Data Wrangling & RMarkdown

Mark Andrews
Psychology Department, Nottingham Trent University

✉ mark.andrews@ntu.ac.uk

What is data wrangling

- Also known as *data munging*, *data cleaning*, *data pre-processing*, *data preparation*, and so on, *data wrangling* is the process of taking data in its unstructured, messy, or complicated original form and converting it into a clean and tidy format that allows data exploration, visualization, and eventually statistical modelling and analysis to proceed efficiently and relatively effortlessly.
- Data science's *80/20* rule is that we spend up to 80% of time doing data wrangling, and the rest doing analysis per se.

Data frames, tibbles, and reading in data

- In everything we do here, and almost always in R generally, data is assumed to be stored in a *data frame*.
- Data frames are heterogeneous rectangular data structures. Columns are homogeneous vectors, but different columns may be of different types.
- The `tidyverse`'s style of data frame is known as a `tibble`. This is a regular R data frame but with some superficial and style changes.

Reading in data

- In practice, it is common to have data in a roughly rectangular format, i.e. with rows and columns, either in text files such as `.csv`, `.tsv`, `.txt`, files.
- The `readr` package, which is loaded when we load `tidyvese`, allows us to efficiently import data that are in these files.
- It includes commands
  - `read_csv` for files where the values on each line are separated by commas
  - `read_tsv` for files where the values are separated by tabs
  - `read_delim` for files where the values are separated by arbitrary delimiters such as '`|`', ':', ';', etc.
  - `read_table` for files where the values are separated by one or more, and possible inconsistently many, whitespaces.
- The `readxl::read_excel` can be used for Excel (`.xlsx`) files.

Width

```r
getOption('width')
```

```
## [1] 80
```

Our main example data-set

```r
library(tidyverse)
blp_df <- read_csv("data/blp-trials-short.txt")
blp_df
```

```
## # A tibble: 1,000 x 7
##    participant lex   spell    resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>    <chr> <dbl>   <dbl>  <dbl>
##  1          20 N     staud    N       977     511    977
##  2           9 N     dinbuss  N       565     765    565
##  3          47 N     snilling N       562     496    562
##  4         103 N     gancens  N       572     656    572
##  5          45 W     filled   W       659     981    659
##  6          73 W     journals W       538    1505    538
##  7          24 W     apache   W       626     546    626
##  8          11 W     flake    W       566     717    566
##  9          32 W     reliefs  W       922    1471    922
## 10          96 N     sarves   N       555     806    555
## # ... with 990 more rows
```

Glimpsing at data with `glimpse`

- We can use the `dplyr` command `glimpse` to look at resulting data frame.

```
glimpse(blp_df)
```

```
## Rows: 1,000
## Columns: 7
## $ participant <dbl> 20, 9, 47, 103, 45, 73, 24, 11, 32,...
## $ lex         <chr> "N", "N", "N", "N", "W", "W", "W", ...
## $ spell       <chr> "staud", "dinbuss", "snilling", "ga...
## $ resp        <chr> "N", "N", "N", "N", "W", "W", "W", ...
## $ rt          <dbl> 977, 565, 562, 572, 659, 538, 626, ...
## $ prev.rt     <dbl> 511, 765, 496, 656, 981, 1505, 546,...
## $ rt.raw      <dbl> 977, 565, 562, 572, 659, 538, 626, ...
```

- As we can see, there are 1000 rows and 7 variables.

The `dplyr` verbs

- The `dplyr` package provides a set of *verbs* for data wrangling:
  1. `select` for selecting variables
  2. `rename` for renaming variables
  3. `slice` for selecting rows by their indices
  4. `filter` for selecting rows by some condition
  5. `mutate` for adding or modifying variables
  6. `arrange` for sorting variables
- In addition, ther is `summarize` (with `group_by`) for summarizing variables.
- Each verb performs a major, if focused, task, and they can be chained together using *pipes*.

Selecting variables

- In our `blp_df` data frames we have 7 variables.
- Let's say, as is often the case when processing raw data, that we only need some of these.
- The `dplyr` command `select` allows us to select those we want.

Selecting variables: Example 1

- Here, we select some variables by name.

```
select(blp_df, participant, lex, resp, rt)
```

```
## # A tibble: 1,000 x 4
##    participant lex   resp     rt
##          <dbl> <chr> <chr> <dbl>
##  1          20 N     N       977
##  2           9 N     N       565
##  3          47 N     N       562
##  4         103 N     N       572
##  5          45 W     W       659
##  6          73 W     W       538
##  7          24 W     W       626
##  8          11 W     W       566
##  9          32 W     W       922
## 10          96 N     N       555
## # ... with 990 more rows
```

Selecting variables: Example 2

We can select a range of variables by specifying the first and last variables in the range with a `:` between them as follows.

```
select(blp_df, spell:prev.rt)
```

```
## # A tibble: 1,000 x 4
##    spell    resp      rt prev.rt
##    <chr>    <chr> <dbl>   <dbl>
##  1 staud    N         977     511
##  2 dinbuss  N         565     765
##  3 snilling N         562     496
##  4 gancens  N         572     656
##  5 filled   W         659     981
##  6 journals W         538    1505
##  7 apache   W         626     546
##  8 flake    W         566     717
##  9 reliefs  W         922    1471
## 10 sarves   N         555     806
## # ... with 990 more rows
```

Selecting variables: Example 3

We can select a range of variables using indices as in the following example.

```
select(blp_df, 2:5) # columns 2 to 5
```

```
## # A tibble: 1,000 x 4
##    lex   spell    resp     rt
##    <chr> <chr>    <chr> <dbl>
##  1 N     staud    N       977
##  2 N     dinbuss  N       565
##  3 N     snilling N       562
##  4 N     gancens  N       572
##  5 W     filled   W       659
##  6 W     journals W       538
##  7 W     apache   W       626
##  8 W     flake    W       566
##  9 W     reliefs  W       922
## 10 N     sarves   N       555
## # ... with 990 more rows
```

Selecting variables: Example 4

We can select variables according to the character or characters that they begin with.

```r
select(blp_df, starts_with('p'))
```

```
## # A tibble: 1,000 x 2
##    participant prev.rt
##          <dbl>   <dbl>
##  1          20     511
##  2           9     765
##  3          47     496
##  4         103     656
##  5          45     981
##  6          73    1505
##  7          24     546
##  8          11     717
##  9          32    1471
## 10          96     806
## # ... with 990 more rows
```

Selecting variables: Example 5

We can select variables by the characters they end with.

```
select(blp_df, ends_with('t'))
```

```
## # A tibble: 1,000 x 3
##    participant    rt prev.rt
##          <dbl> <dbl>   <dbl>
##  1          20   977     511
##  2           9   565     765
##  3          47   562     496
##  4         103   572     656
##  5          45   659     981
##  6          73   538    1505
##  7          24   626     546
##  8          11   566     717
##  9          32   922    1471
## 10          96   555     806
## # ... with 990 more rows
```

