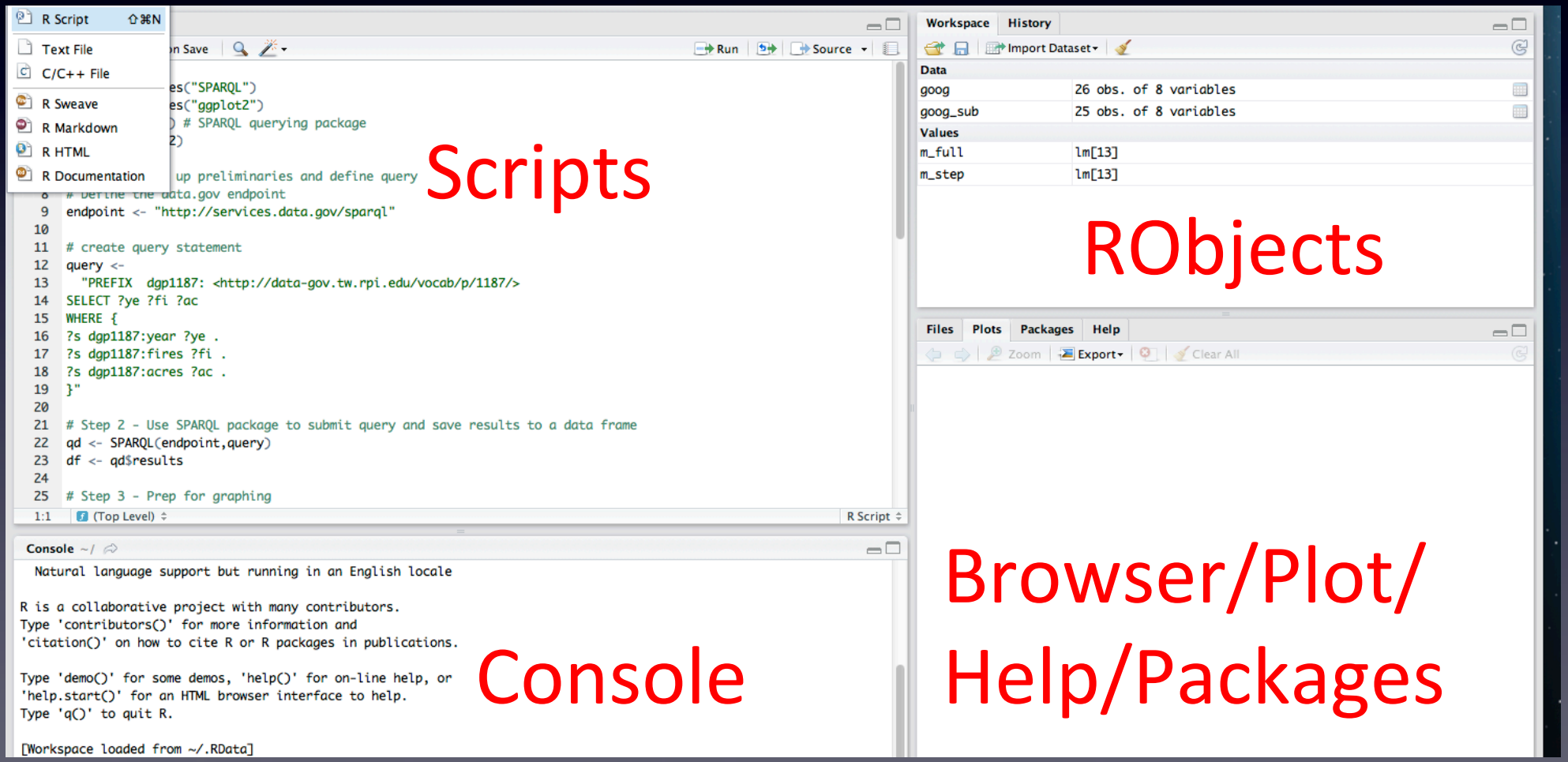
Technology Fundamentals for Analytics

Jason Kuruzovich

Working with RSTUDIO



Tips

- Always work in a script
- Highlight just a few lines and run those when debugging, writing code
- Double click on objects
- When not sure, try gui menu (stwd) or help(command)

General Strategies

- Split, Apply, Combine
  - Sometimes it is relevant to use the above strategy to manage large data transformations
  - http://www.jstatsoft.org/v40/i01
- Train, Test
  - Data is often spit into a component used for training and another used for testing

Object Storage

- All Values in R stored as Objects (some types)

Vector: A one-dimensional array of arbitrary length. Subsets of the vector may be referenced. All elements of the vector must be of the same type--numerical, character, etc.

Matrix: A two-dimensional array with an arbitrary number of rows and columns. Subsets of the matrix may be referenced, and individual rows and columns of the matrix may be handled as vectors. All elements of the matrix must be of the same type.

Data frame: A set of **data** organized similarly to a matrix. However each column of the data frame may contain its own type of data. Columns typically correspond to variables in a statistical study, while rows correspond to observations of these variables. A data frame may be handled similarly to a matrix, and individual columns of the data frame may be handled as vectors.

List: An arbitrary collection of other **R** objects (which may include other lists).

Variables

Numeric: Standard number

Integer: No decimals

String: Free text

Factor: Levels/Groups (Easily translate to dummy variables)

Boolean: (TRUE or FALSE)

Date (or DateTime): Date

AFTER IMPORTING DATA MAKE SURE DATA IS OF CORRECT TYPE

Vector Examples

Create vector with concatenate (c) and then refer to specific position with vector[position]

```
20  # Let's say we want to capture the ages for
21  # 4 students.
22  ages<-c(18,19,18,23)
23
24  #to see  this we can just type the name of object
25  ages
26
27  #to pick a specific value, we can, indicate it.
28  ages[4]
29
30  #or a range
31  ages[2:4]
```

Vectors Examples

Can also create vectors of different types

```
33  #Other vectors
34  names<-c("Sally", "Jason", "Bob", "Susy") #Text
35  female<-c(TRUE, FALSE, FALSE, TRUE)
36  grades<-c(20, 15, 13, 19) #25 points possible
37
```

