

# WHAT IS R?

- ▶ R is a freely-available statistical programming software used by industry professionals and academics alike.
- ▶ It is a programming language, in particular a scripting language, with similarities to Matlab or Python.
- ▶ R is supported by a community of users and is actively being developed.
- ▶ R has a variety of built-in as well as community-provided packages that extend its functionality with code and data; see CRAN (Comprehensive R Archive Network) for the thousands of add-on packages.

Will use R extensively in this class

- ▶ Download R at: <https://www.r-project.org>
- ▶ Download RStudio at: <https://www.rstudio.com>

# DESIGN OF THE R SYSTEM

R is divided into two conceptual parts:

- ▶ The “base” R system that you download from CRAN.
- ▶ Everything else.

R functionality is divided into packages:

- ▶ The “base” R system contains, among other things, the **base** package. This is required to run R and contains the fundamental functions.
- ▶ Other packages like **stats**, **graphics**, **util**, **datasets**, are also contained in the “base” system.
- ▶ A fresh installation of R gets you all of these.
- ▶ However, there are over 4000 other packages on CRAN developed by users and programmers around the world.
- ▶ There are even more packages on individuals’ personal websites or repositories like GitHub.

# WHY USE R?

- ▶ R has the best statistical functionality of any software and is used most widely by statisticians and those practicing statistics.
- ▶ Wide usage helps to improve software quality and reduce bugs.
- ▶ The R community is constantly adding functionality via new packages.
- ▶ R is freely-available for any standard operating system.
- ▶ As a scripting language, R is very powerful, flexible, and easy to use, allowing for reproducibility and task automation.
- ▶ As a language, R can do essentially anything
- ▶ R can interact with other software, databases, the operating system, the web, etc.

# WHY USE SOMETHING OTHER THAN R?

- ▶ Other software is better than R at various tasks.
  - ▶ For example, Python is very good for text manipulation, interacting with the operating system, and as a glue for tying together various applications/software in a workflow.
- ▶ R can be much slower than compiled languages like C++ (but is often quite fast with good coding practices!).
- ▶ R's packages are only as good as the person who wrote them; no explicit quality control.
- ▶ R is a sprawling and unstandardized ecosystem.

# HOW DO WE USE R?

## Modes of Using R:

- ▶ Using the RStudio GUI (our method). Calls itself an ‘integrated development environment’.
- ▶ From the command line in a Linux/Mac terminal window.
- ▶ Running an Rscript in the background on a Linux/Mac machine.
- ▶ Using the Windows/Mac GUIs.

# USING R AND RSTUDIO

- ▶ The *editor* allows you to type and save code that you may want to reuse later. (We will use *scripts* and *Markdown files*.)
- ▶ Basic interaction with R happens in the *console*. This is where you type R code.

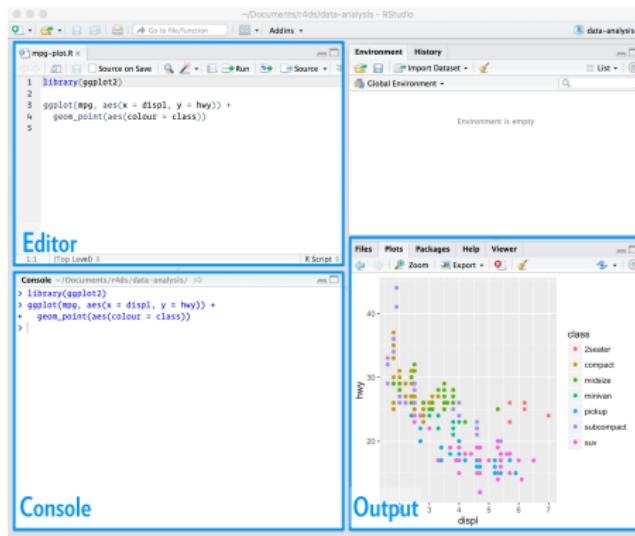


Figure 1 : Image of RStudio from G. Grolemund and H. Wickham

# USING R AND RSTUDIO

- ▶ The **output** is where you can find help files and the graphic window.
- ▶ The top right corner is the workspace which contains your objects holding data and information.

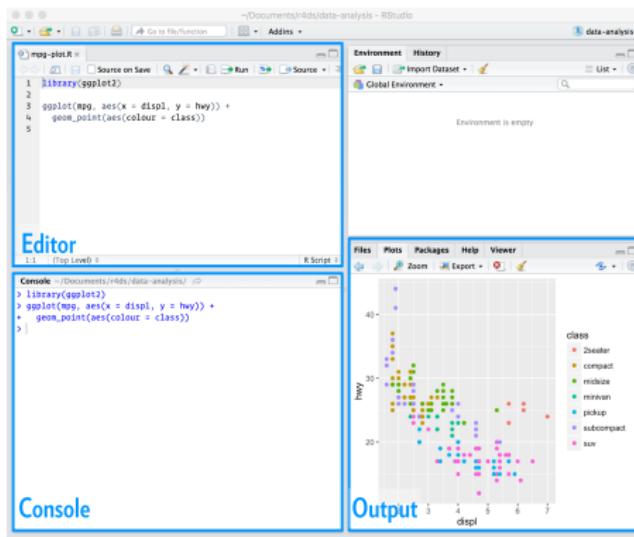


Figure 2 : Image of RStudio from G. Grolemund and H. Wickham

# USING R AND RSTUDIO

- ▶ RStudio provides an ‘integrated development environment in which all of these pieces are in a single application and tightly integrated, with a built-in editor for your code/scripts.
- ▶ If you’d like more information on the other functions and features of RStudio check out the short video tutorial on the Canvas page.

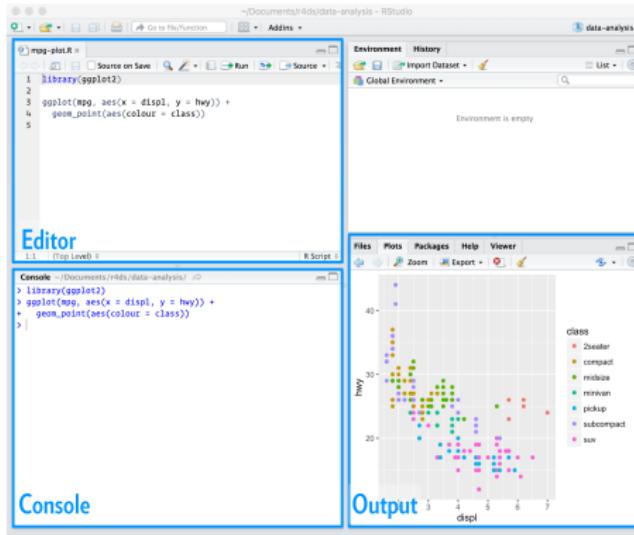


Figure 3 : Image of RStudio from G. Grolemund and H. Wickham

# USING R AND RSTUDIO

Code example.

# RSTUDIO AND RMARKDOWN

## RMarkdown

- ▶ RMarkdown is an extension to the Markdown markup language that makes it easy to write HTML in a simple plain text format.
- ▶ We present our work (homework and labs) using RMarkdown documents with embedded R code.
- ▶ This allows us to both run the R code directly as well as compile on-the-fly to an HTML file that can be used for presentation.

# STEPS TO CREATE AN RMarkdown DOCUMENT

To create a new RMarkdown document open RStudio and go to **File** -> **New File** -> **R Markdown**.

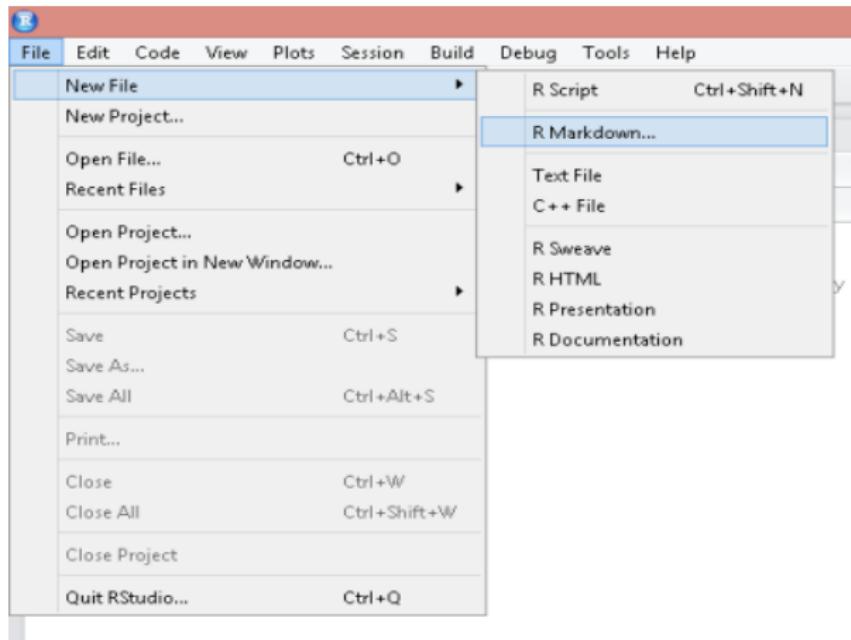


Figure 4 : Image from <https://www.r-bloggers.com>

# STEPS TO CREATE AN RMarkdown DOCUMENT

Enter the title of your document and the author and hit OK.

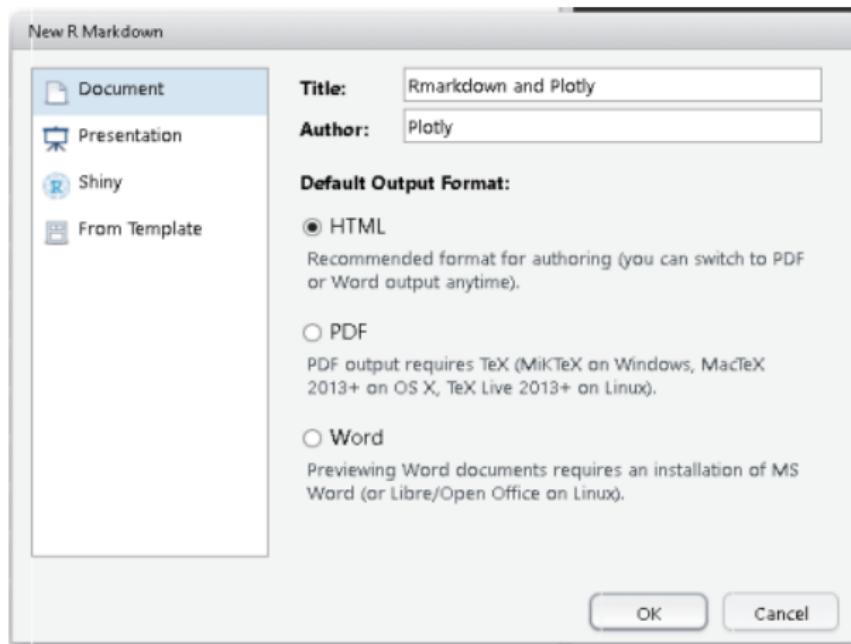


Figure 5 : Image from <https://www.r-bloggers.com>

# STEPS TO CREATE AN RMarkdown DOCUMENT

Clicking the `Knit HTML` button in the editor window generates the document.

## Editing the Markdown Document

- ▶ Writing equations uses LaTex code. So, for example, `$x^2$` produces  $x^2$ .
- ▶ Insert R code directly into the document using the following format:

```
```{r}
x <- rnorm(100)
```
```

- ▶ If you need help, go to `Help -> Markdown Quick Reference`.
- ▶ You'll get practice with this in lab.

# STEPS TO CREATE AN RMARKDOWN DOCUMENT

Code example.

## A QUICK EXAMPLE...

Type the following into your console:

```
> # Assign the value "x" to be 24^2
> x <- 24^2
> x
```

```
| [1] 576
```

Some important ideas:

1. Calculation
2. Commenting
3. Assignment

# R IS A CALCULATOR

Type the following into your console:

```
2 + 2          # add numbers
2 * pi         # multiply by a constant
7 + runif(1)   # add a random number
3^4            # powers
sqrt(4^4)      # functions
log(10)

5000000000 * 1000 # scientific notation
5e9 * 1e3
```

Think of a mathematical operation - can you guess how to do it in R?

# COMMENTING YOUR CODE

- ▶ Anything after the `#` isn't evaluated by R.
- ▶ Used to leave notes for humans reading your code.
- ▶ Very important in our class. Comment your code!

# ASSIGNING VALUES TO R OBJECTS

A key action in R is to store values in the form of R objects, and to examine the value of R objects.

Type the following into your console:

```
> # Assign the value "x" to be 24^2
> x <- 24^2
> x           # Auto-printing the result
```

```
| [1] 576
```

```
| > print(x)      # Explicit printing
```

```
| [1] 576
```

- ▶ The `<-` symbol means assign `x` the value  $24^2$ .
- ▶ Could use `=` instead of `<-` but this is discouraged.
- ▶ All assignments take the same form: `object_name <- value`.
- ▶ Type `x` into the console to print its assignment.

## A QUICK EXAMPLE...

Type the following into your console:

```
> # Assign the value "x" to be 24^2
> x <- 24^2
> x
```

```
[1] 576
```

```
> x <- "Hello"      c mainly means the vector and R will automatically
> print(c(x, x))  ensure the all the elements in the vector are the same
                   type.
```

```
[1] "576"    "Hello"
```

- ▶ Note that if we don't store the output of a command as an object, we haven't saved it for later use.
- ▶ R gives us lots of flexibility in assigning to objects other objects.

## A QUICK EXAMPLE...

Type the following into your console:

```
> # Assign the value "x" to be 24^2
> x <- 24^2
> x
```

```
[1] 576
```

### Note

The [1] tells us that 576 is the first element of the vector 'x'.

```
> # Create a vector in R named "x"
> x <- 1:50
> x
```

```
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
[19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
[37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

# TRY ON YOUR OWN

## Tasks:

1. Open a new RMarkdown file.
2. Save the file in a folder on your computer named for this class.
3. Knit the file.
4. Edit the file to have only three things:
  - ▶ The sentence ‘This is my first Markdown!’. (This should be the only text other than the heading – remove the other stuff).
  - ▶ R code that assigns to an object named `this_year` the value 2017.
  - ▶ R code that prints out the value of `this_year`.
5. Close RStudio and then re-open your saved Markdown file.
6. Knit to PDF instead of knitting to HTML. HINT: Click on the black arrow next to the knit button.

# OBJECT CLASSES, VECTORS, & MATRICES

# OBJECT CLASSES

R has a variety of object classes.

## Classes

1. Numeric (3.7, 15907, 80.333), i.e. double precision real numbers
2. Integer (25L, 59843L, -2L)
3. Complex (1 + 2i)
4. Character (“Columbia”, “Statistics are fun!”, “HELLO WORLD”)
5. Logical (TRUE, FALSE, 1, 0)

In this class, we are primarily concerned with numeric, character, and logical.

**character > complex > numeric > integer > logical**

# VECTORS

The most basic type of R object is a vector.

## Vector Rules

- ▶ Vectors **can only contain one class** of R object!
- ▶ Scalars are thought of as one-element vectors.
- ▶ Vectors are usually created with the concatenate function, e.g.  
`new_vec <- c(1, 2, 7).`

# ATTRIBUTES

R objects like the vector (or the matrix, array, dataframe, and list that we'll learn about later) have *attributes*, which are like metadata for the object.

vector can has the attribute like names, `name(x) <- c('one','two','three')`

## Examples

- ▶ column names or row names
- ▶ dimensions (matrix and array)
- ▶ class (e.g. character or numeric)
- ▶ length

For finding the attributes of an object (if it has any), the `attributes()` function is helpful.

## LET'S CHECK THIS OUT IN R

Type the following in your console:

```
x <- 2
class(x) numeric
class(2L) integer
y <- as.integer(3)
class(y)
class("Columbia University")
class(FALSE)

x <- c(0.5, 0.6)      # numeric
x <- c(TRUE, FALSE)    # logical
x <- 3:12               # integer
```

## LET'S CHECK THIS OUT IN R

Type the following in your console:

```
x <- c(1.7, "a")      # character
x <- c(TRUE, 2)        # numeric
x <- c("a", TRUE)      # character
```

## Mixing Objects

- ▶ In the above cases, we are mixing object classes in a vector – but this is not allowed!
- ▶ When different classes of objects are mixed in a vector, R *coerces* the objects all to the same class.
- ▶ Sometimes R makes the right decision and does what you want. Other times it doesn't. Be careful!

# LET'S CHECK THIS OUT IN R

## Explicit Coercion

- ▶ Objects can be explicitly coerced from one class to another using the `as.*()` function.
- ▶ If R can't figure out how to coerce the object, **NAs are produced**.
- ▶ Often convert numbers being interpreted as characters to numeric or between logical TRUE/FALSE to numeric 1/0, for instance.

Type the following in your console:

```
x <- 0:4
class(x)
as.numeric(x)
as.character(x)
as.logical(x)

x <- c("a", "b", "c")
as.numeric(x)
```

# MATRICES AND LISTS

## Matrices

`dim(x)` 返回行数和列数

- ▶ Matrices are vectors with a `dimension` attribute.
- ▶ The dimension attribute itself is an integer vector of length two.
- ▶ Like vectors, matrices must contain objects of the same class.
- ▶ Matrices are constructed `column-wise`, so entries start in the upper left corner and run down the columns.

## Lists

`unlist(mylist)` : transform a list back into a vector

- ▶ Lists are a special type of vector that can contain elements of different classes.
- ▶ They are usually used because they can contain different object classes.
- ▶ Lists are an important and powerful data type in R.
- ▶ Lists can be explicitly created using the `list()` function, which takes an arbitrary number of arguments.

# DATAFRAMES

- ▶ Data frames store tabular data.
- ▶ A special type of list where every element of the list has the same length.
- ▶ Each element of the list is a column and the length of each element is the number of rows.
- ▶ Unlike matrices and vectors, data frames can store different classes of objects in each column.
- ▶ Dataframes have the attributes: column names, indicating the names of the variables or predictors, and row names, indicating information about each row.
- ▶ Usually created by reading in a dataset using the `read.table()` or `read.csv()` functions.
- ▶ Can also be created explicitly with the `data.frame()` function or they can be coerced from other types of objects like lists.

# MATRICES AND LISTS

```
mymatrix <- matrix(vector,nrow=number of row, ncol=number of col,byrow=FALSE,  
dimnames=list(char_vector_row_name,char_vector_col_name))
```

Type the following in your console:

```
mat      <- matrix(ncol = 3, nrow = 2)  
mat  
dim(mat)  
attributes(mat)
```

```
mat      <- matrix(1:6, ncol = 3, nrow = 2)  
mat  
mat      <- matrix(1:6, ncol = 3, nrow = 2, byrow = TRUE)  
mat
```

```
m      <- 1:10  
m  
dim(m) <- c(2,5)  
m
```

# MATRICES AND LISTS

Type the following in your console:

```
x    <- rep(5, 3)
y    <- 11:13
mat <- cbind(x, y)
mat
mat <- rbind(x, y)
mat

x <- list(1, "a", TRUE, 1 + 4i)
x
```

# NAMING OBJECTS

Type the following in your console:

```
vec <- c("Welcome", "to", "Columbia")
vec
names(vec)
names(vec) <- c("Word1", "Word2", "Word3")
vec
names(vec)

myList <- list(stuff = 3, mat = matrix(1:4, nrow = 2),
               moreStuff = c("china", "japan"), list(5, "bear"))
myList
names(myList)      对于没有命名的部分直接为空 “ ”

mat
dimnames(mat) <- list(c("a", "b"), c("c", "d", "e"))
mat
colnames(mat) <- c("Alpha", "Beta", "Gamma")
mat
rownames(mat) <- c("red", "green")
```

# DATAFRAMES

补充：使用函数简便输入

1. attach ( data frame )

中间输入的不用再重复数据

Type the following in your console: data frame的名字  
attach(data frame)

```
x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
```

```
x
```

```
nrow(x)
```

```
ncol(x)
```

```
names(x)
```

```
names(x) <- c("Cindy", "Rush")
```

```
names(x)
```

```
x
```

2. with ( data frame, {

如果是内部操作 用 <-

如果希望操作可以在全局环境使用用

```
<<-
```

```
)
```

3. within(data frame, {

和with相似，但是允许修改数据框})

```
row.names(x) <- c("00", "01", "10", "11")
```

```
x
```

# A QUICK NOTE ABOUT NAMING

Naming dataframes and matrices are slightly confusing as they use different functions for setting the row and column names of each.

| Object    | Set Column Names        | Set Row Names            |
|-----------|-------------------------|--------------------------|
| dataframe | <code>names()</code>    | <code>row.names()</code> |
| matrix    | <code>colnames()</code> | <code>rownames()</code>  |

# DATA TYPES WE'VE LEARNED SO FAR

Can you summarize the differences?

## Data Types

- ▶ Vectors
- ▶ Scalars
- ▶ Matrices
- ▶ Lists
- ▶ Dataframes

补充：

1. array 数组：维度可以大于2 的matrix

`myarray<- array(vector,dimensions,dimnames)`  
dimensions 先行再列再其他维度依次规定

2. factor 因子

( 1 ) `myfactor <- factor(vector)` ~无序

( 2 ) `myfactor <- factor(vector,ordered = TRUE)` ~ 有序

( 3 ) `myfactor <- factor(vector, ordered =TRUE, levels = unique_vector)` ~ 有序且指定顺序

( 4 ) `myfactor <- factor(vector, ordered=TRUE, levels = unique_vector, labels= unique_vector_name)`  
~有序且指定顺序且命名

# R IS A FUNCTIONAL PROGRAMMING LANGUAGE

- ▶ Operations are carried out with functions.
- ▶ Functions take in R objects and return objects as outputs.
- ▶ An analysis can be considered a sequence of function calls, with the output of one function being the input of another function.
- ▶ In R, functions are themselves objects.
- ▶ Functions take arguments that are sometimes optional.

# R IS A FUNCTIONAL PROGRAMMING LANGUAGE

Type the following into your console:

```
?airquality
head(airquality)

median
class(median)
median(airquality$Temp)
median(airquality$Ozone)
median(airquality$Ozone, na.rm = TRUE)

?median # Getting function help
??median # Getting topic help
```

# INDICES AND SUBSETS

Type the following into your console:

```
vals <- seq(2, 12, by = 2)
vals

# Indexing a vector
vals[3]
vals[3:5]
vals[c(1, 3, 6)]
vals[-c(1, 3, 6)]
vals[c(rep(TRUE, 3), rep(FALSE, 2), TRUE)]

# Substituting Values
vals[4]    <- -35
vals
vals[1:2]  <- 0
vals
```

# TRY ON YOUR OWN!

Use the ? help if you need, but otherwise try to determine what R is doing in the following operations.

Type the following into your console:

```
vals           <- rnorm(100)
vals
vals[vals < 0] <- 0
vals
vals[1:8]

two_sds <- vals[vals > 2]
two_sds
```

补充创建新变量：

1. `dataframe$newcolname <- ...`
2. `dataframe <- transform(dataframe, newcolname1= ..., newcolname2=..., ...)`

# INDICES AND SUBSETS

Type the following into your console:

```
# Indexing a matrix
```

```
mat <- matrix(1:20, ncol = 4)
```

```
mat
```

补充使用subsit 函数

```
mat[2, 2]      newdata <- subset(dataframe, condition_one, condition two,...,  
mat[5, 3]          select=c(col_one,col_two,...))
```

如果list的内容已经命名使用方法类似

```
mat[, 1]
```

```
mat[, 3]
```

```
mat[2, ]
```

```
mat[2:4, ]
```

```
mat[4, 2] <- 100
```

```
mat
```

```
mat[4, ] <- rep(100, ncol(mat))
```

```
mat
```

# INDICES AND SUBSETS

Type the following into your console:

```
colnames(mat) <- c("One", "Two", "Three", "Four")
mat
mat[, "Four"]
mat[, 4]
mat[, c("One", "Three")]
mat[, c(1, 3)]
mat[, c(-1, -2)]
# What's going on?
mat[, -1:2]
mat[, -"One"]
```

补充日期型数据处理方式：

1. `mydata <- as.Date(x,'input_format')`  
%d/D : 缩写/不缩写的日期数  
%a/A : 缩写/不缩写的星期数  
%m : 数字表示的月份  
%b/B : 缩写/不缩写的字母表示的月份  
%y/Y : 二位/四位的年数  
默认： ' yyyy-mm-dd ' = '%y/%m/%d'
2. `Sys.Date date()` 系统当前时间
3. `format ( x, format='output_format' )`
4. `difftime(x,y,unit='week/day/year,...')`  
计算时间差

# INDICES AND SUBSETS

Type the following into your console:

```
myList
```

注意list需要双括号 【【】】

```
myList[[1]]
```

```
myList[["stuff"]]
```

```
myList$stuff
```

```
identical(myList$stuff, myList[[1]])
```

补充重命名：

```
myList$moreStuff[2]
```

1. names ( data ) [对应的那一个] <- 'newname'  
2. library(plyr)

```
myList[[4]]
```

dataframe <- rename(dataframe, c(old\_name1 =  
new\_name1, old\_name2 = new\_name2,...))

```
myList[[4]][[2]]
```

```
myList[1:3]
```

```
myList$newOne <- "more weird stuff"
```

```
myList
```

# INDICES AND SUBSETS

Type the following into your console:

```
head(airquality)  
  
vec <- airquality$Temp  
vec[1:10]
```

```
new_vec <- airquality[, 5]  
new_vec[1:10]
```

补充缺失值的处理：

1. 忽略缺失值 : rm.na=TRUE
2. 移除缺失值整行数据 : na.omit=(dataframe)
3. 判断缺失值 : is.na(data) 返回的是同样大小的由T/F组成的值
4. Nan : not a number : is.nan()
5. NULL: 不存在
6. infinite : is.infinite ( )

# TRY IT YOURSELF!

补充dplyr 常用操作：

```
data <- filter(data, condition_for_col ( 对列数据按条件选出符合的行 ) )
      %>% select(col1,col2,... ( 选择出需要的列 ) )
      %>% summarize(number=n(col1,col2,...),unique=n_distinct
(col1,col2,...),mean=mean(col1,col2,...))
```

- ▶ Create a vector `x` that contains the Wind variable from the `airquality` dataset.
- ▶ Create a variable `avg` that contains the mean of the vector `x`.
- ▶ Use a function in R to round the wind speeds in the vector `x` to the nearest whole number.

数据合并：

1. `merge(x, y, by = intersect(names(x), names(y)), by.x = by, by.y = by, all = FALSE, all.x = all, all.y = all, sort = TRUE, suffixes = c(".x",".y"))`
2. `rbind()`
3. `cbind()`

# AN EXTENDED EXAMPLE: IMAGE DATA

# IMAGE DATA EXAMPLE<sup>1</sup>

- ▶ Images are made up of pixels that are arranged in rows and columns (like a matrix).
- ▶ Image data are matrices where each element is a number representing the intensity or brightness of the corresponding pixel.
- ▶ We will work with a grayscale image with numbers ranging from 0 (black) to 1 (white).

---

<sup>1</sup>Example developed from N. Matloff, “The Art of R Programming: A Tour of Statistical Software Design”.

# BUT FIRST...PACKAGES!

## What are packages?

- ▶ Packages are collections of functions, data, or code that extend the capabilities of base R.
- ▶ Some packages come pre-loaded but others must be downloaded and installed using function `install.packages("package name")` (**only once**).
- ▶ An installed R package must be loaded using function `library("package name")` (**every time you want to use it**).

Type the following into your console:

```
# Installing the "pixmap" package.
```

```
install.packages("pixmap")
library("pixmap")
```

# LOCAL DATA

When importing data from your machine, you need to tell R where to find that data.

## Working Directory

- ▶ `getwd()` tells you where R is currently looking, or where your *working directory* is set.
- ▶ You can change your *working directory* with `setwd(<file path for the data>)`.
- ▶ Can also change *working directory* with Session -> Set Working Directory -> Change Working Directory.
- ▶ Usually your file path will look like  
`"/Users/cynthiarush/Documents/GR5206/Week2"`.

# WORKING DIRECTORY

Type the following into your console:

```
install.packages(" pixmap ")
library(" pixmap ")
getwd()
#setwd()           Where did you save the image data?
list.files() # Is casablanca.pgm there?

# The following will only work if you downloaded the data
# from courseworks and it is saved in your working directory
```

```
casablanca_pic <- read.pnm("casablanca.pgm")
```

```
casablanca_pic      补充图像数据的处理与存取 :
```

```
plot(casablanca_pic)
```

- 1.read.pnm() 读
- 2.pdf('plotname.pdf')  
draw the pic
- dev.off() 画与存

## IMAGE DATA EXAMPLE (CONT.)

Type the following into your console:

```
dim(casablanca_pic@grey)

# Access select pixel values

casablanca_pic@grey[300, 100]
casablanca_pic@grey[200, 10]

# Add stars where we looked at pixel values
# Note the difference (0,0) reference points

points(100, 60, pch = "*", col = "red")
points(10, 160, pch = "*", col = "red")
```

## IMAGE DATA EXAMPLE (CONT.)

Type the following into your console:

```
# Let's erase Rick from the image!  
  
casablanca_pic@grey[15:70, 220:265] <- 1  
plot(casablanca_pic)
```

## IMAGE DATA EXAMPLE (CONT.)

- ▶ Use R's `locator()` function to find the rows and columns corresponding to Rick's face.
- ▶ A call to the function allows the user to click on a point in a plot and then the function returns the coordinates of the click.

the locator function and the read.pnm function have two different coordinates systems.

if locator function returns (x1,y1) and (x2,y2), you shoud input  
`(dim(pic@grey)[1]-y1:y2,x1:x2)`

```
casablanca_pic@grey[dim(casablanca_pic@grey)[1]-292:360,  
210:266] <- 1  
plot(casablanca_pic)
```

# CHECK YOUR UNDERSTANDING

Type the following into your console:

```
z           <- matrix(rep(1:9), nrow = 3)
colnames(z) <- c("First", "Second", "Third")
z
```

Using matrix `z`, what is the output of the following? (Try to guess first, then enter the code to check your answer!)

1. `z[2:3, "Third"]?`
2. `c(z[,-(2:3)], "abc")?`
3. `rbind(z[1,], 1:3)?`

# MORE WITH VECTORS & MATRICES

# REMINDER: VECTOR ALGEBRA

Define vectors:

$$A = (a_1, a_2, \dots, a_N), \quad \text{and} \quad B = (b_1, b_2, \dots, b_N).$$

Then for  $c$  a scalar,

- ▶  $A + c = (a_1 + c, a_2 + c, \dots, a_N + c).$
- ▶  $A + B = (a_1 + b_1, a_2 + b_2, \dots, a_N + b_N).$
- ▶  $cA = (ca_1, ca_2, \dots, ca_N).$
- ▶ Dot product:  $A \cdot B = a_1 b_1 + a_2 b_2 + \dots + a_N b_N.$
- ▶ Norm:  $\|A\|^2 = A \cdot A = a_1^2 + a_2^2 + \dots + a_N^2.$

# REMINDER: MATRIX ALGEBRA

Define matrices:

$$A = \begin{pmatrix} a_1 & a_3 \\ a_2 & a_4 \end{pmatrix}, \quad \text{and} \quad B = \begin{pmatrix} b_1 & b_3 \\ b_2 & b_4 \end{pmatrix}.$$

Then for  $c$  a scalar,

- $A + c = \begin{pmatrix} a_1 + c & a_3 + c \\ a_2 + c & a_4 + c \end{pmatrix}.$
- $A + B = \begin{pmatrix} a_1 + b_1 & a_3 + b_3 \\ a_2 + b_2 & a_4 + b_4 \end{pmatrix}.$
- $cA = \begin{pmatrix} ca_1 & ca_3 \\ ca_2 & ca_4 \end{pmatrix}.$
- Matrix Multiplication:  $AB = \begin{pmatrix} a_1b_1 + a_3b_2 & a_1b_3 + a_3b_4 \\ a_2b_1 + a_4b_2 & a_2b_3 + a_4b_4 \end{pmatrix}.$

What if the dimensions of  $A$  and  $B$  are different?

# REMINDER: MATRIX OPERATIONS

Define matrices:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix}, \quad \text{and } B = \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix}.$$

- ▶ The *transpose* of  $A$  is a  $m \times n$  matrix:

$$t(A) = \begin{pmatrix} a_{1,1} & a_{2,2} & \cdots & a_{n,1} \\ a_{1,2} & a_{2,2} & \cdots & a_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1,m} & a_{2,m} & \cdots & a_{n,m} \end{pmatrix}.$$

- ▶ The *trace* of the square matrix  $B$  is the sum of the diagonal elements:  $tr(B) = b_{1,1} + b_{2,2}$ .

# REMINDER: MATRIX OPERATIONS

Define matrices:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix}, \quad \text{and } B = \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix}.$$

- The **determinant** of square matrix  $B$  is  $\det(B) = b_{1,1}b_{2,2} - b_{1,2}b_{2,1}$ . How do you find the determinant for an  $n \times n$  matrix?
- The **inverse** of square matrix  $B$  is denoted  $B^{-1}$  and

$$BB^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \text{ with } B^{-1} = \frac{1}{\det(B)} \begin{pmatrix} b_{2,2} & -b_{1,2} \\ -b_{2,1} & b_{1,1} \end{pmatrix}.$$

矩阵求逆 : solve ( A )

How do you find the inverse of an  $n \times n$  matrix?

# FUNCTIONS ON NUMERIC VECTORS

## Useful R Functions

| R function                | Description                               |
|---------------------------|---|
| <code>length(x)</code>    | Length of a vector $x$                    |
| <code>sum(x)</code>       | Sum of a vector $x$                       |
| <code>mean(x)</code>      | Arithmetic mean of a vector $x$           |
| <code>quantiles(x)</code> | Sample quantiles of a vector $x$          |
| <code>max(x)</code>       | Maximum of a vector $x$                   |
| <code>min(x)</code>       | Minimum of a vector $x$                   |
| <code>sd(x)</code>        | Sample standard deviation of a vector $x$ |
| <code>var(x)</code>       | Sample variance of a vector $x$           |
| <code>summary(x)</code>   | Summary statistics of vector $x$          |

## Reminder...

To access the help documentation of a known R function, use syntax  
`?function.`

# ELEMENT-WISE OPERATIONS FOR VECTORS

Vectors  $x$  and  $y$  must have the same length. Let  $\mathbf{a}$  be a scalar.

## Element-Wise Operators

| Operator         | Description                        |
|------------------|------------------------------------|
| $\mathbf{a} + x$ | Element-wise scalar addition       |
| $\mathbf{a} * x$ | Element-wise scalar multiplication |
| $x + y$          | Element-wise addition              |
| $x * y$          | Element-wise multiplication        |
| $x ^ \mathbf{a}$ | Element-wise power                 |
| $\mathbf{a} ^ x$ | Element-wise exponentiation        |
| $x ^ y$          | Element-wise exponentiation        |

## Recycling

Recall that a scalar is just a vector of length 1. When a shorter vector is added to a longer one, the elements in the shorter vectored are repeated. This is *recycling*.

# SOME EXAMPLES

Type the following into your console:

```
u <- c(1, 3, 5)
v <- c(1, 3, 5)
v + 4      # Recycling
v + c(1, 3) # Recycling: What is happening here?
v + u
```

# FUNCTIONS FOR NUMERIC MATRICES

## Useful R Functions

| R Function               | Description   |
|--------------------------|---|
| <code>A %*% B</code>     | Matrix multiplication for compatible matrices $A, B$ .          |
| <code>dim(A)</code>      | Dimension of matrix $A$ .                                       |
| <code>t(A)</code>        | Transpose of matrix $A$ .                                       |
| <code>diag(x)</code>     | Returns a diagonal matrix with elements $x$ along the diagonal. |
| <code>diag(A)</code>     | Returns a vector of the diagonal elements of $A$ .              |
| <code>solve(A, b)</code> | Returns $x$ in the equation $b = Ax$ .                          |
| <code>solve(A)</code>    | Inverse of $A$ where $A$ is a square matrix.                    |
| <code>cbind(A, B)</code> | Combine matrices horizontally for compatible matrices $A, B$ .  |
| <code>rbind(A, B)</code> | Combine matrices vertically for compatible matrices $A, B$ .    |

如果A是一个向量 , `diag ( A )` 是一个以该向量元素为对角线元素的矩阵  
如果A是一个矩阵 , `diag ( A )` 是一个以该矩阵对角线元素为元素的向量

# SYSTEM OF LINEAR EQUATIONS EXAMPLE

Solve the system of equations: 
$$\begin{cases} 3x - 2y + z = -1 \\ x + \frac{1}{2}y - 12z = 2 \\ x + y + z = 3 \end{cases}$$

Recall,

We can represent the system using matrices as follows:

$$\begin{pmatrix} 3 & -2 & 1 \\ 1 & \frac{1}{2} & -12 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \\ 3 \end{pmatrix}.$$

Then we would like to solve for vector  $(x, y, z)$ .

# SYSTEM OF LINEAR EQUATIONS EXAMPLE (CONT.)

Type the following into your console:

```
# Define matrix A
```

```
A <- matrix(c(3,1,1,-2,1/2,1,1,-12,3), nrow = 3)
```

```
A
```

```
# Define vector b
```

```
b <- c(-1, 2, 3)
```

```
b
```

Ax=b

x=solve(A,b)

```
# Use the solve function
```

```
x <- solve(A, b)
```

```
# Check the solution
```

```
A %*% x
```

# ELEMENT-WISE OPERATIONS FOR MATRICES

Let  $A$  and  $B$  be matrices of the same dimensions. Let  $a$  be a scalar.

## Element-wise Operators

| Operator | Description                        |
|----------|------------------------------------|
| $a + A$  | Element-wise scalar addition       |
| $a * A$  | Element-wise scalar multiplication |
| $A + B$  | Element-wise addition              |
| $A * B$  | Element-wise multiplication        |
| $A ^ a$  | Element-wise power                 |
| $a ^ A$  | Element-wise exponentiation        |
| $A ^ B$  | Element-wise exponentiation        |

# EIGENVALUE EXAMPLE

Check if 5 is an eigenvalue of the matrix  $A = \begin{pmatrix} 1 & -2 \\ -2 & 4 \end{pmatrix}$ .

Recall,

If  $\lambda$  is an eigenvalue of a square matrix  $A$ , then  $Av = \lambda v$  for some non-zero vector  $v$ . Equivalently, if  $\lambda$  is an eigenvalue of  $A$  then  $\det(A - 5I) = 0$  where  $I$  is an identity matrix.

# EIGENVALUE EXAMPLE

Check if 5 is an eigenvalue of the matrix  $A = \begin{pmatrix} 1 & -2 \\ -2 & 4 \end{pmatrix}$ .

Type the following into your console:

```
A <- matrix(c(1, -2, -2, 4), nrow = 2, byrow = TRUE)
A

# Define a 2 by 2 identity matrix
identity <- diag(2)
identity

# Check if 5 is an eigenvalue of A
det(A - 5*identity)
```

# Lecture 2: Working with Data in R

## STAT GR5206

*Statistical Computing & Introduction to Data Science*

Cynthia Rush  
Columbia University

September 15, 2017

# COURSE NOTES

- ▶ First lab next week. Should only attend one day.
- ▶ I hope everyone has R downloaded.
- ▶ Ask questions on Piazza and answer each other's questions.

# REVIEW

- ▶ Using R, RStudio, and RMarkdown.
- ▶ Vectors, matrices, lists, and dataframes.
- ▶ Subsetting and naming (will wrap this up now).

# OBJECT CLASSES, VECTORS, & MATRICES (CONT.)

# INDICES AND SUBSETS

Type the following into your console:

```
# Indexing a matrix
mat <- matrix(1:20, ncol = 4)
mat

mat[2, 2]
mat[5, 3]

mat[, 1]
mat[, 3]
mat[2, ]
mat[2:4, ]

mat[4, 2] <- 100
mat
mat[4, ] <- rep(100, ncol(mat))
mat
```

# INDICES AND SUBSETS

Type the following into your console:

```
colnames(mat) <- c("One", "Two", "Three", "Four")
mat
mat[, "Four"]
mat[, 4]
mat[, c("One", "Three")]
mat[, c(1, 3)]

mat[, c(-1, -2)]

# What's going on?
mat[, -1:2]      mat[,-1:-2] 去除第一第二列
mat[, -"One"]
```

# EXTRACTING COMPONENTS OF LISTS

Extract an individual component `c` directly from a list called `lst` in the following ways:

- ▶ `lst$c`
- ▶ `lst[[i]]` where `c` is the  $i^{th}$  component.
- ▶ `lst[["c"]]`

# EXTRACTING COMPONENTS OF LISTS

Type the following into your console:

```
myList <- list(stuff = 3, mat = matrix(1:4, nrow = 2),  
    moreStuff = c("china", "japan"), list(5, "bear"))  
myList
```

```
myList[[1]]  
myList[["stuff"]]  
myList$stuff  
  
myList$moreStuff[2]
```

```
myList[[4]]  
myList[[4]][[2]]
```

```
myList[1:3]
```

```
myList$newOne <- "more weird stuff"  
myList
```

# SUBSETTING R OBJECTS

## Three Possible Operators:

- ▶ The `[` operator returns an object of the same class. Can be used to select multiple elements.
- ▶ The `[[` operator extracts elements of a list or a data frame. Can only be used to extract a single element and the class of the returned object will not necessarily be a list or data frame.
- ▶ The `$` operator extracts elements of a list or data frame by literal name. Its semantics are similar to that of `[[`].

# ELEMENT-WISE OPERATIONS FOR VECTORS

Vectors  $x$  and  $y$  must have the same length. Let  $\mathbf{a}$  be a scalar.

## Element-Wise Operators

| Operator                  | Description                        |
|---------------------------|------------------------------------|
| $\mathbf{a} + \mathbf{x}$ | Element-wise scalar addition       |
| $\mathbf{a} * \mathbf{x}$ | Element-wise scalar multiplication |
| $\mathbf{x} + \mathbf{y}$ | Element-wise addition              |
| $\mathbf{x} * \mathbf{y}$ | Element-wise multiplication        |
| $\mathbf{x} ^ \mathbf{a}$ | Element-wise power                 |
| $\mathbf{a} ^ \mathbf{x}$ | Element-wise exponentiation        |
| $\mathbf{x} ^ \mathbf{y}$ | Element-wise exponentiation        |

## Recycling

Recall that a scalar is just a vector of length 1. When a shorter vector is added to a longer one, the elements in the shorter vectored are repeated. This is *recycling*.

## SOME EXAMPLES: RECYCLING

Type the following into your console:

```
u <- c(1, 3, 5)
v <- c(1, 3, 5)
v + 4      # Recycling
v + c(1, 3) # Recycling: What is happening here?
v + u
```

# AN EXTENDED EXAMPLE: IMAGE DATA

# IMAGE DATA EXAMPLE<sup>1</sup>

- ▶ Images are made up of pixels that are arranged in rows and columns (like a matrix).
- ▶ Image data are matrices where each element is a number representing the intensity or brightness of the corresponding pixel.
- ▶ We will work with a grayscale image with numbers ranging from 0 (black) to 1 (white).

---

<sup>1</sup>Example developed from N. Matloff, “The Art of R Programming: A Tour of Statistical Software Design”.

# BUT FIRST...PACKAGES!

## What are packages?

- ▶ Packages are collections of functions, data, or code that extend the capabilities of base R.
- ▶ Some packages come pre-loaded but others must be downloaded and installed using function `install.packages("package name")` (**only once**).
- ▶ An installed R package must be loaded using function `library("package name")` (**every time you want to use it**).

Type the following into your console:

```
# Installing the "pixmap" package.
```

```
install.packages("pixmap")
library("pixmap")
```

# LOCAL DATA

To read and write from R, you need to have a firm grasp of where in the computer's filesystem you are reading and writing from.

## Working Directory

- ▶ `getwd()` tells you where R is currently looking, or where your **working directory** is set.
- ▶ You can change your **working directory** with `setwd(<file path for the data>)`.
- ▶ Can also change **working directory** with Session → Set Working Directory → Change Working Directory.
- ▶ Usually your file path will look like  
`"/Users/cynthiarush/Documents/GR5206/Week2"`.

# WORKING DIRECTORY

Type the following into your console:

```
install.packages(" pixmap ")
library(" pixmap ")
getwd()
#setwd()           Where did you save the image data?
list.files() # Is casablanca.pgm there?

