Introduction to R and Programming

Nathaniel E. Helwig

Assistant Professor of Psychology and Statistics
University of Minnesota (Twin Cities)

Updated 04-Jan-2017

Copyright

Outline of Notes

1) Introduction to R:
- Downloading R
- Basic calculations
- Using R functions
- Object classes in R

2) Statistical Distributions in R:
- Overview
- Normal distribution
- Student's *t* distribution
- Common distributions

3) Basic Programming:
- Logical operators
- If/Else statements
- For loops
- While statements

Introduction to R

R = Free and Open-Source Statistics

R is a free and open-source software environment for statistics.

- Created by Ross Ihaka and Robert Gentleman (at the University of Auckland, New Zealand)
- Based on S language created by John Chambers (at Bell Labs)
- Currently managed by The R Project for Statistical Computing http://www.r-project.org/

You can freely download R for various operating systems:

- Mac
- Windows
- Linux

RStudio IDE

RStudio IDE is a free and open-source integrated development environment (IDE) for R.

- Basic R interface is a bit rough (particularly on Windows)
- RStudio offers a nice environment through which you can use R
- Freely available at http://www.rstudio.com/

You can freely download RStudio IDE for various operating systems:

- Mac
- Windows
- Linux

R Console as a Calculator

Addition and Subtraction

```
> 3+2
[1] 5

> 3-2
[1] 1
```

Multiplication and Division

```
> 3*2
[1] 6

> 3/2
[1] 1.5
```

Exponents in R

```
> 3^2
[1] 9

> 2^3
[1] 8
```

Constants in R

```
> pi
[1] 3.141593

> exp(1)
[1] 2.718282
```

Some Special Values in R

Infinite Values

```
> Inf
[1] Inf

> 1+Inf
[1] Inf
```

Machine Epsilon

```
> .Machine$double.eps
[1] 2.220446e-16

> 0>.Machine$double.eps
[1] FALSE
```

Empty Values

```
> NULL
NULL

> 1+NULL
numeric(0)
```

Missing Values

```
> NA
[1] NA

> 1+NA
[1] NA
```

Storing and Manipulating Values in R

Define objects `x` and `y` with values of 3 and 2, respectively:

```
> x=3
> y=2
```

Some calculations with the defined objects `x` and `y`:

```
> x+y
[1] 5

> x*y
[1] 6
```

Warning: R is case sensitve, so `x` and `X` are not the same object.

Function-Based Languages

R is a function-based language, where a "function" takes in some input $\mathfrak{X}_I$ and creates some output $\mathfrak{X}_O$.

Vegas rules: what happens in a function, stays in a function

- Function only knows the input $\mathfrak{X}_I$
- Function only creates the output $\mathfrak{X}_O$

Each R function has a (unique) name, and the general syntax is

$$\mathfrak{X}_O = \text{fname}(\mathfrak{X}_I, \ldots)$$

where fname is the function name, and $\ldots$ denotes additional inputs.

Some Basic R Functions

Combine

```
> c(1,3,-2)
[1]  1  3 -2

> c("a","a","b","b","a")
[1] "a" "a" "b" "b" "a"
```

Sum and Mean

```
> sum(c(1,3,-2))
[1] 2

> mean(c(1,3,-2))
[1] 0.6666667
```

Variance and Std. Dev.

```
> var(c(1,3,-2))
[1] 6.333333

> sd(c(1,3,-2))
[1] 2.516611
```

Minimum and Maximum

```
> min(c(1,3,-2))
[1] -2

> max(c(1,3,-2))
[1] 3
```

Some More R Functions

Define objects x and y:

```
> x=c(1,3,4,6,8)
> y=c(2,3,5,7,9)
```

Calculate the correlation:

```
> cor(x,y)
[1] 0.988765
```

Calculate the covariance:

```
> cov(x,y)
[1] 7.65
```

Combine as columns

```
> cbind(x,y)
     x y
[1,] 1 2
[2,] 3 3
[3,] 4 5
[4,] 6 7
[5,] 8 9
```

Combine as rows

```
> rbind(x,y)
  [,1] [,2] [,3] [,4] [,5]
x    1    3    4    6    8
y    2    3    5    7    9
```

Object-Oriented Style Programming

R is an object-oriented language, where an "object" is a general term.

Any R object $\mathfrak{X}$ has an associated "class", which indicates the type of object that $\mathfrak{X}$ represents.

Some R functions are only defined for a particular class of input $\mathfrak{X}$.

Other R functions perform different operations depending on the class of the input object $\mathfrak{X}$.

