

Introduction to R Programming

Jorge Andrade, PhD and Wenjun Kang, MS

Center for Research Informatics
University of Chicago

August 30, 2013

Outline

- 1 Introduction
 - Where to find R
 - Why use R
- 2 Basic R Object: Vectors
 - Create Vectors
 - Vector operations
 - Plot vectors
- 3 Basic statistics
 - Mean, median, histogram and boxplot
 - Variance, and standard deviation
- 4 Basic R programming
 - For, while, and repeat loop
 - Data exploration
 - Data transformation and model fitting

R Websites

- ▶ CRAN: <http://cran.r-project.org/>
 - ▶ Manuals: <http://cran.r-project.org/manuals.html>
 - ▶ FAQs: <http://cran.r-project.org/faqs.html>
 - ▶ Contributed Guides: <http://cran.r-project.org/other-docs.html>
- ▶ R Home: <http://www.r-project.org/>
 - ▶ R Wiki: <http://wiki.r-project.org/>
 - ▶ R Journal: <http://journal.r-project.org/>
 - ▶ Mailing Lists: <http://www.r-project.org/mail.html>
 - ▶ Bioconductor: <http://www.bioconductor.org/>

Ultra-short R introduction

Most life scientists use spreadsheet programs (like excel for data analysis) Why?

Ease to use

- ▶ Click buttons, select data by hand
- ▶ You see the data in front of you
- ▶ You can do limited programming

Disadvantages of spreadsheet

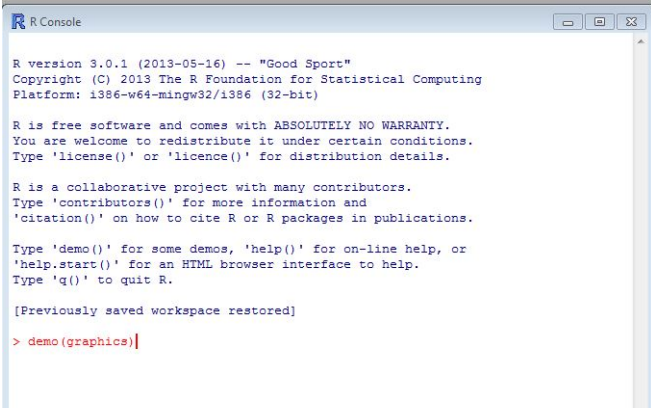
- ▶ Hard to handle large dataset (>1000 data points)
- ▶ Inflexible, few analyses available
- ▶ Hard to repeat analyses systematically with new data

R Advantages

- ▶ R is a computational environment - somewhere between a program and a programming language
- ▶ No buttons, no wizards: only a command line interface
- ▶ Is a professional statistics toolset - likely the only analyses tool you will ever need
- ▶ Is also a programming language
- ▶ Can handle large datasets
- ▶ Very powerful graphics
- ▶ State-of-the-art statistics program for bioinformatics
- ▶ Free, and open source!

First challenge

- ▶ Start R
- ▶ Type: `demo(graphics)`
- ▶ Hit enter a few times



```
R Console

R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> demo(graphics)|
```

Getting help

Most R functions have online documentation.

- ▶ `help(topic)` documentation on topic
> `help(lm)`
- ▶ `?topic`
> `?plot`
- ▶ `help.search("topic")` search the help system
> `help.search("aov")`
- ▶ `apropos("topic")` the names of all objects in the search list matching the regular expression "topic"
> `apropos("plot")`

Vectors

Most R functions have online documentation.

- ▶ The basic Object in R is a vector
- ▶ In statistics, we are almost always dealing with several "data points"
- ▶ A vector is an collection of numbers and/or strings:
("jorge", "wenjun", "ron")
(10, 5.2, 1, 7, 2, 21)
(3)
- ▶ The last example is a vector of length 1

In R, we make a vector by the `c()` command (for concatenate)

```
> c(1,5,10, 7, 2, 1)
```

```
[1] 1 5 10 7 2 1
```

```
> c("jorge", "wenjun", "ron")
```

```
[1] "jorge" "wenjun" "ron"
```

When input strings or characters, we have to surround them with `"` or `'`
If making vectors of size 1, we can skip `c()`

```
> 3
```

```
[1] 3
```

```
> ls()      # List the contents of the workspace.
```

```
character(0)
```

```
> rm(list=ls()) # This completely clears the workspace.
```

```
> ls()      #character(0) means "nothing to see here"
```

```
character(0)
```

Challenge:

- ▶ Make the following vector
45,5,12,10
- ▶ What happens with the following commands?
c(1:100)
c(50:2)

A vector is a data structure, and the most fundamental in R. Almost everything in R is some kind of vector, although sometimes in several dimensions - vectors within vectors.

Reference sheet is your friend!