# The following will only work if you downloaded the data
# from courseworks and it is saved in your working directory

casablanca_pic <- read.pnm("casablanca.pgm")
casablanca_pic
plot(casablanca_pic)
```

## IMAGE DATA EXAMPLE (CONT.)

Type the following into your console:

```
dim(casablanca_pic@grey)

# Access select pixel values

casablanca_pic@grey[300, 100]
casablanca_pic@grey[200, 10]

# Add stars where we looked at pixel values
# Note the difference (0,0) reference points

points(100, 60, pch = "*", col = "red")
points(10, 160, pch = "*", col = "red")
```

## IMAGE DATA EXAMPLE (CONT.)

Type the following into your console:

```
# Let's erase Rick from the image!  
  
casablanca_pic@grey[15:70, 220:265] <- 1  
plot(casablanca_pic)
```

## IMAGE DATA EXAMPLE (CONT.)

- ▶ Use R's `locator()` function to find the rows and columns corresponding to Rick's face.
- ▶ A call to the function allows the user to click on a point in a plot and then the function returns the coordinates of the click.

# CHECK YOUR UNDERSTANDING

Type the following into your console:

```
z           <- matrix(rep(1:9), nrow = 3)
colnames(z) <- c("First", "Second", "Third")
z
```

Using matrix `z`, what is the output of the following? (Try to guess first, then enter the code to check your answer!)

1. `z[2:3, "Third"]?`
2. `c(z[,-(2:3)], "abc")?`
3. `rbind(z[1,], 1:3)?`

# NA AND NULL VALUES

# NA AND NULL

Since it was written by statisticians, R handles missing data relatively well compared to other languages.

- ▶ NA indicates a missing value in a dataset.
- ▶ NULL is a value that doesn't exist and is often returned by expressions and functions whose value is undefined.
- ▶ NaN stands for 'not a number'.

Type the following into your console:

```
length(c(-1, 0, NA, 5))  
length(c(-1, 0, NULL, 5))
```

# NA AND NULL

Type the following into your console:

```
### Use na.rm = TRUE to remove NA values
t <- c(-1, 0, NA, 5)
mean(t)
mean(t, na.rm = TRUE)
```

```
### NA values are missing, but NULL values don't exist.
s <- c(-1, 0, NULL, 5)
mean(s)
```

```
### is.na() returns a logical indicating if its input is NA.
is.na(c(3,-7.5, NA, pi))
c(3/0, 0/0)
is.na(c(3/0, 0/0))
```

Nan 在 is.na()的测试中返回的也是true

## NULL CAN BE USED TO BUILD A VECTOR:

Type the following into your console:

```
# Define an empty vector
x <- NULL
```

```
# Fill in the vector
x[1] <- "Blue"
x[2] <- "Green"
x[3] <- "Red"
x
```

用 NULL 可以删除变量

```
myList <- list(a = 7, b = 5)
myList$a <- NULL # works for data frames too
myList
```

# FACTORS AND TABLES

# FACTORS DEFINITION

- ▶ A **factor** is a special data type in R used for **qualitative data** that can assume only a discrete number of values (i.e. **categorical** data).
- ▶ In R, think of factors as integer vectors with additional information which is a record of the distinct elements of the factor, **called the levels**.
- ▶ R automatically treats factors specially in many functions. In some cases it works like magic, and in others it is incredibly frustrating.
- ▶ Note that factors can be ordered or unordered.

# FACTORS EXAMPLE

The **levels** of the factor group are Control and Treatment.

Type the following into your console:

```
# Creates a character vector
data  <- rep(c("Control", "Treatment"), c(3,4))
data

# Makes a factor vector
group <- factor(data)
group

str(group)
attributes(group)
table(group)
```

# FUNCTIONS ON FACTORS

The `split()` function takes as input a vector and a factor (or list of factors), splitting the input according to the groups of the factor. The output is a list.

Type the following into your console:

```
ages <- c(20, 30, 40, 35, 35, 35, 35)
sex  <- c("M", "M", "F", "M", "F", "F", "F")
group <- list(control,control,control,treatment,treatment,treatment,treatment)
# Use split() to list ages in each group + sex pair
split(ages, list(group, sex))  
# split(ages, list(group, sex)) 这里把group和sex两个作为一个整体作为
# split的依据，对应这两个信息找到对应的
# age并且split成一个list返回
# split() has coerced sex into a factor variable.
```

# TABLES

The `table()` function can be used to produce *contingency tables* in R.

Type the following into your console:

```
group
table(group)
table(sex, group)

# Most matrix operations work on tables as well.

new_table <- table(sex, group)
new_table[, "Control"]
round(new_table/length(group), 3) # Gives proportions
```

# DATAFRAMES

# DATAFRAMES

## Review:

- ▶ Use for standard two-dimensional tables of data, similar to a spreadsheet.
- ▶ Like matrices, but each column can have a different class (character, logical, numeric, ...) ..
- ▶ Can have row and column names.

# DATAFRAMES

## Review:

- ▶ Use for standard two-dimensional tables of data, similar to a spreadsheet.
- ▶ Like matrices, but each column can have a different class (character, logical, numeric, ...).
- ▶ Can have row and column names.

## Creating Dataframes in R

- ▶ Use `data.frame()` to create dataframes in R. (Will usually import data from other sources, though.)
- ▶ `stringsAsFactors = TRUE`, the default, turns character vectors into a `factor` variable.
- ▶ Usually set `stringsAsFactors = FALSE` and set factors manually.

# CREATING A DATAFRAME

Type the following into your console:

```
Name  <- c("John", "Jill", "Jacob", "Jenny")
Year  <- c(1,1,2,4)
Grade <- c("B", "A+", "B-", "A")
student_data <- data.frame(Name, Year, Grade,
                           stringsAsFactors = FALSE)

student_data
dim(student_data)
str(student_data)
summary(student_data)
```

# DATAFRAMES: ANOTHER EXAMPLE

Type the following into your console:

```
library(datasets)
states <- data.frame(state.x77, Region = state.region,
                      Abbr = state.abb)
head(states, 2)
```

**states** combines pre-existing matrix **state.x77** with categorical vector **state.region** and character vector **state.abb**. More info: **?state.x77**.

# FILTERING DATAFRAMES

Type the following into your console:

```
student_data
student_data[3:4,]
student_data$Grade

states["New York", ] # Can also use rownames

# Adding a new row
new_stu      <- c("Bobby", 3, "A")
student_data <- rbind(student_data, new_stu)
student_data
```

# ADDING ROWS AND COLUMNS TO DATAFRAMES

Type the following into your console:

```
# Recycling works too
```

```
student_data$School <- "Columbia"  
student_data
```

- ▶ Note that this construction would not work with a matrix.
- ▶ Can add a new component to an already existing dataframe.

# IMPORTING DATA INTO R

# WHAT KINDS OF DATA?

- ▶ Data can be saved on your computer. What we'll work on this today.
- ▶ Data can be on the internet. We'll do this in a few weeks.
- ▶ Data can be in a database. Also, in a few weeks.
- ▶ Other sources too.

# SPREADSHEET DATA

Often we work with data formatted in a rectangular grid that we want to read into a dataframe.

- ▶ The workhorse for reading into a data frame is `read.table()`, which allows any separator (CSV, tab-delimited, etc.).
- ▶ `read.csv()` is a special case of `read.table()` for CSV files
- ▶ If the below was stored in `stu_data.txt`, use `read.table("stu_data.txt", header = FALSE, as.is = TRUE)`.

|       |   |    |
|-------|---|----|
| John  | 1 | B  |
| Jill  | 1 | A+ |
| Jacob | 2 | B- |
| Jenny | 4 | A  |

- ▶ Use the `sep=` argument if data is separated by something other than whitespace.
- ▶ `as.is = TRUE` is the same as `stringsAsFactors = FALSE`.

# IMPORTING DATA

- ▶ R can also read in (and write out) Excel files, STATA files, netCDF files, HDF5 files, etc., in many cases through add-on packages from CRAN.
- ▶ The `foreign` package can be helpful for this.

# IMPORTING DATA

- ▶ R can also read in (and write out) Excel files, STATA files, netCDF files, HDF5 files, etc., in many cases through add-on packages from CRAN.
- ▶ The `foreign` package can be helpful for this.

## A Note:

Please try to avoid using Excel files as a data storage format. It's proprietary, complicated (can have multiple sheets), allows a limited number of rows/columns, and files are not easily readable/viewable (unlike simple text files).

# IMPORTING DATA

- ▶ Both `read.table()` and `read.csv()` use the function `scan()` to import the data, then they format it.
- ▶ Sometimes want to use `scan()` outright.
- ▶ `scan()` output is a vector with elements anything from the file separated by whitespace.

# IMPORTING DATA

- ▶ Both `read.table()` and `read.csv()` use the function `scan()` to import the data, then they format it.
- ▶ Sometimes want to use `scan()` outright.
- ▶ `scan()` output is a vector with elements anything from the file separated by whitespace.

## Honor Code Example

The file "HonorCode.txt" contains Columbia University's Honor Code:

“Students should be aware that academic dishonesty (for example, plagiarism, cheating on an examination, or dishonesty in dealing with a faculty member or other University official) or the threat of violence or harassment are particularly serious offenses and will be dealt with severely under Dean's Discipline...”

# HONOR CODE EXAMPLE

Type the following into your console:

```
HC <- scan("HonorCode.txt", what = "")  
head(HC, 20)  
str(HC)
```

By default, R expects the input of scan to be numeric data. The argument `what=""` tells R that our data is a vector of character values.

# CLEANING DATA

Some gotchas to look out for when you're importing data into R.

- ▶ Is the first row a header? What about the first column?
- ▶ R interprets words separated by a space as two separate values. Messes up the number of elements per line in your data set. (Use \_ or . between words.)
- ▶ Symbols such as ?, %, &, \*, etc. can make R do funny things.
- ▶ Headers, footers, side comments, and notes will mess up the structure.
- ▶ How are missing values indicated? It should be with NA but often something like 999 or N.A.

# EXPORTING DATA

- ▶ You can write out R objects to an R Data file, using `save()` and `save.image()`.
- ▶ Often we want to export a matrix or dataframe into a file on our machine with delimiters.
  - ▶ This is done with `write.table()` or `write.csv()`.
  - ▶ Note that the default for both is `col.names = TRUE` and `row.names = TRUE`.
- ▶ There are also options for saving plots or even writing R tables with formatting for LaTex.

## SECTION VII

# Data in R: A Text Example

# TEXT DATA

- ▶ Textual Data Mining is commonly studied in machine learning.
- ▶ Examples: Can we write an algorithm to tell whether a newspaper article is a ‘positive’ or ‘negative’ response? Can we identify the author of a novel just from the text?
- ▶ We’ll learn more in a few weeks about how to deal with text data in R.
- ▶ For the time being, a quick example.

## HONOR CODE TEXT DATA <sup>2</sup>

Recall the Honor Code data we've imported into R.

Type the following into your console:

```
HC <- scan("HonorCode.txt", what = "")  
head(HC, 15)  
HC <- factor(HC, levels = unique(HC))
```

---

<sup>1</sup>Example developed from N. Matloff, “The Art of R Programming: A Tour of Statistical Software Design”

# HONOR CODE TEXT DATA <sup>2</sup>

Recall the Honor Code data we've imported into R.

Type the following into your console:

```
HC <- scan("HonorCode.txt", what = "")  
head(HC, 15)  
HC <- factor(HC, levels = unique(HC))
```

- ▶ HC is a vector and each word of the Honor Code is an element of the vector.
- ▶ Let's write a function **findwords()** that compiles a list of the location of each occurrence of each word in the text.

---

<sup>1</sup>Example developed from N. Matloff, "The Art of R Programming: A Tour of Statistical Software Design"

# FUNCTIONS IN R

## Basic Structure

```
function_name <- function(arg1, arg2, ...){  
  statements  
  return(object)  
}
```

- ▶ A **function** is a group of instructions that takes inputs, uses them to compute other values, and returns a result.
- ▶ We can write and add our own functions in R.
- ▶ Functions:
  1. Have names.
  2. *Usually* take in arguments.
  3. Include body of code that does something.
  4. *Usually* return an object at the end.

# EXAMPLE FUNCTION

Type the following into your console:

```
# A function to square its input
square_it <- function(x){
  out <- x*x
  return(out)
}

# Let's try it:
square_it(2); square_it(-4); square_it(146)
```

# HONOR CODE TEXT DATA

Let's write a function `findwords()` that compiles a list of the location of each occurrence of each word in the text.

We can do this with the `split()` function!

```
findwords <- function(text_vec){  
  words <- split(1:length(text_vec), text_vec)  
  return(words)  
}
```

- split 函数是有两个参数，第一参数表示split的目标，第二个参数是一个factor 按照第二个参数去第一参数中找对应的位置，并且split成一个新的list
  - length(text\_vec) is the total number of words in the textfile. In Honor Code it's 443.
  - text\_vec is a factor, with each unique word as a level. There are 243 levels in the Honor Code.
  - split() then determines the locations of each unique word (i.e. each level of the factor) and returns the locations in list form.

# HONOR CODE TEXT DATA

Does it work?

```
findwords(HC)[1:3]
```

```
HC[c(1, 48, 142, 204, 232, 310, 331)] # hopefully 'students'  
HC[c(2, 206)] # hopefully 'should'
```

# HONOR CODE TEXT DATA

Does it work?

```
findwords(HC)[1:3]
```

```
HC[c(1, 48, 142, 204, 232, 310, 331)] # hopefully 'students'  
HC[c(2, 206)] # hopefully 'should'
```

- ▶ Note that `findwords()` returns a list. The list consists of one element per word in the Honor Code.
- ▶ Must we use a list? How about a matrix output or a dataframe?
- ▶ Each row could correspond to a unique word and column to locations of the word. Different words would use different numbers of columns.
- ▶ The list structure makes the most sense here.

# HONOR CODE TEXT DATA

Finally, let's write a function to **alphabetize** our word list (the output of the previous function).

## List in Alphabetical Order

```
alphabetized_list <- function(wordlist) {  
  nms      <- names(wordlist) # The names are the words  
  sorted <- sort(nms)        # The words, but now in ABC order  
  return(wordlist[sorted])  # Returns the sorted version  
}  
输入的word list是findword函数的结果，是一个list，HC是一个vector
```

## Exercise

Break this function apart and run it line by line to make sure you know what it's doing.

# HONOR CODE TEXT DATA

Does it work?

List in Alphabetical Order

```
wl <- findwords(HC)
wl[1:3]
alpha_list <- alphabetized_list(wl)
alpha_list[1:3]
```

# FILTERING

# LOGICAL AND RELATIONAL OPERATORS

| Logical Operator | Description    |
|------------------|----------------|
| !                | Negation (NOT) |
| &                | AND            |
|                  | OR             |

| Relational Operator | Description                                     |
|---------------------|---|
| <, >                | Less than, greater than                         |
| <=, >=              | Less than or equal to, greater than or equal to |
| ==                  | Equal to  |
| !=                  | Not equal to                                    |

# EXAMPLES OF LOGICAL AND RELATIONAL OPERATORS

Type the following into your console:

```
1 > 3
```

```
1 == 3
```

```
1 != 3
```

```
(1 > 3) & (4*5 == 20)
```

```
(1 > 3) | (4*5 == 20)
```

# EXAMPLES OF LOGICAL AND RELATIONAL OPERATORS

Type the following into your console:

```
c(0,1,4) < 3
```

which函数输入的一个逻辑形的变量，返回的是对应的true的位置的序列号。

```
which(c(0,1,4) < 3)
```

```
which(c(TRUE, TRUE, FALSE))
```

```
c(0,1,4) >= c(1,1,3)      --位置对应比较
```

```
c("Cat", "Dog") == "Dog"
```

补充 %in% 函数：

c(1,2,3) %in% c(1,2,4,5) ==> C(TRUE,TRUE, FALSE)

# EXTRACTING ELEMENTS FROM A VECTOR

Sometimes we would like to extract elements from a vector or matrix that satisfy certain criteria.

Type the following into your console:

```
w <- c(-3, 20, 9, 2)
```

```
### Extract elements of w greater than 3
w[w > 3]
```

```
# What's going on here?
w > 3
w[c(FALSE, TRUE, TRUE, FALSE)]
```

# EXTRACTING ELEMENTS FROM A VECTOR

Sometimes we would like to extract elements from a vector or matrix that satisfy certain criteria.

Type the following into your console:

```
w <- c(-3, 20, 9, 2)
```

```
### Extract elements of w with squares between 3 and 10
w[w*w >= 3 & w*w <= 10]
```

```
# What's going on here?
w*w >= 3
w*w <= 10
w*w >= 3 & w*w <= 10
```

# EXTRACTING ELEMENTS FROM A VECTOR

Type the following into your console:

```
w <- c(-3, 20, 9, 2)
v <- c(0, 17, 10, 1)

### Extract elements of w greater than elements from v
w[w > v]

# What's going on here?
w > v
w[c(FALSE, TRUE, FALSE, TRUE)]
```

# FILTERING ELEMENTS OF A MATRIX

Type the following into your console:

```
M <- matrix(c(rep(4,5), 5:8), ncol=3, nrow=3)
M
```

```
### We can do element-wise comparisons with matrices too.
M > 5
M[ ,3] < 8
M[M[ ,3] < 8, ]
```

## REASSIGNING ELEMENTS OF A MATRIX

Type the following into your console:

```
M
```

```
### Assign elements greater than 5 with zero
```

```
M[M > 5] <- 0
```

```
M
```

# CHECK YOUR UNDERSTANDING

Type the following into your console:

```
z           <- matrix(c(1:3, TRUE, FALSE, TRUE, 9, 16, 25),  
                      nrow = 3)  
colnames(z) <- c("First", "Second", "Third")  
z
```

Using matrix `z`, what is the output of the following?

1. `z[z[, "Second"], ]`?
2. `z[, 1] != 1`?
3. `z[(z[, 1] != 1), 3]`?

# Lecture 3: Exploratory Data Analysis in R

## STAT GR5206

*Statistical Computing & Introduction to Data Science*

Cynthia Rush  
Columbia University

September 22, 2017

# COURSE NOTES

- ▶ Questions about lab: internet issues.
- ▶ Homework: due October 2.
- ▶ Ask questions on Piazza and answer others' questions.
- ▶ Knitting to .pdf: need a LaTex installation,  
[tug.org/begin.html](http://tug.org/begin.html).
- ▶ Otherwise, knit to .html and convert to .pdf.

# REVIEW

- ▶ **Filtering.** Accessing elements of a structure based on some criteria. `v[v>5], m[ m[,1] !=0, ]`.
- ▶ **NA and NULL values.** NA is missing data and NULL doesn't exist.
- ▶ **Factors and Tables.** Factors is how R classifies categorical variables.
- ▶ **Dataframes.** Used for data that is organized with rows indicating cases and columns indicating variables.
- ▶ **Importing and Exporting Data in R.** Use `read.csv()` and `read.table()` depending on dataset type. The working directory.

# CONTROLS STATEMENTS: LOOPS, WHILE, IF ELSE

# CONTROL STATEMENTS

- ▶ A **control statement** determines whether other statements will or will not be executed.
- ▶ Allows you to respond to different inputs or features of the data and execute different R expressions accordingly.
- ▶ In R, it is often the case that there exists other code to do the same thing, but is much faster and cleaner. Nevertheless, there are lots of times when control statements are the best option.
- ▶ Usually not used in interactive coding but sometimes needed when writing functions or longer expressions.

# CONTROL STATEMENTS

## Types of Control Statements

- ▶ A **loop** iterates over a statement a certain number of times.
  - ▶ **for loops** execute a controlled statement a fixed number of times.
  - ▶ **while loops** executes a controlled statement as long as a condition is met.
- ▶ An **if** statement gives a condition under which another statement is executed.
- ▶ An **if, else** statement decides which of two statements to execute based on a condition.

# FOR LOOPS

The basic structure of `for` loops is the following:

```
for (i in x) {  
  do something...  
}
```

The above statement,

- ▶ Increments a counter `i` along a vector `x`.
- ▶ Loops through the body of the statement (between `{` and `}`) until the counter runs through the vector.

One iteration of the loop for each component of `x` with `i` taking on the values of `x` in each iteration.

```
1st iteration: i = x[1]  
2nd iteration: i = x[2]  
so on...
```

## FOR LOOPS EXAMPLE

```
for (i in 1:10) {  
  print(i)  
}
```

```
1st iteration: i = 1,    print(1)  
2nd iteration: i = 2,    print(2)  
3rd iteration: i = 3,    print(3)  
...
```

## FOR LOOPS EXAMPLE

```
x <- c(5, 12, -3)
for (i in x) {
  print(i^2)
}
```

对于基本的一维的数据结构，for loop可以自己动循环

```
1st iteration: i = x[1] = 5,      print(25)
2nd iteration: i = x[2] = 12,     print(144)
3rd iteration: i = x[3] = -3,     print(9)
```

## FOR LOOPS EXAMPLE

```
x <- c(5, 12, -3)
for (index in 1:3) {
  print(x[index]^2)
}
```

```
1st iteration: i = 1,  x[1] = 5,      print(25)
2nd iteration: i = 2,  x[2] = 12,     print(144)
3rd iteration: i = 3,  x[3] = -3,    print(9)
```

## FOR LOOPS EXAMPLE

```
x <- matrix(1:9, ncol = 3)

for (i in 1:3) {
  for (j in 1:3) {
    print(x[i, j])
  }
}
```

```
1st iteration: i = 1, j = 1, x[i, j] = 1, print(1)
2nd iteration: i = 1, j = 2, x[i, j] = 4, print(4)
3rd iteration: i = 1, j = 3, x[i, j] = 7, print(7)
...
...
```

# FOR LOOPS

## Notes

- ▶ Body of a `for` loop can contain other `for` loops (called nesting) or other control statements.
- ▶ Can loop over any kind of vector regardless of mode.
- ▶ For example, could loop over file names to be scanned into R.

# WHILE LOOPS

The basic structure of `while` loops is the following:

```
while (condition) {  
    do something...  
}
```

The above loop,

Increments the controlled statement as long as the `condition` is `TRUE`.

- ▶ The `condition` must be a single logical value (`TRUE` or `FALSE`). It can't be a vector of values, for example.
- ▶ Note that this could loop forever.

## WHILE LOOPS EXAMPLE

Note that if the statement code is one line, we don't need braces.  
(Should probably use brackets every time, though.)

```
i <- 1
while (i <= 10) i <- i + 4
```

i

```
Beginning:    i = 1,      i <= 10 is TRUE
1st iteration: i = 5,      i <= 10 is TRUE
2nd iteration: i = 9,      i <= 10 is TRUE
3rd iteration: i = 13,     i <= 10 is FALSE
```

## WHILE LOOPS EXAMPLE

If you aren't careful, you can write infinite `while()`!

```
count <- 1
while (count <= 10) {
  count <- count - 1
}
```

|                |             |                     |
|----------------|-------------|---------------------|
| Beginning:     | count = 1,  | count <= 10 is TRUE |
| 1st iteration: | count = 0,  | count <= 10 is TRUE |
| 2nd iteration: | count = -1, | count <= 10 is TRUE |
| 3rd iteration: | count = -2, | count <= 10 is TRUE |
| ...            |             |                     |

# LOOPING SUMMARY

- ▶ Use `for` loops when the number of times to iterate is clear in advance.
- ▶ Use `while` when you can recognize the stopping point when you've arrived even if you don't know it beforehand.
- ▶ Note that all `for` loops can be written as `while` statements. (Can you show this?)
- ▶ `for` and `while` are examples of iteration: doing the same thing over and over. Usually there is a better way to do it!

# TRY ON YOUR OWN

- ▶ The following code creates a random walk with 100 steps, starting at 0. Note that the code `sample(c(-1, 1), size = 1)` samples one value from  $(-1, 1)$  with equal probability.

```
walk <- 0
for (i in 1:100) {
  step <- sample(c(-1, 1), size = 1)
  walk <- c(walk, walk[i] + step)
}
plot(walk, type = "l")
```

sample 函数：  
sample(c(样本框), size=抽样  
个数, replacement=FALSE  
(是否放回))

Your first task is to understand what the above code is doing. Note that you will get different plots each time you run it.

- ▶ Change the code so that your random walk runs until the walk goes outside the interval  $[-2, 2]$ .

```
walk=0;i=1
while(abs(walk[i])<2{
  step <- sample(c(-0.1,-0.2,0,0.1,0.2),size = 1)
  walk <- c(walk,walk[i]+step)
  i <- i+1}
plot(walk,type='l')
```

## IF, ELSE STATEMENTS

当程序结构是二元的，且希望输入与输出的数据均为向量时：使用ifelse函数

example :

```
grade <- ifelse(score>0.5,'passed','failed')
```

The basic structure of if, else statements is the following:

```
if (condition) {  
  do something...  
} else {  
  do something else...  
}
```

补充switch函数 ( 多元结构的ifelse ) :

```
mydate <- function(type='long'){  
  switch(type,  
    long = format(Sys.time(),'%A %B %Y')  
    short = format(Sys.time(),'%m-%d-%y')  
    cat(type,'is not a recognized type\n')  
  )  
}
```

The above statement,

Decides between different calculations according to some condition.

- The else clause is optional, which would mean if the condition is FALSE nothing is executed.
- Again, the condition must be provided a single logical value.

## IF, ELSE STATEMENTS EXAMPLE

```
for (i in seq(4)) {  
  if (i %% 2 == 0) {print(log(i))}  
  else {print("Odd")}  
}
```

|                |        |             |               |
|----------------|--------|-------------|---------------|
| Beginning:     | i = 1, | i %% 2 = 1, | print("Odd")  |
| 1st iteration: | i = 2, | i %% 2 = 0, | print(log(2)) |
| 2nd iteration: | i = 3, | i %% 2 = 1, | print("Odd")  |
| 3rd iteration: | i = 4, | i %% 2 = 0, | print(log(4)) |

## IF, ELSE STATEMENTS EXAMPLE

```
# Generate a random number between 0 and 10
x <- runif(1, min = 0, max = 10)

if(x > 3) {
  y <- 10
  print("x is greater than three")
} else {
  y <- 0
  print("x is less than three")
}
```

The value of y is set depending on whether  $x > 3$  or not.

# CHECK YOUR UNDERSTANDING

What is the value of `total`?

```
library(matlab)
total <- 0
for (i in 1:10) {
  if(isprime(i)) {
    total <- total + i
  }
}
```

这个函数的功能在与求和10以内的素数：  
\*`isprime ( )` 输入一个数字，返回1 ( TRUE ) 或0 ( FALSE )

# CHECK YOUR UNDERSTANDING

What is the value of `total`?

```
library(matlab)
total <- 0
for (i in 1:10) {
  if(isprime(i)) {
    total <- total + i
  }
}
```

`total`

How? Prime values are  $2 + 3 + 5 + 7 = 17$ .

# VECTORIZED OPERATIONS

# VECTORIZED OPERATORS

Where we can, we'd like to avoid iterations and use **vectorized operators**.

## Vectorized Operators

- ▶ Vectorized operators act on the whole object (vector or matrix, for example), with the operations occurring in parallel, instead of iterating over it.
- ▶ This is conceptually more clear, easier to read, and often more efficient.

# VECTORIZED OPERATIONS

Let's add two vectors: `u <- c(1,2,3)` and `v <- c(10, -20, 30)`.

Consider two ways to do this.

We could write a loop:

```
u <- c(1,2,3)      虽然没有具体的内容，但是提前设置好数据的一些属性是个好习惯，有一定的自我检测的功能
v <- c(10,-20,30)
c <- vector(mode = "numeric", length = length(u))
```

```
for (i in 1:length(u)) {
  c[i] <- u[i] + v[i]
}
c
```

# VECTORIZED OPERATORS

Let's add two vectors: `u <- c(1,2,3)` and `v <- c(10, -20, 30)`.

Consider two ways to do this.

We could use vectorized operators:

```
c <- u + v  
c
```

The second option is obviously more clear and concise.

# VECTORIZED CONDITIONS

The function `ifelse()` vectorizes conditional statements. It takes three arguments `ifelse(test, yes, no)`.

- ▶ `test` is a logical vector.
- ▶ `yes` is the return values when `test` is TRUE.
- ▶ `no` is the return values when `test` is TRUE.

# VECTORIZED CONDITIONS

A simplification in the previous example.

```
for (i in seq(4)) {  
  if (i %% 2 == 0) {print(log(i))}  
  else {print("Odd")}  
}  
  
ifelse(seq(4) %% 2 == 0, log(seq(4)), "Odd")
```

# VECTORIZED CONDITIONS

Sometimes there are surprises about what is fast and tricks for vectorizing things in unexpected ways.

A simplification in the previous example.

```
vals <- rnorm(1e6)
system.time(trunc <- ifelse(vals > 0, vals, 0))
system.time(vals <- vals * (vals > 0))

all(trunc == vals)          使用vectorized operation比使用loop更加有效率
identical(trunc, vals)
```

If you use a trick like this, having a comment in your code is a good idea.

# A NOTE ON PRE-ALLOCATING SPACE

This is slow.

```
vals <- 0
n     <- 50000
system.time({
  for(i in 1:n)
    vals <- c(vals, i)
})
```

This is slow and confusing.

```
vals <- 0
system.time({
  for(i in 1:n)
    vals[i] <- i
})
```

# A NOTE ON PRE-ALLOCATING SPACE

This is not so slow.

```
system.time({  
  vals <- rep(0, n)  
  for(i in 1:n)  
    vals[i] <- i  
})
```

Please ignore the obvious fact that `vals <- 1:n` is the correct way to code this.

# THE APPLY() COMMANDS

R has some built-in functions that implement looping in a compact form.

## The `apply()` Functions

- ▶ `apply()`: Apply a function over the rows or columns of a **matrix**.
- ▶ `lapply()`: Loop over a **list** and apply a function to each element.
- ▶ `sapply()`: Same as `lapply()`, but it **simplifies** the result.
- ▶ `tapply()`: Apply a function over subsets of a **vector**.
- ▶ `mapply()`: **Multivariate** version of `lapply()`.

# THE APPLY() COMMANDS

## apply() Example

Used to apply the same function to each row or column of a matrix. It's not really faster than writing a loop, but it works in one line and is highly compact.

```
mat <- matrix(1:12, ncol = 6)
mat
colSums(mat) # Recall colSums() from lab.

# Here's another way to do the same thing.

apply(mat, 2, sum)
```

# THE APPLY() COMMANDS

## The `apply()` Arguments

Use as `apply(X, MARGIN, FUN, ...)`

- ▶ `X` is a matrix (or dataframe).
- ▶ `MARGIN` is a 1 for the rows or a 2 for the columns.
- ▶ `FUN` is a function to be applied.
- ▶ `...` allows for additional arguments to be made to the `FUN` call.

# THE APPLY() COMMANDS

## apply() Example

```
mat
apply(mat, 2, sum)
apply(mat, 1, sum)

x <- matrix(rnorm(200), nrow = 20, ncol = 10)
apply(x, 2, mean)
apply(x, 1, mean)
apply(x, 2, quantile, probs = c(0.25, 0.75))
```

Note that `colSums()`, `colMeans()`, `rowSums()`, and `rowMeans()` are all functions that are useful shortcuts to some of the `apply()` examples we studied. (Besides being easier to understand they're also heavily optimized and are therefore much faster.)

# THE APPLY() COMMANDS

- ▶ **lapply()**, or **list apply**, works as follows:
  1. It loops over a list, iterating over each element in that list,
  2. It applies a function to each element of the list (a function that you specify),
  3. And returns a list.
- ▶ **sapply()**, or **simplified list apply**, works like **lapply()**, but returns a vector or matrix if possible.

Use [R Help](#) for more info!

Look up further details of the **apply()** function and its variants using  
`?apply`.

# TAKING THE MEAN OF ELEMENTS OF A LIST

## Example

```
vec1 <- c(1.1, 3.4, 2.4, 3.5)
vec2 <- c(1.1, 3.4, 2.4, 10.8)

not_robust <- list(v1 = vec1, v2 = vec2)
not_robust

lapply(not_robust, mean)
sapply(not_robust, mean)

# The sample mean is not robust!

unlist(lapply(not_robust, median))
```

# EXAMPLE: HONOR CODE TEXT DATA

Recall, the Honor Code Text example.

List in Alphabetical Order

```
HC <- scan("HonorCode.txt", what = "")  
head(HC, 15)  
HC <- factor(HC, levels = unique(HC))  
  
findwords <- function(text_vec){  
  words <- split(1:length(text_vec), text_vec)  
  return(words)  
}  
wl <- findwords(HC)  
wl[1:3]
```

这个函数利用split计算出了每一个不同的单词出现的位置，返回的是一个list。names(list) = unique(HC), 每一个element下面的元素是这个单词出现的序号

Let's now sort the words by their frequency.

## EXAMPLE: HONOR CODE TEXT DATA

We use `sapply()` to find the length of each element in our word list. Since the elements are the words, the length is the frequency.

### List in Frequency Order

```
freq_list <- function(wordlist) {  
  freqs <- sapply(wordlist, length) # The frequencies  
  return(wordlist[order(freqs)])  
}  
  
head(sapply(wl, length))  
  
head(freq_list(wl), 3)  
tail(freq_list(wl), 3)
```

`sapply`返回的是一个数值型向量，每一个位置表示所对应的单词出现过的次数。但是这个向量有名字。`names(freq) = unique(HC)`

The `order()` function returns a vector of indices that will permute its input argument into sorted order. Check out `?order`.

# FUNCTIONS ON FACTORS

补充数据排序(order 函数) :

```
newdata <- olldata[order(olldata$colname1 ,  
-olldata$colname2,...),]
```

\*前面加‘ - ’ 表示降序

- ▶ Factors have their own member of the `apply()` family: `tapply()`.
- ▶ Use as follows, `tapply(X, INDEX, FUN)`, to apply a function to subsets of the vector `X`.
- ▶ The above splits the vector `X` into groups according to the levels of the factor (or factors) `INDEX` and then applies the function `FUN` to each group.

order函数返回的是从小到大的元素所对应的序列号，而不是元素内容本身：

```
A <- c(1,3,5,2,6,4,10,7,23) ==order(A)==> 1 4 2 6 3 5 8 7 9
```

ordered是对数据转化为有序型的factor 变量：

```
A <- c(1,3,5,2,6,4,10,7,23) ==ordered(A)==>
```

```
1 3 5 2 6 4 10 7 23
```

```
9 Levels: 1 < 2 < 3 < 4 < 5 < 6 < 7 < ... < 23
```

sort是对数据本身进行排序

```
A <- c(1,3,5,2,6,4,10,7,23) ==sort(A)==> 1 2 3 4 5 6 7 10 23
```

Cynthia Rush : GR5206

# FUNCTIONS ON FACTORS

## tapply() Example

```
# Calculate the average age in each group.  
vec      <- c(rnorm(10), runif(10), rnorm(10, mean = 1))  
groups <- factor(c(rep(1, 10), rep(2, 10), rep(3, 10)))
```

```
tapply(vec, groups, mean)  
tapply(vec, groups, range)
```

这里的group相当于在apply里面指定了是行还是列。最后的结果是按照groups为标准的输出不同groups下面的函数结果

# EXPLORATORY DATA ANALYSIS AND R GRAPHICS

# DIAMONDS DATASET

- ▶ Download `diamonds.csv` from the Canvas page (Homepage, Week 3, Slides).
- ▶ Save to your computer and set your working directory to match that location.
- ▶ Run `diamonds <- read.csv("diamonds.csv", as.is = TRUE)`.

```
diamonds      <- read.csv("diamonds.csv", as.is = T)
diamonds$cut    <- factor(diamonds$cut)
diamonds$color   <- factor(diamonds$color)
diamonds$clarity <- factor(diamonds$clarity)
```

# DIAMONDS DATASET

Info on ~ 54000 diamonds from [www.diamondse.info](http://www.diamondse.info).

## Variables

- ▶ **Carat** – Weight of the diamond (0.2 - 5.01).
- ▶ **Color** – Diamond color from J (worst) to D (best).
- ▶ **Clarity** – A measurement of how clear the diamond is (I1 (worst), SI1, SI2, VS1, VS2, VVS1, VVS2, IF (best)).
- ▶ **Cut** – Quality of the cut (Fair, Good, Very Good, Premium, Ideal).
- ▶ **Price** – Price in US dollars.

# DIAMONDS DATASET

Code example.

# CHECK YOURSELF

## Questions:

1. Think by yourself for a few minutes: what are some interesting questions we could answer using this dataset?
2. Use `tapply()` to find the average price for diamonds grouped according to the factor `cut`. Now group according to the fact `color`. Do these results make sense?

```
tapply(diamonds$price,diamonds$cut,mean)
```

|          | Fair     | Good     | Ideal    | Premium |
|----------|----------|----------|----------|---------|
| 4358.758 | 3928.864 | 3457.542 | 4584.258 |         |

|          | Very Good |  |
|----------|-----------|--|
| 3981.760 |           |  |

```
tapply(diamonds$price,diamonds  
$color,mean)
```

|          | D        | E        | F | G | H |
|----------|----------|----------|---|---|---|
| 3169.954 | 3076.752 | 3724.886 |   |   |   |
| 3999.136 | 4486.669 |          |   |   |   |
|          | I        | J        |   |   |   |
| 5091.875 | 5323.818 |          |   |   |   |

# CHECK YOURSELF

## Questions:

1. Think by yourself for a few minutes: what are some interesting questions we could answer using this dataset?
2. Use `tapply()` to find the average price for diamonds grouped according to the factor `cut`. Now group according to the fact `color`. Do these results make sense?

## Some question ideas:

- ▶ What does the distribution of diamond prices look like?  
Symmetric? Skewed?
- ▶ How does a diamond's price relate to its weight?
- ▶ Does the relationship between the price and the weight change depending on the quality of the diamond's cut?

```
ggplot(data=diamonds)+  
  geom_point(aes(y=diamonds$price,x=diamonds$carat))+  
  facet_wrap(~diamonds$cut,nrow = 3,scales = 'free')
```

# EXPLORATORY DATA ANALYSIS <sup>1</sup>

**Exploratory Data Analysis**, or EDA for short, is exploring data in a systematic way.

It's an iterative process:

1. Generate questions about your data.
2. Search for answers by visualizing, transforming, and modelling your data.
3. Use what you learn to refine your questions and or generate new questions.

---

<sup>1</sup>EDA slides developed from G. Grolemund and H. Wickham.

# EXPLORATORY DATA ANALYSIS

EDA is a way for you to learn about and better understand your data.

## Asking Questions

1. What type of **variation** occurs **within** my variables?
2. What type of **covariation** occurs **between** my variables?

We focus on each of these questions separately. Today we will use base R's graphics tools, later in the class we'll learn more advanced (and more fancy) ways to visualize data.

# VARIATION

**Variation** is the tendency of measured values of a variable to change measurement-to-measurement.

- ▶ Visualizing the distribution of a variable is the best way to understand the patterns of a variable's variance.
- ▶ Visualize the distribution of a **categorical** variable using a bar graph.
- ▶ Visualize the distribution of a **continuous** variable using a histogram.

# PLOTTING IN R

补充常用图形参数 ( par ) :

1. lty : 线的类型
2. pch : 点的类型
3. cex : 指定大小 ( 符号 )
4. lwd : 指定线条宽度
5. par(mfrow=c(2,2)) 指定分图数目

颜色 ( col ) :

1. col.axis 坐标刻度文字颜色
2. col.lab 坐标轴标签文字的颜色
3. col.main 主标题的文字颜色
4. col.sub 副标题的文字颜色
5. fg 前景色
6. bg 背景色

文本 ( cex ) :

1. cex 指定大小
2. cex.axis;cex.lab;cex.main;cex.sub
3. font 输入整数, 字体样式
4. font.axis;font.lab;font.main;font.sub
5. ps字体的磅值
6. family 字体族

The general plot call in R looks like the following:

```
plot(x = , y = , ...)
```

Additional parameters can be passed to customize the plot:

- ▶ type: scatter plot? lines? etc
- ▶ main: a title
- ▶ xlab, ylab: x-axis and y-axis labels
- ▶ col: color, either a string with the color name or a vector of color names for each point

图形的边界与尺寸

1. pin 图形的长与高
2. mai 下左上右的边界大小 ( 英寸 )
3. mar 下左上右的边界大小 ( 英分 )
4. xlim ; ylim

More layers can be added to the plot with additional calls to **lines**, **points**, **text**, etc. (We'll look at examples.)

text ( location , 内容 ( 字符型变量 ) , pos(下左上右) )

图形的图例 :

legend ( location , title , legend ( 字符型变量 ) )

# BAR GRAPHS IN R

Produce a bar graph using `barplot(heights, labels)` where `heights` is a vector of values for the heights of each bar and `labels` is an optional vector of labels for each bar.

补充：1. `barplot`有一个`horiz=TRUE`的属性，用于设置是否需要反转图像

- ▶ Can use `table()` as input for the bar heights.
- ▶ The order that `table()` uses is the order of the factor levels of its input.

利用`table`绘制关于factor下不同类型内容的频度图

## Plotting a Bar graph of Diamond Cut

```
table(diamonds$cut)  
names(table(diamonds$cut))
```

可以做多个数量关系之间的图：  
`A = table ( factor1 , factor2 )`  
`barplot ( A , beside=FALSE )`

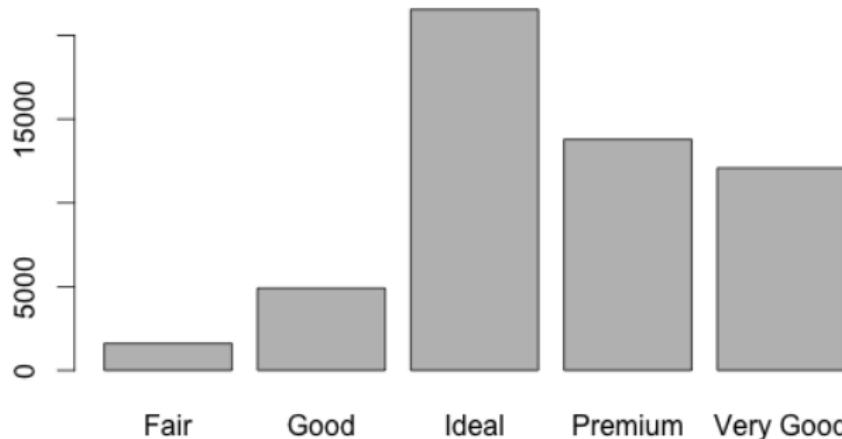
`beside`用于判断是不是需要堆砌额结果

```
barplot(height = table(diamonds$cut),  
        names.arg = names(table(diamonds$cut)))
```

# BAR GRAPHS IN R

## Plotting a Bar graph of Diamond Cut

```
barplot(height = table(diamonds$cut),  
        names.arg = names(table(diamonds$cut)))
```



Oops! The order should be Fair, Good, Very Good, Premium, Ideal.

# BAR GRAPHS IN R

## Plotting a Bar graph of Diamond Cut

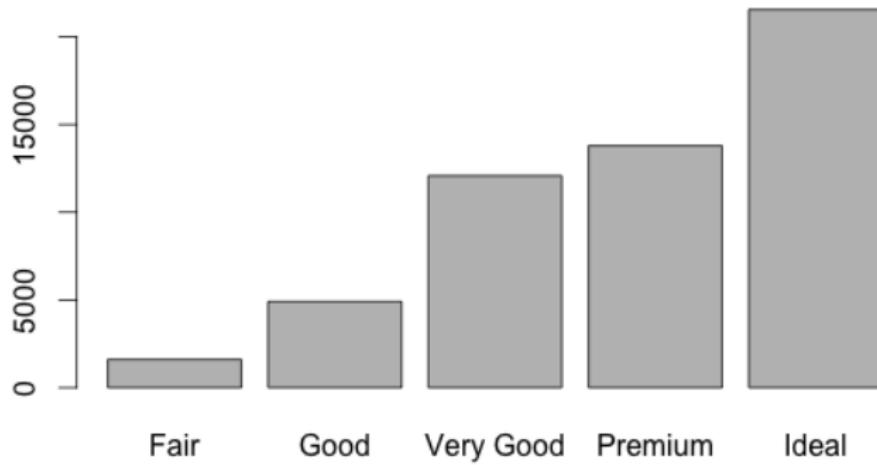
```
levels(diamonds$cut)
diamonds$cut <- factor(diamonds$cut, level = c("Fair",
                                                 "Good", "Very Good", "Premium",
                                                 "Ideal"))
levels(diamonds$cut)          自己指定对应的出现顺序
                                画图问题先改数据

barplot(height = table(diamonds$cut),
        names.arg = names(table(diamonds$cut)))
```

# BAR GRAPHS IN R

## Plotting a Bar graph of Diamond Cut

```
barplot(height = table(diamonds$cut),  
        names.arg = names(table(diamonds$cut)))
```



# HISTOGRAMS IN R

Produce a histogram using `hist(values)` where `values` is a vector of values.

- ▶ A histogram divides the x-axis into equally-spaced bins with the height of the bars used to indicate the number of observations falling within the bin.
- ▶ Change the width of the histogram's bins using `break =` argument. This specifies the number of bins.
- ▶ Make sure to explore different bin widths as different widths will display different patterns in the data.

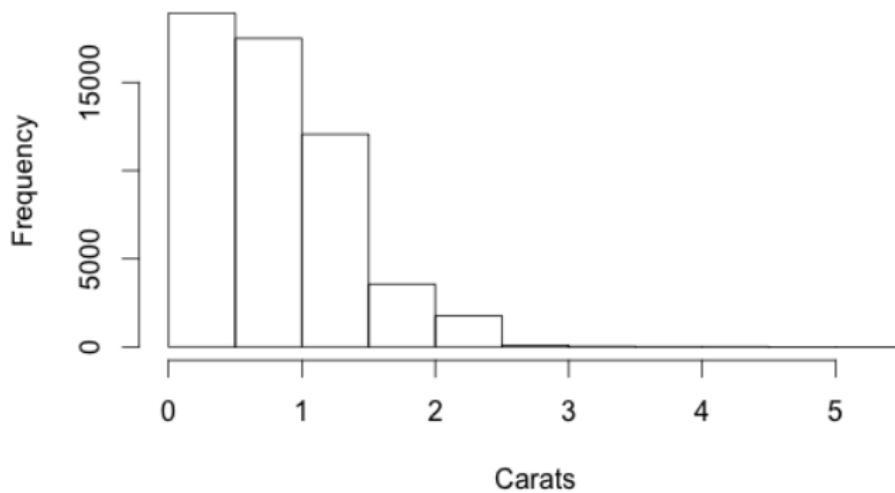
# HISTOGRAMS IN R

## Plotting a Bar graph of Diamond Cut

```
hist(diamonds$carat, main = "Histogram of Carats",  
      xlab = "Carats")
```

补充 : rug (x)

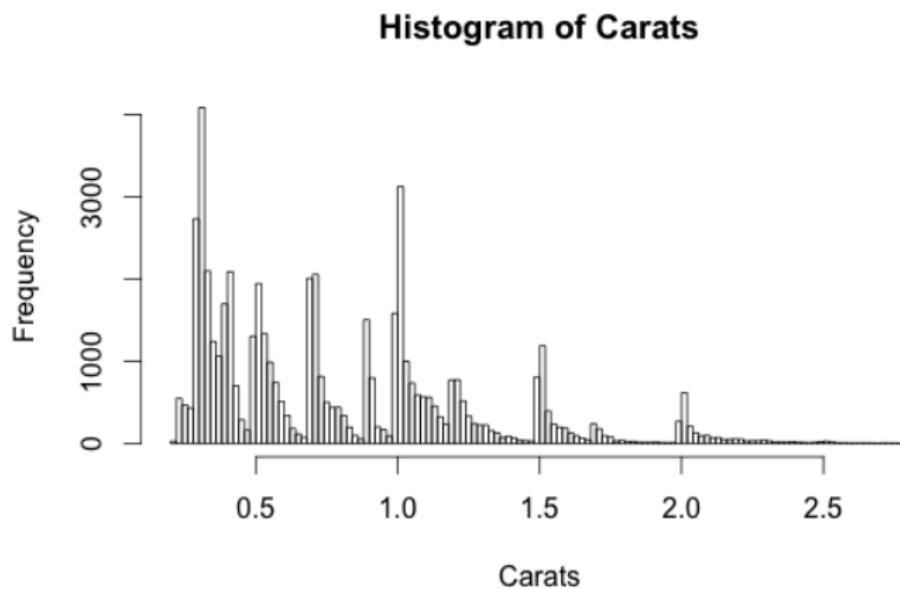
**Histogram of Carats**



# HISTOGRAMS IN R

## Plotting a Bar graph of Diamond Cut

```
hist(diamonds$carat[diamonds$carat < 3], breaks = 100,  
      main = "Histogram of Carats", xlab = "Carats")
```



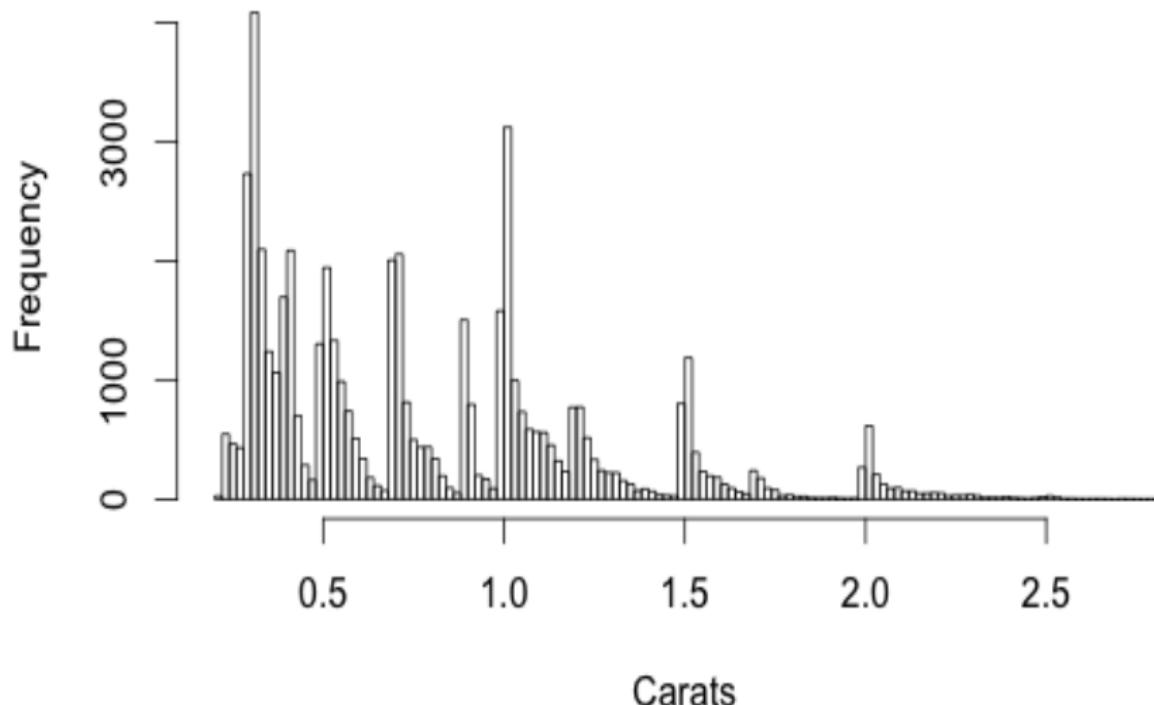
# VISUALIZING VARIATION

What should we be looking for in these plots?

- ▶ Use the plots to create new questions.
  - ▶ What do you want to learn more about?
  - ▶ Are there any interesting patterns I want to explore.
- ▶ Use the plots to better understand the data.
  - ▶ Do these plots match my expectations? Why or why not?
  - ▶ Do the data cluster in interesting ways?
  - ▶ What are the typical values? Outliers? Why?
  - ▶ How could this be misleading?

# VISUALIZING VARIATION

## Histogram of Carats



# COVARIATION

**Covariation** is the tendency for the values of two or more variables to vary together in a related way.

- ▶ Visualizing the relationship between variables is the best way to spot covariation.
- ▶ Visualize the distribution of a **categorical** variable and a **continuous** variable using a boxplot.
- ▶ Visualize the distribution of two **continuous** variables using a scatter plot.

# BOXPLOTS IN R

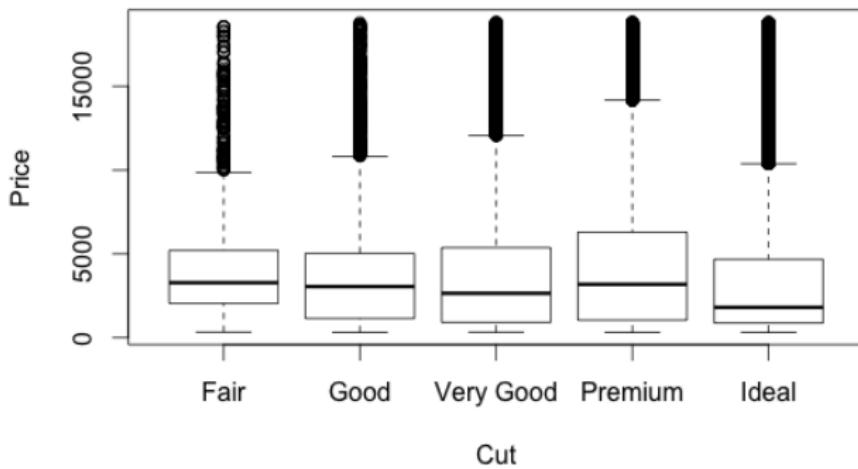
Produce a boxplot (box-and-whisker plot) using `boxplot(values ~ group)` where `values` is a vector of data to be split according to `group`.

- ▶ The box stretches from the  $25^{th}$  percentile of the distribution to the  $75^{th}$  percentile (the IQR).
- ▶ The line in the middle is the median.
- ▶ The ‘whiskers’ extend to 1.5 times the IQR on either end.

# BOXPLOTS IN R

## Plotting a Boxplot of Diamond Price by Cut