Some Basic R Classes

`numeric` class:

```
> x=c(1,3,-2)
> x
[1]  1  3 -2
> class(x)
[1] "numeric"
```

`integer` class:

```
> x=c(1L,3L,-2L)
> x
[1]  1  3 -2
> class(x)
[1] "integer"
```

`character` class:

```
> x=c("a","a","b")
> x
[1] "a" "a" "b"
> class(x)
[1] "character"
```

`factor` class:

```
> x=factor(c("a","a","b"))
> x
[1] a a b
Levels: a b
> class(x)
[1] "factor"
```

Some More R Classes

`matrix` class:

```
> x=c(1,3,-2)
> y=c(2,0,7)
> z=cbind(x,y)
> z
      x y
[1,]  1 2
[2,]  3 0
[3,] -2 7
> class(z)
[1] "matrix"
```

`data.frame` class:

```
> x=c(1,3,-2)
> y=c("a","a","b")
> z=data.frame(x,y)
> z
   x y
1  1 a
2  3 a
3 -2 b
> class(z)
[1] "data.frame"
```

Class-Customized R Functions

Many functions in R are "class-customized", i.e., they execute different code depending the on class of the input object $\mathcal{X}$.

One simple example (that we've already seen) is the `print` function:

```
> x=c(1,3,-2)
> y=factor(c("a","a","b"))
> print(x)
[1]  1  3 -2
> print(y)
[1] a a b
Levels: a b
```

Class-Customized R Functions (continued)

Another simple example is the `summary` function:

```
> x=c(1,3,-2)
> y=factor(c("a","a","b"))
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.0000 -0.5000  1.0000  0.6667  2.0000  3.0000
> summary(y)
a b
2 1
```

Class-Customized R Functions (continued)

Some R functions only work on particular object classes (e.g., `range`):

```
> x=c(1,3,-2)
> y=factor(c("a","a","b"))
> range(x)
[1] -2  3
> range(y)
Error in Summary.factor(c(1L, 1L, 2L), na.rm = FALSE)
  range not meaningful for factors
```

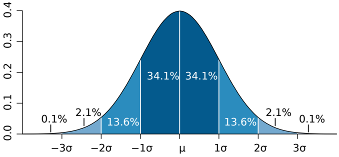
Statistical Distributions in R

Statistical Distributions: Summary

When working with different statistical distributions, we often want to make probabilistic statements based on the distribution.

We typically want to know one of three things:

- The density (pdf) value at a particular value of *x*
- The distribution (cdf) value at a particular value of *x*
- The quantile (*x*) value corresponding to a particular probability

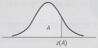


Statistical Distributions: Old School

Statistical tables used to be printed in book appendices:

TABLE B.1 Cumulative Probabilities of the Standard Normal Distribution.

Entry is area A under the standard normal curve from $-\infty$ to $z(A)$



z	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
.0	.5000	.5040	.5080	.5120	.5160	.5199	.5239	.5279	.5319	.5359
.1	.5398	.5438	.5478	.5517	.5557	.5596	.5636	.5675	.5714	.5753
.2	.5793	.5832	.5871	.5910	.5948	.5987	.6026	.6064	.6103	.6141
.3	.6179	.6217	.6255	.6293	.6331	.6368	.6406	.6443	.6480	.6517
.4	.6554	.6591	.6628	.6664	.6700	.6736	.6772	.6808	.6844	.6879
.5	.6915	.6950	.6985	.7019	.7054	.7088	.7123	.7157	.7190	.7224
.6	.7257	.7291	.7324	.7357	.7389	.7422	.7454	.7486	.7517	.7549
.7	.7580	.7611	.7642	.7673	.7704	.7734	.7764	.7794	.7823	.7852
.8	.7881	.7910	.7939	.7967	.7995	.8023	.8051	.8078	.8106	.8133
.9	.8159	.8186	.8212	.8238	.8264	.8289	.8315	.8340	.8365	.8389
1.0	.8413	.8438	.8461	.8485	.8508	.8531	.8554	.8577	.8599	.8621
1.1	.8643	.8665	.8686	.8708	.8729	.8749	.8770	.8790	.8810	.8830
1.2	.8849	.8869	.8888	.8907	.8925	.8944	.8962	.8980	.8997	.9015
1.3	.9032	.9049	.9066	.9082	.9099	.9115	.9131	.9147	.9162	.9177
1.4	.9192	.9207	.9222	.9236	.9251	.9265	.9279	.9292	.9306	.9319
1.5	.9332	.9345	.9357	.9370	.9382	.9394	.9406	.9418	.9429	.9441
1.6	.9452	.9463	.9474	.9484	.9495	.9505	.9515	.9525	.9535	.9545
1.7	.9554	.9564	.9573	.9582	.9591	.9599	.9608	.9616	.9625	.9633
1.8	.9641	.9649	.9656	.9664	.9671	.9678	.9686	.9693	.9699	.9706
1.9	.9713	.9719	.9726	.9732	.9738	.9744	.9750	.9756	.9761	.9767
2.0	.9772	.9778	.9783	.9788	.9793	.9798	.9803	.9808	.9812	.9817
2.1	.9821	.9826	.9830	.9834	.9838	.9842	.9846	.9850	.9854	.9857
2.2	.9861	.9864	.9868	.9871	.9875	.9878	.9881	.9884	.9887	.9890
2.3	.9893	.9896	.9898	.9901	.9904	.9906	.9909	.9911	.9913	.9916
2.4	.9918	.9920	.9922	.9925	.9927	.9929	.9931	.9932	.9934	.9936
2.5	.9938	.9940	.9941	.9943	.9945	.9946	.9948	.9949	.9951	.9952
2.6	.9953	.9955	.9956	.9957	.9959	.9960	.9961	.9962	.9963	.9964
2.7	.9965	.9966	.9967	.9968	.9969	.9970	.9971	.9972	.9973	.9974
2.8	.9974	.9975	.9976	.9977	.9977	.9978	.9979	.9979	.9980	.9981
2.9	.9981	.9982	.9982	.9983	.9984	.9984	.9985	.9985	.9986	.9986
3.0	.9987	.9987	.9987	.9988	.9988	.9989	.9989	.9989	.9990	.9990
3.1	.9990	.9991	.9991	.9991	.9992	.9992	.9992	.9992	.9993	.9993
3.2	.9993	.9993	.9994	.9994	.9994	.9994	.9994	.9995	.9995	.9995
3.3	.9995	.9995	.9995	.9996	.9996	.9996	.9996	.9996	.9996	.9997
3.4	.9997	.9997	.9997	.9997	.9997	.9997	.9997	.9997	.9997	.9998