- ▶ You will get overwhelmed by different command names fast
- ▶ Use the reference sheet to remind yourself in all exercises

Assignment to memory

- ▶ The `c()` command is almost useless in itself - we want to keep the vector for other analyses
- ▶ The assignment concept:

```
> 4+5          # add 4 and 5
[1] 9

> a <- 4       # store 4 as "a"
> b <- 5       # store 5 as "b"
> a           # just checking
[1] 4

> b
[1] 5

> a+b         # add a+b (4+5)
[1] 9
```

Expanding this to a whole vector:

```
> my_vector <- c(1,5,10, 7, 2)
> my_vector

[1] 1 5 10 7 2
```

Note that there is no "return value" now - this is caught by the "my_vector". my_vector is a variable, with the variable name: my_vector. Variable names are totally arbitrary! The anatomy of the vector:

Name	my_vector				
Values	1	5	10	7	2
Index	[1]	[2]	[3]	[4]	[5]

We can access part of the vector like this:

- ▶ `> my_vector[5]` will give the 5th item in the vector
- ▶ What happens if you do this?

```
> my_vector <- c(1,5,10, 7, 2) # define the vector  
> my_vector [c(1,3,5)]  
> my_vector[1:4]  
> my_vector[4:1]
```

We can access part of the vector like this:

- ▶ `> my_vector[5]` will give the 5th item in the vector
- ▶ What happens if you do this?

```
> my_vector<- c(1,5,10, 7, 2) # define the vector  
> my_vector [c(1,3,5)]  
> my_vector[1:4]  
> my_vector[4:1]
```

```
> my_vector<- c(1,5,10, 7, 2)  
> my_vector [c(1,3,5)]  
[1] 1 10 2  
> my_vector[1:4]  
[1] 1 5 10 7  
> my_vector[4:1]  
[1] 7 10 5 1
```


Challenge

Using the reference sheet, figure out at least three ways of making R print your vector in the other direction

```
> my_vector<- c(1,5,10, 7, 2)  # define the vector
```

Challenge

Using the reference sheet, figure out at least three ways of making R print your vector in the other direction

```
> my_vector<- c(1,5,10, 7, 2) # define the vector
```

```
> my_vector[5:1]
```

```
[1] 2 7 10 5 1
```

```
> my_vector[c(5,4,3,2,1)]
```

```
[1] 2 7 10 5 1
```

```
> c<- c(my_vector[5],my_vector[4],my_vector[3],  
+       my_vector[2], my_vector[1])  
> rev(my_vector)
```

```
[1] 2 7 10 5 1
```

Naming rules and the danger of over-writing

Naming: We can name vectors to almost anything. The most basic rule is : Never start a vector name with a number

- ▶ `> a<- c(1,5,4,2) #OK`
- ▶ `> 1a<- c(1,5,4,2) # NOT OK Error: syntax error`
- ▶ `> a1<- c(1,5,4,2) # OK`

Over-writing:

- ▶ `> my_vector<- c(1,5,10, 7, 2)`
- ▶ `> my_vector<- c(10,5,2, 3, 1)`

`#what does my_vector contain now?`

Analyzing vectors

- ▶ Many functions work directly on vectors - most have logical names. For instance, `length(my_vector)` gives the number of items in the vector (= 5)
- ▶ Challenge: make a vector called `big_vector` with values 1 to 10000, find
 - ▶ Length of the vector
 - ▶ Sum of all items in vector: the `sum()` function
 - ▶ Mean(average) of all items in the vector: the `mean()` function

Analyzing vectors

- ▶ Many functions work directly on vectors - most have logical names. For instance, `length(my_vector)` gives the number of items in the vector (= 5)
- ▶ Challenge: make a vector called `big_vector` with values 1 to 10000, find
 - ▶ Length of the vector
 - ▶ Sum of all items in vector: the `sum()` function
 - ▶ Mean(average) of all items in the vector: the `mean()` function

```
> big_vector<-(1:10000); length(big_vector)
```

```
[1] 10000
```

```
> sum(big_vector)
```

```
[1] 50005000
```

```
> mean(big_vector)
```

```
[1] 5000.5
```

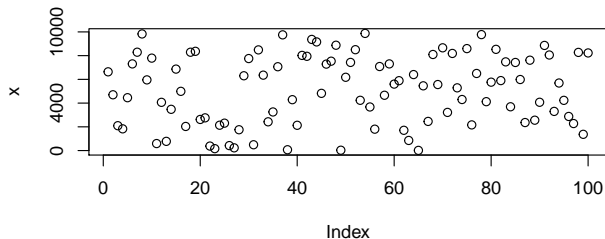
Challenge: help system

- ▶ Look at the help for `sample()` and `sort()` and then try them out on `big_vector`

