Introduction to dplyr

Akshay Bareja

Loading the `proteins` and `mitocarta` datasets into RStudio

The datasets can be found in the `proteins` and `mitocarta` packages on the Hirsheylab Github page

To install and load the packages, run the following

```
devtools::install_github("hirscheylab/proteins")
devtools::install_github("hirscheylab/mitocarta")
library(proteins)
library(mitocarta)
```

Inspecting the `proteins` dataset

Use the `dim()` function to see how many rows (observations) and columns (variables) there are

```
dim(proteins)
```

```
## [1] 20430     8
```

Inspecting the `proteins` dataset

Use the `glimpse()` function to see what kinds of variables the dataset contains

```
glimpse(proteins)
```

```
## Observations: 20,430
## Variables: 8
## $ uniprot_id       <chr> "P04217", "Q9NQ94", "P01023", "A8K2U0", "U3KPV4", ...
## $ gene_name        <chr> "A1BG", "A1CF", "A2M", "A2ML1", "A3GALT2", "A4GALT...
## $ gene_name_alt    <chr> NA, "ACF ASP", "CPAMD5 FWP007", "CPAMD9", "A3GALT2...
## $ protein_name     <chr> "Alpha-1B-glycoprotein ", "APOBEC1 complementation...
## $ protein_name_alt <chr> "Alpha-1-B glycoprotein)", "APOBEC1-stimulating pr...
## $ sequence         <chr> "MSMLVVFLLLWGVTWGPVTEAAIFYETQPSLWAESESLLKPLANVTLTC...
## $ length           <dbl> 495, 594, 1474, 1454, 340, 353, 340, 546, 672, 399...
## $ mass             <dbl> 54254, 65202, 163291, 161107, 38754, 40499, 39497,...
```

Basic Data Types in R

R has 6 basic data ypes -

character - `"a"`, `"tidyverse"`

numeric - `2`, `11.5`

integer - 2L (the `L` tells R to store this as an integer)

logical - `TRUE`, `FALSE`

complex - `1+4i`

(**raw**)

You will also come across the **double** datatype. It is the same as **numeric**

factor. A **factor** is a collection of *ordered* character variables

Basic Data Types in R

In addition to the `glimpse()` function, you can use the `class()` function to determine the data type of a specific column

```
class(proteins$length)
```

```
## [1] "numeric"
```

(Re)Introducing %>%

The %>% operator is a way of "chaining" together strings of commands that make reading your code easy. The following code chunk illustrates how %>% works

```
proteins %>%
  select(uniprot_id, length) %>%
  filter(length > 500) %>%
  head(1)
```

```
## # A tibble: 1 x 2
##   uniprot_id length
##   <chr>       <dbl>
## 1 Q9NQ94        594
```

The above code chunk does the following - it takes you dataset, `proteins`, and "pipes" it into `select()`

(Re)Introducing %>%

The second line selects just the columns named `uniprot_id` and `length` and "pipes" that into `filter()`. The final line selects proteins that are longer than 500 amino acids

When you see %>%, think "and then"

The alternative to using %>% is running the following code

```
filter(select(proteins, uniprot_id, length), length > 500)
```

Although this is only one line as opposed to three, it's both more difficult to write and more difficult to read

Introducing the main dplyr verbs

dplyr is a package that contains a suite of functions that allow you to easily manipulate a dataset

Some of the things you can do are -

- select rows and columns that match specific criteria
- create new variables (columns)
- obtain summary statistics on individual groups within your datsets

The main verbs we will cover are `select()`, `filter()`, `arrange()`, `mutate()`, and `summarise()`. These all combine naturally with `group_by()` which allows you to perform any operation "by group"

`select()`

The `select()` verb allows you to extract specific columns from your dataset

The most basic `select()` is one where you comma separate a list of columns you want included

For example, if you only want to select the `uniprot_id` and `length` columns, run the following code chunk

```r
proteins %>%
  select(uniprot_id, length) %>%
  head(1)
```

```
## # A tibble: 1 x 2
##   uniprot_id length
##   <chr>       <dbl>
## 1 P04217        495
```

`select()`

If you want to select all columns *except* `uniprot_id`, run the following

```r
proteins %>%
  select(-uniprot_id) %>%
  head(1)
```

```
## # A tibble: 1 x 7
##   gene_name gene_name_alt protein_name  protein_name_alt sequence   length  mass
##   <chr>     <chr>         <chr>         <chr>            <chr>       <dbl> <dbl>
## 1 A1BG      <NA>          "Alpha-1B-gl~ Alpha-1-B glyco~ MSMLVVFLL~    495 54254
```