Selected Percentiles

Cumulative probability A:	.90	.95	.975	.98	.99	.995	.999
$z(A)$:	1.282	1.645	1.960	2.054	2.326	2.576	3.090

Statistical Distributions: R Functions

R has functions for obtaining density, distribution, and quantile values.

The general naming structure of the relevant R functions is...

- `dname` calculates density (pdf) value at input quantile
- `pname` calculates distribution (cdf) value at input quantile
- `qname` calculates quantile value at input probability
- `rname` generates random sample of input size

Note that `name` represents the name of the given distribution.

Normal Distribution: Overview

The relevant functions for the normal distribution are...

- `dnorm` calculates density (pdf) value at input quantile
- `pnorm` calculates distribution (cdf) value at input quantile
- `qnorm` calculates quantile value at input probability
- `rnorm` generates random sample of input size

In addition to the input quantile (or probability or size) value, you can input the `mean` and `sd` (standard deviation) of the variable.

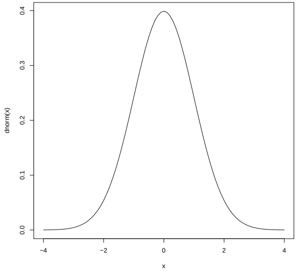
Normal: Density Function

Standard normal density:

```
> dnorm(-4)
[1] 0.0001338302
> dnorm(-2)
[1] 0.05399097
> dnorm(0)
[1] 0.3989423
> dnorm(2)
[1] 0.05399097
> dnorm(4)
[1] 0.0001338302
```

Plot standard normal density:

```
> x=seq(-4,4,by=.1)
> plot(x,dnorm(x),type="l")
```



Normal: Density Function (continued)

Normal density with different mean and variance ($\mu = 1$ and $\sigma^2 = 2$):

```
> dnorm(-3,mean=1,sd=sqrt(2))
[1] 0.005166746
> dnorm(-1,mean=1,sd=sqrt(2))
[1] 0.1037769
> dnorm(1,mean=1,sd=sqrt(2))
[1] 0.2820948
> dnorm(3,mean=1,sd=sqrt(2))
[1] 0.1037769
> dnorm(5,mean=1,sd=sqrt(2))
[1] 0.005166746
```

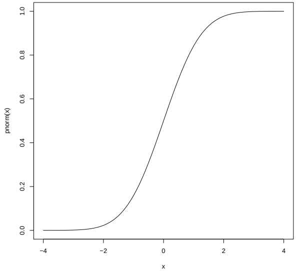
Normal: Distribution Function

Standard normal cdf:

```
> pnorm(-4)
[1] 3.167124e-05
> pnorm(-2)
[1] 0.02275013
> pnorm(0)
[1] 0.5
> pnorm(2)
[1] 0.9772499
> pnorm(4)
[1] 0.9999683
```

Plot standard normal cdf:

```
> x=seq(-4,4,by=.1)
> plot(x,pnorm(x),type="l")
```



Normal: Distribution Function (continued)

Normal cdf with different mean and variance ($\mu = 1$ and $\sigma^2 = 2$):

```
> pnorm(-3,mean=1,sd=sqrt(2))
[1] 0.002338867
> pnorm(-1,mean=1,sd=sqrt(2))
[1] 0.0786496
> pnorm(1,mean=1,sd=sqrt(2))
[1] 0.5
> pnorm(3,mean=1,sd=sqrt(2))
[1] 0.9213504
> pnorm(5,mean=1,sd=sqrt(2))
[1] 0.9976611
```