```
boxplot(price ~ cut, data = diamonds, ylab = "Price",  
       xlab = "Cut")
```



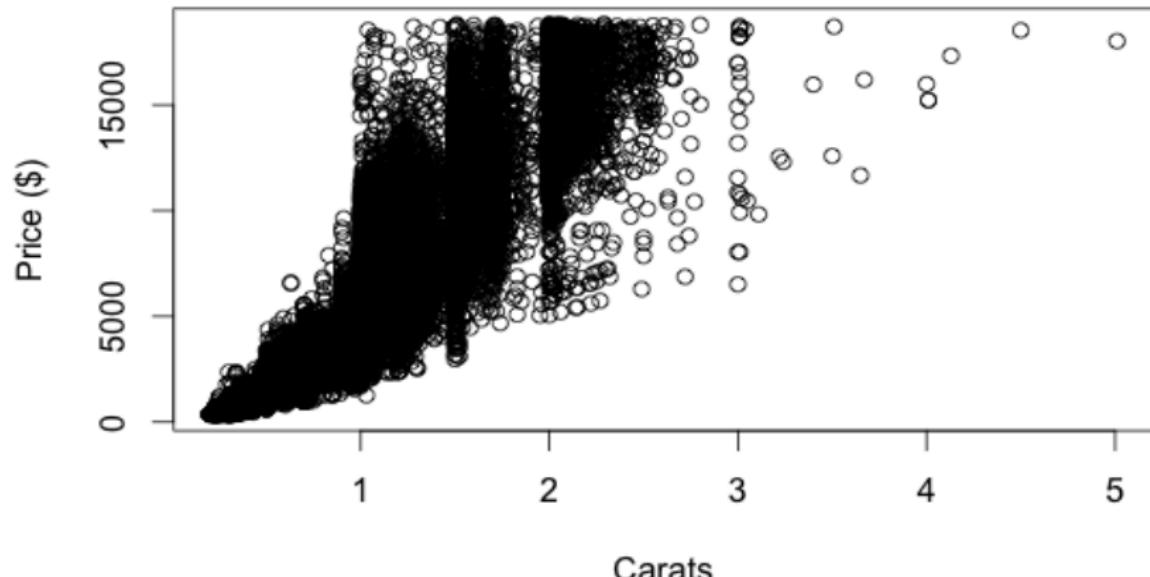
## SCATTERPLOT IN R

Produce a scatter plot using `plot(x,y)` where `x` is a vector of x-values and `y` a vector of y-values.

# SCATTERPLOT IN R

Plotting a Scatterplot of Diamond Price vs. Carat

```
plot(diamonds$carat, diamonds$price, xlab = "Carats",  
      ylab = "Price ($)")
```



# VISUALIZING COVARIATION

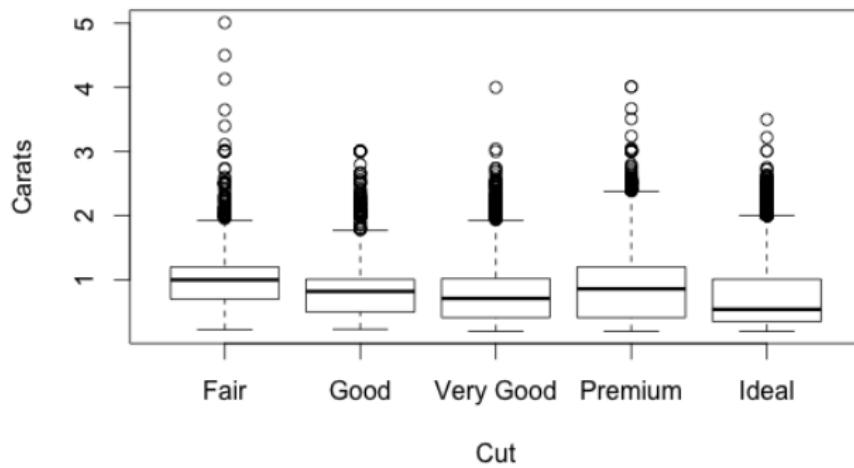
If a relationship exists between two variables it will show up as patterns in your plots.

Ask yourself the following questions.

- ▶ Is that pattern random (due to chance)?
- ▶ What relationship does the pattern imply?
- ▶ Is the relationship strong, weak, linear, non-linear, etc.?
- ▶ What other variables might affect the relationship?
- ▶ Does the relationship change if you look at individual subgroups of the data?

# BOXPLOTS IN R

```
boxplot(carat ~ cut, data = diamonds, ylab = "Carats",  
       xlab = "Cut")
```



# OFTEN VISUALIZATION ISN'T ENOUGH

- ▶ Difficult to understand the relationship between `price` and `cut` because `price` and `carat` and `carat` and `cut` are also related.
- ▶ Here we would need to use a model (for example, linear models) to consider all these relationships simultaneously.

# MOVING ON FROM BASE R GRAPHICS

## Pros/Cons

- ▶ Good for exploratory data analysis and sanity checks
- ▶ Syntax is inconsistent across functions: some take `x`, `y` while others take formulas
- ▶ Defaults plotting parameters are ugly and it can be difficult to customize

## Up Next:

- ▶ We will learn about more advanced plotting tools using the `ggplot2` package soon.
- ▶ `ggplot2` provides more sophisticated graphing tools for communicating your results with standardized syntax.
- ▶ Note that base graphics are object-oriented (they recognize the object type and plot accordingly), while `ggplot2` generally is not.

# CHECK YOURSELF

## Question

- ▶ Read in the `all_medalists.csv` file containing all Summer Olympic medalists from 1896-2008.
- ▶ Make a new dataframe holding only the rows corresponding to gold medals (in variable `Medal`) won by the United States (in variable `Country`).
- ▶ Use the `table()` command to count the number of gold medals won by the US each year.
- ▶ Plot the number of gold medals won against the year.

# CODING STANDARDS

Coding standards aren't universal and are often highly debated. Here are some of my suggestions and the rationale behind them.

- ▶ **Always use scripts or RMarkdown.** Using text files and a text editor is fundamental to coding. (If you're writing your code in an editor like Microsoft Word, you need to stop.)
- ▶ **Indent your code.** This is very important for the readability of your code. Some programming languages actually require it as part of their syntax, but R does not. I indent two spaces whenever I'm inside a loop or a function – you'll see!
- ▶ **Limit the width of your code.** You don't want the code you write to extend beyond the right hand side of your page. This limitation, along with indentation, forces you to write code that is clean, readable, and naturally broken down into modular units
- ▶ **Limit the length of individual functions.** If you are writing functions, it's usually a good idea to not let your functions run for pages and pages. Typically, purpose of a function is to execute one activity or idea. If your function is doing lots of things, it probably needs to be broken into multiple functions.

# Lecture 4: Character Strings, Regular Expressions, and Web Scraping.

STAT GR5206 *Statistical Computing & Introduction to Data Science*

Cynthia Rush  
Columbia University

September 29, 2017

# COURSE NOTES

- ▶ Questions about lab.
- ▶ Homework due on Monday.
- ▶ Midterm coming up soonish. Let me know ahead of time if you have a time conflict – no makeups.
- ▶ Knitting your Markdown file. (Working directory, LaTex, Syntax)

# REVIEW

- ▶ **Control Statements:** `for` loops, `while` loops, and `ifelse`.
- ▶ **Vectorized Operations:** Using vectorized operators when possible, the `apply()` family.
- ▶ **Exploratory Data Analysis:** Helps you to better understand your data and be able to ask good questions.
- ▶ **Base R Graphics.** Plotting with `hist()`, `plot()`, `barplot()`, etc.

# CHARACTER STRINGS AND STRING OPERATIONS

# TEXTUAL DATA

Analyzing textual data is common in machine learning and data science.

## Textual Data Sources

- ▶ Classifying and analyzing tweets from Twitter.
- ▶ Answering, does this email belong in the spam filter?
- ▶ Processing and comparing survey responses.
- ▶ Working with character data such as names, birthdays, or addresses.

R contains many functions for manipulating character data a few of which we study today.

# DEFINITION

- ▶ A **character** is a symbol in a written language - anything you can enter on a keyboard.  
Examples: 'Q', '\*', '+', 'd', 'x', ' ', '{' etc.
- ▶ A **string** is a sequence of characters.  
Examples: 'Columbia University', 'cat, squirrel, hedgehog', etc.

# DEFINITION

- ▶ A **character** is a symbol in a written language - anything you can enter on a keyboard.  
Examples: 'Q', '\*', '+', 'd', 'x', ' ', '{' etc.
- ▶ A **string** is a sequence of characters.  
Examples: 'Columbia University', 'cat, squirrel, hedgehog', etc.
- ▶ Both are type **character** in R.  

```
mode("d")
mode("cat, squirrel")
```
- ▶ Both can go into scalars, vectors, matrices, lists, or dataframes.
- ▶ In R strings are denoted by quotation marks.

# WHITESPACE

As noted above, whitespace ' ' is considered a character and multiple spaces ' ' a string.

```
class("d")
class("cat, squirrel")
```

```
class(" ")
nchar(" "); nchar(" "); nchar("")
```

nchar( ) 用于数字符串内  
元素的个数

```
class("\n"); class("\\""); class("\t")
nchar("\n"); nchar("\\""); nchar("\t")
```

## Special Characters

- ▶ Quotes within a string: \" 在cat 的函数中，在print中还是显示\t
- ▶ Tab: \t
- ▶ New Line: \n

# STRINGS AS ELEMENTS OF A VECTOR

If strings are elements of an object,

- ▶ `length()` reports the number of strings in the object, not the number of characters in the string.
- ▶ `nchar()` reports the number of character values in a string.
- ▶ `nchar()` is vectorized, like most R functions.

```
length("cat, squirrel, hedgehog")
length(c("cat", "squirrel", "hedgehog"))
nchar("cat, squirrel, hedgehog")
nchar(c("cat", "squirrel", "hedgehog"))
```

# PRINTING STRINGS

- ▶ Can be displayed when their name is typed or using `print()`.
- ▶ Often want to use `cat()` to print character strings directly.
- ▶ `cat()` coerces its argument to strings, so can be useful when printing warnings.

```
print("cat, squirrel")
cat("cat, squirrel")
x <- 6
y <- 7
cat("I have", x, "cats and", y, "hedgehogs as pets.")
print("I have", x, "cats and", y, "hedgehogs as pets.")
```

# PRINTING STRINGS

```
print("cat, \n squirrel")
cat("cat, \nsquirrel")
print("In R, an \"array\" is a multi-dimension matrix.")
cat("A group of hedgehogs is called an \"array\".")
```

# CHECK YOURSELF

## Task

Use `print()` and `cat()` to print the following in R. What is the difference?

```
"Columbia\tUniversity"
```

How many characters are in the above? Why?

# SUBSTRINGS

- ▶ A **substring** is a smaller string taken from a larger string, but it is itself still a string.
- ▶ Note that we can't use regular subsetting options like `[[[]]]` or `[]` because a string isn't a vector or list.
- ▶ The `substr()` function can extract or change values of parts of strings.

# SUBSTRINGS

The call `substr(string, start = , stop = )` returns the substring from character position `start` to `stop` in the given `string`.

```
phrase <- "Christmas Bonus"  
substr(phrase, start = 8, stop = 12)  
substr(phrase, start = 13, stop = 13) <- "g"  
phrase  
  
fav_animals <- c("cat", "squirrel", "hedgehog")  
substr(fav_animals, start = 1, stop = 2)  
substr(fav_animals, nchar(fav_animals)-1, nchar(fav_animals))  
substr(fav_animals, start = 4, stop = 4)
```

自动对c (...) 中的每一个字符串用相同的substr规则

# DIVIDING STRINGS INTO VECTORS

`strsplit(string, split =)` divides its input `string` at the appearances of the pattern passed to `split`.

```
todo <- "Lecture, Lab, Homework"  
  
strsplit(todo, split = ",")  
strsplit(todo, split = ", ")  
  
strsplit(c(todo, "Midterm, Final"), split = ",")
```

# CHECK YOURSELF

Tasks      `lum_vector <- c("columbia", "slumber party", "sugarplum")  
substr(lum_vector,start= c(3,2,7),stop = c(5,4,9))`

- ▶ Make a vector of three elements which are “Columbia”, “slumber party”, and “sugarplum”. Make a call to `substr()` that returns the “lum” from each element of the vector. The output should be  
"lum" "lum" "lum"

- ▶ Use `strsplit()` on the vector you created splitting on “lum”. Output should be a list of length three.

`strsplit(lum_vector,split = 'lum')`

# BUILDING STRINGS FROM MULTIPLE PARTS

`paste()` combines strings into one long string and is very flexible.

```
paste("cat", "squirrel", "hedgehog")
paste("cat", "squirrel", "hedgehog", sep = ", ")

animals <- c("cat", "squirrel", "hedgehog")

paste(animals, 1:3) [1] "cat 1"    "squirrel 2" "hedgehog 3"
paste(animals, 1:2) [1] "cat 1"    "squirrel 2" "hedgehog 1"
paste(animals, "(, 1:3, )") [1] "cat ( 1 )"  "squirrel ( 2 )" "hedgehog ( 3 )"
paste(animals, "(, 1:3, )", sep = "")
paste(animals, " (, 1:3, )", sep = "")
paste(animals, " (, 1:3, )", sep = "", collapse = "; ")
```

# CHECK YOURSELF

## Task

Use `paste()` with its first input being `c("Columbia", "slumber party", "sugarplum")` along with the `sep` and `collapse` arguments to create the following string:

`"Columbia [3-5] ; slumber party [2-4] ; sugarplum [7-9]"`.

```
words <- c('Columbia ','slumber party ','sugarplum ')
paste(words,'[',c(3,2,7),'-',c(5,4,9),']',sep=' ',collapse =';')
```

# SEARCHING STRINGS

## Honor Code Example

The file "HonorCode.txt" contains Columbia University's Honor Code:

“Students should be aware that academic dishonesty (for example, plagiarism, cheating on an examination, or dishonesty in dealing with a faculty member or other University official) or the threat of violence or harassment are particularly serious offenses and will be dealt with severely under Dean's Discipline...”

# SEARCHING STRINGS

## Honor Code Example

```
HC  <- readLines("HonorCode.txt")  
HC2 <- scan("HonorCode.txt", what = "")
```

返回的结果是句子  
返回的结果是单词

```
length(HC)  
head(HC, 5)
```

HC is a vector with one element per line of text in the Honor Code.

# SEARCHING STRINGS

The `grep(pattern, x)` function searches for a specified substring given by `pattern` in a vector `x` of strings.

## Honor Code Example

```
grep("students", HC)
grep("Students", HC)
head(grep1("students", HC), 15)
```

`grep()` 输入的第一个参数是需要找的字符串，第二个参数是查找的范围，返回的是这个字符串在范围内出现位置的序号。

`grep1` returns a logical vector (match or not for each element of x)

```
grep("students", HC)
HC[grep("students", HC)]
```

返回students这个单词所出现的句子

```
HC2[grep('students', HC2)]
[1] "students" "students" "students" "students" "students" "students" "students"
```

# SEARCHING STRINGS

Using functions we've learned today, let's make HC a vector with each element a word of the Honor Code (instead of a line of text). Of course, could do this with `scan()`.

## Honor Code Example

```
HC      <- paste(HC, collapse = " ") # One long string
HC
HC.words <- strsplit(HC, split = " ")[[1]] # List output
head(HC.words, 10)                          [[1]]表明锁定到了单词上，每一个单词都是对应的元素，HC.words,是list里面的每一个单词组成的vector,head(10)返回的是前十个单词没有[[1]] HC.words还是一个list，head ( 10 )会返回全部结果
```

# SEARCHING STRINGS

We can count words using `table()`.

## Honor Code Example

```
word_count <- table(HC.words)
word_count <- sort(word_count, decreasing = TRUE)
head(word_count, 10)
```

`word_count2 <- table(HC2)`

结果是一样的，前面的一系列步骤和直接通过  
`scan`而不是`read.line`读取数据的效果是一样的

This is much more simple than the strategy we used a few weeks ago  
to produce the same result!

Some undesirable things are happening here...

- ▶ The null string "" is the seventh most common word.
- ▶ Punctuation and capitalization are messing up our counts.  
("students" vs. "Students" before).

# SEARCHING STRINGS

Some undesirable things are happening here...

```
head(word_count, 10)  
tail(word_count, 10)
```

# CHECK YOURSELF

## Task

Use `grep()` to search over `names(word_count)` to find the number of words in the word count vector that have semi-colons in them.

Hint: ";" should be one of your arguments to grep. Using your result print the words that `grep()` finds.

```
# grep : search for patterns in a string
semicolons <- grep(";",names(word_count))
names(word_count)[semicolons]
```

# `gsub(pattern, replacement, x)` function searches for a specific substring given by pattern in a vector x of strings and replaces it with the substring specified by replacement.

```
names(word_count)<-gsub(";", "",names(word_count))
names(word_count)[semicolons]
```

# SUBSTITUTING STRINGS

The `gsub(pattern, replacement, x)` function searches for a specified substring given by `pattern` in a vector `x` of strings and replaces it with the substring specified by `replacement`.

## Honor Code Example

```
semicolons <- grep(";", HC)
HC[semicolons]
```

```
HC <- gsub(";", "", HC)
HC[semicolons]
```

# FUNCTIONS FOR CHARACTER DATA

## Next

We want to search for text patterns instead of text constants. We can do this with regular expressions.

## Summary

- ▶ `nchar()`: Finds the length of a string.
- ▶ `substring()`: Extracts substrings and substitutes.
- ▶ `strsplit()`: Turns strings into vectors.
- ▶ `paste()`: Turns vectors into a string.
- ▶ `grep()`: Search for patterns in a string.
- ▶ `gsub()`: Replaces patterns in a string with another string.

# REGULAR EXPRESSIONS

# WHY DO WE NEED REGULAR EXPRESSIONS

```
fav_animals <- "cat,squirrel, hedgehog,    octopus"  
  
strsplit(fav_animals, split = ",")  
strsplit(fav_animals, split = " ")  
strsplit(fav_animals, split = ", ")
```

Need to split the entries by a commma and *optionally* some space.

# WHY DO WE NEED REGULAR EXPRESSIONS

It's not just annoying we can't do these things, there are a lot of examples where we have to do such manipulations.

- ▶ When scraping data from a webpage, will need to get rid of formatting instructions buried in the source of the webpage.
- ▶ For example, names may be preceded by titles such as Mr., Mrs., or Dr. that we aren't interested in using.

# REGULAR EXPRESSIONS

- ▶ **Regular expressions** are a method of expressing patterns in character strings.
- ▶ Used to match sets of strings or patterns of strings in R.
- ▶ Can express ideas like match “*this* and then *that*”, “either *this* or *that*”, “*this* repeated some number of times”.
- ▶ **Regular expressions** are rules expressed in a grammar with special symbols.

# RULES FOR REGULAR EXPRESSIONS

1. Every string is a regular expression.
  - ▶ "cat" matches "categorize" and "dogs and cats".
  - ▶ "cat" does not match "Dog is man's best friend" and "work doggedly".
2. Can represent OR with a vertical bar |.
  - ▶ "cat|dog|Dog" matches all of the above.
3. Precede special characters like | with a backslash \ to match exactly.
  - ▶ "A\\|b" matches "P(A|b)".
  - ▶ "A|b" matches twice in "Alabama" and twice in "blueberry".

\\在此做转义符，因为“|”已经有或的意思了，前面加上\\是他原有的意思重现。

# RULES FOR REGULAR EXPRESSIONS

When I say 'matches', I mean in R:

```
grep("cat|dog", c("categorize", "work doggedly")) [1] 1 2
grep("A|b", c("Alabama", "blueberry", "work doggedly")) [1] 1 2
grep("A\\|b", c("Alabama", "blueberry", "work doggedly", "P(A|b)
[1] 4
```

# RULES FOR REGULAR EXPRESSIONS

1. Indicate sets of characters with brackets [].
  - ▶ "[a-z]" matches any lower case letters.
  - ▶ "[[:punct:]]" matches all punctuation marks.
2. The caret ^ negates a character range when in the leading position. **^ 有取反的意思，就是取[]内字符集不属于的部分**
  - ▶ "[^aeiou]" matches any characters except lower-case vowels.
3. The period . stands for any character and doesn't need brackets.
  - ▶ "c..s" matches "cats", "class", "c88s", "c s", etc.  
**一个点表示省略一个，两个点表示省略两个，以此类推**

```
> grep('c.s', c("cats", "class", "c88s", "c s"))
[1] 4
> grep('c..s', c("cats", "class", "c88s", "c s"))
[1] 1 2 3
> grep('c...s', c("cats", "class", "c88s", "c s"))
[1] 2
> grep('c....s', c("cats", "class", "c88s", "c s"))
[1] 0
> grep('c..s', c('c s'))
[1] 0
> grep('c..s', c('c s'))
[1] 1
```

# RULES FOR REGULAR EXPRESSIONS

Quantifiers can be used to tell “how often” an expression occurs.

| Quantifier | Description (Match if the expression is ...) |
|------------|--|
| +          | Repeated one or more times.                  |
| *          | Repeated zero or more times.                 |
| ?          | Repeated zero or one times.                  |
| {n}        | Repeated exactly n times.                    |
| {n, }      | Repeated n or more times.                    |
| {n, m}     | Repeated between n and m times.              |

## Notes

- ▶ Quantifiers apply to the last character before they appear.
- ▶ Any valid expression can be enclosed in parentheses for grouping.

如果表示几个字符作为一个整体一起被重复，将他们放在一个括号内  
如果不放在一个括号内，那么就只有最后一个字符会被重复

# RULES FOR REGULAR EXPRESSIONS

Quantifiers can be used to tell “how often” an expression occurs.

| Quantifier | Description (Match if the expression is ...) |
|------------|--|
| +          | Repeated one or more times.                  |
| *          | Repeated zero or more times.                 |
| ?          | Repeated zero or one times.                  |
| {n}        | Repeated exactly $n$ times.                  |
| {n, }      | Repeated $n$ or more times.                  |
| {n, m}     | Repeated between $n$ and $m$ times.          |

## Examples

- ▶ "[0-9] [0-9] [a-zA-Z] +" matches strings with two digits followed by one or more letters.
- ▶ "(abc){3}" matches three consecutive occurrences of "abc".
- ▶ "abc{3}" matches "abccc".

# RULES FOR REGULAR EXPRESSIONS

Quantifiers can be used to tell “how often” an expression occurs.

| Quantifier | Description (Match if the expression is ...) |
|------------|--|
| +          | Repeated one or more times.                  |
| *          | Repeated zero or more times.                 |
| ?          | Repeated zero or one times.                  |
| {n}        | Repeated exactly n times.                    |
| {n, }      | Repeated n or more times.                    |
| {n, m}     | Repeated between n and m times.              |

## Examples

- ▶ "M[rs] [rs]?\\".?" matches "Mr", "Ms", "Mrs", "Mr.", "Ms.", "Mrs.". **转义之后就是表示“.”**
- ▶ The above also matches "Mrr", "Msr", "Mss", "Mrr.", "Msr.", "Mss." (and nothing else).

# RULES FOR REGULAR EXPRESSIONS

1. The dollar sign \$ means that a pattern only matches at the end of a line.
  - ▶ "[a-z,]\$" matches strings ending in lower-case letters or a comma.
2. The caret ^ outside of brackets means that a pattern only matches at the beginning of a line.
  - ▶ "^[^A-Z]" matches strings not beginning with capital letters.

# REGULAR EXPRESSIONS IN R

Many R functions we've already seen take regular expressions as their arguments.

- ▶ `strsplit()` can use a regular expression to divide a string into a vector.
- ▶ `grep()` can search for patterns represented by regular expressions in a string.
- ▶ `gsub()` can replace patterns represented by regular expressions in a string.

# REGULAR EXPRESSIONS IN R

## Honor Code Example

Without regular expressions we get weird results when we try to count words:

```
head(word_count, 10)  
tail(word_count, 10)
```

# REGULAR EXPRESSIONS IN R

## Honor Code Example

```
paste("1st", "2nd", "3rd", collapse = ", ")  
[1] "1st 2nd 3rd"  
paste("1st", "2nd", "3rd", sep = ", ")  
[1] "1st, 2nd, 3rd"
```

```
HC <- readLines("HonorCode.txt")  
HC <- paste(HC, collapse = " ") # One long string  
HC.words <- strsplit(HC, split=" ")[[1]] # Last Time  
head(HC.words, 10)  
tail(HC.words, 10)  
HC.words <- strsplit(HC, split="(\s|[:punct:])+")[[1]]  
head(HC.words, 10) +表示前面的一串作为整体出现一次或者多次  
tail(HC.words, 10) | 表示的是或者  
                  \\S 表示的是空白  
                  \\ 表示的就是\  
                  [:punct:] 表示的是标点符号
```

- ▶ Splits at blocks of only whitespace and/or punctuation.
- ▶ Regular expression is enclosed in quotation marks.
- ▶ "\s" is a special character like "\n" or "\t".

\n表示的是new line ; \t 表示的是tab

# REGULAR EXPRESSIONS IN R

## Honor Code Example

In the previous we have the following problem: `university's` splits to `university` and `s`.

## Exercise

Check that `split = "\s+|([[:punct:]]+[:space:]+)"` gives us what we want: *either any number of white spaces or at least one punctuation mark followed by at least one space.*

```
help-regexp)
```

It seems like `regex101.com` may be helpful, but I don't know how to use it.

# TRY ON YOUR OWN

## Tasks

Write a regular expression that matches the following:

- ▶ Only the strings "cat", "at", and "t"
- ▶ The strings "cat", "caat", "caaat", etc
- ▶ The strings "dog", "DOG", "Dog", "DoG" (the word dog with any combination of lowercase and captials)

1. "[c]?[a]?t"
2. "ca+t"
3. "[Dd][Oo][Gg]"

## EARTHQUAKES EXAMPLE

- ▶ Recall that the `grep()` functions searches a character string for a specified pattern.
- ▶ Now we can use regular expressions to specify that pattern.
- ▶ We're going to practice using some data from the web.

# EARTHQUAKES EXAMPLE

Code Example. <http://ncedc.org/anss/catalog-search.html>

# EARTHQUAKES EXAMPLE

Every earthquake of magnitude 6 on the Richter scale from January 1, 2002 until January 1, 2017. (We suppress a superfluous warning about end-of-line character in the `readLines()` call.)

```
quakes <- readLines("NCEDC_Search_Results.html", warn = FALSE)
head(quakes)
tail(quakes)
length(quakes)
quakes[8:15]
```

## Tasks

```
records <- grep("[0-9]{4}/[0-9]{2}/[0-9]{2}",quakes)
quakes <- quakes[records][-1]
head(quakes)
```

- ▶ Get rid of the first few lines of HTML formatting code and search parameters.
- ▶ Actual data begins on line 12. Headers on line 10.
- ▶ Strategy: all data lines begin with a date in format YYYY/MM/DD.

# EARTHQUAKES EXAMPLE

## Extracting the Data

```
date_express <- "^[0-9]{4}/[0-9]{2}/[0-9]{2}"  
  
head(grep(quakes, pattern = date_regex))  
tail(grep(quakes, pattern = date_regex))  
head(grep(quakes, pattern = date_regex, value = TRUE))  
head(grep(quakes, pattern = date_regex, value = TRUE))  
grep(quakes, pattern = date_regex, invert = TRUE, value = TRUE)
```

如果设置为value=TRUE , 那么返回的就是直接的结果 ,  
如果是FALSE , 返回的就是包含该字符串的语句的序号

如果设置为invert=TRUE , 那么返回的就是不包含该字符串的内容

# CHECK YOURSELF

Task

```
data_regex2 <- "NEI[0-9]{12}"
```

- ▶ We just extracted the lines we need by noting that they all begin with a date. The lines we need also seem to all end in an event i.d. which is a 12 digit code. Use this idea with `grep()` to extract the lines of actual data.
- ▶ This won't work – we leave behind some data lines. What happened?
- ▶ How else could we search for the data using regular expressions?

# MORE COMMANDS IN THE GREP() FAMILY

All return information about where regular expressions are matched in a string.

- ▶ `grep()` returns the indices containing a match.
- ▶ `grep1()` returns a logical indicating which elements contain a match.
- ▶ `regexpr()` returns the location of the first match with attributes like the length of the match.
- ▶ `gregexpr()` works similarly to `regexpr()`, but returns *all* matching locations. ‘g’ for global.
- ▶ `regmatches()` takes strings and the output of `regexpr()` or `gregexpr()` and returns the actual matching strings.

# MORE COMMANDS IN THE GREP() FAMILY

## Examples

```
# Is there a match?
```

```
grep("a[a-z]", "Alabama") [1] 1
```

```
# Information about the first match.
```

```
regexpr("a[a-z]", "Alabama") [1] 3  
attr("match.length") [1] 2  
attr("useBytes") [1] TRUE
```

```
# Information on all matches.
```

```
gregexpr("a[a-z]", "Alabama") [1] 3 5  
attr("match.length") [1] 2 2  
attr("useBytes") [1] TRUE
```

```
# What are the matches?
```

```
regmatches("Alabama", gregexpr("a[a-z]", "Alabama")) [[1]]  
[1] "ab" "am"
```

# EARTHQUAKES EXAMPLE

Let's Extract the (longitude, latitude) Pairs

```
quakes[11:15]
```

```
coord_exp <- "-?[0-9]+\\.\\.[0-9]{4}"  
full      <- paste(coord_exp, "\\s+", coord_exp, sep = "")  
[1] "-?[0-9]+\\.\\.[0-9]{4}\\s+-?[0-9]+\\.\\.[0-9]{4}"
```

A negative sign zero or one times with digits 0 – 9 one or more times followed by a period and four digits.

# EARTHQUAKES EXAMPLE

Let's Extract the (longitude, latitude) Pairs

grepl函数返回的是每一个是否包含字符串的T和F值

```
head(grepl(quakes, pattern = full), 15)  
coord_log <- grepl(quakes, pattern = full)
```

```
matches <- gregexpr(pattern = full, text = quakes[coord_log])
```

```
head(matches, 1) quakes[coord_log]
```

gregexpr() 返回的是一个列表分别表示所有符合的位置，剩下的就是包含经纬度等  
长度等

```
coords <- regmatches(quakes[coord_log], matches)
```

```
head(coords, 4)
```

regmatches 是以数据和grepexpr的结果为参数的，直接截取出符合内容，并组成列表

```
coords_split <- sapply(coords, strsplit, split="\s+")
```

```
head(coords_split, 3)
```

利用sapply函数对于list中的每一个元素按照“空格”为标志进行分块

```
coords_mat <- matrix(unlist(coords_split), ncol = 2,  
byrow = TRUE)
```

```
colnames(coords_mat) <- c("Latitude", "Longitude")
```

```
head(coords_mat)
```

## EARTHQUAKES EXAMPLE

```
library(maps)
map("world")
points(coords_mat[, "Longitude"], coords_mat[, "Latitude"],
       pch = 19, col = "red", cex = .5)
```

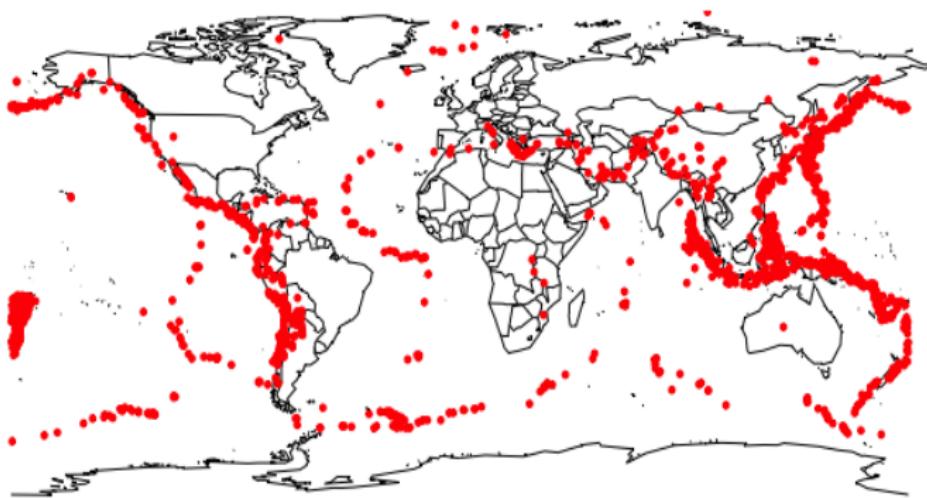


Figure 1: Earthquake Locations

# WEB SCRAPING

# WHAT IS WEB SCRAPING?

- ▶ We've learned about getting data in and out of R when it's structured: `read.table()`, `read.csv()`, etc.
- ▶ Often, like the last example, it's not as structured.
  - ▶ Could have metadata.
  - ▶ Non-tabular arrangement.
- ▶ In general this is true of data on the web.

## Strategy

Read in line-by-line and split into a nicer format (generally requires a lot of regular expressions).

# READING IN DATA FROM A URL CONNECTION

```
con <- url("http://www.columbia.edu", "r")
x   <- readLines(con, warn = FALSE)

head(x, 20)
length(x)
```

- ▶ Reading in a simple webpage may be useful if data are embedded in the HTML somewhere.
- ▶ More often we use the connection to read in specific data file stored on web servers.
- ▶ This is probably preferable to downloading the HTML and loading, but the code you write may not work later if the server is reorganized or changed.

# WHAT IS WEB SCRAPING?

- ▶ Webpages are generally designed for humans to read.
- ▶ Use a computer to extract the information we actually want.
- ▶ Iterate the process.

## Strategy

Take in unstructured pages, return rigidly-formatted data.

# WHAT IS WEB SCRAPING?

How do we use the computer extract the information we want?

- ▶ Information is **somewhere** in the page source, usually in the HTML code.
- ▶ Often some sort of marker or pointer surrounding the data (again, usually HTML).
- ▶ Pick apart the HTML to leave the data using regular expressions.

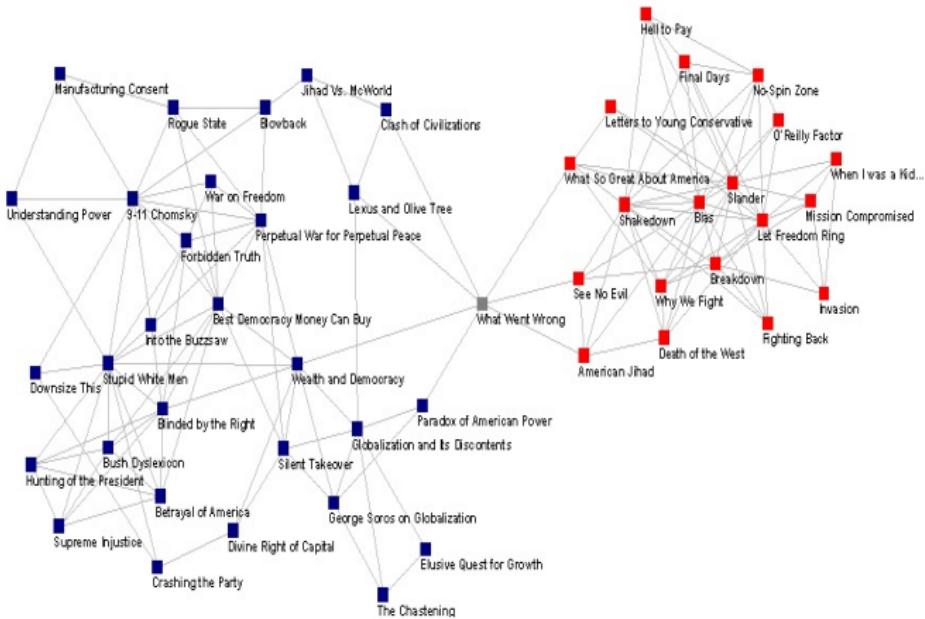
# WHAT IS WEB SCRAPING?

How do we pick apart HTML code with regular expressions?

- ▶ What **exactly** do we want from the page?
- ▶ How is the page organized? Where is the information we want located?
  - ▶ How does it show up on the webpage?
  - ▶ How is that represented in the HTML?
- ▶ Write a function to automate the information extraction.
- ▶ Now iterate over relevant pages.

# BOOK NETWORKS EXAMPLE

Famous example from Vladis Krebs <http://www.orgnet.com/>.



Books are connected if they're purchased together on Amazon.

Figure 2: Book Network

# BOOK NETWORKS EXAMPLE

Code Example.

# BOOK NETWORKS EXAMPLE

Amazon gives away this information (to try to get you to buy more books).

## Strategy

- ▶ Load the page source into R.
- ▶ Use regular expression to locate this line (a line that begins with a certain kind of HTML, must contain at least one ISBN, and then ends).
- ▶ Extract the ISBNs as a string then split using repeating HTML code like ‘&quot’.
- ▶ Record this information, then move down the line of ISBNs and repeat for each. **‘Snowball Sampling’**.

# OPTIONAL READING

- ▶ The eBook **Handling and Processing Strings in R** by Gaston Sanchez.
- ▶ Cosma Shalizi's **Handout: Regular Expressions**.

# Lecture 5: Functions and Environment

STAT GR5206

*Statistical Computing & Introduction to Data Science*

Cynthia Rush  
Columbia University

October 6, 2017

# COURSE NOTES

- ▶ Homework 2 posted.
- ▶ Ask (good) questions on Piazza and answer others' questions.
- ▶ Submit what you have finished at the end of class. Can resubmit later.
- ▶ Labs are not graded for correctness – just completion!

# CODING STANDARDS

Coding standards aren't universal and are often highly debated. Here are some of my suggestions and the rationale behind them.

- ▶ **Always use scripts or RMarkdown.** Using text files and a text editor is fundamental to coding. (If you're writing your code in an editor like Microsoft Word, you need to stop.)
- ▶ **Indent your code.** This is very important for the readability of your code. Some programming languages actually require it as part of their syntax, but R does not. I indent two spaces whenever I'm inside a loop or a function – you'll see!
- ▶ **Limit the width of your code.** You don't want the code you write to extend beyond the right hand side of your page. This limitation, along with indentation, forces you to write code that is clean, readable, and naturally broken down into modular units
- ▶ **Limit the length of individual functions.** If you are writing functions, it's usually a good idea to not let your functions run for pages and pages. Typically, purpose of a function is to execute one activity or idea. If your function is doing lots of things, it probably needs to be broken into multiple functions.

# WRITING FUNCTIONS IN R

# FUNCTIONS IN R

Functions are one of the most important constructs in R (and many other languages). They allow you to modularize your code - encapsulating a set of repeatable operations as an individual function call.

## Why Functions?

- ▶ Like data structures tie related values into one object, functions tie related commands into one object.
- ▶ Both cases: easier to understand, work with, and to build into larger structures.

# FUNCTIONS IN R

You should rely heavily on functions rather than having long sets of expressions in R scripts.

Functions have many important advantages:

- ▶ They reduce bugs by avoiding having multiple instances of the same functionality.
- ▶ They reduce time involved in coding by eliminating redundancy.
- ▶ They make for cleaner and more easily-readable code.

A basic goal in writing functions is **modularity**.

In general, a function should:

- ▶ be fairly short,
- ▶ be focused and specific in what it does, and
- ▶ be designed so that it can be used in combination with other functions to carry out more complicated operations.

# FUNCTIONS IN R

## Basic Structure

```
function_name <- function(arg1, arg2, ... ) {  
  statements  
  return(object)  
}
```

- ▶ A **function** is a group of instructions that takes inputs, uses them to compute other values, and returns a result.
- ▶ We can write and add our own functions in R.
- ▶ Functions:
  1. Have names.
  2. *Usually* take in arguments.
  3. Include body of code that does something.
  4. *Usually* return an object at the end.

# EXAMPLE FUNCTION

## A Function to Check for Significance at $\alpha = 0.05$

```
# Input x should be a single p-value in [0,1]
significant <- function(x) {
  if (x <= 0.05) { return(TRUE) }
  else { return(FALSE) }      ifelse(x<0.5, return(TRUE),return(FALSE))
}
```

- ▶ First, tell R to define a function named `significant`.
- ▶ Brackets `{` and `}` mark the start and close of the body.
- ▶ R tells you you're in the body of the function by using `+` as a prompt (instead of `>`).
- ▶ At the end, use the `return()` command.

# EXAMPLE FUNCTION

## A ‘Robust’ Loss Function (for Outlier-Resistant Regression)

```
# Inputs: A vector of numbers (x)
# Outputs: A loss vector with x^2 for small elements,
#           and 2|x|-1 for large ones

res_loss <- function(x) {
  loss_vec <- ifelse(x^2 > 1, 2*abs(x) - 1, x^2)
  return(loss_vec)
}
```

Let's try it:

```
vec <- c(-0.5, 0.9, -3, 4)
res_loss(vec)
```

# EXAMPLE FUNCTION

## A 'Robust' Loss Function (for Outlier-Resistant Regression)

```
res_loss <- function(x) {  
  loss_vec <- ifelse(x^2 > 1, 2*abs(x) - 1, x^2)  
  return(loss_vec)  
}
```

Break apart the function

What are the...

- ▶ Inputs? `x`
- ▶ Outputs? `loss_vec`
- ▶ Body Statements?

```
loss_vec <- ifelse(x^2 > 1, 2*abs(x) - 1, x^2)  
return(loss_vec)
```

# WHEN SHOULD WE MAKE A FUNCTION?

- ▶ Things you will rerun.
- ▶ Chunks of code that are small parts of bigger analyses.

# CHECK YOURSELF

## Task

Write a function called `FiveTimesSum` that takes as input a vector of numerical values and returns 5 times the sum of those values. Test it on the vector `1:3`. Your output should be 30.

```
FiveTimeSum <- function(x){  
  return(sum(x)*5)  
}  
FiveTimeSum(1:3)
```

# NAMED AND DEFAULT ARGUMENTS

## A ‘Robust’ Loss Function (for Outlier-Resistant Regression)

```
# Inputs: A vector of numbers (x),
#          crossover location (c > 0)
# Outputs: A loss vector with x^2 for small elements,
#          and 2c|x|-c for large ones

res_loss2 <- function(x, c = 1) {
  loss_vec <- ifelse(x^2 > c, 2*c*abs(x) - c, x^2)
  return(loss_vec)
}
```

Let's try it:

```
identical(res_loss(vec), res_loss2(vec, c = 1))
identical(res_loss(vec), res_loss2(vec, c = 2))
```

# NAMED AND DEFAULT ARGUMENTS

- ▶ Default values get used if names are missing:

```
identical(res_loss2(vec, c = 1), res_loss2(vec))
```

- ▶ Named argument can go in any order when they are explicitly tagged:

```
identical(res_loss2(x = vec, c = 2),  
         res_loss2(c = 2, x = vec))
```

Otherwise, the fall-back is to use the position.

- ▶ Funny things can happen when arguments aren't as we expect:

```
vec <- c(-0.5, 0.9, -3, 4)
```

依次遍历c中  
的所有元素  
一一对应

```
res_loss2(vec, c = c(1,1,1,5))
```

```
res_loss2(vec, c = -1)
```

[1] 0.0 -0.8 -5.0 -7.0

```
res_loss2 <- function(x, c = 1) {  
  loss_vec <- ifelse(x^2 > c,  
  2*c*abs(x) - c, x^2)  
  return(loss_vec)  
}
```

# CHECKING ARGUMENTS

Solution: Add some checks to your function.

## A 'Robust' Loss Function (for Outlier-Resistant Regression)

```
res_loss2 <- function(x, c = 1) {  
  # Scale should be a single positive number  
  stopifnot(length(c) == 1, c > 0)  
  loss_vec <- ifelse(x^2 > c^2, 2*c*abs(x) - c, x^2)  
  return(loss_vec)  
}
```

```
res_loss2(vec, c = c(1,1,1,5))  
res_loss2(vec, c = -1)
```

通过设定了stopifnot，这两种情况直接报错

`stopifnot()`

- ▶ Arguments are a series of expressions which should all be TRUE.
- ▶ Execution stops with error at first FALSE.

# CHECK YOURSELF

## Task

Write a function the will take an input vector and set any value below a threshold to be the value of the threshold. Optionally, the function should instead set values above the threshold to the value of the threshold. Hint: The function should have three arguments, two required – the vector and the threshold, and one optional with a default value of "below".

```
f <- function(vec, thresh, dir = "below") {  
  stopifnot(dir == "below" | dir == "above")  
  
  if (dir == "below") {  
    vec[vec < thresh] <- thresh  
  } else {  
    vec[vec > thresh] <- thresh  
  }  
  return(vec)  
}
```

# USE YOUR FUNCTIONS IN OTHER FUNCTIONS

- ▶ Use your own function in built-in R functions like `apply()`.
- ▶ Ex: `curve(expression, from = , to = )` plots a curve.

