Using the dplyr package

Department of Biostatistics and Bioinformatics

Steve Pittard wsp@emory.edu

February 1, 2018

Data Frames

Why Use Data Frames ?

- A data frame is a special type of list that contains data in a format that allows for easier manipulation, reshaping, and open-ended analysis
- Data frames are tightly coupled collections of variables. It is one of the more important constructs you will encounter when using R so learn all you can about it
- A data frame is an analogue to the Excel spreadsheet but is much more flexible for storing, manipulating, and analyzing data
- Data frames can be constructed from existing vectors, lists, or matrices. Many times they are created by reading in comma delimited files, (CSV files), using the read.table command
- Once you become accustomed to working with data frames, R becomes so much easier to use

Why Use Data Frames ?

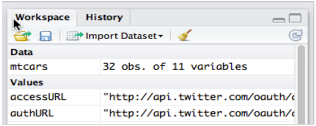
Use the **dataframe()** function to create a data frame. It looks like a matrix but allows for mixed data types

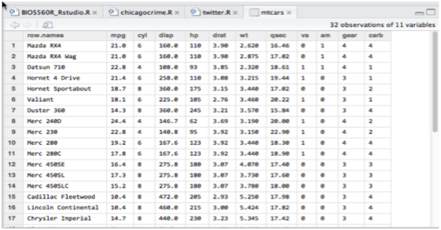
```
names  <- c("P1","P2","P3","P4","P5")
temp   <- c(98.2,101.3,97.2,100.2,98.5)
pulse  <- c(66,72,83,85,90)
gender <- c("M","F","M","M","F")

my_df <- data.frame(names,temp,pulse,gender) # Much more flexible

  names  temp pulse gender
1    P1  98.2    66      M
2    P2 101.3    72      F
3    P3  97.2    83      M
4    P4 100.2    85      M
5    P5  98.5    90      F

plot(my_df$pulse ~ my_df$temp,main="Pulse Rate",xlab="Patient",ylab="BPM")
mean(my_df[,2:3])
 temp pulse
99.08 79.20
```

Why Use Data Frames ?

Once you have a data frame you could edit it with the Workspace viewer in RStudio although this doesn't generalize. Imagine if your data set had 10,000 lines ?

```
data(mtcars)   # Load the builtin mtcars dataframe
```





	row.names	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
1	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
2	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
3	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
4	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
5	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
6	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
7	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
8	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
9	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
10	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
11	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
12	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
13	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
14	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
15	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
16	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
17	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4

Data Frames - Builtin

R comes with a variety of built-in data sets that are very useful for getting used to data sets and how to manipulate them.

```
library(help="datasets")

# Gives detailed descriptions on available data sets

AirPassengers           Monthly Airline Passenger Numbers 1949-1960
BJsales                 Sales Data with Leading Indicator
BOD                     Biochemical Oxygen Demand
CO2                     Carbon Dioxide Uptake in Grass Plants
ChickWeight             Weight versus age of chicks on different diets
DNase                   Elisa assay of DNase
EuStockMarkets          Daily Closing Prices of Major European Stock
                        Indices, 1991-1998
Formaldehyde            Determination of Formaldehyde
HairEyeColor            Hair and Eye Color of Statistics Students

help(mtcars)  # Get details on a given data set
```

Data Frames - Builtin

R comes with a variety of built-in data sets that are very useful for getting used to data sets and how to manipulate them.

```
data(mtcars)

str(mtcars)
'data.frame':  32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...

nrow(mtcars) # How many rows does it have ?
[1] 32

ncol(mtcars) # How many columns are there ?
[1] 11
```

Data Frames - Accessing

There are various ways to select, remove, or exclude rows and columns

```
mtcars[,-11]
                      mpg cyl disp  hp drat    wt  qsec vs am gear
Mazda RX4            21.0   6  160 110 3.90 2.620 16.46  0  1    4
Mazda RX4 Wag        21.0   6  160 110 3.90 2.875 17.02  0  1    4
Datsun 710           22.8   4  108  93 3.85 2.320 18.61  1  1    4

mtcars   # Notice that carb is included
                       mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4             21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag         21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710            22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
```

Data Frames - Accessing

There are various ways to select, remove, or exclude rows and columns

```
mtcars[,-3:-5]  # Print all columns except for columns 3 through 5
                    mpg cyl    wt  qsec vs am gear      carb
Mazda RX4          21.0   6 2.620 16.46  0  1    4 0.6020600
Mazda RX4 Wag      21.0   6 2.875 17.02  0  1    4 0.6020600
Datsun 710         22.8   4 2.320 18.61  1  1    4 0.0000000

mtcars[,c(-3,-5)] # Print all columns except for colums 3 AND 5
                    mpg cyl  hp    wt  qsec vs am gear      carb
Mazda RX4          21.0   6 110 2.620 16.46  0  1    4 0.6020600
Mazda RX4 Wag      21.0   6 110 2.875 17.02  0  1    4 0.6020600
Datsun 710         22.8   4  93 2.320 18.61  1  1    4 0.0000000
```