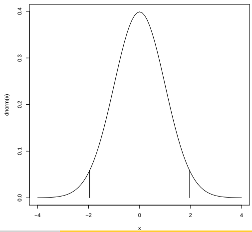
Normal: Quantile Function

Standard normal qualites:

```
> qnorm(.005)
[1] -2.575829
> qnorm(.025)
[1] -1.959964
> qnorm(.5)
[1] 0
> qnorm(.975)
[1] 1.959964
> qnorm(.995)
[1] 2.575829
```

Plot standard normal quantiles:

```
x=seq(-4,4,by=.1)
plot(x,dnorm(x),type="l")
qx=qnorm(.025)
lines(x=rep(qx,2),
      y=c(0,dnorm(qx)))
lines(x=rep(-qx,2),
      y=c(0,dnorm(-qx)))
```



Normal: Quantile Function (continued)

Normal quantiles with different mean and variance ($\mu = 1$ and $\sigma^2 = 2$):

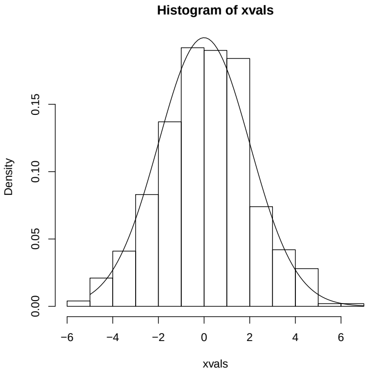
```
> qnorm(.005,mean=1,sd=sqrt(2))
[1] -2.642773
> qnorm(.025,mean=1,sd=sqrt(2))
[1] -1.771808
> qnorm(.5,mean=1,sd=sqrt(2))
[1] 1
> qnorm(.975,mean=1,sd=sqrt(2))
[1] 3.771808
> qnorm(.995,mean=1,sd=sqrt(2))
[1] 4.642773
```

Simulating Normal Data in R

For each distribution, use `rname` to simulate from `name` distribution.

For example, to simulate normal

```
> set.seed(12345)
> xvals=rnorm(1000,mean=0,sd=2)
> xseq=seq(-5,7,l=100)
> hist(xvals,freq=FALSE)
> lines(xseq,dnorm(xseq,sd=2))
```



Testing Normality in R

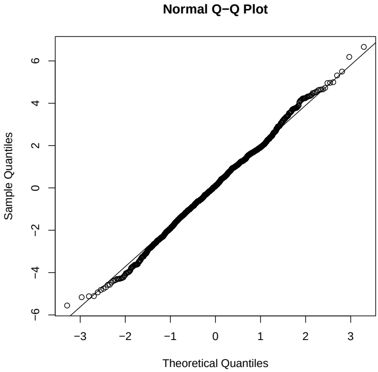
Use `qqnorm` and `qqline` to make Q-Q plot and `shapiro.test` to perform Shapiro-Wilk normality test.

For example, to test normality

```
> set.seed(12345)
> xvals=rnorm(1000,mean=0,sd=2)
> qqnorm(xvals)
> qqline(xvals)
> shapiro.test(xvals)

 Shapiro-Wilk normality test

data:  xvals
W = 0.9978, p-value = 0.1988
```



Student's *t* Distribution: Overview

Family of real-valued continuous distributions that depends on the parameter $\nu > 0$, which is the degrees of freedom.

We encounter *t* distribution when estimating μ (the mean of a normal variable) with σ^2 (the variance of the normal variable) unknown.

Called "Student's" *t* because of William Gosset...

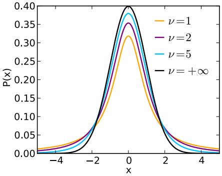
- Worked for Guinness Brewery (Dublin, Ireland) in early 1900s
- Published paper under pseudonym "Student" because Guinness did not allow employees to publish scientific papers

Student's *t* Distribution: Properties

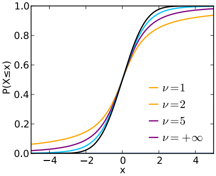
Like the standard normal distribution, the *t* distribution is bell-shaped and symmetric around zero.

For small ν, the *t* distribution has heavy tails; as $\nu \to \infty$, *t* distribution approaches standard normal distribution.

Helpful figures of *t* distribution pdfs and cdfs:



http://en.wikipedia.org/wiki/File:Student_t_pdf.svg



http://en.wikipedia.org/wiki/File:Student_t_cdf.svg

Student's *t* Distribution: R Functions

The relevant functions for the *t* distribution are...

- `dt` calculates density (pdf) value at input quantile
- `pt` calculates distribution (cdf) value at input quantile
- `qt` calculates quantile value at input probability
- `rt` generates random sample of input size

In addition to the input quantile (or probability or size) value, you can input the `df` (degrees of freedom) and `ncp` (non-centrality parameter).

