Aggregating and Analyzing Data with dplyr

Overview

Teaching: 40 min
Exercises: 15 min
Questions

- How can I manipulate dataframes without repeating myself?

Objectives

- Describe what the `dplyr` package in R is used for.
- Apply common `dplyr` functions to manipulate data in R.
- Employ the 'pipe' operator to link together a sequence of functions.
- Employ the 'mutate' function to apply other chosen functions to existing columns and create new columns of data.
- Employ the 'split-apply-combine' concept to split the data into groups, apply analysis to each group, and combine the results.

Bracket subsetting is handy, but it can be cumbersome and difficult to read, especially for complicated operations.

Luckily, the [`dplyr`](https://cran.r-project.org/package=dplyr package provides a number of very useful functions for manipulating dataframes in a way that will reduce repetition, reduce the probability of making errors, and probably even save you some typing. As an added bonus, you might even find the `dplyr` grammar easier to read.

Here we're going to cover 6 of the most commonly used functions as well as using pipes (`%>%`) to combine them.

1. `select()`
2. `filter()`
3. `group_by()`
4. `summarize()`
5. `mutate()`

Packages in R are sets of additional functions that let you do more stuff in R. The functions we've been using, like `str()` , come built into R; packages give you access to more functions. You need to install a package and then load it to be able to use it.

R

```
install.packages("dplyr") ## install
```

You might get asked to choose a CRAN mirror – this is asking you to choose a site to download the package from. The choice doesn't matter too much; I'd recommend choosing the RStudio mirror.

R

```
library("dplyr")          ## load
```

You only need to install a package once per computer, but you need to load it every time you open a new R session and want to use that package.

What is dplyr?

The package `dplyr` is a fairly new (2014) package that tries to provide easy tools for the most common data manipulation tasks. It is built to work directly with data frames. The thinking behind it was largely inspired by the package `plyr` which has been in use for some time but suffered from being slow in some cases. `dplyr` addresses this by porting much of the computation to C++. An additional feature is the ability to work with data stored directly in an external database. The benefits of doing this are that the data can be managed natively in a relational database, queries can be conducted on that database, and only the results of the query returned.

This addresses a common problem with R in that all operations are conducted in memory and thus the amount of data you can work with is limited by available memory. The database connections essentially remove that limitation in that you can have a database of many 100s GB, conduct queries on it directly and pull back just what you need for analysis in R.

Selecting columns and filtering rows

To select columns of a data frame, use `select()` . The first argument to this function is the data frame (`variants`), and the subsequent arguments are the columns to keep.

R

```
select(variants, sample_id, REF, ALT, DP)
```

Output

```
# A tibble: 801 x 4
   sample_id  REF                  ALT                                 DP
   <chr>      <chr>                <chr>                            <dbl>
 1 SRR2584863 T                    G                                    4
 2 SRR2584863 G                    T                                    6
 3 SRR2584863 G                    T                                   10
 4 SRR2584863 CTTTTTTT             CTTTTTTTT                           12
 5 SRR2584863 CCGC                 CCGCGC                              10
 6 SRR2584863 C                    T                                   10
 7 SRR2584863 C                    A                                    8
 8 SRR2584863 G                    A                                   11
 9 SRR2584863 ACAGCCAGCCAGCCAGCCA… ACAGCCAGCCAGCCAGCCAGCCAGCCAGCCAGC…   3
10 SRR2584863 AT                   ATT                                  7
# … with 791 more rows
```

To select all columns *except* certain ones, put a “-“ in front of the variable to exclude it.

R

```
select(variants, -CHROM)
```

Output

```
# A tibble: 801 x 28
   sample_id    POS ID    REF   ALT    QUAL FILTER INDEL   IDV   IMF    DP
   <chr>      <dbl> <lgl> <chr> <chr> <dbl> <lgl>  <lgl> <dbl>  <dbl> <dbl>
 1 SRR25848… 9.97e3 NA    T     G        91 NA     FALSE    NA NA         4
 2 SRR25848… 2.63e5 NA    G     T        85 NA     FALSE    NA NA         6
 3 SRR25848… 2.82e5 NA    G     T       217 NA     FALSE    NA NA        10
 4 SRR25848… 4.33e5 NA    CTTT… CTTT…    64 NA     TRUE     12  1        12
 5 SRR25848… 4.74e5 NA    CCGC  CCGC…   228 NA     TRUE      9  0.9      10
 6 SRR25848… 6.49e5 NA    C     T       210 NA     FALSE    NA NA        10
 7 SRR25848… 1.33e6 NA    C     A       178 NA     FALSE    NA NA         8
 8 SRR25848… 1.73e6 NA    G     A       225 NA     FALSE    NA NA        11
 9 SRR25848… 2.10e6 NA    ACAG… ACAG…    56 NA     TRUE      2  0.667     3
10 SRR25848… 2.33e6 NA    AT    ATT     167 NA     TRUE      7  1         7
# … with 791 more rows, and 17 more variables: VDB <dbl>, RPB <dbl>,
#   MQB <dbl>, BQB <dbl>, MQSB <dbl>, SGB <dbl>, MQ0F <dbl>, ICB <lgl>,
#   HOB <lgl>, AC <dbl>, AN <dbl>, DP4 <chr>, MQ <dbl>, Indiv <chr>,
#   gt_PL <dbl>, gt_GT <dbl>, gt_GT_alleles <chr>
```

`dplyr` also provides useful functions to select columns based on their names. For instance, `ends_with()` allows you to select columns that ends with specific letters. For instance, if you wanted to select columns that end with the letter "B":

