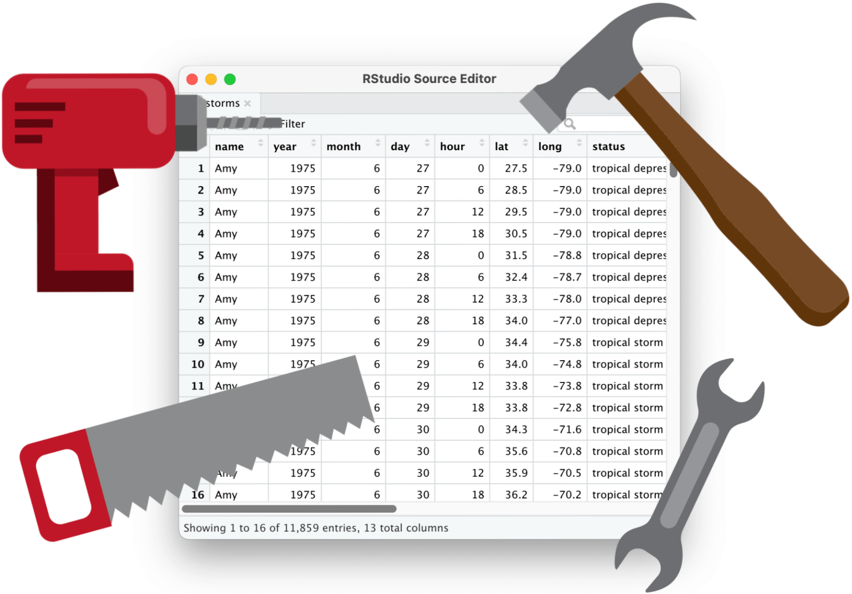
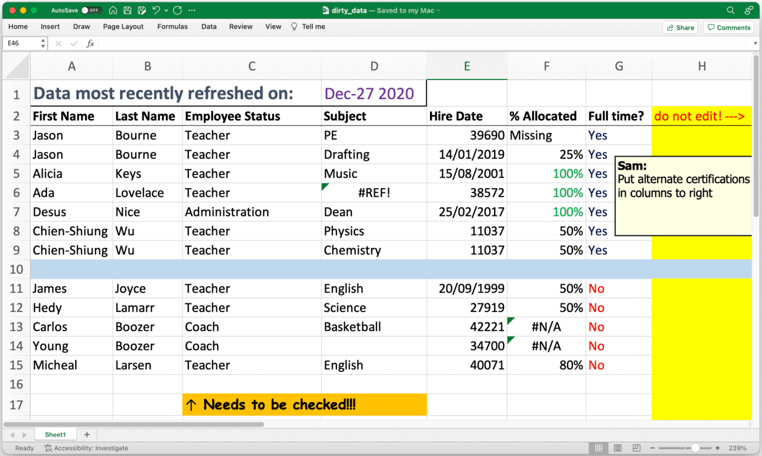
Data manipulation in R


RStudio Source Editor
storms
Filter
name year month day hour lat long status
1 Amy 1975 6 27 0 27.5 -79.0 tropical depres
2 Amy 1975 6 27 6 28.5 -79.0 tropical depres
3 Amy 1975 6 27 12 29.5 -79.0 tropical depres
4 Amy 1975 6 27 18 30.5 -79.0 tropical depres
5 Amy 1975 6 28 0 31.5 -78.8 tropical depres
6 Amy 1975 6 28 6 32.4 -78.7 tropical depres
7 Amy 1975 6 28 12 33.3 -78.0 tropical depres
8 Amy 1975 6 28 18 34.0 -77.0 tropical depres
9 Amy 1975 6 29 0 34.4 -75.8 tropical storm
10 Amy 6 29 6 34.0 -74.8 tropical storm
11 6 29 12 33.8 -73.8 tropical storm
6 29 18 33.8 -72.8 tropical storm
6 30 0 34.3 -71.6 tropical storm
1975 6 30 6 35.6 -70.8 tropical storm
1975 6 30 12 35.9 -70.5 tropical storm
16 Amy 1975 6 30 18 36.2 -70.2 tropical storm
Showing 1 to 16 of 11,859 entries, 13 total columns

You will spend most of your time cleaning data.



	A	B	C	D	E	F	G	H
1	Data most recently refreshed on:			Dec-27 2020				
2	First Name	Last Name	Employee Status	Subject	Hire Date	% Allocated	Full time?	do not edit! --->
3	Jason	Bourne	Teacher	PE	39690	Missing	Yes	
4	Jason	Bourne	Teacher	Drafting	14/01/2019	25%	Yes	
5	Alicia	Keys	Teacher	Music	15/08/2001	100%	Yes	
6	Ada	Lovelace	Teacher	#REF!	38572	100%	Yes	
7	Desus	Nice	Administration	Dean	25/02/2017	100%	Yes	
8	Chien-Shiung	Wu	Teacher	Physics	11037	50%	Yes	
9	Chien-Shiung	Wu	Teacher	Chemistry	11037	50%	Yes	
10								
11	James	Joyce	Teacher	English	20/09/1999	50%	No	
12	Hedy	Lamarr	Teacher	Science	27919	50%	No	
13	Carlos	Boozer	Coach	Basketball	42221	#N/A	No	
14	Young	Boozer	Coach		34700	#N/A	No	
15	Micheal	Larsen	Teacher	English	40071	80%	No	
16								
17			↑ Needs to be checked!!!					

