IST 3420: Introduction to Data Science and Management

Langtao Chen, Fall 2017

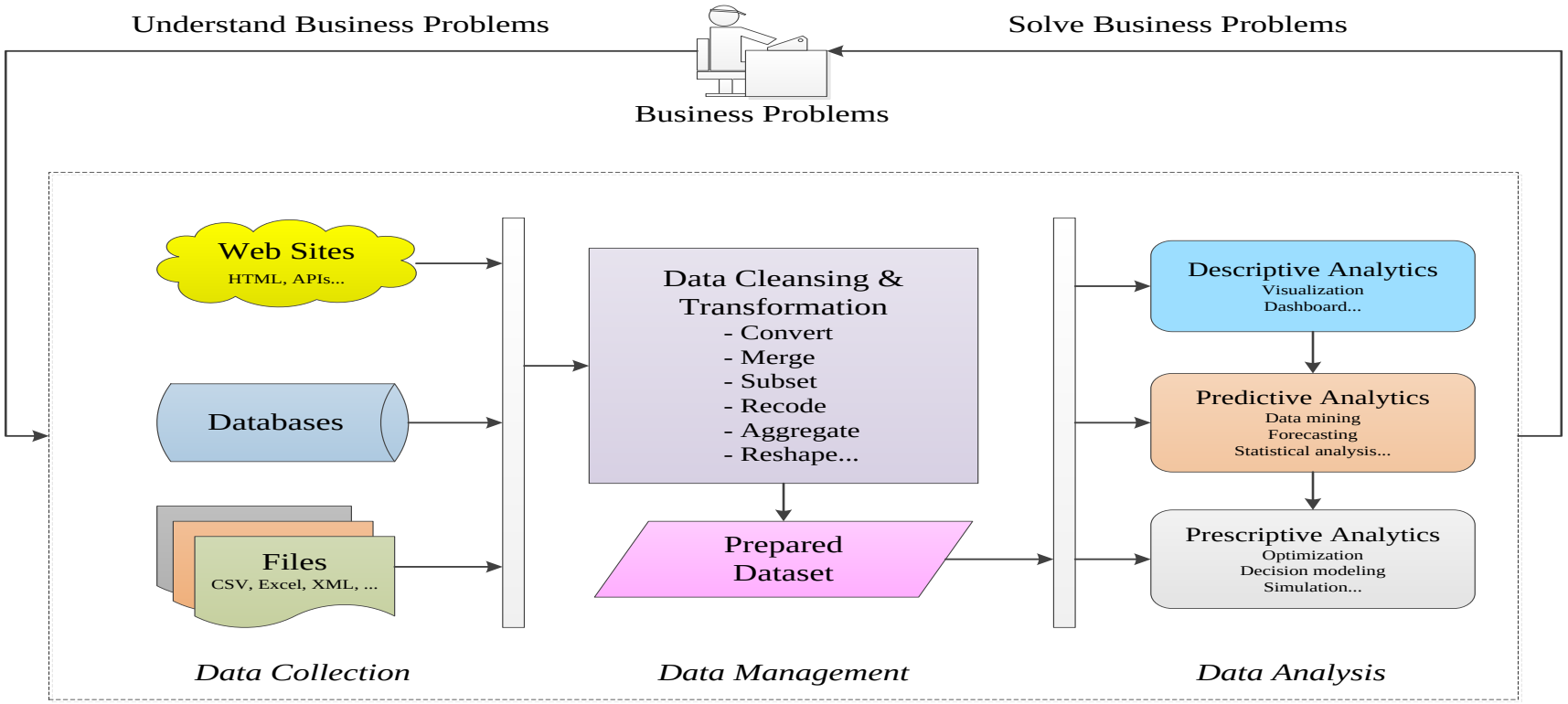
4. Cleansing and Manipulating Data

Reading Assignment 8 (due Sep 17)

- Read article "Introduction to String Matching and Modification in R Using Regular Expressions"
  - Download the article from Canvas

Data Management for Data Science

- Our goal is to get prepared datasets that are ready for in-depth data analysis.
- In R, the prepared datasets are usually data frames, which mimic the SAS or SPSS data set, i.e. a "cases by variables" matrix of data.



Why Data Management Matters?

- Data cleansing/transformation is an essential (usually the most time-consuming) part of a data analytics project.
- A properly prepared dataset is the prerequisite of statistical modeling, prediction, and inference.
- The "Garbage in, garbage out" rule applies.

