

Approaching Big Data Using dplyr

Department of Biostatistics and Bioinformatics

Steve Pittard wsp@emory.edu

April 11, 2017

MATH & STATISTICS

- ☆ Machine learning
- ☆ Statistical modeling
- ☆ Experiment design
- ☆ Bayesian inference
- ☆ Supervised learning: decision trees, random forests, logistic regression
- ☆ Unsupervised learning: clustering, dimensionality reduction
- ☆ Optimization: gradient descent and variants

DOMAIN KNOWLEDGE & SOFT SKILLS

- ☆ Passionate about the business
- ☆ Curious about data
- ☆ Influence without authority
- ☆ Hacker mindset
- ☆ Problem solver
- ☆ Strategic, proactive, creative, innovative and collaborative



PROGRAMMING & DATABASE

- ☆ Computer science fundamentals
- ☆ Scripting language e.g. Python
- ☆ Statistical computing packages, e.g., R
- ☆ Databases: SQL and NoSQL
- ☆ Relational algebra
- ☆ Parallel databases and parallel query processing
- ☆ MapReduce concepts
- ☆ Hadoop and Hive/Pig
- ☆ Custom reducers
- ☆ Experience with xaaS like AWS

COMMUNICATION & VISUALIZATION

- ☆ Able to engage with senior management
- ☆ Story telling skills
- ☆ Translate data-driven insights into decisions and actions
- ☆ Visual art design
- ☆ R packages like ggplot or lattice
- ☆ Knowledge of any of visualization tools e.g. Flare, D3.js, Tableau

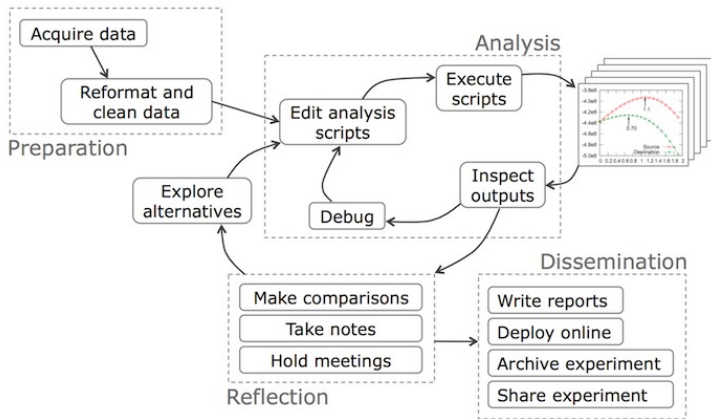
Solutions

Cheating is Good ! Well no not really. What I meant to say was that *Cheat Sheets* are good !

<https://www.rstudio.com/resources/cheatsheets/>

- Data Import
- Data Transformation
- R Markdown
- RStudio
- Shiny
- Data Visualization
- Package Development
- Advanced R
- Regular Expressions

Data Science Workflow



<https://cacm.acm.org/blogs/blog-cacm/169199-data-science-workflow-overview-and-challenges/fulltext>

Data Science Workflow

The acquisition and preparation of data can take up a lot of time and energy and usually does because data can:

- Come from many sources (spreadsheets, lab instruments, URLs, databases, APIs)
- Have different formats (.CSV, XML, JSON, CDF, SQL, HTML, TEXT)
- Come from lab instruments like Genomic Sequencing Machines
- Require different packages to parse and clean
- Be “dirty” because of missing and/or improbable values
- Require merging from different sources into a more manageable format (e.g. data frame or data base)

Data Science Workflow

Need to track the point of origin (“provenance”) for all data to:

- Reference sources accordingly in subsequent publications
- Insure that data is up to date
- Insure that it can be re-downloaded to execute updated experiments
- Enable reproducibility by other users

Managing data can be challenging after downloading. One must:

- Develop reasonable naming conventions
- Create an intelligent folder structure for reference and storage
- Make it easy to add newly generated data that fits existing conventions
- Consider using Databases if multi-user access is required

This is all too much ! The stress !