R

```
select(variants, ends_with("B"))
```

Output

```
# A tibble: 801 x 8
      VDB   RPB   MQB   BQB  MQSB    SGB ICB   HOB
    <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl> <lgl> <lgl>
 1 0.0257    NA    NA    NA NA     -0.556 NA    NA
 2 0.0961     1     1     1 NA     -0.591 NA    NA
 3 0.774     NA    NA    NA  0.975 -0.662 NA    NA
 4 0.478     NA    NA    NA  1     -0.676 NA    NA
 5 0.660     NA    NA    NA  0.916 -0.662 NA    NA
 6 0.268     NA    NA    NA  0.916 -0.670 NA    NA
 7 0.624     NA    NA    NA  0.901 -0.651 NA    NA
 8 0.992     NA    NA    NA  1.01  -0.670 NA    NA
 9 0.902     NA    NA    NA  1     -0.454 NA    NA
10 0.568     NA    NA    NA  1.01  -0.617 NA    NA
# … with 791 more rows
```

Challenge

Create a table that contains all the columns with the letter "i" except for the columns "Indiv" and "FILTER", and the column "POS". Hint: look at the help for the function `ends_with()` we've just covered.

Solution

R

```
select(variants, contains("i"), -Indiv, -FILTER, POS)
```

Output

```
# A tibble: 801 x 7
   sample_id  ID    INDEL   IDV    IMF ICB       POS
   <chr>      <lgl> <lgl> <dbl>  <dbl> <lgl>   <dbl>
 1 SRR2584863 NA    FALSE    NA NA     NA       9972
 2 SRR2584863 NA    FALSE    NA NA     NA     263235
 3 SRR2584863 NA    FALSE    NA NA     NA     281923
 4 SRR2584863 NA    TRUE     12  1     NA     433359
 5 SRR2584863 NA    TRUE      9  0.9   NA     473901
 6 SRR2584863 NA    FALSE    NA NA     NA     648692
 7 SRR2584863 NA    FALSE    NA NA     NA    1331794
 8 SRR2584863 NA    FALSE    NA NA     NA    1733343
 9 SRR2584863 NA    TRUE      2  0.667 NA    2103887
10 SRR2584863 NA    TRUE      7  1     NA    2333538
# … with 791 more rows
```

To choose rows, we can use `filter()`. For instance, to keep the rows for the sample `SRR2584863`:

R

```
filter(variants, sample_id == "SRR2584863")
```

Output

```
# A tibble: 25 x 29
   sample_id CHROM    POS ID    REF   ALT    QUAL FILTER INDEL   IDV    IMF
   <chr>     <chr>  <dbl> <lgl> <chr> <chr> <dbl> <lgl>  <lgl> <dbl>  <dbl>
 1 SRR25848… CP00… 9.97e3 NA    T     G        91 NA     FALSE    NA NA
 2 SRR25848… CP00… 2.63e5 NA    G     T        85 NA     FALSE    NA NA
 3 SRR25848… CP00… 2.82e5 NA    G     T       217 NA     FALSE    NA NA
 4 SRR25848… CP00… 4.33e5 NA    CTTT… CTTT…    64 NA     TRUE     12  1
 5 SRR25848… CP00… 4.74e5 NA    CCGC  CCGC…   228 NA     TRUE      9  0.9
 6 SRR25848… CP00… 6.49e5 NA    C     T       210 NA     FALSE    NA NA
 7 SRR25848… CP00… 1.33e6 NA    C     A       178 NA     FALSE    NA NA
 8 SRR25848… CP00… 1.73e6 NA    G     A       225 NA     FALSE    NA NA
 9 SRR25848… CP00… 2.10e6 NA    ACAG… ACAG…    56 NA     TRUE      2  0.667
10 SRR25848… CP00… 2.33e6 NA    AT    ATT     167 NA     TRUE      7  1
# … with 15 more rows, and 18 more variables: DP <dbl>, VDB <dbl>,
#   RPB <dbl>, MQB <dbl>, BQB <dbl>, MQSB <dbl>, SGB <dbl>, MQ0F <dbl>,
#   ICB <lgl>, HOB <lgl>, AC <dbl>, AN <dbl>, DP4 <chr>, MQ <dbl>,
#   Indiv <chr>, gt_PL <dbl>, gt_GT <dbl>, gt_GT_alleles <chr>
```

Note that this is equivalent to the base R code below, but is easier to read!

R

```
variants[variants$sample_id == "SRR2584863",]
```

`filter()` will keep all the rows that match the conditions that are provided. Here are a few examples:

R

```
## rows for which the reference genome has T or G
filter(variants, REF %in% c("T", "G"))
```

Output

```
# A tibble: 340 x 29
   sample_id CHROM    POS ID    REF   ALT    QUAL FILTER INDEL   IDV   IMF
   <chr>     <chr>  <dbl> <lgl> <chr> <chr> <dbl> <lgl>  <lgl> <dbl> <dbl>
 1 SRR25848… CP00… 9.97e3 NA    T     G      91   NA     FALSE    NA    NA
 2 SRR25848… CP00… 2.63e5 NA    G     T      85   NA     FALSE    NA    NA
 3 SRR25848… CP00… 2.82e5 NA    G     T     217   NA     FALSE    NA    NA
 4 SRR25848… CP00… 1.73e6 NA    G     A     225   NA     FALSE    NA    NA
 5 SRR25848… CP00… 2.62e6 NA    G     T      31.9 NA     FALSE    NA    NA
 6 SRR25848… CP00… 3.00e6 NA    G     A     225   NA     FALSE    NA    NA
 7 SRR25848… CP00… 3.91e6 NA    G     T     225   NA     FALSE    NA    NA
 8 SRR25848… CP00… 9.97e3 NA    T     G     214   NA     FALSE    NA    NA
 9 SRR25848… CP00… 1.06e4 NA    G     A     225   NA     FALSE    NA    NA
10 SRR25848… CP00… 6.40e4 NA    G     A     225   NA     FALSE    NA    NA
# … with 330 more rows, and 18 more variables: DP <dbl>, VDB <dbl>,
#   RPB <dbl>, MQB <dbl>, BQB <dbl>, MQSB <dbl>, SGB <dbl>, MQ0F <dbl>,
#   ICB <lgl>, HOB <lgl>, AC <dbl>, AN <dbl>, DP4 <chr>, MQ <dbl>,
#   Indiv <chr>, gt_PL <dbl>, gt_GT <dbl>, gt_GT_alleles <chr>
```

