04_talk_pdf

Akshay Bareja

Importing data into R

The `readr` package (found in the `tidyverse` collection) contains a number of useful functions of the form `read_*` to import data. For example, if you have a `.csv` file, you would use the `read_csv` function

Download a file from uniprot.org

After selecting some columns of interest, click the Download button and download as a compressed Text file

Rename the file to something simple (yet informative!), like `uniprot` and make sure the extension is `.tsv`

To import into RStudio, run the following

```
uniprot <- read_tsv("uniprot.tsv")
```

```
## Parsed with column specification:
## cols(
##   Entry = col_character(),
##   `Gene names` = col_character(),
##   Length = col_double()
## )
```

You can also use the `readr` package to import data from a URL

For example, to load a dataset from the (very useful) Tidy Tuesday series, run the following

```
pizza <- read_csv("https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2019/2019-
```

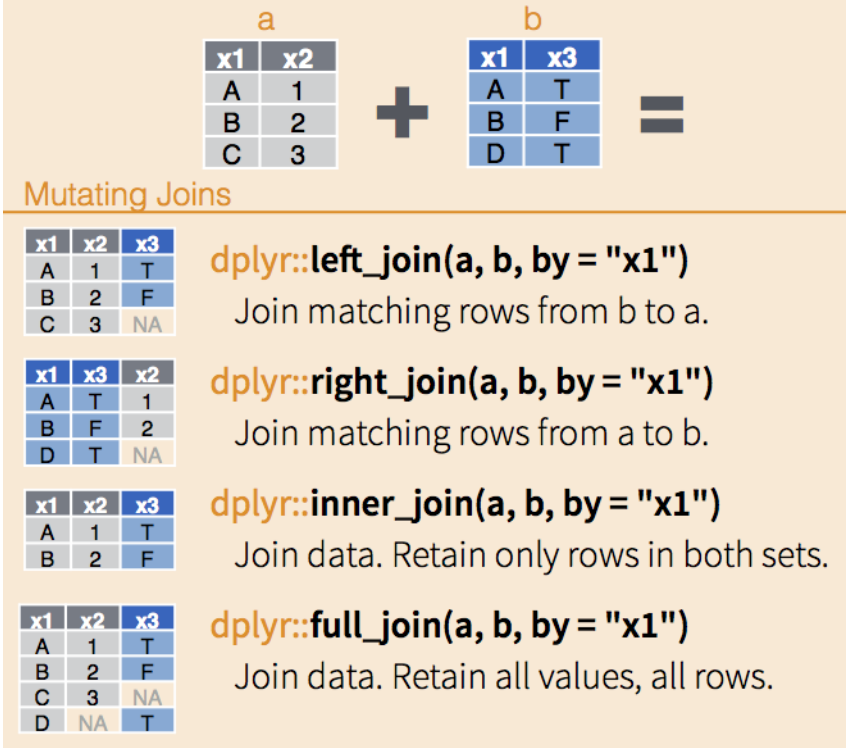
This data set contains ratings of various pizzerias in Manhattan

Combining datasets

There are many times when you have two or more overlapping datasets that you would like to combine

The `dplyr` package has a number of `*_join` functions for this purpose

---



`left_join`

Returns all rows from x, and all columns from x and y

Rows in x with no match in y will have NA values in the new columns

If there are multiple matches between x and y, all combinations of the matches are returned

First, load the two datasets needed for this example - `proteins` and `genes`

```
library(tidybiology)
data(proteins)
data(genes)
```

Take a look at the variables in each dataset

`gene_name`, which contains the gene IDs for each gene is a common variable

Let's join on this

`left_join` example

`left_join` `proteins` with `genes` and assign the output to a new object called `proteins_genes_left`