Data Frames - Accessing

There are various ways to select, remove, or exclude rows and columns

```
mtcars[mtcars$mpg >= 30.0,]
                mpg cyl disp  hp drat    wt  qsec vs am gear carb
Fiat 128       32.4   4 78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4 75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla 33.9   4 71.1  65 4.22 1.835 19.90  1  1    4    1
Lotus Europa   30.4   4 95.1 113 3.77 1.513 16.90  1  1    5    2
```

```
mtcars[mtcars$mpg >= 30.0,2:6]

                mpg cyl disp  hp drat
Fiat 128       32.4   4 78.7  66 4.08
Honda Civic    30.4   4 75.7  52 4.93
Toyota Corolla 33.9   4 71.1  65 4.22
Lotus Europa   30.4   4 95.1 113 3.77
```

```
mtcars[mtcars$mpg >= 30.0 & mtcars$cyl < 6,]
                mpg cyl disp  hp drat    wt  qsec vs am gear carb
Fiat 128       32.4   4 78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4 75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla 33.9   4 71.1  65 4.22 1.835 19.90  1  1    4    1
Lotus Europa   30.4   4 95.1 113 3.77 1.513 16.90  1  1    5    2
```

Data Frames - Interrogating

Find all rows that correspond to Automatic and Count them

```
mtcars[mtcars$am==0,]
                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Hornet 4 Drive     21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant            18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D          24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230           22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
..
..

nrow(mtcars[mtcars$am == 0,])
[1] 19

nrow(mtcars[mtcars$am == 1,])
[1] 13
```

Data Frames - Interrogating

Extract all rows whose MPG value exceeds the mean MPG for the entire data frame

```
mtcars[mtcars$mpg > mean(mtcars$mpg),]
```

```
mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

Data Frames - Interrogating

Extract all rows whose MPG value exceeds the mean MPG for the entire data frame

```
# Find the quartiles for the MPG vector

quantile(mtcars$mpg)
    0%    25%    50%    75%   100%
10.400 15.425 19.200 22.800 33.900

# Now find the cars for which the MPG exceeds the 75% value:

mtcars[mtcars$mpg > quantile(mtcars$mpg)[4],]
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
```

Data Frames - Interrogating

What columns appear to be factors ? Variables with only a “few” different unique values perhaps ?

```
str(mtcars)
'data.frame':  32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...

unique(mtcars$am)   # Tells us what the unique values are
[1] 1 0
```

Data Frames - Factors

See how many unqiue values each column takes on

```
sapply(mtcars, function(x) length(unique(x)))
 mpg  cyl disp   hp drat   wt qsec   vs   am gear carb
  25    3   27   22   22   29   30    2    2    3    6
```

If we summarize one of these potential factors right now, R will treat it as being purely numeric which we might not want

```
summary(mtcars$am)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  0.0000  0.4062  1.0000  1.0000
```

So this really isn't helpful since we know that the "am" values are transmission types

```
mtcars$am <- factor(mtcars$am, levels = c(0,1), labels = c("Auto","Man") )

summary(mtcars$am)
Auto Manu
  19   13
```

Data Frames - Factors

We can add columns to a data frame. Let's say we want to create a new column called "mpgrate" that, based on the output of the quantile command, will have a rating of the that car's MPG in terms of "horrible","bad","good","great".

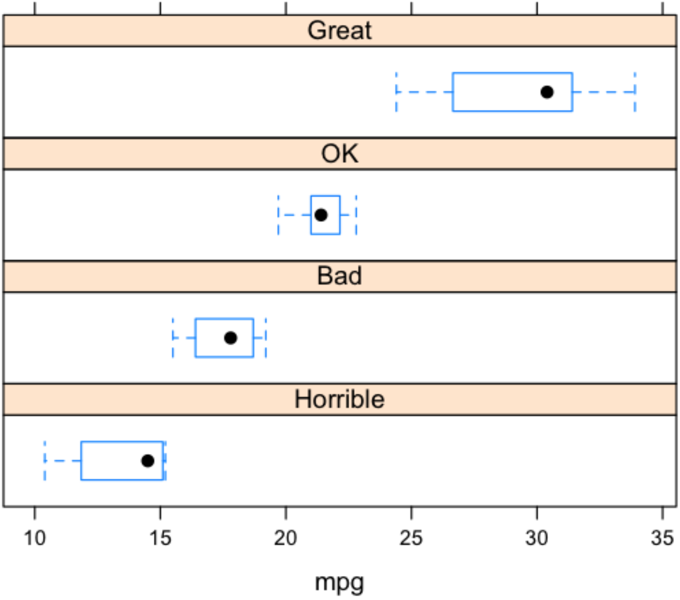
The labels could be more scientific but this is still a good use case. There are a couple of ways to do this:

```
data(mtcars)   # Reload a "pure" copy of mtcars

mpgrate <- cut(mtcars$mpg,
              breaks = quantile(mtcars$mpg),
              labels=c("horrible","Bad","Good","Great"),include.lowest=T)

mtcars <- cbind(mtcars,mpgrate)

-OR-

mtcars$mpgrate <- mpgrate   # The column just magically appears !
```

Data Frames - Factors

```
head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb mpgrate
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4    Good
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4    Good
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1    Good
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1    Good
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2     Bad
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1     Bad

library(lattice)
bwplot(~mpg|mpgrate,data=mtcars,layout=c(1,4))
```

Data Frames - Factors



Data Frames - transform()

You can also use the **transform()** command to change the types/classes of the columns

```
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
transform(mtcars,wt = (wt*1000), qsec = round(qsec),
                       am = factor(am,labels=c("A","M")))
```

```
                     mpg cyl  disp  hp drat   wt qsec vs am gear carb
Mazda RX4           21.0   6 160.0 110 3.90 2620   16  0  M    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2875   17  0  M    4    4
Datsun 710          22.8   4 108.0  93 3.85 2320   19  1  M    4    1
Hornet 4 Drive      21.4   6 258.0 110 3.08 3215   19  1  A    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3440   17  0  A    3    2
```

Data Frames - Reading CSV

Many times data will be read in from a comma delimited ,("CSV"), file exported from Excel. The file can be read from local storage or from the Web.

```
url <- "https://raw.githubusercontent.com/pittardsp/bios545r_spring_2018/master/SUPPORT/hsb2.csv"
```

```
data1 <- read.table(url,header=T,sep=",")
```

```
head(data1)
  gender  id race ses schtyp  prgtype read write math science socst
1      0  70    4   1      1  general   57    52   41      47    57
2      1 121    4   2      1   vocati   68    59   53      63    61
3      0  86    4   3      1  general   44    33   54      58    31
4      0 141    4   3      1   vocati   63    44   47      53    56
5      0 172    4   2      1 academic   47    52   57      53    61
6      0 113    4   2      1 academic   44    52   51      63    61
```

Motivations

dplyr is an add on package designed to efficiently transform and summarize tablular data such as data frames. The package has a number of functions ("verbs") that perform a number of data manipulation tasks:

- Filtering rows
- Select specific columns
- Re-ordering or arranging rows
- Summarizing and aggregating data

One of the unique strengths of **dplyr** is that it implements what is known as a Split-Apply-Combine technique that we will explore in this session.

The dyplr function can also be used with the **magrittr** package for setting up workflows or pipelines to process data.

Motivations

- **dplyr** is designed to work with data frames but it can also connect to relational databases that are locally or remotely available.
- Access to data frames or databases is accomplished uisng the same set of tools. You don't have to use different commands.
- Relative to databases you use the "verbs" provided with dplyr that in turn are translated into the appropriate SQL statements necessary to interact with the databases.

How to Install dplyr ?

```
# install the package

install.packages("dplyr")
install.packages("readr")    # Get's the equivalent to data.table's fread p

# Loads the package

library(dplyr)

# Launches a browser to explore

browseVignettes(package = "dplyr")
```

Motivations

This slide deck references "Becoming a data ninja with dplyr" `https://speakerdeck.com/dpastoor/becoming-a-data-ninja-with-dplyr` and the dplyr tutorial
`http://genomicsclass.github.io/book/pages/dplyr_tutorial.html`

Motivations

- A data frame is a set of columns. Every column is same length but of possibly different types.
- It has characteristics of both a matrix, (each row is the same data type),
- Each column can be a different data type
- Bracket notation offers a convenient way to search through the data drame

Motivations

```
head(mtcars,12)
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
```

Motivations

There are some common activites associated with a data frame:

- filter - find observations satisfying some condition(s)
- select - selecting specific columns by name
- mutate - adding new columns or changing existing ones
- arrange - reorder or sort the rows
- summarize - do some aggregation or summary by groups

Motivations

```
df <- data.frame(id = 1:5,
                 gender = c("MALE","MALE","FEMALE","MALE","FEMALE"),
                 age = c(70,76,60,64,68))
```

ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

Filter

```
filter(df,gender == "FEMALE")
  id gender age
1  3 FEMALE  60
2  5 FEMALE  68
```

ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

ID	GENDER	AGE
3	FEMALE	60
5	FEMALE	68

Filter