R

```
## rows with QUAL values greater than or equal to 100
filter(variants, QUAL >= 100)
```

Output

```
# A tibble: 666 x 29
   sample_id CHROM    POS ID    REF   ALT    QUAL FILTER INDEL   IDV   IMF
   <chr>     <chr>  <dbl> <lgl> <chr> <chr> <dbl> <lgl>  <lgl> <dbl> <dbl>
 1 SRR25848… CP00… 2.82e5 NA    G     T       217 NA     FALSE    NA  NA
 2 SRR25848… CP00… 4.74e5 NA    CCGC  CCGC…   228 NA     TRUE      9   0.9
 3 SRR25848… CP00… 6.49e5 NA    C     T       210 NA     FALSE    NA  NA
 4 SRR25848… CP00… 1.33e6 NA    C     A       178 NA     FALSE    NA  NA
 5 SRR25848… CP00… 1.73e6 NA    G     A       225 NA     FALSE    NA  NA
 6 SRR25848… CP00… 2.33e6 NA    AT    ATT     167 NA     TRUE      7   1
 7 SRR25848… CP00… 2.41e6 NA    A     C       104 NA     FALSE    NA  NA
 8 SRR25848… CP00… 2.45e6 NA    A     C       225 NA     FALSE    NA  NA
 9 SRR25848… CP00… 2.67e6 NA    A     T       225 NA     FALSE    NA  NA
10 SRR25848… CP00… 3.00e6 NA    G     A       225 NA     FALSE    NA  NA
# … with 656 more rows, and 18 more variables: DP <dbl>, VDB <dbl>,
#   RPB <dbl>, MQB <dbl>, BQB <dbl>, MQSB <dbl>, SGB <dbl>, MQ0F <dbl>,
#   ICB <lgl>, HOB <lgl>, AC <dbl>, AN <dbl>, DP4 <chr>, MQ <dbl>,
#   Indiv <chr>, gt_PL <dbl>, gt_GT <dbl>, gt_GT_alleles <chr>
```

R

```
## rows that have TRUE in the column INDEL
filter(variants, INDEL)
```

Output

```
# A tibble: 101 x 29
   sample_id CHROM    POS ID    REF   ALT    QUAL FILTER INDEL   IDV   IMF
   <chr>     <chr>  <dbl> <lgl> <chr> <chr> <dbl> <lgl>  <lgl> <dbl> <dbl>
 1 SRR25848… CP00… 4.33e5 NA    CTTT… CTTT…  64   NA     TRUE     12 1
 2 SRR25848… CP00… 4.74e5 NA    CCGC  CCGC… 228   NA     TRUE      9 0.9
 3 SRR25848… CP00… 2.10e6 NA    ACAG… ACAG…  56   NA     TRUE      2 0.667
 4 SRR25848… CP00… 2.33e6 NA    AT    ATT   167   NA     TRUE      7 1
 5 SRR25848… CP00… 3.90e6 NA    A     AC     43.4 NA     TRUE      2 1
 6 SRR25848… CP00… 4.43e6 NA    TGG   T     228   NA     TRUE     10 1
 7 SRR25848… CP00… 1.48e5 NA    AGGGG AGGG… 122   NA     TRUE      8 1
 8 SRR25848… CP00… 1.58e5 NA    GTTT… GTTT…  19.5 NA     TRUE      6 1
 9 SRR25848… CP00… 1.73e5 NA    CAA   CA    180   NA     TRUE     11 1
10 SRR25848… CP00… 1.75e5 NA    GAA   GA    194   NA     TRUE     10 1
# … with 91 more rows, and 18 more variables: DP <dbl>, VDB <dbl>,
#   RPB <dbl>, MQB <dbl>, BQB <dbl>, MQSB <dbl>, SGB <dbl>, MQ0F <dbl>,
#   ICB <lgl>, HOB <lgl>, AC <dbl>, AN <dbl>, DP4 <chr>, MQ <dbl>,
#   Indiv <chr>, gt_PL <dbl>, gt_GT <dbl>, gt_GT_alleles <chr>
```

R

```
## rows that don't have missing data in the IDV column
filter(variants, !is.na(IDV))
```