```
curve(res_loss2, from = -2, to = 2)
```

# THE R ENVIRONMENT

- ▶ The **global environment** (or the workspace) in R consists of the collection of your named objects.
- ▶ When you start an R session, a new environment is initialized (unless you load a saved environment).
- ▶ When a function is called, a new **local environment** is created within the body of the function.

# THE R ENVIRONMENT

Code Example.

# THE FUNCTION ENVIRONMENT

- ▶ Each function has its own (internal) **environment**.
- ▶ Names in the function environment override names from the **global environment**.
- ▶ Assignments in the internal environment don't change the global environment.
- ▶ In general, functions should not make use of variables outside the function – this reduces bugs and provides modularity (some exceptions to this rule).
- ▶ Functions search for named variables (undefined in the function itself or the argument) in the environment in which the function was created (in our case, the global environment).

# IMPORTANT FUNCTION CONCEPTS

Functions return an R object and in general functions should be designed such that the only effect of the function on the global environment is through the return object.

Side effects are when a function affects the state of the world in addition to its return value. Sometimes this is OK. Examples:

- ▶ `library()`
- ▶ `plot()`
- ▶ `setwd()`

# THE FUNCTION ENVIRONMENT

```
x <- 2
f <- function(y) {
  return(x + y)
}
f(1)
```

如果function内部没有赋值，系统会自动在global environment中寻找

```
g <- function(y) {
  x <- 10
  return(x + y)
}
g(1)
x
```

# THE FUNCTION ENVIRONMENT

```
g <- function(y) {  
  f <- function(y) {  
    return(x + y)  
  }  
  x <- 10  
  return(f(y))  
}  
g(1)
```

函数首先只是被创建了但是没有被调用，函数内部也没有对x赋值，在调用过程中，他向上一级函数g中寻找到了x的赋值，所以可以正常运行

# USE YOUR FUNCTION INTERFACES

The function **interfaces** are the places where the function interacts with the global environment: at the **inputs** and the **outputs**.

- ▶ Interact with the rest of the system only at the interfaces:
  - ▶ Arguments should give your function all the information it needs
  - ▶ Reduces the risk of bugs
  - ▶ Exception would be universal constants like `pi`.
- ▶ Output should be only through `return()`.

# KEY PRINCIPLES IN R

Everything that happens is a function call

```
3+2
```

```
'+'(3,2)
```

```
x <- matrix(runif(100), ncol = 10)
x[, 2]
'['](x, , 2)
```

Everything that exists is an object

```
class(1)
class(runif)
class(function(x) x^2)
square <- class(function(x) x^2)
class(square)
```

# EXTENDED EXAMPLE: FITTING A MODEL

# THE MODEL

We study the following idea: bigger cities tend to produce more economically per capita.

Geoffrey West et al.

A proposed statistical model for this relationship:

$$Y = \beta_0 X^{\beta_1} + \epsilon,$$

where

- ▶ Y: per-capita ‘gross metropolitan product’ of a city,
- ▶ X: is the population of the city,
- ▶  $\beta_0, \beta_1$ : parameters,
- ▶  $\epsilon$ : noise.

# THE DATA

```
gmp <- read.table("gmp.txt", as.is = TRUE, header = TRUE)  
head(gmp)[1:3, ]
```

## Variables

- ▶ city
- ▶ gmp: gross metropolitan product (money the city makes)
- ▶ pcgmp: per-capita gross metropolitan product (money the city makes divided by the number of people in the city)

## THE DATA (CONT.)

Our model uses the population, so let's create a variable for that in the dataset.

$$\text{pcgmp} = \frac{\text{gmp}}{\text{population}} \quad \rightarrow \quad \text{population} = \frac{\text{gmp}}{\text{pcgmp}}.$$

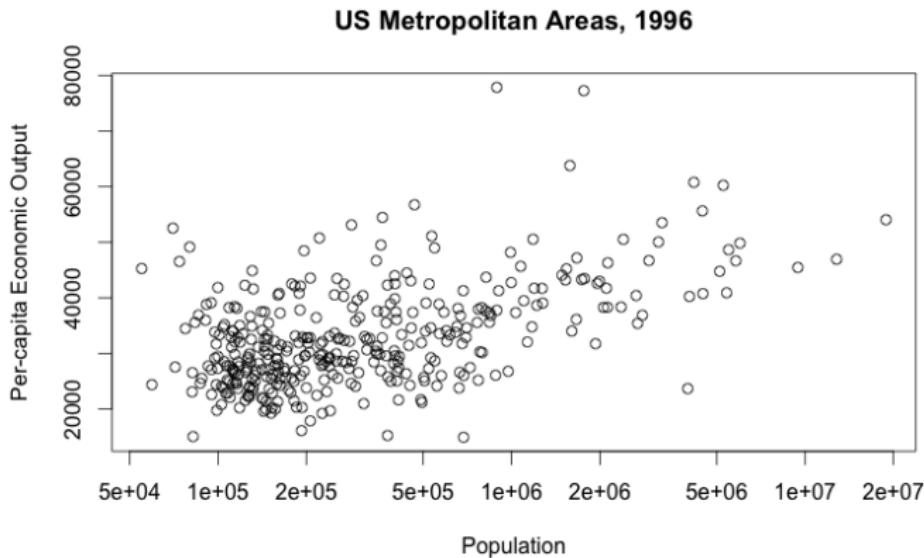
```
gmp$pop <- gmp$gmp/gmp$pcgmp
head(gmp) [1:3, ]
```

## THE DATA (CONT)

### Plotting Per-Capita GMP vs. Population

log= 'x' 表明对x的数据取对数

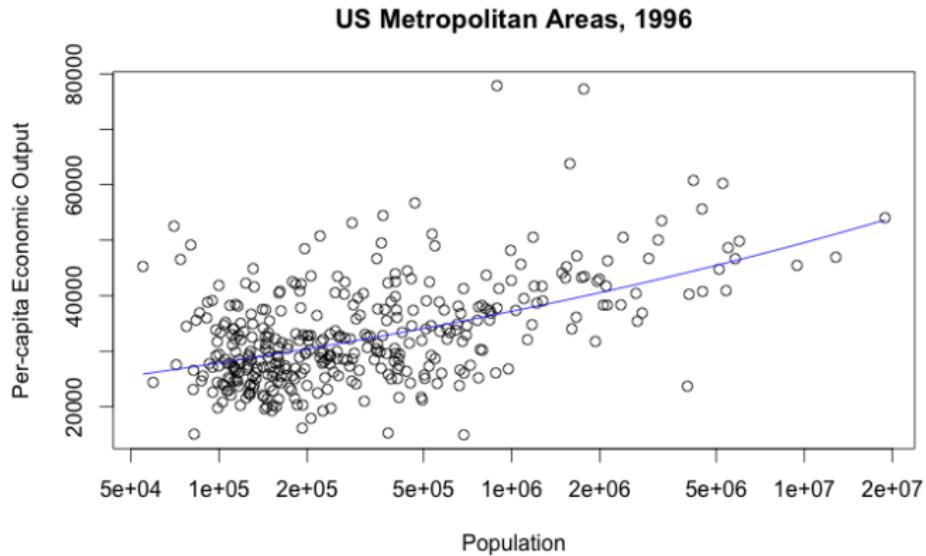
```
plot(gmp$pop, gmp$pcgmp, log = "x", xlab = "Population",  
      ylab = "Per-capita Economic Output")
```



# THE DATA (CONT)

## Plotting Per-Capita GMP vs. Population

```
# beta_0 = 6611; beta_1 = 1/8
curve(6611*x^{1/8}, add = TRUE, col = "blue")
```



# FITTING THE FUNCTION

Want to fit the model:

$$Y = 6611X^{\beta_1} + \epsilon.$$

We're assuming  $\beta_0 = 6611$  and trying to estimate  $\beta_1$  using the data.

## Strategy

Minimize the (mean) sum of the squares!

$$\min_{\beta} \frac{1}{n} \sum_{i=1}^n (Y_i - 6611X_i^{\beta})^2$$

where  $n$  is the number of data points.

Note this is the **training mean square error**.

# OPTIMIZATION PROBLEMS

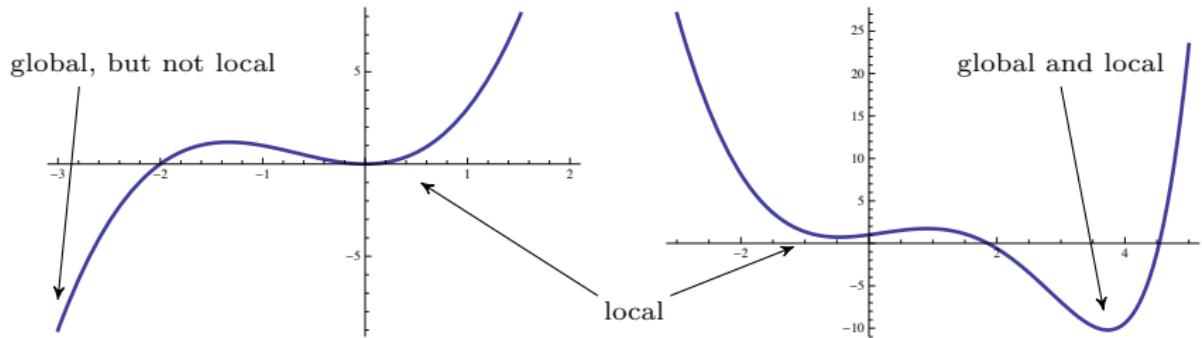
## Terminology

An **optimization problem** for a given function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is a problem of the form

$$\min_{\mathbf{x}} f(\mathbf{x})$$

which we read as “find  $\mathbf{x}_0 = \arg \min_{\mathbf{x}} f(\mathbf{x})$ ”.

# TYPES OF MINIMA



## Local and global minima

A minimum of  $f$  at  $x$  is called:

- ▶ **Global** if  $f$  assumes no smaller value on its domain.
- ▶ **Local** if there is some open neighborhood  $U$  of  $x$  such that  $f(x)$  is a global minimum of  $f$  restricted to  $U$ .

## Analytic criteria for local minima

Recall that  $\mathbf{x}$  is a local minimum of  $f$  if

$$f'(\mathbf{x}) = 0 \quad \text{and} \quad f''(\mathbf{x}) > 0 .$$

## Numerical methods

All numerical minimization methods perform roughly the same steps:

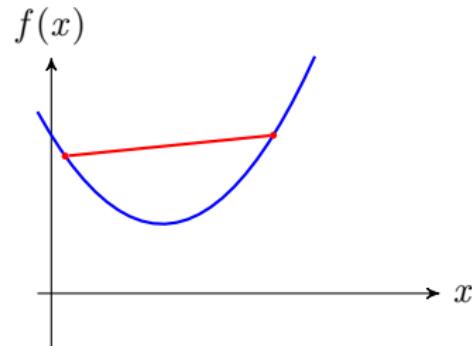
- ▶ Start with some point  $x_0$ .
- ▶ Our goal is to find a sequence  $x_0, \dots, x_m$  such that  $f(x_m)$  is a minimum.
- ▶ At a given point  $x_n$ , compute properties of  $f$  (such as  $f'(x_n)$  and  $f''(x_n)$ ).
- ▶ Based on these values, choose the next point  $x_{n+1}$ .

The information  $f'(x_n)$ ,  $f''(x_n)$  etc is always *local at  $x_n$* , and we can only decide whether a point is a local minimum, not whether it is global.

# CONVEX FUNCTIONS

## Definition

A function  $f$  is **convex** if every line segment between function values lies above the graph of  $f$ .



## Analytic criterion

A twice differentiable function is convex if  $f''(x) \geq 0$  (or  $H_f(\mathbf{x})$  positive semidefinite) for all  $\mathbf{x}$ .

## Implications for optimization

If  $f$  is convex, then:

- ▶  $f'(x) = 0$  is a sufficient criterion for a minimum.
- ▶ Local minima are global.
- ▶ If  $f$  is **strictly convex** ( $f'' > 0$  or  $H_f$  positive definite), there is only one minimum (which is both global and local).

# GRADIENT DESCENT

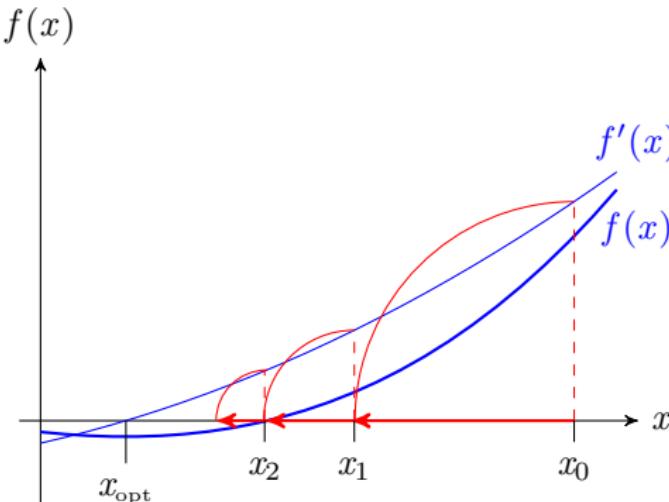
## Algorithm

Gradient descent searches for a minimum of  $f$ .

1. Start with some point  $x \in \mathbb{R}$  and fix a precision  $\varepsilon > 0$ .
2. Repeat for  $n = 1, 2, \dots$

$$x_{n+1} := x_n - f'(x_n)$$

3. Terminate when  $|f'(x_n)| < \varepsilon$ .



# FITTING OUR FUNCTION

## Strategy

Minimize the (mean) sum of the squares!

$$\min_{\beta} \frac{1}{n} \sum_{i=1}^n (Y_i - 6611X_i^{\beta})^2.$$

In our example,

Initial point  $\beta_0$ . Approximate the derivative w.r.t.  $\beta$  and move in the opposite direction:

$$MSE(\beta_0) = \frac{1}{n} \sum_{i=1}^n (Y_i - 6611X_i^{\beta_0})^2$$

$$MSE'(\beta_0) \approx \frac{MSE(\beta_0 + h) - MSE(\beta_0)}{h} \quad h \text{ is small}$$

$$\beta_1 \approx \beta_0 - c * MSE'(\beta_0) \quad c \text{ scales our step size}$$

## A first attempt at code:

```
max.iter      <- 100      # How long we run the alg.
stop.deriv    <- 1/100    # If derivative is small, stop
deriv.step    <- 1/1000   # This is h
step.scale    <- 1e-12    # This is c

iter          <- 0        # Iteration counter
deriv         <- Inf
beta          <- 0.15

while((iter < max.iter) & (deriv > stop.deriv)) {
  iter  <- iter + 1
  mse.1 <- mean((gmp$pcgmp - 6611*gmp$pop^beta)^2)
  mse.2 <- mean((gmp$pcgmp - 6611*gmp$pop^(beta + deriv.step))^2)
  deriv <- (mse.2 - mse.1)/deriv.step
  beta  <- beta - step.scale*deriv
}
list(beta = beta, iteration = iter, conv = (iter < max.iter))
```

# WHAT'S WRONG WITH THE PREVIOUS ATTEMPT?

- ▶ **Not encapsulated:** Re-run by copying and pasting. Note easy to fit into bigger project.
- ▶ **Inflexible:** To edit initializations must copy, paste, re-run
- ▶ **Error-prone:** If you change the dataset, not sure if it would still work...
- ▶ **Hard to fix:** Should stop when *absolute value* of derivative is small.

## First Fix

改进：设置为函数，把主要的参数放到里面来

```
est.exp <- function(beta) {  
  max.iter      <- 100      # How long we run the alg.  
  stop.deriv    <- 1/100    # If derivative is small, stop  
  deriv.step    <- 1/1000   # This is h  
  step.scale    <- 1e-12    # This is c  
  iter          <- 0  
  deriv          <- Inf  
  deriv只是一个初始值，为了避免靠近0，所以设置为inf  
  while((iter < max.iter) & (abs(deriv) > stop.deriv)) {  
    iter <- iter + 1  
    mse.1 <- mean((gmp$pcgmp - 6611*gmp$pop^beta)^2)  
    mse.2 <- mean((gmp$pcgmp - 6611*gmp$pop^(beta + deriv.step))^2)  
    deriv <- (mse.2 - mse.1)/deriv.step  
    beta  <- beta - step.scale*deriv  
  }  
  fit <- list(beta = beta, iteration = iter, conv = (iter < max.iter))  
  return(fit)  
}
```

## Second Fix

**Problem:** Have to rerun if we want to change defined parameters.

**Solution:** Let's make them arguments (with default values) of the function.

## Second Fix

```
est.exp <- function(beta, beta_0 = 6611, max.iter = 100,
                      stop.deriv = .01, deriv.step = .001,
                      step.scale = 1e-12) {

  iter  <- 0
  deriv <- Inf

  while((iter < max.iter) & (abs(deriv) > stop.deriv)) {
    iter  <- iter + 1
    mse.1 <- mean((gmp$pcgmp - beta_0*gmp$pop^beta)^2)
    mse.2 <- mean((gmp$pcgmp - beta_0*gmp$pop^(beta + deriv.step))^2)
    deriv <- (mse.2 - mse.1)/deriv.step
    beta  <- beta - step.scale*deriv
  }
  fit <- list(beta = beta, iteration = iter, conv = (iter < max.iter))
  return(fit)
}
```

## Third Fix

**Problem:** Don't need to write out the MSE calculations twice in the body of the function.

**Solution:** Write a `MSE()` function.

## Third Fix

```
est.exp <- function(beta, beta_0 = 6611, max.iter = 100,
                      stop.deriv = .01, deriv.step = .001,
                      step.scale = 1e-12) {

  iter  <- 0
  deriv <- Inf

  mse <- function(b) {mean((gmp$pcgmp - beta_0*gmp$pop^b)^2)}

  while((iter < max.iter) & (abs(deriv) > stop.deriv)) {
    iter  <- iter + 1
    deriv <- (mse(beta + deriv.step) - mse(beta))/deriv.step
    beta  <- beta - step.scale*deriv
  }
  fit <- list(beta = beta, iteration = iter, conv = (iter < max.iter))
  return(fit)
}
```

`mse()` is declared inside the function so it's not added to the global environment.

## Fourth Fix

**Problem:** Locked into using specific columns of `gmp` – if we want to use a different data set, have to rewrite the function.

**Solution:** Make them arguments.

## Fourth Fix

```
est.exp <- function(beta, beta_0 = 6611, response = gmp$pcgmp,
                     predictor = gmp$pop, max.iter = 100,
                     stop.deriv = .01, deriv.step = .001,
                     step.scale = 1e-12) {

  iter <- 0
  deriv <- Inf

  mse <- function(b) {mean((response - beta_0*predictor^b)^2)}

  while((iter < max.iter) & (abs(deriv) > stop.deriv)){
    iter <- iter + 1
    deriv <- (mse(beta + deriv.step) - mse(beta))/deriv.step
    beta <- beta - step.scale*deriv
  }
  fit <- list(beta = beta, iteration = iter, conv = (iter < max.iter))
  return(fit)
}
```

## Fifth Fix

**Problem:** Want to make it easy for humans to read.

**Solution:** Change the `while` loop to `for` loop with a `break()` command.

## Fifth Fix

```
est.exp <- function(beta, beta_0 = 6611, response = gmp$pcgmp,
                     predictor = gmp$pop, max.iter = 100,
                     stop.deriv = .01, deriv.step = .001,
                     step.scale = 1e-12) {

  iter  <- 0
  deriv <- Inf
  mse <- function(b) {mean((response - beta_0*predictor^b)^2)}

  for (i in 1:max.iter) {
    iter  <- iter + 1
    deriv <- (mse(beta + deriv.step) - mse(beta))/deriv.step
    beta  <- beta - step.scale*deriv
    if (abs(deriv) < stop.deriv) {break()}
  }
  fit <- list(beta = beta, iteration = iter, conv = (iter < max.iter))
  return(fit)
}
```

# SUMMARY

Final code is shorter, more flexible, easier to understand, and more re-usable!

- ▶ **Exercise:** Run the code with the default values to get an estimate of  $\beta$ . Plot the curve with the data points to check out the fit.
- ▶ **Exercise:** Randomly remove one data point – how much does the estimate change?
- ▶ **Exercise:** Run the code from different starting points – how much does the estimate change?

# ASSESSING MODEL ACCURACY

# SELECTING A MODEL

## Some Questions

- ▶ The optimization algorithm finds different solutions each time we run it – which one is better?
- ▶ What if we used some value other than 6611 in the model? How could we compare?
- ▶ Should we use a different form of curve all together?
- ▶ Suppose we have additional variables, should they be included in the model?

Often we must compare multiple models. How do we know which is best?

# QUALITY OF FIT

To evaluate the performance of a method on a given data set, we need to measure how well its predictions match the observed data.

- We used **mean squared error** (MSE):

$$\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{\beta}_0 X_i^{\hat{\beta}_1})^2.$$

Called **training MSE** since it's calculated on the training data.

- But what we actually want to minimize is the **test MSE**:

$$Ave(Y_{test} - \hat{\beta}_0 X_{test}^{\hat{\beta}_1})^2,$$

using **test** data that was not used to fit the model.

We are interested in the accuracy of the predictions that we obtain when we apply our method to previously unseen **test data**.

# THE TEST MSE

How do we minimize the test MSE?

- ▶ Sometimes have a **test** dataset (that wasn't used to train the model.)
- ▶ What if no **test** data is available?
- ▶ Idea: Use the model with the lowest **training** MSE.
  - ▶ Unfortunately, no guarantee that the method with the lowest **training** MSE will also have the lowest **test** MSE.
- ▶ Soon we'll learn a strategy for estimating the **test** MSE when no **test** data is available: *Cross-validation*.

# CLASSIFICATION

# WHAT IS CLASSIFICATION?

- ▶ Linear regression is used when we want to predict a quantitative response variable.
- ▶ What if we have a categorical response variable?
- ▶ The study of predicting categorical response variables is called **classification**.

## Examples

- ▶ A person has symptoms that could possibly be attributed to one of three medical conditions. Which condition does he have?
- ▶ A bank must be able to determine whether an online transaction is fraudulent, on the basis of the user's IP address, past transaction history, etc.
- ▶ Using DNA sequence data for patients with and without a disease, a biologist would like to figure out which DNA mutations cause diseases and which do not.

# TYPES OF CLASSIFIERS

Many methods of classification:

- ▶ Logistic Regression.
- ▶ Linear Discriminant Analysis.
- ▶ K Nearest Neighbors.
- ▶ Trees and Random Forests.
- ▶ Support Vector Machines.

## Set-up

- ▶ Just as in regression, we have a set of training observations (data):  
$$(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n), \quad Y_1, Y_2, \dots, Y_n \text{ categorical.}$$
- ▶ Use the data to build the classifier.
- ▶ We want our classifier to perform well not only on the training data, but also on test observations that were not used to build the classifier.

# ASSESSING THE ACCURACY

Recall, for our model earlier, we calculate the **training** mean squared error:

$$\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{\beta}_0 X_{i1}^{\hat{\beta}_1})^2.$$

```
mse <- function(b) {mean((response - beta_0*predictor^b)^2)}
```

For classification, we study the **error rate**:

$$\frac{1}{n} \sum_{i=1}^n \mathbb{I}[Y_i \neq \hat{Y}_i].$$

```
error_rate <- mean(prediction[i]==testing[i])
```

- $\hat{Y}_i$  is the predicted classification for the  $i^{th}$  observation.
- $\mathbb{I}[Y_i \neq \hat{Y}_i]$  is an *indicator variable* that equals 1 if  $Y_i \neq \hat{Y}_i$  and 0 if  $Y_i = \hat{Y}_i$ .
- If  $\mathbb{I}[Y_i \neq \hat{Y}_i] = 0$  then the  $i^{th}$  observation was classified correctly.
- The *error rate* computed the fraction of incorrect classifications.

# ASSESSING THE ACCURACY

- ▶ Similarly to the model discussed previously, the following is referred to as the **training** error rate:

$$\frac{1}{n} \sum_{i=1}^n \mathbb{I}[Y_i \neq \hat{Y}_i].$$

- ▶ The **test** error rate associated with a set of **test** observations  $(X_{test}, Y_{test})$  is given by

$$Ave(\mathbb{I}[Y_{test} \neq \hat{Y}_{test}]).$$

- ▶ A good classifier is one for which the **test** error rate is the smallest.

# BAYES CLASSIFIER

The best classifier (in terms of **test** error) *assigns each observation to the most likely class*, given its predictor values.

- ▶ Assign test observation with predictor vector  $X_{test}$  to the class  $j$  for which the following probability is the largest:

$$Pr(Y = j | X = X_{test}).$$

- ▶ This is a *conditional probability*. Recall ‘Bayes Rule’:

$$Pr(A|B) = \frac{Pr(A \text{ and } B)}{Pr(B)}$$

# BAYES CLASSIFIER

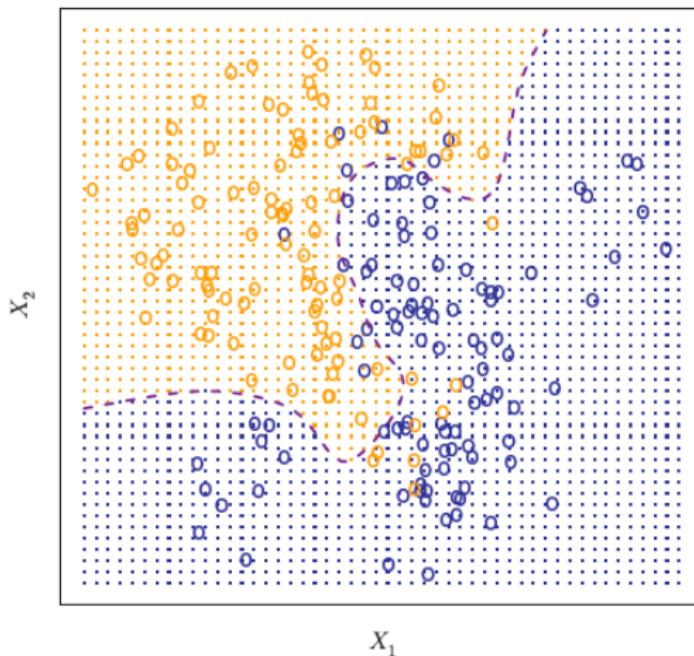
Consider a two-class problem, meaning  $Y_{test}$  is either class  $A$  or class  $B$ .

The Bayes classifier predicts:

$$\begin{cases} \text{Class A} & \text{if } Pr(Y = A | X = X_{test}) > 0.5, \\ \text{Class B} & \text{otherwise.} \end{cases} \quad (1)$$

# BAYES CLASSIFIER EXAMPLE <sup>1</sup>

Simulated data of 100 observations. Response  $Y$  either blue or orange, with two predictors  $X_1$  and  $X_2$ . The purple line represents the **Bayes decision boundary**.



<sup>1</sup>Image from 'Introduction to Statistical Learning'.

# $K$ NEAREST NEIGHBORS

In theory would like to always use the Bayes classifier. In practice, don't know  $Pr(Y|X)$ !

## $K$ Nearest Neighbors (KNN)

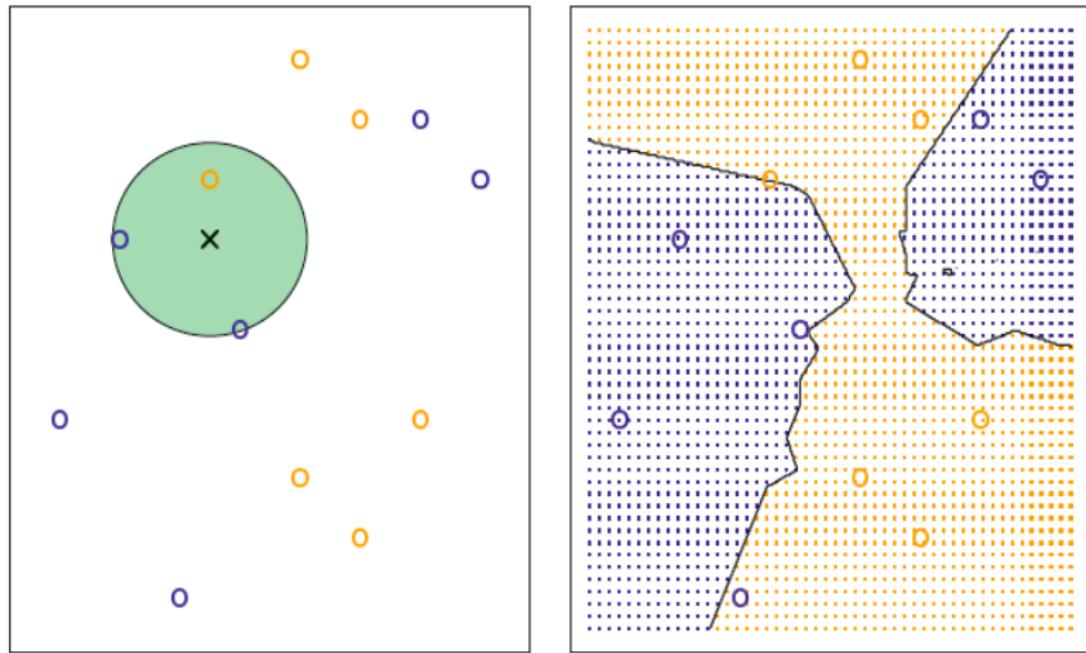
- ▶ Estimates  $Pr(Y|X)$  and then classifies observations to the class with highest estimated probability.
- ▶ Given a positive integer  $K$  and a test observation  $X_{test}$ :
  - ▶ Identify  $K$  points in training data closest to  $X_{test}$ . Label  $\mathcal{N}_{test}$ .
  - ▶ Estimate conditional probability for class  $j$  as fraction of points in  $\mathcal{N}_{test}$  whose response values equal  $j$ :

$$Pr(Y = j | X = X_{test}) = \frac{1}{K} \sum_{i \in \mathcal{N}_{test}} \mathbb{I}(Y_i = j).$$

- ▶ Classify the test observation to class with the largest probability.

# KNN CLASSIFIER EXAMPLE <sup>2</sup>

Training data with six blue and six orange observations. Use  $K = 3$ .  
**KNN decision boundary** shown in black.

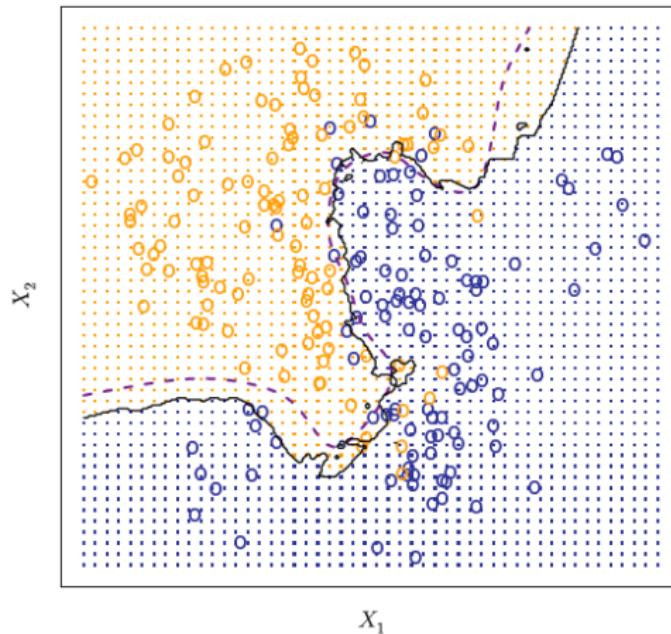


<sup>1</sup>Image from 'Introduction to Statistical Learning'.

# KNN CLASSIFIER EXAMPLE <sup>3</sup>

Though simple, KNN can be surprisingly close to Bayes optimal classifier!

KNN: K=10



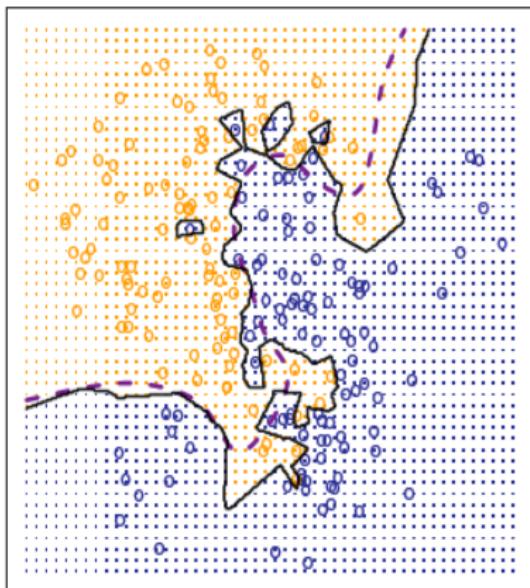
---

<sup>1</sup>Image from 'Introduction to Statistical Learning'.

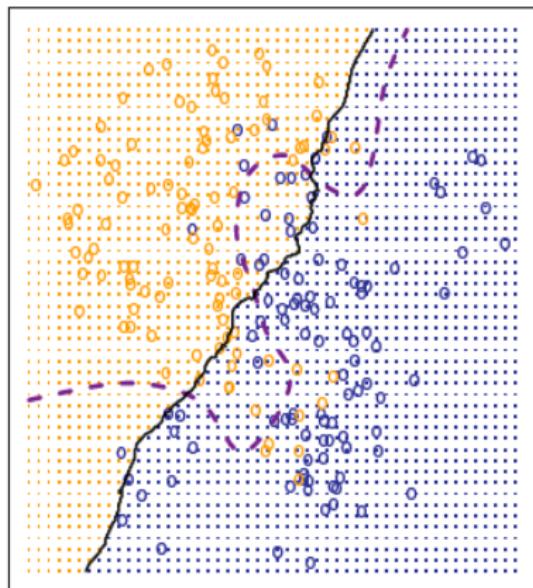
# KNN CLASSIFIER EXAMPLE <sup>4</sup>

Choice of  $K$  matters!

KNN:  $K=1$



KNN:  $K=100$



<sup>1</sup>Image from 'Introduction to Statistical Learning'.

# ASSESSING MODEL ACCURACY

As with regression, **training** error not a good predictor of **test** error.  
(Previous example!)

# CLASSIFICATION: AN EXAMPLE

Can we predict the direction of the stock market today using the previous two days' movements?

```
# install.packages("ISLR")
library(ISLR)
head(Smarket, 3)
```

# CLASSIFICATION: AN EXAMPLE

## Variables

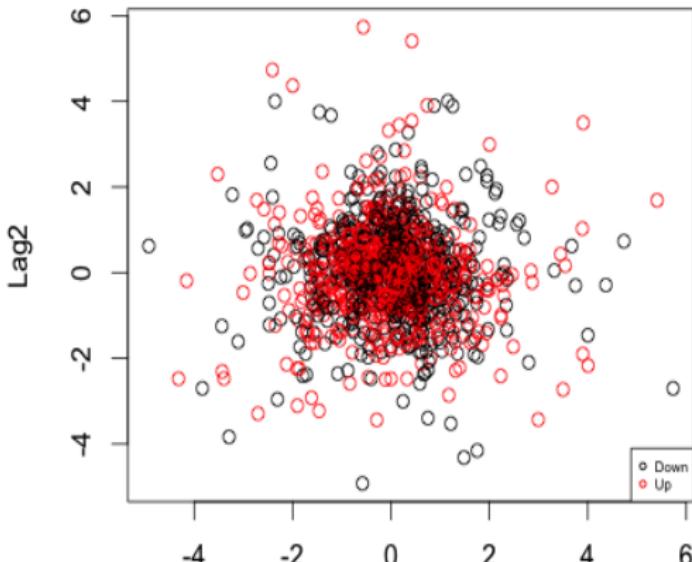
- ▶ **Year:** The year that the observation was recorded
- ▶ **LagX:** Percentage return X days ago.
- ▶ **Volume:** Volume of shares traded (number of daily shares traded in billions)
- ▶ **Today:** Percentage return for today
- ▶ **Direction:** A factor with levels Down and Up indicating whether the market had a positive or negative return on a given day.

```
mean(Smarket$Lag1[Smarket$Direction == "Up"])
mean(Smarket$Lag1[Smarket$Direction == "Down"])
```

# CLASSIFICATION: AN EXAMPLE

```
plot(Smarket$Lag1, Smarket$Lag2, col = Smarket$Direction,  
      xlab="Lag1", ylab="Lag2", main="Today's Direction")  
legend("bottomright", legend = levels(Smarket$Direction),  
      col=1:length(levels(Smarket$Direction)), pch=1)
```

Today's Direction



# CLASSIFICATION: AN EXAMPLE

## Procedure

For a new point  $(Lag1_{new}, Lag2_{new})$ ,

- ▶ Calculate the Euclidean distance between the new point and all data points. For a data point  $(L1, L2)$ ,

$$\text{dist}^2 = ((Lag1_{new} - L1)^2 + (Lag2_{new} - L2)^2).$$

- ▶ Create the set  $\mathcal{N}_{new}$  containing the  $K$  closest points.
- ▶ Determine the number of ‘UPs’ and ‘DOWNs’ in  $\mathcal{N}_{new}$  and classify the new point.

# CLASSIFICATION: AN EXAMPLE

## Coding The Procedure

```
K           <- 5
L1.new     <- 2
L2.new     <- 4.25

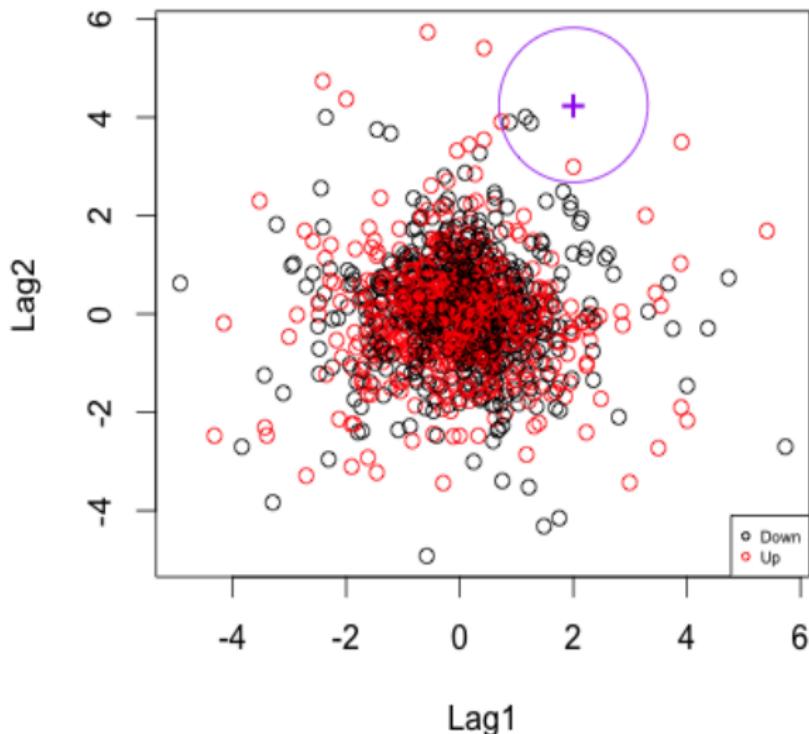
# K = 5 and new point (2, 4.25).

dists      <- sqrt((Smarket$Lag1 - L1.new)^2
                  + (Smarket$Lag2 - L2.new)^2)

neighbors  <- order(dists)[1:K]
neighb.dir <- Smarket$Direction[neighbors]
choice     <- names(which.max(table(neighb.dir)))
choice
```

# CLASSIFICATION: AN EXAMPLE

## Today's Direction



# CHECK YOURSELF

## Task

Write a function, `KNNclass`, that returns the class decision for any new point  $(Lag1_{new}, Lag2_{new})$  and any choice of  $K$ , with  $K = 5$  as default.

# TESTING OUR MODEL

```
KNNclass <- function(NewPoint, K=5, Lags = Weekly[, 2:6], Dir = Weekly$Direction) {  
  n <- nrow(Lags)  
  stopifnot(length(NewPoint) == 5, ncol(Lags) == 5, K <= n)  
  
  dists <- rowSums((Lags - rep(NewPoint, each = n))^2)  
  neighbors <- order(dists)[1:K]  
  neighb.dir <- Dir[neighbors]  
  choice <- names(which.max(table(neighb.dir)))  
  return(choice)  
}
```

Let's build our model using data from 2001 - 2004 and use the 2005 data as a test. Can we predict market direction better than a random guess?

```
NewPoint <- c(-.5, .5, -.5, -.5, .5)  
KNNclass(NewPoint)  
  
test <- Weekly[Weekly$Year >= 2009, ]  
train <- Weekly[Weekly$Year < 2009, ]  
  
n.test <- nrow(test)  
predictions <- rep(NA, n.test)  
  
for (i in 1:n.test) {  
  newp <- as.numeric(test[i, 2:6])  
  predictions[i] <- KNNclass(newp, Lags = train[, 2:6], Dir = train  
  $Direction)  
}  
  
test.error <- mean(predictions != test$Direction)  
test.error
```

# TESTING OUR MODEL

```
test  <- Smarket[Smarket$Year == 2005, ]
train <- Smarket[Smarket$Year != 2005, ]

n.test      <- nrow(test)
predictions <- rep(NA, n.test)

for (i in 1:n.test){
  predictions[i] <- KNNclass(test$Lag1[i], test$Lag2[i],
                               L1 = train$Lag1, L2 = train$Lag2,
                               Dir = train$Direction)
}

test.error <- mean(predictions != test$Direction)
test.error
```

## TESTING OUR MODEL: ANOTHER TRY

```
test  <- Smarket[Smarket$Year == 2005, ]
train <- Smarket[Smarket$Year != 2005, ]

n.test      <- nrow(test)
predictions <- rep(NA, n.test)

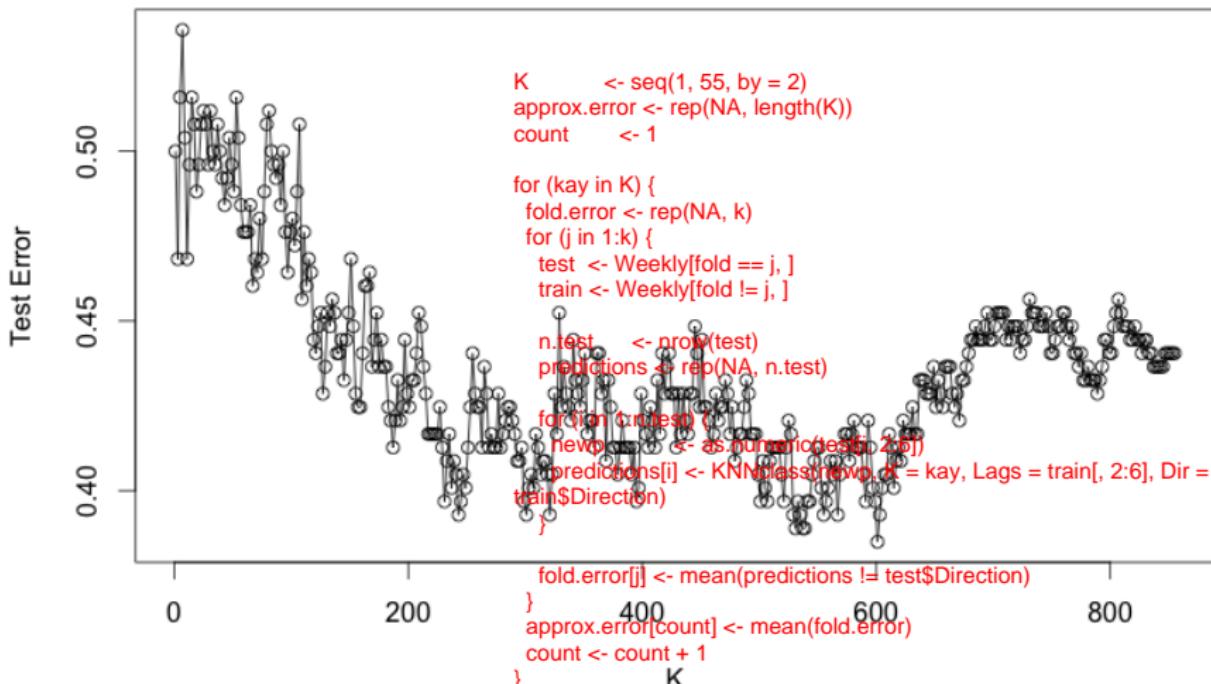
for (i in 1:n.test){
  predictions[i] <- KNNclass(test$Lag1[i], test$Lag2[i],
                               K = 7, L1 = train$Lag1,
                               L2 = train$Lag2,
                               Dir = train$Direction)
}

test.error <- mean(predictions != test$Direction)
test.error
```

# CLASSIFICATION: AN EXAMPLE

```
approx.error[1:4]
plot(K, approx.error, xlab = "K", lty = 1, ylab = "Approximate Test Error",
     main = "Predicting Market Direction", pch = 1)
lines(K, approx.error, lty = 1)
```

## Predicting Market Direction



# Lecture 6: Cross-validation and Visualization

## STAT GR5206

*Statistical Computing & Introduction to Data Science*

Cynthia Rush  
Columbia University

October 13, 2017

# COURSE NOTES

- ▶ Homework 2 due.
- ▶ Midterm November 3.
- ▶ One more homework before the midterm.
- ▶ Please visit office hours if you have questions about homework/lecture/lab.

# CLASSIFICATION

# WHAT IS CLASSIFICATION?

- ▶ Linear regression is used when we want to predict a quantitative response variable.
- ▶ What if we have a categorical response variable?
- ▶ The study of predicting categorical response variables is called **classification**.

## Examples

- ▶ A person has symptoms that could possibly be attributed to one of three medical conditions. Which condition does he have?
- ▶ A bank must be able to determine whether an online transaction is fraudulent, on the basis of the user's IP address, past transaction history, etc.
- ▶ Using DNA sequence data for patients with and without a disease, a biologist would like to figure out which DNA mutations cause diseases and which do not.

# TYPES OF CLASSIFIERS

Many methods of classification:

- ▶ Logistic Regression.
- ▶ Linear Discriminant Analysis.
- ▶ K Nearest Neighbors.
- ▶ Trees and Random Forests.
- ▶ Support Vector Machines.

## Set-up

- ▶ Just as in regression, we have a set of training observations (data):  
$$(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n), \quad Y_1, Y_2, \dots, Y_n \text{ categorical.}$$
- ▶ Use the data to build the classifier.
- ▶ We want our classifier to perform well not only on the training data, but also on test observations that were not used to build the classifier.

# ASSESSING THE ACCURACY

Recall, for our model earlier, we calculate the **training** mean squared error:

$$\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{\beta}_0 X_{i1}^{\hat{\beta}_1})^2.$$

For classification, we study the **error rate**:

$$\frac{1}{n} \sum_{i=1}^n \mathbb{I}[Y_i \neq \hat{Y}_i].$$

- ▶  $\hat{Y}_i$  is the predicted classification for the  $i^{th}$  observation.
- ▶  $\mathbb{I}[Y_i \neq \hat{Y}_i]$  is an *indicator variable* that equals 1 if  $Y_i \neq \hat{Y}_i$  and 0 if  $Y_i = \hat{Y}_i$ .
- ▶ If  $\mathbb{I}[Y_i \neq \hat{Y}_i] = 0$  then the  $i^{th}$  observation was classified correctly.
- ▶ The *error rate* computed the fraction of incorrect classifications.

# ASSESSING THE ACCURACY

- ▶ Similarly to the model discussed previously, the following is referred to as the **training** error rate:

$$\frac{1}{n} \sum_{i=1}^n \mathbb{I}[Y_i \neq \hat{Y}_i].$$

- ▶ The **test** error rate associated with a set of **test** observations  $(X_{test}, Y_{test})$  is given by

$$Ave(\mathbb{I}[Y_{test} \neq \hat{Y}_{test}]).$$

- ▶ A good classifier is one for which the **test** error rate is the smallest.

# $K$ NEAREST NEIGHBORS

In theory would like to always use the Bayes classifier. In practice, don't know  $Pr(Y|X)$ !

## $K$ Nearest Neighbors (KNN)

- ▶ Estimates  $Pr(Y|X)$  and then classifies observations to the class with highest estimated probability.
- ▶ Given a positive integer  $K$  and a test observation  $X_{test}$ :
  - ▶ Identify  $K$  points in training data closest to  $X_{test}$ . Label  $\mathcal{N}_{test}$ .
  - ▶ Estimate conditional probability for class  $j$  as fraction of points in  $\mathcal{N}_{test}$  whose response values equal  $j$ :

$$Pr(Y = j | X = X_{test}) = \frac{1}{K} \sum_{i \in \mathcal{N}_{test}} \mathbb{I}(Y_i = j).$$

- ▶ Classify the test observation to class with the largest probability.

# CLASSIFICATION: AN EXAMPLE

Can we predict the direction of the stock market today using the previous two days' movements?

```
# install.packages("ISLR")
library(ISLR)
head(Smarket, 3)
```

# CLASSIFICATION: AN EXAMPLE

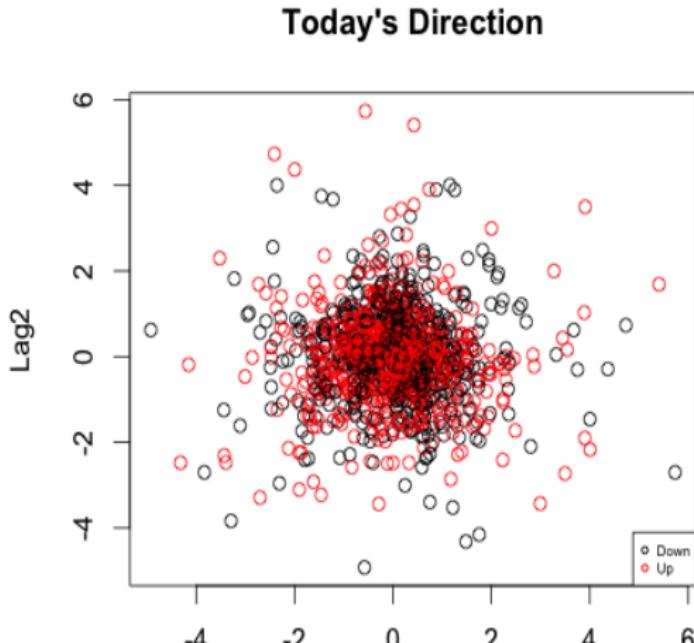
## Variables

- ▶ **Year:** The year that the observation was recorded
- ▶ **LagX:** Percentage return X days ago.
- ▶ **Volume:** Volume of shares traded (number of daily shares traded in billions)
- ▶ **Today:** Percentage return for today
- ▶ **Direction:** A factor with levels Down and Up indicating whether the market had a positive or negative return on a given day.