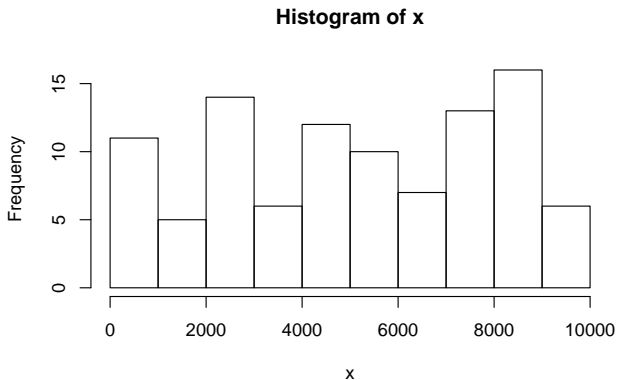
```
> x <-sample(big_vector,100)
> x[1:20]

[1] 6636 4702 2096 1831 4456 7304 8290 9829 5957 7791 591
[16] 4992 2032 8289 8365 2622

> plot(x)
```



```
> sort(x, decreasing = FALSE) [1:20]
 [1] 16 29 69 161 230 379 419 490 591 777 853
[16] 1831 2032 2096 2131 2137
> hist(x)
```



Adding and multiplying a number to a vector Sometimes we want to add a number, like 10, to each element in the vector:

```
> big_vector + 10
```

Test this:

```
big_vector2<-big_vector +10
```

Also test

```
min(big_vector)
```

```
max(big_vector)
```

```
min(big_vector2)
```

```
max(big_vector2)
```

What happens?

```
> big_vector2<-big_vector +10  
> min(big_vector)  
[1] 1  
  
> max(big_vector)  
[1] 10000  
  
> min(big_vector2)  
[1] 11  
  
> max(big_vector2)  
[1] 10010
```

Adding vectors

- ▶ We can also add one vector to another vector
- ▶ Say that we have the three vectors

```
A<-c(10, 20, 30, 50)
```

```
B<-c(1,4,2,3)
```

```
C<-c(2.5, 3.5)
```

- ▶ Test what happens and explain the outcome:

```
A+B
```

```
A+C
```

Adding vectors

```
> A<-c(10 , 20, 30, 50)
```

```
> B<-c(1 , 4, 2, 3)
```

```
> C<-c(2.5, 3.5 )
```

```
> A+B
```

```
[1] 11 24 32 53
```

```
> A+C
```

```
[1] 12.5 23.5 32.5 53.5
```

A+B is easy to understand : $A[1]+B[1]$, etc.

A+C is trickier - the C vector is just of length 2. It is re-used! So

$A[1]+C[1]=12.5$

$A[2]+C[2]=23.5$

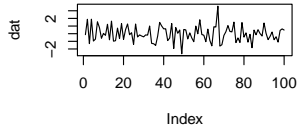
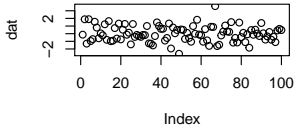
$A[3]+C[1]=32.5$

$A[4]+C[2]=53.5$. Actually, this is what is happening also with

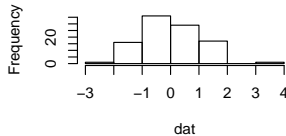
$A+10$. The 10 is used many times.

Plotting vectors

- ▶ Lets make up some semi-random data:
`dat<-rnorm (100)` draw 100 random, normal distributed data points
- ▶ Test the following:
`plot(dat)`
`plot(dat,type='l')`
`barplot(dat)`
`hist(dat)`
- ▶ What is the difference?



Histogram of dat



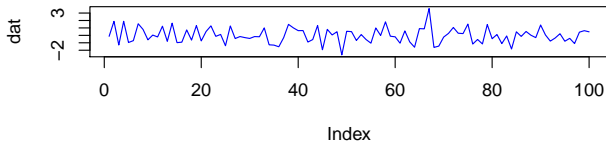
Why are your three first plots different from mine? Why is your last plot more similar to mine?

Graph options

- ▶ Generally, you can give extra options to graphical commands like this
- ▶ `> plot(dat, col='blue', type='l')`
- ▶ In `plot`: try to vary the following options - note that you can use several at once (and figure out what they do)
 - `type='b'`
 - `col='hotpink'`
 - `main='plot'`
 - `type='h'`
 - `type='S'`

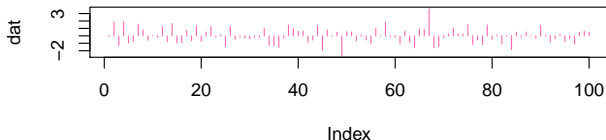
These options are really arguments to the `plot()` function

```
> plot(dat, col='blue', type='l')
```



```
> plot(dat, col='hotpink', type='h', main='Plot',)
```

Plot



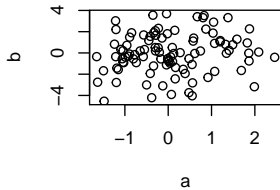
More about functions

- ▶ In almost all cases, a function needs some input, like `plot(dat)`.
- ▶ 'dat' here is an unnamed argument, and this works because `plot()` assumes we mean `x values = dat`.
- ▶ We could also say `plot(x=dat)` - a named argument. If you have many arguments, most of them are named - such as `plot(some_vector, col="blue", type="s")`
- ▶ The help pages will tell you what type of arguments you can use

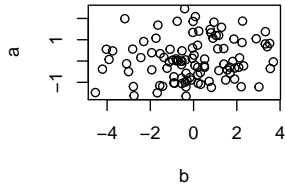
The danger of unnamed arguments.