- We will not discuss non-central *t* distributions
- You only need to worry about the `df` input

Student's *t* Distribution: Example Code

Student's *t* pdf (at $x = 0$):

```
> dt(0,df=1)
[1] 0.3183099
> dt(0,df=10)
[1] 0.3891084
> dt(0,df=100)
[1] 0.3979462
```

Student's *t* cdf (at $x = 0$):

```
> pt(0,df=1)
[1] 0.5
> pt(0,df=10)
[1] 0.5
> pt(0,df=100)
[1] 0.5
```

Student's *t* quantiles (at $p = .975$):

```
> qt(.975,df=1)
[1] 12.7062
> qt(.975,df=10)
[1] 2.228139
> qt(.975,df=100)
[1] 1.983972
```

Student's *t* quantiles (at $p = .995$):

```
> qt(.995,df=1)
[1] 63.65674
> qt(.995,df=10)
[1] 3.169273
> qt(.995,df=100)
[1] 2.625891
```

One Sample t Test: Overview

Suppose $x_i \stackrel{\text{iid}}{\sim} \mathrm{N}(\mu, \sigma^2)$ and want to test $H_0 : \mu = \mu_0$ versus $H_1 : \mu \neq \mu_0$

Assuming σ is unknown, use the one-sample Student's t test statistic:

$$T = \frac{\bar{x} - \mu_0}{s/\sqrt{n}} \sim t_{n-1}$$

where $\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$ and $s = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}}$

$100(1 - \alpha)\%$ confidence interval (CI) for μ is given by

$$\bar{x} \pm t_{n-1}^{(\alpha/2)}(s/\sqrt{n})$$

where $t_{n-1}^{(\alpha/2)}$ is critical t_{n-1} value such that $P\left(T > t_{n-1}^{(\alpha/2)}\right) = \alpha/2$.

One Sample *t* Test: Example

A store sells "16-ounce" boxes of Captain Crisp cereal. A random sample of 9 boxes was taken and weighed. The results were

15.5 16.2 16.1 15.8 15.6 16.0 15.8 15.9 16.2

ounces. Assume the weight of cereal in a box is normally distributed.

The sample mean and variance are given by

$$\bar{x} = (1/n)\sum_{i=1}^{n} x_i = (1/9)(15.5 + \cdots + 16.2) = (1/9)(143.1) = 15.9$$

$$s^2 = (n-1)^{-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 = (n-1)^{-1}\left[\sum_{i=1}^{n} x_i^2 - n\bar{x}^2\right]$$

$$= (1/8)\left[2275.79 - 9(15.9^2)\right] = (1/8)(0.5) = 0.0625$$

One Sample *t* Test: Example (continued)

$t_8^{(.025)} = 2.306$, so the 95% CI for the average weight of a cereal box is: $15.9 \pm 2.306\sqrt{0.0625/9} = [15.708;\ 16.092]$

The company that makes Captain Crisp cereal claims that the average weight of its box is at least 16 ounces. Use a 0.05 level of significance to test the company's claim. What is the p-value of this test?

To test $H_0 : \mu \geq 16$ versus $H_1 : \mu < 16$, the test statistic is

$$T = \frac{15.9 - 16}{\sqrt{0.0625/9}} = -1.2$$

We know that $T \sim t_8$, so we have that $P(T < -1.2) = 0.1322336$. Therefore, we retain H_0 at the $\alpha = .05$ level.

One Sample *t* Test: R Code

```
> x=c(15.5, 16.2, 16.1, 15.8, 15.6, 16.0, 15.8, 15.9, 16.2)
> mean(x)
[1] 15.9
> sd(x)
[1] 0.25
> var(x)
[1] 0.0625
> t.test(x)

 One Sample t-test

data:  x
t = 190.8, df = 8, p-value = 6.372e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 15.70783 16.09217
sample estimates:
mean of x
     15.9
```

One Sample *t* Test: R Code (continued)

```
> t.test(x,mu=16,alternative="less",conf.level=.95)

 One Sample t-test

data:  x
t = -1.2, df = 8, p-value = 0.1322
alternative hypothesis: true mean is less than 16
95 percent confidence interval:
     -Inf 16.05496
sample estimates:
mean of x
     15.9
```

Two Sample t Test: Overview

Suppose $x_i \stackrel{\text{iid}}{\sim} \mathrm{N}(\mu_x, \sigma^2)$ and $y_i \stackrel{\text{iid}}{\sim} \mathrm{N}(\mu_y, \sigma^2)$
Want to test $H_0 : \mu_x - \mu_y = \mu_0$ versus $H_1 : \mu_x - \mu_y \neq \mu_0$

Assuming σ is unknown, use the two-sample Student's t test statistic:

$$T = \frac{(\bar{x} - \bar{y}) - \mu_0}{s_p\sqrt{\frac{1}{n} + \frac{1}{m}}} \sim t_{n+m-2}$$

where $\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$, $\bar{y} = \frac{\sum_{i=1}^m y_i}{m}$, and $s_p^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2 + \sum_{i=1}^m (y_i - \bar{y})^2}{n+m-2}$

$100(1-\alpha)\%$ confidence interval (CI) for $\mu_x - \mu_y$ is given by

$$(\bar{x} - \bar{y}) \pm t_{n+m-2}^{(\alpha/2)} \left(s_p\sqrt{\frac{1}{n} + \frac{1}{m}} \right)$$

where $t_{n+m-2}^{(\alpha/2)}$ is critical t_{n+m-2} value such that $P\left(T > t_{n+m-2}^{(\alpha/2)}\right) = \alpha/2$.