```
filter(df, id %in% c(1,3,5))
  id gender age
1  1   MALE  70
2  3 FEMALE  60
3  5 FEMALE  68
```

ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

ID	GENDER	AGE
1	MALE	70
3	FEMALE	60
5	FEMALE	68

Mutate

Mutate is used to add or remove columns in a data frame

```
mutate(df,meanage = mean(age))
  id gender age meanage
1  1   MALE  70    67.6
2  2   MALE  76    67.6
3  3 FEMALE  60    67.6
4  4   MALE  64    67.6
5  5 FEMALE  68    67.6
```

ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

ID	GENDER	AGE	MEANWT
1	MALE	70	67.6
2	MALE	76	67.6
3	FEMALE	60	67.6
4	MALE	64	67.6
5	FEMALE	68	67.6

Mutate

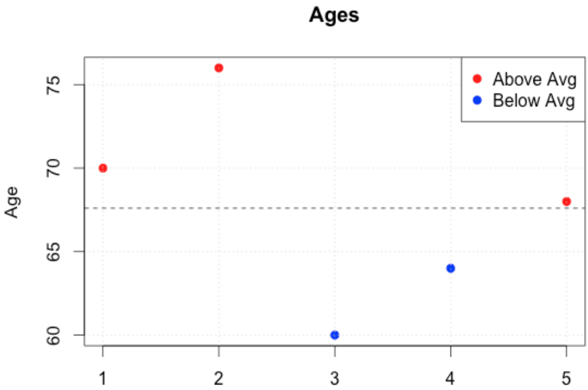
Here we create a new column designed to tell us if a given observation has an age that is greater than or equal to the average age.

We create a variable called old_young and assing a value of "Y" if the are above the mean age and a value of "N" if they are not.

```
mutate(df,old_young=ifelse(df$age>=mean(df$age),"Y","N"))
  id gender age old_young
1  1   MALE  70         Y
2  2   MALE  76         Y
3  3 FEMALE  60         N
4  4   MALE  64         N
5  5 FEMALE  68         Y
```

Mutate

```
tmp <- mutate(df, color = ifelse(age > mean(age),"red","blue"))
plot(tmp$age,col=tmp$color,type="p",pch=19,main="Ages",ylab="Age")
grid()
abline(h=mean(tmp$age),lty=2)
legend("topright",c("Above Avg","Below Avg"),col=c("red","blue"),pch=19)
```



Arrange

Use arrange for sorting the data frame by a column(s)

```
# Sort df by age from highest to lowest

arrange(df, desc(age))
  id gender age
1  2   MALE  76
2  1   MALE  70
3  5 FEMALE  68
4  4   MALE  64
5  3 FEMALE  60

# Sort df by gender (alphabetically) and then by age
# from highest to lowest

arrange(df, gender,desc(age))
  id gender age
1  5 FEMALE  68
2  3 FEMALE  60
3  2   MALE  76
4  1   MALE  70
5  4   MALE  64
```

Select

Select allows us to select groups of columns from a data frame

```
select(df,gender,id,age)  # Reorder the columns
  gender id age
1   MALE  1  70
2   MALE  2  76
3 FEMALE  3  60
4   MALE  4  64
5 FEMALE  5  68

select(df,-age)   # Select all but the age column
  id gender
1  1   MALE
2  2   MALE
3  3 FEMALE
4  4   MALE
5  5 FEMALE

select(df,id:age)  # Can use : to select a range
  id gender age
1  1   MALE  70
2  2   MALE  76
3  3 FEMALE  60
4  4   MALE  64
5  5 FEMALE  68
```

Select

You can select by regular expressions or numeric paterns

```
library(ggplot2)
data(diamonds)
names(diamonds)
 [1] "carat"   "cut"     "color"   "clarity" "depth"   "table"   "price"
 [8] "x"       "y"       "z"

head(select(diamonds,starts_with("c")))
  carat       cut color clarity
1  0.23     Ideal     E     SI2
2  0.21   Premium     E     SI1
3  0.23      Good     E     VS1
4  0.29   Premium     I     VS2
5  0.31      Good     J     SI2
6  0.24 Very Good     J    VVS2

head(select(diamonds,ends_with("t")))
  carat       cut
1  0.23     Ideal
2  0.21   Premium
3  0.23      Good
4  0.29   Premium
5  0.31      Good
6  0.24 Very Good
```

Select

You can select by regular expressions or numeric paterns

```
testdf <- expand.grid(m_1=seq(60,70,10),age=c(25,32),m_2=seq(50,60,10))

head(testdf, 4)
  m_1 age m_2
1  60  25  50
2  70  25  50
3  60  32  50
4  70  32  50

head( select(testdf,matches("_")) ,2)
  m_1 m_2
1  60  50
2  70  50

head( select(testdf,contains("_"), 2)
  m_1 m_2
1  60  50
2  70  50

head( select(testdf,num_range("m_",1:2)), 2)
  m_1 m_2
1  60  50
2  70  50
```

group_by

group_by let's you organize a data frame by some factor or grouping variable

```
df
  id gender age
1  1   MALE  70
2  2   MALE  76
3  3 FEMALE  60
4  4   MALE  64
5  5 FEMALE  68

group_by(df,gender)   # Hmm. Did this really do anything ?