- ▶ ... is that the order of them will make a big difference. Try this out - what is the difference between the plot commands?
 - > a<-rnorm(100)
 - > b<-rnorm(100)*2
 - > plot(a,b)
 - > plot(b,a)
 - > plot(x=b, y=a)

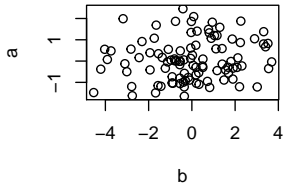
plot(a,b)



plot(b,a)



plot(x=b,y=a)

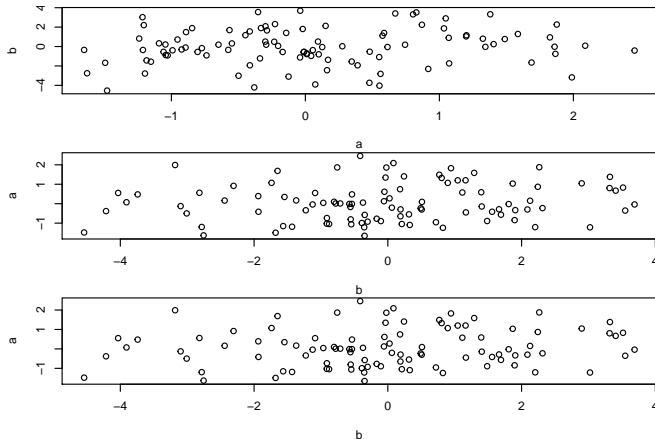


Some generic R arguments to plots - the `par()` function

- ▶ The `par()` function is used to set general plot properties. It has hundreds of possible arguments - see: `?par`
- ▶ Two very handy `par()` arguments is `mfrow()` and `mfcop()` - these will allow many plots in one page
- ▶ You give these functions a vector of length 2 - this gives the number of cells in the page (see example)

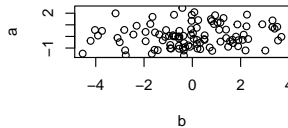
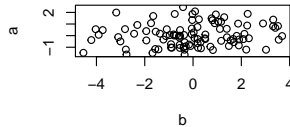
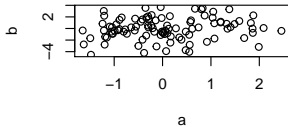
Example:

```
> par( mfrow=c(3,1) )  
> plot(a,b); plot(b,a); plot(x=b, y=a)
```



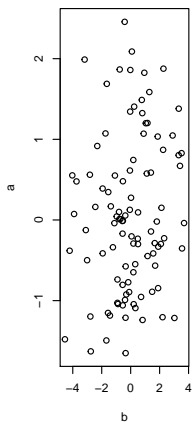
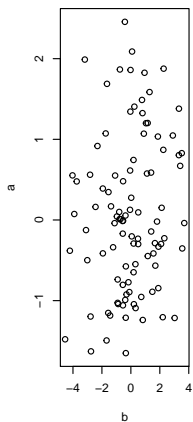
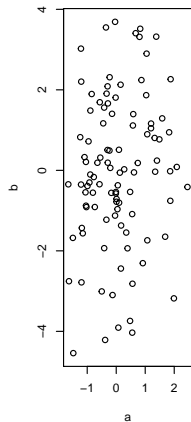
Example:

```
> par( mfrow=c(2,2) )  
> plot(a,b); plot(b,a); plot(x=b, y=a)
```



Challenge - can you get the three plots in a row using mfrow?

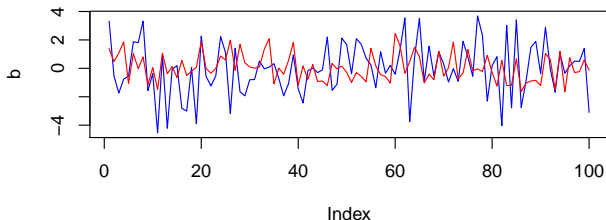
```
> par( mfrow=c(1,3) )  
> plot(a,b); plot(b,a); plot(x=b, y=a)
```



Overlying plots

- ▶ Sometimes we want to put many data sets within one graph, on top of each other
- ▶ This is often made by the `lines()` or `points()` command, like this:

```
> plot(b, type="l", col="blue")  
> lines(a, col="red")
```



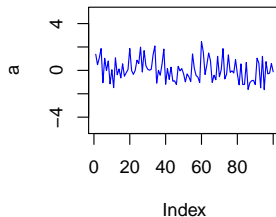
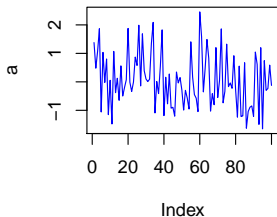
Why did I start with plotting b?