Two Sample *t* Test: Example

Assume that the distributions of X and Y are $\mathrm{N}(\mu_1, \sigma^2)$ and $\mathrm{N}(\mu_2, \sigma^2)$, respectively. Given the $n = 6$ observations of X,

70, 82, 78, 74, 94, 82

and the $m = 8$ observations of Y,

64, 72, 60, 76, 72, 80, 84, 68

find the p-value for the test $H_0 : \mu_1 = \mu_2$ versus $H_1 : \mu_1 > \mu_2$.

Two Sample t Test: Example (continued)

First, note that the sample means and variances are given by

$$
\begin{aligned}
\bar{x} &= (1/6)\textstyle\sum_{i=1}^{6} x_i = (1/6)480 = 80 \\
\bar{y} &= (1/8)\textstyle\sum_{i=1}^{8} y_i = (1/8)576 = 72 \\
s_x^2 &= (1/5)\textstyle\sum_{i=1}^{6}(x_i - \bar{x})^2 = (1/5)344 = 68.8 \\
s_y^2 &= (1/7)\textstyle\sum_{i=1}^{8}(y_i - \bar{y})^2 = (1/7)448 = 64
\end{aligned}
$$

which implies that the pooled variance estimate is given by

$$
\begin{aligned}
s_p^2 &= \frac{(n-1)s_x^2 + (m-1)s_y^2}{n+m-2} \\
&= \frac{344 + 448}{12} \\
&= 66
\end{aligned}
$$

Two Sample t Test: Example (continued)

Thus, the relevant t test statistic is given by

$$
\begin{aligned}
T &= \frac{(\bar{x} - \bar{y}) - \mu_0}{s_p\sqrt{\frac{1}{n} + \frac{1}{m}}} \\
&= \frac{(80 - 72) - 0}{\sqrt{66}\sqrt{\frac{1}{6} + \frac{1}{8}}} \\
&= 1.82337
\end{aligned}
$$

Note that $T \sim t_{12}$, so the corresponding p-value is

$$P(T > 1.82337) = 0.04661955$$

Therefore, we reject H_0 at the $\alpha = .05$ level.

Two Sample *t* Test: R Code

```
> x=c(70, 82, 78, 74, 94, 82)
> y=c(64, 72, 60, 76, 72, 80, 84, 68)
> t.test(x,y,alternative="greater",var.equal=TRUE)

 Two Sample t-test

data:  x and y
t = 1.8234, df = 12, p-value = 0.04662
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.1802451       Inf
sample estimates:
mean of x mean of y
       80        72
```

Chi-Squared Distribution: Overview

Family of positive real-valued continuous distributions that depends on the parameter $k > 0$, which is the degrees of freedom.

If $Z_1, \ldots, Z_k$ are iid $\mathrm{N}(0, 1)$, then $Q = (\sum_{i=1}^{k} Z_i^2) \sim \chi_k^2$

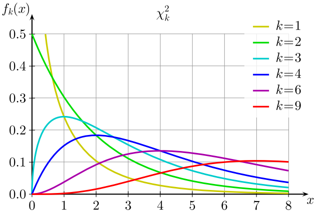
- iid: independent identically distributed
- χ_k^2 denotes a chi-squared distribution with k degrees of freedom

Chi-Squared Distribution: Properties

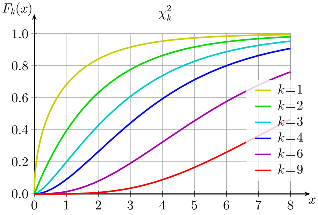
Chi-squared variable must be nonnegative (squared normal variable).

χ^2_k distribution takes a variety of different shapes depending on k.

Helpful figures of χ^2_k distribution pdfs and cdfs:



http://en.wikipedia.org/wiki/File:Chi-square_pdf.svg



http://en.wikipedia.org/wiki/File:Chi-square_cdf.svg

Chi-Squared Distribution: R Functions

The relevant functions for the χ^2_k distribution are...

- `dchisq` calculates density (pdf) value at input quantile
- `pchisq` calculates distribution (cdf) value at input quantile
- `qchisq` calculates quantile value at input probability
- `rchisq` generates random sample of input size

In addition to the input quantile (or probability or size) value, you can input the `df` (degrees of freedom) and `ncp` (non-centrality parameter).