`select()`

Finally, you can provide a range of columns to return two columns and everything in between. For example

```r
proteins %>%
  select(uniprot_id:protein_name) %>%
  head(1)
```

```
## # A tibble: 1 x 4
##   uniprot_id gene_name gene_name_alt protein_name
##   <chr>      <chr>     <chr>         <chr>
## 1 P04217     A1BG      <NA>          "Alpha-1B-glycoprotein "
```

This code selects the following columns - `uniprot_id`, `gene_name`, `gene_name_alt`, and `protein_name`

`select()` exercise

Select the following columns - `uniprot_id`, `sequence`, `length`, and `mass`

```r
proteins %>%
  select(uniprot_id, sequence:mass)
```

`filter()`

The `filter()` verb allows you to choose rows based on certain condition(s) and discard everything else

All filters are performed on some logical statement

If a row meets the condition of this statement (i.e. is true) then it gets chosen (or "filtered"). All other rows are discarded

`filter()`

Filtering can be performed on categorical data

```r
mitocarta %>%
  filter(mito_domain_score == "MitoDomain") %>%
  head(1)
```

```
## # A tibble: 1 x 43
##   training_dataset human_gene_id mouse_ortholog_~ symbol synonyms description
##   <chr>                    <dbl>            <dbl> <chr>  <chr>    <chr>
## 1 Tmito                       33            11363 ACADL  ACAD4|L~ acyl-CoA d~
## # ... with 37 more variables: ensembl_gene_id <chr>, protein_length <dbl>,
## #   target_p_score <dbl>, mito_domain_score <chr>,
## #   coexpression_gnf_n50_score <dbl>, pgc_induction_score <dbl>,
## #   yeast_mito_homolog_score <chr>, rickettsia_homolog_score <chr>,
## #   msms_score <chr>, mcarta2_score <dbl>, mcarta2_fdr <dbl>,
## #   mcarta2_list <dbl>, mcarta2_evidence <chr>, hg19_chromosome <fct>,
## #   hg19_start <dbl>, hg19_stop <dbl>, msms_num_tissues <dbl>,
## #   msms_num_peptides_unique <dbl>, msms_num_spectra <dbl>,
## #   msms_total_intensity <dbl>, msms_percent_coverage <dbl>, tissues <chr>,
## #   cerebrum_total_peak_intensity_log10 <dbl>,
## #   cerebellum_total_peak_intensity_log10 <dbl>,
## #   brainstem_total_peak_intensity_log10 <dbl>,
## #   spinalcord_total_peak_intensity_log10 <dbl>,
## #   kidney_total_peak_intensity_log10 <dbl>,
## #   liver_total_peak_intensity_log10 <dbl>,
## #   heart_total_peak_intensity_log10 <dbl>,
## #   skeletalmuscle_total_peak_intensity_log10 <dbl>,
## #   adipose_total_peak_intensity_log10 <dbl>,
## #   smallintestine_total_peak_intensity_log10 <dbl>,
## #   largeintestine_total_peak_intensity_log10 <dbl>,
## #   stomach_total_peak_intensity_log10 <dbl>,
## #   placenta_total_peak_intensity_log10 <dbl>,
## #   testis_total_peak_intensity_log10 <dbl>,
## #   hpa_primary_subcellular_localization_2015 <chr>
```

The code chunk above only shows you proteins with a mito domain score that is equal to MitoDomain