What would have happened if using points() instead of lines()?

Sizing graphs

- ▶ Simple concept, but awkward to write
- ▶ Change X scale: `xlim=c(start_value, end_value)`
- ▶ Change Y scale: `ylim=c(start_value, end_value)`

```
> par(mfrow=c(1,2))  
> plot(a, type="l", col="blue")  
> plot(a, type="l", col="blue", ylim=c(-5,5))
```



Saving graphs

- ▶ Different on different systems!
- ▶ All systems can use the `device()` function - see `?device`
 - > Saving a chart on a .pdf file
 - > `pdf('plot.pdf')`
 - > `plot(a,b)`
 - > `dev.off()`

 - > Saving a chart on a .jpg file
 - > `jpeg('rplot.jpg')`
 - > `plot(a,b)`
 - > `dev.off()`

Saving graphs

- ▶ Windows: Right-click on graph, copy as metafile or bitmap, paste.
- ▶ OSX: Click on the graph, and just copy it. Will become a pdf or a bitmap when pasting.

Some statistics:

Summary statistics

- ▶ `hist()` (= Histogram) is a graphical way of summarizing distributions - it creates a number of "bins" and calculates how many of the data points fall into each bin.
- ▶ We can also summarize by the center points in the data:
 - ▶ `mean()`:

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i = \frac{1}{n} (X_1 + X_2 + \dots + X_n)$$

- ▶ `median()`: Sort all the data, pick the number in the center. If the number of data points is even, take the mean of the two center points

Challenge:

- ▶ We make another vector

```
> dat2<-rnorm(10)
```

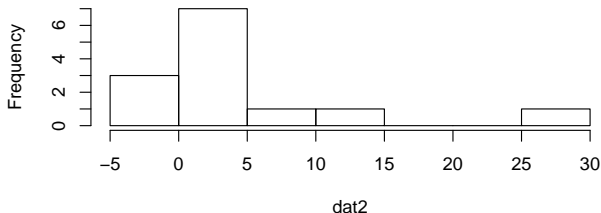
- ▶ And add a few extra points to it

```
> dat2<-c(dat2, 10, 10.5, 30 )
```

- ▶ Test `mean()` and `median()` on `dat2`. Are they the same? Can you explain the differences by plotting a histogram? What is the advantage/disadvantage of each measure?

```
> dat<-rnorm(10)
> dat2<-c(dat, 10, 10.5, 30 )
> median(dat2)
[1] 1.290037
> mean(dat2)
[1] 4.398899
> hist(dat2)
```

Histogram of dat2

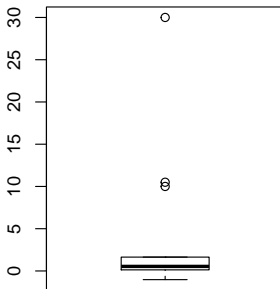
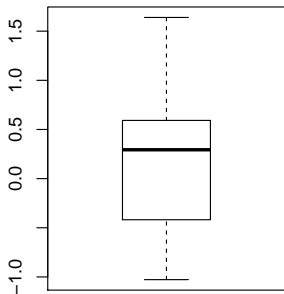


Means are sensitive to outliers! Very common situation in genomics.

> `boxplot(dat2)` is a better way to visualize outliers

Challenge: Boxplot 2 vector with and without outliers and compare

```
> dat <- rnorm(10)
> dat2<-c(dat, 10, 10.5, 30 )
> par( mfrow=c(1,2) )
> boxplot(dat); boxplot(dat2)
```



Percentiles

- ▶ An extension of the median concept
- ▶ Best explained by example:
 - ▶ the 20th percentile is the value (or score) below which 20 percent of the observations may be found.
- ▶ The median is the same as the 50th percentile
- ▶ The first quartile is the 25th percentile, the third is the 75th
- ▶ Try `summary(dat)` and `summary(dat2)`

```
> summary(dat)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-1.0280	-0.2814	0.2936	0.2759	0.5786	1.6390

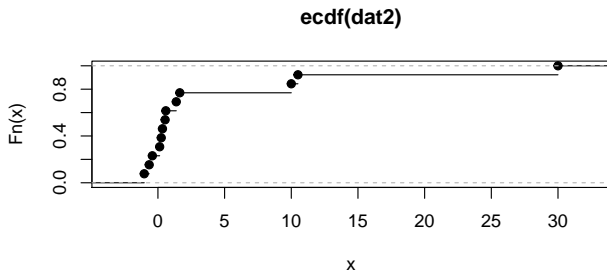
```
> summary(dat2)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-1.0280	0.1292	0.5374	4.0970	1.6390	30.0000

