Tidy data with tidyr (basics)

Outline

The tidyr package and tidy data review

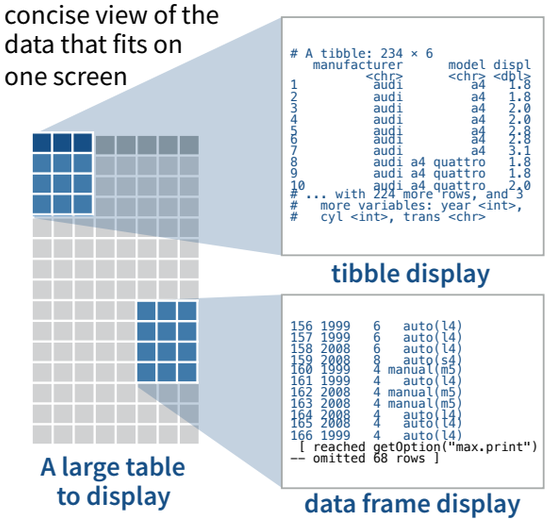
Functions

- Important ones: pivot_wider(), pivot_longer(), and friends
- Others that might be of some use!

Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve three behaviors:

- **Subsetting** - [always returns a new tibble, [[and $ always return a vector.
- **No partial matching** - You must use full column names when subsetting
- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen



- Control the default appearance with options:

  options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf**)**

- View full data set with **View()** or **glimpse()**
- Revert to data frame with **as.data.frame()**

CONSTRUCT A TIBBLE IN TWO WAYS

tibble(…**)**
Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

tribble(…**)**
Construct by rows.
tribble(~x, ~y,
1, "a",
2, "b",
3, "c")

Both make this tibble

```
A tibble: 3 × 2
      x     y
  <int> <chr>
1     1     a
2     2     b
3     3     c
```

as_tibble(x, …**)** Convert data frame to tibble.

enframe(x, name = "name", value = "value"**)**
Convert named vector to a tibble

is_tibble(x**)** Test whether x is a tibble.

Tidy Data with tidyr

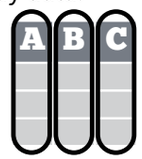
Tidy data is a way to organize tabular data. It provides a consistent data structure across packages.

A table is tidy if:

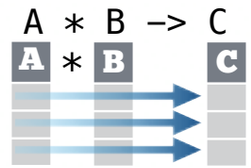
Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

Tidy data:



Makes variables easy to access as vectors



Preserves cases during vectorized operations

Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout.

gather(data, key, value, …, na.rm = FALSE, convert = FALSE, factor_key = FALSE**)**

gather() moves column names into a **key** column, gathering the column values into a single **value** column.

table4a

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

gather(table4a, `1999`, `2000`, key = "year", value = "cases")

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL**)**

spread() moves the unique values of a **key** column into the column names, spreading the values of a **value** column across the new columns.

table2

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

spread(table2, type, count)

Handle Missing Values

drop_na(data, …**)**
Drop rows containing NA's in … columns.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
D	3

drop_na(x, x2)

fill(data, …, .direction = c("down", "up"))**
Fill in NA's in … columns with most recent non-NA values.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	1
C	1
D	3
E	3

fill(x, x2)

replace_na(data, replace = list(), …**)**
Replace NA's by column.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	2
C	2
D	3
E	2

replace_na(x, list(x2 = 2))

Expand Tables - quickly create tables with combinations of values

complete(data, …, fill = list()**)**
Adds to the data missing combinations of the values of the variables listed in …
complete(mtcars, cyl, gear, carb)

expand(data, …**)**
Create new tibble with all possible combinations of the values of the variables listed in …
expand(mtcars, cyl, gear, carb)

Split Cells

Use these functions to split or combine cells into individual, isolated values.

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", …**)**

Separate each cell in a column to make several columns.

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"))

separate_rows(data, …, sep = "[^[:alnum:].]+", convert = FALSE**)**

Separate each cell in a column to make several rows.