Session overview

1. Alternative packages
2. Pipes
3. Essential data manipulation tasks

a) Select columns or rows
b) Sorting a dataset
c) Creating or modifying columns
d) Combining datasets
e) Reshaping a dataset
f) Grouping and summarising data

There are many approaches to data manipulation in R.

Base R	tidyverse	data.table

- Use the right tool for the job.
- Use whatever feels most comfortable and productive.

What do I use?

The tidyverse

"an opinionated collection of R packages designed for data science. All packages share an underlying philosophy and common APIs".

www.tidyverse.org

Core:

For specific data types:

For import/export:

You don't need to learn all these packages.

Just use what you need.

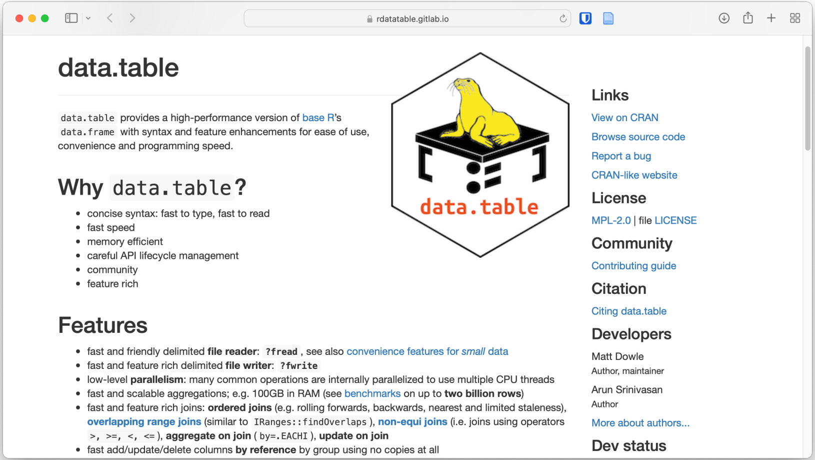
(Use the "Reference" help pages).

This session uses functions from `dplyr` and `tidyr`.

I can never remember which functions come from which package.

It's fine.

https://rdatatable.gitlab.io/data.table/


rdatatable.gitlab.io

data.table

`data.table` provides a high-performance version of base R's `data.frame` with syntax and feature enhancements for ease of use, convenience and programming speed.

Why `data.table`?

- concise syntax: fast to type, fast to read
- fast speed
- memory efficient
- careful API lifecycle management
- community
- feature rich

Features

- fast and friendly delimited **file reader**: `?fread`, see also convenience features for *small* data
- fast and feature rich delimited **file writer**: `?fwrite`
- low-level **parallelism**: many common operations are internally parallelized to use multiple CPU threads
- fast and scalable aggregations; e.g. 100GB in RAM (see benchmarks on up to **two billion rows**)
- fast and feature rich joins: **ordered joins** (e.g. rolling forwards, backwards, nearest and limited staleness), **overlapping range joins** (similar to `IRanges::findOverlaps`), **non-equi joins** (i.e. joins using operators `>, >=, <, <=`), **aggregate on join** (`by=.EACHI`), **update on join**
- fast add/update/delete columns **by reference** by group using no copies at all

data.table

Links

View on CRAN
Browse source code
Report a bug
CRAN-like website

License

MPL-2.0 | file LICENSE

Community

Contributing guide

Citation

Citing data.table

Developers

Matt Dowle
Author, maintainer
Arun Srinivasan
Author
More about authors...

Dev status

See this page for a comparison of `dplyr` and `data.table`

See also: dtplyr

Pipes

We can use |> to chain commands together

R code is traditionally written as a series of statements.

```
df <- read_dta("stata_dataset.dta")
df <- df[df$age > 18]
df$log_income <- log(df$income)
df$female <- data$gender == "Female"
```

We can use |> to chain commands together

R code is traditionally written as a series of statements.

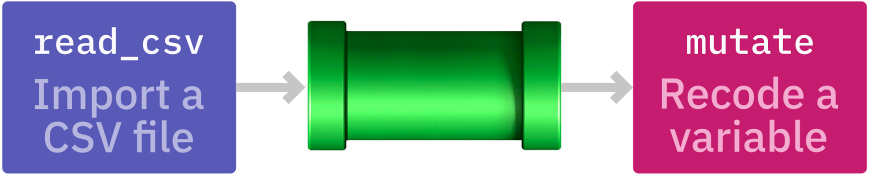
```
df <- read_dta("stata_dataset.dta")
df <- df[df$age > 18]
df$log_income <- log(df$income)
df$female <- data$gender == "Female"
```

The pipe allows us to chain together several statements.

```
df <- read_dta("stata_dataset.dta") |>
    filter(age > 18) |>
    mutate(log_income = log(income),
           female = gender == "Female")
```

How does it work?