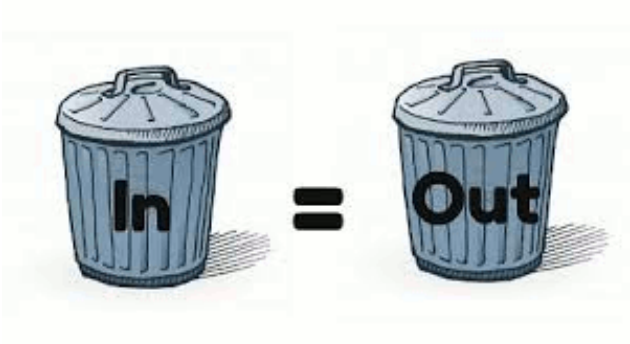


Image source: http://www.simplebi.com/wp-system1/uploads/2015/01/gigo.gif

Agenda

- Manipulate Strings
- Manipulate Datasets
  - Create, Recode, and Rename Variables
  - Convert
  - Sort
  - Subset
  - Merge
  - Aggregate
  - Recode
  - Reshape

Popular Data Manipulation Packages

- dplyr
  - https://github.com/hadley/dplyr
  - https://cran.r-project.org/web/packages/dplyr/dplyr.pdf
- tidyr
  - https://github.com/hadley/tidyr
  - https://cran.r-project.org/web/packages/tidyr/tidyr.pdf

Strings Manipulation

String Manipulation

- General
  - tolower(): translate to lower case
  - toupper(): translate to upper case
  - nchar(): count the number of characters
  - trimws(): trim whitespace [\t\r\n]
  - stringi::stri_reverse(): reverse a string
- Pattern matching and replacement
  - regexpr(): match patterns
  - grep(): match patterns
  - sub(): replace the first match
  - gsub(): replace all matches

(cont.)

- Substrings
  - substr(x, start, stop)
  - substring(text, first, last = 1000000L)
- Split Strings
  - strsplit(x, split, fixed = FALSE, perl = FALSE, useBytes = FALSE)
- Concatenate
  - paste()
  - paste0()

R Code: String Manipulation

```r
car <- rownames(mtcars)
tolower(car)  # Translate to lower case
toupper(car)  # Translate to upper case
nchar(car)  # Count the number of characters

# Trim whitespace [ \t\r\n] (space, horizontal tab, line feed, carriage
return)
trimws("\n  Hello R! \t \r \n   ")
# Reverse strings
stringi::stri_reverse("abcdef")

# Pattern matching and replacement
sub("a","1", "abcabcabc")  # replace the first match
gsub("a","1", "abcabcabc")  # replace all matches
gsub("[[:digit:]]","", "a1b2c3ef4gh55")  # Remove all digits

# Substrings
substr("abcdef", 2, 5)  # Syntax: substr(x, start, stop)
substring("abcdef", 2, 5)  # Syntax: substring(text, first, last =
1000000L)
substring("abcdef", 1, 1:6)
substring("abcdef", 1:6, 1:6)
```

(cont.)

```
# Split strings
strsplit("abcdef", NULL)  # Format: strsplit(x, split)
strsplit("a.b.c.d.e.f", "[.]")  # split argument is a regular expression
strsplit("a.b.c.d.e.f", ".", fixed = TRUE) # Use exact matching
strsplit("12 23   14 21 56 78 99  "," ")
strsplit("12 23   14 21 56 78 99  ","[[:blank:]]+") # Match blank one or
more times
unlist(strsplit("12 23   14 21 56 78 99  ","[[:blank:]]+"))  # To get a
vector

# Concatenate
paste("abc","123",sep = "")
paste("abc","123",sep = ",")
# paste0(...) is equivalent to paste(..., sep = ""), slightly more
efficiently.
paste0("abc","123")
```

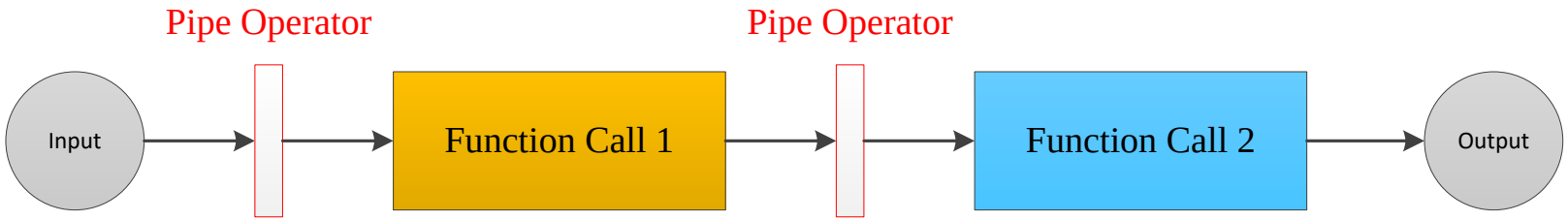
Notes

- To count the number of characters in a string, do NOT use length().
  - length() returns the length of the vector containing the string.
  - Instead, use nchar() function.
- To find the position of matches in a string, do NOT use grep().
  - grep() returns the index of matched string in a vector, NOT the position of the match in the text.
  - Instead, use regexpr() to get the position of the 1st match, and use gregexpr() to get positions of all matches.

