why do we want tidy data?

1. Reproducible code (fewer errors)!

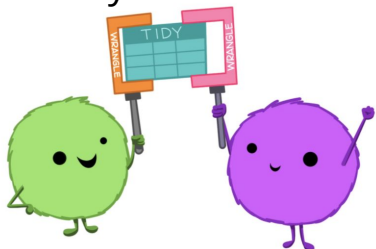
When working with tidy data, we can use the **same tools** in similar ways for different datasets...



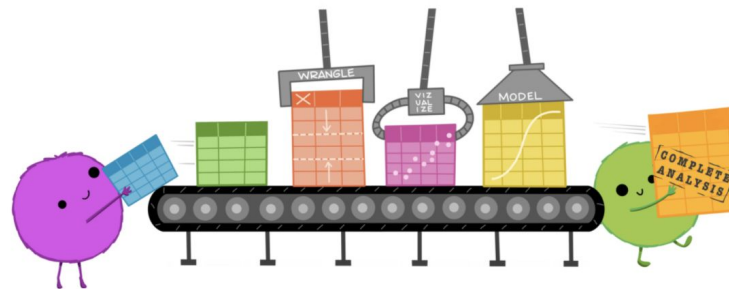
...but working with untidy data often means reinventing the wheel with **one-time** approaches that are **hard to iterate or reuse**.



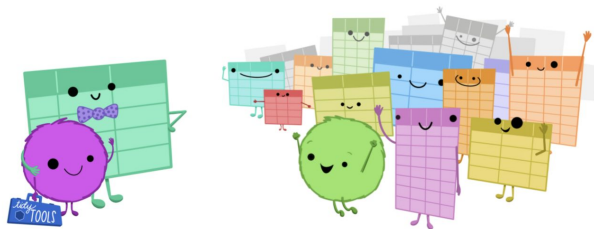
2. Easy collaboration



3. Automated pipelines (efficient and consistent)!



4. Data sharing (easy to interpret and combine with other data)



Is our data tidy?

Open `penguins.csv`

Check the basic structure




- Is every column a variable?
- Is every row an observation?
- Is every cell one value?

An observation might mean something different for different data! Here each penguin is an observation.

We're in good shape but there is still more processing to do to get the data we want for our analyses.

Open `penguins_clean.csv`

What are some of the differences between these two dataframes?

1. Selected only a few of the variables  Changes to `columns`
2. Filtered observations by a specific year  Changes to `rows`
3. Removed missing values  Changes to `cells`
4. New column "bill_sum"

It's often useful to follow this hierarchy when *removing* data

There is an easy way to do all this in R!

Introducing our favorite library: Tidyverse!

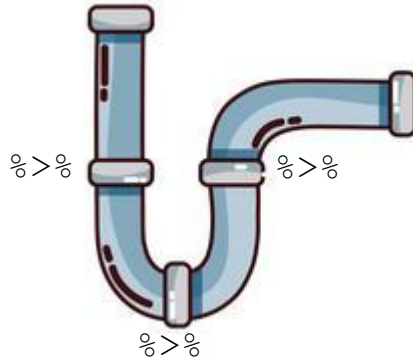
Blast off into the...



- A *library* is an organized collection of code and functions written by other members of the R community.
- Tidyverse is a library created specifically for organizing and processing your data
 - Includes *dplyr*, *ggplot* etc.
- Install `tidyverse` and unlock a whole new world of functions and commands.

A new operator: Pipes %>%

- Once you have installed tidyverse you have access to a new symbol: %>%
- The pipe operator (%>%) allows you to string together many functions on the same data frame.
- This lets you make a workflow of tasks that you perform sequentially on a dataframe.



Let's remember the steps we want to perform on the penguins dataset

1. Select only a few of the variables
2. Filter observations by a specific year
3. Remove NAs
4. Create a new column

Base R

```
penguins <- read.csv("penguins.csv")  
penguins_select <- "Select certain columns in penguins"  
  
penguins_select_filter <- "Filter penguins_select by a specific year"  
  
penguins_select_filter_NA <- "Remove missing values from  
penguins_select_filter"  
  
penguins_clean <- "Create a new column in  
penguins_select_filter_NA"
```

This is "pseudo code"

Let's remember the steps we want to perform on the penguins dataset

1. Select only a few of the variables
2. Filter observations by a specific year
3. Remove NAs
4. Create a new column

```
penguins <- read.csv("penguins.csv")
```

```
penguins_select <- "Select certain columns in penguins"
```

```
penguins_select_filter <- "Filter penguins_select by a specific year"
```

```
penguins_select_filter_NA <- "Remove missing values from  
penguins_select_filter"
```

```
penguins_clean <-
```

Let's remember the steps we want to perform on the penguins dataset

In tidyverse we can combine these steps using the %>% operator and save it all as one new dataframe.

```
penguins <- read.csv("penguins.csv")  
  
penguins_clean <- penguins %>%
```

New df (under penguins_clean)
is (between penguins_clean and penguins)
original df (under penguins)
then (under %>%)

1. Select only a few of the variables
2. Filter observations by a specific year
3. Remove NAs
4. Create a new column

```
"Select certain columns" %>%
```

then (under %>%)

```
"Filter by a specific year" %>%
```

then (under %>%)

```
"Remove missing values" %>%
```

then (under %>%)

```
"Create a new column"
```

Pipes help make your code:

- *Reproducible*
- *Readable*
- *Easy to automate*



Tidy data and happy collaborators



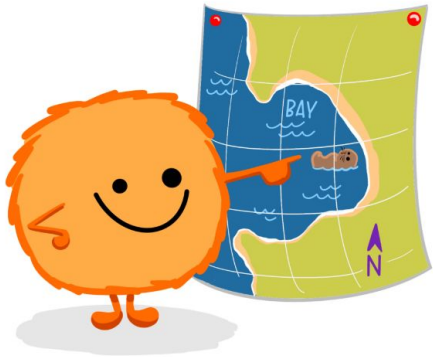
Now lets venture into the tidyverse and replace the pseudo code with real code.

dplyr::filter()

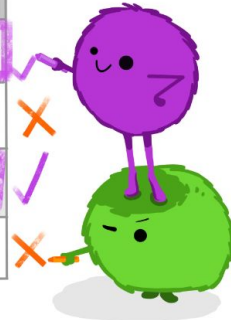
KEEP ROWS THAT
satisfy
your **CONDITIONS**

keep rows from... this data... ONLY IF... AND site is "bay"

```
filter(df, type == "otter" & site == "bay")
```



type	food	site
otter	urchin	bay
shark	seal	channel
otter	abalone	bay
otter	crab	wharf



@allison_horst



[Allison Horst](#)

dplyr::case_when()

IF ELSE...
(but you love it?)

```
df %>% ADD COLUMN 'danger'  
  mutate(danger = case_when(IF type is kraken type == "kraken" THEN ~ "extreme!",  
    T ~ "high" ~ "high"))
```

danger is extreme!

OTHERWISE, danger is high.