Vector Examples

We can apply functions to vectors

```
38  #We can apply funtions to Vectors
39  names.length<-nchar(names) #Calc number of letters and return vector
40  names.length #Prints integer vector vector
41
42  #You can also include logic
43  names.length.gt4<-nchar(names) > 4
44  names.length.gt4 #what type of vector is returned?
45  #or
46  names.length.gt4b<-names.length > 4
47  names.length.gt4b #what type of vector is returned?
48
```

Combining Vectors

Combining Boolean with other vectors can be used for filtering

```
33  #Other vectors
34  names<-c("Sally", "Jason", "Bob", "Susy") #Text
35  female<-c(TRUE, FALSE, FALSE, TRUE)
36  grades<-c(20, 15, 13, 19) #25 points possible
37
```

```
46  names.length.gt4b<-names.length > 4
47  names.length.gt4b #what type of vector is returned?
48
49  #look at select we can do when combining vectors
50  names[female] #what type of vector is returned and what content?
51  names[names.length.gt4] #what type of vector is returned and what content?
52
```

Vector Examples

Often it is very useful to apply a function on an entire vector at once

```
53  #We can also do math on the entire vector at once
54  #our grades were out of 25, lets curve 3 points.
55  curve<-grades+3
56
57  #Now we can calculate a percentage
58  percent<-grades*(100/25)  #same as * 4
59  percent
60
61  #We can also take the log
62  logpercent<-log(percent)
```

Matrix Example

A Matrix is similar to a vector, but in 2 dimensions

matrixname[row, column]

```
#Create a matix by combining vectors
mat<-cbind(ages, grades)
mat #show entire matrix

#Matrices can be specified by mat[row,column]
mat[2,1] #Row=2, Column=1
mat[1,]  #Row=1 and all columns
mat[,1]  #Column=1, all rows
```

Matrix Limitations

- A matrix must be of all one data type.
- A data frame is more flexible and can have multiple data types, so usually we will use data frames.

```
#Now let's combine data of different types
#You can see that all data is being treated as
# string, which is not appropriate.
mat2<-cbind(names, ages, grades)
mat2

#A data frame is a more flexible format and one
#we will use for the majority of our analyses.
df1<-data.frame(cbind(names,ages,grades))
df1<-data.frame(cbind(names,ages,grades))
```

Working with Files/Scripts

When working with data, it is relevant to understand the consoles current working directory, where it is looking for files

```
85  #In reality most of the time we will be working with
86  #files (but can also use file browser)
87  getwd()
88  setwd("~/Dropbox/30_classes/analytics/2013_fall/labs/rlab1")
89  list.files()
90  #(if this doesn't show "batting.csv"
91  #you set the wrong working directory)
```

Working with Files

Note: This was pulled from a database download, so had to specify the na.strings

Because working directory is specified, no need to indicate full path

```
#We don't have to specify the full path here
#This is the baseball batting data
batting=read.csv(file="batting.csv", header=TRUE,sep=",", na.strings = "NULL")
teams=read.csv(file="teams.csv", header=TRUE,sep=",", na.strings = "NULL")
```

Viewing Data

```
98  #Now let's view the data
99  View(teams) #show data browser
100 names(teams) #show the names
101 dim(teams) #show the dimensions of the data frame
102 head(teams, 2) #show the first 2 records
103 tail(teams, 4) #show the final 2 records
104 teams$yearID #show the years in the data frame
105 summary(teams) #summarize all variables
106 str(teams) #shows the structure of an R Object
```

Factors and "Dummy Variables"

Factors correspond with different groups or categories

```
#Notice the differences, factors, integers, numeric
#League ID (just note that this is a factor object)
#This is the type of object that incorporates different "levels"
#and can be translated into "dummy variables" quite easily
teams$lgID  #show the variable and levels
```

Dummy Coding/R

R takes care of the process of dummy coding variables automatically

Dummy coding is necessary for categorical and (usually) ordinal variables

Assigns a binary indicator (dummy variable) to indicate group membership

For **n exclusive categories** (i.e., you can only be member of 1 category), **you need n-1 dummy variables**

Dummy Coding Example

Color	C1	C2
Red	1	0
Blue	0	1
Red	1	0
Yellow	0	0
Yellow	0	0
Blue	….	….

Here 3 colors are coded into 2 dummy variables, which then can be incorporated into a regression

Working with Data Frames

- Changing Data Type
- Subseting Data
- Recoding Data
- Calculating new fields

Recoding Data

Often necessary to change type of data

```
114  #recode it as a character
115  as.character(teams$lgID) #translate factor to string and print results
116  teams$lgIDS<-as.character(teams$lgID) #factor->string->new dataframe field
117  #Notice the factor levels correspond to different numbers.
118  cbind(teams$lgID, teams$lgIDS)
119
120  #Sometimes you may have to translate string to a factor.
121  str(teams$lgIDS) #this indicates the variable is a charcacter
122  teams$lgIDS<-as.factor(teams$lgIDS)
123
124  #You might have to treat integers as numeric.
125  str(teams$W)
126  teams$W<-as.numeric(teams$W)
127  str(teams$W)
```

Recoding Data

When working with time series data, dates must be specified as dates

```
133  #Now let's do a date conversion.
134  #YearID is set so that there is only a year, but dates need day month.
135  datez <- paste("01","01", teams$yearID, sep = "/")
136
137  #Two different functions to change date
138  as.Date(datez, "%m/%d/%y")
139  #Look at the help (?strptime) to see why capital Y
140  #This is a more flexible datetime See
141  #http://www.stat.berkeley.edu/classes/s133/dates.html
142  strptime(datez,"%m/%d/%Y")
143
```

How would you add the recoded date to the data frame?

