Be Lazy - Intro to dplyr

Department of Biostatistics and Bioinformatics Emory University

Steve Pittard wsp@emory.edu

October 26, 2016

Goals

- R has many different ways to do the same thing
- This is both a strength and weakness of R
- Memorizing individual commands and associated arguments is tedious
- Identify tools and packges that use a philosophy or approach (e.g. lattice, ggplot2, plyr, data.table, dplyr, caret)
- Simplicity in workflow is everything!
- After you have a solid workflow you can be lazy (at least as it relates to the various commands)

Data - First Steps

When first approaching data there are usually three major activities one pursues (not in any specific order):

- Data Aggregation: The process of taking some data and putting it into a form that lends itself easily to summary, e.g. replace groups of observations with summary statistics
- Data Restructuring: Change the structure of the data so that its new form is more convenient for a specific purpose (usually analysis). This can also involve merging existing data frames or bringing in new data (e.g. using SQL)
- Oata Cleaning: This could be considered as a sub activity to Data Restructuring. Here we address things like missing values, improbable or illegal values, and merging of disparate data sources

Aggregation

Command	Package	Purpose	
table, xtabs	Native R	Summarize grouping variables with	
		other grouping variables, Create	
		Contingency Tables	
tapply, split	Native R	Summarize a continuous variable by	
		grouping variables	
aggregate	Native R	Summarize continuous variable(s) by	
		grouping variables	
data table	data.table	Select rows and columns and aggre-	
		gat and summarize	
group_by, summarize	dplyr	Implement Split-Apply-Combine ap-	
		proach for aggregation and summary	

Restructuring / Reshaping / Tidying

Command	Package	Purpose
melt, cast	reshape2	Convert data frames from wide to long
		format and do aggregation
ddply, ldply, l_ply	plyr	Implement Split-Apply-Combine ap-
		proach for aggregation and summary
gather, spread	tidyr	Change a data frame from wide to long
		(or vice versa)
reshape	Native R	Change a data frame from wide to long
		(or vice versa)
stack, unstack	Native R	Change a data frame from wide to long
		(or vice versa)

Data - First Steps

mtcars is the most (in)famous data frame in R education

Motor Trend Car Road Tests

Description

The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).

Usage

mtcars

Format

A data frame with 32 observations on 11 variables.

- [, 1] mpg Miles/(US) gallon
- [, 2] cyl Number of cylinders
- [, 3] disp Displacement (cu.in.)
- [, 4] hp Gross horsepower
- [, 5] drat Rear axle ratio
- [, 6] wt Weight (1000 lbs)
- [, 6] Wt Weight (1000 ibs
- [, 7] qsec 1/4 mile time
- [, 8] vs V/S
- [, 9] am Transmission (0 = automatic, 1 = manual)
- [,10] gear Number of forward gears
- [,11] carb Number of carburetors

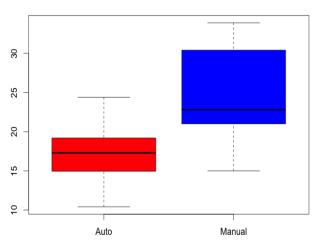
Data - First Steps

mtcars is the most infamous data frame in R education

```
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 ...
 $ cvl : num 6646868446 ...
 $ 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 ...
# How many unique values does each column take ? This helps identify
# possible factors
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
```

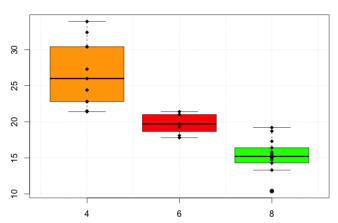
mtcars\$am <- factor(mtcars\$am,labels=c("Auto","Manual"))
boxplot(mpg~am,data=mtcars,col=c(2,4),main="MPG by Transmission Type")</pre>