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(table3, rate, sep = "/")

unite(data, col, …, sep = "_", remove = TRUE**)**

Collapse cells across several columns to make a single column.

table5

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

The *tidyr* package is a part of the *tidyverse* and is the main function for tidying data

Tidy data principles:

1. Every column is a variable
2. Every row is an observation
3. Every cell is a single value

There are five main actions/categories that *tidyr* addresses

1. Pivotting
2. Rectangling
3. Nesting
4. Splitting
5. Implicit/Explicit

Pivotting Data

There are many circumstances in which you need to change the shape of the data

I've encountered it most often in plotting, but it happens elsewhere!

Example:

name	hw1	hw2	hw3	test1	test2
"John"	60	89	93	85	89
"Mary"	89	93	75	90	82
"Ben"	76	98	83	87	76
"Steph"	88	81	87	90	95

I want to plot how the students did over time/assignment with one point per student. How do I do that?

pivot_longer()

pivot_longer() takes data in a “wide” format and gathers the data into a long format.

Now the third iteration of the same function: melt(), gather(), pivot_longer()

pivot_longer()

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  names_prefix = NULL,
  names_sep = NULL,
  names_pattern = NULL,
  names_ptypes = list(),
  names_repair = “check_unique",
  values_to = "value",
  values_drop_na = FALSE,
  values_ptypes = list()
)
```

The data to be pivoted

The columns to pivot into longer format

Name of the column where variable names go

Name of the column where values go

pivot_longer()

```
tb
```

```
 name    hw1 hw2 hw3 test1 test2
"John"   60  89  93   85    89
"Mary"   89  93  75   90    82
"Ben"    76  98  83   87    76
"Steph"  88  81  87   90    95
```

```
tb %>%
   pivot_longer(-name,
                names_to = "assignment",
                values_to = "grade")
```

```
# A tibble: 20 x 3
   name  assignment grade
   <chr> <chr>      <dbl>
 1 John  hw1           60
 2 John  hw2           89
 3 John  hw3           93
 4 John  test1         85
 5 John  test2         89
 6 Mary  hw1           89
 7 Mary  hw2           93
 8 Mary  hw3           75
 9 Mary  test1         90
10 Mary  test2         82
11 Ben   hw1           76
12 Ben   hw2           98
13 Ben   hw3           83
14 Ben   test1         87
15 Ben   test2         76
16 Steph hw1           88
17 Steph hw2           81
18 Steph hw3           87
19 Steph test1         90
20 Steph test2         95
```

pivot_wider()

pivot_wider() takes data in a “long” format and spreads the data into a wide format.

Now the third iteration of the same function: dcast(), spread(), pivot_wider()

It is the inverse of pivot_longer()

To see how it works, let’s take the data back to its original form!

pivot_wider()

```
tb2
```

```
# A tibble: 20 x 3
   name  assignment grade
   <chr> <chr>      <dbl>
 1 John  hw1           60
 2 John  hw2           89
 3 John  hw3           93
 4 John  test1         85
 5 John  test2         89
 6 Mary  hw1           89
 7 Mary  hw2           93
 8 Mary  hw3           75
 9 Mary  test1         90
10 Mary  test2         82
11 Ben   hw1           76
12 Ben   hw2           98
13 Ben   hw3           83
14 Ben   test1         87
15 Ben   test2         76
16 Steph hw1           88
17 Steph hw2           81
18 Steph hw3           87
19 Steph test1         90
20 Steph test2         95
```

```
tb2 %>%
  pivot_wider(names_from = assignment,
              values_from = grade)
```

```
# A tibble: 4 x 6
  name    hw1   hw2   hw3 test1 test2
  <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
1 John     60    89    93    85    89
2 Mary     89    93    75    90    82
3 Ben      76    98    83    87    76
4 Steph    88    81    87    90    95
```

More Examples!

Let’s look at other specifications and datasets here