Selecting variables: Example 6

We can select variables that contain a certain set of characters in any position.

```
select(blp_df, contains('rt'))
```

```
## # A tibble: 1,000 x 4
##    participant    rt prev.rt rt.raw
##          <dbl> <dbl>   <dbl>  <dbl>
##  1          20   977     511    977
##  2           9   565     765    565
##  3          47   562     496    562
##  4         103   572     656    572
##  5          45   659     981    659
##  6          73   538    1505    538
##  7          24   626     546    626
##  8          11   566     717    566
##  9          32   922    1471    922
## 10          96   555     806    555
## # ... with 990 more rows
```

Selecting variables: Example 7

Using `matches`, the regular expression `^rt|rt$` will match the `rt` if it begins or ends a string.

```r
select(blp_df, matches('^rt|rt$'))
```

```
## # A tibble: 1,000 x 3
##       rt prev.rt rt.raw
##    <dbl>   <dbl>  <dbl>
##  1   977     511    977
##  2   565     765    565
##  3   562     496    562
##  4   572     656    572
##  5   659     981    659
##  6   538    1505    538
##  7   626     546    626
##  8   566     717    566
##  9   922    1471    922
## 10   555     806    555
## # ... with 990 more rows
```

Selecting variables: Example 8

To remove a variable, we precede its name with a minus sign.

```r
select(blp_df, -participant) # remove `participant`
```

```
## # A tibble: 1,000 x 6
##    lex   spell    resp     rt prev.rt rt.raw
##    <chr> <chr>    <chr> <dbl>   <dbl>  <dbl>
##  1 N     staud    N       977     511    977
##  2 N     dinbuss  N       565     765    565
##  3 N     snilling N       562     496    562
##  4 N     gancens  N       572     656    572
##  5 W     filled   W       659     981    659
##  6 W     journals W       538    1505    538
##  7 W     apache   W       626     546    626
##  8 W     flake    W       566     717    566
##  9 W     reliefs  W       922    1471    922
## 10 N     sarves   N       555     806    555
## # ... with 990 more rows
```

Selecting variables: Example 9

```
select(blp_df, -(2:6))
```

```
## # A tibble: 1,000 x 2
##    participant rt.raw
##          <dbl>  <dbl>
##  1          20    977
##  2           9    565
##  3          47    562
##  4         103    572
##  5          45    659
##  6          73    538
##  7          24    626
##  8          11    566
##  9          32    922
## 10          96    555
## # ... with 990 more rows
```

Selecting variables: Example 9

```r
select(blp_df, -contains('rt'))
```

```
## # A tibble: 1,000 x 3
##    lex   spell    resp
##    <chr> <chr>    <chr>
##  1 N     staud    N
##  2 N     dinbuss  N
##  3 N     snilling N
##  4 N     gancens  N
##  5 W     filled   W
##  6 W     journals W
##  7 W     apache   W
##  8 W     flake    W
##  9 W     reliefs  W
## 10 N     sarves   N
## # ... with 990 more rows
```

Selecting variables: Example 10

We can reorder variables using `everything()` as follows.

```r
select(blp_df, spell, participant, resp, everything())
```

```
## # A tibble: 1,000 x 7
##    spell    participant resp  lex      rt prev.rt rt.raw
##    <chr>          <dbl> <chr> <chr> <dbl>   <dbl>  <dbl>
##  1 staud             20 N     N       977     511    977
##  2 dinbuss            9 N     N       565     765    565
##  3 snilling          47 N     N       562     496    562
##  4 gancens          103 N     N       572     656    572
##  5 filled            45 W     W       659     981    659
##  6 journals          73 W     W       538    1505    538
##  7 apache            24 W     W       626     546    626
##  8 flake             11 W     W       566     717    566
##  9 reliefs           32 W     W       922    1471    922
## 10 sarves            96 N     N       555     806    555
## # ... with 990 more rows
```

The `_if`, `_at`, `_all` verb variants

- Almost all `dplyr` verbs have derivates — either `_if` or `_at` or `_all`, or all of them — that extend their power.
- For example, `select_if` function allows us to select variables according to properties of their values.

Selecting variables: Example 11

Here, we select all variables that are character vectors.

```
select_if(blp_df, is.character)
```

Selecting variables: Example 12

The following function will return `TRUE` if the variable is a numeric variable with a mean that is less than 700.

```r
has_low_mean <- function(x){
  is.numeric(x) && (mean(x, na.rm = T) < 700)
}
```

Selecting variables: Example 12

Now, we can select variables that meet this criterion as follows.

```
select_if(blp_df, has_low_mean)
```

```
## # A tibble: 1,000 x 3
##    participant    rt prev.rt
##          <dbl> <dbl>   <dbl>
##  1          20   977     511
##  2           9   565     765
##  3          47   562     496
##  4         103   572     656
##  5          45   659     981
##  6          73   538    1505
##  7          24   626     546
##  8          11   566     717
##  9          32   922    1471
## 10          96   555     806
## # ... with 990 more rows
```

Renaming variables

When we select individual variables with `select`, we can rename them too.

```
select(blp_df, subject=participant, reaction_time=rt)
```

```
## # A tibble: 1,000 x 2
##    subject reaction_time
##      <dbl>         <dbl>
##  1      20           977
##  2       9           565
##  3      47           562
##  4     103           572
##  5      45           659
##  6      73           538
##  7      24           626
##  8      11           566
##  9      32           922
## 10      96           555
## # ... with 990 more rows
```

Renaming variables

If we want to rename some variables, and get a data frame with all variables, including the renamed ones, we should use `rename`.

```r
rename(blp_df, subject=participant, reaction_time=rt)
```

```
## # A tibble: 1,000 x 7
##    subject lex   spell    resp  reaction_time prev.rt rt.raw
##      <dbl> <chr> <chr>    <chr>         <dbl>   <dbl>  <dbl>
##  1      20 N     staud    N               977     511    977
##  2       9 N     dinbuss  N               565     765    565
##  3      47 N     snilling N               562     496    562
##  4     103 N     gancens  N               572     656    572
##  5      45 W     filled   W               659     981    659
##  6      73 W     journals W               538    1505    538
##  7      24 W     apache   W               626     546    626
##  8      11 W     flake    W               566     717    566
##  9      32 W     reliefs  W               922    1471    922
## 10      96 N     sarves   N               555     806    555
## # ... with 990 more rows
```

Renaming all with `rename_all`

The `rename_all` function allows us to rename all the variables using some renaming function, e.g.,

```r
rename_all(blp_df, ~str_replace_all(., '\\.', '_'))
```

```
## # A tibble: 1,000 x 7
##    participant lex   spell    resp      rt prev_rt rt_raw
##          <dbl> <chr> <chr>    <chr> <dbl>   <dbl>  <dbl>
##  1          20 N     staud    N       977     511    977
##  2           9 N     dinbuss  N       565     765    565
##  3          47 N     snilling N       562     496    562
##  4         103 N     gancens  N       572     656    572
##  5          45 W     filled   W       659     981    659
##  6          73 W     journals W       538    1505    538
##  7          24 W     apache   W       626     546    626
##  8          11 W     flake    W       566     717    566
##  9          32 W     reliefs  W       922    1471    922
## 10          96 N     sarves   N       555     806    555
## # ... with 990 more rows
```