Recoding Data

```
144  #Now let's add to the dataframe
145  teams$date<-as.Date(datez, "%m/%d/%y")
146  teams$datetime<-strptime(datez,"%m/%d/%Y")
147  str(teams$date) #View the Structure
148  str(teams$datetime) #View the Structure
```

Subset Data

Subset enables selection of specific rows and columns

```
#We want to subset the data, as the game has changed
#over time, so we will drop everything before 1980
#This is performing the same function of where clause.
batting.1980<-subset(batting, yearID > 1980)
teams.1980<-subset(teams, yearID > 1980)

#We could also just select out those variables we are actually
#interested in
teams.1980.small<-subset(teams, yearID > 1980,
                         select = c("yearID","teamID","W","L"))
View(teams.1980.small)
```

← ROWS

Columns

Recoding Data

Often we need to recode data -> Making one variable from another

For Loops (Recoding data)

```
166  #Now let's create a different variable from another variable. First
167  #let's get the average number of wins
168  mean.W<-mean(teams.1980$W)
169  mean.W
170
171  #Let's call teams who wone more that average a "winner" while
172  #those less than the mean "loser."
173  condition <-  teams.1980$W>mean.W
174  teams.1980$Season=ifelse(condition, "winner", "loser")
175
176  #We can also generate a boolean variable right from the
177  #condition
178  condition  #This will print the values
179  teams.1980$Winsea<-condition
```

Recoding Data

You can do calculations either by fully specifying the variables with the data frame or by using the "with" command

```
212  #In our SQL Statement, we had calcualted these.
213  #H/AB AS AVG,
214  #(H+BB+HBP)/(AB+BB+HBP+SF) AS OBP
215  #(H+2B+2*3B+3*HR)/AB AS SLG
216  batting$AVG<-(batting$H/batting$AB)
217
218  #This works, but we don't want to always have to repeat the dataframe
219  batting$AVGb<-with(batting, H/AB)
220
```

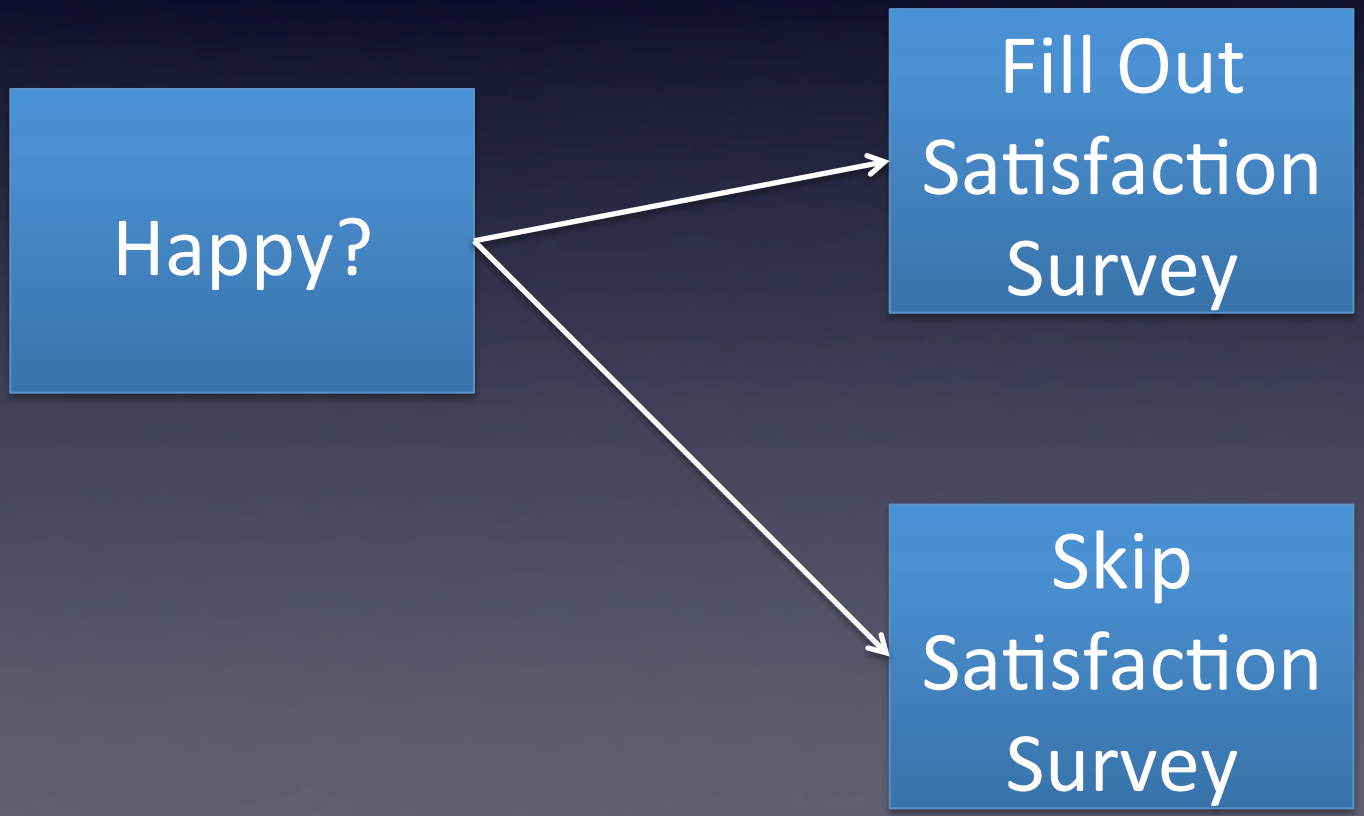
Recoding Multiple Variables

"with" can calculate one variable while "within" can calculate multiple variables.

```
224  # WE could calculate each this way
225  batting$AVGb<-with(batting, H/AB)
226  batting$OBP<-with(batting, (H+BB+HBP)/(AB+BB+HBP+SF))
227  batting$SLG<-with(batting, (H+B2+2*B3+3*HR)/AB)
228
229  #Here we can use within to find a number of different variables
230  batting2 <- within(batting, {
231                     AVG<-(H/AB)
232                     OBP<-(H+BB+HBP)/(AB+BB+HBP+SF)
233                     SLG<-(H+B2+2*B3+3*HR)/AB
234                     })
235
```

Dealing with Missing Values

Often missing values can be quite important, and "why" they are missing should be investigated



In this case the reason why the values are missing may lead to a bias in the results. This is known as selection bias.