MPG by Transmission Type



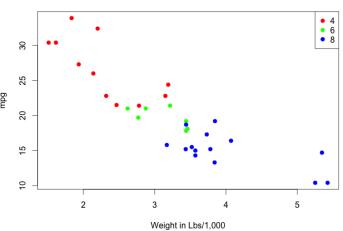
```
mtcars$cyl <- factor(mtcars$cyl)</pre>
boxplot(mpg~cyl,data=mtcars,
        col=c("orange", "red", "green"),
        main="MPG by Cylinders")
grid()
# We will also plot the MPG points for each category. To do this
# we will "split" up the data frame on Cylinder which takes a
# value of 4,6, or 8
mydf <- split(mtcars,mtcars$cyl)</pre>
lapply(1:3,function(x) {
                         points(rep(x,length(mydf[[x]]$mpg)),
                                   mydf [[x]] $mpg,
                                   pch=18)
                         })
```

MPG by Cylinders



```
# Make a scatterplot of all MPG and provide each cylinder
# group with its own color
plot(mpg~wt,data=mtcars,type="n",main="MPG vs. Weight",
     xlab="Weight in Lbs/1,000")
# Split the data frame by cylinder value and create
# a color vector
mys <- split(mtcars,mtcars$cyl)</pre>
colors <- rainbow(3)
# Loop through the split data frame and put up points for each
# cylinder group
lapply(seq_along(mys),function(x) {
                          points(mys[[x]]$wt,
                                 mvs[[x]]$mpg.
                                 col=colors[x],pch=19)
                        })
# Draw a legend
legend("topright",legend=sort(unique(mtcars$cyl)),pch=19,col=colors)
grid()
```





Aggregation - tables

```
# How many 4,6, or 8 cylinder cars are there for each transmission type ?
table(Transmission=mtcars$am,Cylinders=mtcars$cyl)
           Cvlinders
Transmission 4 6 8
          0 3 4 12
          1 8 3 2
# Create a proportion table
prop.table(table(Transmission=mtcars$am,Cylinders=mtcars$cyl))
           Cvlinders
Transmission
          0 0.09375 0.12500 0.37500
          1 0.25000 0.09375 0.06250
# Add margins if you wish
addmargins(prop.table(table(Transmission=mtcars$am,Cylinders=mtcars$cyl)))
           Cvlinders
Transmission
                      6
                                        Sum
          0.09375 0.12500 0.37500 0.59375
            0.25000 0.09375 0.06250 0.40625
        Sum 0.34375 0.21875 0.43750 1.00000
```

Aggregation - tables

R knows how to plot tables without you having to supply too much information.

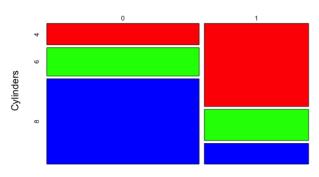
```
# How many 4,6, or 8 cylinder cars are there for each transmission type ?
```

```
myt <- table(Transmission=mtcars$am,Cylinders=mtcars$cyl)</pre>
```

```
plot(myt,color=rainbow(3),main="Transmissions vs Cylinders")
```

Aggregation - tables

Transmissions vs Cylinders



Transmission

Aggregation - xtabs

The **xtabs** function does much the same as **table** but it has the advantage of supporting a formula interface which R uses in many statistical modeling functions.

```
xtabs(~cyl,data=mtcars)
cyl
11 7 14
xtabs(~am+cyl,data=mtcars)
   cyl
  0 3 4 12
# We can subset out data
xtabs(~am+cyl,data=mtcars,subset=mpg < 25)</pre>
   cyl
am 4 6 8
  0 3 4 12
```

Aggregation - tapply

Summarize a continuous quantity in terms of each category.

```
# Average MPG for each transmission type
tapply(mtcars$mpg,mtcars$am,mean)
17.14737 24.39231
# For each cylinder group
tapply(mtcars$mpg,mtcars$cyl,mean)
26.66364 19.74286 15.10000
# We can supply our own function
tapply(mtcars$mpg,mtcars$am,function(x) return(c(mean=mean(x),sd=sd(x))))
$ 0
                 sd
     mean
17.147368 3.833966
$11
                 sd
     mean
24.392308 6.166504
```

Aggregation - tapply

• With tapply we can summarize a single continuous quantity across multiple categories

 But we cannot conveniently summarize multiple continuous quantities across categories

```
tapply(list(mtcars$mpg,mtcars$hp), list(mtcars$am, mtcars$vs), mean)
Error in tapply(list(mtcars$mpg, mtcars$hp), list(mtcars$am, mtcars$vs), :
    arguments must have same length
```