Motivations

Data frames are an excellent way to host data:

- Intuitive, Easy to Subset and Interrogate
- Import tools like **read.csv** give you a data frame

```
head(mtcars,2)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4

```
# Find all rows where MPG > 30 MPG
```

```
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

Motivations

You can easily read data straight off the Internet. This results in a Data frame

```
# This data relates to attributes of some Wines. Let's read it  
# from my Github. The URL is long so I create it in steps
```

```
str1 <- "https://raw.githubusercontent.com/steviep42/youtube/"
```

```
url <- paste0(str1,"master/YOUTUBE.DIR/wines.csv")
```

```
my.wines <- read.csv(url, header=TRUE, stringsAsFactors=FALSE)
```

```
str(my.wines)
```

```
'data.frame': 5 obs. of 8 variables:
```

```
$ Wine : Factor w/ 5 levels "Wine_1","Wine_2",...: 1 2 3 4 5
```

```
$ Hedonic: int 14 10 8 2 6
```

```
$ Meat : int 7 7 5 4 2
```

```
$ Dessert: int 8 6 5 7 4
```

```
$ Price : int 7 4 10 16 13
```

```
$ Sugar : int 7 3 5 7 3
```

```
$ Alcohol: int 13 14 12 11 10
```

```
$ Acidity: int 7 7 5 3 3
```

Motivations

There are different ways to interrogate a data frame:

```
# Find all cars with automatic transmission and MPG > 20
```

```
subset(mtcars, mpg > 20 & 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
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
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1

```
# Same as
```

```
mtcars[mtcars$mpg > 20 & mtcars$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
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
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1

Motivations

There are various aggregation commands

```
# Look at the average MPG and Wt for each Transmission type
```

```
mtcars$am <- factor(mtcars$am,labels=c("Auto","Manual"))
aggregate(cbind(mpg,wt)~am,mtcars,mean)
```

	am	mpg	wt
1	0	17.14737	3.768895
2	1	24.39231	2.411000

```
# We can supply our own Function
```

```
myfunc <- function(x) {
  return(c(avg=mean(x),variance=var(x)))
}
```

```
aggregate(cbind(mpg,wt)~am, mtcars, myfunc)
```

	am	mpg.avg	mpg.variance	wt.avg	wt.variance
1	Auto	17.14737	14.69930	3.7688947	0.6043510
2	Manual	24.39231	38.02577	2.4110000	0.3806663

Motivations

But R was written at a time when data was not so BIG. You could fit data into the memory of your desktop and/or laptop.

- R tries to load everything into RAM
- Do you really need to see all rows of a data frame ?
- Experimental data types can be HUGE (many GB or even TB)
- What about accessing databases ? (Oracle, MySQL, etc)
- There are multiple ways to interrogate or transform data frames
- Is there a “best practice” approach ?

Solutions

If you data is biological or genomic in nature it is quite possible that there are specialized packages available to help you manage these files

- R BioConductor has many such tools
-



<https://www.bioconductor.org/>

Solutions

For more general forms of data see the CRAN Task View that lists packages by research domain. Here is a subset of packages

- CRAN Task Views

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
ExtremeValue	Extreme Value Analysis
Finance	Empirical Finance
Genetics	Statistical Genetics

<https://cran.r-project.org/web/views/>

Solutions

Data can be so large that we have to “split” the data across multiple computers (aka “nodes”)

Assume $N = 1,000,000$

	Col 1	Col 2	..	Col X
Row 1				
Row 2				
..				
..				
Row N				

	Col 1	Col 2	..	Col X
Row 1				
Row 2				
..				
Row 100,000				

Node 1

	Col 1	Col 2	..	Col X
Row 100,001				
..				
Row 200,000				

Node 2

.....
.....

	Col 1	Col 2	..	Col X
Row 900,001				
..				
Row 1,000,000				

Node 10

Solutions

But we don't want to have to (re)learn alot to be productive !