Dealing with Missing Values

```
236  #Sometimes it is necessary to go through the process
237  #Of dropping NA records. First let us subselect some of the batting data
238  batting.1980.small<-subset(batting2, yearID > 1980,
239                     select = c("playerID","yearID", "teamID","AVG","OBP","SLG"))
240  View(batting.1980.small)
241
242  #These are 2 different ways of eliminating our cases with NA.
243  batting.1980.smallna<- na.omit(batting.1980.small)
244  batting.1980.smallnab <- airquality[complete.cases(batting.1980.small), ]
245
```

Aggregating Data

Many times data must be aggregated so that it can be included in an analysis

(SQL -> Group by)

Aggregating Data

Notice how we are specifying columns using specifically using 3:4 (this is W/L)

```
250  #This is the aggregation of data for W and L
251  aggdatac <-aggregate(teams.1980.small[,3:4],
252                       by=list(teams.1980.small$yearID), FUN=mean, na.rm=TRUE)
253  aggdatad <-aggregate(teams.1980.small[,3:4],
254                       by=list(teams.1980.small$teamID), FUN=mean, na.rm=TRUE)
255
```

Aggregate Data

Alternative to standard function:

plyr

The split-apply-combine strategy for R

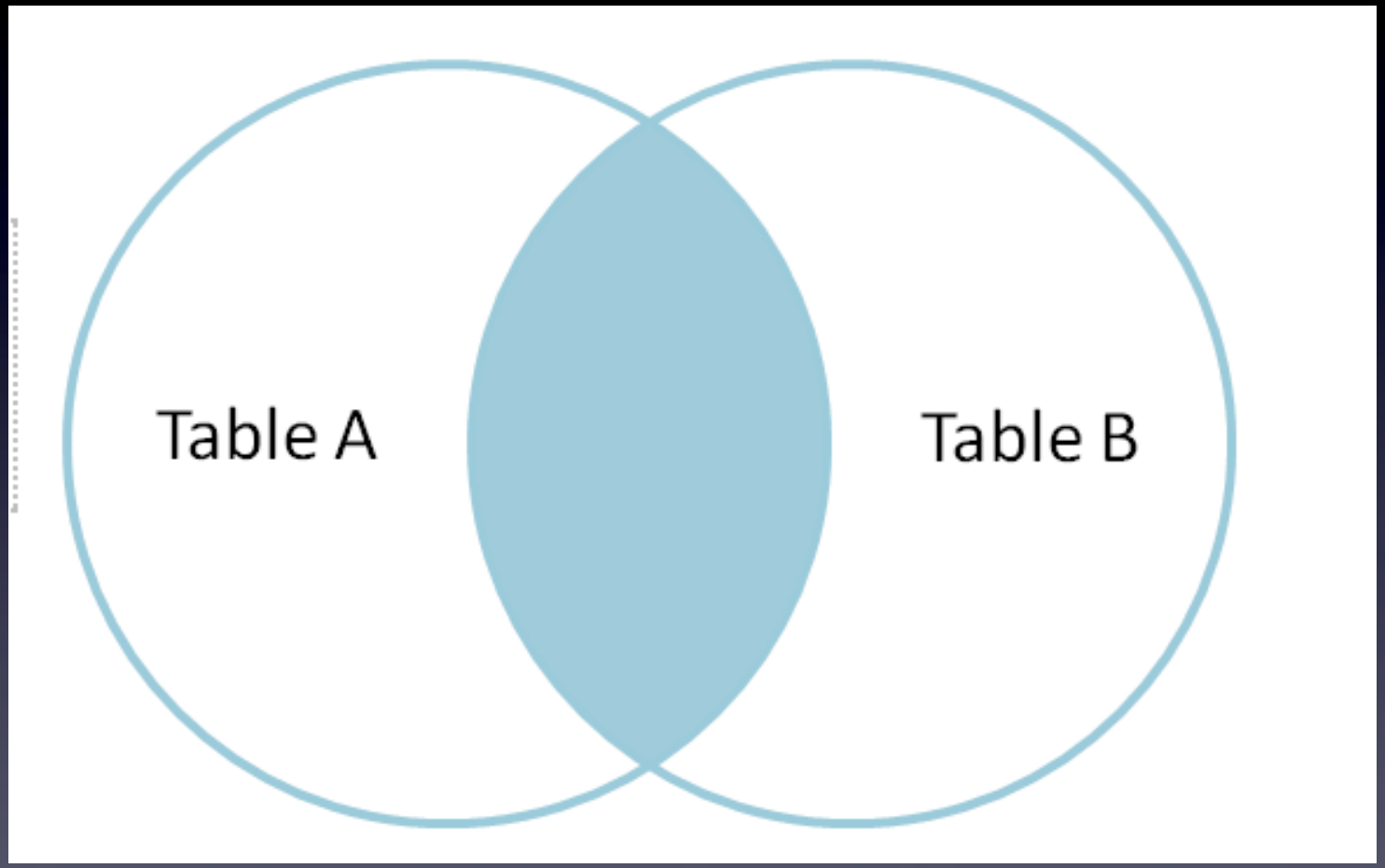
plyr is a set of tools for a common set of problems: you need to **split** up a big data structure into homogeneous pieces, **apply** a function to each piece and then **combine** all the results back together. For example, you might want to:

- fit the same model to subsets of a data frame
- quickly calculate summary statistics for each group
- perform group-wise transformations like scaling or standardising