The command `ecdf()` (empirical cumulative distribution) calculates "all" percentiles in your data - and also understands `plot()` Try:

```
> plot(ecdf(dat2))
```

```
> plot (ecdf(dat2))
```

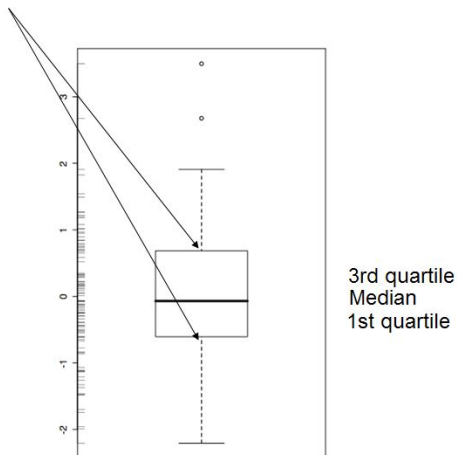


What fraction of the data that has been covered at point X?

Boxplots

- ▶ As we have seen, an "easier" representation of ECDFs. Is based on making boxes that tell us about both center point and "spread" of the data
- ▶ First, calculate the first quartile, the median and the third quartile
- ▶ Calculate the "inter-quartile range" (IQR): 3rd quartile -1st quartile
- ▶ These will be used to draw a "box"
 - > `boxplot(dat)`
 - > `rug(dat, side=2)`

Interquartile range (IQR)

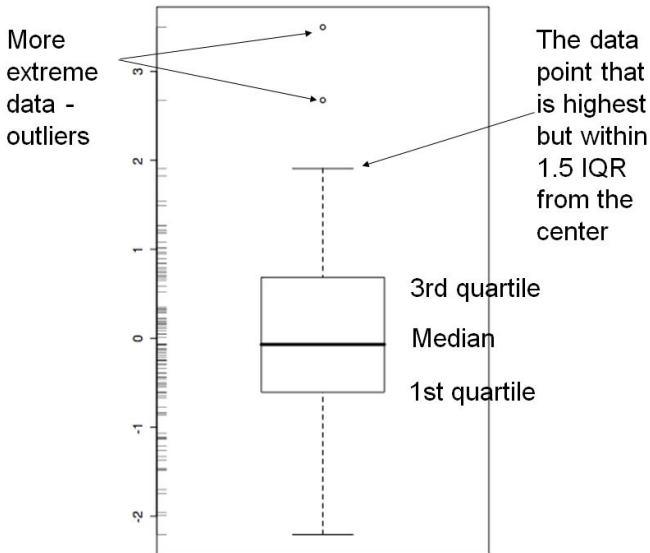


... continued

- ▶ Sounds more complicated than it is: Any data observation which lies more than $1.5 \cdot \text{IQR}$ lower than the first quartile is considered an outlier.

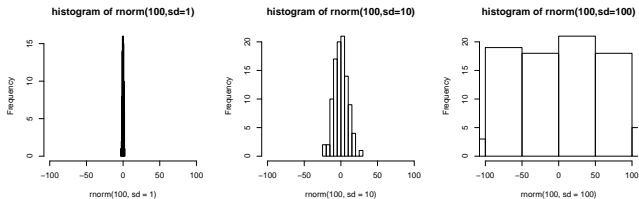
Indicate where the smallest value that is not an outlier is by a vertical tic mark or "whisker", and connect the whisker to the box via a horizontal line.

Do the same for higher values



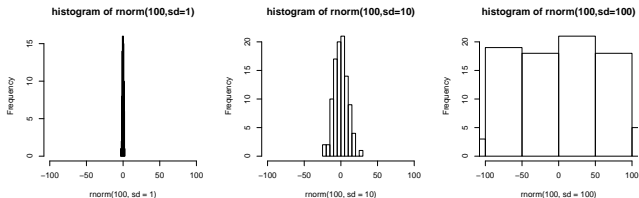
Variance, standard deviation and data spread

What is the difference between these distributions?



Variance, standard deviation and data spread

What is the difference between these distributions?



- ▶ Same mean and median, but different spread over the x axis
- ▶ This can be measured by the variance of the data:

$$\frac{1}{N-1} \sum_{i=1}^N (X_i - \bar{X})^2$$

- ▶ It is basically the difference between each point and the mean, squared

Variance, standard deviation and data spread

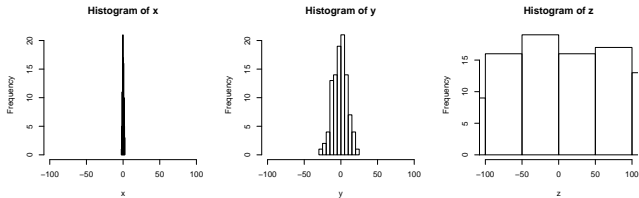
- ▶ Sample Standard deviation is simply variance squared.

