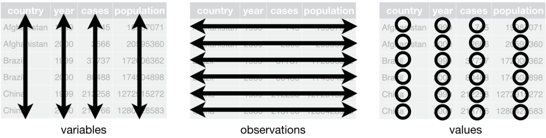
Tidy datasets

Definition of tidy data

Tidy datasets are all alike but every messy dataset is messy in its own way.

A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types. In tidy data:

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.



Ref: https://r4ds.had.co.nz/tidy-data.html Figure 12.1

Why do we need tidy data?

Tidy datasets are all alike but every messy dataset is messy in its own way. This clearly states that:

1. There is uniformity
2. R works in vectorised nature.

Are we ever going to work with untidy datasets? Answer is all the time...

What type of issues can we have with untidy datasets?

1. One variable might be spread across multiple columns.
2. One observation might be spread across multiple rows.

Example: Treatment

Table 1: Treatment a-b values for patients

	treatmenta	treatmentb
JS	-	2
JD	16	11
MJ	3	1

```
> treatmentdata <- tibble(
treatmenta = as.numeric(c(NaN, 16,3)),
treatmentb = c(12,11,1),
person = c("JS", "JD", "MJ"))
```

Table 2: Treatment a-b values for patients

	JS	JD	MJ
treatmenta	-	16	3
treatmentb	2	11	1

Table 2 shows the same data as Table 1, but the rows and columns have been transposed. The data is the same, but the layout is different.

Reorganised dataset

Table 3: Treatment a-b values for patients

person	treatment	result
JS	treatmenta	-
JS	treatmentb	2
JD	treatmenta	16
JD	treatmentb	11
MJ	treatmenta	3
MJ	treatmentb	1

Table 3 is the tidy version of Table 1. Each row represents an observation, the result of one treatment on one person, and each column is a variable.
We will see how we can obtain this tidy dataset in R.

Tidying messy datasets

Most common problems with messy datasets

1. Column headers are values, not variable names.
2. One variable might be spread across multiple columns.
3. A single observational unit might be scattered across multiple rows.
4. Multiple variables are stored in one column.
5. Variables are stored in both rows and columns.
6. Multiple types of observational units are stored in the same table.

To fix these problems, you'll need the two most important functions in tidyr: `pivot_longer()` and `pivot_wider()`

pivot_longer()

`pivot_longer()` makes datasets longer by increasing the number of rows and decreasing the number of columns.

1. Column headers are values, not variable names.
2. One variable might be spread across multiple columns.

```
> table4a
```

```
# A tibble: 3 x 3
country      `1999` `2000`
* <chr>        <int>  <int>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

```
> tidy4a <- table4a %>%
pivot_longer(c(`1999`, `2000`), names_to = "year",
values_to = "cases")
```

pivot_longer()

```
> table4b
```

```
# A tibble: 3 x 3
country       `1999`     `2000`
* <chr>         <int>      <int>
1 Afghanistan   19987071   20595360
2 Brazil       172006362  174504898
3 China       1272915272 1280428583
```

```
> tidy4b <- table4b %>%
pivot_longer(c(`1999`, `2000`), names_to = "year",
values_to = "population")
```

```
> tidy4a
```

```
# A tibble: 6 x 3
  country      year   cases
  <chr>        <chr>  <int>
1 Afghanistan  1999     745
2 Afghanistan  2000    2666
3 Brazil       1999   37737
4 Brazil       2000   80488
5 China        1999  212258
6 China        2000  213766
```

```
> tidy4b
```

```
# A tibble: 6 x 3
  country      year  population
  <chr>        <chr>      <int>
1 Afghanistan  1999    19987071
2 Afghanistan  2000    20595360
3 Brazil       1999   172006362
4 Brazil       2000   174504898
5 China        1999  1272915272
6 China        2000  1280428583
```

full_join()

```
> full_join(tidy4a, tidy4b)
```

```
Joining, by = c("country", "year")
# A tibble: 6 x 4
country      year   cases  population
<chr>        <chr>  <int>       <int>
1 Afghanistan 1999     745    19987071
2 Afghanistan 2000    2666    20595360
3 Brazil      1999   37737   172006362
4 Brazil      2000   80488   174504898
5 China       1999  212258  1272915272
6 China       2000  213766  1280428583
```

pivot_wider()

3. A single observational unit might be scattered across multiple rows.

```
> table2
```

```
# A tibble: 12 x 4
country      year type           count
<chr>       <int> <chr>          <int>
1 Afghanistan  1999 cases            745
2 Afghanistan  1999 population  19987071
3 Afghanistan  2000 cases           2666
4 Afghanistan  2000 population  20595360
5 Brazil       1999 cases          37737
6 Brazil       1999 population 172006362
7 Brazil       2000 cases          80488
8 Brazil       2000 population 174504898
9 China        1999 cases         212258
10 China        1999 population 1272915272
11 China        2000 cases         213766
12 China        2000 population 1280428583
```

pivot_wider()

- We need to tell R from which column the variable names are coming from.
- We also need to tell R from which column the variable values are coming from.
- For more examples and explanation see `vignette("pivot")`.

```
> table2 %>%
pivot_wider(names_from = type, values_from = count)
```

```
# A tibble: 6 x 4
country      year  cases population
<chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

separate()

4. Multiple variables are stored in one column.

```
> table3
```

```
# A tibble: 6 x 3
country      year rate
* <chr>       <int> <chr>
1 Afghanistan  1999 745/19987071
2 Afghanistan  2000 2666/20595360
3 Brazil       1999 37737/172006362
4 Brazil       2000 80488/174504898
5 China        1999 212258/1272915272
6 China        2000 213766/1280428583
```

separate()

4. Multiple variables are stored in one column.

```
> table3 %>%
separate(rate, into = c("cases", "population"), sep = "/",
    convert=TRUE)
```

```
# A tibble: 6 x 4
country        year   cases  population
<chr>         <int>   <int>       <int>
1 Afghanistan  1999     745    19987071
2 Afghanistan  2000    2666    20595360
3 Brazil       1999   37737   172006362
4 Brazil       2000   80488   174504898
5 China        1999  212258  1272915272
6 China        2000  213766  1280428583
```

`unite()`

`unite()` combines multiple columns into a single column.