Aggregate Data

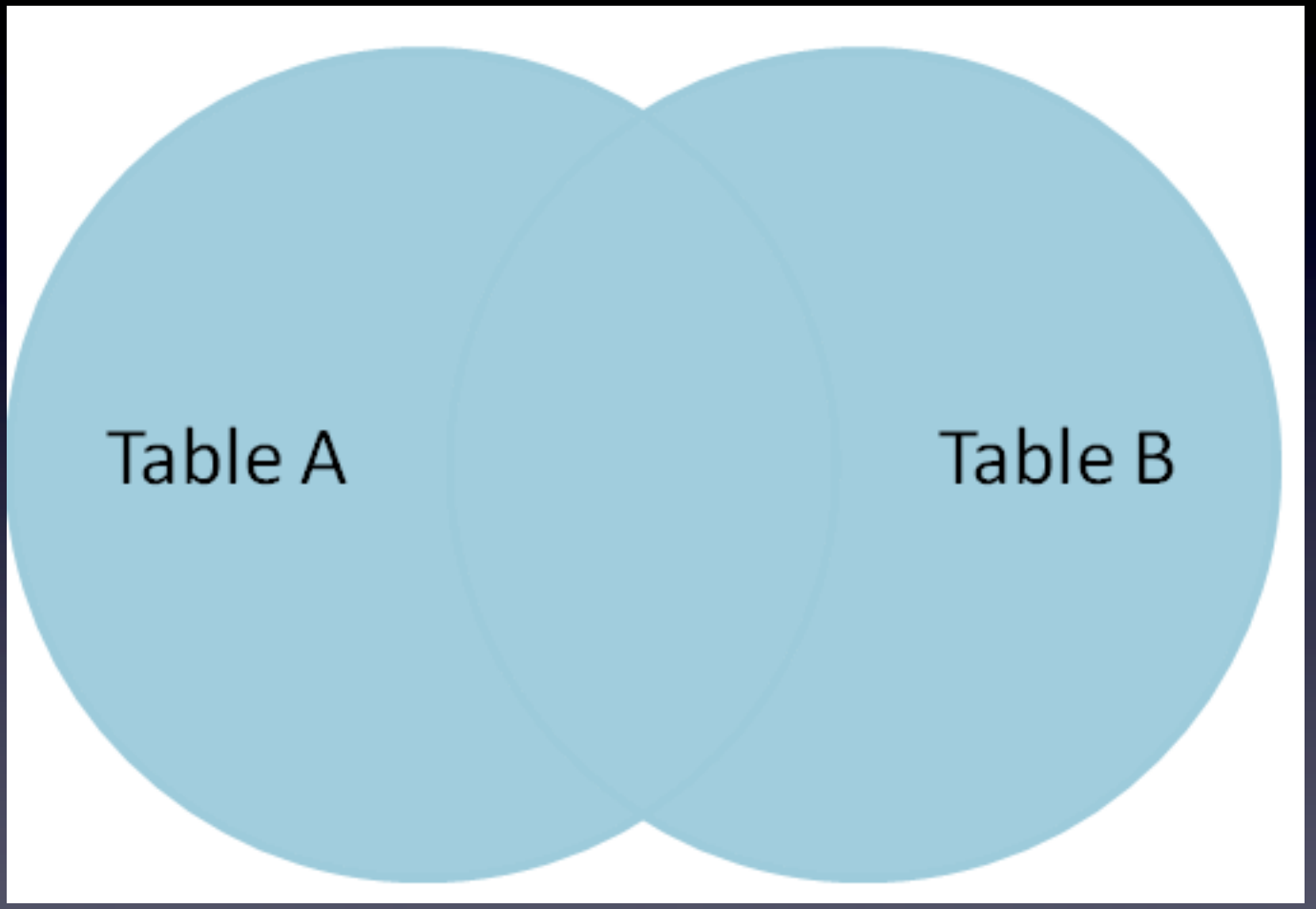
```
256 #Plyr is a commonly used package that automates the split, apply, combine model
257 #
258 install.packages("plyr")
259 library(plyr)
260 #ddply takes care of a number of summarizing
261 ddply(teams.1980.small, .(teamID), summarize,
262                                    N  = length(W), #Gives records (rows)
263                                mean_W = mean(W),   #Gives average
264                                std_W  = sd(W),     #Gives standard deviation
265                                mean_L = mean(L),
266                                std_L   = sd(L))
```

Join Datasets



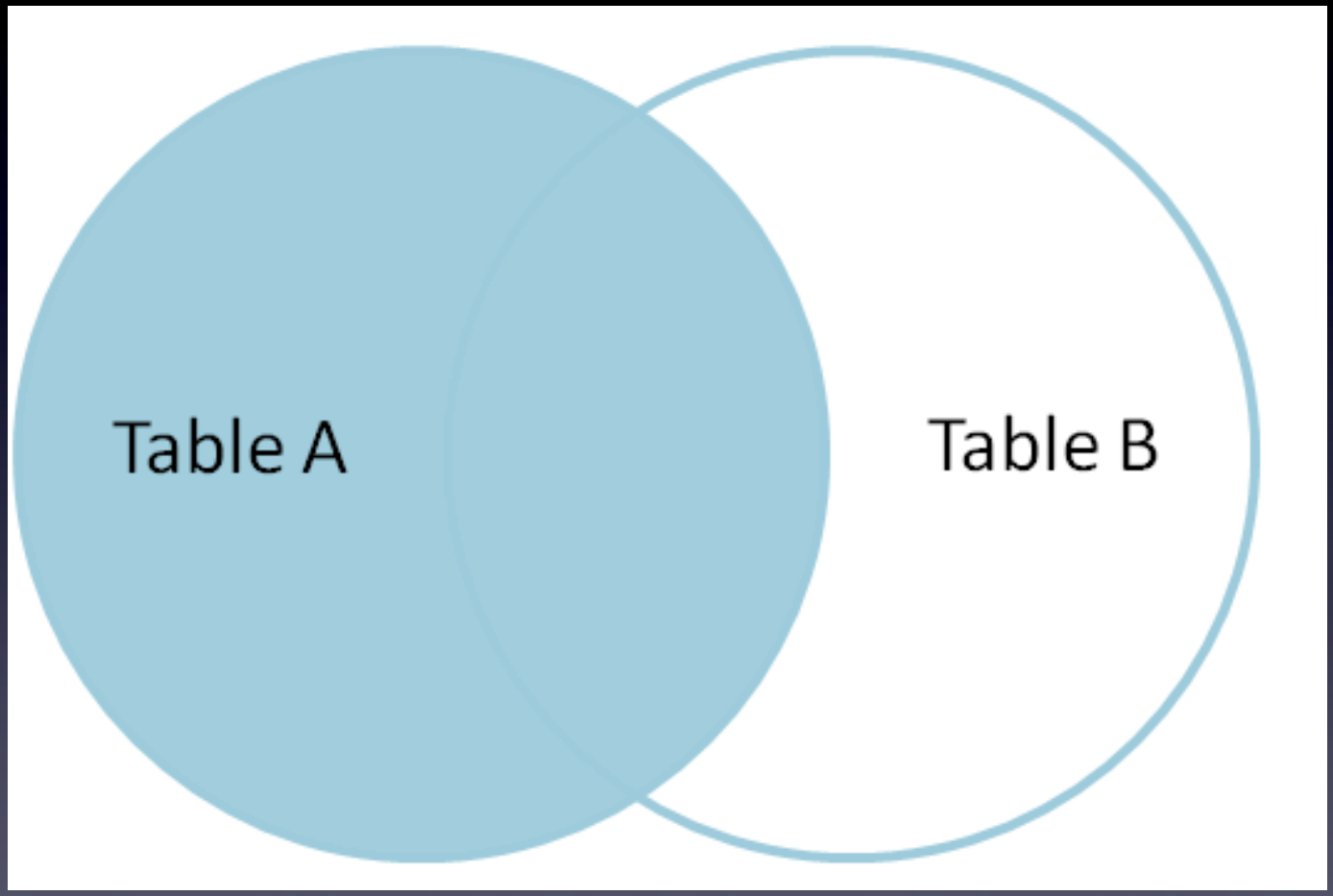
Inner or Natural Join (only intersection)