Note that `filter()` only applies to rows, and has no effect on columns

`filter()`

Filtering can also be performed on numerical data

For example, to select proteins with a `mcarta2_fdr` value that is less than 0.05, run the following code

```r
mitocarta %>%
  filter(mcarta2_fdr < 0.05) %>%
  head(1)
```

```
## # A tibble: 1 x 43
##   training_dataset human_gene_id mouse_ortholog_~ symbol synonyms description
##   <chr>                    <dbl>            <dbl> <chr>  <chr>    <chr>
## 1 Tmito                       18           268860 ABAT   GABA-AT~ 4-aminobut~
## # ... with 37 more variables: ensembl_gene_id <chr>, protein_length <dbl>,
## #   target_p_score <dbl>, mito_domain_score <chr>,
## #   coexpression_gnf_n50_score <dbl>, pgc_induction_score <dbl>,
## #   yeast_mito_homolog_score <chr>, rickettsia_homolog_score <chr>,
## #   msms_score <chr>, mcarta2_score <dbl>, mcarta2_fdr <dbl>,
## #   mcarta2_list <dbl>, mcarta2_evidence <chr>, hg19_chromosome <fct>,
## #   hg19_start <dbl>, hg19_stop <dbl>, msms_num_tissues <dbl>,
## #   msms_num_peptides_unique <dbl>, msms_num_spectra <dbl>,
## #   msms_total_intensity <dbl>, msms_percent_coverage <dbl>, tissues <chr>,
## #   cerebrum_total_peak_intensity_log10 <dbl>,
## #   cerebellum_total_peak_intensity_log10 <dbl>,
## #   brainstem_total_peak_intensity_log10 <dbl>,
## #   spinalcord_total_peak_intensity_log10 <dbl>,
## #   kidney_total_peak_intensity_log10 <dbl>,
## #   liver_total_peak_intensity_log10 <dbl>,
## #   heart_total_peak_intensity_log10 <dbl>,
## #   skeletalmuscle_total_peak_intensity_log10 <dbl>,
## #   adipose_total_peak_intensity_log10 <dbl>,
## #   smallintestine_total_peak_intensity_log10 <dbl>,
## #   largeintestine_total_peak_intensity_log10 <dbl>,
## #   stomach_total_peak_intensity_log10 <dbl>,
## #   placenta_total_peak_intensity_log10 <dbl>,
## #   testis_total_peak_intensity_log10 <dbl>,
## #   hpa_primary_subcellular_localization_2015 <chr>
```

`filter()`

To filter on multiple conditions, you can write a sequence of `filter()` commands

For example, to choose proteins with a mito domain score that is equal to MitoDomain **and** a `mcarta2_fdr` value that is less than 0.05

```
mitocarta %>%
  filter(mito_domain_score == "MitoDomain") %>%
  filter(mcarta2_fdr < 0.05) %>%
  head(1)
```

```
## # A tibble: 1 x 43
##   training_dataset human_gene_id mouse_ortholog_~ symbol synonyms description
##   <chr>                    <dbl>            <dbl> <chr>  <chr>    <chr>
## 1 Tmito                       33            11363 ACADL  ACAD4|L~ acyl-CoA d~
## # ... with 37 more variables: ensembl_gene_id <chr>, protein_length <dbl>,
## #   target_p_score <dbl>, mito_domain_score <chr>,
## #   coexpression_gnf_n50_score <dbl>, pgc_induction_score <dbl>,
## #   yeast_mito_homolog_score <chr>, rickettsia_homolog_score <chr>,
## #   msms_score <chr>, mcarta2_score <dbl>, mcarta2_fdr <dbl>,
## #   mcarta2_list <dbl>, mcarta2_evidence <chr>, hg19_chromosome <fct>,
## #   hg19_start <dbl>, hg19_stop <dbl>, msms_num_tissues <dbl>,
## #   msms_num_peptides_unique <dbl>, msms_num_spectra <dbl>,
## #   msms_total_intensity <dbl>, msms_percent_coverage <dbl>, tissues <chr>,
## #   cerebrum_total_peak_intensity_log10 <dbl>,
## #   cerebellum_total_peak_intensity_log10 <dbl>,
## #   brainstem_total_peak_intensity_log10 <dbl>,
## #   spinalcord_total_peak_intensity_log10 <dbl>,
## #   kidney_total_peak_intensity_log10 <dbl>,
## #   liver_total_peak_intensity_log10 <dbl>,
## #   heart_total_peak_intensity_log10 <dbl>,
## #   skeletalmuscle_total_peak_intensity_log10 <dbl>,
## #   adipose_total_peak_intensity_log10 <dbl>,
## #   smallintestine_total_peak_intensity_log10 <dbl>,
## #   largeintestine_total_peak_intensity_log10 <dbl>,
## #   stomach_total_peak_intensity_log10 <dbl>,
## #   placenta_total_peak_intensity_log10 <dbl>,
## #   testis_total_peak_intensity_log10 <dbl>,
## #   hpa_primary_subcellular_localization_2015 <chr>
```