Renaming some with `rename_at`

In the following example, we select all variables whose names contain `rt` at their start or end, and then replace their occurrences of `rt` with `reaction_time`.

```
rename_at(blp_df,
          vars(matches('^rt|rt$')),
          ~str_replace_all(., 'rt', 'reaction_time'))
```

```
## # A tibble: 1,000 x 7
##    participant lex   spell resp  reaction_time prev.reaction_t~ reaction_time.r~
##          <dbl> <chr> <chr> <chr>         <dbl>            <dbl>            <dbl>
##  1          20 N     staud N               977              511              977
##  2           9 N     dinb~ N               565              765              565
##  3          47 N     snil~ N               562              496              562
##  4         103 N     ganc~ N               572              656              572
##  5          45 W     fill~ W               659              981              659
##  6          73 W     jour~ W               538             1505              538
##  7          24 W     apac~ W               626              546              626
##  8          11 W     flake W               566              717              566
##  9          32 W     reli~ W               922             1471              922
## 10          96 N     sarv~ N               555              806              555
## # ... with 990 more rows
```

Rename some with `rename_if`

Using `rename_if`, for example, if we wanted to capitalize the names of those variables that are character variables, we could do the following.

```
rename_if(blp_df, is.character, str_to_upper)
```

```
## # A tibble: 1,000 x 7
##    participant LEX   SPELL    RESP     rt prev.rt rt.raw
##          <dbl> <chr> <chr>    <chr> <dbl>   <dbl>  <dbl>
##  1          20 N     staud    N       977     511    977
##  2           9 N     dinbuss  N       565     765    565
##  3          47 N     snilling N       562     496    562
##  4         103 N     gancens  N       572     656    572
##  5          45 W     filled   W       659     981    659
##  6          73 W     journals W       538    1505    538
##  7          24 W     apache   W       626     546    626
##  8          11 W     flake    W       566     717    566
##  9          32 W     reliefs  W       922    1471    922
## 10          96 N     sarves   N       555     806    555
## # ... with 990 more rows
```

Slicing data frames

We use `slice` to select observations by their indices.

```
slice(blp_df, c(10, 20, 50, 100, 500))
```

```
## # A tibble: 5 x 7
##   participant lex   spell  resp     rt prev.rt rt.raw
##         <dbl> <chr> <chr>  <chr> <dbl>   <dbl>  <dbl>
## 1          96 N     sarves N       555     806    555
## 2          46 W     mirage W       778     571    778
## 3          72 N     gright N       430     675    430
## 4           3 W     gleam  W       361     370    361
## 5          92 W     coaxes W       699     990    699
```

Slice: Example 2

We can select consecutive ranges as follows.

```
slice(blp_df, 10:100)
```

```
## # A tibble: 91 x 7
##    participant lex   spell     resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>     <chr> <dbl>   <dbl>  <dbl>
##  1          96 N     sarves    N       555     806    555
##  2          82 W     deceits   W       657     728    657
##  3          37 W     nothings  N        NA     552    712
##  4          52 N     chuespies N       427     539    427
##  5          96 N     mowny     N      1352    1020   1352
##  6          96 N     cranned   N       907     573    907
##  7          89 N     flud      N       742     834    742
##  8           3 N     bromble   N       523     502    523
##  9           7 N     trubbles  N       782     458    782
## 10          35 N     playfound N       643     663    643
## # ... with 81 more rows
```

Slice: Example 3

Drop the first 10 rows.

```
slice(blp_df, -(1:10))
```

```
## # A tibble: 990 x 7
##    participant lex   spell     resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>     <chr> <dbl>   <dbl>  <dbl>
##  1          82 W     deceits   W       657     728    657
##  2          37 W     nothings  N        NA     552    712
##  3          52 N     chuespies N       427     539    427
##  4          96 N     mowny     N      1352    1020   1352
##  5          96 N     cranned   N       907     573    907
##  6          89 N     flud      N       742     834    742
##  7           3 N     bromble   N       523     502    523
##  8           7 N     trubbles  N       782     458    782
##  9          35 N     playfound N       643     663    643
## 10          46 W     mirage    W       778     571    778
## # ... with 980 more rows
```

Slice: Example 4

We can use `n()` to indicate the last row, as in the following example.

```r
slice(blp_df, 600:n())
```

```
## # A tibble: 401 x 7
##    participant lex   spell      resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>      <chr> <dbl>   <dbl>  <dbl>
##  1          16 W     earthworms W       767     659    767
##  2          50 W     markers    W       664     852    664
##  3          35 N     spoton     N       522     721    522
##  4          88 W     tawny      N        NA     535    856
##  5          51 N     gember     N       562     598    562
##  6          63 W     classed    W       706     429    706
##  7          63 N     clallers   N       401     495    401
##  8           8 W     pauper     W       734    1126    734
##  9           2 W     badges     W       485     498    485
## 10          97 N     foarded    N       802     464    802
## # ... with 391 more rows
```

Slice: Example 5

```r
slice(blp_df, (n()-10):n())
```

```
## # A tibble: 11 x 7
##    participant lex   spell     resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>     <chr> <dbl>   <dbl>  <dbl>
##  1          29 N     khandles  N       511     777    511
##  2          88 N     ixcurs    N       504     552    504
##  3          50 N     homply    N       518     583    518
##  4         103 W     baste     W       683     454    683
##  5          67 W     tall      W       476     572    476
##  6          45 W     gardens   W       586    1023    586
##  7         105 W     goldfinch N        NA     903    775
##  8          72 W     varmint   N        NA     507    653
##  9           3 W     lurked    W       537     520    537
## 10           3 W     village   W       538     522    538
## 11          17 W     fudge     W       410     437    410
```

Filtering data frames

The `filter` command is a powerful means to filter observations according to their values. For example, we can select all the observations where the `lex` variable is `N` as follows.

```
filter(blp_df, lex == 'N')
```

```
## # A tibble: 502 x 7
##    participant lex   spell     resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>     <chr> <dbl>   <dbl>  <dbl>
##  1          20 N     staud     N       977     511    977
##  2           9 N     dinbuss   N       565     765    565
##  3          47 N     snilling  N       562     496    562
##  4         103 N     gancens   N       572     656    572
##  5          96 N     sarves    N       555     806    555
##  6          52 N     chuespies N       427     539    427
##  7          96 N     mowny     N      1352    1020   1352
##  8          96 N     cranned   N       907     573    907
##  9          89 N     flud      N       742     834    742
## 10           3 N     bromble   N       523     502    523
## # ... with 492 more rows
```

Filter: Example 2

```
filter(blp_df, lex == 'N', resp=='W')
```

```
## # A tibble: 35 x 7
##    participant lex   spell    resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>    <chr> <dbl>   <dbl>  <dbl>
##  1          73 N     bunding  W        NA     978   1279
##  2          63 N     gallays  W        NA     589    923
##  3          50 N     droper   W        NA     741    573
##  4           6 N     flooder  W        NA     524    557
##  5          73 N     khantum  W        NA     623   1355
##  6          81 N     seaped   W        NA     765    691
##  7          43 N     gafers   W        NA     556    812
##  8         101 N     winchers W        NA     632    852
##  9          81 N     flaged   W        NA     674    609
## 10          11 N     frocker  W        NA     653    665
## # ... with 25 more rows
```