Aggregation - aggregate

This is where it gets confusing. We have to resort to another function. Why so many different approaches ?

```
# Here we summarize two continuous values in terms of two categories
aggregate(cbind(mpg,hp)~am+vs,data=mtcars,mean)
      am vs
                mpg
   Auto 0 15.05000 194.16667
2 Manual 0 19.75000 180.83333
   Auto 1 20.74286 102.14286
4 Manual 1 28.37143 80.57143
# We summarize mpg in terms of two categories
aggregate(mpg~cyl+am,data=mtcars,mean)
 cyl
                 mpg
      Auto 22,90000
   6 Auto 19,12500
       Auto 15.05000
   4 Manual 28,07500
   6 Manual 20.56667
```

8 Manual 15,40000

6

Aggregation - aggregate

```
# We can even filter out data as we do the aggregate
# Get the mean MPG per Cylinder and Transmission groups
# but only in cases where MPG is > 20
aggregate(mpg~cyl+am,data=mtcars,mean,subset=mpg>20)
 cyl
         am
               mpg
      Auto 22,900
  6 Auto 21,400
3 4 Manual 28.075
   6 Manual 21,000
# Similar but
aggregate(mpg~cyl,data=mtcars,range,subset=wt>mean(wt))
 cvl
         am
                 mpg
   4 Auto 22,90000
   6 Auto 19,12500
   8 Auto 15.05000
  4 Manual 28.07500
5
   6 Manual 20.56667
6
   8 Manual 15,40000
```

Are you Confused? / R you Confused?



dplyr



Hadley Wickham is also the author of these packages:

Command	Package Purpose
ggplot2	Data Visualization
tidyr	For tidying data
stringr	Working with Strings
lubridate	Working with dates and times
readr	Efficiently read .csv and fwf files
readxl	Reading .xls and .xlsx files
haven	For Reading SAS, SPSS, and Stata files
rvest	Scraping Web Sites
xml2	For Importing XML files
reshape2	Restructure and aggregate data
plyr	Tools for Splitting, Applying, and Combining Data

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.

- **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.
- Install dplyr as you would any other R package

There are some common activities 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

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
      MALE 70
                 67.6
      MALE
            76
               67.6
    FEMALE
           60
               67.6
      MALE 64
               67.6
5
    FEMALE
            68
                 67.6
```

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

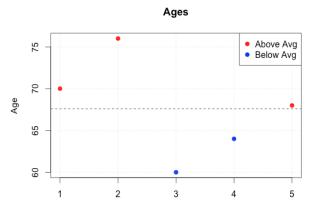
ID	GENDER	AGE	MEANWT
1	MALE	70	67.6
2	MALE	76	67.6
3	FEMALE	60	67.6
4	MALE	64	67.6
5	FEMALE	68	67.6

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

```
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
   2
       MALE
             76
   1
       MAT.F.
             70
  5 FEMALE
            68
4
       MAT.F.
            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
   5 FEMALE
            68
   3 FEMALE
             60
3
   2
       MALE
             76
      MALE
             70
5
       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
   MALE
            70
   MAT.F.
        2 76
3 FEMALE 3 60
   MALE 4 64
5 FEMALE 5 68
select(df,-age) # Select all but the age column
 id gender
      MALE
      MAT.F.
  3 FEMALE
      MALE
  5 FEMALE
select(df,id:age) # Can use : to select a range
  id gender age
      MALE 70
      MAT.F.
            76
  3 FEMALE
           60
      MALE
           64
  5 FEMALE
            68
```

group_by

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

```
df
 id gender age
      MALE 70
      MALE
           76
  3 FEMALE 60
      MALE 64
  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
      MALE 70
      MALE 76
  3 FEMALE 60
      MALE 64
  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

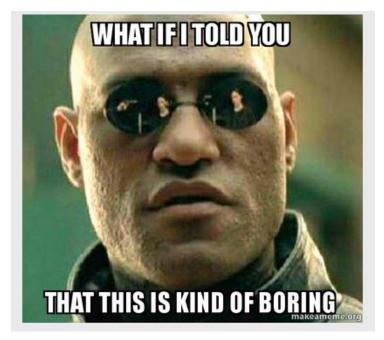
Summarize

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



- 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

Pipes

- A pipeline is a sequence of processes chained together by their standard streams, so that the output of each process (stdout) feeds directly as input (stdin) to the next one - Wikipedia
- The standard shell syntax for pipelines is to list multiple commands, separated by vertical bars ("pipes" in common unix verbiage) -Wikipedia