Output

```
# A tibble: 101 x 29
   sample_id CHROM    POS ID    REF   ALT    QUAL FILTER INDEL   IDV   IMF
   <chr>     <chr>  <dbl> <lgl> <chr> <chr> <dbl> <lgl>  <lgl> <dbl> <dbl>
 1 SRR25848… CP00… 4.33e5 NA    CTTT… CTTT…  64   NA     TRUE     12 1
 2 SRR25848… CP00… 4.74e5 NA    CCGC  CCGC… 228   NA     TRUE      9 0.9
 3 SRR25848… CP00… 2.10e6 NA    ACAG… ACAG…  56   NA     TRUE      2 0.667
 4 SRR25848… CP00… 2.33e6 NA    AT    ATT   167   NA     TRUE      7 1
 5 SRR25848… CP00… 3.90e6 NA    A     AC     43.4 NA     TRUE      2 1
 6 SRR25848… CP00… 4.43e6 NA    TGG   T     228   NA     TRUE     10 1
 7 SRR25848… CP00… 1.48e5 NA    AGGGG AGGG… 122   NA     TRUE      8 1
 8 SRR25848… CP00… 1.58e5 NA    GTTT… GTTT…  19.5 NA     TRUE      6 1
 9 SRR25848… CP00… 1.73e5 NA    CAA   CA    180   NA     TRUE     11 1
10 SRR25848… CP00… 1.75e5 NA    GAA   GA    194   NA     TRUE     10 1
# … with 91 more rows, and 18 more variables: DP <dbl>, VDB <dbl>,
#   RPB <dbl>, MQB <dbl>, BQB <dbl>, MQSB <dbl>, SGB <dbl>, MQ0F <dbl>,
#   ICB <lgl>, HOB <lgl>, AC <dbl>, AN <dbl>, DP4 <chr>, MQ <dbl>,
#   Indiv <chr>, gt_PL <dbl>, gt_GT <dbl>, gt_GT_alleles <chr>
```

`filter()` allows you to combine multiple conditions. You can separate them using a `,` as arguments to the function, they will be combined using the `&` (AND) logical operator. If you need to use the `|` (OR) logical operator, you can specify it explicitly:

R

```
## this is equivalent to:
##   filter(variants, sample_id == "SRR2584863" & QUAL >= 100)
filter(variants, sample_id == "SRR2584863", QUAL >= 100)
```

Output

```
# A tibble: 19 x 29
   sample_id CHROM    POS ID    REF   ALT    QUAL FILTER INDEL   IDV   IMF
   <chr>     <chr>  <dbl> <lgl> <chr> <chr> <dbl> <lgl>  <lgl> <dbl> <dbl>
 1 SRR25848… CP00… 2.82e5 NA    G     T       217 NA     FALSE    NA  NA  
 2 SRR25848… CP00… 4.74e5 NA    CCGC  CCGC…   228 NA     TRUE      9   0.9
 3 SRR25848… CP00… 6.49e5 NA    C     T       210 NA     FALSE    NA  NA  
 4 SRR25848… CP00… 1.33e6 NA    C     A       178 NA     FALSE    NA  NA  
 5 SRR25848… CP00… 1.73e6 NA    G     A       225 NA     FALSE    NA  NA  
 6 SRR25848… CP00… 2.33e6 NA    AT    ATT     167 NA     TRUE      7   1  
 7 SRR25848… CP00… 2.41e6 NA    A     C       104 NA     FALSE    NA  NA  
 8 SRR25848… CP00… 2.45e6 NA    A     C       225 NA     FALSE    NA  NA  
 9 SRR25848… CP00… 2.67e6 NA    A     T       225 NA     FALSE    NA  NA  
10 SRR25848… CP00… 3.00e6 NA    G     A       225 NA     FALSE    NA  NA  
11 SRR25848… CP00… 3.34e6 NA    A     C       211 NA     FALSE    NA  NA  
12 SRR25848… CP00… 3.40e6 NA    C     A       225 NA     FALSE    NA  NA  
13 SRR25848… CP00… 3.48e6 NA    A     G       200 NA     FALSE    NA  NA  
14 SRR25848… CP00… 3.49e6 NA    A     C       225 NA     FALSE    NA  NA  
15 SRR25848… CP00… 3.91e6 NA    G     T       225 NA     FALSE    NA  NA  
16 SRR25848… CP00… 4.10e6 NA    A     G       225 NA     FALSE    NA  NA  
17 SRR25848… CP00… 4.20e6 NA    A     C       225 NA     FALSE    NA  NA  
18 SRR25848… CP00… 4.43e6 NA    TGG   T       228 NA     TRUE     10   1  
19 SRR25848… CP00… 4.62e6 NA    A     C       185 NA     FALSE    NA  NA  
# … with 18 more variables: DP <dbl>, VDB <dbl>, RPB <dbl>, MQB <dbl>,
#   BQB <dbl>, MQSB <dbl>, SGB <dbl>, MQ0F <dbl>, ICB <lgl>, HOB <lgl>,
#   AC <dbl>, AN <dbl>, DP4 <chr>, MQ <dbl>, Indiv <chr>, gt_PL <dbl>,
#   gt_GT <dbl>, gt_GT_alleles <chr>
```

R

```r
## using `|` logical operator
filter(variants, sample_id == "SRR2584863", (INDEL | QUAL >= 100))
```