- We will not discuss non-central χ^2_k distributions
- You only need to worry about the `df` input

Chi-Squared Distribution: Example Code

χ^2_k pdf (at $x = 1$):

```
> dchisq(1,df=1)
[1] 0.2419707
> dchisq(1,df=10)
[1] 0.0007897535
> dchisq(1,df=100)
[1] 8.856214e-79
```

χ^2_k cdf (at $x = 1$):

```
> pchisq(1,df=1)
[1] 0.6826895
> pchisq(1,df=10)
[1] 0.0001721156
> pchisq(1,df=100)
[1] 1.788777e-80
```

χ^2_k quantiles (at $p = .975$):

```
> qchisq(.975,df=1)
[1] 5.023886
> qchisq(.975,df=10)
[1] 20.48318
> qchisq(.975,df=100)
[1] 129.5612
```

χ^2_k quantiles (at $p = .995$):

```
> qchisq(.995,df=1)
[1] 7.879439
> qchisq(.995,df=10)
[1] 25.18818
> qchisq(.995,df=100)
[1] 140.1695
```

F Distribution: Overview

Family of positive real-valued continuous distributions that depends on the parameters $k_1, k_2 > 0$, which are the numerator and denominator degrees of freedom, respectively.

If $Q_1 \sim \chi^2_{k_1}$ and $Q_2 \sim \chi^2_{k_2}$ are independent, then $F = \frac{Q_1/k_1}{Q_2/k_2} \sim F_{k_1,k_2}$

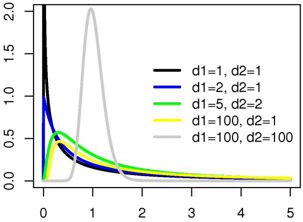
- independent: Q_1 and Q_2 are statistically independent
- F_{k_1,k_2} denotes an F distribution with k_1 numerator degrees of freedom and k_2 denominator degrees of freedom

F Distribution: Properties

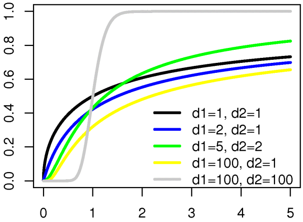
F variables must be nonnegative (ratio of scaled chi-squared).

F distribution takes a variety of different shapes depending on k_1, k_2.

Helpful figures of *F* distribution pdfs and cdfs:



http://en.wikipedia.org/wiki/File:F_distributionPDF.png



http://en.wikipedia.org/wiki/File:F_distributionCDF.png

F Distribution: R Functions

The relevant functions for the F_{k_1,k_2} distribution are...

- `df` calculates density (pdf) value at input quantile
- `pf` calculates distribution (cdf) value at input quantile
- `qf` calculates quantile value at input probability
- `rf` generates random sample of input size

In addition to the input quantile/probability/size, you can input `df1` and `df2` (degrees of freedom), and `ncp` (non-centrality parameter).

- We will not discuss non-central F_{k_1,k_2} distributions
- You only need to worry about the `df1` and `df2` inputs

F Distribution: Example Code

F_{k_1,k_2} pdf (at $x = 1$):

```
> df(1,df1=1,df2=1)
[1] 0.1591549
> df(1,df1=1,df2=10)
[1] 0.230362
> df(1,df1=10,df2=10)
[1] 0.6152344
```

F_{k_1,k_2} cdf (at $x = 1$):

```
> pf(1,df1=1,df2=1)
[1] 0.5
> pf(1,df1=1,df2=10)
[1] 0.6591069
> pf(1,df1=10,df2=10)
[1] 0.5
```

F_{k_1,k_2} quantiles (at $p = .975$):

```
> qf(.975,df1=1,df2=1)
[1] 647.789
> qf(.975,df1=1,df2=10)
[1] 6.936728
> qf(.975,df1=10,df2=10)
[1] 3.716792
```

F_{k_1,k_2} quantiles (at $p = .995$):

```
> qf(.995,df1=1,df2=1)
[1] 16210.72
> qf(.995,df1=1,df2=10)
[1] 12.82647
> qf(.995,df1=10,df2=10)
[1] 5.846678
```

Other Distributions in R

R has many more distributions that we will not discuss, e.g.:

- Beta distribution (`dbeta`, `pbeta`, `qbeta`, `rbeta`)
- Binomial distribution (`dbinom`, `pbinom`, `qbinom`, `rbinom`)
- Exponential distribution (`dexp`, `pexp`, `qexp`, `rexp`)
- Gamma distribution (`dgamma`, `pgamma`, `qgamma`, `rgamma`)
- Log Normal distribution (`dlnorm`, `plnorm`, `qlnorm`, `rlnorm`)
- Negative Binomial (`dnbinom`, `pnbinom`, `qnbinom`, `rnbinom`)
- Poisson distribution (`dpois`, `ppois`, `qpois`, `rpois`)
- Uniform distribution (`dunif`, `punif`, `qunif`, `runif`)

Note: all R distributions follow the same naming convention.