A pipe takes output from one command and uses it as input to the next.



They are written as |> at the end of a line.

You need to save the output by assigning to an object.

```
clean_data <- messy_data |>
    mutate(...) |>
    pivot_wider(...)
```

Session overview

1. ~~Alternative packages~~
2. ~~Pipes~~
3. Essential data manipulation tasks

Essential data manipulation tasks

a) Select columns or rows

b) Sorting a dataset

c) Creating or modifying columns

d) Combining datasets

e) Reshaping a dataset

f) Grouping and summarising data

Let's load some data...

```
> library(tidyverse) # Load the tidyverse package
> data(starwars)     # Load a built-in dataset
> head(starwars)
# A tibble: 6 x 13
name height  mass  hair_color  skin_color eye_color birt
<chr>  <int> <dbl>       <chr>       <chr>     <chr>
1 Luke Skywalker    172    77       blond         fair
2          C-3PO    167    75        <NA>         gold
3          R2-D2     96    32        <NA> white, blue
4    Darth Vader    202   136        none        white
5    Leia Organa    150    49       brown        light
6      Owen Lars    178   120 brown, grey        light
# ... with 4 more variables: species <chr>, films <list>
```

- Get a list of the variables in this data frame.
- How many rows and columns are there?

a) Selecting rows and columns

We've already seen how subsetting can be used to select parts of objects.

For example, we can select rows and columns by number:

```
starwars[1:5, ]
starwars[, c(1, 3, 8)]
```

Or by name:

```
starwars$height
starwars[, c("birth_year", "homeworld")]
```

Select and drop columns with `select`

```r
starwars |>
    select(birth_year, homeworld)

# Negate to remove a column
starwars |>
    select(-eye_color)

# Select several columns
starwars |>
    select(starts_with("h"),
           ends_with("color"),
           matches("or$"))
```

You can rename at the same time.

```
starwars |>
    select(new_name = old_name)
```

To rename without dropping other variables, use `rename`:

```
starwars |>
    rename(new_name = old_name)
```

Select rows with `filter`

`filter` selects rows based on a condition.

For example, select all rows where `mass` is above 100:

```
starwars |>
    filter(mass > 100)
```

Separate multiple conditions with a comma:

```
starwars |>
    filter(mass > 100,
           eye_color == "yellow",
           homeworld == "Tatooine")
```

b) Sorting a dataset with arrange

```
starwars |>
    arrange(height)

# Sorting on multiple columns
starwars |>
    arrange(height, mass)

# Sort in descending order
starwars |>
    arrange(desc(height))
```

Practical: pipes, select, filter, and arrange

From the `starwars` data frame:

1. Select the columns `height`, `mass`, `gender`, and `species`.
2. Filter to select rows with `height` less than 191 and with `species` equal to "Human".
3. Sort the result by `height`.

Use pipes to combine each operation; store the result as a new data frame.

c) Create or modify variables with `mutate`

In base R, we can create new columns using the assignment operator:

```
df$eligible <- TRUE
df$log_income <- log(df$income)
df$female <- df$sex == "Female"
```

We can transform existing variables using subsetting:

Replace “Not applicable” with `NA`.

```
df$wstat[df$wstat == "Not applicable"] <- NA
```

Create binary measure of age:

```
df$older <- 0
df$older[df$age > 50] <- 1
```

But `mutate` makes this easier.

```
df <- df |>
    mutate(eligible = TRUE,
           log_income = log(income),
           female = sex == "Female")
```

But mutate makes this easier.

```
df <- df |>
    mutate(eligible = TRUE,
           log_income = log(income),
           female = sex == "Female")
```

Things to note:

- We're not using subsetting or quoting.
- We can include multiple statements inside a single `mutate` function.
- We can use earlier computations in later ones.
- We need to store the result.

Practical: Creating and modifying columns

1. Load the `tidyverse` package and the `mtcars` dataset.
2. Add a new column indicating whether a car weighs over 3000 lbs (i.e. `wt > 3`).
   i. Using subsetting
   ii. Using mutate
3. Tabulate this new column against the number of cylinders (`cyl`).

Practical: Creating and modifying columns

2.
   i. Using subsetting

   ```
   mtcars$heavy <- mtcars$wt > 3
   ```

   ii. Using mutate

   ```
   mtcars  <- mtcars |>
     mutate(heavy = wt > 3)
   ```

3. `table(mtcars$heavy, mtcars$cyl)`

d) Combining datasets

Appending two data frames

id	age
001	45
002	33
003	63

+

height
1.87
1.43
1.68

→

id	age	height
001	45	1.87
002	33	1.43
003	63	1.68

d) Combining datasets

Appending two data frames

id	age
001	45
002	33
003	63

\+

height
1.87
1.43
1.68

→

id	age	height
001	45	1.87
002	33	1.43
003	63	1.68

Merging or 'joining' two data frames

id	age
001	45
002	33
003	63

\+

id	height
002	1.87
001	1.43
003	1.68

id	age	height
001	45	1.43
002	33	1.87
003	63	1.68

Append with bind_rows and bind_cols

```
# 1. Select some columns
a <- starwars[, 2:4]
b <- starwars[, 9]
```