Output

```
# A tibble: 22 x 29
   sample_id CHROM    POS ID    REF   ALT    QUAL FILTER INDEL   IDV    IMF
   <chr>     <chr>  <dbl> <lgl> <chr> <chr> <dbl> <lgl>  <lgl> <dbl>  <dbl>
 1 SRR25848… CP00… 2.82e5 NA    G     T       217 NA     FALSE    NA NA    
 2 SRR25848… CP00… 4.33e5 NA    CTTT… CTTT…    64 NA     TRUE     12  1    
 3 SRR25848… CP00… 4.74e5 NA    CCGC  CCGC…   228 NA     TRUE      9  0.9  
 4 SRR25848… CP00… 6.49e5 NA    C     T       210 NA     FALSE    NA NA    
 5 SRR25848… CP00… 1.33e6 NA    C     A       178 NA     FALSE    NA NA    
 6 SRR25848… CP00… 1.73e6 NA    G     A       225 NA     FALSE    NA NA    
 7 SRR25848… CP00… 2.10e6 NA    ACAG… ACAG…    56 NA     TRUE      2  0.667
 8 SRR25848… CP00… 2.33e6 NA    AT    ATT     167 NA     TRUE      7  1    
 9 SRR25848… CP00… 2.41e6 NA    A     C       104 NA     FALSE    NA NA    
10 SRR25848… CP00… 2.45e6 NA    A     C       225 NA     FALSE    NA NA    
# … with 12 more rows, and 18 more variables: DP <dbl>, VDB <dbl>,
#   RPB <dbl>, MQB <dbl>, BQB <dbl>, MQSB <dbl>, SGB <dbl>, MQ0F <dbl>,
#   ICB <lgl>, HOB <lgl>, AC <dbl>, AN <dbl>, DP4 <chr>, MQ <dbl>,
#   Indiv <chr>, gt_PL <dbl>, gt_GT <dbl>, gt_GT_alleles <chr>
```

Challenge

Select all the mutations that occurred between the positions 1e6 (one million) and 2e6 (included) that are not indels and have QUAL greater than 200.

Solution

R

```
filter(variants, POS >= 1e6 & POS <= 2e6, !INDEL, QUAL > 200)
```

Output

```
# A tibble: 77 x 29
   sample_id CHROM     POS ID    REF   ALT    QUAL FILTER INDEL   IDV   IMF
   <chr>     <chr>   <dbl> <lgl> <chr> <chr> <dbl> <lgl>  <lgl> <dbl> <dbl>
 1 SRR25848… CP00… 1.73e6 NA    G     A       225 NA     FALSE    NA    NA
 2 SRR25848… CP00… 1.00e6 NA    A     G       225 NA     FALSE    NA    NA
 3 SRR25848… CP00… 1.02e6 NA    A     G       225 NA     FALSE    NA    NA
 4 SRR25848… CP00… 1.06e6 NA    C     T       225 NA     FALSE    NA    NA
 5 SRR25848… CP00… 1.06e6 NA    A     G       206 NA     FALSE    NA    NA
 6 SRR25848… CP00… 1.07e6 NA    G     T       225 NA     FALSE    NA    NA
 7 SRR25848… CP00… 1.07e6 NA    T     C       225 NA     FALSE    NA    NA
 8 SRR25848… CP00… 1.10e6 NA    C     T       225 NA     FALSE    NA    NA
 9 SRR25848… CP00… 1.11e6 NA    C     T       212 NA     FALSE    NA    NA
10 SRR25848… CP00… 1.11e6 NA    A     G       225 NA     FALSE    NA    NA
# … with 67 more rows, and 18 more variables: DP <dbl>, VDB <dbl>,
#   RPB <dbl>, MQB <dbl>, BQB <dbl>, MQSB <dbl>, SGB <dbl>, MQ0F <dbl>,
#   ICB <lgl>, HOB <lgl>, AC <dbl>, AN <dbl>, DP4 <chr>, MQ <dbl>,
#   Indiv <chr>, gt_PL <dbl>, gt_GT <dbl>, gt_GT_alleles <chr>
```

Pipes

But what if you wanted to select and filter? We can do this with pipes. Pipes, are a fairly recent addition to R. Pipes let you take the output of one function and send it directly to the next, which is useful when you need to many things to the same data set. It was possible to do this before pipes were added to R, but it was much messier and more difficult. Pipes in R look like `%>%` and are made available via the `magrittr` package, which is installed as part of `dplyr`. If you use RStudio, you can type the pipe with Ctrl + Shift + M if you're using a PC, or Cmd + Shift + M if you're using a Mac.

R

```
variants %>%
  filter(sample_id == "SRR2584863") %>%
  select(REF, ALT, DP)
```

Output

```
# A tibble: 25 x 3
   REF                       ALT                                      DP
   <chr>                     <chr>                                 <dbl>
 1 T                         G                                         4
 2 G                         T                                         6
 3 G                         T                                        10
 4 CTTTTTTT                  CTTTTTTTT                                12
 5 CCGC                      CCGCGC                                   10
 6 C                         T                                        10
 7 C                         A                                         8
 8 G                         A                                        11
 9 ACAGCCAGCCAGCCAGCCAGCCAG… ACAGCCAGCCAGCCAGCCAGCCAGCCAGCCAGCCAGCCA…     3
10 AT                        ATT                                       7
# … with 15 more rows
```

In the above code, we use the pipe to send the `variants` dataset first through `filter()`, to keep rows where `sample_id` matches a particular sample, and then through `select()` to keep only the `REF`, `ALT`, and `DP` columns. Since `%>%` takes the object on its left and passes it as the first argument to the function on its right, we don't need to explicitly include the data frame as an argument to the `filter()` and `select()` functions any more. We then pipe the results to the `head()` function so that we only see the first six rows of data.

Some may find it helpful to read the pipe like the word "then". For instance, in the above example, we took the data frame `variants`, *then* we `filter`ed for rows where `sample_id` was SRR2584863, *then* we `select`ed the `REF`, `ALT`, and `DP` columns, *then* we showed only the first six rows. The **`dplyr`** functions by themselves are somewhat simple, but by combining them into linear workflows with the pipe, we can accomplish more complex manipulations of data frames.