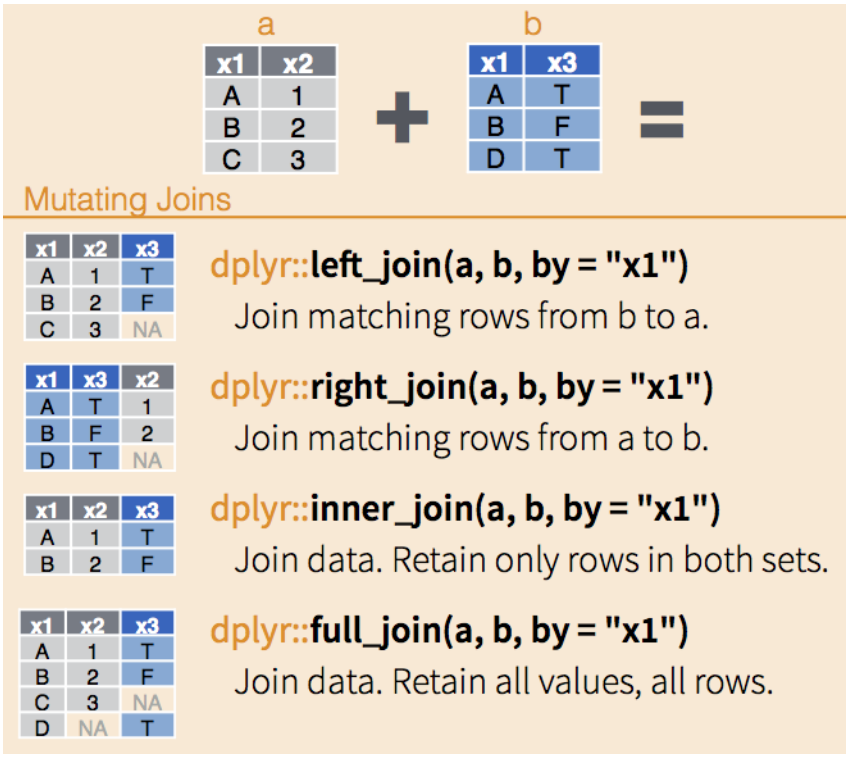
```
proteins_genes_left <- left_join(proteins, genes, by = "gene_name")
proteins_genes_left %>% head(1)
```

```
## # A tibble: 1 x 17
##   uniprot_id gene_name gene_name_alt protein_name protein_name_alt sequence
```

```
##   <chr>      <chr>     <chr>          <chr>         <chr>             <chr>
## 1 P04217     A1BG      <NA>           "Alpha-1B-g~ Alpha-1-B glyco~ MSMLVVF~
## # ... with 11 more variables: length <dbl>, mass <dbl>,
## #   gene_description <chr>, chromosome_scaffold_name <chr>, strand <dbl>,
## #   transcript_start_bp <dbl>, transcript_end_bp <dbl>,
## #   transcript_length <dbl>, gene_percent_gc_content <dbl>,
## #   gene_stable_id <chr>, transcript_stable_id <chr>
```

Now you have one dataset with additional useful information, like %GC content

---



`right_join`

Returns all rows from y, and all columns from x and y

Rows in y with no match in x will have NA values in the new columns

If there are multiple matches between x and y, all combinations of the matches are returned

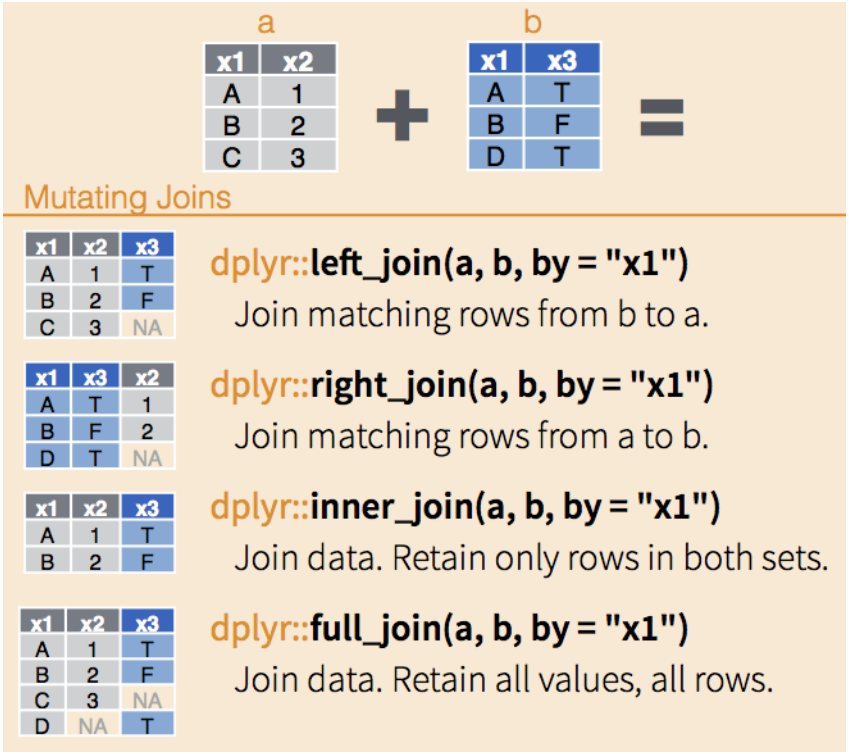
`right_join` example

`right_join` `proteins` with `genes` and assign the output to a new object called `proteins_genes_right`

```
proteins_genes_right <- right_join(proteins, genes, by = "gene_name")
proteins_genes_right %>% head(1)
```

```
## # A tibble: 1 x 17
##   uniprot_id gene_name gene_name_alt protein_name protein_name_alt sequence
##   <chr>      <chr>     <chr>         <chr>        <chr>             <chr>
## 1 <NA>       DDX11L1   <NA>          <NA>         <NA>              <NA>
## # ... with 11 more variables: length <dbl>, mass <dbl>,
## #   gene_description <chr>, chromosome_scaffold_name <chr>, strand <dbl>,
## #   transcript_start_bp <dbl>, transcript_end_bp <dbl>,
## #   transcript_length <dbl>, gene_percent_gc_content <dbl>,
## #   gene_stable_id <chr>, transcript_stable_id <chr>
```

You will notice a lot of `NA`s in the first few columns. Why might this be?