Join Datasets



FULL Outer Join

Join Datasets



Left Outer Join

Join = Merge

```
275  #Natural join: (only data from both tables)
276  aggdata3<-merge(x = aggdata, y = aggdata2, by = "yearID")
277  View(aggdata3)
278  #Full Outer join:
279  aggdata4<-merge(x = aggdata, y = aggdata2, by = "yearID", all = TRUE)
280  View(aggdata4)
281  #Left outer:
282  aggdata5<-merge(x = aggdata, y = aggdata2, by = "yearID", all.x = TRUE)
283  View(aggdata5)
284  #Right outer:
285  aggdata5<-merge(x = aggdata, y = aggdata2, by = "yearID", all.x = TRUE)
286  View(aggdata6)
```

for Loops (Recoding data)

There is a saying that if you are using for loops in R, you are doing something wrong. Why?

- There is probably a function to do what you need to do
- For loops will be much slower

However, for small/moderate size data they can be very effective and easy for beginning programmer to implement

for Loops (Recoding data)

The direct way

```
174  teams.1980$Season=ifelse(condition, "winner", "loser")
```

for Loop

```
181  #We can also do some processes with for loops.
182  #(Thought there is a saying that if you are using
183  #for loops in R, you are doing something wrong)
184  #Here we are going to go iterate through the data frame
185  N<-nrow(teams.1980)
186  for(i in 1:N){
187    if (teams.1980$W[i]>mean.W) {
188      teams.1980$Seasonb[i]="winner"
189    } else {
190      teams.1980$Seasonb[i]="loser"
191    }
192  }
193
```

for Loop (Reorganize Data)

This changes a data frame to a list:

```
194  #Lists can be useful.  Again, these are similar to JSON objects.
195  #Here we will take statistics on each team using summary and
196  #but the results into a list.
197  teams.names=unique(teams.1980.small$teamID)
198  teams.summary<-list()
199  for (t in teams.names){
200    these <-teams.1980.small$teamID == t
201    teams.summary[[t]]<-summary(teams.1980.small[these,])
202  }
203
204  #This retrieves the statistics for certain teams
205  teams$BAL
206  teams$ARI
207
```

Functions

Functions are a useful way to organize calculations

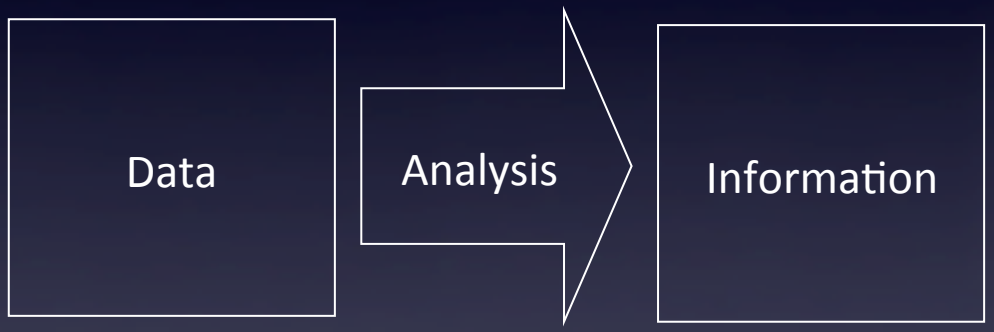
- Similar to a method in programming

Functions

```
296  #Functions
297  testfunction <- function(x) {
298      m  <- mean(x)
299      return(m)
300  }
301  #test function which returns the mean
302  testfunction (teams$W)
303  #This yields the same value
304  mean(teams$W)
```

Analysis

Review: The Role of Analysis

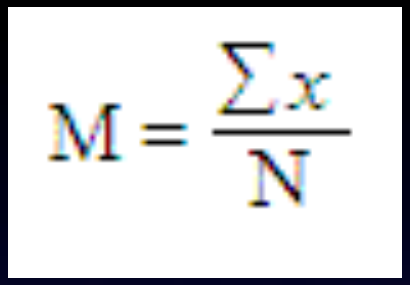


What are the steps to answering a business question?

- Clarify the question
- **Analysis**
- Identify alternative solutions
- Access alternative solutions
- Choose one solution
- Implement solutions
- Examine to what extent your solution answered

Basic Statistics

- Center
  - Mean



  - Median -The middle number or item in a set of numbers or objects arranged from least to greatest
- Spread
  - Variance/Standard Deviation
- Density curves/Distribution curve/Histogram
  - Center and spread