Append with bind_rows and bind_cols

```
# 1. Select some columns
a <- starwars[, 2:4]
```

```
# a                                    # b
  height  mass    hair_color              homeworld
   <int> <dbl>         <chr>                  <chr>
1    172    77         blond           1   Tatooine
2    167    75          <NA>           2   Tatooine
3     96    32          <NA>           3      Naboo
4    202   136          none           4   Tatooine
5    150    49         brown           5   Alderaan
6    178   120   brown, grey           6   Tatooine
7    165    75         brown           7   Tatooine
8     97    32          <NA>           8   Tatooine
# ... with 79 more rows                # ... with 79 more row
```

```r
# Bind them together
bind_cols(a, b)

# A tibble: 87 x 4
   height  mass    hair_color homeworld
    <int> <dbl>         <chr>     <chr>
 1    172    77         blond  Tatooine
 2    167    75          <NA>  Tatooine
 3     96    32          <NA>     Naboo
 4    202   136          none  Tatooine
 5    150    49         brown  Alderaan
 6    178   120   brown, grey  Tatooine
 7    165    75         brown  Tatooine
 8     97    32          <NA>  Tatooine
 9    183    84         black  Tatooine
10    182    77 auburn, white   Stewjon
# ... with 77 more rows
```

```r
# For rows...
a <- starwars[1:5, ]
b <- starwars[20:30, ]

bind_rows(a, b)
```

```r
# For rows...
a <- starwars[1:5, ]
b <- starwars[20:30, ]

bind_rows(a, b)
```

These commands are replacements the `cbind` and `rbind` from base R.

Merging with `*_join`

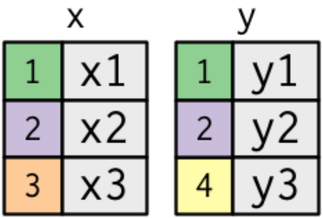
- We merge to combine variables held in separate datasets based on one or more common keys.

Merging with `*_join`

- We merge to combine variables held in separate datasets based on one or more common keys.
- These operations are referred to as joins.

Merging with `*_join`

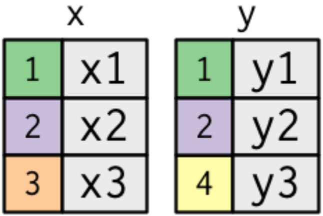
- We merge to combine variables held in separate datasets based on one or more common keys.
- These operations are referred to as joins.



Source: R for Data Science (Wickham and Grolemund, 2017)

Merging with `*_join`

- We merge to combine variables held in separate datasets based on one or more common keys.
- These operations are referred to as joins.

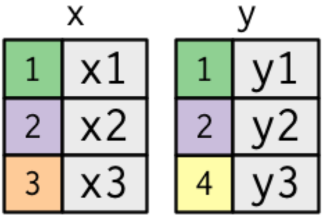


- A join is a way of connecting each row in x to zero, one, or more rows in y.

Source: R for Data Science (Wickham and Grolemund, 2017)

Merging with `*_join`

- We merge to combine variables held in separate datasets based on one or more common keys.
- These operations are referred to as joins.



- A join is a way of connecting each row in x to zero, one, or more rows in y.
- The type of join we need depends on how many keys from x are also found in y.

Source: R for Data Science (Wickham and Grolemund, 2017)

We're going to focus on four types of join:

`inner_join`	matches pairs of observations whenever their keys are equal.
`left_join`	keeps all observations in `x`.
`right_join`	keeps all observations in `y`.
`full_join`	keeps all observations in `x` and `y`.

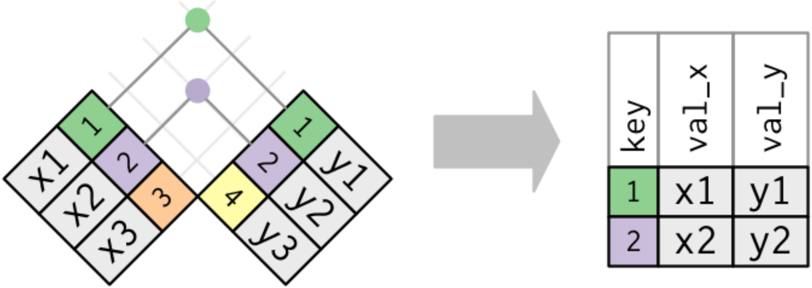
We’re going to focus on four types of join:

`inner_join` matches pairs of observations whenever their keys are equal.

`left_join` keeps all observations in `x`.

`right_join` keeps all observations in `y`.

`full_join` keeps all observations in `x` and `y`.



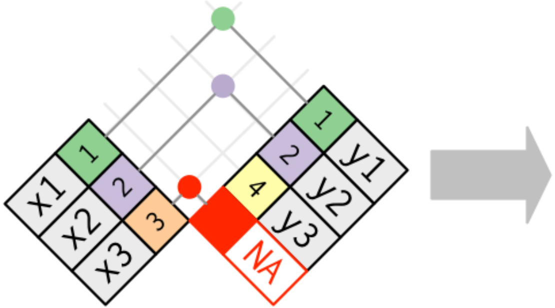
We’re going to focus on four types of join:

`inner_join` matches pairs of observations whenever their keys are equal.