```
mean(Smarket$Lag1[Smarket$Direction == "Up"])
mean(Smarket$Lag1[Smarket$Direction == "Down"])
```

# CLASSIFICATION: AN EXAMPLE

```
plot(Smarket$Lag1, Smarket$Lag2, col = Smarket$Direction,  
      xlab="Lag1", ylab="Lag2", main="Today's Direction")  
legend("bottomright", legend = levels(Smarket$Direction),  
      col=1:length(levels(Smarket$Direction)), pch=1)
```



# CLASSIFICATION: AN EXAMPLE

## Procedure

For a new point  $(Lag1_{new}, Lag2_{new})$ ,

- ▶ Calculate the Euclidean distance between the new point and all data points. For a data point  $(L1, L2)$ ,

$$\text{dist}^2 = ((Lag1_{new} - L1)^2 + (Lag2_{new} - L2)^2).$$

- ▶ Create the set  $\mathcal{N}_{new}$  containing the  $K$  closest points.
- ▶ Determine the number of ‘UPs’ and ‘DOWNs’ in  $\mathcal{N}_{new}$  and classify the new point.

# CLASSIFICATION: AN EXAMPLE

## Coding The Procedure

```
K           <- 5
L1.new     <- 2
L2.new     <- 4.25

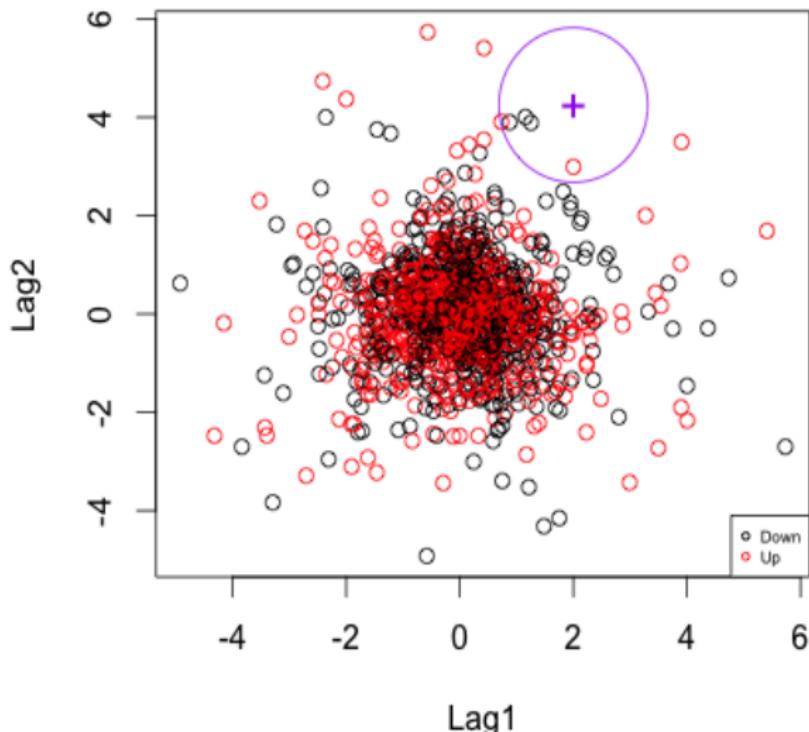
# K = 5 and new point (2, 4.25).

dists      <- sqrt((Smarket$Lag1 - L1.new)^2
                  + (Smarket$Lag2 - L2.new)^2)

neighbors  <- order(dists)[1:K]
neighb.dir <- Smarket$Direction[neighbors]
choice     <- names(which.max(table(neighb.dir)))
choice
```

# CLASSIFICATION: AN EXAMPLE

## Today's Direction



# CHECK YOURSELF

## Task

Write a function, `KNNclass`, that returns the class decision for any new point  $(Lag1_{new}, Lag2_{new})$  and any choice of  $K$ , with  $K = 5$  as default.

# CHECK YOURSELF

One Solution:

```
KNNclass <- function(L1.new, L2.new, K = 5, L1 = Smarket$Lag1,
                      L2 = Smarket$Lag2, Dir = Smarket$Direction)

  n <- length(L1)
  stopifnot(length(L2) == n, length(L1.new) == 1,
            length(L2.new) == 1, K <= n)

  dists <- sqrt((L1-L1.new)^2 + (L2-L2.new)^2)

  neighbors <- order(dists)[1:K]
  neighb.dir <- Dir[neighbors]
  choice      <- names(which.max(table(neighb.dir)))
  return(choice)
}
```

# TESTING OUR MODEL

Let's build our model using data from 2001 - 2004 and use the 2005 data as a test. Can we predict market direction better than a random guess?

# TESTING OUR MODEL

```
test  <- Smarket[Smarket$Year == 2005, ]
train <- Smarket[Smarket$Year != 2005, ]

n.test      <- nrow(test)
predictions <- rep(NA, n.test)

for (i in 1:n.test){
  predictions[i] <- KNNclass(test$Lag1[i], test$Lag2[i],
                               L1 = train$Lag1, L2 = train$Lag2,
                               Dir = train$Direction)
}

test.error <- mean(predictions != test$Direction)
test.error
```

## TESTING OUR MODEL: ANOTHER TRY

```
test  <- Smarket[Smarket$Year == 2005, ]
train <- Smarket[Smarket$Year != 2005, ]

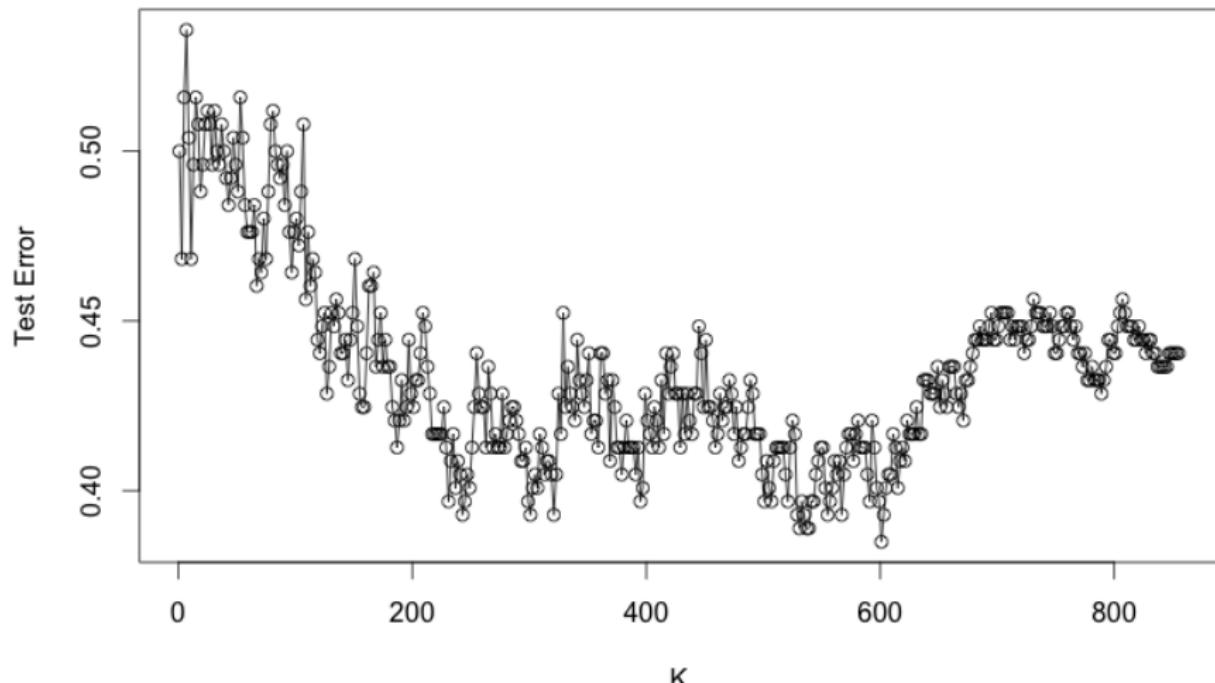
n.test      <- nrow(test)
predictions <- rep(NA, n.test)

for (i in 1:n.test){
  predictions[i] <- KNNclass(test$Lag1[i], test$Lag2[i],
                               K = 7, L1 = train$Lag1,
                               L2 = train$Lag2,
                               Dir = train$Direction)
}

test.error <- mean(predictions != test$Direction)
test.error
```

# CLASSIFICATION: AN EXAMPLE

## Predicting Market Direction



# CROSS-VALIDATION

# CROSS VALIDATION

## Objective

- ▶ Cross validation is a method which tries to select the best model from a given set of models.
- ▶ Assumption: Quality measure is predictive performance.
- ▶ “Set of models” can simply mean “set of different parameter values”.

## Terminology

The problem of choosing a good model is called **model selection**.

# SPECIFICALLY: KNN

## Model selection problem for KNN

- ▶ The KNN is a *family* of models indexed by the parameter  $K$ , the number of neighbors.
- ▶ Our goal is to find a value of  $K$  for which we can expect small test error.
- ▶ recall the difference between training and test errors.

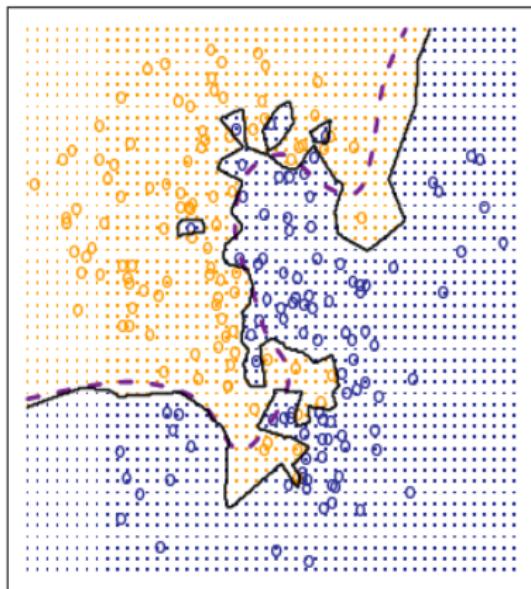
If we aren't careful...

- ▶ **Overfitting:** The classifier adapts too closely to specific properties of the training data, rather than the underlying distribution.

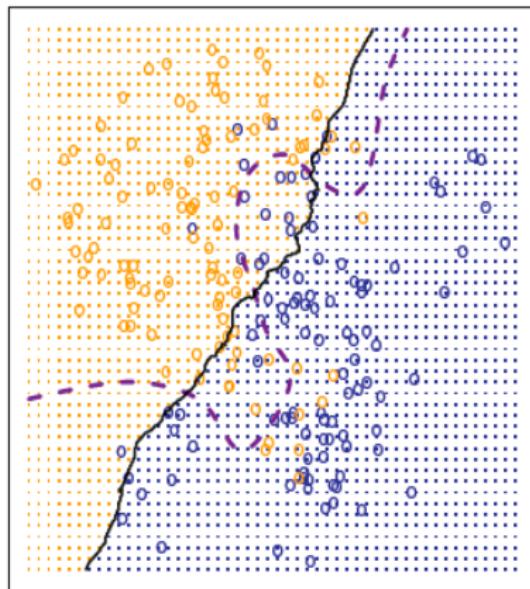
# KNN CLASSIFIER EXAMPLE <sup>1</sup>

Training error when  $K = 100$  is small, but test error won't be.

KNN: K=1



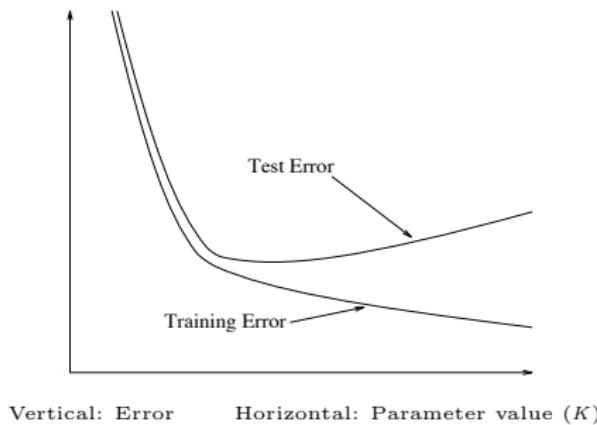
KNN: K=100



<sup>1</sup>Image from 'Introduction to Statistical Learning'.

# TRAINING VS TEST ERROR

## Conceptual illustration



- ▶ If classifier can adapt (too) well to data: Small training error, but possibly large test error.
- ▶ If classifier can hardly adapt at all: Large training and test error.
- ▶ Somewhere in between, there is a sweet spot.
- ▶ Trade-off is controlled by the parameter.

# MODEL SELECTION BY CROSS VALIDATION

## Cross Validation: Procedure

### Model selection:

1. Randomly split data into two sets: training and validation.
2. Train classifier on training data for different values of  $K$ .
3. Evaluate each trained classifier on validation data (ie compute error rate).
4. Select the value of  $K$  with lowest error rate.

# INTERPRETATION

## Meaning

- ▶ The quality measure by which we are comparing different classifiers (for different parameter values  $K$ ) is the test error

$$Ave(\mathbb{I}[Y_{test} \neq \hat{Y}_{test}]) .$$

- ▶ Since we do not know the true value, we estimate it from the data as  $err = \frac{1}{n} \sum_{i=1}^n \mathbb{I}[Y_i \neq \hat{Y}_i]$ .

# CROSS VALIDATION

## Procedure in detail

We consider possible parameter values  $K_1, \dots, K_m$ .

1. For each value  $K_j$ , train a classifier on the training set.
2. Use the validation set to estimate test error as the empirical error

$$err_{K_j, val} = \frac{1}{n_{val}} \sum_{i=1}^{n_{val}} \mathbb{I}[Y_i \neq \hat{Y}_i].$$

$n_{val}$  is the size of the validation set.

3. Select the value  $K^*$  which achieves the smallest estimated error.
4. Re-train the classifier with parameter  $K^*$  on all data (i.e. on training + validation data).

# $k$ -FOLD CROSS VALIDATION

## Idea

Each of the error estimates computed on validation set is computed from a single example of a trained classifier. Can we improve the estimate?

## Strategy

- ▶ Set aside the test set.
- ▶ Split the remaining data into  $k$  blocks.
- ▶ Use each block in turn as validation set. Perform cross validation and average the results over all  $k$  combinations.

This method is called  **$k$ -fold cross validation**.

Example:  $k=5$ , step 3

| 1     | 2     | 3          | 4     | 5     |
|-------|-------|------------|-------|-------|
| Train | Train | Validation | Train | Train |

# $k$ -FOLD CROSS VALIDATION: PROCEDURE

## Risk estimation

To estimate the risk of a classifier built for  $K = K_j$ :

1. Split data into  $k$  equally sized blocks.
2. For each fold  $k$ :
  - ▶ Train the classifier using all blocks except block  $k$  as training data.
  - ▶ Then calculate the error of the classifier on block  $k$  data,  $err_{K_j, \text{block } k}$ .
3. Compute the cross validation estimate

$$err_{K_j, CV} := \frac{1}{k} \sum_{i=1}^k err_{K_j, \text{block } k}.$$

Repeat this for all parameter values  $K_1, \dots, K_m$ .

## Selecting a model

Choose the parameter value  $K^*$  for which estimated risk is minimal.

## EXAMPLE: KNN

```
library(ISLR)
# From lab:
KNNclass <- function(NewPoint, K = 5, Lags = Weekly[, 2:6],
                      Dir = Weekly$Direction) {

  n <- nrow(LagData)

  stopifnot(length(NewPoint) == 5, ncol(Lags) == 5, K <= n)

  dists      <- rowSums((Lags - rep(NewPoint, each = n))^2)
  neighbors  <- order(dists)[1:K]
  neighb.dir <- Dir[neighbors]
  choice     <- names(which.max(table(neighb.dir)))
  return(choice)
}

# A test point: NewPoint = c(-.5, .5, -.5, -.5, .5)
KNNclass(c(-.5, .5, -.5, -.5, .5))
```

## EXAMPLE: TEST AND TRAINING DATA

```
test  <- Weekly[Weekly$Year >= 2009, ]
train <- Weekly[Weekly$Year < 2009, ]

n.test      <- nrow(test)
predictions <- rep(NA, n.test)

for (i in 1:n.test){
  newp <- as.numeric(test[i, 2:6])
  predictions[i] <- KNNclass(newp, Lags = train[, 2:6],
                               Dir = train$Direction)
}

test.error <- mean(predictions != test$Direction)
test.error
```

## EXAMPLE: 9-FOLD CROSS-VALIDATION

```
# Divide data into k=9 folds

k      <- 9
nums <- rep(1:k, each = nrow(Weekly)/k)
fold <- sample(nums)
```

## EXAMPLE: 9-FOLD CROSS-VALIDATION

```
# Iterate over the folds
fold.error <- rep(NA, k)
for (j in 1:k) {
  test <- Weekly[fold == j, ]
  train <- Weekly[fold != j, ]

  n.test      <- nrow(test)
  predictions <- rep(NA, n.test)
  for (i in 1:n.test){
    newp <- as.numeric(test[i, 2:6])
    predictions[i] <- KNNclass(newp, Lags = train[, 2:6],
                                Dir = train$Direction)
  }
  fold.error[j] <- mean(predictions != test$Direction)
}
approx.error <- mean(fold.error)
approx.error
```

## EXAMPLE: MODEL SELECTION

```
K           <- seq(1, 55, by = 2)
approx.error <- rep(NA, length(K))
count        <- 1
```

## EXAMPLE: MODEL SELECTION

```
for (kay in K) {  
  fold.error <- rep(NA, k)  
  for (j in 1:k) {  
    test  <- Weekly[fold == j, ]  
    train <- Weekly[fold != j, ]  
  
    n.test      <- nrow(test)  
    predictions <- rep(NA, n.test)  
  
    for (i in 1:n.test){  
      newp <- as.numeric(test[i, 2:6])  
      out <- KNNclass(newp, K= kay, Lags = train[, 2:6],  
                        Dir = train$Direction)  
      predictions[i] <- out  
    }  
    fold.error[j] <- mean(predictions != test$Direction)  
  }  
  approx.error[count] <- mean(fold.error)  
  count              <- count + 1  
}  
}
```

## EXAMPLE: MODEL SELECTION

```
approx.error[1:4]
plot(K, approx.error, xlab = "K", lty = 1,
      ylab = "Approximate Test Error",
      main = "Predicting Market Direction", pch = 1)
lines(K, approx.error, lty = 1)
```

# SOME MORE PLOTTING WITH BASE R

# BASICS OF PLOTTING

Recall,

- ▶ Visualization variation (of a single variable):
  - ▶ `hist()` – Histograms.
  - ▶ `barplot()` – Bargraphs.
- ▶ Visualizing covariation (of multiple variables):
  - ▶ `plot()` – Scatterplots.
  - ▶ `boxplot()` – Boxplots (box-and-whisker plots).

# BASICS OF PLOTTING

## The `plot()` function.

- ▶ The foundation of many of R's graphics functions.
- ▶ Often one builds up the graph in stages with `plot()` as a base.
- ▶ Each call to `plot()` begins a new graph window.
- ▶ Takes arguments, called **graphical parameters**, to change various aspects of the plot. (`?par`)

# DIAMONDS DATASET

First, we create a smaller dataset from `diamonds` by randomly selecting 1000 rows.

```
diamonds <- read.csv("diamonds.csv", as.is = T)
lev_vec <- c("Fair", "Good", "Very Good", "Premium", "Ideal")
diamonds$cut <- factor(diamonds$cut, level = lev_vec)
diamonds$color <- factor(diamonds$color)
diamonds$clarity <- factor(diamonds$clarity)

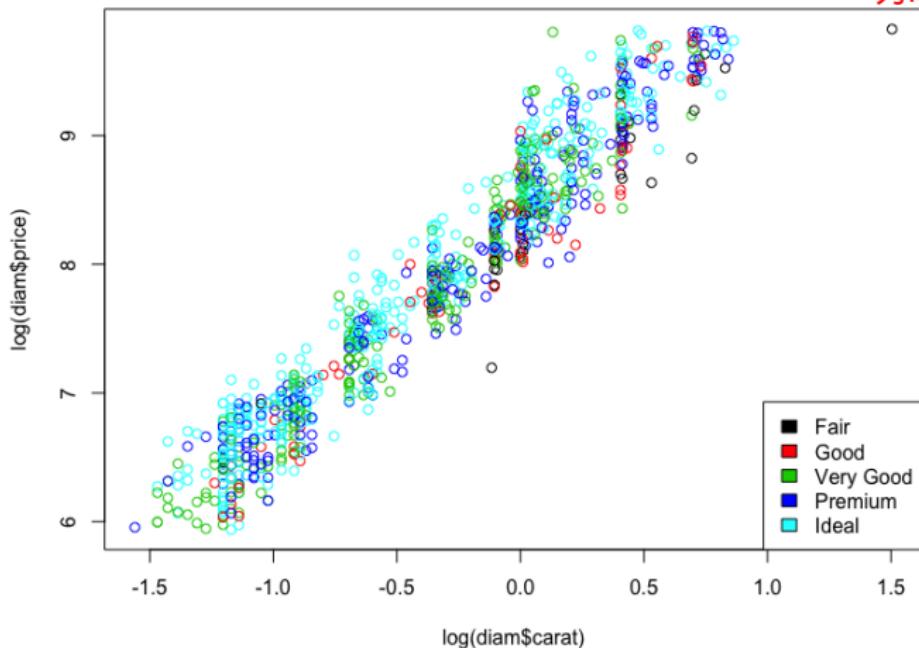
set.seed(1)

rows <- dim(diamonds)[1]
diam <- diamonds[sample(1:rows, 1000), ]
```

# BUILDING A VISUALIZATION: AN EXAMPLE

```
plot(log(diam$carat), log(diam$price), col = diam$cut)
legend("bottomright", legend = levels(diam$cut), fill = 1:length(levels(diam$cut)))
```

设置为col的变量最好提前转化为factor变量



# HOW DOES COLOR WORK?

```
plot(1:5, 1:5, col = "red")
plot(1:5, 1:5, col = "red", pch = 19)
plot(1:5, 1:5, col = 2, pch = 19)
plot(1:5, 1:5, col = 1:5, pch = 19)
legend("bottomright", legend = 1:5, fill = 1:5)
```

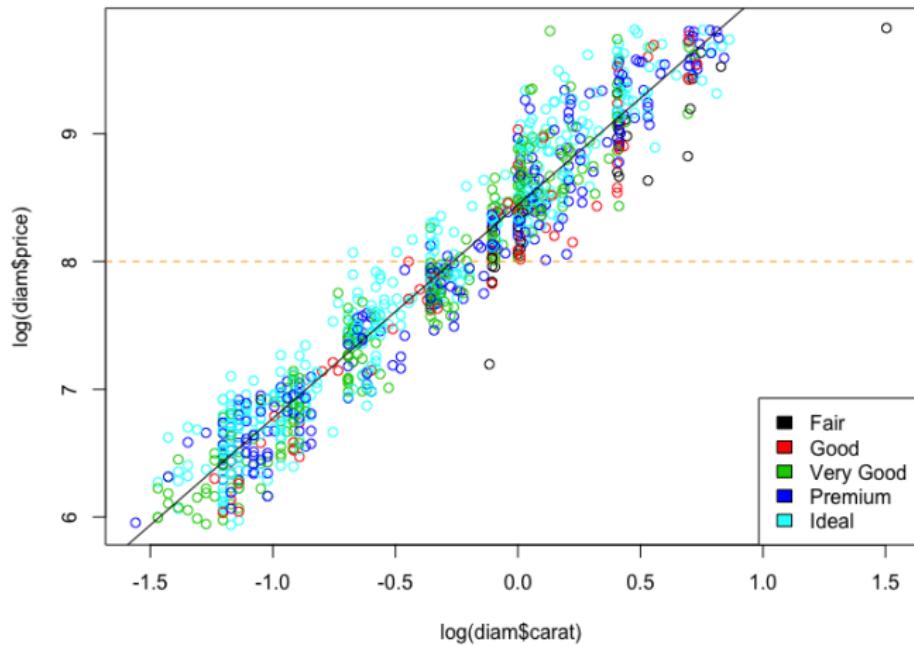
# BUILDING A VISUALIZATION: AN EXAMPLE

## Adding Lines to a Scatterplot

- ▶ Add a straight line with `abline(int, slope)`.
  - ▶ `int` is the intercept of the line.
  - ▶ `slope` is the slope of the line.
- ▶ `lines()` can also be used.
  - ▶ Most simply, pass to the `lines()` function `x` and `y` vectors and it connects the points.
  - ▶ These are interpreted as  $(x,y)$  pairs which are plotted on the graph with lines connecting them.

# BUILDING A VISUALIZATION: AN EXAMPLE

```
abline(8, 0, col = "orange", lty = 2)
lm1 <- lm(log(diam$price) ~ log(diam$carat))
abline(lm1)
```



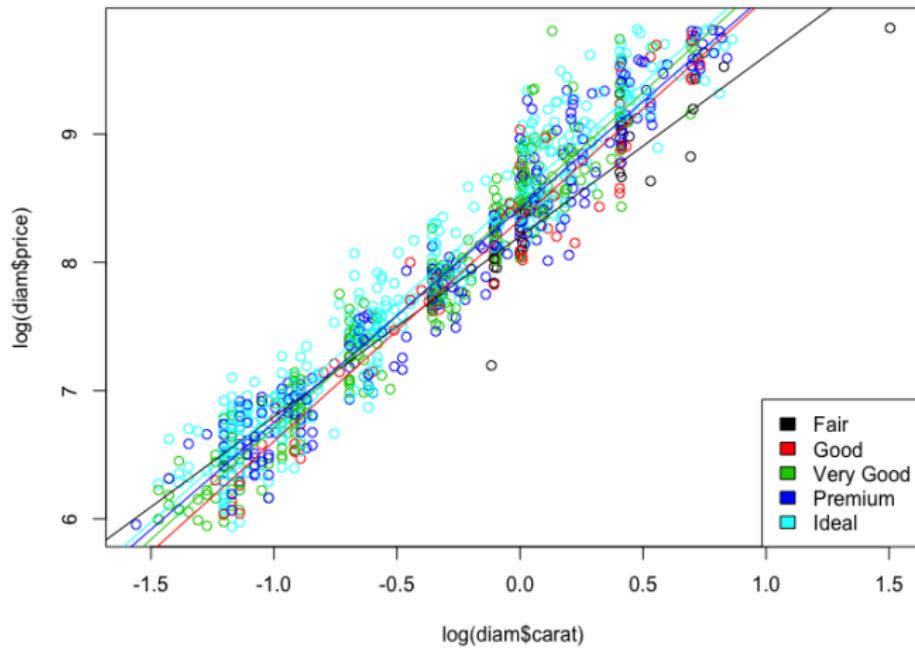
# BUILDING A VISUALIZATION: AN EXAMPLE

Let's instead plot a regression line for each cut separately.

```
cuts      <- levels(diam$cut)
col_counter <- 1

for (i in cuts) {
  this_cut      <- diam$cut == i
  this_data     <- diam[this_cut, ]
  this_lm       <- lm(log(this_data$price) ~ log(this_data$carat))
  abline(this_lm, col = col_counter)
  col_counter <- col_counter + 1
}
```

# BUILDING A VISUALIZATION: AN EXAMPLE



# CHECK YOURSELF

```
diamonds$color2 <- "bad"
diamonds$color2[diamonds$color %in% c("D", "E", "F")] <- "good"
diamonds$color2 <- as.factor(diamonds$color2)

plot(log(diamonds$carat), log(diamonds$price), col = diamonds$color2)
legend("bottomright", legend = levels(diamonds$color2),
       fill = 1:length(levels(diamonds$color2)))
```

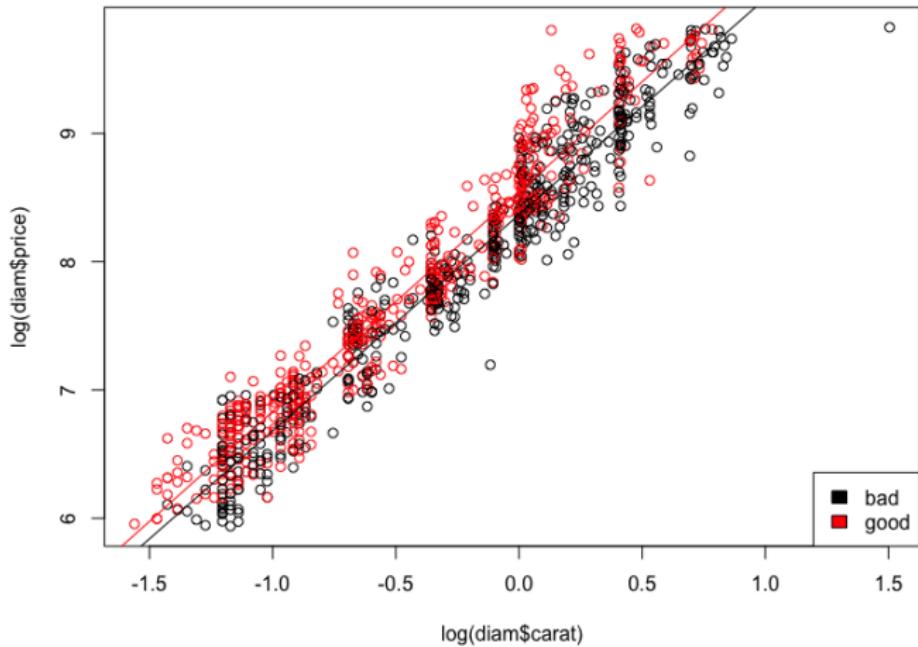
## Exercise:

- ▶ Make a new column in the `diam` dataset called `color2` that is a factor variable with levels `good` and `bad` where `good` corresponds to color levels "D" - "F" and `bad` the others. You shouldn't write a loop here.
- ▶ Plot `log(diam$carat)` on the x-axis and `log(diam$price)` on the y-axis. Color the points according to whether the color is good or bad.
- ▶ Plot two regression lines on the plot, one for good color and one for bad color.

```
cols      <- levels(diamonds$color2)
col_counter <- 1

for (i in cols) {
  this_col  <- diamonds$color2 == i
  this_data <- diamonds[this_col, ]
  this_lm   <- lm(log(this_data$price) ~ log(this_data$carat))
  abline(this_lm, col = col_counter)
  col_counter <- col_counter + 1
}
```

# EXERCISE



# BUILDING A VISUALIZATION: AN EXAMPLE

## Adding Points to a Scatterplot

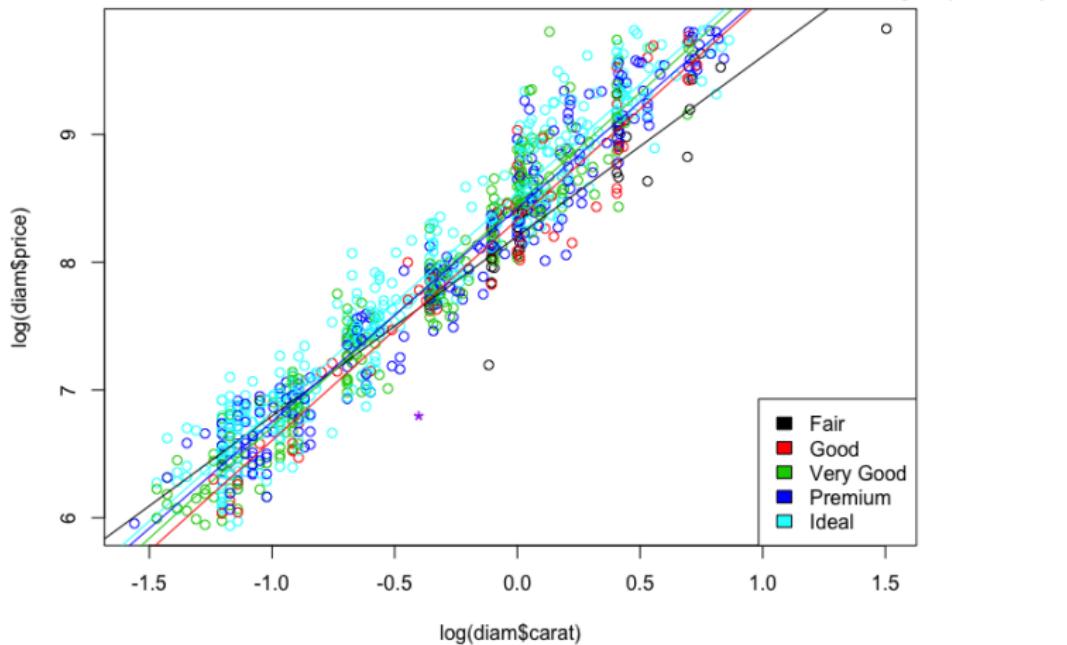
- ▶ Most easily done with `points(x,y)`.
- ▶  $(x,y)$  is the location of the point to be added.
- ▶ `example(points)` could be helpful.

# BUILDING A VISUALIZATION: AN EXAMPLE

We add a new point for a diamond that is \$898 and 0.67 carats.

```
points(-0.4, 6.8, pch = "*", col = "purple", cex = 1.5)
```

cex表示的是大小



# BUILDING A VISUALIZATION: AN EXAMPLE

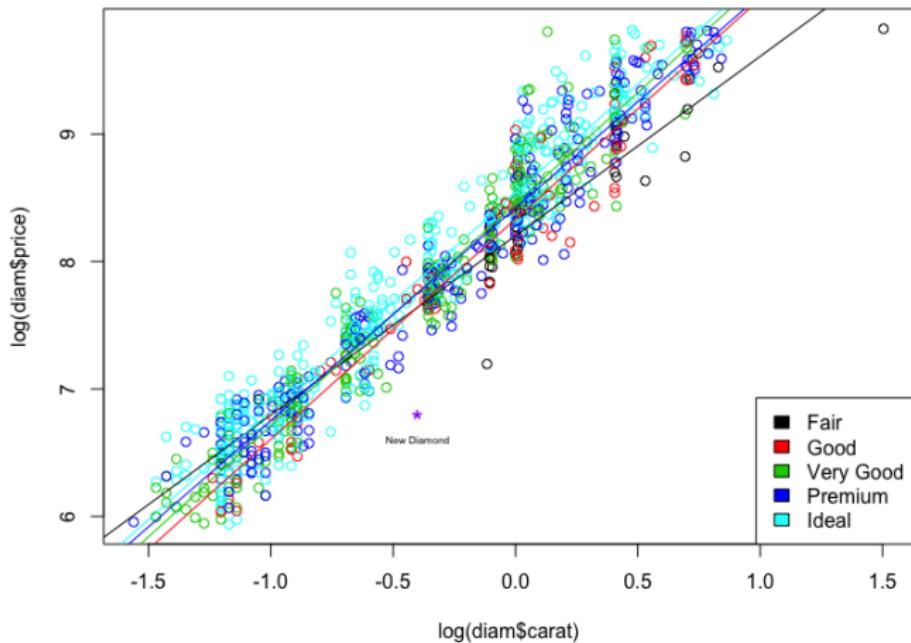
## Adding Text to a Scatterplot

- ▶ Most easily done with `text(x,y, label)`.
  - ▶ `(x,y)` is the location of the text to be added.
  - ▶ `label` is the the text to be added at the specified location.
- ▶ The `locator()` function we saw in Lecture 1 can be useful here.

# BUILDING A VISUALIZATION: AN EXAMPLE

We add text to the new point we just added.

```
text(-0.4, 6.8 - .2, "New Diamond", cex = .5)
```



# USEFUL GRAPHICAL PARAMETERS

The table below lists a selection of R's graphical parameters. More info at <http://www.statmethods.net/advgraphs/parameters.html> or using `?par`.

| Parameter               | Description  |
|-------------------------|--|
| <code>pch</code>        | <b>Point Character.</b> Character of the points in the plot.     |
| <code>main</code>       | Title of the plot.   |
| <code>xlab, ylab</code> | Axes labels.   |
| <code>lty</code>        | <b>Line Type.</b> E.g. 'dashed', 'dotted', etc.                  |
| <code>lwd</code>        | <b>Line Width.</b> Line width relative to default = 1.           |
| <code>cex</code>        | <b>Character Expand.</b> Character size relative to default = 1. |
| <code>xlim, ylim</code> | The limits of the axes.  |
| <code>mfrow</code>      | Plot figures in an array (e.g. next to each other).              |
| <code>col</code>        | Plotting color.  |

# ADVANCED VISUALIZATION TECHNIQUES

# GGPLOT2

- ▶ R has several systems for making graphs (we've looked at the base R functions).
- ▶ `ggplot2` is one of the most elegant and flexible.
- ▶ `ggplot2` uses a coherent system (or 'grammar') for describing and building graphs.

## GGPLOT2

- ▶ R has several systems for making graphs (we've looked at the base R functions).
- ▶ `ggplot2` is one of the most elegant and flexible.
- ▶ `ggplot2` uses a coherent system (or 'grammar') for describing and building graphs.

```
install.packages("ggplot2")
library("ggplot2")
```

## GGPLOT2

We study `ggplot2` using the `mpg` dataset. Let's try to answer the question: do cars with bigger engines use more fuel than cars with small engines?

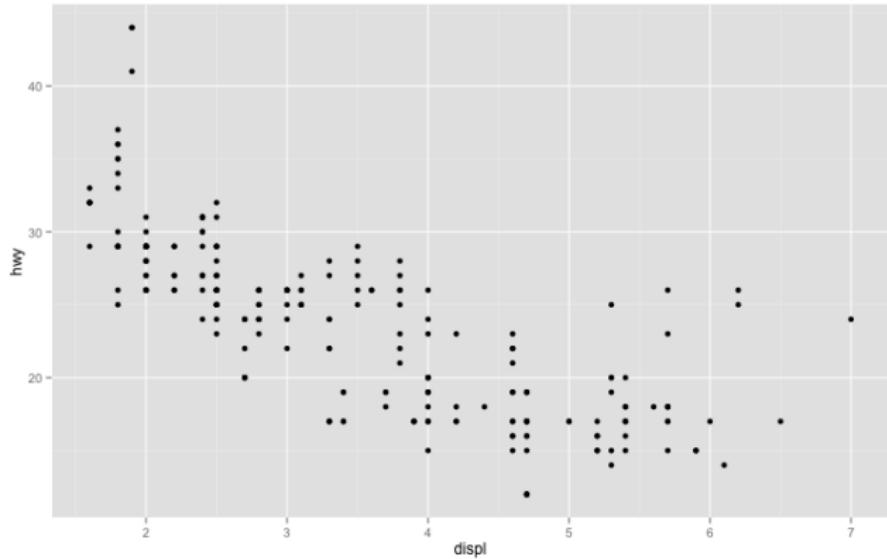
Read about the data using `?mpg`.

```
dim(mpg)  
head(mpg, 3)
```

We look at `displ`, a car's engine size in liters, and `hwy`, a car's fuel efficiency on the highway in miles per gallon (mpg).

# A FIRST PLOT

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))
```



# A FIRST PLOT

Let's break apart the code:

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))
```

- ▶ Begin a plot with `ggplot()`.
  - ▶ It creates the coordinate axis that you add to.
  - ▶ The first argument is the dataset
- ▶ Next you want to add layers to the plot.
  - ▶ In our example: `geom_point()` adds a layer of points.
  - ▶ Lots of different `geom` functions doing different things.
- ▶ `geom` functions take `mapping` arguments.
  - ▶ Defines how variables in your dataset are mapped to visual properties.
  - ▶ Always paired with `aes()`.
  - ▶ The `x` and `y` arguments specify which variables to map to the axes.

# A FIRST PLOT

General structure:

```
ggplot(data = <DATA>) +  
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

To create a plot, replace the bracketed sections in the code above with a dataset, a `geom` function, and a set of mappings.

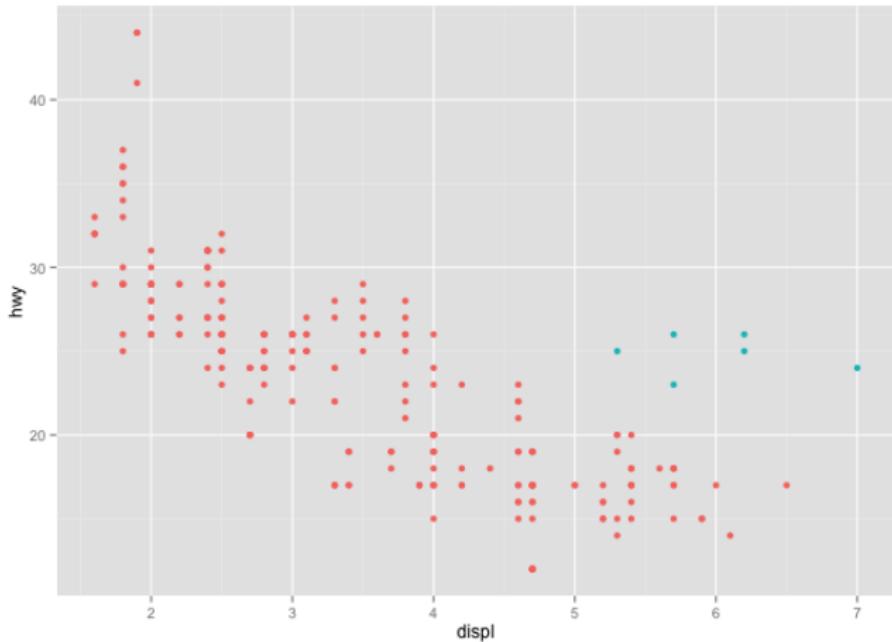
From this template, we can make many different kinds of graphs using `ggplot`.

# CHECK YOURSELF

## Tasks

- ▶ Plot just `ggplot(data = mpg)`. What do you get?
- ▶ Make a scatterplot of `hwy` vs. `cyl`.
- ▶ Make a scatterplot of `class` vs. `drv`. Why is this plot not useful?

# AESTHETIC MAPPINGS



The blue points seem to have a different trend than the rest – possibly hybrids? We study car **class** to find out.

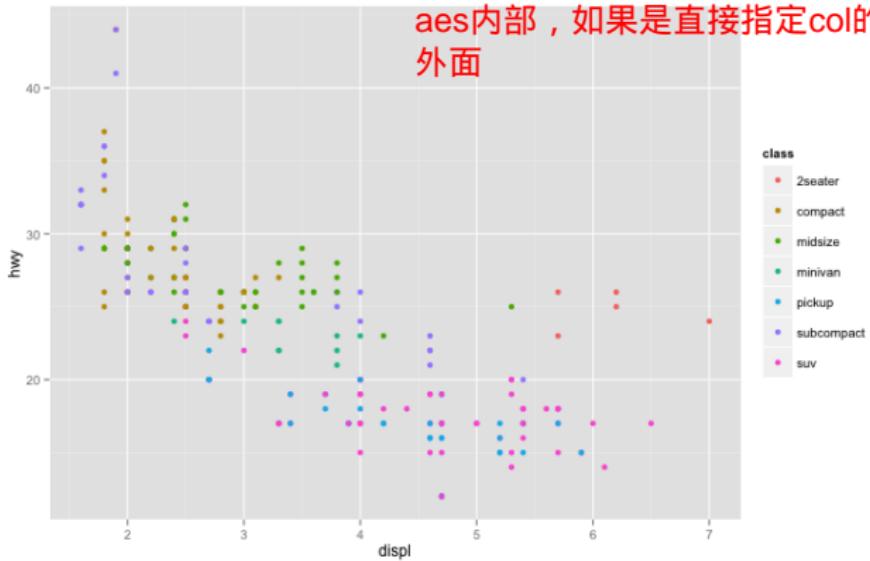
# AESTHETIC MAPPINGS

- ▶ We can add a third variable to a scatterplot by mapping it to an **aesthetic**.
- ▶ An **aesthetic** is a visual property of the objects in the plot.
- ▶ Things like size, color, shape of points.

# MAPPING AESTHETICS

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x=displ, y=hwy, color=class))
```

如果使用的是变量名作为col的值，需要放在  
aes内部，如果是直接指定col的值，放在aes  
外面



# CHECK YOURSELF

## Tasks

- ▶ Instead of mapping `class` to the `color` aesthetic, map it to the `alpha` aesthetic or the `size` aesthetic.
- ▶ Instead of mapping `class` to the `color` aesthetic, map it to the `shape` aesthetic. Note that `ggplot()` will only use 6 shapes at a time. What does this mean for our plot?
- ▶ What does the following code do?

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x=displ, y=hwy), color="blue")
```

- ▶ Map a continuous variable in the `mpg` dataset, like `cty`, to the `alpha`, `shape`, and `size` aesthetics. What does this do?