Source: local data frame [5 x 3]
Groups: gender

  id gender age
1  1   MALE  70
2  2   MALE  76
3  3 FEMALE  60
4  4   MALE  64
5  5 FEMALE  68
```

group_by

group_by let's you organize a data frame by some factor or grouping variable

```
df
  id gender age
1  1   MALE  70
2  2   MALE  76
3  3 FEMALE  60
4  4   MALE  64
5  5 FEMALE  68

( gdf <- group_by(df,gender)    # Hmm. Did this really do anything ?

Source: local data frame [5 x 3]
Groups: gender

  id gender age
1  1   MALE  70
2  2   MALE  76
3  3 FEMALE  60
4  4   MALE  64
5  5 FEMALE  68
```

Summarize

```
summarize(group_by(df,gender),total=n())
Source: local data frame [2 x 2]

  gender total
1 FEMALE     2
2   MALE     3
```

ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

GENDER	TOTAL
FEMALE	2
MALE	3

Summarize

```
summarize(group_by(df,gender),av_age=mean(age))
Source: local data frame [2 x 2]

  gender av_age
1 FEMALE     64
2   MALE     70
```

ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

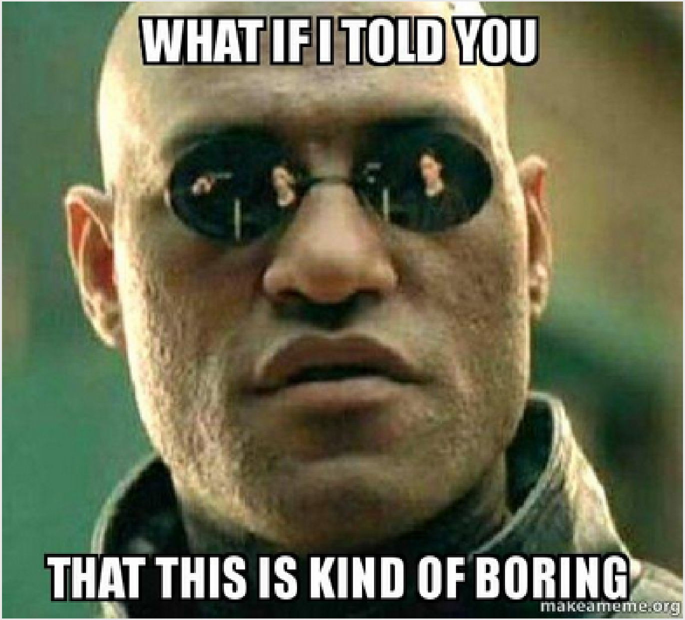
GENDER	AV_AGE
FEMALE	64
MALE	70

Summarize

```
summarize(group_by(df,gender),av_age=mean(age),total=n())
Source: local data frame [2 x 3]

  gender av_age total
1 FEMALE     64     2
2   MALE     70     3
```

ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

GENDER	AV_AGE	TOTAL
FEMALE	64	2
MALE	70	3


WHAT IF I TOLD YOU
THAT THIS IS KIND OF BORING
makeameme.org

Split -> Apply -> Combine

Split -> Apply -> Combine group_by

ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

ID	GENDER	AGE
1	MALE	70
2	MALE	76
4	MALE	64

ID	GENDER	AGE
3	FEMALE	60
5	FEMALE	68

AVG
70

AVG
64

ID	GENDER	AVG
1	MALE	70
2	FEMALE	64

Split -> Apply -> Combine

But do you really need dplyr to do this ? No but it makes it a lot easier

```
df
  id gender age
1  1   MALE  70
2  2   MALE  76
3  3 FEMALE  60
4  4   MALE  64
5  5 FEMALE  68

tapply(df$age,df$gender,mean)    # tapply function
FEMALE   MALE
    64     70

aggregate(age~gender,data=df,mean) # aggregate works also
 gender age
1 FEMALE  64
2   MALE  70

lapply(split(df,df$gender),function(x) mean(x$age)) # complicated
$FEMALE
[1] 64

$MALE
[1] 70
```

Split -> Apply -> Combine: Chaining

- Before moving forward let us consider the "pipe" operator that is included with the **magrittr** package. This is used to make it possible to "pipe" the results of one command into another command and so on.
- The inspiration for this comes from the UNIX/LINUX operating system where pipes are used all the time. So in effect using "pipes" is nothing new in the world of research computation.
- Warning: Once you get used to pipes it is hard to go back to not using them

Split -> Apply -> Combine: Chaining

When you load the dplyr package it in turn loads the necessary packages for supporting the piping capability. Let's use the mtcars data frame to illustrate the basics of the piping mechanism as used by dplyr.

Here we will select the mpg and am column from mtcars and view the top 5 rows.