`left_join` keeps all observations in `x`.

`right_join` keeps all observations in `y`.

`full_join` keeps all observations in `x` and `y`.



key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

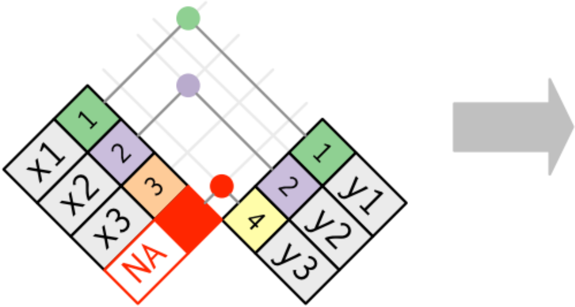
We're going to focus on four types of join:

`inner_join` matches pairs of observations whenever their keys are equal.

`left_join` keeps all observations in `x`.

`right_join` keeps all observations in `y`.

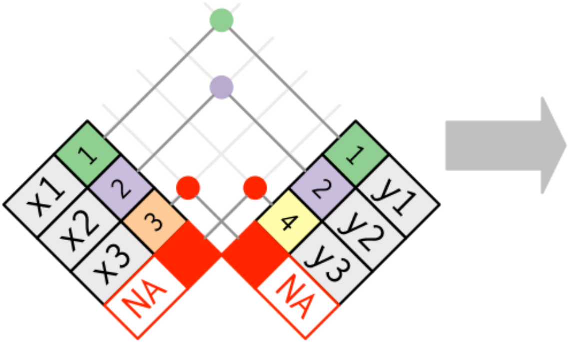
`full_join` keeps all observations in `x` and `y`.



key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

We're going to focus on four types of join:

`inner_join`	matches pairs of observations whenever their keys are equal.
`left_join`	keeps all observations in `x`.
`right_join`	keeps all observations in `y`.
`full_join`	keeps all observations in `x` and `y`.



key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

- By default, data frames are joined based on variables that appear in both tables.
- Unlike other packages, you don't always need to specify the joining key.

  If in doubt, I try `full_join` first and drop matches that aren't needed.

- By default, data frames are joined based on variables that appear in both tables.
- Unlike other packages, you don't always need to specify the joining key.

If in doubt, I try `full_join` first and drop matches that aren't needed.

`band_members`

name	band
Mick	Stones
John	Beatles
Paul	Beatles

`band_instruments`

name	plays
John	guitar
Paul	bass
Keith	guitar

```r
# inner_join
band_members |> inner_join(band_instruments)
```

```r
# inner_join
band_members |> inner_join(band_instruments)
# left_join
band_members |> left_join(band_instruments)
```

```r
# inner_join
band_members |> inner_join(band_instruments)
# left_join
band_members |> left_join(band_instruments)
# right_join
band_members |> right_join(band_instruments)
```

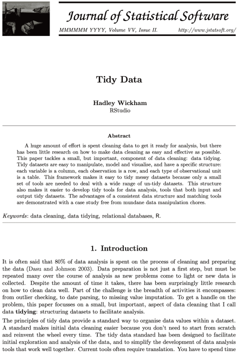
```r
# inner_join
band_members |> inner_join(band_instruments)
# left_join
band_members |> left_join(band_instruments)
# right_join
band_members |> right_join(band_instruments)
# full_join
band_members |> full_join(band_instruments)
```

Tidy data

What is tidy data?

> *"All happy families resemble one another; every unhappy family is unhappy in its own way."*
>
> *Leo Tolstoy (1878)*

> *"Tidy datasets are all alike, but every messy dataset is messy in its own way."*
>
> *Hadley Wickham (2014)*



Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. http://www.jstatsoft.org/

Tidy Data

Hadley Wickham
RStudio

Abstract

A huge amount of effort is spent cleaning data to get it ready for analysis, but there has been little research on how to make data cleaning as easy and effective as possible. This paper tackles a small, but important, component of data cleaning: data tidying. Tidy datasets are easy to manipulate, model and visualise, and have a specific structure: each variable is a column, each observation is a row, and each type of observational unit is a table. This framework makes it easy to tidy messy datasets because only a small set of tools are needed to deal with a wide range of un-tidy datasets. This structure also makes it easier to develop tidy tools for data analysis, tools that both input and output tidy datasets. The advantages of a consistent data structure and matching tools are demonstrated with a case study free from mundane data manipulation chores.

Keywords: data cleaning, data tidying, relational databases, R.

1. Introduction

It is often said that 80% of data analysis is spent on the process of cleaning and preparing the data (Dasu and Johnson 2003). Data preparation is not just a first step, but must be repeated many over the course of analysis as new problems come to light or new data is collected. Despite the amount of time it takes, there has been surprisingly little research on how to clean data well. Part of the challenge is the breadth of activities it encompasses: from outlier checking, to date parsing, to missing value imputation. To get a handle on the problem, this paper focusses on a small, but important, aspect of data cleaning that I call data **tidying**: structuring datasets to facilitate analysis.