Review: Sample Variance s^2

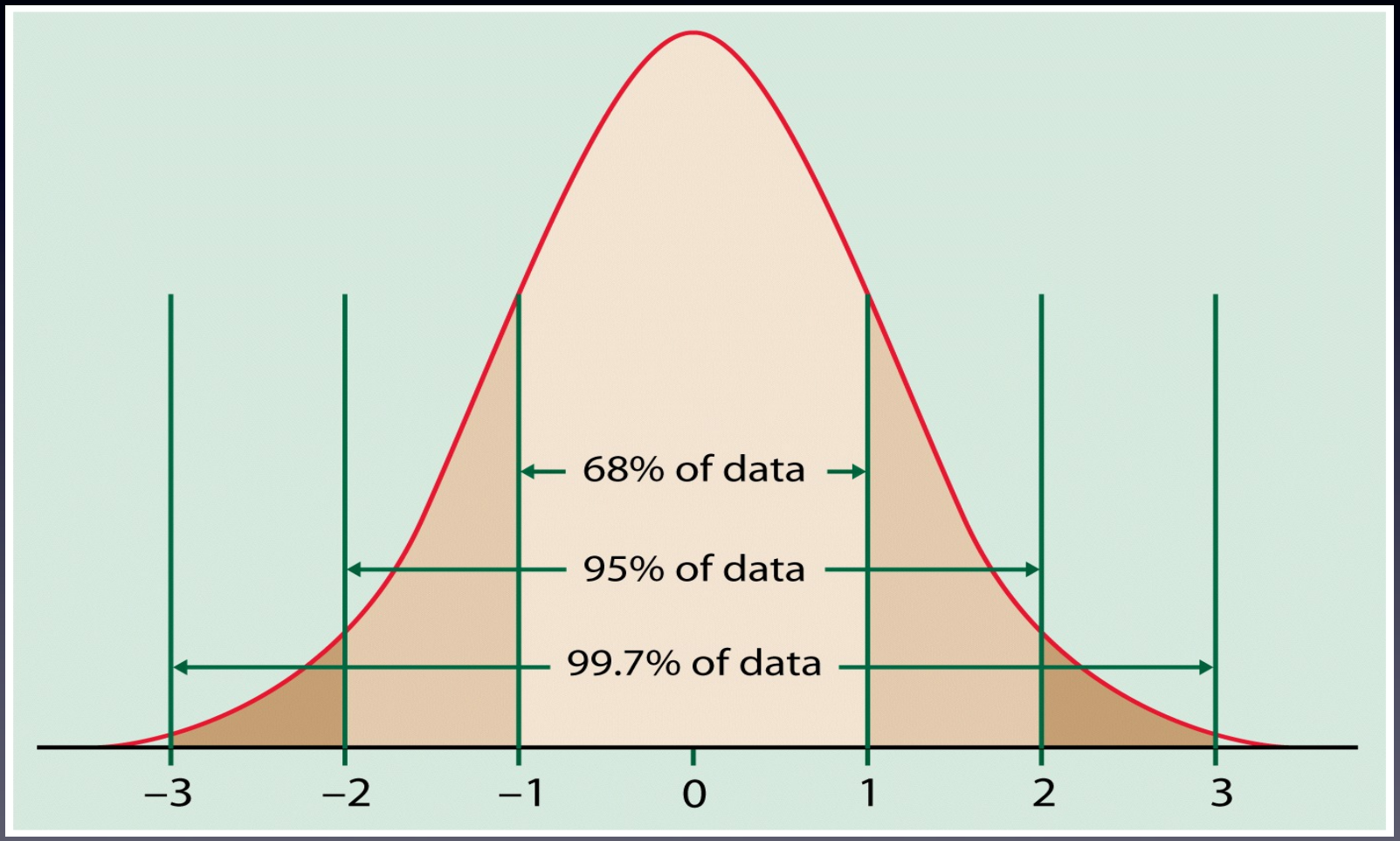
Deviation from mean: the difference between an observation and the sample mean:

Sample Variance s^2: the average of the squares of the deviations of the observations from their mean.

$$S^2 = \frac{\sum (X - \overline{X})^2}{n - 1}$$

Sample Standard Deviation s: the square root of the sample variance.

Review: 68-95-99.7 Rule w/Normal Distributions



Standardizing and z-Scores

an observation x comes from a distribution with mean μ and standard deviation σ

The standardized value of x is defined as

$$z = \frac{x - \mu}{\sigma},$$

which is also called a **z-score**.

A z-score indicates how many standard deviations the original observation is away from the mean, and in which direction.

Correlation

$$\frac{\sum z_X z_Y}{N-1} = r$$

It can also be defined as the average product of z-scores because the two equations are identical.

The correlation, *r*, is a quantitative index of the association between two variables. It is the average of the products of the *z*-scores.

When this average is positive, there is a positive correlation; when negative, a negative correlation

Causation

The only way to conclude that X->Y is:

X must precede Y

Y must not occur when X does not occur

Y must occur whenever X occurs

Difficult to fully test causation with statistical methods

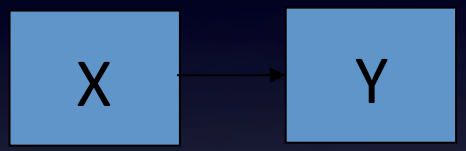
Causation

In social sciences research, you argue for causation based on your knowledge of the world, you support this through data analysis

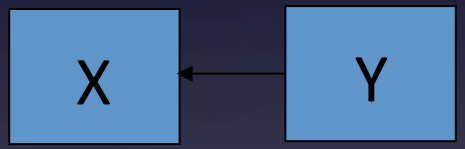
You **cannot** conclude causation through statistical analysis

Causation

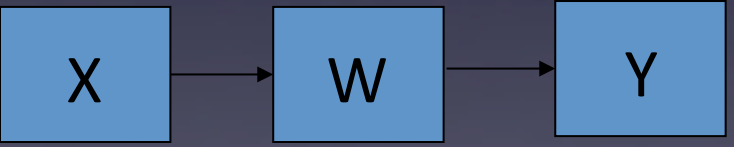
If X and Y are correlated…



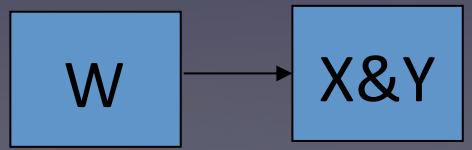
It could be that X causes Y.



It could be that Y causes X.



It could be that X causes W and W causes Y.