`inner_join`

Returns all rows from x where there are matching values in y, and all columns from x and y

If there are multiple matches between x and y, all combination of the matches are returned

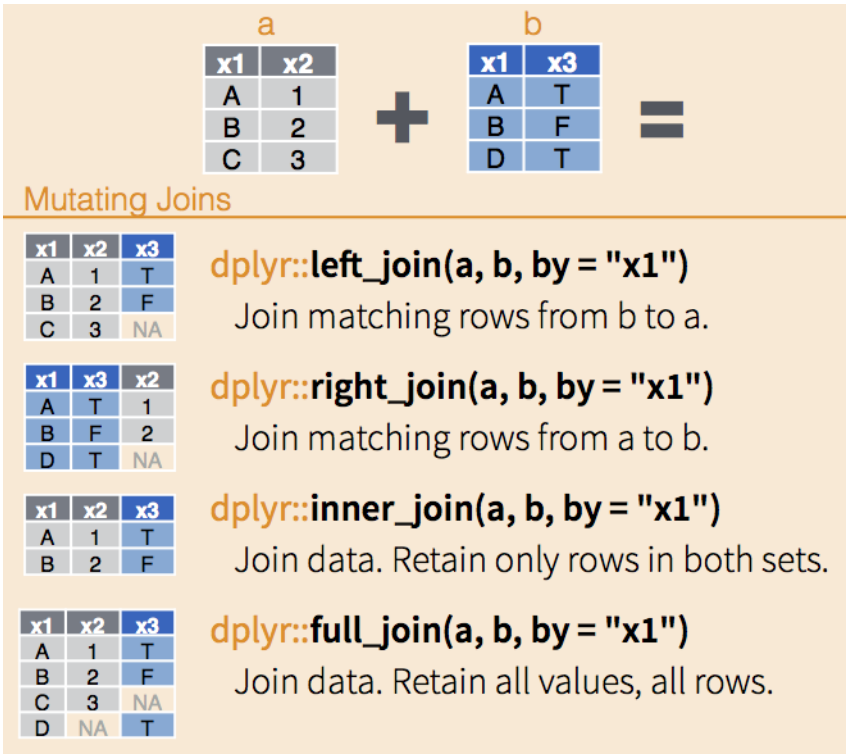
`inner_join` example

`inner_join` `proteins` with `genes` and assign the output to a new object called `proteins_genes_inner`

```
proteins_genes_inner <- inner_join(proteins, genes, by = "gene_name")
proteins_genes_inner %>% head(1)
```

```
## # A tibble: 1 x 17
##   uniprot_id gene_name gene_name_alt protein_name protein_name_alt sequence
##   <chr>      <chr>     <chr>         <chr>        <chr>             <chr>
## 1 P04217     A1BG      <NA>          "Alpha-1B-g~ Alpha-1-B glyco~ MSMLVVF~
## # ... with 11 more variables: length <dbl>, mass <dbl>,
## #   gene_description <chr>, chromosome_scaffold_name <chr>, strand <dbl>,
## #   transcript_start_bp <dbl>, transcript_end_bp <dbl>,
## #   transcript_length <dbl>, gene_percent_gc_content <dbl>,
## #   gene_stable_id <chr>, transcript_stable_id <chr>
```

Why might this type of join be useful?



`full_join`

Returns all rows and all columns from both x and y

Where there are no matching values, returns NA for the one missing

`full_join` example

`full_join` `proteins` with `genes` and assign the output to a new object called `proteins_genes_full`

```
proteins_genes_full <- full_join(proteins, genes, by = "gene_name")
proteins_genes_full %>% head(1)
```

```
## # A tibble: 1 x 17
##   uniprot_id gene_name gene_name_alt protein_name protein_name_alt sequence
##   <chr>      <chr>     <chr>         <chr>        <chr>             <chr>
## 1 P04217     A1BG      <NA>          "Alpha-1B-g~ Alpha-1-B glyco~ MSMLVVF~
## # ... with 11 more variables: length <dbl>, mass <dbl>,
## #   gene_description <chr>, chromosome_scaffold_name <chr>, strand <dbl>,
## #   transcript_start_bp <dbl>, transcript_end_bp <dbl>,
## #   transcript_length <dbl>, gene_percent_gc_content <dbl>,
## #   gene_stable_id <chr>, transcript_stable_id <chr>
```