```
head(select(mtcars, mpg, am))
                   mpg am
Mazda RX4         21.0  1
Mazda RX4 Wag     21.0  1
Datsun 710        22.8  1
Hornet 4 Drive    21.4  0
Hornet Sportabout 18.7  0
Valiant           18.1  0
```

Split -> Apply -> Combine: Chaining

Here we will select the mpg and am column from mtcars and view the top 5 rows but using dplyr and the piping operator. Instead of nesting functions (reading from the inside to the outside), the idea of of piping is to read the functions from left to right.

```
mtcars %>% select(mpg, am) %>% head
```

```
                   mpg am
Mazda RX4         21.0  1
Mazda RX4 Wag     21.0  1
Datsun 710        22.8  1
Hornet 4 Drive    21.4  0
Hornet Sportabout 18.7  0
Valiant           18.1  0
```

Split -> Apply -> Combine: Chaining

What about this ? We can chain together the output of one command to the input of another !

```
df %>% group_by(gender) %>% summarize(avg=mean(age))
Source: local data frame [2 x 2]

  gender avg
1 FEMALE  64
2   MALE  70

df %>% group_by(gender) %>% summarize(avg=mean(age),total=n())
Source: local data frame [2 x 3]

  gender avg total
1 FEMALE  64     2
2   MALE  70     3

df %>% filter(gender == "MALE") %>% summarize(med_age=median(age))
  med_age
1      70
```

Split -> Apply -> Combine: Chaining

What about this ? We can chain together the output of one command to the input of another !

```
df %>% filter(gender == "MALE") %>% summarize(med_age=median(age))
  med_age
1      70
```

df

ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

filter

ID	GENDER	AGE
1	MALE	70
2	MALE	76
4	MALE	64

summarize

med_age
70

Split -> Apply -> Combine: Chaining

Using the built in mtcars dataframe filter out records where the wt is greater than 3.3 tons.

Then create a column called ab_be (Y or N) that indicates whether that observation's mpg is greater (or not) than the average mpg for the filtered set.

Then present the average mpg for each group

```
mtcars %>% filter(wt > 3.3)  %>%
         mutate(ab_be=ifelse(mpg > mean(mpg),"Y","N")  ) %>%
         group_by(ab_be) %>% summarize(mean_mpg=mean(mpg))

Source: local data frame [2 x 2]

  ab_be mean_mpg
1     N 13.77778
2     Y 18.10000
```

Split -> Apply -> Combine: Chaining

Using the built in mtcars dataframe filter out records where the wt is greater than 3.3 tons.

```
mtcars %>% filter(wt > 3.3)
```

```
    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
1  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
2  18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
3  14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
4  19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
5  17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
6  16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
7  17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
8  15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
9  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
10 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
11 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
12 15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
13 15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
14 13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
15 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
16 15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
```

Split -> Apply -> Combine: Chaining

Create a column called ab_be (Y or N) that indicates whether that observation's mpg is greater (or not) than the average mpg for the filtered set.

```
mtcars %>% filter(wt > 3.3) %>%
             mutate(ab_be=ifelse(mpg > mean(mpg),"Y","N")  )
    mpg cyl  disp  hp drat    wt  qsec vs am gear carb ab_be
1  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2     Y
2  18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1     Y
3  14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4     N
4  19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4     Y
5  17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4     Y
6  16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3     Y
7  17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3     Y
8  15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3     N
9  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4     N
10 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4     N
11 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4     N
12 15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2     N
13 15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2     N
14 13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4     N
15 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2     Y
16 15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8     N
```

Split -> Apply -> Combine: Chaining

Then present the average mpg for each group as defined by ab_be

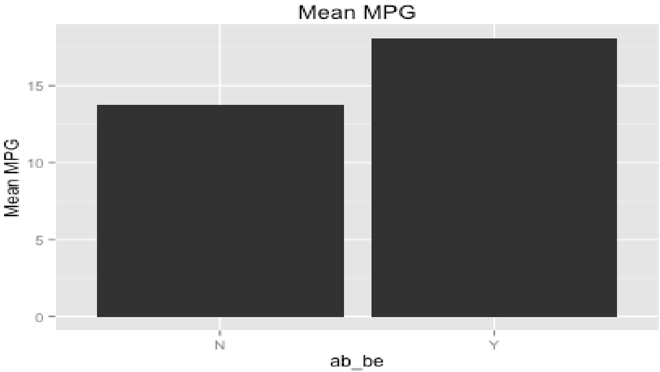
```
mtcars %>% filter(wt > 3.3)  %>%
          mutate(ab_be=ifelse(mpg > mean(mpg),"Y","N")  ) %>%
          group_by(ab_be) %>% summarize(mean_mpg=mean(mpg))
```