Datasets Manipulation

Forward Pipe Operator magrittr::%>%

- Pipe an object forward into a function or call expression.



- Forward pipe operator makes R code more readable and elegant.

Syntax of magrittr::%>%

- Syntax: lhs %>% rhs
  - lhs: left-hand side, rhs: right-hand side
  - Work with unary function calls:

    x %>% f is equivalent to f(x)
  - Work with multi-argument function calls:

    x %>% f(y) is equivalent to f(x, y)

    y %>% f(x, ., z) is equivalent to f(x, y, z)

R Code: Forward Pipe Operator

```
####### Pipe Operator magrittr::%>% #######
# Load magrittr package
library(magrittr)

# Learn about pipe operator%>%
?magrittr::`%>%`

# The pipe operator syntax is "lhs %>% rhs"
speed <- cars$speed

# Find the max speed
speed %>% max

# List unique speed values
speed %>% unique

# List unique speed values and sort them by descending order
speed %>% unique %>% sort(decreasing = TRUE)

# List top 5 speed values and sort them by descending order
speed %>% unique %>% sort(decreasing = TRUE) %>% head(5)

# When the lhs is not the first argument in rhs call, we can use the dot place-holder
"Ceci n'est pas une pipe" %>% gsub("une", "un", .)
sample(1:10) %>% paste0(LETTERS[.])
```

Manipulate Datasets

- Create, Recode, and Rename Variables
- Convert
- Sort
- Subset
- Merge
- Aggregate
- Reshape

Create Variables

- Use index ($ operator)
- Use transform() function
- Use dplyr::mutate() function

R Code: Create Variables

```
patient <- data.frame(id = c("A01","A02","A03"),
                      first.name = c("Mike","Emily","Hannah"),
                      last.name = c("Smith","Johson", "Williams"),
                      age = c(26,20,24),
                      mass = c(150, 120, 110),
                      height = c(70,68,67))
patient

## Use Base R Feature ##
# Use index
patient2 <- patient # Copy the original dataset
# Add new variable full.name
patient2$full.name <- paste(patient2$first.name,patient2$last.name,sep = " ")
# Add new variable bmi
patient2$bmi <- 703*patient2$mass/(patient2$height^2)
patient2
# Use transform()
patient3 <- transform(patient,
                      full.name = paste(first.name,last.name,sep = " "),
                      bmi = 703*mass/(height^2))
patient3

## Use dplyr::mutate() ##
patient4 <- dplyr::mutate(patient,
                          full.name = paste(first.name,last.name,sep = " "),
                          bmi = 703*mass/(height^2))
patient4
```

Recode Variables

- Recode the values of variables usually involves applying conditions.

```
####### Recode Variables #######
patient4$bmi.category[patient4$bmi>=16 & patient4$bmi<18.5] <- "Underweight"
patient4$bmi.category[patient4$bmi>=18.5 & patient4$bmi<25] <- "Normal"

patient4$age.group = ifelse(patient4$age>=25,"Older","Younger")

patient4

# Recode the ID
patient4$id <- dplyr::recode(patient4$id,A01 = "B01",A02 = "B02",A03 = "B03")
# Recode the age
patient4$age <- dplyr::recode(patient4$age,`26` = 27L,`20` = 21L,`24` = 25L)

patient4
```

Rename Variables

```
####### Rename Variables #######

names(patient4)[names(patient4) == "bmi.category"] <- "health.status“

patient4 <- dplyr::rename(patient4,weight = mass)  # Rename mass as weight

patient4
```

Manipulate Datasets

- Create, Recode, and Rename Variables
- Convert
- Sort
- Subset
- Merge
- Aggregate
- Reshape

Data Conversion

- Basic Data Structure Conversion Functions

To Data Type

From Data Type	Long Vector	Matrix	Data Frame
Vector	c(x,y)	cbin(x,y), rbin(x,y)	data.frame(x,y)
Matrix	as.vector()		as.data.frame()
Data Frame		as.matrix()	