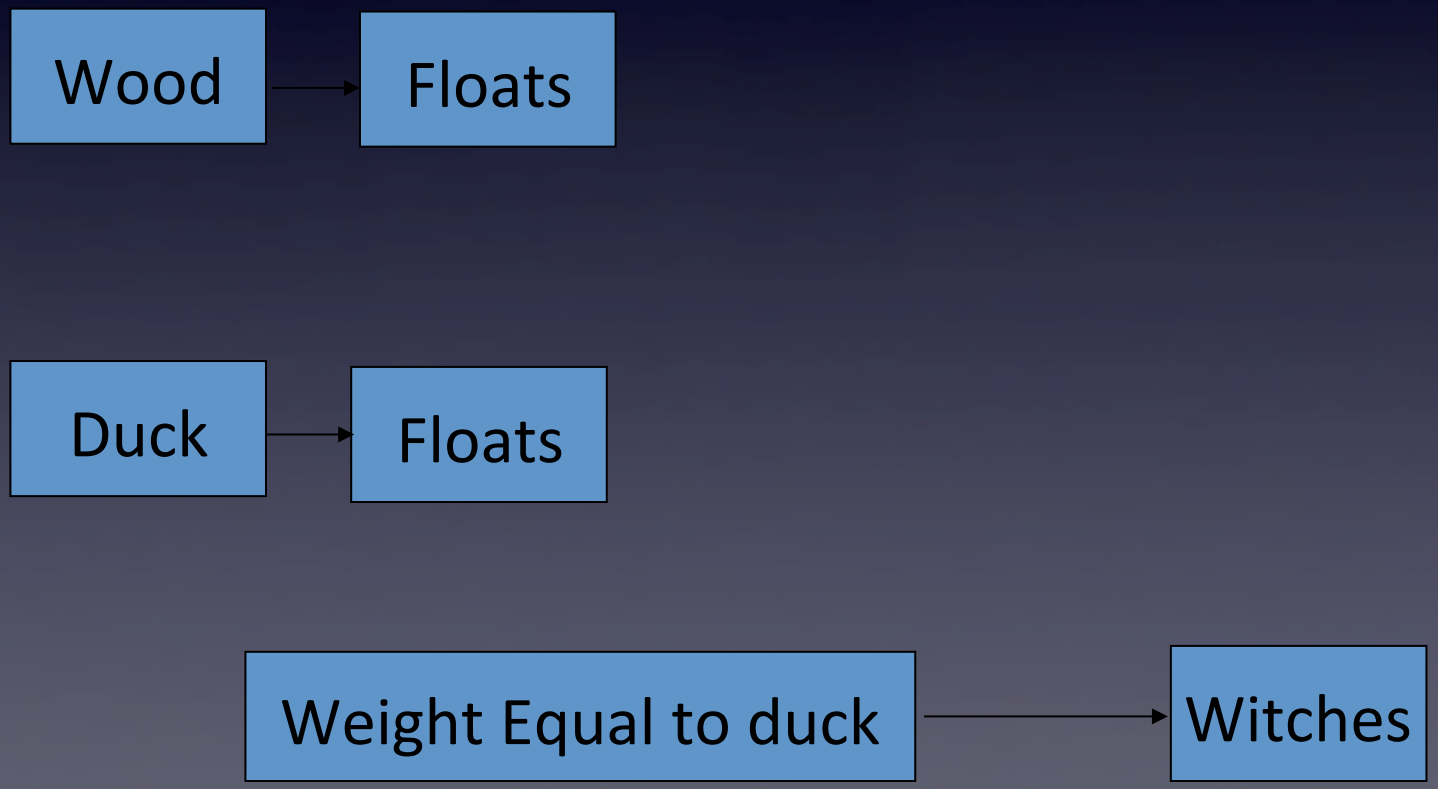
It could be that W causes X&Y.

Monty Python

http://www.youtube.com/watch?v=yp_l5ntikaU

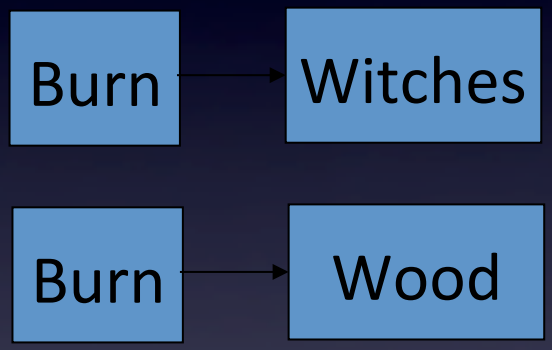
Causation

If you extend your logic analysis



Causation

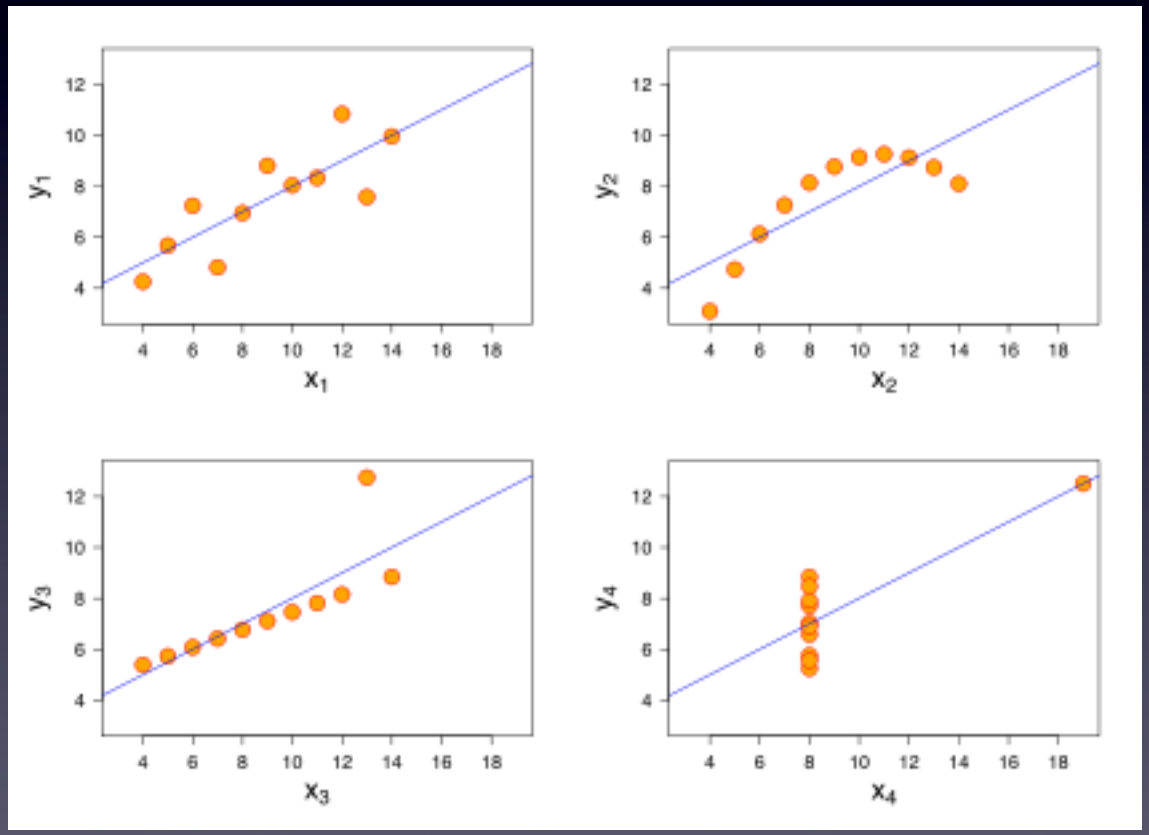
If you extend your logic analysis



Therefore

What do each of these graphs indicate?

All 4 sets for variables have the same mean, standard deviation, and correlation, and regression line



It is important to visualize data

Next Time

Regression