```
Source: local data frame [2 x 2]

  ab_be mean_mpg
1     N 13.77778
2     Y 18.10000
```

Split -> Apply -> Combine: Chaining

This could then be chained to the ggplot command

```
mtcars %>% filter(wt > 3.3)  %>%
         mutate(ab_be=ifelse(mpg > mean(mpg),"Y","N")  ) %>%
         group_by(ab_be) %>% summarize(mean_mpg=mean(mpg)) %>%
         ggplot(aes(x=ab_be,y=mean_mpg)) + geom_bar(stat="identity") +
         ggtitle("Mean MPG") + labs(x = "ab_be", y = "Mean MPG")
```



Large Files

Let's read in the file combined_wiki.txt.gz

```
library(readr)

dt <- read_delim("combined_wiki.zip",delim=" ")

nrow(dt)
[1] 31164567

head(dt,5)
   proj                              page acc bytes
1: aa.b                         Main_Page   1  5565
2: aa.b              MediaWiki:Image_sample   1  5179
3: aa.b        MediaWiki:Upload_source_file   1  5195
4: aa.b          Wikibooks:Privacy_policy   1  4925
5: aa.d MediaWiki:Group-abusefilter-member   1  4912
```

Large Files

Using dplyr commands, summarize the mean number of bytes (in megabytes) per unique project page and sort the resulting table in descencing order by the average in megabytes.

```
nrow(dt)
[1] 31164567

head(dt,5)
   proj                                page acc bytes
1: aa.b                           Main_Page   1  5565
2: aa.b              MediaWiki:Image_sample   1  5179
3: aa.b         MediaWiki:Upload_source_file   1  5195
4: aa.b            Wikibooks:Privacy_policy   1  4925
5: aa.d MediaWiki:Group-abusefilter-member   1  4912
```

Split -> Apply -> Combine: Chaining

Using dplyr commands, summarize the mean number of bytes (in megabytes) per unique project page and sort the resulting table in descencing order by the average in megabytes.

```
dt %>% mutate(MB=bytes/1000000) %>%
       group_by(proj)%>%
       summarize(avg=round(mean(MB),2)) %>%
       arrange(desc(avg))
Source: local data table [1,266 x 2]   # Note we have 1,266 rows

      V1      avg
1  en.mw 77518.22
2  ja.mw  9126.98
3  fr.mw  2020.45
4  ru.mw  1311.16
5  de.mw  1214.59
6  es.mw  1187.93
7  it.mw   472.27
8  zh.mw   374.91
9  ko.mw   234.63
10 pt.mw   207.78
```

Split -> Apply -> Combine: Chaining

Using dplyr commands, summarize the mean number of bytes (in megabytes) per unique project page and sort the resulting table in descencing order by the average in megabytes.

```
system.time( dt %>% mutate(MB=bytes/1000000) %>%
        group_by(proj)%>%
        summarize(avg=round(mean(MB),2)) %>%
        arrange(desc(avg)) )
```

```
   user  system elapsed
   1.93    2.58    7.79
```

dply vs Native R Commands

How long with this take using the standard native R commands ? First we create a function so we can easily time things. This example also assumes that we have read in the data using the native R command **read.csv** file command. That is we are not benefitting from the conveniences offered by the data.table command.

```
myaggre <- function(df) {
  df$bytes <- round(df$bytes/1000000,2)
  hold <- aggregate(bytes~proj,df,mean)
  hold <- hold[order(-hold$bytes),]
  return(hold)
}

system.time( myaggre(df))
   user  system elapsed
351.826  11.120 378.115
```

dply additional commands

Other activities are possible

```
mtcars %>% sample_n(2) # Sample 2 records from a data frame

                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4

# Sample 2 records from each cylinder group

mtcars %>% group_by(cyl) %>% do(sample_n(.,2))
Source: local data frame [6 x 11]
Groups: cyl

   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
1 21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
2 27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
3 19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
4 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
5 17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
6 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
```

dply additional commands

Other activities are possible. You can use “do” to perform arbitrary computation, returning either a data frame or arbitrary objects which will be stored in a list. This is particularly useful when working with models

```
by_cyl <- group_by(mtcars, cyl)
models <- by_cyl %>% do(mod = lm(mpg ~ disp, data = .))