Source: http://www.statmethods.net/management/typeconversion.html

Data Conversion

- Convert factor to numeric: as.numeric(as.character(x))

```
# Read the dataset we collected from https://nrf.com/2015/top100-table
df <- read.csv("top100retailers2015.csv")
# Show the structure of the dataset
str(df)  # All variables except the first two are factor data (i.e., nominal scale)
summary(df)  # Cannot get summary statistics such as min, max etc. for nominal data

df$RetailSales2014 <- df$RetailSales2014 %>%
  gsub(",","",.) %>%
  gsub("$","",.,fixed = TRUE) %>%
  as.character() %>%
  as.numeric()

# Show the structure of the new variable
str(df$RetailSales2014)
# Get summary statistics of the new variable
summary(df$RetailSales2014)
```

Manipulate Datasets

- Create, Recode, and Rename Variables
- Convert
- Sort
- Subset
- Merge
- Aggregate
- Reshape

Sort

- To sort data frames into ascending or descending order along one or more variables
- Why is sorting important?

If I care much about mpg, which cars should I choose?

	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

Sorting Methods

- Two general methods
  - Use order() function in base R
  - Use arrange() function in dplyr package

R Code: Sort Data

```
## Use base R features

# Sort by mpg (ascending)
mtcars[order(mtcars$mpg),]
# Sort by cyl (ascending) and hp (ascending)
mtcars[order(mtcars$cyl, mtcars$hp),]
# Sort by cyl (ascending) and hp (descending)
mtcars[order(mtcars$cyl, -mtcars$hp),]

## Use arrange() in dplyr package

# Sort by mpg (ascending)
dplyr::arrange(mtcars,mpg)
# Sort by cyl (ascending) and hp (ascending)
dplyr::arrange(mtcars,cyl,hp)
# Sort by cyl (ascending) and hp (descending)
dplyr::arrange(mtcars,cyl,desc(hp))
```

Manipulate Datasets

- Create, Recode, and Rename Variables
- Convert
- Sort
- Subset
- Merge
- Aggregate
- Reshape

Subset

- Subset variables
- Subset observations

Subset Methods

- Use base R features
  - Use index
  - Use which() function
  - Use subset() function
- Use dplyr package
  - Use dplyr::select() to select variables
  - Use dplyr::filter() to select observations

R Code: Use Base R Index Feature

```
data(mtcars)  # Load the built-in dataset mtcars
str(mtcars)  # Show the structure

# Use index to select variables
car1 <- mtcars[c("mpg","wt")]  # Select miles per gallon and weight
names(car1)
car2 <- mtcars[c(1,6)]  # Select 1st and 6th variables, i.e. mpg and wt
names(car2)
car3 <- mtcars[c(1:5,10)]  # Select 1st through 5th and 10th variables
names(car3)
car4 <- mtcars[c(-3,-5)]  # Exclude the 3rd and 5th variables
names(car4)
# Use index and which() to select observations
car5 <- mtcars[1:4,]  # First 4 observations
car6 <- mtcars[which(mtcars$mpg >= 25),]  # Select cars whose mpg >= 25
car6
car7 <- mtcars[which(mtcars$mpg >= 25 & mtcars$gear == 4),]  # Select
cars whose mpg >= 25 and gear == 4
car7
# Use index and which() to select both observations and variables
car8 <- mtcars[which(mtcars$mpg >= 25),c("mpg", "wt")]  # Select mpg
and wt variables with mpg >= 25
car8
```

R Code: Use Base R subset() Function

```
# Use subset() function to select both observations and variables
car11 <- subset(mtcars, select = c("mpg","wt"))  # Select mpg and
wt variables
names(car11)
car12 <- subset(mtcars, select = c(1,6))
names(car12)
car13 <- subset(mtcars, select = mpg:wt)  # Select all variables
between mpg and wt
names(car13)

car14 <- subset(mtcars,mpg >= 25,select = c(mpg,wt)) # Select mpg
and wt variables with mpg >= 25
car14
car15 <- subset(mtcars,cyl == 6,select = c(1:6)) # Select 1st
through 6th variables with cyl == 6
car15
```

R Code: Use dplyr Package

```r
## Use dplyr package
library(dplyr)

# Use dplyr::select() to select variables
car21 <- select(mtcars,c(mpg,wt))  # Select mpg and wt
variables
names(car21)
car22 <- select(mtcars,-c(mpg,wt))  # Select variables except
mpg and wt
names(car22)
car23 <- select(mtcars,contains("p"))  # Select variables
containing "p"
names(car23)

# Use dplyr::filter() to select observations
filter(mtcars,mpg >= 25)  # Select cars with mpg >= 25
filter(mtcars,mpg >= 25 & gear == 4)  # Select cars with mpg >=
25 and gear == 4

# Combine dplyr::select() and dplyr::filter()
select(filter(mtcars,mpg >= 25),c(mpg,wt))  # Select mpg and wt
variables with mpg >= 25
filter(select(mtcars,c(mpg,wt)),mpg >= 25)  # Another way
```