Filter: Example 3

```r
filter(blp_df, lex == 'N', resp=='W', rt.raw <= 500)
```

```
## # A tibble: 5 x 7
##   participant lex   spell    resp     rt prev.rt rt.raw
##         <dbl> <chr> <chr>    <chr> <dbl>   <dbl>  <dbl>
## 1          28 N     cown     W        NA     680    498
## 2          17 N     beeched  W        NA     450    469
## 3          29 N     conforn  W        NA     495    497
## 4          35 N     blear    W        NA     592    461
## 5          89 N     stumming W        NA     571    442
```

Filter: Example 4

This command is equivalent to making a conjunction of conditions using & as follows.

```r
filter(blp_df, lex == 'N' & resp=='W' & rt.raw <= 500)
```

```
## # A tibble: 5 x 7
##   participant lex   spell    resp     rt prev.rt rt.raw
##         <dbl> <chr> <chr>    <chr> <dbl>   <dbl>  <dbl>
## 1          28 N     cown     W        NA     680    498
## 2          17 N     beeched  W        NA     450    469
## 3          29 N     conforn  W        NA     495    497
## 4          35 N     blear    W        NA     592    461
## 5          89 N     stumming W        NA     571    442
```

Filter: Example 5

We can make a *disjunction* of conditions for filtering using the logical-or symbol `|`.

```
filter(blp_df, rt.raw < 500 | rt.raw > 1000)
```

```
## # A tibble: 296 x 7
##    participant lex   spell       resp      rt prev.rt rt.raw
##          <dbl> <chr> <chr>       <chr> <dbl>   <dbl>  <dbl>
##  1          52 N     chuespies   N       427     539    427
##  2          96 N     mowny       N      1352    1020   1352
##  3          28 W     stelae      N        NA     678    497
##  4          85 W     forewarned  N        NA     525    350
##  5          24 W     owl         W       470     535    470
##  6          97 W     soda        W       436     447    436
##  7          81 N     fugate      N       425     403    425
##  8         105 N     pamps       N        NA     884   1494
##  9          27 W     outgrowth   N        NA     633   1014
## 10          82 W     kitty       W       431     476    431
## # ... with 286 more rows
```

Filter: Example 6

```r
filter(blp_df, rt.raw %in% 500:510)
```

```
## # A tibble: 26 x 7
##    participant lex   spell       resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>       <chr> <dbl>   <dbl>  <dbl>
##  1          44 W     subscribed  W       509     475    509
##  2          89 W     snatcher    W       506    1004    506
##  3           2 N     tronculling N       508     490    508
##  4          43 N     trabnate    N       510     542    510
##  5          75 N     dousleens   N       508     924    508
##  6          94 W     strangeness W       508     522    508
##  7          68 W     greed       W       505     653    505
##  8          32 N     krifo       N       508     607    508
##  9           2 W     tweaks      W       508     474    508
## 10          85 N     waffs       N       506     471    506
## # ... with 16 more rows
```

Filter: Example 7

```
filter(blp_df,
       lex == 'W',
       str_length(spell) < 5 & (resp != lex | rt.raw > 900))
```

```
## # A tibble: 14 x 7
##    participant lex   spell resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr> <chr> <dbl>   <dbl>  <dbl>
##  1          21 W     bosk  N        NA     608   1532
##  2          68 W     wily  N        NA     723    636
##  3          30 W     sew   N        NA     473    524
##  4          34 W     jibs  N        NA     781    756
##  5          85 W     rote  N        NA     505    458
##  6          13 W     oofs  N        NA     560    654
##  7          72 W     awed  N        NA    1203   1801
##  8          14 W     yids  N        NA     625    620
##  9          68 W     oho   N        NA     633    630
## 10         103 W     carl  N        NA    1046   1042
## 11          46 W     brae  N        NA     644    720
## 12          81 W     bloc  N        NA     759    575
## 13          75 W     kind  W       903    1067    903
```

Filter: Example 8

Using `filter_all`, we can filter rows that contain at least one `NA` value.

```
filter_all(blp_df, any_vars(is.na(.)))
```

```
## # A tibble: 179 x 7
##    participant lex   spell      resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>      <chr> <dbl>   <dbl>  <dbl>
##  1          37 W     nothings   N        NA     552    712
##  2          28 W     stelae     N        NA     678    497
##  3          85 W     forewarned N        NA     525    350
##  4         105 N     pamps      N        NA     884   1494
##  5          27 W     outgrowth  N        NA     633   1014
##  6          89 W     chards     N        NA     545    754
##  7          63 N     shrudule   N        NA       0   2553
##  8          73 W     chiggers   N        NA     726    654
##  9          73 N     bunding    W        NA     978   1279
## 10          22 W     aitches    N        NA     521    665
## # ... with 169 more rows
```

Filter: Example 9

For example, the following filters all observations where the value of all variables that start or end with `rt` are greater than 500.

```
filter_at(blp_df, vars(matches('^rt|rt$')), all_vars(. > 500))
```

```
## # A tibble: 530 x 7
##    participant lex   spell    resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>    <chr> <dbl>   <dbl>  <dbl>
##  1          20 N     staud    N       977     511    977
##  2           9 N     dinbuss  N       565     765    565
##  3         103 N     gancens  N       572     656    572
##  4          45 W     filled   W       659     981    659
##  5          73 W     journals W       538    1505    538
##  6          24 W     apache   W       626     546    626
##  7          11 W     flake    W       566     717    566
##  8          32 W     reliefs  W       922    1471    922
##  9          96 N     sarves   N       555     806    555
## 10          82 W     deceits  W       657     728    657
## # ... with 520 more rows
```

Filter: Example 10

Values of the `rt` variables that are lower than the medians of their variables.

```
filter_at(blp_df,
          vars(matches('^rt|rt$')),
          all_vars(. < median(., na.rm=T)))
```

```
## # A tibble: 251 x 7
##    participant lex   spell     resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>     <chr> <dbl>   <dbl>  <dbl>
##  1          47 N     snilling  N       562     496    562
##  2          52 N     chuespies N       427     539    427
##  3           3 N     bromble   N       523     502    523
##  4          36 W     outposts  W       560     461    560
##  5          24 W     owl       W       470     535    470
##  6          97 W     soda      W       436     447    436
##  7          18 N     tesslier  N       560     477    560
##  8          81 N     fugate    N       425     403    425
##  9          29 N     placker   N       542     558    542
## 10          82 W     kitty     W       431     476    431
## # ... with 241 more rows
```

Filter: Example 11

Values of numeric variables that are lower than the medians of their variables.

```
filter_if(blp_df,
          is.numeric,
          all_vars(. < median(., na.rm=T)))
```

```
## # A tibble: 138 x 7
##    participant lex   spell     resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>     <chr> <dbl>   <dbl>  <dbl>
##  1           3 N     bromble   N       523     502    523
##  2          36 W     outposts  W       560     461    560
##  3          24 W     owl       W       470     535    470
##  4          18 N     tesslier  N       560     477    560
##  5          29 N     placker   N       542     558    542
##  6           6 N     checsons  N       491     555    491
##  7          19 N     jontage   N       413     471    413
##  8          44 W     snows     W       437     432    437
##  9          13 N     lavo      N       479     510    479
## 10          17 N     basyl     N       413     508    413
## # ... with 128 more rows
```

Adding or modifying variables

- The `mutate` command is a very powerful tool in the `dplyr` toolbox.
- It allows us to create new variables and alter the values of existing ones.