The principles of tidy data provide a standard way to organise data values within a dataset. A standard makes initial data cleaning easier because you don't need to start from scratch and reinvent the wheel every time. The tidy data standard has been designed to facilitate initial exploration and analysis of the data, and to simplify the development of data analysis tools that work well together. Current tools often require translation. You have to spend time

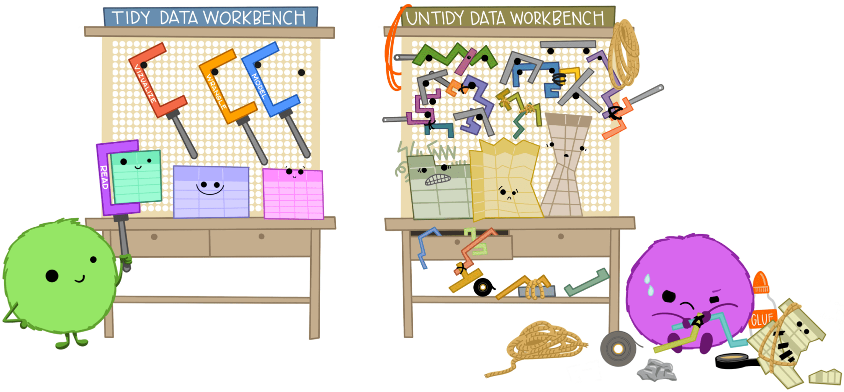
Journal of Statistical Software, **59**(10), 1–23.
`https://doi.org/10.18637/jss.v059.i10`

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

Variables Observations Values

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.



Illustrations adapted from the Openscapes blog *Tidy Data for reproducibility, efficiency, and collaboration* by Julia Lowndes and Allison Horst.

a) Select columns or rows

b) Sorting a dataset

c) Creating or modifying columns

d) Combining datasets

e) Reshaping a dataset

f) Grouping and summarising data

e) From WIDE to LONG with `pivot_longer`

```
> relig_income
# A tibble: 18 x 11
   religion `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k` `$50-75k`
   <chr>      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
 1 Agnostic      27        34        60        81        76       137
 2 Atheist       12        27        37        52        35        70
 3 Buddhist      27        21        30        34        33        58
 4 Catholic     418       617       732       670       638      1116
 5 Don't know    15        14        15        11        10        35
 6 Evangel...   575       869      1064       982       881      1486
 7 Hindu          1         9         7         9        11        34
 8 Histori...   228       244       236       238       197       223
 9 Jehovah       20        27        24        24        21        30
10 Jewish        19        19        25        25        30        95
11 Mainlin      289       495       619       655       651      1107
12 Mormon        29        40        48        51        56       112
13 Muslim         6         7         9        10         9        23
14 Orthodox      13        17        23        32        32        47
15 Other C...     9         7        11        13        13        14
16 Other F...    20        33        40        46        49        63
17 Other W...     5         2         3         4         2         7
18 Unaffil...   217       299       374       365       341       528
# ...with 4 more variables: `$75-100k` <dbl>, `$100-150k` <dbl>,
# >150k` <dbl>, `Don't know/refused` <dbl>
```

From WIDE to LONG with `pivot_longer`

```
pivot_longer(data,
             cols,
             names_to = "name",
             values_to = "value")
```

Name for **new** label column

Columns to reshape

Name of **new** values column

```
> relig_income
# A tibble: 18 x 11
   religion `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k` `$50-75k`
   <chr>      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
 1 Agnostic      27        34        60        81        76       137
 2 Atheist       12        27        37        52        35        70
 3 Buddhist      27        21        30        34        33        58
 4 Catholic     418       617       732       670       638      1116
 5 Don't know    15        14        15        11        10        35
 6 Evangel...   575       869      1064       982       881      1486
 7 Hindu          1         9         7         9        11        34
 8 Histori...   228       244       236       238       197       223
 9 Jehovah       20        27        24        24        21        30
10 Jewish        19        19        25        25        30        95
11 Mainlin      289       495       619       655       651      1107
12 Mormon        29        40        48        51        56       112
13 Muslim         6         7         9        10         9        23
14 Orthodox      13        17        23        32        32        47
15 Other C...     9         7        11        13        13        14
16 Other F...    20        33        40        46        49        63
17 Other W...     5         2         3         4         2         7
18 Unaffil...   217       299       374       365       341       528
# ...with 4 more variables: `$75-100k` <dbl>, `$100-150k` <dbl>,
# >150k` <dbl>, `Don't know/refused` <dbl>
```

```r
relig_income |>
    pivot_longer(cols = -religion,
                 names_to = "income",
                 values_to = "count")
```

```
# A tibble: 180 x 3
   religion             income             count
   <chr>                <chr>              <dbl>
 1 Agnostic             <$10k                 27
 2 Agnostic             $10-20k               34
 3 Agnostic             $20-30k               60
 4 Agnostic             $30-40k               81
 5 Agnostic             $40-50k               76
 6 Agnostic             $50-75k              137
 7 Agnostic             $75-100k             122
 8 Agnostic             $100-150k            109
 9 Agnostic             >150k                 84
10 Agnostic             Don't know/refused    96
11 Atheist              <$10k                 12
12 Atheist              $10-20k               27
13 Atheist              $20-30k               37
14 Atheist              $30-40k               52
15 Atheist              $40-50k               35
16 Atheist              $50-75k               70
```