Manipulate Datasets

- Create, Recode, and Rename Variables
- Convert
- Sort
- Subset
- Merge
- Aggregate
- Reshape

Merge/Join Data Frames

- 4 Major Types of Merging/Joining x and y
  - Inner join:
    - Only rows with matching keys in both x and y
  - Left outer join
    - All rows in x, adding matching columns from y
  - Right outer join
    - All rows in y, adding matching columns from x
  - Full outer join
    - All rows in x with matching columns in y, then the rows of y that don't match x.

Examples

Enrollment

CourseID	StudentID
IST3420	201
IST3420	202
IST3420	501

Enrollment.StudentID = Student.StudentID

Student

StudentID	Name	Department
201	Mike	IST
202	Emily	BUS
203	Hannah	IST

Inner Join

CourseID	StudentID	Name	Department
IST3420	201	Mike	IST
IST3420	202	Emily	BUS

Left Outer Join

CourseID	StudentID	Name	Department
IST3420	201	Mike	IST
IST3420	202	Emily	BUS
IST3420	501		

Right Outer Join

CourseID	StudentID	Name	Department
IST3420	201	Mike	IST
IST3420	202	Emily	BUS
	203	Hannah	IST

Full Outer Join

CourseID	StudentID	Name	Department
IST3420	201	Mike	IST
IST3420	202	Emily	BUS
IST3420	501		
	203	Hannah	IST

Merge/Join Methods

- Use merge() function
- Use join() function in plyr package
- Use dplyr package
  - inner_join()
  - left_join()
  - right_join()
  - full_join()

R Code: Use merge() Function

```r
enrollment <- data.frame(courseId = "IST3420",
                         studentId = c("201","202","501"))

student <- data.frame(studentId = c("201","202","203"),
                      name = c("Mike","Emily","Hannah"),
                      department = c("IST","BUS","IST"))

print(enrollment)
print(student)

# Inner join
merge(x = enrollment, y = student, by = "studentId")

# Left outer join
merge(x = enrollment, y = student, by = "studentId", all.x = TRUE)

# Right outer join
merge(x = enrollment, y = student, by = "studentId", all.y = TRUE)

# Full outer join
merge(x = enrollment, y = student, by = "studentId", all = TRUE)
```

R Code: Use join() Function in plyr Package

```
# install.packages("plyr")
library("plyr")

print(enrollment)
print(student)

# Inner join
join(enrollment,student,by = "studentId",
type = "inner")

# Left outer join
join(enrollment,student,by = "studentId",
type = "left")

# Right outer join
join(enrollment,student,by = "studentId",
type = "right")

# Full outer join
join(enrollment,student,by = "studentId",
type = "full")
```

```
> # Inner join
> join(enrollment,student,by = "studentId", type = "inner")
  courseId studentId  name department
1  IST3420       201  Mike        IST
2  IST3420       202 Emily        BUS
>
> # Left outer join
> join(enrollment,student,by = "studentId", type = "left")
  courseId studentId  name department
1  IST3420       201  Mike        IST
2  IST3420       202 Emily        BUS
3  IST3420       501  <NA>       <NA>
>
> # Right outer join
> join(enrollment,student,by = "studentId", type = "right")
  studentId courseId   name department
1       201  IST3420   Mike        IST
2       202  IST3420  Emily        BUS
3       203     <NA> Hannah        IST
>
> # Full outer join
> join(enrollment,student,by = "studentId", type = "full")
  courseId studentId   name department
1  IST3420       201   Mike        IST
2  IST3420       202  Emily        BUS
3  IST3420       501   <NA>       <NA>
4     <NA>       203 Hannah        IST
```

R Code: Use dplyr Package

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

# Inner join
inner_join(enrollment,student,by = "studentId") # This does not work

# Show the structure of data frames
str(enrollment)
str(student)
# Change vectors from factor type to character type
enrollment$courseId <- as.character(enrollment$courseId)
enrollment$studentId <- as.character(enrollment$studentId)
student$studentId <- as.character(student$studentId)
student$name <- as.character(student$name)
student$department <- as.character(student$department)
str(enrollment)
str(student)

inner_join(enrollment,student,by = "studentId") # Now it works

# Left outer join
left_join(enrollment,student,by = "studentId")

# Right outer join
right_join(enrollment,student,by = "studentId")

# Full outer join
full_join(enrollment,student,by = "studentId")
```

Manipulate Datasets

- Create, Recode, and Rename Variables
- Convert
- Sort
- Subset
- Merge
- Aggregate
- Reshape

Aggregate

- Base R
  - aggregate(): group data and calculate summary statistics
- dplyr Package
  - group_by(): group data
  - summarize(): calculate summary statistics

R Code: Aggregate