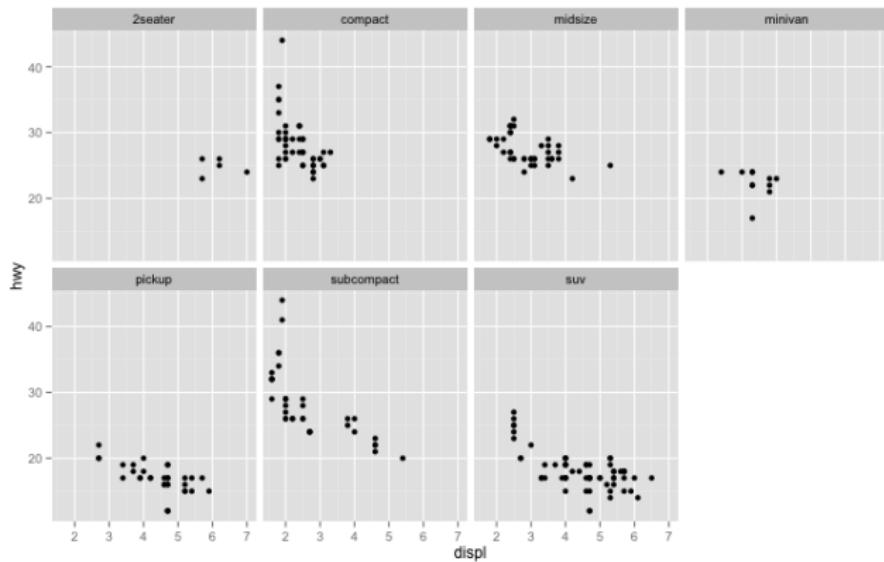
# FACETS

- ▶ We saw we could add categorical variables to plots using aesthetics.
- ▶ Can also do this by splitting the plot into **facets**, which are subplots that each display one subset of the data.
- ▶ Use the `fact_wrap()` command to facet a plot by a single variable.
- ▶ The argument is a formula created with `~` followed by a variable name.

facet\_wrap(~ class, nrow = 2) : 横着显示  
facet\_wrap(class ~, nrow = 2) : 竖着显示  
facet\_wrap(class~name) : 混合显示

# FACETS

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  facet_wrap(~ class, nrow = 2)
```



# CHECK YOURSELF

## Tasks

- ▶ Facet on two variables use the `facet_grid()` command. An example is the following:

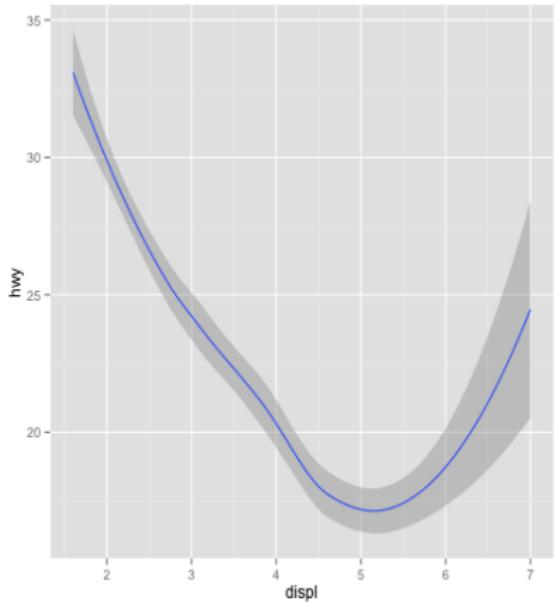
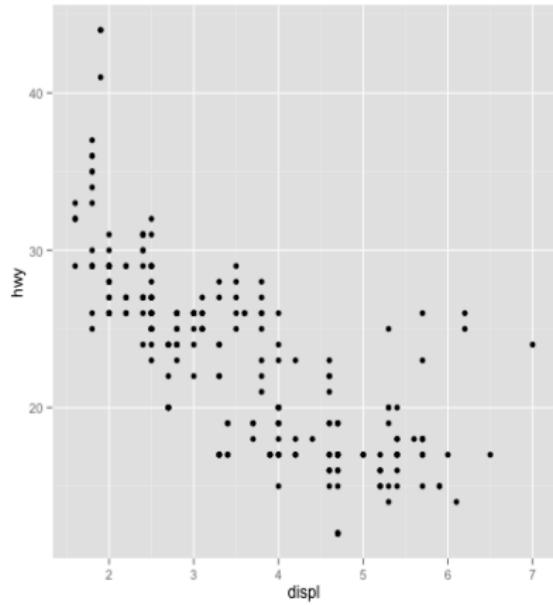
```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  facet_grid(drv ~ class)
```

What do the empty cells mean?

- ▶ Look at `?facet_wrap`. What do `nrow` and `ncol` do? Why doesn't `facet_grid()` have `nrow` and `ncol` arguments?
- ▶ What happens if you facet on a continuous variable?

# GEOMETRIC OBJECTS

geom\_smooth()

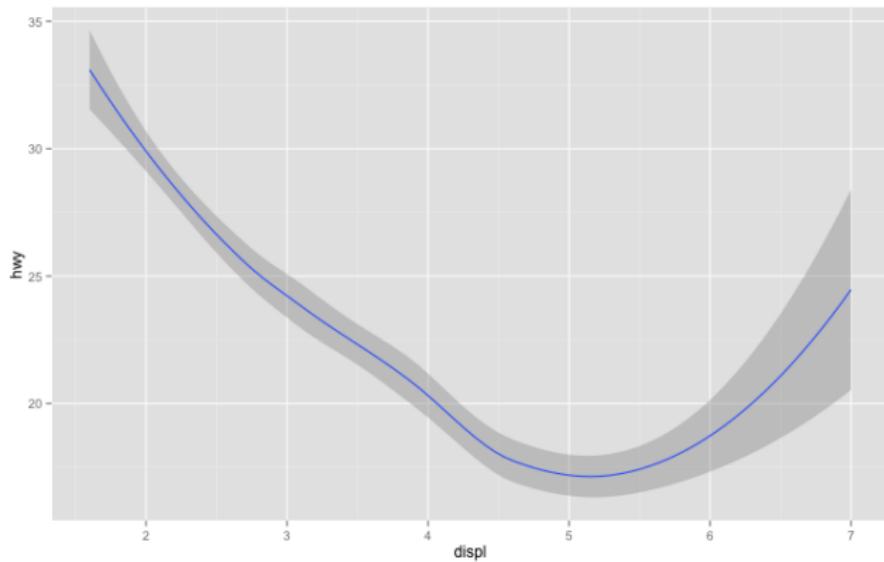


# GEOMETRIC OBJECTS

- ▶ In the previous slide, each plot used a different **visual object** to represent the data.
- ▶ Produce this by using different **geoms**.
- ▶ A **geom** is a geometrical object used to represent data in a plot.
- ▶ Often describe plots by the type of **geom** they use. For example, bar graphs use **bar geoms**.

# GEOMETRIC OBJECTS

```
ggplot(data = mpg) +  
  geom_smooth(mapping = aes(x = displ, y = hwy))
```



# GEOMETRIC OBJECTS

- ▶ Every `geom` takes a `mapping` argument but not every aesthetic works with every `geom`.
  - ▶ E.g., you can set the `shape` of a point, but not a line. You can set the `linetype` of a line.
- ▶ `ggplot2` has around 30 different `geoms`.
- ▶ Can get help with `?geom_smooth`, for example.

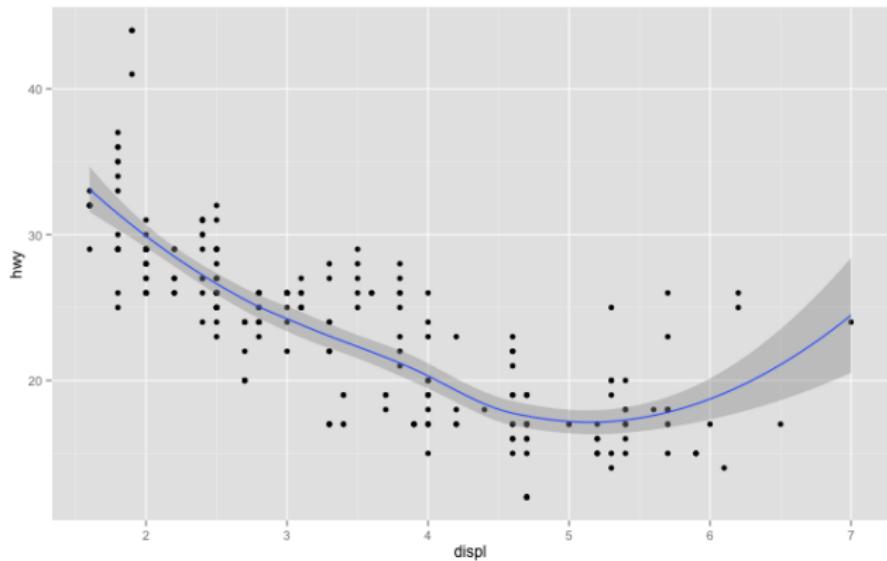
# GEOMETRIC OBJECTS

## Some Commonly-used geoms

| geom Name                   | Used to...                                  | Aesthetics                |
|-----------------------------|---|---------------------------|
| <code>geom_histogram</code> | Visualize a Continuous Variable             | <code>x</code> .          |
| <code>geom_bar</code>       | Visualize a Discrete Variable               | <code>x</code> .          |
| <code>geom_point</code>     | Visualize a Two Continuous Variables        | <code>x, y.</code>        |
| <code>geom_text</code>      | Add Labels to a Plot                        | <code>x, y, label.</code> |
| <code>geom_boxplot</code>   | Visualize Continuous and Discrete Variables | <code>x, y.</code>        |
| <code>geom_jitter</code>    | Visualize a Two Variables                   | <code>x, y.</code>        |
| <code>geom_abline</code>    | Plot a line                                 | <code>int, slope.</code>  |
| many more ...               |   |                           |

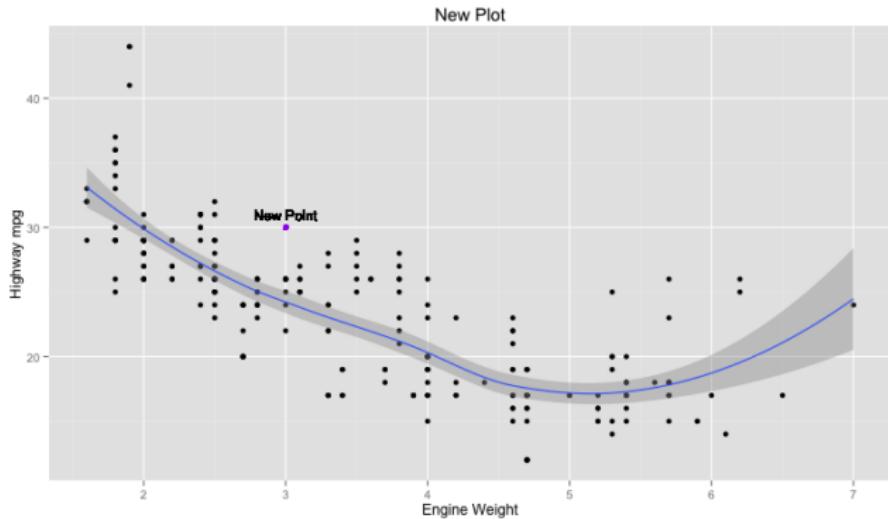
# LAYERING GEOMS

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  geom_smooth(mapping = aes(x = displ, y = hwy))
```



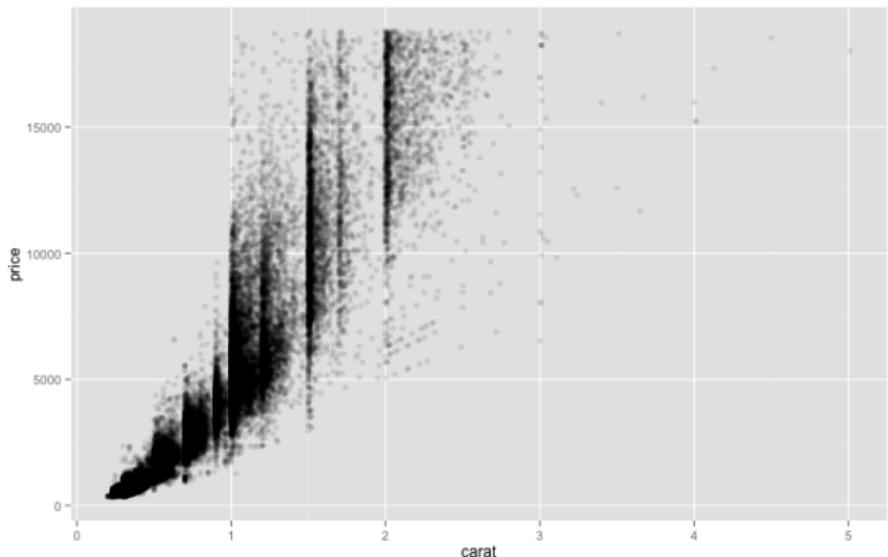
# ADDING TEXT + AXIS LABELS

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  geom_smooth(mapping = aes(x = displ, y = hwy)) +  
  geom_point(mapping = aes(x=3, y=30), color = "purple") +  
  geom_text(mapping = aes(x=3, y=31, label = "New Point"), size=4) +  
  labs(title = "New Plot", x = "Engine Weight", y = "Highway mpg")
```



## SOME MORE EXAMPLES

```
ggplot(data = diamonds) +  
  geom_point(mapping = aes(x = carat, y = price), alpha = 1/10)
```



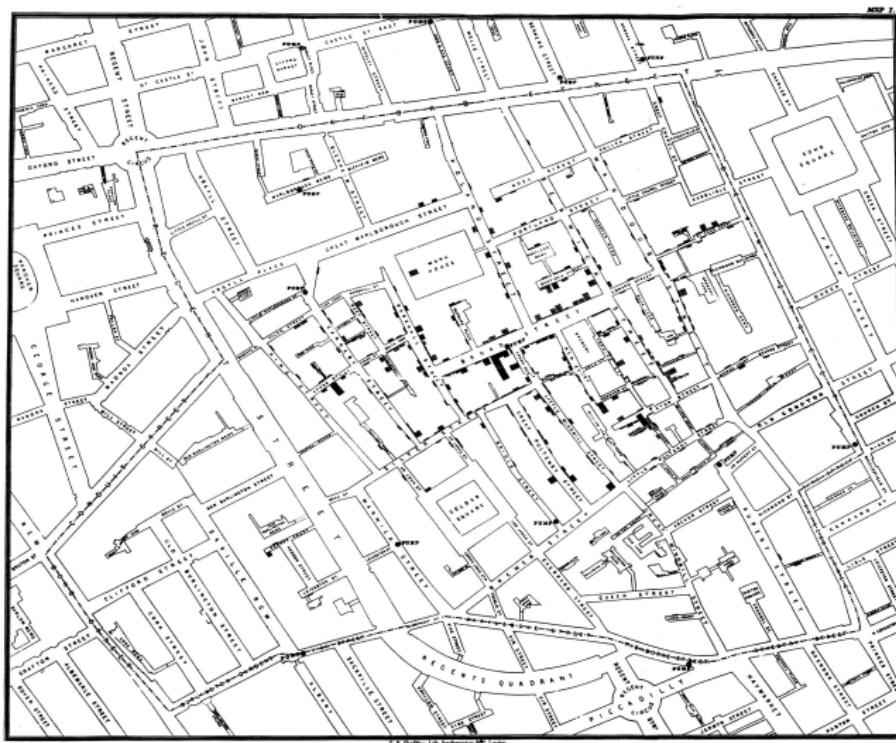
# DATA VISUALIZATION

## SECTION II

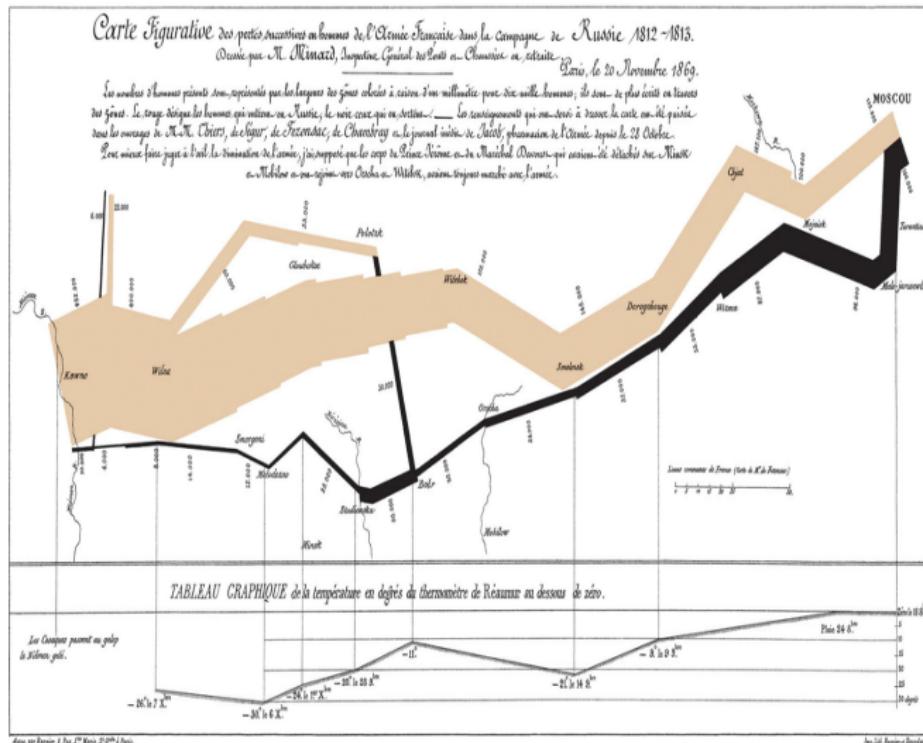
### Good Visualizations

In data science, good visualizations should give you more information than you can see in just the data table itself.

# GOOD VISUALIZATIONS - JOHN SNOW 1854 (WIKIPEDIA)



# GOOD VISUALIZATIONS - CHARLES JOSEPH MINARD 1896 (WIKIPEDIA)



# GOOD VISUALIZATIONS - CHARLES JOSEPH MINARD 1896 (WIKIPEDIA)

## Monard Graph

Minard shows six variables:

- ▶ Number of soldiers,
- ▶ Direction of the march,
- ▶ Location coordinates,
- ▶ Temperature on the return journey,
- ▶ Location on dates in November and December.

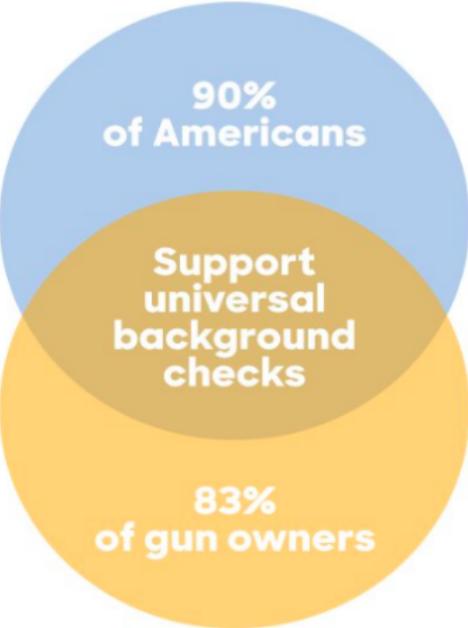
# GOOD VISUALIZATIONS - ARY SCHAFFER



# BAD VISUALIZATIONS

Hillary Clinton will be a great president, but could use some work on  
her Venn Diagram skills.

# BAD VISUALIZATIONS - HILLARY CLINTON



90%  
of Americans

Support  
universal  
background  
checks

83%  
of gun owners

 **Hillary Clinton**   
@HillaryClinton

 Follow

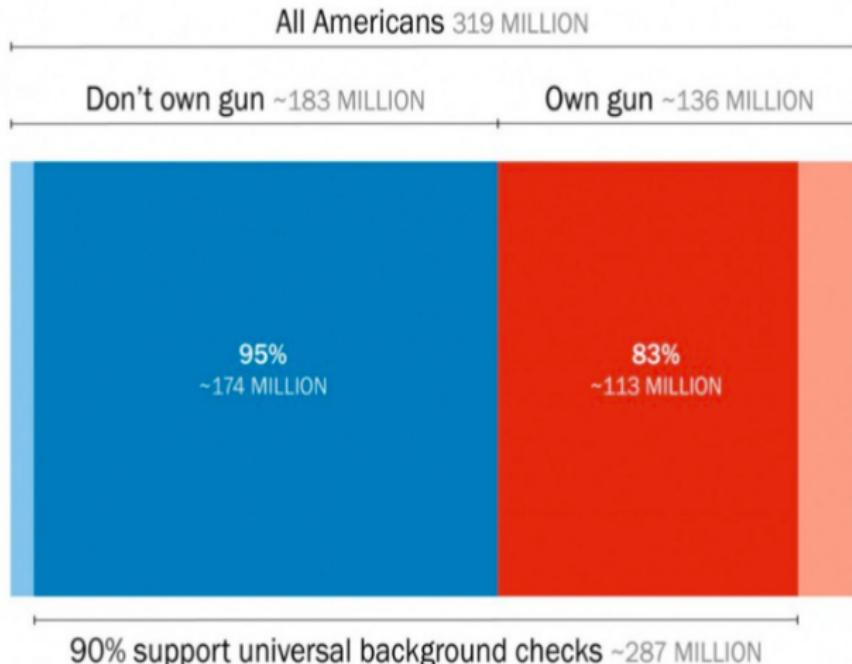
Dear Congress,

Let's get this done.

# GOOD VISUALIZATIONS - WASHINGTON POST

## Improving the Clinton campaign's terrible graph

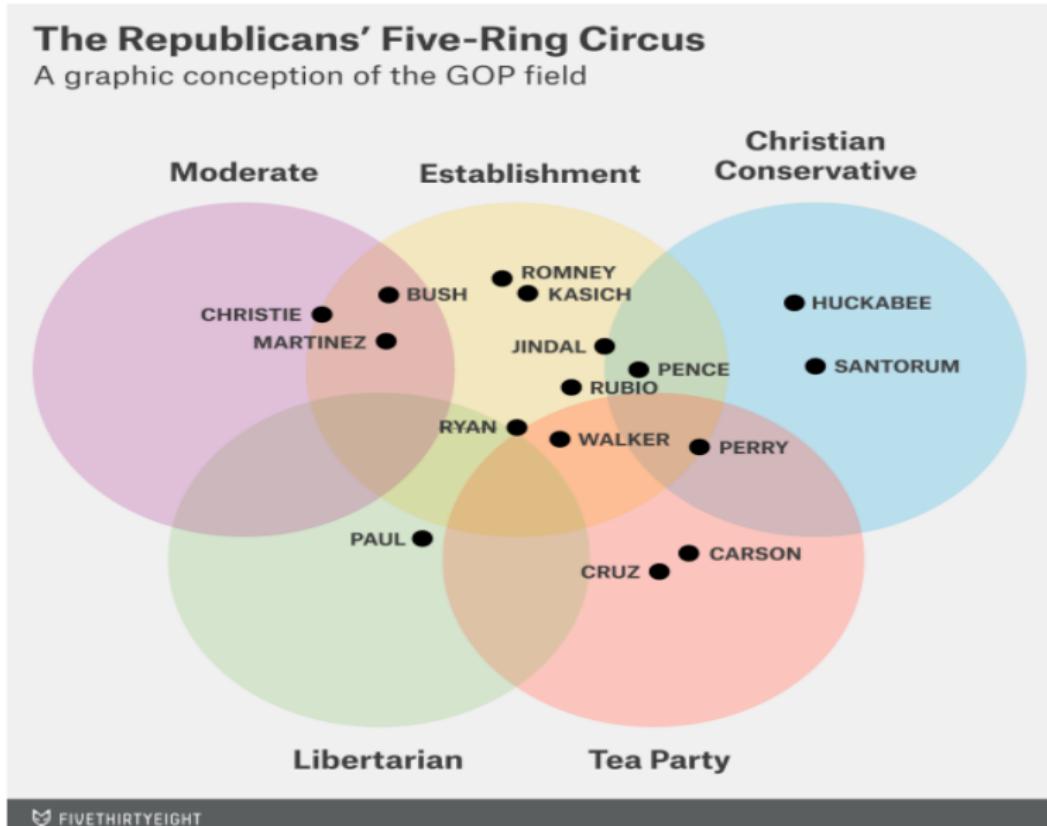
Population estimates from the Census Bureau. Gun ownership estimate based on calculations from Gallup compared with Census household data. Percentages based on Clinton campaign figures.



# BAD VISUALIZATIONS

It's OK Hillary, even statisticians are sometimes bad at making visualizations!

# BAD VISUALIZATIONS - NATE SILVER



# BAD VISUALIZATIONS - NATE SILVER

| Candidate | Moderate | Establishment | Christian Conservative | Libertarian | Tea Party |
|-----------|----------|---------------|------------------------|-------------|-----------|
| Bush      | X        | X             |                        |             |           |
| Carson    |          |               |                        |             | X         |
| Christie  | X        | X             |                        |             |           |
| Cruz      |          |               |                        |             | X         |
| Huckabee  |          |               | X                      |             |           |
| Jindal    |          | X             | X                      |             |           |
| Kasich    |          | X             |                        |             |           |
| Martinez  | X        | X             |                        |             |           |
| Paul      |          |               |                        | X           |           |
| Pence     |          | X             | X                      |             |           |
| Perry     |          | X             | X                      |             | X         |
| Romney    |          | X             |                        |             |           |
| Rubio     |          | X             |                        |             |           |
| Ryan      |          | X             |                        |             |           |
| Santorum  |          |               | X                      |             |           |
| Walker    |          | X             |                        |             | X         |

# CHECK YOURSELF – GOOD OR BAD?

## National Household Survey (NHS)

### Search NHS

By topic

By geography

### Products

Analytical products

Data products

### Reference products

### Other links

NHS corrections/updates

Accessing my NHS information

Census

Geography

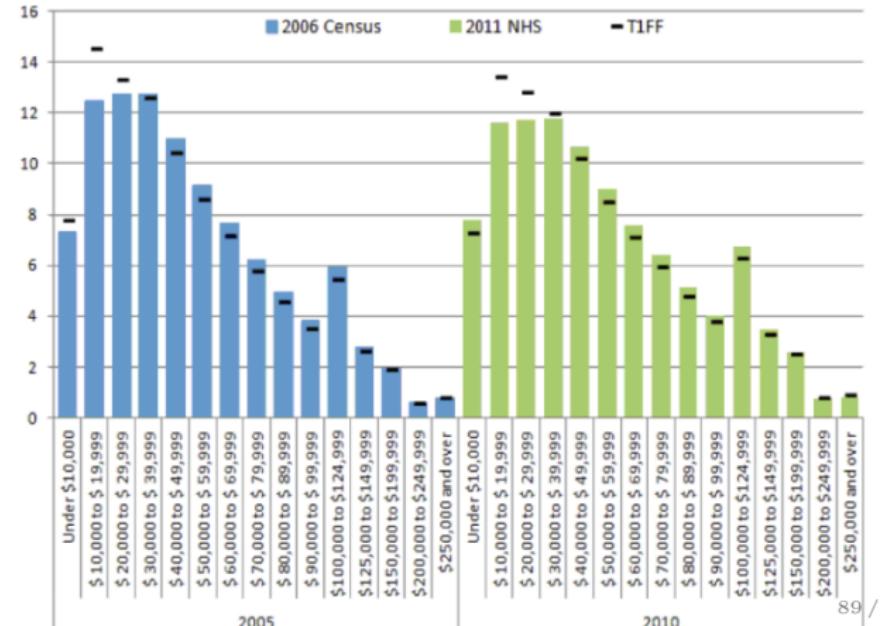
Open data

## Figure 2

### Distribution of after-tax income of census family units for Canada, 2005 and 2010

Description for figure 2

percentage



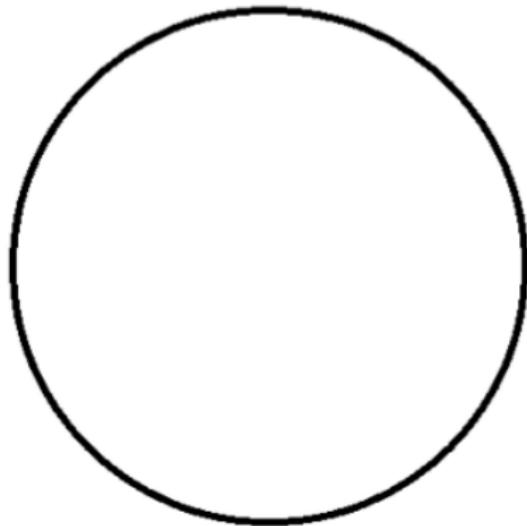
# GOOD VISUALIZATIONS

Keep things simple in terms of color and presentation!

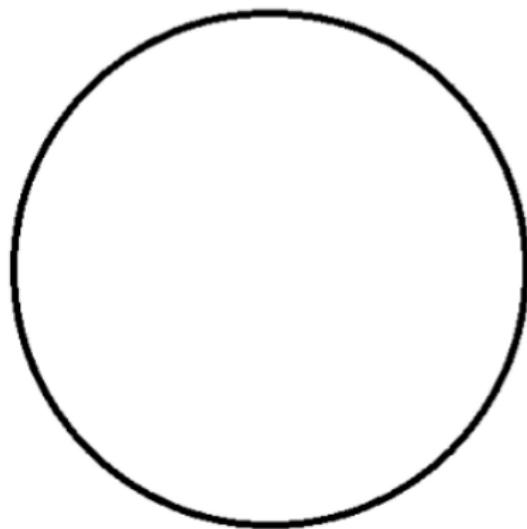
# GOOD VISUALIZATIONS

Keep things simple in terms of color and presentation!

Venn Diagrams



Things Team Clinton Does Well



# Lecture 7: Randomness

STAT GR5206

*Statistical Computing & Introduction to Data Science*

Cynthia Rush  
Columbia University

October 20, 2017

# COURSE NOTES

- ▶ Homework 3 due.
- ▶ Midterm November 3.
- ▶ No lab meetings next week.

# RANDOM NUMBER GENERATION

# RANDOM NUMBER GENERATION

## Random Numbers in R

There are many ways to generate random numbers in R. Below we generate 10 random variables distributed uniformly over the unit interval.

```
> runif(10)
```

```
[1] 0.9938169 0.7368609 0.1737553 0.6171869 0.4413878
[6] 0.7145963 0.4885156 0.3684298 0.8331707 0.4945066
```

On your machine, you'll see different random numbers.

# RANDOM NUMBER GENERATION

## Random Numbers in R

To recreate the same random numbers, use the function `set.seed()`.

```
> set.seed(10)  
> runif(10)
```

```
[1] 0.50747820 0.30676851 0.42690767 0.69310208 0.08513597  
[6] 0.22543662 0.27453052 0.27230507 0.61582931 0.42967153
```

Try it again.

```
> set.seed(10)  
> runif(10)
```

```
[1] 0.50747820 0.30676851 0.42690767 0.69310208 0.08513597  
[6] 0.22543662 0.27453052 0.27230507 0.61582931 0.42967153
```

In this part of the lecture, we want to understand how R generates random numbers.

# RANDOM NUMBERS IN R

Many machine learning and statistical techniques require randomness.  
To name a few:

- ▶ Markov Chain Monte Carlo
- ▶ Bagging, Boosting, Random Forests
- ▶ The bootstrap
- ▶ Cross-validation

# RANDOM NUMBER GENERATION

## Today's Lecture

- ▶ How does R produce random numbers?
- ▶ It doesn't! Generation of random numbers on a computer is complicated because computer programs are inherently deterministic.
- ▶ Split the problem in two:
  - ▶ Generating any randomness at all – concentrate on simple case of uniform random values in  $[0, 1]$ .
  - ▶ Generating random numbers from different distributions, using the previous step as the basis.

# RANDOM NUMBER GENERATION

## Generating Uniform Random Variables on $[0, 1]$ in R

- ▶ R uses tricks that generate **pseudo random numbers** that are indistinguishable from real random numbers using a computer program.
- ▶ Usually fast and resource effective, but not ‘truly random’. **Pseudo random generators** produce a deterministic sequence that is indistinguishable from a true random sequence if you don’t know how it started.

# RANDOM NUMBER GENERATION

## In Contrast, True Randomness

Uses real-world noise (i.e. some physical phenomenon that is random):

- ▶ Tossing a coin or throwing dice.
- ▶ Thermal detection uses trailing decimal points on a thermometer.
- ▶ Can use cosmic ray/radioactive decay arrival timings.
- ▶ [Random.org](http://Random.org) offers random numbers produced using atmospheric noise.

In general, requires specialized hardware that is expensive and slow.

# LINEAR CONGRUENTIAL GENERATOR (LCG)

- ▶ A **Linear Congruential Generator (LCG)** is a simple example of a pseudo-random number generator.
- ▶ While it's no longer practically important, it shares important characteristics with more complicated generators used today.

The LCG is an algorithm that produces a sequence of pseudo random numbers based on the recurrence relation formula:

$$X_n = (aX_{n-1} + c) \mod m$$

In the above,

- ▶  $m > 1$  is the modulus
- ▶  $a \in \{1, 2, \dots, m - 1\}$  is the multiplier
- ▶  $c \in \{0, 1, 2, \dots, m - 1\}$  is the increment
- ▶  $X_0 \in \{0, 1, 2, \dots, m - 1\}$  is the seed

# LINEAR CONGRUENTIAL GENERATOR (LCG)

$$X_n = (aX_{n-1} + c) \pmod{m}$$

- ▶  $m > 1$  is the modulus
- ▶  $a \in \{1, 2, \dots, m - 1\}$  is the multiplier
- ▶  $c \in \{0, 1, 2, \dots, m - 1\}$  is the increment
- ▶  $X_0 \in \{0, 1, 2, \dots, m - 1\}$  is the seed

## Simulating from $[0, 1]$

- ▶ Recurrence:  $X_1$  is produced from  $X_0$  and then used to generate  $X_2$ . Then  $X_2$  is used to generate  $X_3$ , and so on.
- ▶  $0 \leq X_i \leq m - 1$  for all  $i = 0, 1, 2, \dots$  and the sequence repeats every  $m$  occurrences.
- ▶ Divide by  $m$  to give values between 0 and 1 (but never quite hitting 1).
- ▶ For well-chosen  $m, a$ , and  $c$  the resulting sequence behaves ‘similarly’ to a sequence of independent, uniformly distributed random variables.

# LINEAR CONGRUENTIAL GENERATOR (LCG)

Type the following into your console:

```
new.random <- function(a = 5, c = 12, m = 16, seed, len) {  
  seq      <- rep(NA, len)  
  seq[1]  <- seed  
  for (i in 2:len) {  
    seq[i] <- (a*seq[i-1] + c) %% m  
  }  
  return(seq)  
}  
  
new.random(seed = 10, len = 20)
```

# LINEAR CONGRUENTIAL GENERATOR (LCG)

Type the following into your console:

```
new.random(a = 131, c = 7, m = 16, seed = 10, len = 20)
new.random(a = 129, c = 7, m = 16, seed = 10, len = 20)
new.random(a=1664545, c=1013904223, m=2^32, seed = 10, len = 20)
```

- ▶ Type `?Random` to get more info on random number generators used in R.
- ▶ The period on the final LCG can be shown to be  $2^{32}$  – for most purposes this gives us a random sequence.

# QUALITY OF RANDOM NUMBER GENERATOR

PRNGs used in practice are more sophisticated than the LCG but share many characteristics.

- ▶ All PRNGs are periodic – good ones have periods longer than the amount of random numbers ever needed in a reasonable amount of time.
- ▶ Almost all PRNGs output approximately uniformly distributed random numbers. The result could demonstrate non-uniformity if (1) some values appear more often than others or if (2) the sequence is too regular.
- ▶ All PRNGs are deterministic, so dependence between samples is a concern.

In general, we will never implement our own PRNG – we will use R's.

# SIMULATING RANDOM VARIABLES FROM COMMON PROBABILITY DISTRIBUTIONS

# SIMULATING FROM PROBABILITY DISTRIBUTIONS

How do we simulate from a probability distribution?

Two Situations:

- ▶ First: **Common Distributions.** Use built-in R functions (normal, gamma, Poisson, binomial, etc..).
- ▶ Second: **Uncommon Distributions.** Will transform i.i.d. sequences of uniform  $[0, 1]$  -distributed random variables. (Note that for the common distributions, R just does this for us.)

# SIMULATING FROM PROBABILITY DISTRIBUTIONS

For common distributions, R has many built-in PRNGs for simulating and working with random variables. These functions allow us to, for example,

- ▶ Plot density functions,
- ▶ Compute probabilities,
- ▶ Compute quantiles,
- ▶ Simulate random draws from the distribution.

# R COMMANDS FOR DISTRIBUTIONS

## R Commands

For each distribution, there are typically four distributions available with “r”, “d”, “p”, and “q” prefixes.

- ▶ `rfoo` draws random numbers from distribution `foo`.
- ▶ `dfoo` is the probability density function (pdf) or probability mass function (pmf) of `foo`.
- ▶ `pfoo` is the cumulative probability function (cdf) of `foo`.
- ▶ `qfoo` is the quantile function (inverse cdf) of `foo`.

Example `foo`'s: `norm`, `pois`, `exp`, `beta`, `binom`, ...

`?distribution` for the options.

# R COMMANDS FOR DISTRIBUTIONS

## R Commands

```
rnorm(n, mean = 0, sd = 1)
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
```

log表示是否对输入的x , p或者输出的p取log

## Example: Gaussians

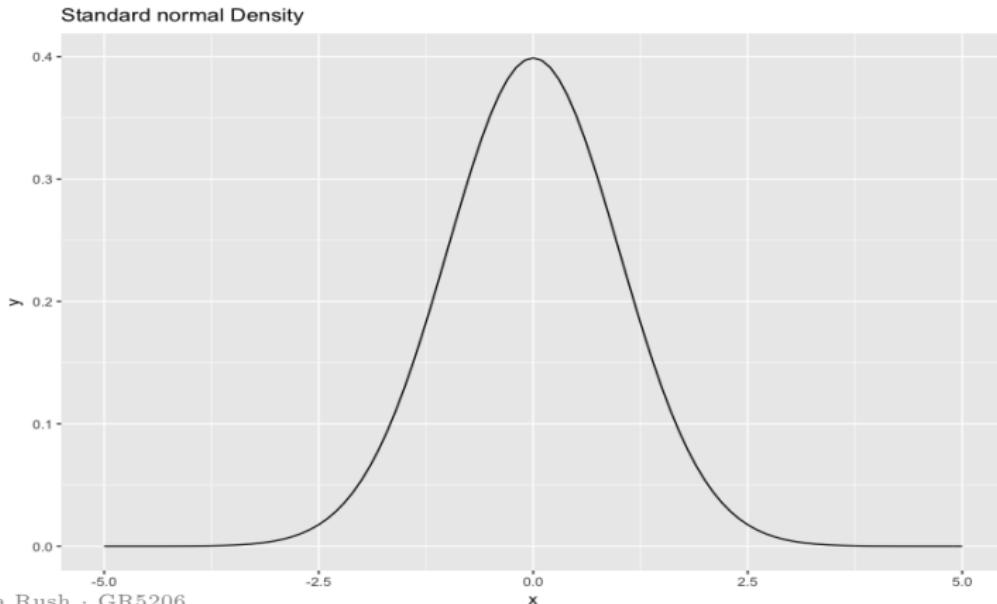
```
dnorm(0, mean = 0, sd = 1)
1/sqrt(2*pi)

rnorm(1)
rnorm(5)
x <- rnorm(10, mean = 100, sd = 1)
summary (x)
```

# R COMMANDS FOR DISTRIBUTIONS

## Example: Gaussians

```
ggplot(data.frame(x = c(-5, 5))) +  
  stat_function(mapping = aes(x = x), fun = dnorm) +  
  labs(title = "Standard Normal Density")
```



# R COMMANDS FOR DISTRIBUTIONS

## Example: Gaussians

```
pnorm(0)                      # P(Z < 0)
pnorm(1.96) - pnorm(-1.96) # P(-1.96 < Z < 1.96)

qnorm(.5)                      # P(Z < ?) = 0.5
qnorm(.975)                     # P(Z < ?) = 0.975
```

# EXAMPLE

## Tasks

Let  $X$  be a binomial random variable with  $n$  trials and success probability  $p$ . For large  $n$ , recall the normal approximation to the binomial distribution:

$$P(X \leq x) \approx \Phi\left(\frac{x + .5 - np}{\sqrt{np(1 - p)}}\right),$$

where  $\Phi(z)$  is the cdf of the standard normal distribution.

- ▶ Let  $X$  be binomial with  $n = 1000, p = 0.20$ . Using the normal approximation to the binomial distribution, compute the approximate probability  $P(X \leq 190)$ .
- ▶ Calculate the exact probability  $P(X \leq 190)$ .
- ▶ Let  $X \sim \text{Binom}(n = 1000, p = 0.20)$ . Simulate 500 realizations of  $X$  and create a histogram (or bargraph) of the values.

## EXAMPLE

The approximation is given by

$$P(X \leq 190) \approx \Phi\left(\frac{190 + .5 - (1000)(0.20)}{\sqrt{(1000)(0.20)(0.80)}}\right),$$

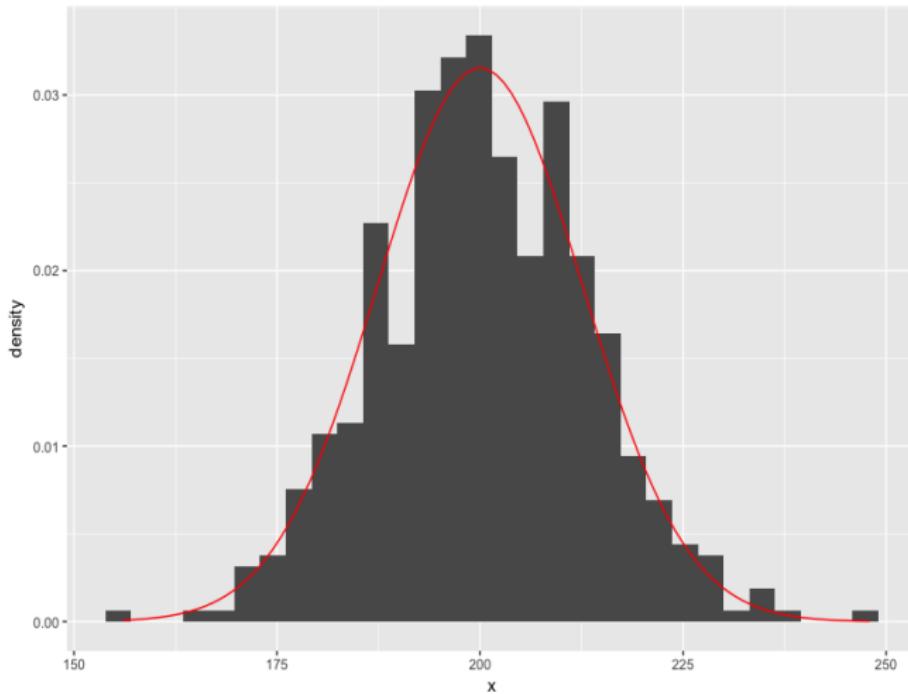
```
val <- 190; n    <- 1000; p    <- 0.20
correction <- (val + 0.5 - n*p)/(sqrt(n*p*(1-p)))
pnorm(correction)           # P(Z < correction)

pbinom(val, size = n, prob = p)           # P(X <= 190)
sum(dbinom(0:val, size = n, prob = p))

x <- rbinom(500, size = n, prob = p)
ggplot(data = data.frame(x)) +
  geom_histogram(aes(x = x, y = ..density..)) +
  stat_function(mapping = aes(x = x), fun = dnorm,
    args = list(mean = 200, sd = sqrt(160)), color = "red") +
  labs(title = "Normal Approximation to the Binomial")
```

# EXAMPLE

Normal Approximation to the Binomial



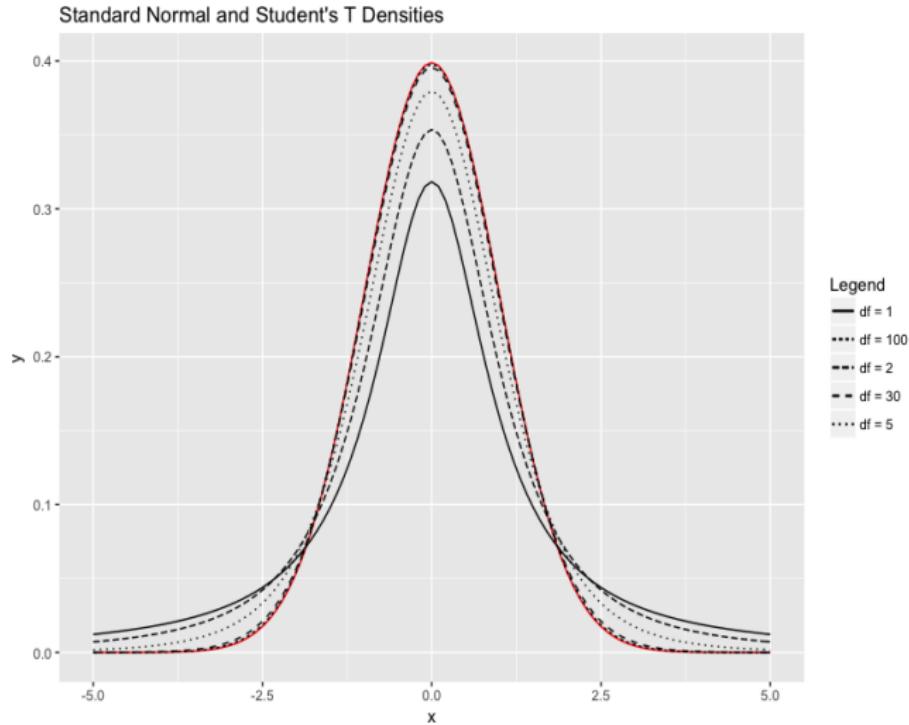
## EXAMPLE

- ▶ Plot the density function of the student's t distribution with degrees of freedom,  $df = 1, 2, 5, 30, 100$ . Hint: Use something like `stat_function(mapping = aes(x = x), fun = dt, args = list(df = 1))`.
- ▶ Plot the standard normal density on the same figure. Plot this curve in red.

## EXAMPLE

```
ggplot(data.frame(x = c(-5, 5))) +  
  stat_function(aes(x = x), fun = dnorm, color = "red") +  
  stat_function(aes(x = x, linetype = "df = 1"),  
                fun = dt, args = list(df = 1)) +  
  stat_function(aes(x = x, linetype = "df = 2"),  
                fun = dt, args = list(df = 2)) +  
  stat_function(aes(x = x, linetype = "df = 5"),  
                fun = dt, args = list(df = 5)) +  
  stat_function(aes(x = x, linetype = "df = 30"),  
                fun = dt, args = list(df = 30)) +  
  stat_function(aes(x = x, linetype = "df = 100"),  
                fun = dt, args = list(df = 100)) +  
  labs(title = "Standard Normal and Student's T Densities", line
```

# STUDENT'S T



# CHECK YOURSELF

```
pgamma(2, shape = 2, rate = 1) # P(0 < X < 2)  
1 - pgamma(2, shape = 2, rate = 1) # P(X > 2)
```

## Tasks

Recall that the gamma density function is:

$$f(x|\alpha, \beta) = \frac{x^{\alpha-1} e^{-x/\beta}}{\Gamma(\alpha)\beta^\alpha}, \quad 0 < x < \infty, \quad \alpha > 0, \quad \beta > 0,$$

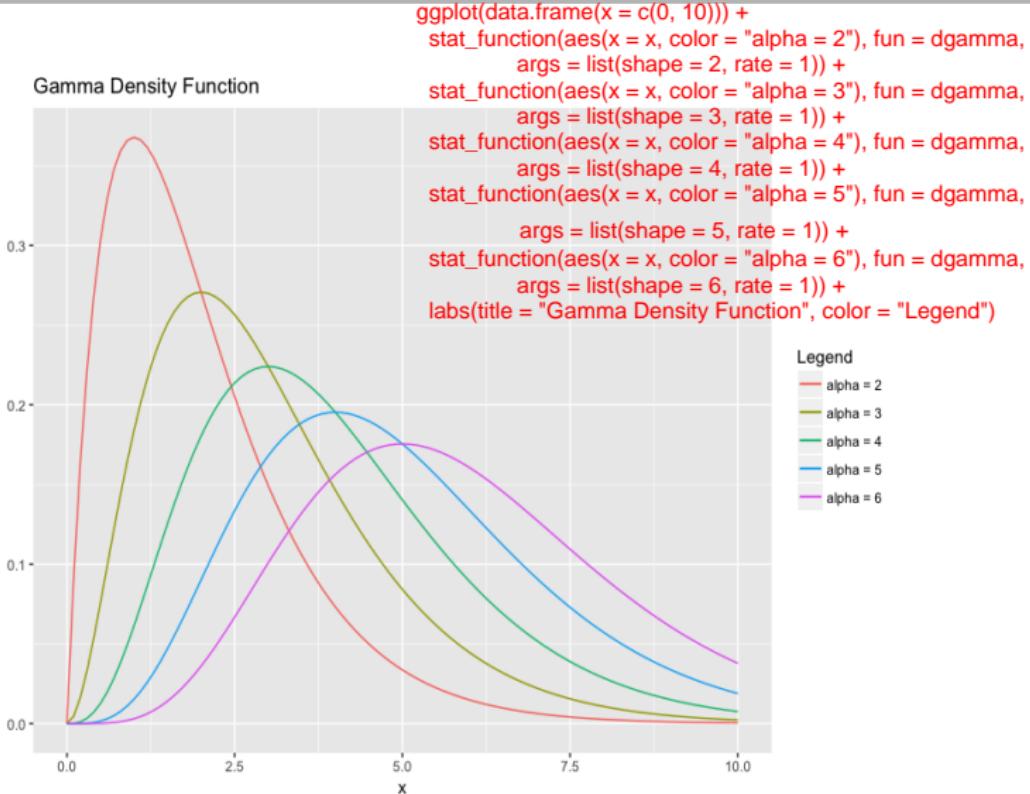
where  $\alpha$  is the shape parameter and  $\beta$  is the scale parameter. Use R to do the following:

- ▶ For  $\alpha = 2$  and  $\beta = 1$  compute

$$\int_2^{\infty} f(x|\alpha, \beta) dx$$

- ▶ Plot the gamma density using shape parameters  $\alpha = 2, 3, 4, 5, 6$ .

# CHECK YOURSELF



# CHECK YOURSELF

## Tasks

Suppose we want to simulate from the following linear model:

$$y = \beta_0 + \beta_1 x + \epsilon$$

where  $\epsilon \sim \mathcal{N}(0, 2^2)$ ,  $x \sim \mathcal{N}(0, 1)$ ,  $\beta_0 = 0.5$ , and  $\beta_1 = 2$ . Simulate 100 realizations from this model and then make a scatterplot of the  $(x, y)$  pairs.

```
x <- rnorm(100)
e <- rnorm(100, sd = 2)
y <- 0.5 + 2*x + e
ggplot(data.frame(x = x, y = y)) +
  geom_point(aes(x = x, y = y))
```

## sample() FUNCTION

The `sample()` function draws randomly from a specified set of objects allowing you to sample from arbitrary distributions of numbers.

```
sample(1:10, 4)
sample(1:10, 4)
sample(letters, 5)
```

```
sample(1:10)
sample(1:10)
sample(1:10, replace = TRUE)
```

# SAMPLE() FUNCTION

## Example

We'd like to generate rvs from the following discrete distribution:

|        |     |     |     |
|--------|-----|-----|-----|
| $x$    | 1   | 2   | 3   |
| $f(x)$ | 0.1 | 0.2 | 0.7 |

```
n <- 1000
p <- c(0.1, 0.2, 0.7)
x <- sample(1:3, size = n, prob = p, replace = TRUE)
head(x, 10)
rbind(p, p.hat = table(x)/n)
```

# CHECK YOURSELF

## Tasks

- ▶ Use `sample()` to simulate 100 fair die rolls.
- ▶ Use `runif()` to simulate 100 fair die rolls. You may also want to use something like `round()`.

# SIMULATING RANDOM VARIABLES FROM UNCOMMON PROBABILITY DISTRIBUTIONS

# SIMULATING FROM UNCOMMON PROBABILITY DISTRIBUTIONS

We study methods for transforming i.i.d. uniform  $[0, 1]$ –distributed random variables into a prescribed target distribution.

- ▶ Useful when we want to sample from a distribution that's not one with built-in R functions.
- ▶ Helpful in understanding how R's simulation functions build off of PRNGs.

Discuss different methods, applicable to different classes of target distributions.

# A SIMPLE EXAMPLE

## Target Distribution

Want  $X$  uniformly distributed on the set  $\{0, 1, \dots, n - 1\}$ , i.e.

$$P(X = k) = \frac{1}{n} \text{ for } k = 0, 1, \dots, n - 1.$$

- ▶ Maybe use a PRNG with a state space  $0, 1, \dots, n - 1$ ? Not a good idea since we want a long period.
- ▶ Instead: first generate a rv  $U \sim \text{uniform } [0, 1]$ , then transform with  $X = \lfloor nU \rfloor$ .
- ▶ Can show rigorously that  $X$  has the desired distribution.

