Introduction to dplyr

36-290 – Statistical Research Methodology

Week 2 Tuesday – Fall 2021

A First Dataset

Below we will read in data that are stored on disk in an ASCII (i.e., human-readable) file in csv (or "comma-separated values") format. csv is a common format, so even though we do not cover file I/O here, you may be able to utilize this example to start exploring your own data.

```
df = read.csv("http://www.stat.cmu.edu/~pfreeman/GalaxyMass.csv")
```

The variable `df` stands for "data frame." (To be clear, you can use whatever name you want for the variable that points to the data frame.) A data frame is a table of data. Here, those data consist of 3456 rows and 10 columns:

```
dim(df)
```

```
## [1] 3456   10
```

A First Dataset

In a data frame, each row corresponds to a single object under study. Here, those objects are galaxies. Each column, by contrast, represents a different measurement associated with the object under study. Here, there are ten. To see their names, use `names()`:

```
names(df)
```

```
##  [1] "field"  "Gini"   "M20"    "C"      "A"      "size"   "n"      "q"      "z.mode" "mass"
```

The first column represents a sector of the sky in which the galaxy lies. This is a *factor* variable, because it takes on a few discrete values that don't necessarily map to any particular numerical order:

```
unique(df$field)
```

```
## [1] "COSMOS" "EGS"    "GOODSN" "GOODSS" "UDS"
```

(Note the use of the dollar sign!) The next seven variables represent different statistics that summarize the galaxy's appearance. The penultimate variable is an estimate of galaxy distance, and the last is an estimate of galaxy mass.

dplyr

`dplyr` is a package within the larger `tidyverse` that one uses to manipulate data frames. In this set of notes, we will demonstrate basic data frame manipulation using `dplyr` functions and "pipes" (from the `magrittr` package).

```
library(magrittr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following object is masked from 'package:MASS':
##
##     select

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

We *could* just load the whole `tidyverse` instead, if we wish.

dplyr

Think of data frames as nouns and `dplyr` functions as verbs, actions that you apply to the data frames. The following are the most basic `dplyr` verbs:

- `slice()`: choose particular rows based on integer indexing
- `filter()`: choose particular rows based on logical criteria
- `group_by()`: split the data frame into groups of rows based on factor variable values
- `select()`: choose particular columns
- `arrange()`: order rows by value of a column
- `mutate()`: create new columns

NOTE: calling `dplyr` verbs always outputs a new data frame; it *does not alter* the existing data frame.

So to keep the changes, we have to reassign the data frame to be the output of the pipe, as we will see below.

slice()

Use `slice()` when you want to indicate certain row numbers need to be kept:

```
df %>% slice(.,c(7,8,14:15))
```

```
##    field      Gini       M20        C          A      size      n      q z.mode     mass
## 1 COSMOS 0.3319734 -1.294080 2.975167 0.09361421 0.8088991 0.7442 0.2343   1.96 10.68240
## 2 COSMOS 0.3581774 -1.656436 3.730035 0.07200192 0.3814856 0.9118 0.5919   1.49  9.82788
## 3 COSMOS 0.4839426 -1.609840 3.005040 0.17674453 0.4727087 1.3563 0.4967   1.69 10.18260
## 4 COSMOS 0.3170045 -1.285205 2.929714 0.09791615 0.4604285 0.5090 0.3333   2.00  9.75075
```

The `%>%` is a *pipe*. A pipe takes the output of one `R` command and uses it as input to a following command. Here, we pipe the output of `df` (which is simply the whole data frame) to the `slice()` function. The period in the `slice()` function indicates where the data frame would go if we were to simply call `slice()` without any piped input. (In other words, we could replace the construction above with `slice(df,c(7,8,14:15))`.) The `slice()` function then outputs the 7th, 8th, 14th, and 15th rows.

slice()

Slicing can also be done "negatively":

```
df %>% slice(.,-c(1:2,19:23)) %>% nrow(.)
```

```
## [1] 3449
```

Here, we *keep* all the rows *except* the 1st, 2nd, and 19th through 23rd rows, then check to see how many rows we have left.

filter()

Use `filter()` when you want to choose rows based on logical conditions:

```
df %>% filter(.,field=="COSMOS") %>% head(.,2)
```

```
##    field      Gini       M20        C         A      size      n      q z.mode     mass
## 1 COSMOS 0.4132853 -1.132330 2.257530 0.1211518 0.7554244 0.3398 0.3579   2.04 10.55120
## 2 COSMOS 0.4177935 -1.603147 2.970555 0.1002479 0.6537786 0.6722 0.8124   1.79  9.87608
```

```
df %>% filter(.,mass>10) %>% nrow(.)
```

```
## [1] 1862
```

```
df %>% filter(.,(field=="GOODSS" & mass<10)) %>% nrow(.)
```

```
## [1] 276
```

Note the use of & in the third example. This combines conditions via the logical `and`. | is the analogous symbol for `or`.

group_by()

Use `group_by()` to split your data frame into groups of rows based on the values of a factor variable. Note that `group_by()` in and of itself is only useful if you pipe its output to something else…like `summarize()`:

```
df %>% group_by(.,field) %>% summarize(.,Mean=mean(mass),Number=n())
```

```
## # A tibble: 5 × 3
##   field   Mean Number
##   <chr>  <dbl>  <int>
## 1 COSMOS  10.2    905
## 2 EGS     10.0    750
## 3 GOODSN  10.1    464
## 4 GOODSS  10.1    588
## 5 UDS     10.2    749
```

Here, `Mean` is a column name used in the output data frame; what is shown in that column is the average value of the galaxy mass in each field. Similarly, `Number` shows the number of galaxies in the sample from each field.

select()

Use `select()` when you want to choose certain columns:

```
df %>% select(.,Gini,q,z.mode) %>% slice(.,c(1:4))
```

```
##        Gini      q z.mode
## 1 0.4132853 0.3579   2.04
## 2 0.4177935 0.8124   1.79
## 3 0.4212831 0.3585   2.02
## 4 0.4725081 0.4927   1.94
```

Note: here we chose columns *and* rows.

arrange()

Use `arrange()` to order rows by values of a column:

```
df %>% arrange(.,desc(mass)) %>% select(.,C,A,z.mode) %>% head(.,2)
```

```
##          C         A z.mode
## 1 3.594274 0.1711388   1.68
## 2 3.655988 0.2116704   1.78
```

If you just do `arrange(.,mass)` then the masses will be ordered in *ascending* order, by default.

mutate()

Use `mutate()` when you want to create one or several columns:

```
df %>% mutate(.,mass.linear=10^mass) %>% select(.,mass,mass.linear) %>% arrange(.,mass) %>% head(.,3)
```

```
##    mass mass.linear
## 1 8.285   192752491
## 2 8.540   346736850
## 3 8.550   354813389
```

Saving the New Data Frame

As stated above, the original data frame is not altered in piping operations. To save the result of a series of piping operations, just use = or `->`:

```
df.new = df %>% filter(.,field=="EGS") %>% select(.,z.mode,mass)
nrow(df.new)
```

```
## [1] 750
```

```
df %>% filter(.,field=="COSMOS") %>% select(.,Gini,M20,C,A) -> df.new
nrow(df.new)
```

```
## [1] 905
```