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (X_i - \bar{X})^2}$$

- ▶ This gives nice statistical features

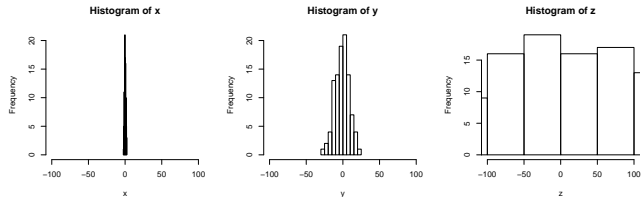
Challenge:

Produce the R code to generate the plot below:



Challenge:

Produce the R code to generate the plot below:



- ```
> x<-rnorm(100, sd=1)
> y<-rnorm(100, sd=10)
> z<-rnorm(100, sd=100)
> par(mfrow=c(1,3))
> hist(x, xlim=c(-100, 100)); hist(y, xlim=c(-100, 100))
> hist(z, xlim=c(-100, 100))
```



## Why is variance and standard deviation important?

- ▶ Variance tells you something about the quality of measurements
- ▶ The higher the variance, the harder it is to say with certainty that two measurements are different

## Why is variance and standard deviation important?

- ▶ What are the R functions for variance and standard deviation?  
Let's make some random data

```
> smallset<-rnorm(100)
```

```
> largeset<-rnorm(10000)
```

- ▶ What is the variance and standard deviation for these?
- ▶ Is the standard deviation really the square root of the variance (what is the function for square root?)

```
> smallset<-rnorm(100); largeset<-rnorm(10000)
> var(smallset)
[1] 0.7188849
> var(largeset)
[1] 0.9849719
> sd(largeset)
[1] 0.9924575
> sd(smallset)
[1] 0.8478708
> sqrt(var(smallset))
[1] 0.8478708
```

Why do we get about the same variance?

```
> ?rnorm
```

## Basic R programming

### User defined function

```
> findSumSquare <- function(a,b) {
+ return(a^2+b^2)
+ }
> a=3
> b=5
> findSumSquare(a,b)

[1] 34

> findSumSquare(1,2)

[1] 5
```

## For loop

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

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

## While loop

```
> i = 0
> while (i<10) {
+ i=i+1
+ print(i)
+ }
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

## Repeat until the break condition met

```
> repeat{
+ a=runif(1)
+ print(a)
+ if (a > 0.8) break
+ }
```

```
[1] 0.3077661
```

```
[1] 0.2576725
```

```
[1] 0.5523224
```

```
[1] 0.05638315
```

```
[1] 0.4685493
```

```
[1] 0.4837707
```

```
[1] 0.8124026
```

```
> mydata <- read.csv(
+ file="http://www.ats.ucla.edu/stat/data/binary.csv",
+ header=T)
> head(mydata, 4)
```

```
 admit gre gpa rank
1 0 380 3.61 3
2 1 660 3.67 3
3 1 800 4.00 1
4 1 640 3.19 4
```

```
> summary(mydata[,2:3])
```

```
 gre gpa
Min. :220.0 Min. :2.260
1st Qu.:520.0 1st Qu.:3.130
Median :580.0 Median :3.395
Mean :587.7 Mean :3.390
3rd Qu.:660.0 3rd Qu.:3.670
Max. :800.0 Max. :4.000
```



## Names of the variable in the dataset

```
> names(mydata)
```

```
[1] "admit" "gre" "gpa" "rank"
```

## Number of rows and columns in the dataset

```
> dim(mydata)
```

```
[1] 400 4
```

## Cross tab between admit and rank

```
> xtabs(~admit+rank, data=mydata)
```

```
 rank
admit 1 2 3 4
 0 28 97 93 55
 1 33 54 28 12
```

## Add a new column: id

```
> mydata$id=1:400
> head(mydata, 5)
```

|   | admit | gre | gpa  | rank | id |
|---|-------|-----|------|------|----|
| 1 | 0     | 380 | 3.61 | 3    | 1  |
| 2 | 1     | 660 | 3.67 | 3    | 2  |
| 3 | 1     | 800 | 4.00 | 1    | 3  |
| 4 | 1     | 640 | 3.19 | 4    | 4  |
| 5 | 0     | 520 | 2.93 | 4    | 5  |

## Subset the dataset

### 1. By index