Another example...

```
> billboard
         artist                  track date.entered wk1 wk2 wk3 wk4 wk
          2 Pac Baby Don't Cry (Keep...   2000-02-26  87  82  72  77  8
        2Ge+her The Hardest Part Of ...   2000-09-02  91  87  92  NA  N
   3 Doors Down              Kryptonite   2000-04-08  81  70  68  67  6
   3 Doors Down                   Loser   2000-10-21  76  76  72  69  6
       504 Boyz           Wobble Wobble   2000-04-15  57  34  25  17  1
           98^0 Give Me Just One Nig...   2000-08-19  51  39  34  26  2
        A*Teens           Dancing Queen   2000-07-08  97  97  96  95 10
        Aaliyah           I Don't Wanna   2000-01-29  84  62  51  41  3
        Aaliyah               Try Again   2000-03-18  59  53  38  28  2
 Adams, Yolanda           Open My Heart   2000-08-26  76  76  74  69  6
  Adkins, Trace                    More   2000-04-29  84  84  75  73  7
   Alice Deejay        Better Off Alone   2000-04-08  79  65  53  48  4
    Allan, Gary Smoke Rings In The D...   2000-01-22  80  78  76  77  9
          Amber                  Sexual   1999-07-17  99  99  96  96 10
      Anastacia          I'm Outta Love   2000-04-01  92  NA  NA  95  N
  Anthony, Marc             My Baby You   2000-09-16  82  76  76  70  8
  Anthony, Marc          You Sang To Me   2000-02-26  77  54  50  43  3
          Avant           My First Love   2000-11-04  70  62  56  43  3
          Avant               Separated   2000-04-29  62  32  30  23  2
          BBMak               Back Here   2000-04-29  99  86  60  52  3
   Badu, Erkyah                Bag Lady   2000-08-19  67  53  42  41  4
       Baha Men    Who Let The Dogs Out   2000-07-22  99  92  85  76  6
```

```
> billboard |>
>   pivot_longer(starts_with("wk"),
>                names_to = "week",
>                values_to = "chart_position")
# A tibble: 24,092 × 5
   artist track                        date.entered week  cha
   <chr>  <chr>                        <date>       <chr>
 1 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk1
 2 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk2
 3 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk3
 4 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk4
 5 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk5
 6 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk6
 7 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk7
 8 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk8
 9 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk9
10 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk10
```

```
> billboard |>
>   pivot_longer(starts_with("wk"),
>                names_to = "week",
>                values_to = "chart_position") |>
>   mutate(week = parse_number(week))
# A tibble: 24,092 × 5
   artist track                   date.entered week  cha
   <chr>  <chr>                   <date>       <chr>
 1 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk1
 2 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk2
 3 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk3
 4 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk4
 5 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk5
 6 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk6
 7 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk7
 8 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk8
 9 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk9
10 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk10
```

```
> billboard |>
>   pivot_longer(starts_with("wk"),
>                names_to = "week",
>                values_to = "chart_position",
>                values_drop_na = TRUE) |>
>   mutate(week = parse_number(week))
# A tibble: 5,307 × 5
   artist  track                   date.entered  week ch
   <chr>   <chr>                   <date>       <dbl>
 1 2 Pac   Baby Don't Cry (Keep... 2000-02-26       1
 2 2 Pac   Baby Don't Cry (Keep... 2000-02-26       2
 3 2 Pac   Baby Don't Cry (Keep... 2000-02-26       3
 4 2 Pac   Baby Don't Cry (Keep... 2000-02-26       4
 5 2 Pac   Baby Don't Cry (Keep... 2000-02-26       5
 6 2 Pac   Baby Don't Cry (Keep... 2000-02-26       6
 7 2 Pac   Baby Don't Cry (Keep... 2000-02-26       7
 8 2Ge+her The Hardest Part Of ... 2000-09-02       1
 9 2Ge+her The Hardest Part Of ... 2000-09-02       2
10 2Ge+her The Hardest Part Of ... 2000-09-02       3
```

```
> longer <- billboard |>
>   pivot_longer(starts_with("wk"),
>                names_to = "week",
>                values_to = "chart_position",
>                values_drop_na = TRUE) |>
>   mutate(week = parse_number(week))
```

From LONG to WIDE with `pivot_wider`

An **existing** column that will become the column headings

```
pivot_wider(data,
            names_from = "name",
            values_from = "value")
```

An **existing** column containing the values.