Source: local data frame [3 x 2]
Groups: <by row>

  cyl     mod
1   4 <S3:lm>
2   6 <S3:lm>
3   8 <S3:lm>

summarise(models, rsq = summary(mod)$r.squared)
Source: local data frame [3 x 1]

        rsq
1 0.64840514
2 0.01062604
3 0.27015777

# Here is a one liner that does the above
```

Joining data frames

```
idatime <- data.frame(id=rep(1:3,each=2),time=rep(0:1,each=3))

  id time
1  1    0
2  1    0
3  2    0
4  2    1
5  3    1
6  3    1

idawt <- data.frame(id=c(1,2,4),wt=c(110,130,115))

  id  wt
1  1 110
2  2 130
3  4 115
```

Inner joins - inner_join(x,y)

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

idatime

id	time
1	0
1	0
2	0
2	1
3	1
3	1

idawt

id	wt
1	110
2	130
4	115

inner_join(idatime, idawt)

id	time	wt
1	0	110
1	0	110
2	0	130
2	1	130

inner_join(idawt, idatime)

id	wt	time
1	110	0
1	110	0
2	130	0
2	130	1

Inner joins - inner_join(x,y)

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

```
inner_join(idatime,idawt)
```

```
Joining by: "id"
  id time  wt
1  1    0 110
2  1    0 110
3  2    0 130
4  2    1 130
```

Joining data frames - left_join(x,y)

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

idatime

id	time
1	0
1	0
2	0
2	1
3	1
3	1

idawt

id	wt
1	110
2	130
4	115

left_join(idatime, idawt)

id	time	wt
1	0	110
1	0	110
2	0	130
2	1	130
3	1	NA
3	1	NA

left_join(idawt, idatime)

id	wt	time
1	110	0
1	110	0
2	130	0
2	130	1
4	115	NA

Joining data frames - anti_join(x,y)

returns all rows from x where there are not any matching values in y, keeping just the columns from x

idatime

id	time
1	0
1	0
2	0
2	1
3	1
3	1

idawt

id	wt
1	110
2	130
4	115

anti_join(idatime, idawt)

id	time
3	1
3	1

anti_join(idawt, idatime)

id	wt
4	115

Joining data frames

idatime

id	time
1	0
1	0
2	0
2	1
3	1
3	1

idawt

id	wt
1	110
2	130
4	115

semi_join(idatime, idawt)

id	time
1	0
1	0
2	0
2	1

semi_join(idawt, idatime)

id	wt
1	110
2	130

Joining data frames

Return all rows from x where there are matching values in y, keeping just columns from x

idatime

id	time
1	0
1	0
2	0
2	1
3	1
3	1

idawt

id	wt
1	110
2	130
4	115

semi_join(idatime, idawt)

id	time
1	0
1	0
2	0
2	1

semi_join(idawt, idatime)

id	wt
1	110
2	130

Exercises

Now it is your turn. Let's do some exercises

```
url <- "https://raw.githubusercontent.com/pittardsp/bios545r_spring_2018/master/SUPPORT/msleep_ggplot2.csv"

library(readr)

download.file(url,"msleep_ggplot2.csv")
msleep <- read_csv("msleep_ggplot2.csv")

names(msleep)
 [1] "name"          "genus"         "vore"          "order"         "conservation"
 [6] "sleep_total"   "sleep_rem"     "sleep_cycle"   "awake"         "brainwt"
[11] "bodywt"
```

Exercises

Here is a description of the columns / variables

```
COLUMN NAME   DESCRIPTION

name            common name
genus           taxonomic rank
vore            carnivore, omnivore or herbivore?
order           taxonomic rank
conservation    the conservation status of the mammal
sleep_total     total amount of sleep, in hours
sleep_rem       rem sleep, in hours
sleep_cycle     length of sleep cycle, in hours
awake           amount of time spent awake, in hours
brainwt         brain weight in kilograms
bodywt          body weight in kilograms
```

Exercises

Using the msleep data frame let's do the following activities and answer some questions. Try to use the chaining operator:

- Select the name and sleep_total columns
- Using the colon operator select all columns between name and order
- Select all columns that begin with "sl"
- Filter the data frame to find only rows with a sleep_total $>=$ 16
- Filter the rows for mammals that sleep a total of more than 16 hours and have a body weight of greater than 1 kilogram
- Arrange the data frame using the **order** column

Exercises

For these you will need to use the chaining operator:

- Select three columns from msleep, arrange the rows by the taxonomic order and then arrange the rows by sleep_total. Finally show the head of the final data frame
- Same as above, except here we filter the rows for mammals that sleep for 16 or more hours instead of showing the head of the final data frame
- Use the **mutate** function to create a new column called rem_proportion which is the ratio of rem sleep to total amount of sleep

Exercises

- Use the summarise() function: to compute the average number of hours of sleep, apply the mean() function to the column sleep_total and call the summary value avg_sleep.
- Group the msleep data frame by taxanomic order and then summarize the mean sleep total
- Same as above except summarize the mean, max, and min sleep total