```
> mydata[1:10, 2:3]
```

```
 gre gpa
1 380 3.61
2 660 3.67
3 800 4.00
4 640 3.19
5 520 2.93
6 760 3.00
7 560 2.98
8 400 3.08
9 540 3.39
10 700 3.92
```

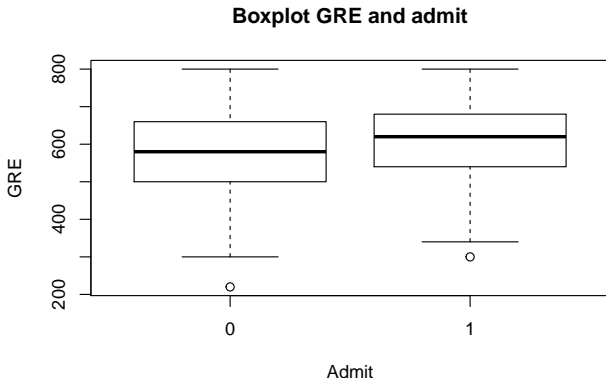
## 2. By conditions

```
> mydata[mydata$id>=10 & mydata$id<=20,]
```

|    | admit | gre | gpa  | rank | id |
|----|-------|-----|------|------|----|
| 10 | 0     | 700 | 3.92 | 2    | 10 |
| 11 | 0     | 800 | 4.00 | 4    | 11 |
| 12 | 0     | 440 | 3.22 | 1    | 12 |
| 13 | 1     | 760 | 4.00 | 1    | 13 |
| 14 | 0     | 700 | 3.08 | 2    | 14 |
| 15 | 1     | 700 | 4.00 | 1    | 15 |
| 16 | 0     | 480 | 3.44 | 3    | 16 |
| 17 | 0     | 780 | 3.87 | 4    | 17 |
| 18 | 0     | 360 | 2.56 | 3    | 18 |
| 19 | 0     | 800 | 3.75 | 2    | 19 |
| 20 | 1     | 540 | 3.81 | 1    | 20 |

## Boxplot for GRE and admit

```
> boxplot(gre~admit, data=mydata, xlab="Admit",
+ ylab="GRE", main="Boxplot GRE and admit")
```



## T-test for gre scores for admit=0 vs admit=1

```
> t.test(gre~admit,data=mydata)
```

Welch Two Sample t-test

```
data: gre by admit
```

```
t = -3.8292, df = 260.181, p-value = 0.0001611
```

```
alternative hypothesis: true difference in means is not equal to
```

```
95 percent confidence interval:
```

```
-69.21683 -22.20482
```

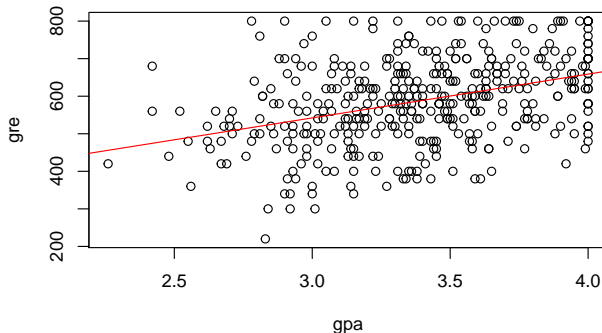
```
sample estimates:
```

```
mean in group 0 mean in group 1
```

```
573.1868 618.8976
```

## Simple regression between gre and gpa

```
> fit1 <- lm(gre~gpa, data=mydata)
> plot(gre~gpa, data=mydata)
> abline(fit1,col="red")
```



## Logistic regression

```
> fit <- glm(admit~gpa+gre+factor(rank), data=mydata,
+ family=binomial)
> print(summary(fit)$coef, digits=2)
```

|               | Estimate | Std. Error | z value | Pr(> z ) |
|---------------|----------|------------|---------|----------|
| (Intercept)   | -3.9900  | 1.1400     | -3.5    | 0.00047  |
| gpa           | 0.8040   | 0.3318     | 2.4     | 0.01539  |
| gre           | 0.0023   | 0.0011     | 2.1     | 0.03847  |
| factor(rank)2 | -0.6754  | 0.3165     | -2.1    | 0.03283  |
| factor(rank)3 | -1.3402  | 0.3453     | -3.9    | 0.00010  |
| factor(rank)4 | -1.5515  | 0.4178     | -3.7    | 0.00020  |



## Write dataset to a text file

```
> write.table(mydata, file="test.txt", sep="\t",
+ row.names=FALSE, quote=FALSE)
> list.files(path=getwd(), pattern="test.txt",
+ full.names=T)

[1] "C:/Users/wkang2.CRI/Desktop/Rtraining/test.txt"
```

## CRI Contact

Thank you for your participation in this training. If you have questions, please email us at **bioinformatics@bsd.uchicago.edu**

Please leave your feedback for this training on:

**<https://biocore.cri.uchicago.edu/cgi-bin/survey.cgi?id=7>**