```
n <- 10
samp <- 100
table(floor(n*runif(samp)))/samp
samp <- 10000
table(floor(n*runif(samp)))/samp
```

# ANOTHER SIMPLE EXAMPLE

## Target Distribution

Want  $X$  to take values in  $\{1, 2, 3\}$  with probabilities  $(1/2, 1/4, 1/4)$ .

- ▶ Use the fact that for a rv  $U \sim \text{uniform } [0, 1]$  and an event  $E = \{U \leq p\}$  then  $P(E) = p$ .
- ▶ Define  $X$  as follows:

$$X = \begin{cases} 1 & \text{if } 0 \leq U < 1/2 \\ \textcolor{blue}{\cancel{1}} \textcolor{red}{2} & \text{if } 1/2 \leq U < 3/4 \\ \textcolor{blue}{\cancel{1}} \textcolor{red}{3} & \text{if } 3/4 \leq U \leq 1 \end{cases}$$

# INVERSE TRANSFORM METHOD

- ▶ The Inverse Transform Method is a general method that can be used when the target distribution is one-dimensional.
- ▶ Uses the inverse cdf denoted  $F^{-1}$  of the target distribution:

$$F^{-1}(u) = \inf\{x \in \mathbb{R} | F(x) \geq u\}$$

for all  $u \in (0, 1)$ .

- ▶ If  $F$  is bijective (i.e.  $F$  is strictly monotonically increasing with no jumps), then  $F^{-1}$  is the usual inverse and can be found by solving  $F(x) = u$  for  $x$ .

# INVERSE TRANSFORM METHOD

## Method

Whenever  $F^{-1}$  can be determined, then  $X = F^{-1}(U)$  is such that  $X \sim F$ .

Why does this work?

$$\begin{aligned} P(X \leq x) &= P(F^{-1}(U) \leq x) \\ &\stackrel{(a)}{=} P(F(F^{-1}(U)) \leq F(x)) \\ &= P(U \leq F(x)) \\ &= F(x), \end{aligned}$$

where (a) follows by monotonicity of  $F$ .

# INVERSE TRANSFORM METHOD

## Algorithm

1. Derive the inverse function  $F^{-1}$ . To do this:
  - ▶ Solve  $F(x) = u$  for  $x$  to find  $x = F^{-1}(u)$ .
2. Write a function to compute  $x = F^{-1}(u)$ .
3. For each realization:
  - ▶ Generate a random value  $u$  from Uniform(0,1).
  - ▶ Compute  $x = F^{-1}(u)$  as a realization from the target distribution.

# INVERSE TRANSFORM METHOD

Example: Target distribution is exponential with  $\lambda = 2$ .

The pdf of the exponential distribution is  $f(x) = \lambda e^{-\lambda t}$ , so the cdf is

$$F(x) = \int_0^x f(t) dt = \int_0^x \lambda e^{-\lambda t} dt = 1 - e^{-\lambda x}.$$

Now we invert the cdf.

$$u = 1 - e^{-\lambda x} \quad \rightarrow \quad x = -\frac{1}{\lambda} \log(1 - u)$$

# INVERSE TRANSFORM METHOD

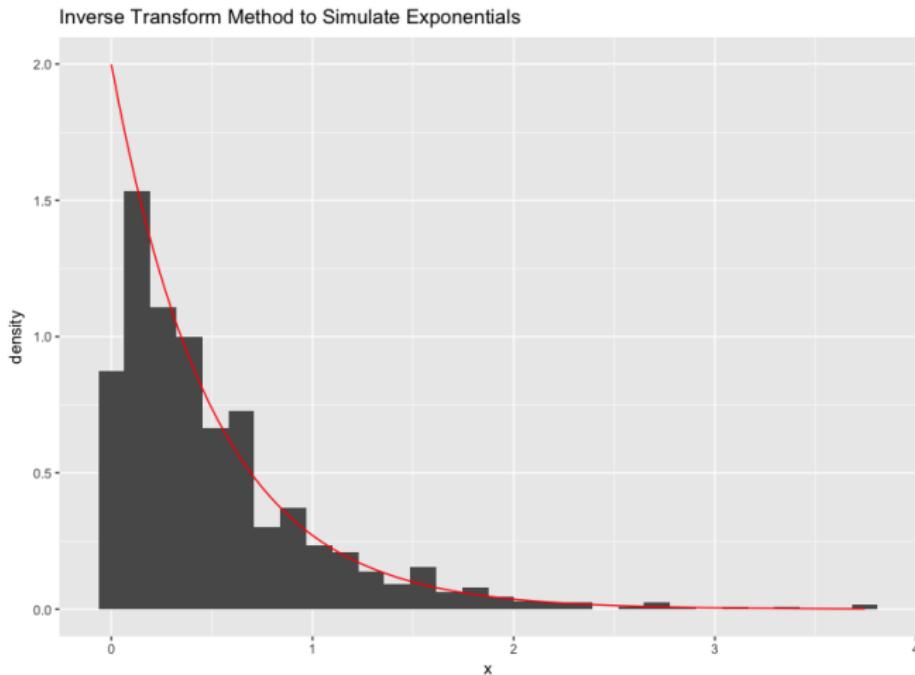
Example: Target distribution is exponential with  $\lambda = 2$ .

```
lambda <- 2
n      <- 1000
u      <- runif(n) # Simulating uniform rvs

Finverse <- function(u, lambda) {
  # Function for the inverse transform
  stopifnot(u > 0 & u < 1)
  return(-(1/lambda)*log(1-u))
}
x <- Finverse(u, lambda)

ggplot(data = data.frame(x)) +
  geom_histogram(aes(x = x, y = ..density..)) +
  stat_function(mapping = aes(x = x), fun = dexp,
                args = list(rate = 2), color = "red") +
  labs(title = "Inverse Transform Method to Simulate Exponential
```

# INVERSE TRANSFORM METHOD: EXPONENTIAL



# CHECK YOURSELF

## Task

Simulate a random sample of size 1000 from the pdf  $f_X(x) = 3x^2$ ,  $0 \leq x \leq 1$ .

- ▶ Find  $F$  and then  $F^{-1}$ .
- ▶ Plot the empirical distribution (histogram) with the correct density also shown on the plot.

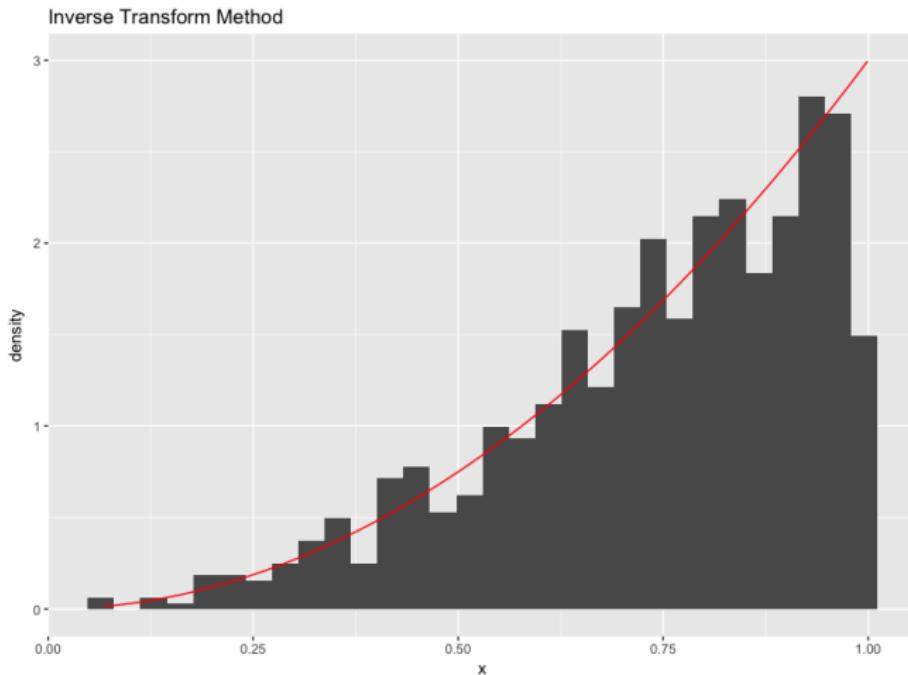
```
n      <- 1000
u      <- runif(n) # Simulate uniforms

F.inverse <- function(u) {return(u^(1/3))}
x      <- F.inverse(u)

# x <- u^(1/3)

ggplot(data.frame(x)) +
  geom_histogram(aes(x = x, y = ..density..)) +
  stat_function(aes(x = x), fun = function(x) {3*x^2}, color = "red") +
  labs(title = "Inverse Transform Method")
```

# INVERSE TRANSFORM METHOD



# ACCEPTANCE-REJECTION ALGORITHM

The inverse transform method can be applied when  $F^{-1}$  is easy to evaluate. Sometimes this isn't the case (e.g. the normal distribution).

- ▶ **Rejection sampling** is a more advanced and very popular method for random number generation.
- ▶ How does it work? By sampling candidates from an easier to generate distribution then correcting the sampling probability by randomly rejecting some candidates.

## Pros and Cons

- ▶ Not restricted to Uniform[0,1] input samples. Idea: generate samples of approximately the correct distribution, then reject some to hit target distribution exactly.
- ▶ Can be generalized to work on very general spaces beyond  $\mathbb{R}^d$ .
- ▶ Requires a random and potentially large number of input samples to generate one output, so efficiency can be a concern.

# THE REJECTION METHOD

## Basic Algorithm

- ▶ Input:
  - ▶ A probability density  $g$  (the **proposal** density)
  - ▶ A function  $p$  with values in  $[0,1]$  (the **acceptance probability**)
- ▶ Random Values:
  - ▶  $X_n$  i.i.d. with density  $g$  (the **proposals**)
  - ▶  $U_n$  i.i.d. Uniform[0,1]
- ▶ Output: A sequence of i.i.d. random variables with density

$$f(x) = \frac{1}{Z} p(x)g(x) \text{ where } Z = \int p(x)g(x)dx.$$

- ▶ Algorithm: For  $n = 1, 2, 3, \dots$ 
  - ▶ Generate  $X_n \sim g$  and  $U_n \sim \text{Uniform}[0,1]$
  - ▶ if  $U_n \leq p(X_n)$  then output  $X_n$ .

# THE REJECTION METHOD

## Notes

- ▶  $U_n$  values randomly decide whether to output or ignore  $X_n$ .
- ▶ Value  $X_n$  output with probability  $p(X_n)$ .
- ▶ If the  $X_n$  value is chosen, say it is **accepted**. Otherwise it is **rejected**.
- ▶ Idea: we can sample from  $g$  easily and then use this to sample from  $f$ .

## Theory

- ▶ The elements of the output are i.i.d. with density  $f$ :

$$f(x) = \frac{1}{Z} p(x)g(x) \text{ where } Z = \int p(x)g(x)dx.$$

- ▶ Each proposal is accepted with probability  $Z$  so the number of proposals required to generate each output is geometrically distributed with mean  $Z$ .

# THE REJECTION METHOD

## Example

- ▶ Let the proposal density  $g$  be Uniform[-1, +1] and the acceptance probability  $p(x) = \sqrt{1 - x^2}$ .
- ▶ Accepted samples have density

$$f(x) = \frac{1}{Z} p(x)g(x) = \frac{4}{\pi} \sqrt{1 - x^2} \times \frac{1}{2} \mathbb{I}\{-1 \leq x \leq +1\}.$$

- ▶ This is the ‘semicircle distribution’.

# THE REJECTION METHOD

```
p <- function(x) {sqrt(1 - x^2)}
```

```
generate_one <- function(p_func) {
  val <- NULL
  while (is.null(val)) {
    x <- runif(1, min = -1, max = +1)
    u <- runif(1)
    if (u <= p_func(x)) {val <- x}
  }
  return(val)
}
```

```
num <- 1000
samp <- rep(NA, num)
for (i in 1:num) {samp[i] <- generate_one(p)}
hist(samp)
```

# THE ENVELOPE REJECTION METHOD

- ▶ Usually run basic algorithm by choosing the acceptance probabilities  $p$  such that the output density

$$f(x) = \frac{1}{Z} p(x) g(x)$$

coincides with the target density.

- ▶ We can write the algorithm more directly with this in mind.

# THE ENVELOPE REJECTION METHOD

## Algorithm

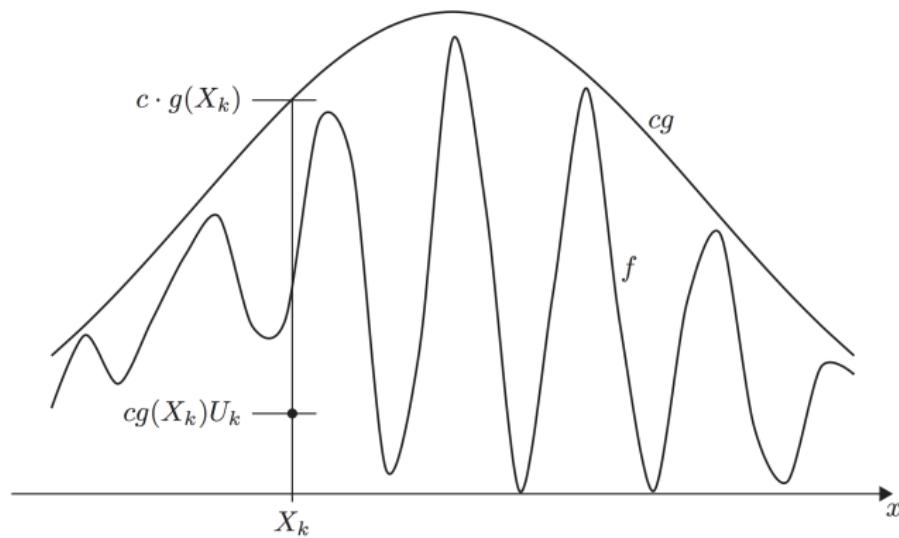
- ▶ Input:
  - ▶ A probability density  $f$  (the **target density**)
  - ▶ A probability density  $g$  (the **proposal** density)
  - ▶ A constant  $c > 0$  such that  $f(x) \leq cg(x)$  for all  $x$
- ▶ Random Values:
  - ▶  $X_n$  i.i.d. with density  $g$  (the **proposals**)
  - ▶  $U_n$  i.i.d. Uniform[0,1]
- ▶ Output: A sequence of i.i.d. random variables with density  $f$
- ▶ Algorithm: For  $n = 1, 2, 3, \dots$ 
  - ▶ Generate  $X_n \sim g$  and  $U_n \sim \text{Uniform}[0,1]$
  - ▶ if  $cg(X_n) U_n \leq f(X_n)$  then output  $X_n$ .

# THE ENVELOPE REJECTION METHOD

## Notes

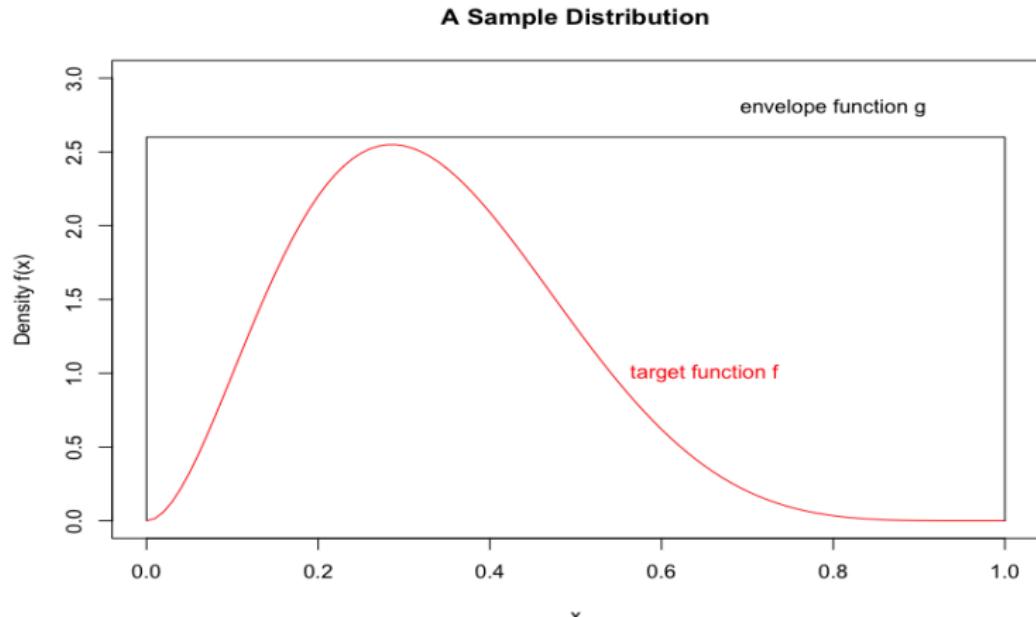
- ▶ This is the basic algorithm with acceptance probability  $p(x) = \frac{f(x)}{cg(x)}$  if  $g(x) > 0$  and  $p(x) = 1$  otherwise.
- ▶ The elements of the output are i.i.d. with density  $f$ .
- ▶ Each proposal is accepted with probability  $1/c$  so the number of proposals required to generate each output is geometrically distributed with mean  $1/c$ .
- ▶ Function  $cg$  sometimes called the ‘envelope’ for target density  $f$ .
- ▶ Actually could just use input: a function  $f$  (the non-normalized **target density**). Don’t need to be able to calculate  $Z = \int f(x)dx$  for the algorithm to work!

# THE ENVELOPE REJECTION METHOD



# AN EASY EXAMPLE

Suppose the pdf  $f$  is zero outside an interval  $[c, d]$  (in the image  $[0,1]$ ), and  $\leq M$  on the interval (in the image  $M \approx 2.5$ ). Can then choose proposal density  $g$  to be uniform.



# GEOMETRIC IDEA

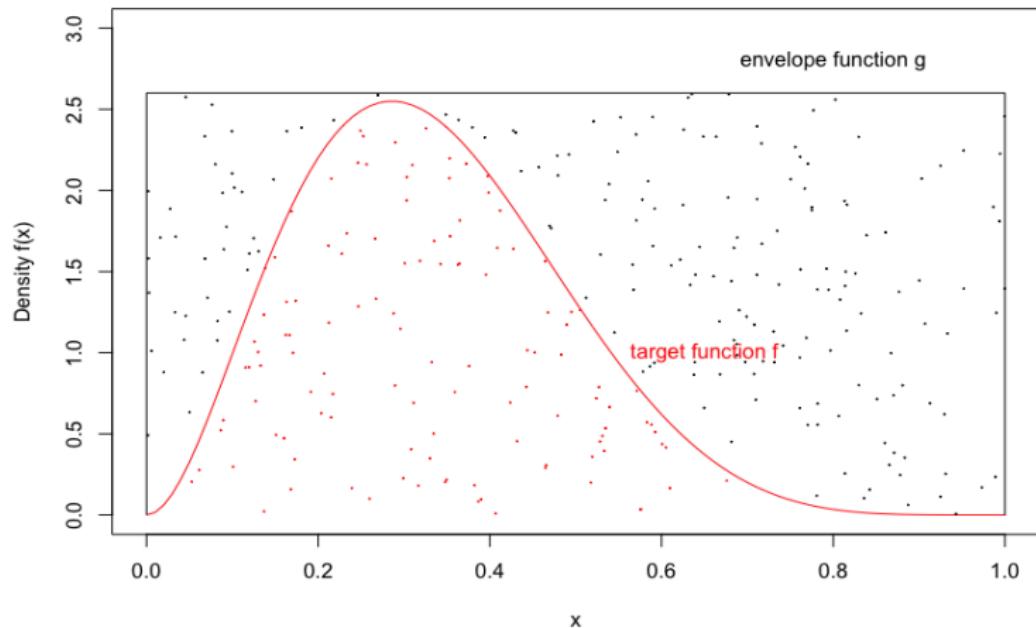
```
plot(c(0,1), c(0,3), ty = "n", main = "A Sample Distribution",
      ylab = "Density f(x)", xlab = "x")
curve (dbeta(x, 3, 6), add = TRUE, col = "red")
text(.65, 1, "target function f", col = "red")
text(.8, 2.8, "envelope function g")
lines(c(0,0,1,1), c(0,2.6,2.6,0))

x1      <- runif(300, min = 0, max = 1);
y1      <- runif(300, min = 0, max = 2.6)
selected <- y1 < dbeta(x1, 3, 6)

points (x1, y1, col = 1+selected, cex = 0.1)
```

# GEOMETRIC IDEA

A Sample Distribution



# GEOMETRIC IDEA

```
mean(selected) # Proportion selected
accepted.points <- x1[selected]

# Proportion of sample points less than 0.5.
mean(accepted.points < 0.5)

# The true distribution.
pbeta(0.5, 3, 6)
```

# GEOMETRIC IDEA

For this to work efficiently, we have to cover the target distribution with one that sits close to it.

```
plot(c(0,1), c(0,10), ty = "n", main = "A Sample Distribution",
      ylab = "Density f(x)", xlab = "x")
curve (dbeta(x, 3, 6), add = TRUE, col = "red")
text(.65, 1, "target function f", col = "red")
text(.8, 9.7, "envelope function g")
lines(c(0,0,1,1), c(0,10,10,0))

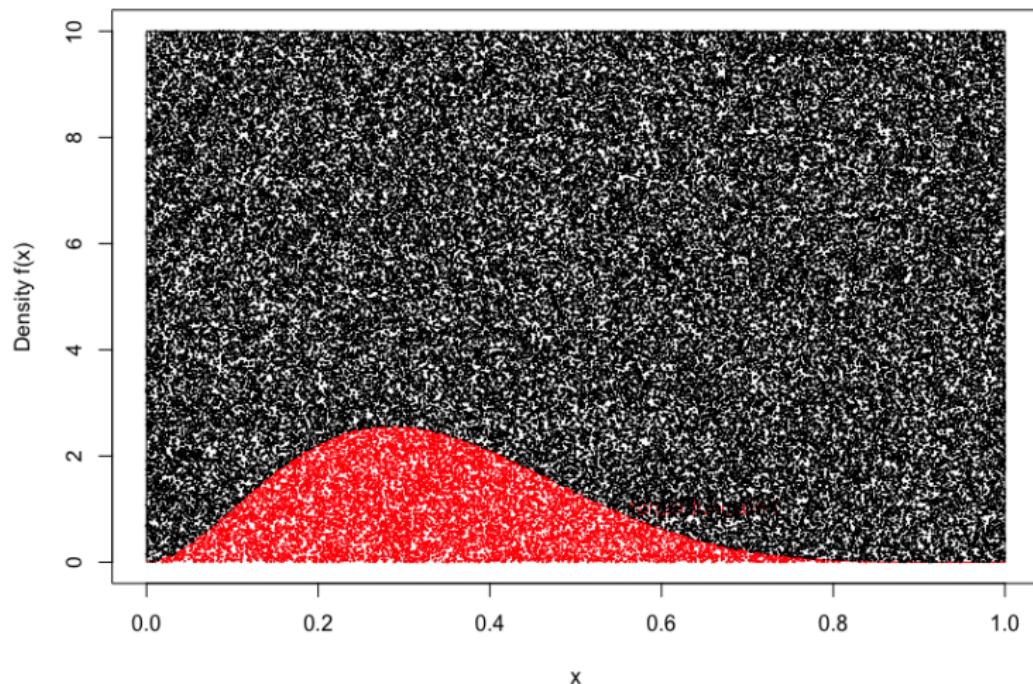
x2      <- runif(100000, 0, 1)
y2      <- runif(100000, 0, 10)
selected <- y2 < dbeta(x2, 3, 6)

mean(selected) # Proportion selected

points (x2, y2, col = 1+selected, cex = 0.1)
```

# GEOMETRIC IDEA

A Sample Distribution



# THE ENVELOPE REJECTION METHOD

Why does it work?

$$P(X_n \leq x) = P\left(X_n \leq x \middle| U \leq \frac{f(X)}{cg(X)}\right) = \dots = \int_{-\infty}^x f(z) dz$$

Exercise: Fill in the missing pieces with Bayes' Rule.

# THE ENVELOPE REJECTION METHOD

Good methods have the following properties:

1. Constant  $c > 0$  chosen such that  $cg(x) > f(x)$  for all  $x$ .
2. Easy to sample from  $g$ .
3. Generate few rejected draws, i.e.  $1/c$  is big or  $c$  is small.

A simple approach to finding the envelope:

Determine  $\max_x\{f(x)\}$ , then use a uniform distribution as  $g$ , and  $c = \max_x\{f(x)\}$ .

## EXAMPLE: BETA DISTRIBUTION

### Beta(4,3) distribution

Target density:  $f(x) = 60x^3(1 - x)^2$  for  $0 \leq x \leq 1$ .

- ▶ Can't invert  $f(x)$  analytically, so can't use inverse transform method.
- ▶ We'll take  $g$  to be the uniform distribution on  $[0, 1]$ . Then,  $g(x) = 1$ .
- ▶ Let  $f.\max = \max_{x \in [0,1]} f(x)$ , then we form envelope with  $c = f.\max$ , so that

$$cg(x) = f.\max \geq f(x).$$

- ▶ Take the derivative to find the maximum:

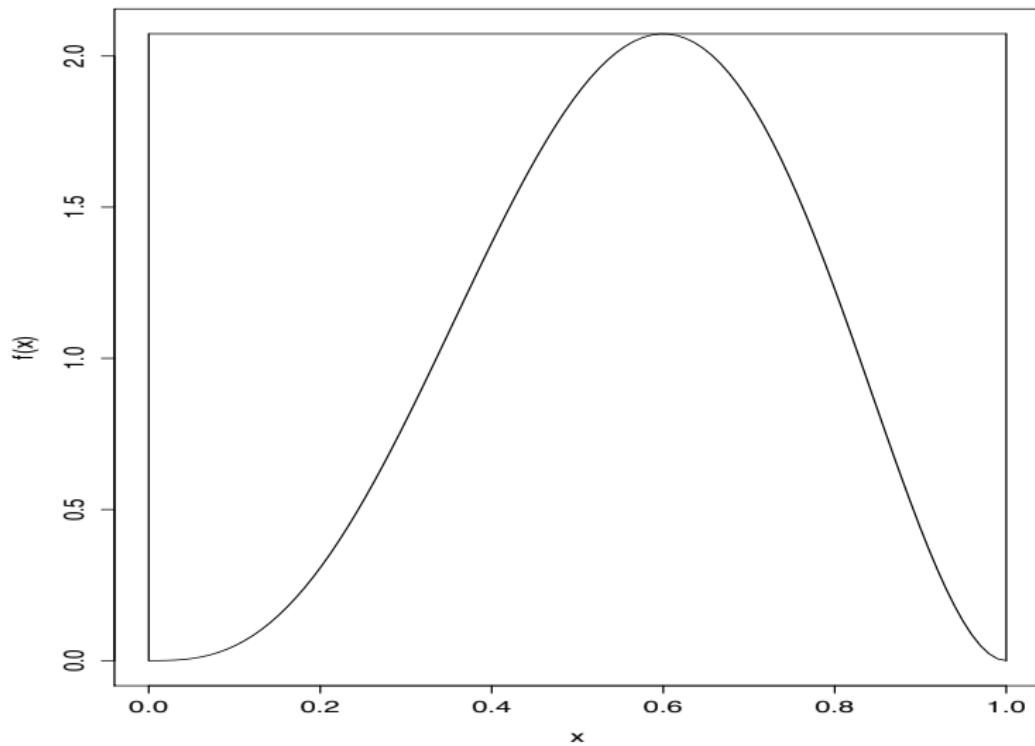
$$f'(x) = 180x^2(1 - x)^2 - 120x^3(1 - x) = 0 \quad \rightarrow \quad x.\max = 0.6.$$

# EXAMPLE: BETA PDF AND ENVELOPE

## Solution Part I

```
f <- function(x) {  
  stopifnot(x >= 0 & x <= 1)  
  return(60*x^3*(1-x)^2)  
}  
  
x <- seq(0, 1, length = 100)  
plot(x, f(x), type="l", ylab="f(x)", col = "red")  
  
xmax <- 0.6  
f.max <- 60*xmax^3*(1-xmax)^2  
  
lines(c(0, 0), c(0, f.max), lty = 1)  
lines(c(0, 1), c(f.max, f.max), lty = 1)  
lines(c(1, 1), c(f.max, 0), lty = 1)
```

## EXAMPLE: BETA PDF AND ENVELOPE



# EXAMPLE: ACCEPT-REJECT ALGORITHM FOR BETA DISTRIBUTION

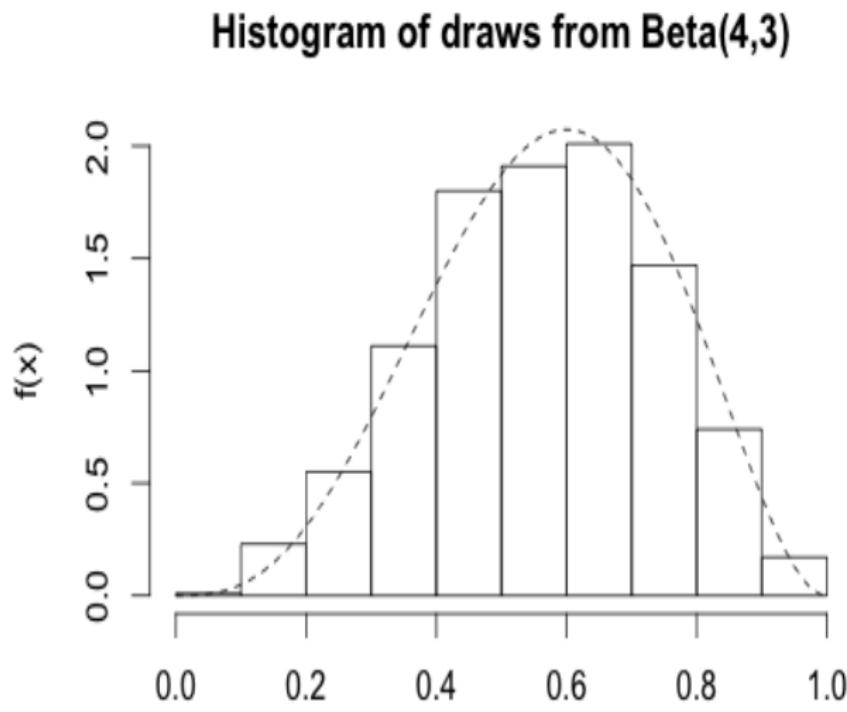
## Solution Part II

```
n.samps <- 1000      # number of samples desired
n       <- 0          # counter for number samples accepted
samps  <- numeric(n.samps) # initialize the vector of output

while (n < n.samps) {
  x <- runif(1)    #random draw from g
  u <- runif(1)
  if (f.max*u < f(x)) {
    n       <- n + 1
    samps[n] <- x
  }
}

x <- seq(0, 1, length = 100)
hist(samps, prob = T, ylab = "f(x)", xlab = "x",
      main = "Histogram of draws from Beta(4,3)")
lines(x, dbeta(x, 4, 3), lty = 2)
```

## EXAMPLE: ACCEPT-REJECT ALGORITHM FOR BETA DISTRIBUTION



# MONTE CARLO METHODS

# MONTE CARLO METHODS

- ▶ From what we've studied up until now, we have the tools to simulate statistical models.
- ▶ Now we'll briefly discuss how these simulations can be used to study properties of the underlying models.
- ▶ Approach: generate a large number of samples from a model and learn about the behavior of the model by studying the statistical properties of the samples.

## Examples

- ▶ Expected value of a random variable can be approximated by the average of a large number of samples (LLN).
- ▶ The probability of an event can be approximated by the proportion of occurrences in a large number of samples.
- ▶ The quality of an inference method can be assessed by repeatedly generating synthetic data and testing how well the method recovers (known) properties of the synthetic data.

# MONTE CARLO METHODS

## Today

- ▶ Focus on computing expectations of the form  $\mathbb{E}[f(X)]$  since many interesting questions can be reduced to this kind of problem.
- ▶ Approaches:
  - ▶ Analytic: sometimes can compute directly by integration – if  $X$  has density  $\phi(\cdot)$  then

$$\mathbb{E}[f(X)] = \int f(x)\phi(x)dx.$$

- ▶ Numerical integration: when the analytic approach fails, we can try to approximate the integral.
- ▶ We study Monte Carlo Integration – use the idea that if  $X_1, X_2, \dots$  are i.i.d. with the same distribution as  $X$  then with probability 1,

$$\mathbb{E}[f(X)] = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{j=1}^N f(X_j).$$

# MONTE CARLO METHODS

## Monte Carlo Integration

To estimate  $\mathbb{E}[f(X)]$  we use the approximation

$\mathbb{E}[f(X)] \approx \frac{1}{N} \sum_{j=1}^N f(X_j)$  for large  $N$  when  $X_1, X_2, \dots$  are i.i.d. with the same distribution as  $X$ . This also allows us to estimate:

- ▶ Probabilities:

$$\mathbb{P}(X \leq a) = \mathbb{E}[\mathbb{I}\{X \leq a\}].$$

- ▶ Definite Integrals:

$$\int_a^b f(x) dx = (b - a) \int_a^b f(x) \frac{1}{b - a} dx = (b - a) \mathbb{E}[f(X)],$$

where  $X \sim \text{Uniform}[a, b]$ .

- ▶ Indefinite Integrals: using a known density  $p(\cdot)$ ,

$$\int g(x) dx = \int \frac{g(x)}{p(x)} p(x) dx = \mathbb{E} \left[ \frac{g(X)}{p(X)} \right],$$

where  $X \sim p$ .

## EXAMPLE

Suppose  $X \sim \mathcal{N}\left(0, \frac{1}{\sqrt{2}}\right)$ . Then computing the expectation  $\mathbb{E}[\sin(X)^2]$  or the probability  $\mathbb{P}(X \leq 1.36)$  are analytically difficult but easily done via simulation.

```
n      <- 10000000
norms <- rnorm(n, sd = 1/sqrt(2))
est <- mean(sin(norms)^2)
est
est2 <- mean(norms <= 1.36)
est2
pnorm(1.36, sd = 1/sqrt(2))
```

# EXAMPLE

Estimate the integral

$$\int_{-\infty}^{\infty} \sin(x)^2 e^{-x^2} dx,$$

using MC techniques.

## Solution

We'll use standard normal random variables and a standard normal density denoted  $p(x)$ :

$$\int \sin(x)^2 e^{-x^2} dx = \int \frac{\sin(x)^2 e^{-x^2}}{\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}} p(x) dx = \sqrt{2\pi} \mathbb{E} \left[ \sin(X)^2 e^{-\frac{1}{2}X^2} \right]$$

```
n      <- 10000000
norms <- rnorm(n)
est    <- sqrt(2*pi) * mean(sin(norms)^2 * exp(-(1/2)*norms^2))
est
```

## EXAMPLE

There's an easier way...

Looking again:

$$\int \sin(x)^2 e^{-x^2} dx = \sqrt{\pi} \int \sin(x)^2 \left( \frac{1}{\sqrt{\pi}} e^{-x^2} \right) dx = \sqrt{\pi} \mathbb{E}[\sin(X)^2],$$

where  $X \sim \mathcal{N}(0, \frac{1}{\sqrt{2}})$ . From work earlier:

```
n      <- 10000000
norms <- rnorm(n, sd = 1/sqrt(2))
est    <- sqrt(pi)*mean(sin(norms)^2)
est
```

# MONTE CARLO INTEGRATION

By the Central Limit Theorem,

$$\frac{1}{N} \sum_{i=1}^N f(X_i) \xrightarrow{d} \mathcal{N} \left( \int f(x) dx, \frac{1}{\sqrt{N}} \text{sd}(f(X)) \right).$$

- ▶ The Monte Carlo approximation is unbiased.
- ▶ Note that by increasing  $N$  the error can get as small as you'd like, even if  $f$  or  $X$  are very complicated.
- ▶ On the other hand, larger  $N$  means more computational time.

## EXAMPLE

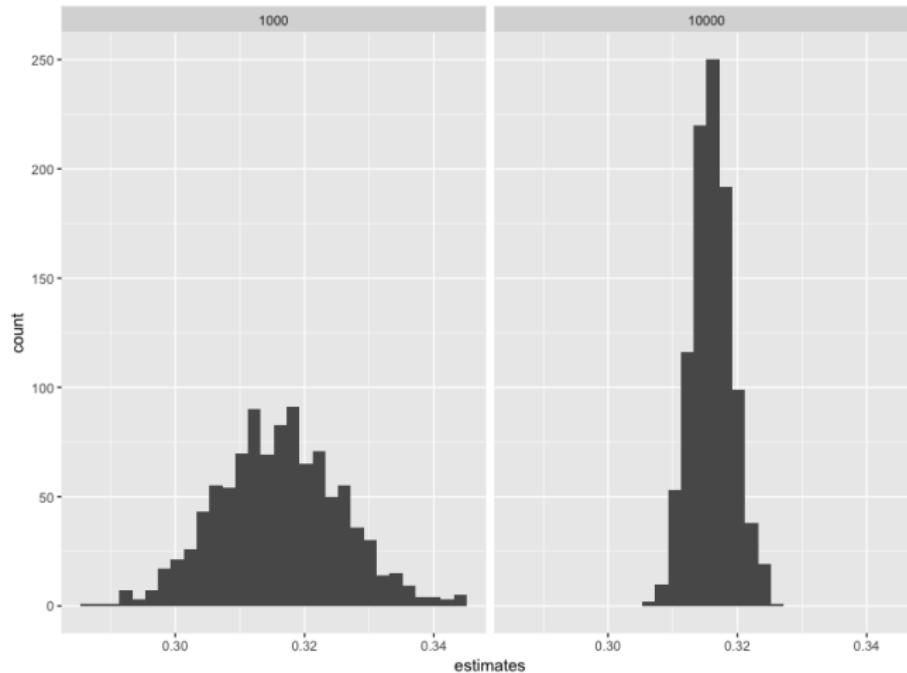
Computing the expectation  $\mathbb{E}[\sin(X)^2]$  for  $X \sim \mathcal{N}\left(0, \frac{1}{\sqrt{2}}\right)$ .

```
n1 <- 10000; n2 <- 1000
estvec1 <- rep(NA, 1000); estvec2 <- rep(NA, 1000)

for (i in 1:1000) {
  norms1      <- rnorm(n1, sd = 1/sqrt(2))
  estvec1[i] <- mean(sin(norms1)^2)
}
for (i in 1:1000) {
  norms2      <- rnorm(n2, sd = 1/sqrt(2))
  estvec2[i] <- mean(sin(norms2)^2)
}

df <- data.frame(estimates = c(estvec1, estvec2),
                  n = c(rep(n1, 1000), rep(n2, 1000)))
ggplot(df) +
  geom_histogram(aes(x = estimates)) + facet_wrap(~ n, ncol = 2)
```

# EXAMPLE



# CHECK YOURSELF

## Tasks

- ▶ Estimate  $P(X < 3)$  where  $X$  is an exponentially distributed random variable with  $rate = 1/3$ .
- ▶ Use built-in R functions to find the exact probability.

```
n  <- 1000000
samps <- rexp(n, rate = 1/3)
mean(samps < 3)
pexp(3, rate = 1/3)
```

# CHECK YOURSELF

## Tasks

Recall, the binomial dist with size  $n$  and success prob  $p$  describes the number of successes in  $n$  independent trials, each with prob  $p$  of success.

- ▶ Generate 5000 binomial rvs with  $n = 16$  and  $p = 0.5$ .
- ▶ Compute the sample mean and sample standard deviation of these rvs. Are they close to what you would expect?
- ▶ Plot a histogram of the rvs, with a large value of the breaks input (say `breaks=100`), on the probability scale. What do you notice about its support? Is this surprising?
- ▶ R's capacity for simulation and computation is impressive: generate 5 million binomial rvs with  $n = 1$  million and  $p = 0.5$ .
- ▶ Standardize the rvs: that is, subtract off the sample mean, and then divide by the sample standard deviation then plot the values as above.

# Lecture 8: Distributions and Simulations

## STAT GR5206

*Statistical Computing & Introduction to Data Science*

Cynthia Rush  
Columbia University

October 27, 2017

# COURSE NOTES

- ▶ I'll leave the last 20 minutes to of class for questions about the exam.
- ▶ Homework due Monday.
- ▶ Lab next week is a review session – nothing planned, bring questions.

# SIMULATING RANDOM VARIABLES FROM UNCOMMON PROBABILITY DISTRIBUTIONS

# SIMULATING FROM UNCOMMON PROBABILITY DISTRIBUTIONS

We study methods for transforming i.i.d. uniform  $[0, 1]$ –distributed random variables into a prescribed target distribution.

- ▶ Useful when we want to sample from a distribution that's not one with built-in R functions.
- ▶ Helpful in understanding how R's simulation functions build off of PRNGs.

Discuss different methods, applicable to different classes of target distributions.

# A SIMPLE EXAMPLE

## Target Distribution

Want  $X$  uniformly distributed on the set  $\{0, 1, \dots, n - 1\}$ , i.e.

$$P(X = k) = \frac{1}{n} \text{ for } k = 0, 1, \dots, n - 1.$$

- ▶ Maybe use a PRNG with a state space  $0, 1, \dots, n - 1$ ? Not a good idea since we want a long period.
- ▶ Instead: first generate a rv  $U \sim \text{uniform } [0, 1]$ , then transform with  $X = \lfloor nU \rfloor$ .
- ▶ Can show rigorously that  $X$  has the desired distribution.

```
n <- 10
samp <- 100
table(floor(n*runif(samp)))/samp
samp <- 10000
table(floor(n*runif(samp)))/samp
```

# ANOTHER SIMPLE EXAMPLE

## Target Distribution

Want  $X$  to take values in  $\{1, 2, 3\}$  with probabilities  $(1/2, 1/4, 1/4)$ .

- ▶ Use the fact that for a rv  $U \sim \text{uniform } [0, 1]$  and an event  $E = \{U \leq p\}$  then  $P(E) = p$ .
- ▶ Define  $X$  as follows:

$$X = \begin{cases} 1 & \text{if } 0 \leq U < 1/2 \\ 1 & \text{if } 1/2 \leq U < 3/4 \\ 1 & \text{if } 3/4 \leq U \leq 1 \end{cases}$$

# INVERSE TRANSFORM METHOD

- ▶ The Inverse Transform Method is a general method that can be used when the target distribution is one-dimensional.
- ▶ Uses the inverse cdf denoted  $F^{-1}$  of the target distribution:

$$F^{-1}(u) = \inf\{x \in \mathbb{R} | F(x) \geq u\}$$

for all  $u \in (0, 1)$ .

- ▶ If  $F$  is bijective (i.e.  $F$  is strictly monotonically increasing with no jumps), then  $F^{-1}$  is the usual inverse and can be found by solving  $F(x) = u$  for  $x$ .

# INVERSE TRANSFORM METHOD

## Method

Whenever  $F^{-1}$  can be determined, then  $X = F^{-1}(U)$  is such that  $X \sim F$ .

Why does this work?

$$\begin{aligned} P(X \leq x) &= P(F^{-1}(U) \leq x) \\ &\stackrel{(a)}{=} P(F(F^{-1}(U)) \leq F(x)) \\ &= P(U \leq F(x)) \\ &= F(x), \end{aligned}$$

where (a) follows by monotonicity of  $F$ .

# INVERSE TRANSFORM METHOD

## Algorithm

1. Derive the inverse function  $F^{-1}$ . To do this:
  - ▶ Solve  $F(x) = u$  for  $x$  to find  $x = F^{-1}(u)$ .
2. Write a function to compute  $x = F^{-1}(u)$ .
3. For each realization:
  - ▶ Generate a random value  $u$  from Uniform(0,1).
  - ▶ Compute  $x = F^{-1}(u)$  as a realization from the target distribution.

# INVERSE TRANSFORM METHOD

Example: Target distribution is exponential with  $\lambda = 2$ .

The pdf of the exponential distribution is  $f(x) = \lambda e^{-\lambda t}$ , so the cdf is

$$F(x) = \int_0^x f(t) dt = \int_0^x \lambda e^{-\lambda t} dt = 1 - e^{-\lambda x}.$$

Now we invert the cdf.

$$u = 1 - e^{-\lambda x} \quad \rightarrow \quad x = -\frac{1}{\lambda} \log(1 - u)$$

# INVERSE TRANSFORM METHOD

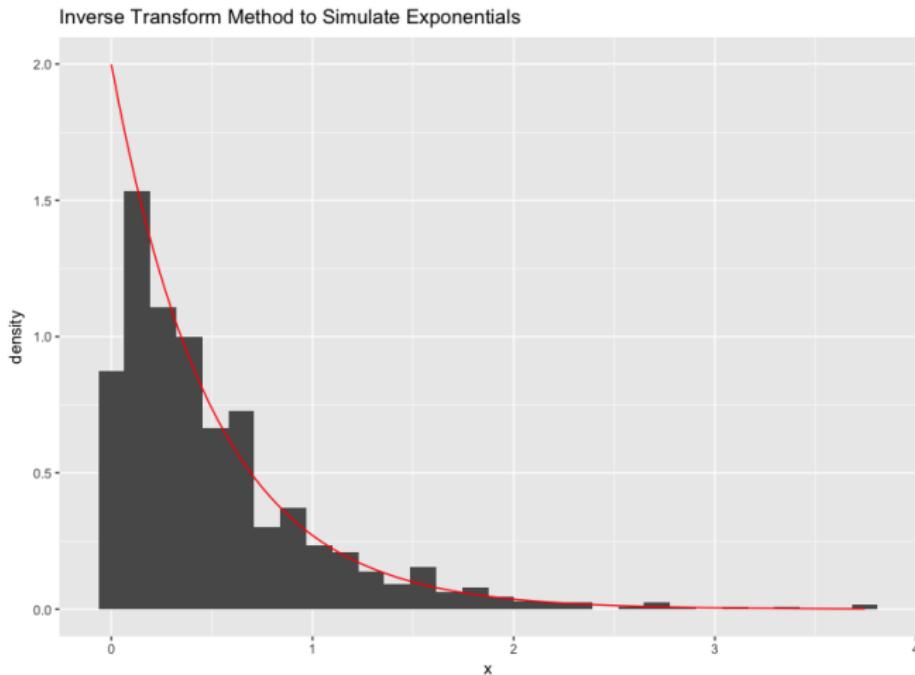
Example: Target distribution is exponential with  $\lambda = 2$ .