For each line in the passwd file list only those userids

```
# that do not begin with a "_" character. Also ignore lines
# beginning with "#"

$ cat /etc/passwd | awk -F: '{print $1}' | grep -v _ | grep -v "#"
nobody
root
daemon
```

- When you load the dplyr package it in turn loads the necessary packages for supporting the piping capability.
- Let's use our "toy" data frame to illustrate the basics of the piping mechanism as used by dplyr.
- Here we will select the gender and age columns from df and view the top 3 rows

```
head(select(df, gender, age),3)
gender age
1 MALE 70
2 MALE 76
3 FEMALE 60
```

- Here we will select the gender and age columns from df and view the top 3 rows 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.

```
df %>% select(gender, age) %>% head(3)
  gender age
1   MALE  70
2   MALE  76
3  FEMALE  60
```

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
   MAI.F. 70
df %>% group_by(gender) %>% summarize(avg=mean(age),total=n())
Source: local data frame [2 x 3]
 gender avg total
1 FEMALE 64
   MALE 70
df %>% filter(gender == "MALE") %>% summarize(med_age=median(age))
 med_age
1
      70
```

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
```

_	_
_	•
\sim	•
•	

filter

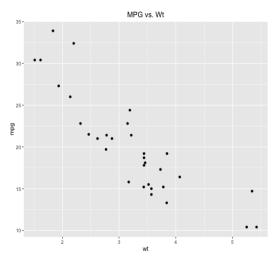
summarize

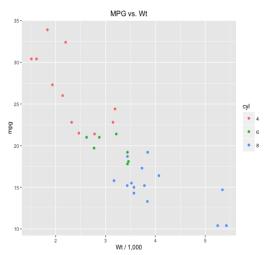
ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

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

med_age
70

mtcars %>% ggplot(aes(x=wt,y=mpg)) + geom_point() + ggtitle("MPG vs. Wt")





Split -> Apply -> Combine: Chaining Much easier and more logical than this:

```
# Make a scatterplot of all MPG and provide each cylinder
# group with its own color
plot(mpg~wt,data=mtcars,type="n",main="MPG vs. Weight",
     xlab="Weight in Lbs/1,000")
# Split the data frame by cylinder value and create
# a color vector
mys <- split(mtcars,mtcars$cyl)</pre>
colors <- rainbow(3)
# Loop through the split data frame and put up points for each
# cylinder group
lapply(seq_along(mys),function(x) {
                        points(mys[[x]]$wt,
                               mys[[x]]$mpg,
                               col=colors[x],pch=19)
                      })
# Draw a legend
legend("topright",legend=sort(unique(mtcars$cyl)),pch=19,col=colors)
grid()
```

- 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

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

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

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

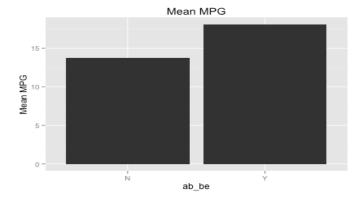
```
mtcars %>% filter(wt > 3.3) %>%
    mutate(ab_be=ifelse(mpg > mean(mpg),"Y","N") ) %>%
    group_by(ab_be) %>% summarize(mean_mpg=mean(mpg))
```

Source: local data frame [2 x 2]

```
ab_be mean_mpg
1 N 13.77778
2 Y 18.10000
```

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")
```



The Bay Area Bike Share program allows Bay area residents to use bicycles for commuting in the area. Here is a description of the service from the website: I obtained usage data from 2013-2015.



THE DATA

Each trip is anonymized and includes:

- Bike number
- · Trip start day and time
- · Trip end day and time
- · Trip start station
- Trip end station
- Rider type Annual or Casual (24-hour or 3-day member)
- If an annual member trip, it will also include the member's home zip code

The data set also includes:

- · Weather information per day per service area
- Bike and dock availability per minute per station

Separately, participants may use our live JSON feed:

- Access our feed at bayareabikeshare.com/stations/json
- No API key needed, however apps should not fetch this feed more than once per minute

The Bay Area Bike Share program allows Bay area residents to use bicycles for commuting in the area. Here is a description of the service from the website: I obtained usage data from 2013-2015.