Basic Programming

Logical Operators: Overview

Logical operators derive from Boolean algebra, where values of variables are either `TRUE` or `FALSE`.

We use logical operators to execute different code depending on whether a condition is met.

Logical operators are used within *many* R functions, so an understanding of logical operators is crucial to understanding R code.

Logical Operators: R Syntax

Operator	Summary
$<$	Less than
$>$	Greater than
$<=$	Less than or equal to
$>=$	Greater than or equal to
$==$	Equal to
$!=$	Not equal to
$!x$	NOT x
$x\|y$	x OR y
$x\&y$	x AND y

Logical Operators: Example

Define objects x and y:

```
> x=y=10
```

Less than:

```
> x<y
[1] FALSE
> x<=y
[1] TRUE
```

Greater than:

```
> x>y
[1] FALSE
> x>=y
[1] TRUE
```

Equal to (not equal to):

```
> x==y
[1] TRUE
> x!=y
[1] FALSE
```

OR and AND:

```
> x=10
> y=11
> (x<11 | y<11)
[1] TRUE
> (x<11 & y<11)
[1] FALSE
```

If/Else Statements: Overview

If/Else statements are fundamental in any programming language.

We use if/else statements (in combination with logical operators) to execute different code depending on whether a condition is met.

If/Else statements always appear with logical operators.

If/Else Statements: R Syntax

General if/else syntax:

```
if(...){

     some R code

} else {

     more R code

}
```

Nested if/else syntax:

```
if(...){

     some R code

} else if(...) {

     more R code

} else {

     even more R code

}
```

If/Else Statements: Example

```
> x=10
> if(x>5){
+     x=x/2
+     y=2*x
+ } else {
+     x=x*2
+     y=x
+ }
> x
[1] 5
> y
[1] 10
```

```
> x=4
> if(x>5){
+     x=x/2
+     y=2*x
+ } else {
+     x=x*2
+     y=x
+ }
> x
[1] 8
> y
[1] 8
```

Note: the + signs are NOT part of the R code; these are included by R when entering multiline statements.

If/Else Statements: Example (continued)

To be more efficient, we could write an R function:

```
> myfun<-function(x){
+     if(x>5){
+         x=x/2
+         y=2*x
+     } else {
+         x=x*2
+         y=x
+     }
+     list(x=x,y=y)
+ }
> class(myfun)
[1] "function"
```

```
> myfun(10)
$x
[1] 5

$y
[1] 10

> myfun(4)
$x
[1] 8

$y
[1] 8
```

For Loops: Overview

For loops (or do loops) are fundamental in any programming language.

We use for loops to execute the same code repeatedly with the loop index changing at each step of the loop.

Warning: for loops in R can be slow; vectorize your code if possible!

For Loops: Syntax

```
for(j in J){

     some R code depending on j

}
```

Note: `j` is the loop index and `J` is the index set.

For Loops: Example

For loop version:

```
> x=11:15
> x
[1] 11 12 13 14 15
> for(idx in 1:5){
+     x[idx]=x[idx]+1
+ }
> x
[1] 12 13 14 15 16
```

Vectorized version:

```
> x=11:15
> x
[1] 11 12 13 14 15
> x=x+1
> x
[1] 12 13 14 15 16
```

While Statements: Overview

While statements are fundamental in any programming language.

We use while statements (in combination with logical operators) to execute the same code repeatedly until some condition is met.

While statements always appear with logical operators.

While Statements: Syntax

```
while(...){

     some R code

}
```

Note: keeps repeating R code until logical statement `...` is `FALSE`

While Statements: Example

Simple while statement:

```
> x=80
> iter=0
> while(x<100){
+       x=x+sqrt(x)/10
+       iter=iter+1
+ }
> x
[1] 100.8293
> iter
[1] 22
```

Another while statement:

```
> x=80
> iter=0
> while(x<100 & iter<20){
+       x=x+sqrt(x)/10
+       iter=iter+1
+ }
> x
[1] 98.83599
> iter
[1] 20
```

While Statements: Example (continued)

Improper while statement:

```
> iter=0
> while(x<100){
+     x=x-sqrt(x)/10
+     iter=iter+1
+ }
Error in while (x < 100) {
     : missing value where TRUE/FALSE needed
In addition: Warning message:
In sqrt(x) : NaNs produced
```

Note: we get error message because `x` becomes negative, so we get `NaN` when we take the square-root.

While Statements: Example (continued)

Infinite while statement:

```
> x=80
> iter=0
> while(x<100){
+     x=x-x/10
+     iter=iter+1
+ }
```

Note: while statement will run infinitely (until we manually stop it) because logical statement is always true (i.e., $x < 100$ always).