`filter()`

To avoid writing multiple `filter()` commands, multiple logical statements can be put inside a single `filter()` command, separated by commas

```
mitocarta %>%
  filter(mito_domain_score == "MitoDomain",
         mcarta2_fdr < 0.05) %>%
  head(1)
```

```
## # A tibble: 1 x 43
##   training_dataset human_gene_id mouse_ortholog_~ symbol synonyms description
##   <chr>                    <dbl>            <dbl> <chr>  <chr>    <chr>
## 1 Tmito                       33            11363 ACADL  ACAD4|L~ acyl-CoA d~
## # ... with 37 more variables: ensembl_gene_id <chr>, protein_length <dbl>,
## #   target_p_score <dbl>, mito_domain_score <chr>,
## #   coexpression_gnf_n50_score <dbl>, pgc_induction_score <dbl>,
## #   yeast_mito_homolog_score <chr>, rickettsia_homolog_score <chr>,
## #   msms_score <chr>, mcarta2_score <dbl>, mcarta2_fdr <dbl>,
## #   mcarta2_list <dbl>, mcarta2_evidence <chr>, hg19_chromosome <fct>,
## #   hg19_start <dbl>, hg19_stop <dbl>, msms_num_tissues <dbl>,
## #   msms_num_peptides_unique <dbl>, msms_num_spectra <dbl>,
## #   msms_total_intensity <dbl>, msms_percent_coverage <dbl>, tissues <chr>,
## #   cerebrum_total_peak_intensity_log10 <dbl>,
## #   cerebellum_total_peak_intensity_log10 <dbl>,
## #   brainstem_total_peak_intensity_log10 <dbl>,
## #   spinalcord_total_peak_intensity_log10 <dbl>,
## #   kidney_total_peak_intensity_log10 <dbl>,
## #   liver_total_peak_intensity_log10 <dbl>,
## #   heart_total_peak_intensity_log10 <dbl>,
## #   skeletalmuscle_total_peak_intensity_log10 <dbl>,
## #   adipose_total_peak_intensity_log10 <dbl>,
## #   smallintestine_total_peak_intensity_log10 <dbl>,
## #   largeintestine_total_peak_intensity_log10 <dbl>,
## #   stomach_total_peak_intensity_log10 <dbl>,
## #   placenta_total_peak_intensity_log10 <dbl>,
## #   testis_total_peak_intensity_log10 <dbl>,
## #   hpa_primary_subcellular_localization_2015 <chr>
```

`filter()` exercise

Filter all proteins with a mito domain score that is **not** equal to MitoDomain **and** a `mcarta2_fdr` value that is **greater** than 0.05

`!=` = "not equal to"

`<=` = "less than or equal to"

```
mitocarta %>%
  filter(mito_domain_score != "MitoDomain",
         mcarta2_fdr > 0.05)
```

`arrange()`

You can use the `arrange()` verb to sort rows

The input for arrange is one or many columns, and `arrange()` sorts the rows in ascending order i.e. from smallest to largest

For example, to sort rows from smallest to largest protein, run the following

```
proteins %>%
  arrange(length) %>%
  head(3)
```

```
## # A tibble: 3 x 8
##   uniprot_id gene_name gene_name_alt protein_name protein_name_alt sequence
##   <chr>      <chr>     <chr>         <chr>        <chr>            <chr>
## 1 P0DPR3     TRDD1     <NA>          T cell rece~ <NA>             EI
## 2 P0DPI4     TRBD1     <NA>          T cell rece~ <NA>             GTGG
## 3 P01858     <NA>      <NA>          "Phagocytos~ Tuftsin)         TKPR
## # ... with 2 more variables: length <dbl>, mass <dbl>
```

`arrange()`

To reverse this order, use the `desc()` function within `arrange()`

```
proteins %>%
  arrange(desc(length)) %>%
  head(3)
```

```
## # A tibble: 3 x 8
##   uniprot_id gene_name gene_name_alt protein_name protein_name_alt sequence
##   <chr>      <chr>     <chr>         <chr>        <chr>            <chr>
## 1 Q8WZ42     TTN       <NA>          "Titin "     EC 2.7.11.1) (C~ MTTQAPT~
## 2 Q8WXI7     MUC16     CA125         "Mucin-16 "  MUC-16) (Ovaria~ MLKPSGL~
## 3 Q8NF91     SYNE1     C6orf98 KIAA~ "Nesprin-1 " Enaptin) (KASH ~ MATSRGA~
## # ... with 2 more variables: length <dbl>, mass <dbl>
```