Mutate: Example 1

The following creates a new variable `acc`.

```
mutate(blp_df, acc = lex == resp)
```

```
## # A tibble: 1,000 x 8
##    participant lex   spell    resp     rt prev.rt rt.raw acc  
##          <dbl> <chr> <chr>    <chr> <dbl>   <dbl>  <dbl> <lgl>
##  1          20 N     staud    N       977     511    977 TRUE 
##  2           9 N     dinbuss  N       565     765    565 TRUE 
##  3          47 N     snilling N       562     496    562 TRUE 
##  4         103 N     gancens  N       572     656    572 TRUE 
##  5          45 W     filled   W       659     981    659 TRUE 
##  6          73 W     journals W       538    1505    538 TRUE 
##  7          24 W     apache   W       626     546    626 TRUE 
##  8          11 W     flake    W       566     717    566 TRUE 
##  9          32 W     reliefs  W       922    1471    922 TRUE 
## 10          96 N     sarves   N       555     806    555 TRUE 
## # ... with 990 more rows
```

Mutate: Example 2

```
mutate(blp_df, len = str_length(spell))
```

```
## # A tibble: 1,000 x 8
##    participant lex   spell    resp      rt prev.rt rt.raw   len
##          <dbl> <chr> <chr>    <chr> <dbl>   <dbl>  <dbl> <int>
##  1          20 N     staud    N         977     511    977     5
##  2           9 N     dinbuss  N         565     765    565     7
##  3          47 N     snilling N         562     496    562     8
##  4         103 N     gancens  N         572     656    572     7
##  5          45 W     filled   W         659     981    659     6
##  6          73 W     journals W         538    1505    538     8
##  7          24 W     apache   W         626     546    626     6
##  8          11 W     flake    W         566     717    566     5
##  9          32 W     reliefs  W         922    1471    922     7
## 10          96 N     sarves   N         555     806    555     6
## # ... with 990 more rows
```

Mutate: Example 3

```
mutate(blp_df,
       acc = lex == resp,
       fast = rt.raw < mean(rt.raw, na.rm=TRUE))
```

```
## # A tibble: 1,000 x 9
##    participant lex   spell    resp       rt prev.rt rt.raw acc   fast
##          <dbl> <chr> <chr>    <chr> <dbl>   <dbl>  <dbl> <lgl> <lgl>
##  1          20 N     staud    N       977     511    977 TRUE  FALSE
##  2           9 N     dinbuss  N       565     765    565 TRUE  TRUE
##  3          47 N     snilling N       562     496    562 TRUE  TRUE
##  4         103 N     gancens  N       572     656    572 TRUE  TRUE
##  5          45 W     filled   W       659     981    659 TRUE  TRUE
##  6          73 W     journals W       538    1505    538 TRUE  TRUE
##  7          24 W     apache   W       626     546    626 TRUE  TRUE
##  8          11 W     flake    W       566     717    566 TRUE  TRUE
##  9          32 W     reliefs  W       922    1471    922 TRUE  FALSE
## 10          96 N     sarves   N       555     806    555 TRUE  TRUE
## # ... with 990 more rows
```

Mutate: Example 4

Change all variables to character vectors.

```
mutate_all(blp_df, as.character)
```

```
## # A tibble: 1,000 x 7
##    participant lex   spell    resp  rt    prev.rt rt.raw
##    <chr>       <chr> <chr>    <chr> <chr> <chr>   <chr>
##  1 20          N     staud    N     977   511     977
##  2 9           N     dinbuss  N     565   765     565
##  3 47          N     snilling N     562   496     562
##  4 103         N     gancens  N     572   656     572
##  5 45          W     filled   W     659   981     659
##  6 73          W     journals W     538   1505    538
##  7 24          W     apache   W     626   546     626
##  8 11          W     flake    W     566   717     566
##  9 32          W     reliefs  W     922   1471    922
## 10 96          N     sarves   N     555   806     555
## # ... with 990 more rows
```

Mutate: Example 5

Apply a log transform to all the rt variables.

```
mutate_at(blp_df, vars(matches('^rt|rt$')), log)
```

```
## # A tibble: 1,000 x 7
##    participant lex   spell    resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>    <chr> <dbl>   <dbl>  <dbl>
##  1          20 N     staud    N      6.88    6.24   6.88
##  2           9 N     dinbuss  N      6.34    6.64   6.34
##  3          47 N     snilling N      6.33    6.21   6.33
##  4         103 N     gancens  N      6.35    6.49   6.35
##  5          45 W     filled   W      6.49    6.89   6.49
##  6          73 W     journals W      6.29    7.32   6.29
##  7          24 W     apache   W      6.44    6.30   6.44
##  8          11 W     flake    W      6.34    6.58   6.34
##  9          32 W     reliefs  W      6.83    7.29   6.83
## 10          96 N     sarves   N      6.32    6.69   6.32
## # ... with 990 more rows
```

Mutate: Example 6

Change all character vectors to factors.

```
mutate_if(blp_df, is.character, as.factor)
```

```
## # A tibble: 1,000 x 7
##    participant lex   spell    resp     rt prev.rt rt.raw
##          <dbl> <fct> <fct>    <fct> <dbl>   <dbl>  <dbl>
##  1          20 N     staud    N       977     511    977
##  2           9 N     dinbuss  N       565     765    565
##  3          47 N     snilling N       562     496    562
##  4         103 N     gancens  N       572     656    572
##  5          45 W     filled   W       659     981    659
##  6          73 W     journals W       538    1505    538
##  7          24 W     apache   W       626     546    626
##  8          11 W     flake    W       566     717    566
##  9          32 W     reliefs  W       922    1471    922
## 10          96 N     sarves   N       555     806    555
## # ... with 990 more rows
```

Mutate: Example 8

Create a new variable speed that takes the value of fast if rt.raw is less than 750, and slow otherwise.

```
mutate(blp_df,
       speed = if_else(rt.raw < 750,
                       'fast',
                       'slow')
)
```

```
## # A tibble: 1,000 x 8
##    participant lex   spell    resp     rt prev.rt rt.raw speed
##          <dbl> <chr> <chr>    <chr> <dbl>   <dbl>  <dbl> <chr>
##  1          20 N     staud    N       977     511    977 slow
##  2           9 N     dinbuss  N       565     765    565 fast
##  3          47 N     snilling N       562     496    562 fast
##  4         103 N     gancens  N       572     656    572 fast
##  5          45 W     filled   W       659     981    659 fast
##  6          73 W     journals W       538    1505    538 fast
##  7          24 W     apache   W       626     546    626 fast
##  8          11 W     flake    W       566     717    566 fast
##  9          32 W     reliefs  W       922    1471    922 slow
## 10          96 N     sarves   N       555     806    555 fast
## # ... with 990 more rows
```