Assume N = 1,000,000

	Col 1	Col 2	..	Col X
Row 1				
Row 2				
..				
..				
Row N				

	Col 1	Col 2	..	Col X
Row 1				
Row 2				
..				
Row 100,000				

Node 1

	Col 1	Col 2	..	Col X
Row 100,001				
..				
Row 200,000				

Node 2

.....

.....

	Col 1	Col 2	..	Col X
Row 900.001				
..				
Row 1,000,000				

Node 10

Solutions

Some solutions for handling large and unwieldy data

- The data.table package
 - ▶ Handles large files well
 - ▶ Combines aggregation with subsetting
- Use SQL to query databases
 - ▶ Very flexible
 - ▶ Requires knowledge of SQL
- Use Apache Spark (a Distributed data frame usually on the cloud)
 - ▶ Handles HUGE (multi-terabyte) data frames
 - ▶ Requires a HUGE computer or use of Amazon
 - ▶ Has an R interface

dplyr is an add on package designed to efficiently transform and summarize tabular 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.

tidyverse

- **dplyr** is part of the tidyverse which is a “collection of R packages that share common philosophies and are designed to work together”
- The best place to learn the complete philosophy of the tidyverse is the “R for Data Science” book. <http://r4ds.had.co.nz/>
- Summary of tidyverse functions:
 - ▶ **readr** for importing data from files
 - ▶ **tibble** a modern iteration on data frames
 - ▶ **tidyr** functions to rearrange data for analysis
 - ▶ **dplyr** functions to filter, arrange, subset, modify and aggregate data frames
 - ▶ **ggplot2** a system for declaratively creating graphics, based on The Grammar of Graphics

- The tidyverse was written mostly by the same person, Hadley Wickham, who has insured that these programs work together smoothly
- These packages do not overwrite any of the native R commands. So there are no collisions between existing and new commands
- Once you learn the philosophy behind a given package it becomes much easier than trying to piece together individual native R commands
- Do not feel obligated to learn them all at once. Start with say **ggplot2** and **dplyr** and use the others as needed
- We'll focus on **dplyr** for this session



- **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 can be accomplished as before so older code will not break
- But you should use the “verbs” provided by dplyr since they simplify things (in my opinion)
- Relative to databases use the “verbs” provided with dplyr that in turn are translated into appropriate SQL statements to interact with the databases.

How to Install dplyr ?

```
# install the package
```

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

```
# Get's the equivalent to data.table's fread package
```

```
install.packages("readr")
```

```
# Loads the package
```

```
library(dplyr)
```

```
# Launches a browser to explore
```

```
browseVignettes(package = "dplyr")
```

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

dplyr - data frames

- 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 frame

dplyr - data frames

```
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

dplyr - data frames

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

dplyr - data frames

```
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.

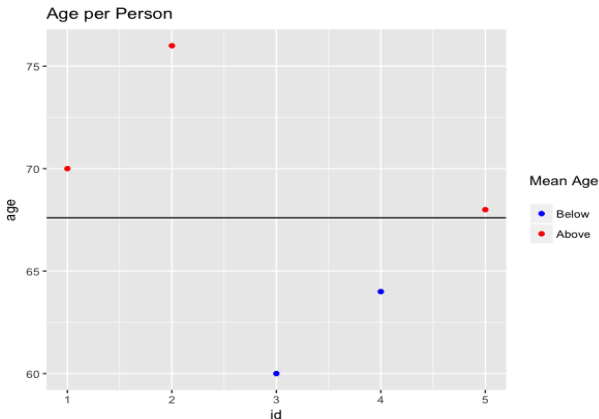
We create a variable called `old_young` and assign a value of “Y” if they are above the mean age and a value of “N” if they are not.

```
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"))
ggplot(tmp,aes(x=id,y=age)) +
  geom_point(aes(color=color)) +
  geom_hline(aes(yintercept=mean(age))) +
  labs(title="Age per Person",color="Mean Age\n") +
  scale_color_manual(labels = c("Below", "Above"), values = c("blue", "red"))
```