```
library(readr)
url <- "http://steviep42.bitbucket.org/YOUTUBE.DIR/SF.ZIP"</pre>
download.file(url, "SF.ZIP")
system("unzip SF.ZIP")
stations <- read_csv("station.csv")</pre>
# Read in the trip date - you might get some messages
# about missing zipcodes but don't worry if you do
trips <- read_csv("trip.csv")</pre>
str(stations)
```

Classes tbl df, tbl and data.frame: 70 obs. of 7 variables:

```
str(stations)
```

```
$ id
                   : int 2 3 4 5 6 7 8 9 10 11 ...
$ name
                   : chr "San Jose Diridon Caltrain Station" "San Jose Civic Center" "Santa Clara at Almaden"
$ lat
                   : num 37.3 37.3 37.3 37.3 37.3 ...
$ long
                  : num -122 -122 -122 -122 -122 ...
                  : int 27 15 11 19 15 15 15 15 15 19 ...
$ dock count
                   : chr "San Jose" "San Jose" "San Jose" "San Jose" ...
$ city
$ installation date: chr "8/6/2013" "8/5/2013" "8/6/2013" "8/5/2013" ...
str(trips)
Classes tbl df, tbl and data.frame:
                                   669959 obs. of 11 variables:
$ id
                    : int 4576 4607 4130 4251 4299 4927 4500 4563 4760 4258 ...
$ duration
                    : int 63 70 71 77 83 103 109 111 113 114 ...
$ start date
                    : chr "8/29/2013 14:13" "8/29/2013 14:42" "8/29/2013 10:16" "8/29/2013 11:29" ...
$ start_station_name: chr "South Van Ness at Market" "San Jose City Hall" "Mountain View City Hall" "San Jose
$ start_station_id : int 66 10 27 10 66 59 4 8 66 10 ...
$ end date
                    : chr "8/29/2013 14:14" "8/29/2013 14:43" "8/29/2013 10:17" "8/29/2013 11:30" ...
$ end_station_name : chr "South Van Ness at Market" "San Jose City Hall" "Mountain View City Hall" "San Jose
$ end_station_id : int 66 10 27 10 67 59 5 8 66 11 ...
$ bike id
                    : int 520 661 48 26 319 527 679 687 553 107 ...
$ subscription_type : chr "Subscriber" "Subscriber" "Subscriber" "Subscriber" ...
                    : int. 94127 95138 97214 95060 94103 94109 95112 95112 94103 95060 ...
$ zip_code
```

How Many Bikes Are There ?

```
trips %>% select(bike_id) %>% distinct()
Source: local data frame [700 x 1]
   bike id
     (int)
       520
       661
3
        48
        26
5
       319
6
       527
7
       679
8
       687
       553
10
       107
```

But if we wanted a single result we could do this
trips %>% select(bike_id) %>% distinct() %>% nrow()
[1] 700

. . .

How many times was each bike used ?

. . .

Source: local data frame [700 x 2]

```
bike id times used
     (int)
                  (int)
1
       392
                   2061
       489
                   1975
3
       558
                   1955
       267
                   1951
5
       631
                   1948
6
       518
                   1942
7
       532
                   1933
8
       592
                   1932
9
       395
                   1927
10
       368
                   1926
```

. . .

How many times was each bike used ?

```
trips %>% count(bike_id,sort=TRUE)
Source: local data frame [700 x 2]
```

```
bike_id
                n
     (int) (int)
       392
             2061
2
3
       489
            1975
       558
             1955
       267
            1951
5
       631
            1948
6
       518
             1942
7
       532
            1933
8
       592
            1932
9
       395
             1927
10
       368
            1926
```

How many cities use this service? How many stations per city are there?