Mutate: Example 9

Replace the `lex` variable's values `W` and `N` with `word` and `nonword`.

```
mutate(blp_df,
       lex = recode(lex, 'W'='word', 'N'='nonword')
)
```

```
## # A tibble: 1,000 x 7
##    participant lex     spell    resp     rt prev.rt rt.raw
##          <dbl> <chr>   <chr>    <chr> <dbl>   <dbl>  <dbl>
##  1          20 nonword staud    N       977     511    977
##  2           9 nonword dinbuss  N       565     765    565
##  3          47 nonword snilling N       562     496    562
##  4         103 nonword gancens  N       572     656    572
##  5          45 word    filled   W       659     981    659
##  6          73 word    journals W       538    1505    538
##  7          24 word    apache   W       626     546    626
##  8          11 word    flake    W       566     717    566
##  9          32 word    reliefs  W       922    1471    922
## 10          96 nonword sarves   N       555     806    555
## # ... with 990 more rows
```

Mutate: Example 10

We could use `case_when` to convert values of `prev.rt` that are below 500 to `fast`, and those above 1500 to `slow`, and those in between 500 and 1500 to `medium`.

```
mutate(blp_df,
       prev.rt = case_when(
                   prev.rt < 500 ~ 'fast',
                   prev.rt > 1500 ~ 'slow',
                   TRUE ~ 'medium'
       )
)
```

```
## # A tibble: 1,000 x 7
##    participant lex   spell    resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>    <chr> <dbl> <chr>    <dbl>
##  1          20 N     staud    N       977 medium     977
##  2           9 N     dinbuss  N       565 medium     565
##  3          47 N     snilling N       562 fast       562
##  4         103 N     gancens  N       572 medium     572
##  5          45 W     filled   W       659 medium     659
##  6          73 W     journals W       538 slow       538
##  7          24 W     apache   W       626 medium     626
##  8          11 W     flake    W       566 medium     566
##  9          32 W     reliefs  W       922 medium     922
## 10          96 N     sarves   N       555 medium     555
## # ... with 990 more rows
```

Mutate: Example 11

Use `plyr::mapvalues` to map a range of values to another.

```
mutate(blp_df,
       rt_reverse = plyr::mapvalues(rt, from=500:1000, to=1000:500)
)
```

```
## # A tibble: 1,000 x 8
##    participant lex   spell    resp     rt prev.rt rt.raw rt_reverse
##          <dbl> <chr> <chr>    <chr> <dbl>   <dbl>  <dbl>      <dbl>
##  1          20 N     staud    N       977     511    977        523
##  2           9 N     dinbuss  N       565     765    565        935
##  3          47 N     snilling N       562     496    562        938
##  4         103 N     gancens  N       572     656    572        928
##  5          45 W     filled   W       659     981    659        841
##  6          73 W     journals W       538    1505    538        962
##  7          24 W     apache   W       626     546    626        874
##  8          11 W     flake    W       566     717    566        934
##  9          32 W     reliefs  W       922    1471    922        578
## 10          96 N     sarves   N       555     806    555        945
## # ... with 990 more rows
```

Sorting data frames

We can sort data frames with `arrange`. For example to sort by `participant` and then by `spell`, we would do the following.

```
arrange(blp_df, participant, spell)
```

```
## # A tibble: 1,000 x 7
##    participant lex   spell     resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>     <chr> <dbl>   <dbl>  <dbl>
##  1           1 W     abyss     W       629     683    629
##  2           1 N     baisees   N       524     574    524
##  3           1 W     carport   W       779     605    779
##  4           1 N     cellies   N       792     652    792
##  5           1 W     chafing   W       601     720    601
##  6           1 N     dametails N       694     635    694
##  7           1 N     foother   N       789     566    789
##  8           1 W     gantries  W       644     581    644
##  9           1 N     hogtush   N       679     568    679
## 10           1 N     lisedess  N       679     619    679
## # ... with 990 more rows
```

Sorting in descending order

We can use `desc` to sort in descending order.

```
arrange(blp_df, participant, desc(spell))
```

```
## # A tibble: 1,000 x 7
##    participant lex   spell     resp     rt prev.rt rt.raw
##          <dbl> <chr> <chr>     <chr> <dbl>   <dbl>  <dbl>
##  1           1 N     wintes    N       545     629    545
##  2           1 N     treeps    N       607     610    607
##  3           1 W     squashes  W       494     491    494
##  4           1 N     sinkhicks N       536     519    536
##  5           1 W     shafting  W       553     571    553
##  6           1 W     month     W       500     498    500
##  7           1 N     lisedess  N       679     619    679
##  8           1 N     hogtush   N       679     568    679
##  9           1 W     gantries  W       644     581    644
## 10           1 N     foother   N       789     566    789
## # ... with 990 more rows
```

Summarizing variables

We can use `summarize` to calculate some summary statistics of particular variables.

```
summarize(blp_df,
          mean_rt = mean(rt, na.rm = T),
          median_rt = median(rt, na.rm = T),
          sd_rt.raw = sd(rt.raw, na.rm = T)
)
```

```
## # A tibble: 1 x 3
##   mean_rt median_rt sd_rt.raw
##     <dbl>     <dbl>     <dbl>
## 1    638.       588      474.
```

Summarize: Example 2

We can use the `summarize_all` variant of `summarize` to apply a summarisation function to all variables, as in the following example.

```r
summarize_all(blp_df, n_distinct)
```

```
## # A tibble: 1 x 7
##   participant   lex spell  resp    rt prev.rt rt.raw
##         <int> <int> <int> <int> <int>   <int>  <int>
## 1          78     2   990     2   421     493    516
```

Summarize: Example 3

In the following example, we calculate the mean of all the reaction times variables.

```
summarize_at(blp_df, vars(matches('^rt|rt$')), ~mean(., na.rm=T)
```

```
## # A tibble: 1 x 3
##        rt prev.rt rt.raw
##     <dbl>   <dbl>  <dbl>
## 1  638.    660.   708.
```

Summarize: Example 4

The `summarize_if` will apply the summary function to numeric variables.

```r
summarize_if(blp_df, is.numeric, ~mean(., na.rm=T))
```

```
## # A tibble: 1 x 4
##   participant    rt prev.rt rt.raw
##         <dbl> <dbl>   <dbl>  <dbl>
## 1        49.5  638.    660.   708.
```

Summarize: Example 5

In the following, we calculate the same three summary statistics for two variables.

```
summarise_at(blp_df,
             vars(rt, rt.raw),
             list(mean = ~mean(., na.rm=T),
                  median = ~median(., na.rm=T),
                  sd = ~sd(., na.rm=T)
             )
)
```

```
## # A tibble: 1 x 6
##   rt_mean rt.raw_mean rt_median rt.raw_median rt_sd rt.raw_sd
##     <dbl>       <dbl>     <dbl>         <dbl> <dbl>     <dbl>
## 1    638.        708.       588           605  191.      474.
```

The %>% pipe

- To understand pipes, let us begin with a very simple example.
- The following is a vector of the first 10 prime numbers.

```r
primes <- c(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
```

- We can calculate the sum of `primes` as follows.

```r
sum(primes)
```

```
## [1] 129
```

- We may then calculate the square root of this sum.

```r
sqrt(sum(primes))
```

```
## [1] 11.35782
```