If we want to create a new object with this smaller version of the data we can do so by assigning it a new name:

R

```
SRR2584863_variants <- variants %>%
  filter(sample_id == "SRR2584863") %>%
  select(REF, ALT, DP)
```

This new object includes all of the data from this sample. Let's look at it to confirm it's what we want:

R

```
SRR2584863_variants
```

Output

```
# A tibble: 25 x 3
   REF                       ALT                                      DP
   <chr>                     <chr>                                 <dbl>
 1 T                         G                                         4
 2 G                         T                                         6
 3 G                         T                                        10
 4 CTTTTTTT                  CTTTTTTTT                                12
 5 CCGC                      CCGCGC                                   10
 6 C                         T                                        10
 7 C                         A                                         8
 8 G                         A                                        11
 9 ACAGCCAGCCAGCCAGCCAGCCAG… ACAGCCAGCCAGCCAGCCAGCCAGCCAGCCAGCCAGCCA…     3
10 AT                        ATT                                       7
# … with 15 more rows
```

Exercise: Pipe and filter

Starting with the `variants` dataframe, use pipes to subset the data to include only observations from SRR2584863 sample, where the filtered depth (DP) is at least 10. Retain only the columns `REF`, `ALT`, and `POS`.

Solution

R

```
variants %>%
filter(sample_id == "SRR2584863" & DP >= 10) %>%
select(REF, ALT, POS)
```

Output

```
# A tibble: 16 x 3
   REF      ALT           POS
   <chr>    <chr>       <dbl>
 1 G        T          281923
 2 CTTTTTTT CTTTTTTTT  433359
 3 CCGC     CCGCGC     473901
 4 C        T          648692
 5 G        A         1733343
 6 A        C         2446984
 7 G        T         2618472
 8 A        T         2665639
 9 G        A         2999330
10 A        C         3339313
11 C        A         3401754
12 A        C         3488669
13 G        T         3909807
14 A        G         4100183
15 A        C         4201958
16 TGG      T         4431393
```

Mutate

Frequently you'll want to create new columns based on the values in existing columns, for example to do unit conversions or find the ratio of values in two columns. For this we'll use the `dplyr` function `mutate()`.

We have a column titled "QUAL". This is a Phred-scaled confidence score that a polymorphism exists at this position given the sequencing data. Lower QUAL scores indicate low probability of a polymorphism existing at that site. We can convert the confidence value QUAL to a probability value according to the formula:

Probability = 1- 10 ^ -(QUAL/10)

Let's add a column (`POLPROB`) to our `variants` dataframe that shows the probability of a polymorphism at that site given the data. We'll show only the first six rows of data.

R

```
variants %>%
  mutate(POLPROB = 1 - (10 ^ -(QUAL/10)))
```

Output

```
# A tibble: 801 x 30
   sample_id CHROM    POS ID    REF   ALT    QUAL FILTER INDEL   IDV    IMF
   <chr>     <chr>  <dbl> <lgl> <chr> <chr> <dbl> <lgl>  <lgl> <dbl>  <dbl>
 1 SRR25848… CP00… 9.97e3 NA    T     G        91 NA     FALSE    NA NA
 2 SRR25848… CP00… 2.63e5 NA    G     T        85 NA     FALSE    NA NA
 3 SRR25848… CP00… 2.82e5 NA    G     T       217 NA     FALSE    NA NA
 4 SRR25848… CP00… 4.33e5 NA    CTTT… CTTT…    64 NA     TRUE     12  1
 5 SRR25848… CP00… 4.74e5 NA    CCGC  CCGC…   228 NA     TRUE      9  0.9
 6 SRR25848… CP00… 6.49e5 NA    C     T       210 NA     FALSE    NA NA
 7 SRR25848… CP00… 1.33e6 NA    C     A       178 NA     FALSE    NA NA
 8 SRR25848… CP00… 1.73e6 NA    G     A       225 NA     FALSE    NA NA
 9 SRR25848… CP00… 2.10e6 NA    ACAG… ACAG…    56 NA     TRUE      2  0.667
10 SRR25848… CP00… 2.33e6 NA    AT    ATT     167 NA     TRUE      7  1
# … with 791 more rows, and 19 more variables: DP <dbl>, VDB <dbl>,
#   RPB <dbl>, MQB <dbl>, BQB <dbl>, MQSB <dbl>, SGB <dbl>, MQ0F <dbl>,
#   ICB <lgl>, HOB <lgl>, AC <dbl>, AN <dbl>, DP4 <chr>, MQ <dbl>,
#   Indiv <chr>, gt_PL <dbl>, gt_GT <dbl>, gt_GT_alleles <chr>,
#   POLPROB <dbl>
```

Exercise

There are a lot of columns in our dataset, so let's just look at the `sample_id`, `POS`, `QUAL`, and `POLPROB` columns for now. Add a line to the above code to only show those columns.

Solution

R

```
variants %>%
 mutate(POLPROB = 1 - 10 ^ -(QUAL/10)) %>%
 select(sample_id, POS, QUAL, POLPROB)
```

Output

```
# A tibble: 801 x 4
   sample_id      POS  QUAL POLPROB
   <chr>        <dbl> <dbl>   <dbl>
 1 SRR2584863    9972    91   1.000
 2 SRR2584863  263235    85   1.000
 3 SRR2584863  281923   217   1
 4 SRR2584863  433359    64   1.000
 5 SRR2584863  473901   228   1
 6 SRR2584863  648692   210   1
 7 SRR2584863 1331794   178   1
 8 SRR2584863 1733343   225   1
 9 SRR2584863 2103887    56   1.000
10 SRR2584863 2333538   167   1
# … with 791 more rows
```

We are interested in knowing the most common size for the indels. Let's create a new column, called "indel_size" that contains the size difference between the our sequences and the reference genome. The function, `nchar()` returns the number of letters in a string.