Combining Datasets

Mini example 1

Table 4: Mutating joins

a		b	
x1	x2	x1	x3
A	1	A	T
B	2	B	F
C	3	D	T

```
> a <- tibble(x1 = c("A", "B", "C"),
             x2 = c(1, 2, 3)
               )
```

```
> b <- tibble(x1 = c("A", "B", "D"),
     x3 = c("T", "F", "T")
    )
```

left_join

Join all rows from the second dataset that match exactly to the first dataset.

```
> leftJoin <- left_join(a, b, by = "x1")
```

Table 5: Left join

a		b
x1	x2	x3
A	1	T
B	2	F
C	3	NA

right_join

Join all rows from the first dataset that match exactly to the second dataset.

```
> rightJoin <- right_join(a, b, by = "x1")
# try with by x2 and see what happens
```

Table 6: Right join

x1	x2	x3
A	1	T
B	2	F
D	NA	T

inner_join

Join the two datasets by retaining only the exact matching rows in both datasets.

```
> innerJoin <- inner_join(a, b, by = "x1")
```

Table 7: Inner join

a		b
x1	x2	x3
A	1	T
B	2	F

full_join

Join the two datasets by retaining all rows in both datasets.

```
> fullJoin <- full_join(a, b, by = "x1")
```

Table 8: Full join

a		b
x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

semi_join and anti_join

Retain all rows in the 1st dataset that have a match in the 2nd

```
> semiJoin <- semi_join(a, b, by = "x1")
```

Table 9: Semi join

a	
x1	x2
A	1
B	2

Retain only the rows in the 1st dataset that DO NOT have a match in the 2nd

```
> antiJoin <- anti_join(a, b, by = "x1")
```

Table 10: Anti join

a	
x1	x2
C	3

Mini example 2

Table 11: Set operations

y		z	
x1	x2	x1	x2
A	1	B	2
B	2	C	3
C	3	D	4

```
> y <- tibble(x1 = c("A", "B", "C"),
x2 = c(1, 2, 3)
)
```

```
> z <- tibble(x1 = c("B", "C", "D"),
x2 = c(2, 3, 4)
)
```

intersect()

Rows that appear in both y AND z (exact match) for all variables

```
> inters <- intersect(y, z)
```

Table 12: Intersected datasets

x1	x2
B	2
C	3

union()

Rows that appear in EITHER y OR z for all variables

```
> union <- union(y, z)
```

Table 13: Unioned datasets

x1	x2
A	1
B	2
C	3
D	4

What could be the potential error here?

setdiff()

Rows that appear in both y BUT NOT z for all variables

```
> setdifferent <- setdiff(y, z)
```

Table 14: Different observations

x1	x2
A	1

bind_rows()

Append both datasets along the rows without ANY MATCHING, datasets can have different variables, different number of rows.

```
> bindrows <- bind_rows(y, z)
```

Table 15: Row bind datasets

x1	x2
A	1
B	2
C	3
B	2
C	3
D	4

When would this be the most useful? What could be improved?

bind_rows() with id

Append both datasets along the rows without ANY MATCHING, and include which dataset the observation belongs to

```
> bindrows <- bind_rows(y, z, .id = "id")
```

Table 16: Row bind datasets with an id

id	x1	x2
1	A	1
1	B	2
1	C	3
2	B	2
2	C	3
2	D	4

bind_rows() with id as a year variable

Append both datasets along the rows without ANY MATCHING, and include which dataset the observation belongs to

```
> bindrows <- bind_rows("1990" = y, "2001" = z, .id = "year
    ")
```

Table 17: Row bind datasets with an id

year	x1	x2
1990	A	1
1990	B	2
1990	C	3
2001	B	2
2001	C	3
2001	D	4

Check the class for `year` variable!

bind_cols()

Append both datasets along columns without ANY MATCHING (except row index, need to have exact number of observations in both datasets)

```
> bindcolumns <- bind_cols(y, z)
```

Table 18: Column bind datasets

x1	x2	x1	x2
A	1	B	2
B	2	C	3
C	3	D	4

What would happen if the datasets have different number of observations?

Sample functions: training and test sets

Firstly, you need to have an ID per row, if you don't have one, use the following code to create an ID:

```
> df <- df %>% mutate(id = row_number())
```

Usually train/test set split is 70/30 or 80/20. To create a training set with no replacement::

```
> train <- df %>% sample_frac(.70, replace = FALSE)
```

In order to find those observations that are not in the training set, we can use anti join.

```
test  <- anti_join(df, train, by = 'id')
```

One final note

Some older functions don't work with tibbles. If you encounter one of these functions, use as.data.frame() to turn a tibble back to a data.frame:

```
> tibbledata <- tibble(x1 = c(1,2,3))
> class(tibbledata)
```

```
[1] "tbl_df"     "tbl"         "data.frame"
```

```
> dataframe <- as.data.frame(tibbledata)
> class(dataframe)
```

```
[1] "data.frame"
```

References

https://r4ds.had.co.nz/tidy-data.html
https://rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf
https://dplyr.tidyverse.org/reference/bind.html
https://simplystatistics.org/2016/02/17/non-tidy-data/

`vignette("pivot")`