- We may then calculate the logarithm of this square root.

```r
log(sqrt(sum(primes)))
```

```
## [1] 2.429906
```

The %>% pipe

- Using the pipe %>%, we can replace

```
log(sqrt(sum(primes)))
```

```
## [1] 2.429906
```

with

```
primes %>% sum() %>% sqrt() %>% log()
```

```
## [1] 2.429906
```

- In general, if we have a variable `x` and a function `f()`, then `f(x)` is equivalent to the following.

```
x %>% f()    # equivalent to f(x)
```

Grouped summaries

- Using `group_by` and `summarize` together is a powerful way to create new (reduced) data frames.
- For example, for each participant, and for each of the two stimuli types (i.e. the `N` and `W` values of `lex`), we can calculate summary statistics.

```
group_by(blp_df, participant, lex) %>%
summarize(n_stimuli = n(),
          correct_resp = sum(resp == lex, na.rm=T),
          reaction_time = mean(rt.raw, na.rm=T))
```

```
## # A tibble: 156 x 5
## # Groups:   participant [78]
##    participant lex   n_stimuli correct_resp reaction_time
##          <dbl> <chr>     <int>        <int>         <dbl>
##  1           1 N             9            9          649.
##  2           1 W             7            7          600
##  3           2 N             7            6          625.
##  4           2 W             6            5          477.
##  5           3 N             4            4          540.
##  6           3 W             8            7          529
##  7           4 N             5            5          589.
##  8           4 W             5            4          465.
##  9           5 N             1            1          495
## 10           5 W             3            2          571
## # ... with 146 more rows
```

Pipelines: Example 1

```
blp_df %>%
  mutate(accuracy = resp == lex,
         stimulus = recode(lex, 'W'='word', 'N'='nonword')
  ) %>%
  select(participant, stimulus, item=spell, accuracy, speed=rt.raw) %>%
  arrange(participant, speed)
```

```
## # A tibble: 1,000 x 5
##    participant stimulus item      accuracy speed
##          <dbl> <chr>    <chr>     <lgl>    <dbl>
##  1           1 word     squashes  TRUE       494
##  2           1 word     month     TRUE       500
##  3           1 nonword  baisees   TRUE       524
##  4           1 nonword  sinkhicks TRUE       536
##  5           1 nonword  wintes    TRUE       545
##  6           1 word     shafting  TRUE       553
##  7           1 word     chafing   TRUE       601
##  8           1 nonword  treeps    TRUE       607
##  9           1 word     abyss     TRUE       629
## 10           1 word     gantries  TRUE       644
## # ... with 990 more rows
```

Pipelines: Example 2

```
blp_df %>%
  filter(lex == 'W') %>%
  mutate(word_length = str_length(spell),
         accuracy = resp == lex) %>%
  rename(speed = rt.raw) %>%
  group_by(word_length) %>%
  summarize_at(vars(accuracy, speed), ~mean(., na.rm=T)) %>%
  ungroup() %>%
  select(word_length, accuracy, speed) %>%
  arrange(word_length, accuracy, speed)
```

```
## # A tibble: 9 x 3
##   word_length accuracy speed
##         <int>    <dbl> <dbl>
## 1           3    0.7    551.
## 2           4    0.744  649.
## 3           5    0.718  825.
## 4           6    0.807  723.
## 5           7    0.821  704.
## 6           8    0.835  678.
## 7           9    0.595  914.
## 8          10    0.714  670.
## 9          11    0.5    700.
```

Combining data frames

- There are two main ways to combine data frames: *binds* and *joins*.
- A *bind* operation is a simple operation that either vertically stack data frames that share common variables, or horizontally stack data frames that have the same number of observations.
- A *join* merges data frames by shared variables.

Binds

- To illustrate, we will create three small data frames.

```
Df_1 <- tibble(x = c(1, 2, 3),
               y = c(2, 7, 1),
               z = c(0, 2, 7))

Df_2 <- tibble(y = c(5, 7),
               z = c(6, 7),
               x = c(1, 2))

Df_3 <- tibble(a = c(5, 6, 1),
               b = c('a', 'b', 'c'),
               c = c(T, T, F))
```

Bind rows

- The `Df_1` and `Df_2` data frames share common variable names.
- They can be vertically stacked using a `bind_rows` operation.

```r
bind_rows(Df_1, Df_2)
```

```
## # A tibble: 5 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1     1     2     0
## 2     2     7     2
## 3     3     1     7
## 4     1     5     6
## 5     2     7     7
```

Bind columns

- The Df_1 and Df_3 data frames have the same number of observations and so can be stacked side by side with a bind_cols operation.

```
bind_cols(Df_1, Df_3)
```

```
## # A tibble: 3 x 6
##       x     y     z     a b     c
##   <dbl> <dbl> <dbl> <dbl> <chr> <lgl>
## 1     1     2     0     5 a     TRUE
## 2     2     7     2     6 b     TRUE
## 3     3     1     7     1 c     FALSE
```

Combining data frames by joins

- A *join* operation is a common operation in relational databases using SQL.
- It allows us to join separate tables according to shared keys.
- As an example, consider `stimuli`. It has a `spell` variable like `blp_df` file, and an three additional variables.

```
stimuli <- read_csv('data/blp_stimuli.csv')
stimuli
```

```
## # A tibble: 55,865 x 4
##    spell   old20   bnc subtlex
##    <chr>   <dbl> <dbl>   <dbl>
##  1 a/c      1.95    14       0
##  2 aas      1.55     9       1
##  3 aback    1.85   327      15
##  4 abaft    2        8       2
##  5 aband    1.95     0       0
##  6 abase    1.7      6       2
##  7 abased   1.75     6       0
##  8 abashed  1.85    57       0
##  9 abate    1.75    69       5
## 10 abates   1.75     9       2
## # ... with 55,855 more rows
```

Join with inner_join

- We can join blp_df and stimuli with inner_join.
- An inner_join operation, searches through the values of variables that are shared by the two data frames in order to find matching values.
- In blp_df and stimuli, there is just one shared variable, namely spell.
- Thus, aninner_join of blp_df and stimuli will find values of spell on the left hand data frame that occur as values of spell on the right hand side.