Suppose we already have our data in LONG format…

artist	track	date.entered	week	chart_position
2 Pac	Baby Don't	2000-02-26	1	87
2 Pac	Baby Don't	2000-02-26	2	82
2 Pac	Baby Don't	2000-02-26	3	72
2 Pac	Baby Don't	2000-02-26	4	77
2 Pac	Baby Don't	2000-02-26	5	87
2 Pac	Baby Don't	2000-02-26	6	94
2 Pac	Baby Don't	2000-02-26	7	99
2Ge+her	The Hardest	2000-09-02	1	91
2Ge+her	The Hardest	2000-09-02	2	87
2Ge+her	The Hardest	2000-09-02	3	92
3 Doors Down	Kryptonite	2000-04-08	1	81
3 Doors Down	Kryptonite	2000-04-08	2	70
3 Doors Down	Kryptonite	2000-04-08	3	68
3 Doors Down	Kryptonite	2000-04-08	4	67
3 Doors Down	Kryptonite	2000-04-08	5	66
3 Doors Down	Kryptonite	2000-04-08	6	57
3 Doors Down	Kryptonite	2000-04-08	7	54
3 Doors Down	Kryptonite	2000-04-08	8	53
3 Doors Down	Kryptonite	2000-04-08	9	51
3 Doors Down	Kryptonite	2000-04-08	10	51
3 Doors Down	Kryptonite	2000-04-08	11	51
3 Doors Down	Kryptonite	2000-04-08	12	51
3 Doors Down	Kryptonite	2000-04-08	13	47
3 Doors Down	Kryptonite	2000-04-08	14	44

```
pivot_wider(data,
            names_from = ?,
            values_from = ?)
```

Practical: pivoting between LONG and WIDE

1. Reshape the `fish_encounters` dataset to WIDE format, such that each column represents a different monitoring station. After reshaping, is this dataset 'tidy'? Why?

2. Reshape the `world_bank_pop` dataset to LONG format, such that it contains three columns: `country`, `indicator`, and `year`.

3. (If time) Reshape the `table2` dataset such that there is a separate column for 'cases' and 'population'.

f) Grouping and summarising data

The column to summarise

```
data |>
  summarise(new = function(old))
```

The name for a new column

The function with which to summarise (e.g., mean).

Note the differences with `mutate`

`mutate` 1 row = 1 row

`summarise` Many rows = 1 row

For example, calculate the mean and standard deviation of a column:

```
> mtcars |>
+   summarise(mean = mean(wt),
+             sd = sd(wt))
     mean        sd
1 3.21725 0.9784574
```

As with mutate, we can have multiple expressions, separated by commas.

Grouping data with `group_by`

We often want to calculate summaries for subgroups in our data.

```
> # Average fuel efficiency by number
> # of cylinders?
> mtcars |>
+ group_by(cyl) |>
+ summarise(efficiency = mean(mpg))
# A tibble: 3 x 2
    cyl efficiency
  <dbl>      <dbl>
      4   26.66364
      6   19.74286
      8   15.10000
```

Note that, once you define the grouping, all subsequent operations to be grouped.

For example, `mutate`:

```
> mtcars |>
+   group_by(cyl) |>
+   mutate(max = max(mpg))
```

This will calculate the maximum per group.

Phew, that was a lot...

1. Alternative packages
2. Pipes
3. Essential data manipulation tasks

a) Select columns or rows
b) Sorting a dataset
c) Creating or modifying columns
d) Combining datasets
e) Reshaping a dataset
f) Grouping and summarising data

Recoding variables...

Recoding with `if_else` and `case_when`

Two functions that solve many common data cleaning tasks.

A statement that returns TRUE or FALSE

The value to return if CONDITION is TRUE

```
if_else(CONDITION,
        VALUE IF TRUE,
        VALUE IF FALSE)
```

The value to return if CONDITION is FALSE.

```
> starwars |>
+   mutate(weight = if_else(mass > 80,
+                           "Heavy",
+                           "Not heavy")) |>
+   select(mass, weight)
# A tibble: 87 × 2
    mass weight
   <dbl> <chr>
 1    77 Not heavy
 2    75 Not heavy
 3    32 Not heavy
 4   136 Heavy
 5    49 Not heavy
 6   120 Heavy
 7    75 Not heavy
 8    32 Not heavy
 9    84 Heavy
10    77 Not heavy
```

However, if you're creating a binary indicator (`TRUE`/`FALSE`), there's a simpler option:

```
> starwars |>
>   mutate(heavy = mass > 80) |>
>   select(mass, heavy)
# A tibble: 87 × 2
    mass heavy
   <dbl> <lgl>
 1    77 FALSE
 2    75 FALSE
 3    32 FALSE
 4   136 TRUE
 5    49 FALSE
 6   120 TRUE
 7    75 FALSE
 8    32 FALSE
```

`case_when`

To evaluate multiple conditions <u>in order</u>.

The value to return if
this CONDITION is <u>TRUE</u>

```
case_when(CONDITION 1 ~ VALUE 1,   1
          CONDITION 2 ~ VALUE 2,   2
          CONDITION 3 ~ VALUE 3,   3
          TRUE        ~ VALUE 4)   4
```

Each condition is evaluated
in turn; the order is important.

Bonus: `tidylog`

cran.r-project.org/package=tidylog

```
library(tidyverse)
library(tidylog, warn.conflicts = FALSE)

filtered <- filter(mtcars, cyl == 4)

merged <- left_join(band_members,
                    band_instruments,
                    by = "name")
```

Demonstration