```
####### Aggregate #######
library(dplyr)

attach(mtcars)
mtcars %>% head

# Use aggregate() function
mtcars %>% aggregate(by=list(cyl,vs),FUN=mean, na.rm=TRUE)

# Use dplyr::group_by() and dplyr::summarize() functions
mtcars %>% group_by(cyl,vs) %>%
summarize(mean(mpg),mean(disp),mean(hp),mean(drat),mean(wt),mean(qsec),mean(am),
mean(gear),mean(carb))

# Aggregate by cyl and vs, show mean of mpg for each group
mtcars %>% group_by(cyl,vs) %>% summarize(mean(mpg))

# Aggregate by cyl and vs, show maximum mpg for each group
mtcars %>% group_by(cyl,vs) %>% summarize(max(mpg))

detach(mtcars)
```

Manipulate Datasets

- Create, Recode, and Rename Variables
- Convert
- Sort
- Subset
- Merge
- Aggregate
- Reshape

Reshape: Change the Layout of a Data Set

- Use tidyr Package
  - gather(data, key, value, …): to gather columns into rows
  - spread(data, key, value): to spread rows into columns

wide format

	name	IST3420	IST1750	IST3423
1	Mike	80	78	90
2	Sarah	85	81	93
3	Helen	90	91	88

gather →

← spread

long format

	name	course	score
1	Mike	IST3420	80
2	Sarah	IST3420	85
3	Helen	IST3420	90
4	Mike	IST1750	78
5	Sarah	IST1750	81
6	Helen	IST1750	91
7	Mike	IST3423	90
8	Sarah	IST3423	93
9	Helen	IST3423	88

R Code: Reshape

```
####### Reshape #######
library(tidyr)

# A student score dataset in wide format
score_wide <- data.frame(
  name = c("Mike","Sarah","Helen"),
  IST3420 = c(80,85,90),
  IST1750 = c(78,81,91),
  IST3423 = c(90,93,88)
)
print(score_wide)

## Gather columns into key-value pairs
score_long <- score_wide %>% gather(course,score,IST3420, IST1750,IST3423)
print(score_long)
# Another way to specify columns to gather
score_wide %>% gather(course,score,c(IST3420, IST1750,IST3423))
# Still another way to specify columns to gather
score_wide %>% gather(course,score,-name)

## Split a single character column into multiple columns
print(score_long)

score_long %>% spread(course,score)
```

Reference

- Data Transformation with dplyr Cheatsheet
  - https://github.com/rstudio/cheatsheets/raw/master/source/pdfs/data-transformation-cheatsheet.pdf

Data Manipulation in a Project

Weblog Analytics

Business Scenario

- Company XYZ runs a website. The company is confronting the challenge of extracting information and knowledge from this website to better understand how visitors access the online service.
- The objective of this data analytics project is to answer the following questions:
  - What are the top 10 countries from which the visitors come?
  - How many visits do we have for the FAQ page?
  - More……

Apache Web Server Log

- The log file records all HTTP requests from browser clients to web server.
  - Remote host
  - Request time
  - Request method
  - Request URI
  - Request protocol
  - Status
  - Size of request

sample_apache.log