R

```
variants %>%
  mutate(indel_size = nchar(ALT) - nchar(REF))
```

Output

```
# A tibble: 801 x 30
   sample_id CHROM    POS ID    REF   ALT    QUAL FILTER INDEL   IDV    IMF
   <chr>     <chr>  <dbl> <lgl> <chr> <chr> <dbl> <lgl>  <lgl> <dbl>  <dbl>
 1 SRR25848… CP00… 9.97e3 NA    T     G        91 NA     FALSE    NA NA
 2 SRR25848… CP00… 2.63e5 NA    G     T        85 NA     FALSE    NA NA
 3 SRR25848… CP00… 2.82e5 NA    G     T       217 NA     FALSE    NA NA
 4 SRR25848… CP00… 4.33e5 NA    CTTT… CTTT…    64 NA     TRUE     12  1
 5 SRR25848… CP00… 4.74e5 NA    CCGC  CCGC…   228 NA     TRUE      9  0.9
 6 SRR25848… CP00… 6.49e5 NA    C     T       210 NA     FALSE    NA NA
 7 SRR25848… CP00… 1.33e6 NA    C     A       178 NA     FALSE    NA NA
 8 SRR25848… CP00… 1.73e6 NA    G     A       225 NA     FALSE    NA NA
 9 SRR25848… CP00… 2.10e6 NA    ACAG… ACAG…    56 NA     TRUE      2  0.667
10 SRR25848… CP00… 2.33e6 NA    AT    ATT     167 NA     TRUE      7  1
# … with 791 more rows, and 19 more variables: DP <dbl>, VDB <dbl>,
#   RPB <dbl>, MQB <dbl>, BQB <dbl>, MQSB <dbl>, SGB <dbl>, MQ0F <dbl>,
#   ICB <lgl>, HOB <lgl>, AC <dbl>, AN <dbl>, DP4 <chr>, MQ <dbl>,
#   Indiv <chr>, gt_PL <dbl>, gt_GT <dbl>, gt_GT_alleles <chr>,
#   indel_size <int>
```

When you want to create a new variable that depends on multiple conditions, the function `case_when()` in combination with `mutate()` is very useful. Our current dataset has a column that tells use whether each mutation is an indel, but we don't know if it's an insertion or a deletion. Let's create a new variable, called `mutation_type` that will take the values: `insertion`, `deletion`, or `point` depending on the value found in the `indel_size` column. We will save this data frame in a new variable, called `variants_indel`.

R

```r
variants_indel <- variants %>%
  mutate(
    indel_size = nchar(ALT) - nchar(REF),
    mutation_type = case_when(
      indel_size > 0 ~ "insertion",
      indel_size < 0 ~ "deletion",
      indel_size == 0 ~ "point"
    ))
```

When `case_when()` is used within `mutate()`, each row is evaluated for the condition listed in the order listed. The first condition that returns `TRUE` will by used to fill the content of the new column (here `mutation_type`) with the value listed on the right side of the `~` is used. If none of the conditions are met, the function returns `NA` (missing data).

We can check that we captured all possibilities by looking for missing data in the new `mutation_type` column, and confirm that no row matches this condition:

R

```r
variants_indel %>%
  filter(is.na(mutation_type))
```

Output

```
# A tibble: 0 x 31
# … with 31 variables: sample_id <chr>, CHROM <chr>, POS <dbl>, ID <lgl>,
#   REF <chr>, ALT <chr>, QUAL <dbl>, FILTER <lgl>, INDEL <lgl>,
#   IDV <dbl>, IMF <dbl>, DP <dbl>, VDB <dbl>, RPB <dbl>, MQB <dbl>,
#   BQB <dbl>, MQSB <dbl>, SGB <dbl>, MQ0F <dbl>, ICB <lgl>, HOB <lgl>,
#   AC <dbl>, AN <dbl>, DP4 <chr>, MQ <dbl>, Indiv <chr>, gt_PL <dbl>,
#   gt_GT <dbl>, gt_GT_alleles <chr>, indel_size <int>,
#   mutation_type <chr>
```

Split-apply-combine data analysis and the summarize() function

Many data analysis tasks can be approached using the "split-apply-combine" paradigm: split the data into groups, apply some analysis to each group, and then combine the results. `dplyr` makes this very easy through the use of the `group_by()` function, which splits the data into groups. When the data is grouped in this way `summarize()` can be used to collapse each group into a single-row summary. `summarize()` does this by applying an aggregating or summary function to each group.

For example, if we wanted to group by `mutation_type` and find the average size for the insertions and deletions:

R

```
variants_indel  %>%
  group_by(mutation_type) %>%
  summarize(
    mean_size = mean(indel_size)
  )
```

Output

```
# A tibble: 3 x 2
  mutation_type mean_size
  <chr>             <dbl>
1 deletion          -1.38
2 insertion          1.61
3 point              0
```

We can have additional columns by adding arguments to the `summarize()` function. For instance, if we also wanted to know the median indel size:

R

```
variants_indel %>%
  group_by(mutation_type) %>%
  summarize(
    mean_size = mean(indel_size),
    median_size = median(indel_size)
  )
```

Output

```
# A tibble: 3 x 3
  mutation_type mean_size median_size
  <chr>             <dbl>       <dbl>
1 deletion          -1.38          -1
2 insertion          1.61           1
3 point              0              0
```

So to view the highest filtered depth (`DP`) for each sample:

R

```
variants_indel %>%
  group_by(sample_id) %>%
  summarize(max(DP))
```

Output

```
# A tibble: 3 x 2
  sample_id  `max(DP)`
  <chr>          <dbl>
1 SRR2584863        20
2 SRR2584866        79
3 SRR2589044        16
```

Challenge

What are the largest insertions and deletions? Hint: the function `abs()` returns the absolute value.