Join with inner_join

```
inner_join(blp_df, stimuli)
```

```
## # A tibble: 1,000 x 10
##    participant lex   spell    resp     rt prev.rt rt.raw old20   bnc
##          <dbl> <chr> <chr>    <chr> <dbl>   <dbl>  <dbl> <dbl> <dbl>
##  1          20 N     staud    N       977     511    977  1.85     0
##  2           9 N     dinbuss  N       565     765    565  2.9      0
##  3          47 N     snilling N       562     496    562  1.8      0
##  4         103 N     gancens  N       572     656    572  2.3      0
##  5          45 W     filled   W       659     981    659  1.45  5340
##  6          73 W     journals W       538    1505    538  2.7   1030
##  7          24 W     apache   W       626     546    626  2.45   130
##  8          11 W     flake    W       566     717    566  1.5    274
##  9          32 W     reliefs  W       922    1471    922  2.25   185
## 10          96 N     sarves   N       555     806    555  1.65     0
## # ... with 990 more rows
```

Inner, Left, right, full joins

- In general, in an `inner_join`, if the left hand data frame has no values on the shared variables that match those on the right hand data frame, the observations from the left hand data frame are dropped.
- In addition, all observations on the right hand data frame that do not have matching observations on the left always get dropped too. However, consider the following two data frames.

```r
Df_a <- tibble(x = c(1, 2, 3),
               y = c('a', 'b', 'c'))
Df_b <- tibble(x = c(2, 3, 4),
               z = c('d', 'e', 'f'))
```

Inner, Left, right, full joins

- The first value of `x` in `Df_a` does not match any value of `x` in `Df_b`, and so the corresponding observation is dropped in an `inner_join`.

```
inner_join(Df_a, Df_b)
```

```
## # A tibble: 2 x 3
##       x y     z
##   <dbl> <chr> <chr>
## 1     2 b     d
## 2     3 c     e
```

Left join

- A left_join, on the other hand, will preserve all values on the left and put NA as the corresponding values of the right's variables if there are no matching values.

```
left_join(Df_a, Df_b)
```

```
## # A tibble: 3 x 3
##       x y     z
##   <dbl> <chr> <chr>
## 1     1 a     <NA>
## 2     2 b     d
## 3     3 c     e
```

Right join

- A `right_join` preserves all observations from the right, and places `NA` as the corresponding values of variables from the left that are not matched.

```
right_join(Df_a, Df_b)
```

```
## # A tibble: 3 x 3
##       x y     z
##   <dbl> <chr> <chr>
## 1     2 b     d
## 2     3 c     e
## 3     4 <NA>  f
```

Full joins

- A `full_join` keeps all observation in both the left and right data frames.

```r
full_join(Df_a, Df_b)
```

```
## # A tibble: 4 x 3
##       x y     z
##   <dbl> <chr> <chr>
## 1     1 a     <NA>
## 2     2 b     d
## 3     3 c     e
## 4     4 <NA>  f
```

Specifying the `by` variables

- Consider the following cases.

```
Df_4 <- tibble(x = c(1, 2, 3),
               y = c(2, 7, 1),
               z = c(0, 2, 7))

Df_5 <- tibble(a = c(1, 1, 7),
               b = c(2, 3, 7),
               c = c('a', 'b', 'c'))
```

Specifying the `by` variables

- In the following example, we look for matches between `x` and `y` on the left and `a` and `b` on the right.

```r
inner_join(Df_4, Df_5, by=c('x' = 'a', 'y' = 'b'))
```

```
## # A tibble: 1 x 4
##       x     y     z c    
##   <dbl> <dbl> <dbl> <chr>
## 1     1     2     0 a    
```

Reshaping with `pivot_longer` and `pivot_wider`

- A so-called *tidy* data set is a data set where all rows are observations, all columns are variables, and each variable is a single value.
- Consider the following data frame.

```r
recall_df <- read_csv('data/repeated_measured_a.csv')
recall_df
```

```
## # A tibble: 5 x 4
##   Subject   Neg   Neu   Pos
##   <chr>   <dbl> <dbl> <dbl>
## 1 Faye       26    12    42
## 2 Jason      29     8    35
## 3 Jim        32    15    45
## 4 Ron        22    10    38
## 5 Victor     30    13    40
```

- In this data frame, the `Neg`, `Neu`, `Pos` are, in fact, *values* of a variable, namely the condition of the experiment.

pivot_longer

```r
recall_long <- pivot_longer(recall_df,
                            cols = -Subject,
                            names_to = 'condition',
                            values_to = 'score')
recall_long
```

```
## # A tibble: 15 x 3
##    Subject condition score
##    <chr>   <chr>     <dbl>
##  1 Faye    Neg          26
##  2 Faye    Neu          12
##  3 Faye    Pos          42
##  4 Jason   Neg          29
##  5 Jason   Neu           8
##  6 Jason   Pos          35
##  7 Jim     Neg          32
##  8 Jim     Neu          15
##  9 Jim     Pos          45
## 10 Ron     Neg          22
## 11 Ron     Neu          10
## 12 Ron     Pos          38
## 13 Victor  Neg          30
## 14 Victor  Neu          13
## 15 Victor  Pos          40
```

pivot_wider

- The inverse of a `pivot_longer` is a `pivot_wider`.

```
pivot_wider(recall_long, names_from = 'condition', values_from =
```

```
## # A tibble: 5 x 4
##   Subject   Neg   Neu   Pos
##   <chr>   <dbl> <dbl> <dbl>
## 1 Faye       26    12    42
## 2 Jason      29     8    35
## 3 Jim        32    15    45
## 4 Ron        22    10    38
## 5 Victor     30    13    40
```

pivot_longer: example 2

Consider the following data.

```
recall_2_df <- read_csv('data/repeated_measured_b.csv')
recall_2_df
```

```
## # A tibble: 5 x 7
##   Subject Cued_Neg Cued_Neu Cued_Pos Free_Neg Free_Neu Free_P
##   <chr>      <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <db
## 1 Faye          15       16       14       13       13
## 2 Jason          4        9       10        6        7
## 3 Jim            7        9       10        8        9
## 4 Ron           17       18       20       12       14
## 5 Victor        16       13       14       12       13
```

pivot_longer: example 2

```r
pivot_longer(recall_2_df,
             cols = -Subject,
             names_to = c('cue', 'emotion'),
             names_pattern = '(Cued|Free)_(Neg|Pos|Neu)',
             values_to = 'score')
```

```
## # A tibble: 30 x 4
##    Subject cue   emotion score
##    <chr>   <chr> <chr>   <dbl>
##  1 Faye    Cued  Neg        15
##  2 Faye    Cued  Neu        16
##  3 Faye    Cued  Pos        14
##  4 Faye    Free  Neg        13
##  5 Faye    Free  Neu        13
##  6 Faye    Free  Pos        12
##  7 Jason   Cued  Neg         4
##  8 Jason   Cued  Neu         9
##  9 Jason   Cued  Pos        10
## 10 Jason   Free  Neg         6
## # ... with 20 more rows
```

pivot_longer: example 3

- Consider the following data frame from earlier.

```
tmp_df <- summarise_at(blp_df,
             vars(rt:rt.raw),
             list(avg = ~mean(., na.rm = T),
                  med = ~median(., na.rm = T),
                  stdev = ~sd(., na.rm = T))
)
tmp_df
```

```
## # A tibble: 1 x 9
##   rt_avg prev.rt_avg rt.raw_avg rt_med prev.rt_med rt.raw_med rt_stdev
##    <dbl>       <dbl>      <dbl>  <dbl>       <dbl>      <dbl>    <dbl>
## 1   638.        660.       708.    588         594        605     191.
## # ... with 2 more variables: prev.rt_stdev <dbl>, rt.raw_stdev <dbl>
```

pivot_longer: example 3

```
tmp_df %>% pivot_longer(cols = everything(),
                        names_to = c('variable', '.value'),
                        names_pattern = "(.*)_(.*)")
```

```
## # A tibble: 3 x 4
##   variable   avg   med stdev
##   <chr>    <dbl> <dbl> <dbl>
## 1 rt        638.   588  191.
## 2 prev.rt   660.   594  253.
## 3 rt.raw    708.   605  474.
```