```
1  65.55.147.227 - - [15/Oct/2009:02:00:24 +0000] "GET /index.html HTTP/1.1" 200 21878
2  65.55.86.34 - - [15/Oct/2009:02:00:58 +0000] "GET /index.html HTTP/1.1" 200 1416
3  148.188.55.88 - - [15/Oct/2009:02:01:41 +0000] "GET /faq.html HTTP/1.1" 200 10946
4  72.30.57.238 - - [15/Oct/2009:02:01:59 +0000] "GET /contribute.txt HTTP/1.0" 200 39943
5  66.249.139.233 - - [15/Oct/2009:02:02:09 +0000] "GET /faq.html HTTP/1.1" 200 17247
6  72.30.50.248 - - [15/Oct/2009:02:02:13 +0000] "GET /index.html HTTP/1.0" 200 7883
7  216.129.13.4 - - [15/Oct/2009:02:02:37 +0000] "GET /contribute.txt HTTP/1.0" 200 18119
8  66.249.61.232 - - [15/Oct/2009:02:02:39 +0000] "GET /contribute.txt HTTP/1.1" 200 10946
9  65.55.80.97 - - [15/Oct/2009:02:02:51 +0000] "GET /index.html HTTP/1.1" 200 1416
```

Data Management Task

- Import the Apache log file
- Cleanse and transform the data into the following data frame

Filter

	remote.host	status	response.size	request.date	request.method	request.uri	request.protocol
1	65.55.147.227	200	21878	2009-10-15	GET	index.html	HTTP/1.1
2	65.55.86.34	200	1416	2009-10-15	GET	index.html	HTTP/1.1
3	148.188.55.88	200	10946	2009-10-15	GET	faq.html	HTTP/1.1
4	72.30.57.238	200	39943	2009-10-15	GET	contribute.txt	HTTP/1.0
5	66.249.139.233	200	17247	2009-10-15	GET	faq.html	HTTP/1.1
6	72.30.50.248	200	7883	2009-10-15	GET	index.html	HTTP/1.0
7	216.129.13.4	200	18119	2009-10-15	GET	contribute.txt	HTTP/1.0
8	66.249.61.232	200	10946	2009-10-15	GET	contribute.txt	HTTP/1.1
9	65.55.80.97	200	1416	2009-10-15	GET	index.html	HTTP/1.1
10	65.55.161.41	200	37122	2009-10-15	GET	index.html	HTTP/1.1
11	65.55.119.204	200	64380	2009-10-15	GET	faq.html	HTTP/1.1

Data Management Task

- Convert IP address to IP number
  - Refer to http://www.ip2country.net/ip2country/ip_number.html
- Look up GEO information of an IP address
  - Use the "GeoLite Country" database provided by http://www.maxmind.com

R Code

- Refer to “Manage_Weblog_Data.Rmd”

Web Analytics Summary

- Common reasons to use web analytics
  - Understand website traffic
  - Track mass user activity
  - Improve site design and user experience
- Two common data sources
  - **Web server log files** (e.g., Apache log file)


```
1  65.55.147.227 - - [15/Oct/2009:02:00:24 +0000] "GET /index.html HTTP/1.1" 200 21878
2  65.55.86.34 - - [15/Oct/2009:02:00:58 +0000] "GET /index.html HTTP/1.1" 200 1416
3  148.188.55.88 - - [15/Oct/2009:02:01:41 +0000] "GET /faq.html HTTP/1.1" 200 10946
4  72.30.57.238 - - [15/Oct/2009:02:01:59 +0000] "GET /contribute.txt HTTP/1.0" 200 39943
5  66.249.139.233 - - [15/Oct/2009:02:02:09 +0000] "GET /faq.html HTTP/1.1" 200 17247
```

  - **Page tagging or "Web bugs"** (e.g., analytics.js)

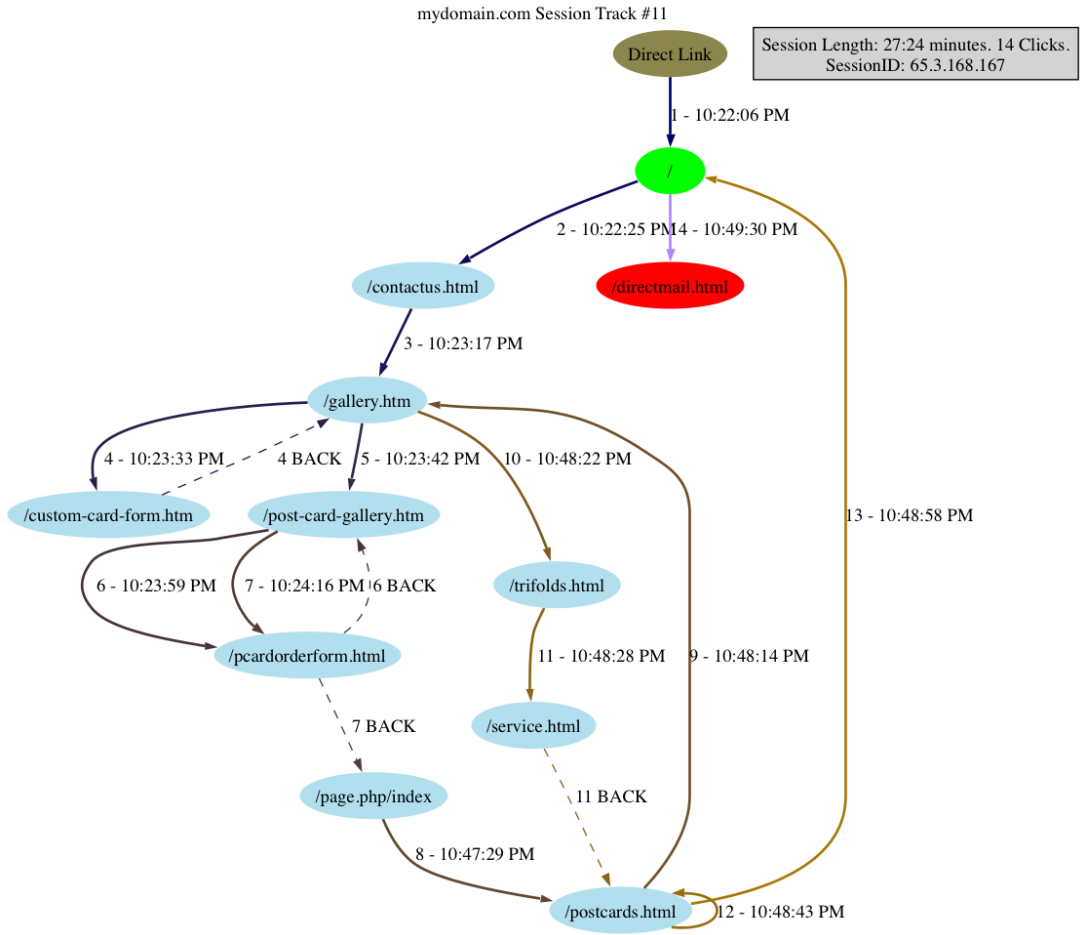
```
<!-- Google Analytics -->
<script>
(function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
(i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
})(window,document,'script','//www.google-analytics.com/analytics.js','ga');