Solution

R

```
variants_indel %>%
  group_by(mutation_type) %>%
  summarize(
    max_size = max(abs(indel_size))
  )
```

Output

```
# A tibble: 3 x 2
  mutation_type max_size
  <chr>            <int>
1 deletion             6
2 insertion           24
3 point                0
```

Callout: missing data and built-in functions

R has many built-in functions like `mean()`, `median()`, `min()`, and `max()` that are useful to compute summary statistics. These are called "built-in functions" because they come with R and don't require that you install any additional packages. By default, all **R functions operating on vectors that contains missing data will return NA**. It's a way to make sure that users know they have missing data, and make a conscious decision on how to deal with it. When dealing with simple statistics like the mean, the easiest way to ignore `NA` (the missing data) is to use `na.rm = TRUE` (`rm` stands for remove).

It is often useful to calculate how many observations are present in each group. The function `n()` helps you do that:

R

```
variants_indel %>%
  group_by(mutation_type) %>%
  summarize(
    n = n()
  )
```

Output

```
# A tibble: 3 x 2
  mutation_type     n
  <chr>         <int>
1 deletion         39
2 insertion        62
3 point           700
```

Because it's a common operation, the `dplyr` verb, `count()` is a "shortcut" that combines these 2 commands:

R

```
variants_indel %>%
  count(mutation_type)
```

Output

```
# A tibble: 3 x 2
  mutation_type     n
  <chr>         <int>
1 deletion         39
2 insertion        62
3 point           700
```

`group_by()` (and therfore `count()`) can also take multiple column names.

Challenge

- How many mutations are found in each sample?

Solution

R

```
variants_indel %>%
  count(sample_id)
```

Output

```
# A tibble: 3 x 2
  sample_id      n
  <chr>      <int>
1 SRR2584863    25
2 SRR2584866   766
3 SRR2589044    10
```

- How many mutation types are found in sample?

Solution

R

```
variants_indel %>%
  count(sample_id, mutation_type)
```

Output

```
# A tibble: 9 x 3
  sample_id  mutation_type     n
  <chr>      <chr>         <int>
1 SRR2584863 deletion          1
2 SRR2584863 insertion         5
3 SRR2584863 point            19
4 SRR2584866 deletion         37
5 SRR2584866 insertion        54
6 SRR2584866 point           675
7 SRR2589044 deletion          1
8 SRR2589044 insertion         3
9 SRR2589044 point             6
```

Reshaping data frames

While the tidy format is useful to analyze and plot data in R, it can sometimes be useful to transform the "long" tidy format, into the wide format. This transformation can be done with the `spread()` function provided by the `tidyr` package (also part of the `tidyverse`).

`spread()` takes a data frame as the first argument, and two arguments: the column name that will become the columns and the column name that will become the cells in the wide data.

R

```
variants_wide <- variants_indel %>%
  count(sample_id, mutation_type) %>%
  spread(mutation_type, n)
variants_wide
```

Output

```
# A tibble: 3 x 4
  sample_id  deletion insertion point
  <chr>         <int>     <int> <int>
1 SRR2584863        1         5    19
2 SRR2584866       37        54   675
3 SRR2589044        1         3     6
```

The opposite operation of `spread()` is taken care by `gather()` . We specify the names of the new columns, and here add `-sample_id` as this column shouldn't be affected by the reshaping:

R

```
variants_wide %>%
  gather(mutation_type, n, -sample_id)
```

Output

```
# A tibble: 9 x 3
  sample_id  mutation_type     n
  <chr>      <chr>         <int>
1 SRR2584863 deletion          1
2 SRR2584866 deletion         37
3 SRR2589044 deletion          1
4 SRR2584863 insertion         5
5 SRR2584866 insertion        54
6 SRR2589044 insertion         3
7 SRR2584863 point            19
8 SRR2584866 point           675
9 SRR2589044 point             6
```

Challenge (optional)

Based on the probability scores we calculated when we first introducted `mutate()`, classify each mutation in 3 categories: high (> 0.95), medium (between 0.7 and 0.95), and low (< 0.7), and create a table with `sample_id` as rows, the 3 levels of quality as columns, and the number of mutations in the cells.

Solution

R

```
variants %>%
  mutate(POLPROB = 1 - (10 ^ -(QUAL/10))) %>%
  mutate(prob_cat = case_when(
    POLPROB >=  .95 ~ "high",
    POLPROB >=  .7 & POLPROB < .95 ~ "medium",
    POLPROB < .7 ~ "low"
  )) %>%
  count(sample_id, prob_cat) %>%
  spread(prob_cat, n)
```

Output

```
# A tibble: 3 x 4
  sample_id   high   low medium
  <chr>      <int> <int>  <int>
1 SRR2584863    25    NA     NA
2 SRR2584866   754     4      8
3 SRR2589044    10    NA     NA
```

Exporting

We can export this new dataset using `write_csv()`:

R

```
write_csv(variants_indel, "variants_indel.csv")
```

Resources

- Handy dplyr cheatsheet (https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf)

Much of this lesson was copied or adapted from Jeff Hollister's materials (http://usepa.github.io/introR/2015/01/14/03-Clean/)

Key Points

- Use the `dplyr` package to manipulate dataframes.
- Use `select()` to choose variables from a dataframe.
- Use `filter()` to choose data based on values.
- Use `group_by()` and `summarize()` to work with subsets of data.
- Use `mutate()` to create new variables.

‹ (../03-basics-factors-dataframes/index.html)

› (../05-data-visuali