```
stations %>% count(city)
Source: local data frame [5 x 2]
           city
          (chr) (int)
1 Mountain View
      Palo Alto
  Redwood City
4 San Francisco
                   35
5
       San Jose
                   16
# We could also sort the result from highest count to lowest
stations %>% count(city,sort=TRUE)
Source: local data frame [5 x 2]
           city
          (chr) (int)
 San Francisco
       San Jose
                16
3 Mountain View
  Redwood City
5
      Palo Alto
```

Recall that in the **stations** data frame we have latitutde and logitude information.

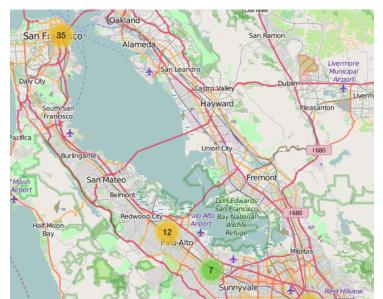
```
str(stations)
```

```
Classes tbl_df, tbl and data.frame: 70 obs. of 7 variables:
                   : int 234567891011...
 $ id
 $ name
                  : chr "San Jose Diridon Caltrain Station" "San Jose Civic Cent
 $ lat
                  : num
                         37.3 37.3 37.3 37.3 ...
$ long
                  : num -122 -122 -122 -122 ...
 $ dock_count
                : int
                         27 15 11 19 15 15 15 15 15 19 ...
 $ city
                  : chr
                         "San Jose" "San Jose" "San Jose" "San Jose" ...
 $ installation_date: chr
                         "8/6/2013" "8/5/2013" "8/6/2013" "8/5/2013" ...
```

Use the leaflet package to create an interactive map. This is a package that integrates with the Javascript library of the same name. The numbers on the graphic are interactive. Click on one of the cicles to drill down into the count data

 \mathbf{m}

Use the leaflet package to create an interactive map



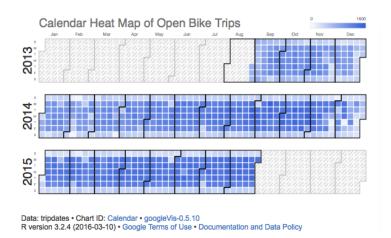
How many trips did not finish on the same day they began ?

How many trips were there for each year ?

trips %>% count(substr(start_date,1,4))

We will look at the number of trips for each day of each year and use the **googleVis** package to create a heatmap to visualize this

```
library(googleVis)
trips %>% mutate(start_date=as.Date(start_date),
                 end_date=as.Date(end_date)) %>%
                 filter(start date == end date) %>%
                 count(start_date) -> tripdates
# Create a Gvisplot and then plot it
plot(
  gvisCalendar(data=tripdates, datevar="start_date", numvar="n",
               options=list(
                 title="Calendar Heat Map of Open Bike Trips",
                 calendar="{cellSize:10.
                 yearLabel:{fontSize:20, color:'#444444'},
                 focusedCellColor:{stroke:'red'}}".
                 width=590, height=320),
               chartid="Calendar")
```



- How do the dplyr commands work on a really large data file? Here we want to read in a 31 million row file that relates to some Wikipedia statistics
- I will be using a supporting package called readr which can efficiently read in large files.
- My computer has 8GB of RAM. If you have less then this example might be slow.

Let's read in the file

```
library(readr)
url <- "http://steviep42.bitbucket.org/YOUTUBE.DIR/combined_wiki.zip"
download.file(url, "combined_wiki.zip")
system("unzip combined_wiki.zip")
dt <- read_delim("combined_wiki.zip",delim=" ")</pre>
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
```

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
  en.mw 77518.22
  ja.mw
         9126.98
  fr.mw
         2020.45
         1311.16
  111. MW
         1214.59
  de.mw
         1187.93
 es.mw
  it.mw 472.27
  zh.mw 374.91
8
  ko.mw 234.63
10 pt.mw
          207.78
```

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.

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.

Ackowledgements

This slide deck references:

- Becoming a data ninja with dplyr https://speakerdeck.com/ dpastoor/becoming-a-data-ninja-with-dplyr
- dplyr tutorial http://genomicsclass.github.io/book/pages/ dplyr_tutorial.html

Additional Material

Also see my Blog and Youtube Channel

- http://rollingyours.wordpress.com
- http://youtube.com/biorsph
- My Consulting and R Training Company at http://pconsdec.com