```
lambda <- 2
n      <- 1000
u      <- runif(n) # Simulating uniform rvs

Finverse <- function(u, lambda) {
  # Function for the inverse transform
  stopifnot(u > 0 & u < 1)
  return(-(1/lambda)*log(1-u))
}
x <- Finverse(u, lambda)

ggplot(data = data.frame(x)) +
  geom_histogram(aes(x = x, y = ..density..)) +
  stat_function(mapping = aes(x = x), fun = dexp,
                args = list(rate = 2), color = "red") +
  labs(title = "Inverse Transform Method to Simulate Exponential
```

# INVERSE TRANSFORM METHOD: EXPONENTIAL



# CHECK YOURSELF

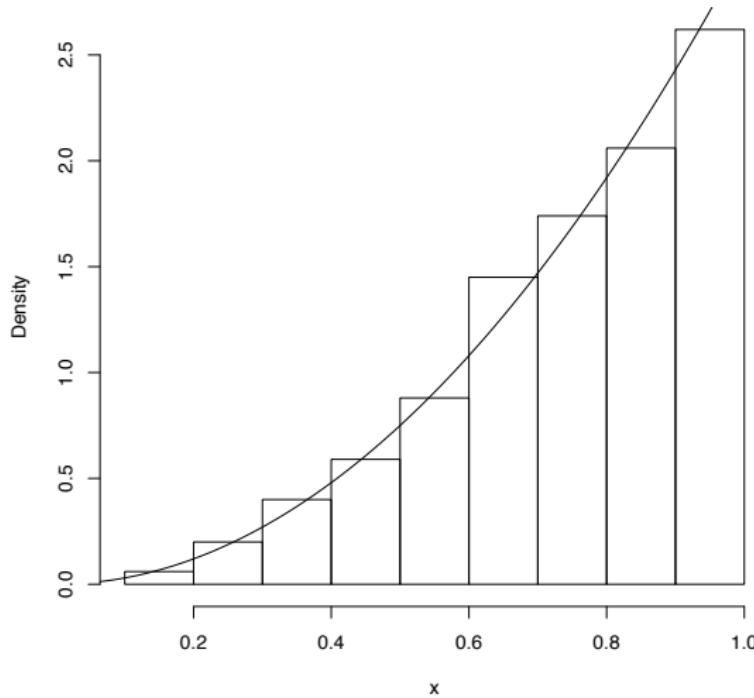
## Task

Simulate a random sample of size 1000 from the pdf  $f_X(x) = 3x^2$ ,  $0 \leq x \leq 1$ .

- ▶ Find  $F$  and then  $F^{-1}$ .
- ▶ Plot the empirical distribution (histogram) with the correct density also shown on the plot.

# INVERSE TRANSFORM METHOD

$$f(x) = 3x^2$$



# ACCEPTANCE-REJECTION ALGORITHM

The inverse transform method can be applied when  $F^{-1}$  is easy to evaluate. Sometimes this isn't the case (e.g. the normal distribution).

- ▶ **Rejection sampling** is a more advanced and very popular method for random number generation.
- ▶ How does it work? By sampling candidates from an easier to generate distribution then correcting the sampling probability by randomly rejecting some candidates.

## Pros and Cons

- ▶ Not restricted to Uniform[0,1] input samples. Idea: generate samples of approximately the correct distribution, then reject some to hit target distribution exactly.
- ▶ Can be generalized to work on very general spaces beyond  $\mathbb{R}^d$ .
- ▶ Requires a random and potentially large number of input samples to generate one output, so efficiency can be a concern.

# THE REJECTION METHOD

## Basic Algorithm

- ▶ Input:
  - ▶ A probability density  $g$  (the **proposal** density)
  - ▶ A function  $p$  with values in  $[0,1]$  (the **acceptance probability**)
- ▶ Random Values:
  - ▶  $X_n$  i.i.d. with density  $g$  (the **proposals**)
  - ▶  $U_n$  i.i.d. Uniform[0,1]
- ▶ Output: A sequence of i.i.d. random variables with density

$$f(x) = \frac{1}{Z} p(x)g(x) \text{ where } Z = \int p(x)g(x)dx.$$

- ▶ Algorithm: For  $n = 1, 2, 3, \dots$ 
  - ▶ Generate  $X_n \sim g$  and  $U_n \sim \text{Uniform}[0,1]$
  - ▶ if  $U_n \leq p(X_n)$  then output  $X_n$ .

# THE REJECTION METHOD

## Notes

- ▶  $U_n$  values randomly decide whether to output or ignore  $X_n$ .
- ▶ Value  $X_n$  output with probability  $p(X_n)$ .
- ▶ If the  $X_n$  value is chosen, say it is **accepted**. Otherwise it is **rejected**.
- ▶ Idea: we can sample from  $g$  easily and then use this to sample from  $f$ .

## Theory

- ▶ The elements of the output are i.i.d. with density  $f$ :

$$f(x) = \frac{1}{Z} p(x)g(x) \text{ where } Z = \int p(x)g(x) dx.$$

- ▶ Each proposal is accepted with probability  $Z$  so the number of proposals required to generate each output is geometrically distributed with mean  $Z$ .

# THE REJECTION METHOD

## Example

- ▶ Let the proposal density  $g$  be Uniform[-1, +1] and the acceptance probability  $p(x) = \sqrt{1 - x^2}$ .
- ▶ Accepted samples have density

$$f(x) = \frac{1}{Z} p(x)g(x) = \frac{4}{\pi} \sqrt{1 - x^2} \times \frac{1}{2} \mathbb{I}\{-1 \leq x \leq +1\}.$$

- ▶ This is the ‘semicircle distribution’.

# THE REJECTION METHOD

```
p <- function(x) {sqrt(1 - x^2)}
```

```
generate_one <- function(p_func) {
  val <- NULL
  while (is.null(val)) {
    x <- runif(1, min = -1, max = +1)
    u <- runif(1)
    if (u <= p_func(x)) {val <- x}
  }
  return(val)
}
```

```
num <- 1000
samp <- rep(NA, num)
for (i in 1:num) {samp[i] <- generate_one(p)}
hist(samp)
```

# THE ENVELOPE REJECTION METHOD

- ▶ Usually run basic algorithm by choosing the acceptance probabilities  $p$  such that the output density

$$f(x) = \frac{1}{Z} p(x) g(x)$$

coincides with the target density.

- ▶ We can write the algorithm more directly with this in mind.

# THE ENVELOPE REJECTION METHOD

## Algorithm

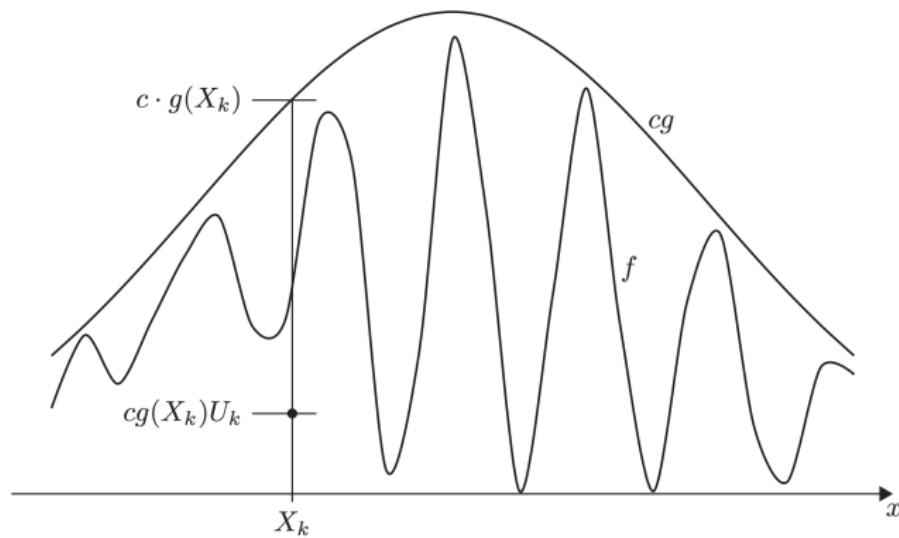
- ▶ Input:
  - ▶ A probability density  $f$  (the **target density**)
  - ▶ A probability density  $g$  (the **proposal** density)
  - ▶ A constant  $c > 0$  such that  $f(x) \leq cg(x)$  for all  $x$
- ▶ Random Values:
  - ▶  $X_n$  i.i.d. with density  $g$  (the **proposals**)
  - ▶  $U_n$  i.i.d. Uniform[0,1]
- ▶ Output: A sequence of i.i.d. random variables with density  $f$
- ▶ Algorithm: For  $n = 1, 2, 3, \dots$ 
  - ▶ Generate  $X_n \sim g$  and  $U_n \sim \text{Uniform}[0,1]$
  - ▶ if  $cg(X_n) U_n \leq f(X_n)$  then output  $X_n$ .

# THE ENVELOPE REJECTION METHOD

## Notes

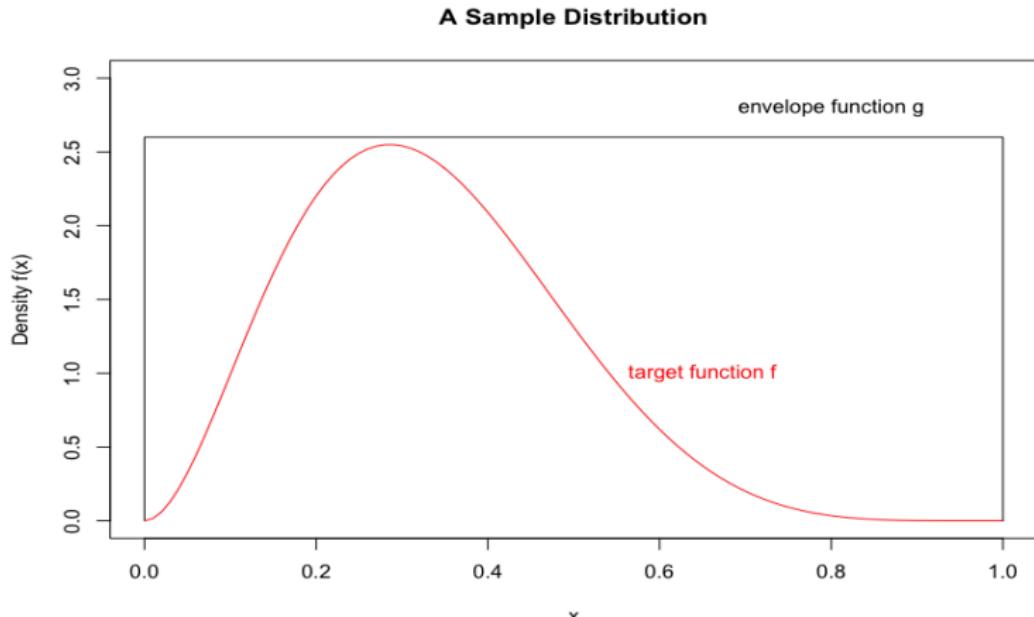
- ▶ This is the basic algorithm with acceptance probability  $p(x) = \frac{f(x)}{cg(x)}$  if  $g(x) > 0$  and  $p(x) = 1$  otherwise.
- ▶ The elements of the output are i.i.d. with density  $f$ .
- ▶ Each proposal is accepted with probability  $1/c$  so the number of proposals required to generate each output is geometrically distributed with mean  $1/c$ .
- ▶ Function  $cg$  sometimes called the ‘envelope’ for target density  $f$ .
- ▶ Actually could just use input: a function  $f$  (the non-normalized **target density**). Don’t need to be able to calculate  $Z = \int f(x)dx$  for the algorithm to work!

# THE ENVELOPE REJECTION METHOD



# AN EASY EXAMPLE

Suppose the pdf  $f$  is zero outside an interval  $[c, d]$  (in the image  $[0,1]$ ), and  $\leq M$  on the interval (in the image  $M \approx 2.5$ ). Can then choose proposal density  $g$  to be uniform.



# GEOMETRIC IDEA

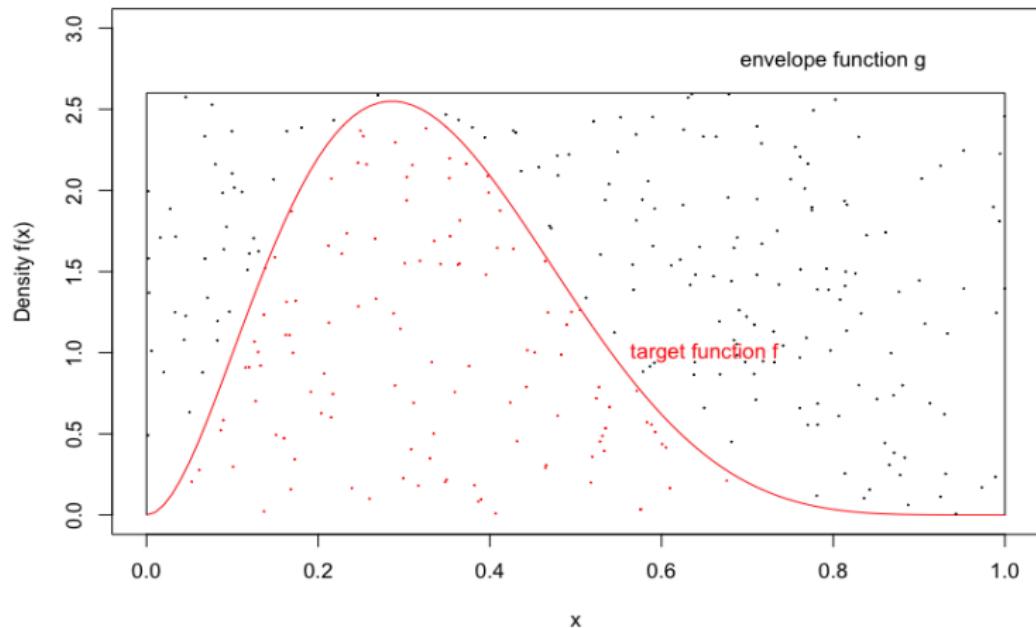
```
plot(c(0,1), c(0,3), ty = "n", main = "A Sample Distribution",
      ylab = "Density f(x)", xlab = "x")
curve (dbeta(x, 3, 6), add = TRUE, col = "red")
text(.65, 1, "target function f", col = "red")
text(.8, 2.8, "envelope function g")
lines(c(0,0,1,1), c(0,2.6,2.6,0))

x1      <- runif(300, min = 0, max = 1);
y1      <- runif(300, min = 0, max = 2.6)
selected <- y1 < dbeta(x1, 3, 6)

points (x1, y1, col = 1+selected, cex = 0.1)
```

# GEOMETRIC IDEA

A Sample Distribution



# GEOMETRIC IDEA

```
mean(selected) # Proportion selected
accepted.points <- x1[selected]

# Proportion of sample points less than 0.5.
mean(accepted.points < 0.5)

# The true distribution.
pbeta(0.5, 3, 6)
```

# GEOMETRIC IDEA

For this to work efficiently, we have to cover the target distribution with one that sits close to it.

```
plot(c(0,1), c(0,10), ty = "n", main = "A Sample Distribution",
      ylab = "Density f(x)", xlab = "x")
curve (dbeta(x, 3, 6), add = TRUE, col = "red")
text(.65, 1, "target function f", col = "red")
text(.8, 9.7, "envelope function g")
lines(c(0,0,1,1), c(0,10,10,0))

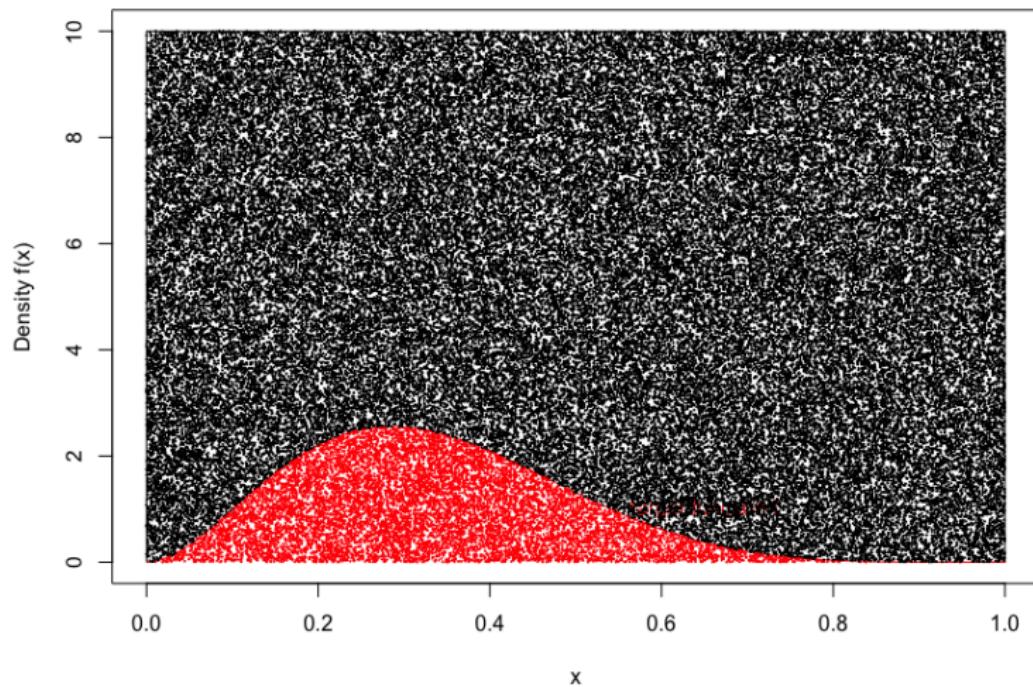
x2      <- runif(100000, 0, 1)
y2      <- runif(100000, 0, 10)
selected <- y2 < dbeta(x2, 3, 6)

mean(selected) # Proportion selected

points (x2, y2, col = 1+selected, cex = 0.1)
```

# GEOMETRIC IDEA

A Sample Distribution



# THE ENVELOPE REJECTION METHOD

Why does it work?

$$P(X_n \leq x) = P\left(X_n \leq x \middle| U \leq \frac{f(X)}{cg(X)}\right) = \dots = \int_{-\infty}^x f(z) dz$$

Exercise: Fill in the missing pieces with Bayes' Rule.

# THE ENVELOPE REJECTION METHOD

Good methods have the following properties:

1. Constant  $c > 0$  chosen such that  $cg(x) > f(x)$  for all  $x$ .
2. Easy to sample from  $g$ .
3. Generate few rejected draws, i.e.  $1/c$  is big or  $c$  is small.

A simple approach to finding the envelope:

Determine  $\max_x\{f(x)\}$ , then use a uniform distribution as  $g$ , and  $c = \max_x\{f(x)\}$ .

## EXAMPLE: BETA DISTRIBUTION

### Beta(4,3) distribution

Target density:  $f(x) = 60x^3(1 - x)^2$  for  $0 \leq x \leq 1$ .

- ▶ Can't invert  $f(x)$  analytically, so can't use inverse transform method.
- ▶ We'll take  $g$  to be the uniform distribution on  $[0, 1]$ . Then,  $g(x) = 1$ .
- ▶ Let  $f.\max = \max_{x \in [0,1]} f(x)$ , then we form envelope with  $c = f.\max$ , so that

$$cg(x) = f.\max \geq f(x).$$

- ▶ Take the derivative to find the maximum:

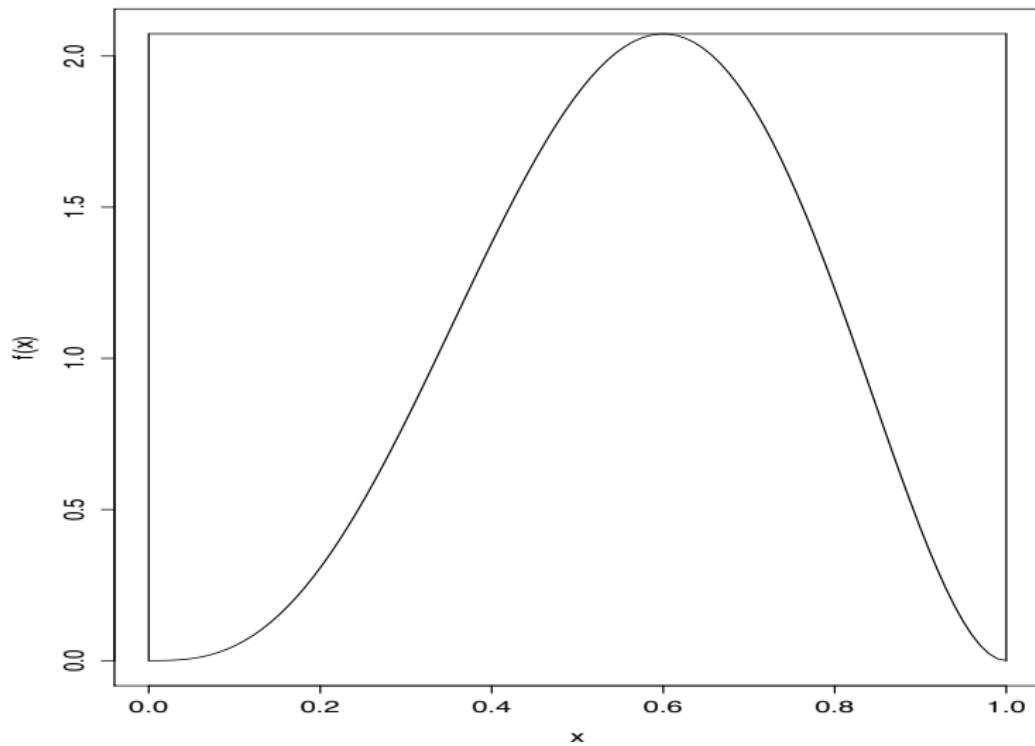
$$f'(x) = 180x^2(1 - x)^2 - 120x^3(1 - x) = 0 \quad \rightarrow \quad x.\max = 0.6.$$

# EXAMPLE: BETA PDF AND ENVELOPE

## Solution Part I

```
f <- function(x) {  
  stopifnot(x >= 0 & x <= 1)  
  return(60*x^3*(1-x)^2)  
}  
  
x <- seq(0, 1, length = 100)  
plot(x, f(x), type="l", ylab="f(x)", col = "red")  
  
xmax <- 0.6  
f.max <- 60*xmax^3*(1-xmax)^2  
  
lines(c(0, 0), c(0, f.max), lty = 1)  
lines(c(0, 1), c(f.max, f.max), lty = 1)  
lines(c(1, 1), c(f.max, 0), lty = 1)
```

## EXAMPLE: BETA PDF AND ENVELOPE



# EXAMPLE: ACCEPT-REJECT ALGORITHM FOR BETA DISTRIBUTION

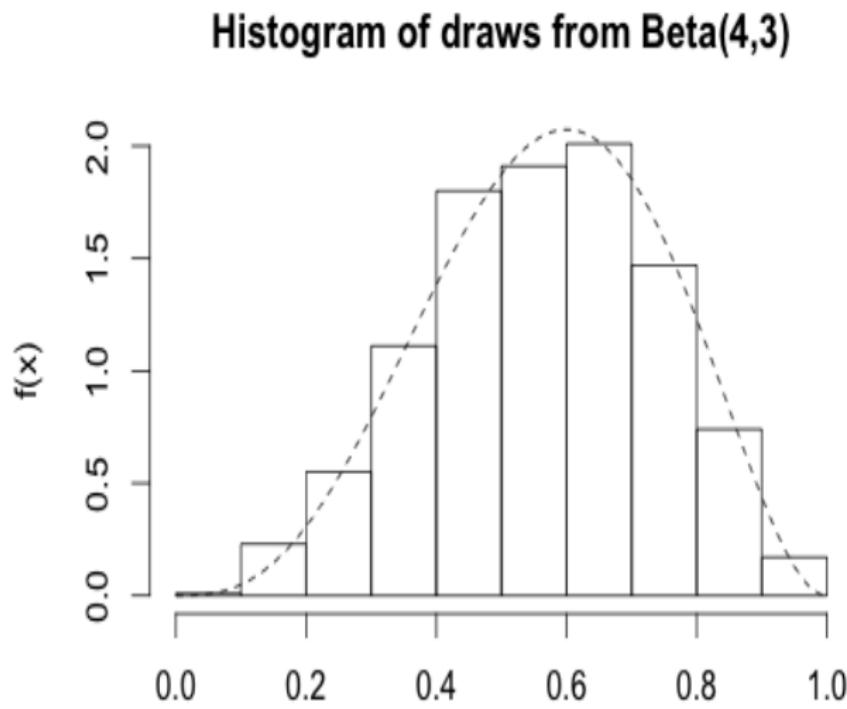
## Solution Part II

```
n.samps <- 1000      # number of samples desired
n       <- 0          # counter for number samples accepted
samps  <- numeric(n.samps) # initialize the vector of output

while (n < n.samps) {
  x <- runif(1)    #random draw from g
  u <- runif(1)
  if (f.max*u < f(x)) {
    n       <- n + 1
    samps[n] <- x
  }
}

x <- seq(0, 1, length = 100)
hist(samps, prob = T, ylab = "f(x)", xlab = "x",
      main = "Histogram of draws from Beta(4,3)")
lines(x, dbeta(x, 4, 3), lty = 2)
```

## EXAMPLE: ACCEPT-REJECT ALGORITHM FOR BETA DISTRIBUTION



# MONTE CARLO METHODS

# MONTE CARLO METHODS

- ▶ From what we've studied up until now, we have the tools to simulate statistical models.
- ▶ Now we'll briefly discuss how these simulations can be used to study properties of the underlying models.
- ▶ Approach: generate a large number of samples from a model and learn about the behavior of the model by studying the statistical properties of the samples.

## Examples

- ▶ Expected value of a random variable can be approximated by the average of a large number of samples (LLN).
- ▶ The probability of an event can be approximated by the proportion of occurrences in a large number of samples.
- ▶ The quality of an inference method can be assessed by repeatedly generating synthetic data and testing how well the method recovers (known) properties of the synthetic data.

# MONTE CARLO METHODS

## Today

- ▶ Focus on computing expectations of the form  $\mathbb{E}[f(X)]$  since many interesting questions can be reduced to this kind of problem.
- ▶ Approaches:
  - ▶ Analytic: sometimes can compute directly by integration – if  $X$  has density  $\phi(\cdot)$  then

$$\mathbb{E}[f(X)] = \int f(x)\phi(x)dx.$$

- ▶ Numerical integration: when the analytic approach fails, we can try to approximate the integral.
- ▶ We study Monte Carlo Integration – use the idea that if  $X_1, X_2, \dots$  are i.i.d. with the same distribution as  $X$  then with probability 1,

$$\mathbb{E}[f(X)] = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{j=1}^N f(X_j).$$

# MONTE CARLO METHODS

## Monte Carlo Integration

To estimate  $\mathbb{E}[f(X)]$  we use the approximation

$\mathbb{E}[f(X)] \approx \frac{1}{N} \sum_{j=1}^N f(X_j)$  for large  $N$  when  $X_1, X_2, \dots$  are i.i.d. with the same distribution as  $X$ . This also allows us to estimate:

- ▶ Probabilities:

$$\mathbb{P}(X \leq a) = \mathbb{E}[\mathbb{I}\{X \leq a\}].$$

- ▶ Definite Integrals:

$$\int_a^b f(x) dx = (b - a) \int_a^b f(x) \frac{1}{b - a} dx = (b - a) \mathbb{E}[f(X)],$$

where  $X \sim \text{Uniform}[a, b]$ .

- ▶ Indefinite Integrals: using a known density  $p(\cdot)$ ,

$$\int g(x) dx = \int \frac{g(x)}{p(x)} p(x) dx = \mathbb{E} \left[ \frac{g(X)}{p(X)} \right],$$

where  $X \sim p$ .

## EXAMPLE

Suppose  $X \sim \mathcal{N}\left(0, \frac{1}{\sqrt{2}}\right)$ . Then computing the expectation  $\mathbb{E}[\sin(X)^2]$  or the probability  $\mathbb{P}(X \leq 1.36)$  are analytically difficult but easily done via simulation.

```
n      <- 10000000
norms <- rnorm(n, sd = 1/sqrt(2))
est   <- mean(sin(norms)^2)
est
est2 <- mean(norms <= 1.36)
est2
pnorm(1.36, sd = 1/sqrt(2))
```

# EXAMPLE

Estimate the integral

$$\int_{-\infty}^{\infty} \sin(x)^2 e^{-x^2} dx,$$

using MC techniques.

## Solution

We'll use standard normal random variables and a standard normal density denoted  $p(x)$ :

$$\int \sin(x)^2 e^{-x^2} dx = \int \frac{\sin(x)^2 e^{-x^2}}{\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}} p(x) dx = \sqrt{2\pi} \mathbb{E} \left[ \sin(X)^2 e^{-\frac{1}{2}X^2} \right]$$

```
n      <- 10000000
norms <- rnorm(n)
est    <- sqrt(2*pi) * mean(sin(norms)^2 * exp(-(1/2)*norms^2))
est
```

## EXAMPLE

There's an easier way...

Looking again:

$$\int \sin(x)^2 e^{-x^2} dx = \sqrt{\pi} \int \sin(x)^2 \left( \frac{1}{\sqrt{\pi}} e^{-x^2} \right) dx = \sqrt{\pi} \mathbb{E}[\sin(X)^2],$$

where  $X \sim \mathcal{N}(0, \frac{1}{\sqrt{2}})$ . From work earlier:

```
n      <- 10000000
norms <- rnorm(n, sd = 1/sqrt(2))
est    <- sqrt(pi)*mean(sin(norms)^2)
est
```

# MONTE CARLO INTEGRATION

By the Central Limit Theorem,

$$\frac{1}{N} \sum_{i=1}^N f(X_i) \xrightarrow{d} \mathcal{N} \left( \int f(x) dx, \frac{1}{\sqrt{N}} sd(f(X)) \right).$$

- ▶ The Monte Carlo approximation is unbiased.
- ▶ Note that by increasing  $N$  the error can get as small as you'd like, even if  $f$  or  $X$  are very complicated.
- ▶ On the other hand, larger  $N$  means more computational time.

## EXAMPLE

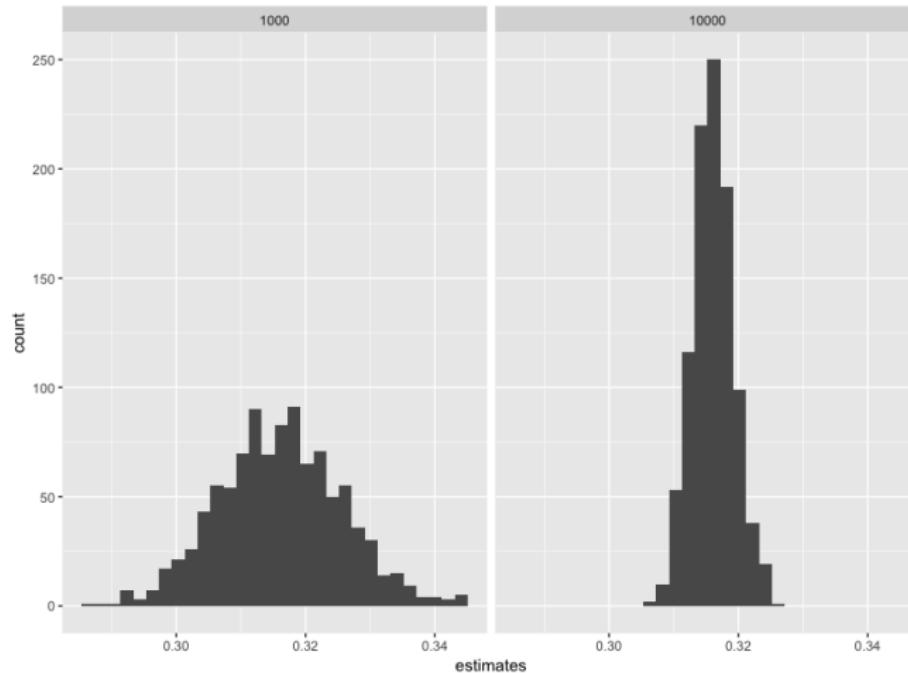
Computing the expectation  $\mathbb{E}[\sin(X)^2]$  for  $X \sim \mathcal{N}\left(0, \frac{1}{\sqrt{2}}\right)$ .

```
n1 <- 10000; n2 <- 1000
estvec1 <- rep(NA, 1000); estvec2 <- rep(NA, 1000)

for (i in 1:1000) {
  norms1      <- rnorm(n1, sd = 1/sqrt(2))
  estvec1[i] <- mean(sin(norms1)^2)
}
for (i in 1:1000) {
  norms2      <- rnorm(n2, sd = 1/sqrt(2))
  estvec2[i] <- mean(sin(norms2)^2)
}

df <- data.frame(estimates = c(estvec1, estvec2),
                  n = c(rep(n1, 1000), rep(n2, 1000)))
ggplot(df) +
  geom_histogram(aes(x = estimates)) + facet_wrap(~ n, ncol = 2)
```

# EXAMPLE



# CHECK YOURSELF

## Tasks

- ▶ Estimate  $P(X < 3)$  where  $X$  is an exponentially distributed random variable with  $rate = 1/3$ .
- ▶ Use built-in R functions to find the exact probability.

# MORE SIMULATIONS

# RANDOM NUMBER GENERATION IN R

Recall,

We can simulate random numbers in R from various distributions:

- ▶ `rnorm()`: generate normal random variables
- ▶ `pnorm()`: normal distribution function,  $\Phi(x) = P(Z < x)$
- ▶ `dnorm()`: normal density function,  $\phi(x) = \Phi'(x)$
- ▶ `qnorm()`: normal quantile function,  $\Phi^{-1}(u)$

Replace `norm` with other distribution names, all the same functions apply.

# WHY DO WE SIMULATE?

Recall,

R gives us great simulation tools (unlike many other languages). Why should we simulate, though?

- ▶ Often simulations are easier than hand calculations.
- ▶ Often simulations can be made more realistic than hand calculations.

# TODAY WE'LL DO A FUN SIMULATION!

## Darts<sup>1</sup>

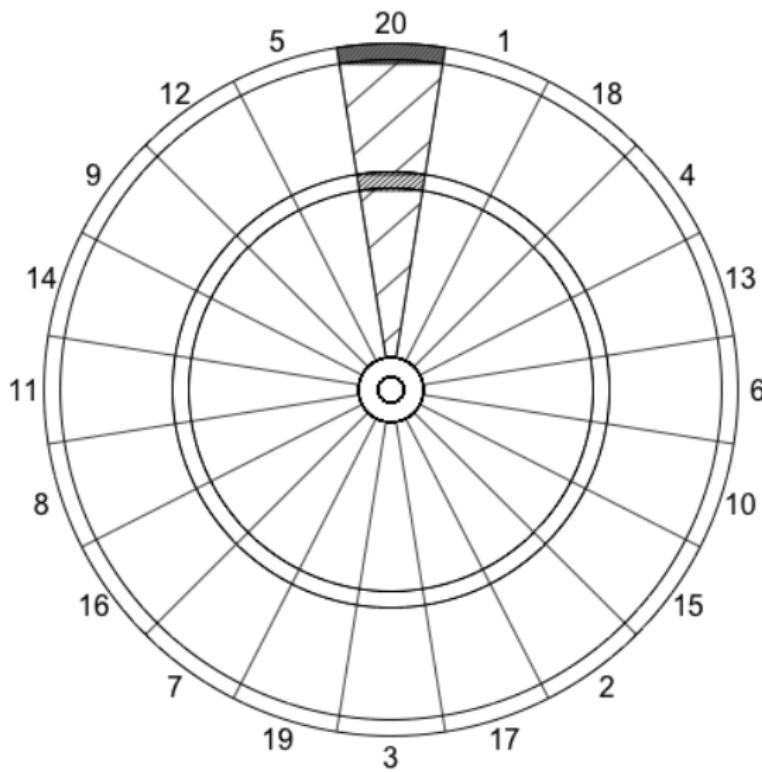
Darts is a game where you throw metal darts at a dartboard and receive different scores for landing the dart in different regions.

- ▶ Land in a slice, obtain the score of the corresponding number.
- ▶ Land in the “double ring”, get double the score.
- ▶ Land in the “triple ring”, get triple the score.
- ▶ Land in the “double bullseye”, get 50 points.
- ▶ Land in the “single bullseye”, get 25 points.

---

<sup>1</sup>Darts simulation notes adapted from ‘A Statistician Plays Darts’

# TODAY WE'LL DO A FUN SIMULATION!



# SOME QUESTIONS TO ANSWER WITH SIMULATION

- ▶ What strategy provides the better score? Throwing such that darts land uniformly at random on the board, or aiming at the center, with your throws being normally distributed with some variance.
- ▶ If you're aiming at a spot and your throws are normally distributed with some variance, what spot should you aim for?
- ▶ Can I estimate the variance of my throws, if I aim at the center and receive a certain score?

# LET'S SIMULATE DART THROWS

## Board Measurements

We'll make a list of standard dart board measurements and the numbers of the board. All measurements are in mm.

```
board = list(  
  R1 = 6.35,    # center to double bullseye ring  
  R2 = 15.9,    # center to single bullseye ring  
  R3 = 99,      # center to inner triple ring  
  R4 = 107,     # center to outer triple ring  
  R5 = 162,     # center to inner double ring  
  R = 170,      # center to outer double ring  
  nums = c(20,1,18,4,13,6,10,15,2,17,3,19,  
         7,16,8,11,14,9,12,5)) # numbers in order
```

# LET'S SIMULATE DART THROWS

## Plotting the Board

We'll write a `drawBoard` function with the following features:

- ▶ Inputs: `board` a list containing dart board measurements.
- ▶ Outputs: None, but it will plot the board. Then we can plot on top of the board with functions like `points()` or `lines()`.

This function on Courseworks and it essentially does the following:

- ▶ Use  $(0, 0)$  as the center of the board.
- ▶ Plot concentric circles centered at  $(0, 0)$  to show the rings.
- ▶ Plot the 'spokes'.
- ▶ Add the numbers around the outside.

# LET'S SIMULATE DART THROWS

```
drawBoard = function(board) {  
  R1 = board$R1; R2 = board$R2; R3 = board$R3; R4 = board$R4;  
  R5 = board$R5; R = board$R; nums = board$nums  
  
  mar.orig = par()$mar  
  par(mar = c(0, 0, 0, 0))  
  plot(c(), c(), axes = FALSE, xlim = c(-R - 15, R + 15),  
       ylim = c(-R - 15, R + 15))  
  t = seq(0, 2 * pi, length = 5000)  
  x = cos(t); y = sin(t)  
  points(R * x, R * y, type = "l")  
  points(R5 * x, R5 * y, type = "l")  
  points(R4 * x, R4 * y, type = "l")  
  points(R3 * x, R3 * y, type = "l")  
  points(R2 * x, R2 * y, type = "l")  
  points(R1 * x, R1 * y, type = "l")  
  t0 = pi/2 + 2 * pi/40  
  points(c(R2 * cos(t0), R * cos(t0)),  
         c(R2 * sin(t0), R * sin(t0)), type = "l")  
  for (i in 1:19) {
```

# LET'S SIMULATE DART THROWS

## Simulating the Scoring

We write a `scorePositions()` function with the following features.

- ▶ Inputs:
  - ▶ `x`: vector, horizontal positions of dart throws in mm, with the center of the board being 0
  - ▶ `y`: vector, vertical positions of dart throws in mm, with the center of the board being 0
  - ▶ `board`: list, containing dart board measurements
- ▶ Outputs: vector of scores, same length as `x` (and as `y`)

This function is on Courseworks and it essentially does the following:

- ▶ Calculate the radius at which the throw lands.
- ▶ Calculate the angle at which the throw lands.
- ▶ Use these to calculate the score.

# LET'S SIMULATE DART THROWS

```
scorePositions = function(x, y, board) {  
  R1 = board$R1; R2 = board$R2; R3 = board$R3; R4 = board$R4  
  R5 = board$R5; R = board$R; nums = board$nums  
  
  n = length(x)  
  rad = sqrt(x^2 + y^2)  
  raw.angles = atan2(x,y)  
  slice = 2*pi/20  
  tilted.angles = (raw.angles + slice/2) %% (2*pi)  
  scores = nums[floor(tilted.angles/slice) + 1]  
  
  # Bullseyes  
  scores[rad <= R1] = 50  
  scores[R1 < rad & rad <= R2] = 25  
  
  # Triples  
  scores[R3 < rad & rad <= R4] = 3*scores[R3 < rad & rad <= R4]  
  
  # Doubles  
  scores[R5 < rad & rad <= R] = 2*scores[R5 < rad & rad <= R]}
```

# LET'S SIMULATE DART THROWS

## Our Task

Let  $x$ ,  $y$  denote the horizontal and vertical positions of the throws, measured from the center of the board which is  $(0, 0)$ . Let's consider the following model,  $x$  and  $y$  are both  $\mathcal{N}(0, 50^2)$  (the standard deviation here is 50mm). We will:

- ▶ Generate 100 throws from this model.
- ▶ Plot the throw positions on the dart board, with  $\text{pch} = 20$  (to make small solid dots).
- ▶ Compute the scores of the 100 throws and add the scores corresponding to each throw on the plot.

# LET'S SIMULATE DART THROWS

## Our Solution

```
# Simulate 100 throws with the x and y
# location modeled by N(0, 50^2).

throws <- 100
std.dev <- 50

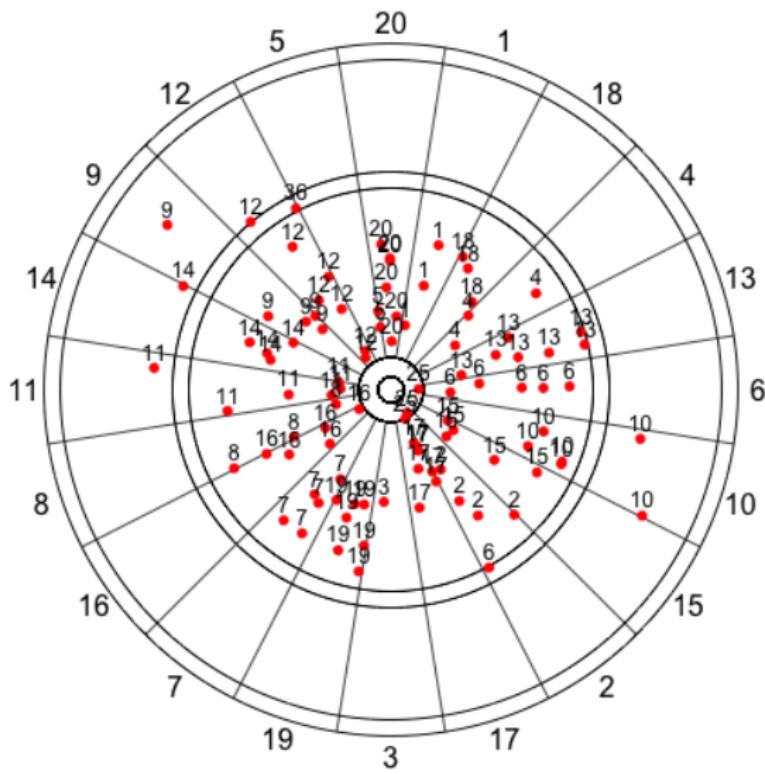
x <- rnorm(throws, sd = std.dev)
y <- rnorm(throws, sd = std.dev)

drawBoard(board)
points(x, y, pch = 20, col = "red")

# Score the throws and add the values to the plot

scores <- scorePositions(x, y, board)
text(x, y + 8, scores, cex = .75)
```

## LET'S SIMULATE DART THROWS



# LET'S SIMULATE DART THROWS

## Our Task

Let  $x, y$  denote the horizontal and vertical positions of the throws, measured from the center of the board which is  $(0, 0)$ . Consider the following model,  $x$  and  $y$  are both  $\text{Uniform}(-R, R)$  where  $R = 170$  is the radius of the board. Note this is the smallest rectangle that contains the dart board. Do the following:

- ▶ Generate 100 throws from this model.
- ▶ Plot the throw positions on the dart board, with  $\text{pch} = 20$  (to make small solid dots).
- ▶ Compute the scores of the 100 throws and add the scores corresponding to each throw on the plot.

# CHECK YOURSELF

## Solution

```
# Simulate 100 throws with the x and y
# location modeled by Uniform(-R, R).

throws  <- 100
R        <- 170

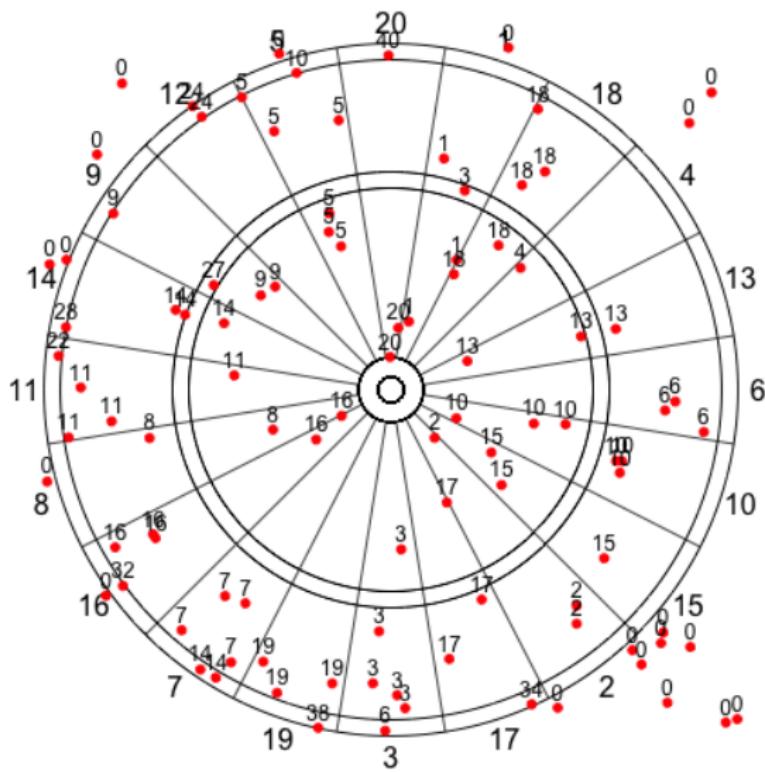
x <- runif(throws, min = -R, max = R)
y <- runif(throws, min = -R, max = R)

drawBoard(board)
points(x, y, pch = 20, col = "red")

# We score our throws and add the values to the plot

scores <- scorePositions(x, y, board)
text(x, y + 8, scores, cex = .75)
```

## LET'S SIMULATE DART THROWS



# WHICH STRATEGY IS BETTER?

## Our Task

Now let's simulate 10,000 dart throws according to the two possible models:

1.  $x$  and  $y$  are both  $\mathcal{N}(0, 50^2)$ .
2.  $x$  and  $y$  are both Uniform( $-R, R$ ).

What are the average scores under each model, and which is higher?

```
throws <- 10000
x1 <- rnorm(throws, sd = std.dev)
y1 <- rnorm(throws, sd = std.dev)
x2 <- runif(throws, min = -R, max = R)
y2 <- runif(throws, min = -R, max = R)

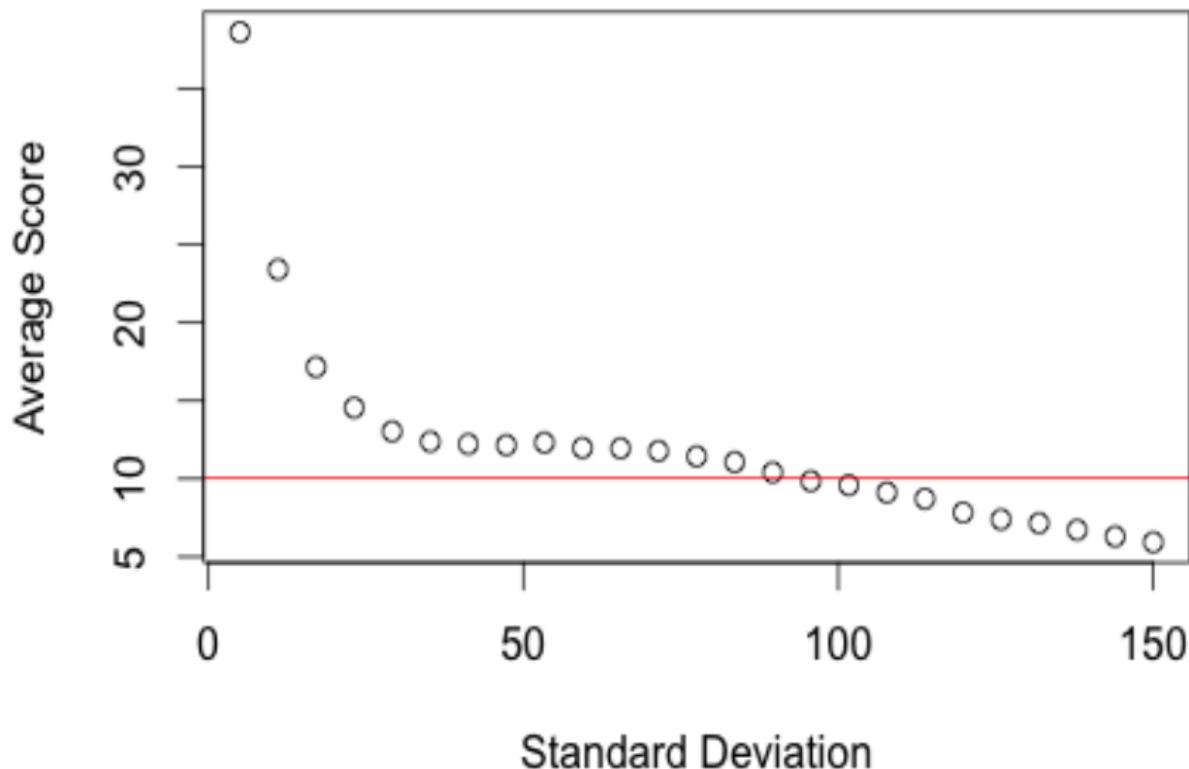
scores1 <- scorePositions(x1, y1, board)
scores2 <- scorePositions(x2, y2, board)
mean(scores1)
mean(scores2)
```

# CHECK YOURSELF

## Tasks

- ▶ For 25 values of standard deviation (sd) between 5 mm and 150 mm, draw 10,000 throws from model (1). For each value of sd, compute the average score.
- ▶ Make a plot showing the average score as a function of sd. Label the axes appropriately, and draw the average score calculated for the uniform model (2) as a horizontal line, in red.
- ▶ At what value of sd does it become better to throw uniformly at the board?

# LET'S SIMULATE DART THROWS



# CHECK YOURSELF

Consider fixed the value  $sd = 35$  mm, and consider a normal model for throws where  $x$  is  $\mathcal{N}(\text{mean.x}, 35^2)$  and  $y$  is  $\mathcal{N}(\text{mean.y}, 35^2)$ , with

1.  $(\text{mean.x}, \text{mean.y}) = (0, 0)$
2.  $(\text{mean.x}, \text{mean.y}) = (0, 103)$
3.  $(\text{mean.x}, \text{mean.y}) = (-32, -98)$

## Tasks

- ▶ Either by plotting or by looking at scores, determine where the throws are being aimed in each of the three models. That is, the choice of  $(\text{mean.x}, \text{mean.y}) = (0, 0)$  in model (1) means that we are aiming at the center of the board. Where are we aiming in models (2) and (3)?
- ▶ For each of the models, draw 10,000 throws, and compute the average score. How do they compare? Can you explain the results from an intuitive perspective?

# EXAM NOTES

- ▶ Exam is next week – on your computer. (i.e. bring a charged computer with R and RMarkdown).
- ▶ You will edit a Markdown file I provide and turn it in via Courseworks before time is up.
- ▶ You can not communicate with any living, breathing person while you are taking the exam.
- ▶ You can use any class materials (have them downloaded on your machine), but not the internet.
- ▶ We will have a seating chart (some of you will be in Havemeyer 309).
- ▶ The exam will be 1.5 hours. You will not have time to look up everything in your notes.
- ▶ When the time is up, you must close your computer or we will take off points. If you haven't submitted to Courseworks yet, you will have to stay afterwards to do this one at a time with the TA.