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 patterns

```
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 patterns

```
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

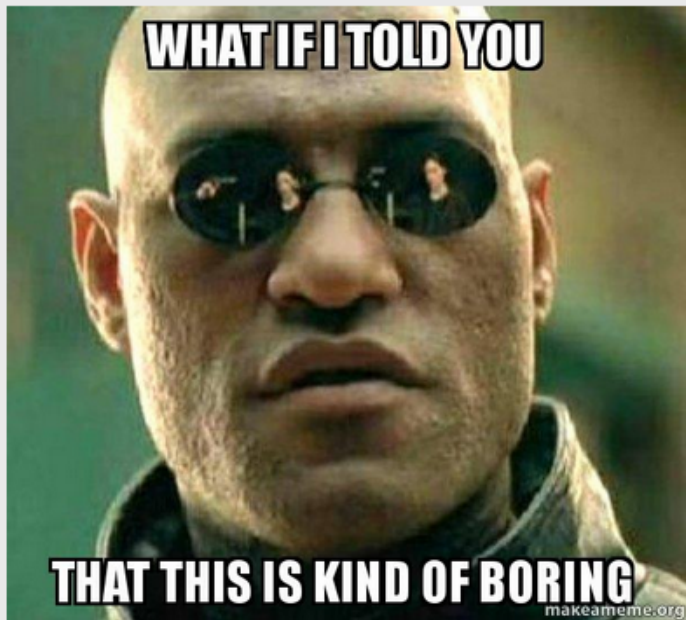
```
summarize(group_by(df,gender),av_age=mean(age),total=n())
```

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



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

AVG
70

ID	GENDER	AGE
3	FEMALE	60
5	FEMALE	68

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

- Loading the dplyr package also loads the necessary packages for supporting the piping capability
- Here we will select the age and id column from df and view the top 3 rows.

```
head(select(df, age, id),3)
```

	age	id
1	70	1
2	76	2
3	60	3

Split -> Apply -> Combine: Chaining

Here we will select the age and id column from df and view the top 3 rows. Instead of nesting functions, the idea of piping is to read the functions from left to right.

```
df %>% select(age, id) %>% head(3)
```

	age	id
1	70	1
2	76	2
3	60	3

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

<u>med_age</u>
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))
```

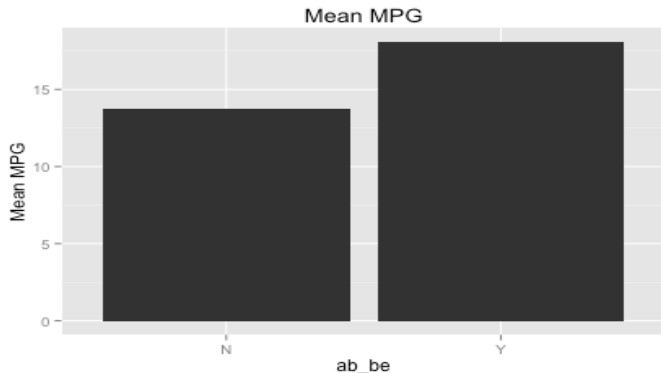
Source: local data frame [2 x 2]

	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")
```



dplyr 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

dplyr 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		idawt		inner_join(idatime, idawt)			inner_join(idawt, idatime)		
id	time	id	wt	id	time	wt	id	wt	time
1	0	1	110	1	0	110	1	110	0
1	0	2	130	1	0	110	1	110	0
2	0	4	115	2	0	130	2	130	0
2	1			2	1	130	2	130	1
3	1								
3	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

Exercises

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

```
load(dplyr)

url <- "https://raw.githubusercontent.com/stevie42/youtube/master/YOUTUBE.DIR/msleep_ggplot2.csv"

msleep <- read.csv(url)

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

Exercises

Remember - here are the following commands available to you from dplyr

- 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

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 taxonomic order and then summarize the mean sleep total
- Same as above except summarize the mean, max, and min sleep total