`arrange()` exercise

What happens when you apply `arrange()` to a categorical variable?

```
proteins %>%
  arrange(gene_name_alt) %>%
  head(6)
```

```
## # A tibble: 6 x 8
##   uniprot_id gene_name gene_name_alt protein_name protein_name_alt sequence
##   <chr>      <chr>     <chr>         <chr>        <chr>            <chr>
## 1 O14569     CYB561D2  101F6 LUCA12~ "Cytochrome~ EC 7.2.1.3) (Pu~ MALSAET~
## 2 P18054     ALOX12    12LO LOG12    "Arachidona~ 12S-LOX) (12S-l~ MGRYRIR~
## 3 O43715     TRIAP1    15E1.1 HSPC1~ "TP53-regul~ Protein 15E1.1)~ MNSVGEA~
## 4 O43716     GATC      15E1.2        Glutamyl-tR~ Gln) amidotrans~ MWSRLVW~
## 5 Q14596     NBR1      1A13B KIAA00~ "Next to BR~ Cell migration-~ MEPQVTL~
## 6 O14931     NCR3      1C7 LY117     "Natural cy~ Activating natu~ MAWMLLL~
## # ... with 2 more variables: length <dbl>, mass <dbl>
```

`mutate()`

The `mutate()` verb, unlike the ones covered so far, creates new variable(s) i.e. new column(s). For example

```
proteins %>%
  mutate(sqrt_length = sqrt(length)) %>%
  head(1)
```

```
## # A tibble: 1 x 9
##   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 3 more variables: length <dbl>, mass <dbl>, sqrt_length <dbl>
```

The code chunk above takes all the elements of the column `length`, evaluates the square root of each element, and populates a new column called `sqrt_length` with these results

`mutate()`

Multiple columns can be used as inputs. For example

```
proteins %>%
  mutate(protein_length_mass = length/mass) %>%
  head(1)
```

```
## # A tibble: 1 x 9
##   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 3 more variables: length <dbl>, mass <dbl>,
## #   protein_length_mass <dbl>
```

This code takes the length of each protein and divides it by its mass

The results are stored in a new column called `protein_length_mass`

`mutate()` exercise

Create a new column (give it any name you like) and fill it with protein lengths divided by 100

```
proteins %>%
  mutate(protein_length_100 = length/100)
```

`summarise()`

`summarise()` produces a new dataframe that aggregates that values of a column based on a certain condition.

For example, to calculate the mean protein length and mass, run the following

```
proteins %>%
  summarise(mean(length), mean(mass))
```

```
## # A tibble: 1 x 2
##   `mean(length)` `mean(mass)`
##            <dbl>        <dbl>
## 1           557.       62061.
```

`summarise()`

You can assign your own names by running the following

```r
proteins %>%
  summarise(mean_length = mean(length),
            mean_mass = mean(mass))
```

```
## # A tibble: 1 x 2
##   mean_length mean_mass
##         <dbl>     <dbl>
## 1        557.    62061.
```

`summarise()` exercise

Make a new table that contains the mean, median and standard deviations of protein lengths

Use the `median()` and `sd()` functions to calculate median and standard deviation

```r
proteins %>%
  summarise(protein_mean = mean(length),
            protein_median = median(length),
            protein_sd = sd(length))
```

```
## # A tibble: 1 x 3
##   protein_mean protein_median protein_sd
##          <dbl>          <dbl>      <dbl>
## 1         557.            414       596.
```

`group_by()`

`group_by()` and `summarise()` can be used in combination to summarise by groups

For example, if you'd like to know the mean length of both mitochondrial and non mitochondrial proteins, run the following

```r
mitocarta %>%
  group_by(mcarta2_list) %>%
  summarise(mean(protein_length))
```

```
## # A tibble: 2 x 2
##   mcarta2_list `mean(protein_length)`
##          <dbl>                  <dbl>
## 1            0                   590.
## 2            1                   400.
```

Saving a new dataset

If you'd like to save the output of your wrangling, you will need to use the `<-` or `->` operators

```r
mito_new <- mitocarta %>%
              group_by(mcarta2_list) %>%
              summarise(mean(protein_length))
```

To save `mito_new` as a new file (e.g. csv)

```r
write_csv(mito_new, "mito_new.csv")
```

For more help

Run the following to access the Dplyr vignette

```r
browseVignettes("dplyr")
```