ga('create', 'UA-XXXXX-Y', 'auto');
ga('send', 'pageview');
</script>
<!-- End Google Analytics -->
```

Web Analytics - Clickstream

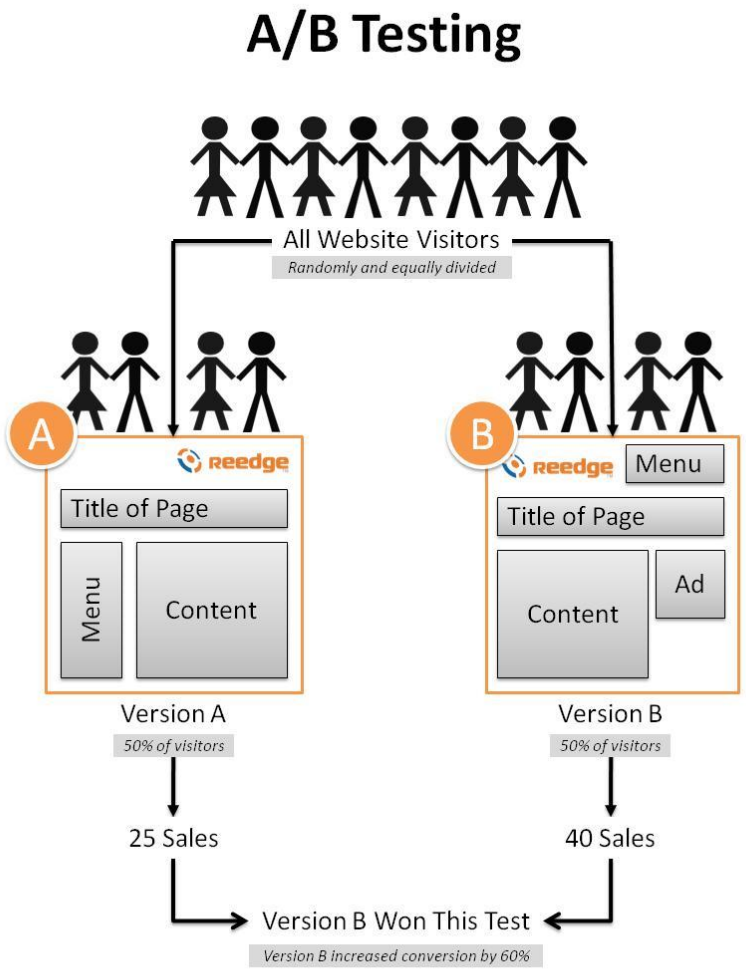
Clickstream analysis focused on a series of page requests

- Optimize click-path
- Understand consumer behavior
- Segment customers
- Allocate website resources
- Enhance user experience
- Reduce bounce rates
- Increase conversions



Pic source: http://statviz.sourceforge.net/statviz-session-track.png

Web Analytics - Field Experiment



https://receiptful.com/blog/ab-testing-for-ecommerce/

Q & A