

PROJECT Statistical Methods I (MATH 250)

Home Sweet Home: Using Confidence Intervals to Analyze and Compare Home Prices

Overview. One of the biggest purchases we make in our lives is a home. As we buy a home we ask ourselves many questions such as:

How much should I spend for a home?

How many bathrooms are there?

What is the cost per square foot?

Problem. Suppose you are looking for a house near Charleston in Mount Pleasant, SC, and you have narrowed your search to three subdivisions: Carolina Park, Dunes West, and Park West. Now answer the following questions:

- Download the Mount Pleasant Real Estate data set available on OAKS in csv and excel formats. Import the data into R statistical software or Minitab.
- Use technology to make a histogram for List Price and describe the distribution.
- Create a scatterplot of List Price vs. Squarefootage. That is let y = List Price and x = Squarefootage and describe any patterns.
- For the variable List Price, calculate the sample mean, the sample standard deviation, and the sample size for the three different subdivisions. Put the calculations in a table and round to the nearest dollar for the sample standard deviation and the mean.
- Based on the data set and the information we have, which confidence interval should we use here, a z or a t interval? Why?
- Find the critical value for a 95% confidence level for each subdivision for the variable List Price.
- Construct an interval to estimate the true average List Price for each subdivision with 95% confidence. Based on these confidence intervals, is it possible that Carolina Park and Dunes West have the same average List Price. Discuss.
- Do you think a List Price of \$520,000 is a reasonable value for the Carolina Park subdivision?
- Do you think a List Price of \$670,000 is a reasonable value for the Dunes West subdivision?
- Do you think a List Price of \$568,000 is a reasonable value for both the Carolina Park and Park West subdivisions?

Grading: This project will be worth of **10 points** toward the final grade in this course.

Instruction: Submit your project as a single pdf on dropbox available on OAKS. Also upload the R script you used to get the answer of these above problems.

The submission is due on/before November 29th (Sunday), 5:00 PM.

Introduction

The programming language R is a free, open-source statistical software package. It is an object-oriented programming language, which means we create objects in R (vectors, matrices, etc.) that hold our data and then perform various commands on that object. Of the many objects in R, we will only discuss vectors and matrices. Commands are entered line by line; similar to a Texas Instruments calculator. Once a line is entered, R will immediately compile (evaluate) it. To start, simply open up R and a console will pop up where you can begin typing in commands. For the sake of simplicity we will mainly use short names (x, y, etc.) when labeling our data objects throughout this card, but these are just labels and you can always adjust the names as you see fit.

The program can be downloaded for free at: <http://cran.r-project.org/>. If you have questions about the commands described in this card, you can use the R Help feature: `> ? Command_Name`

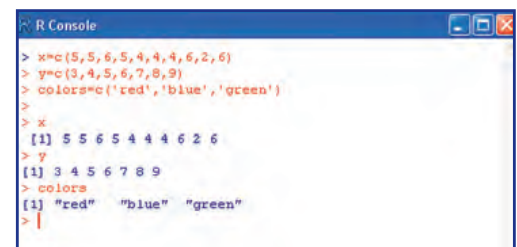
Replace **Command_Name** with the command of interest (for example: `mean`). Additionally, you can go to the CRAN website to find information on additional commands not discussed in this document. A useful starting point for further information is <http://cran.r-project.org/doc/manuals/R-intro.html>

Creating Vectors

Before performing any calculations, we first need to input our data into R. Suppose we wanted to create a vector with the values (1, 2, 3, 4) into R. To do this, we would use the command:

```
> x=c(1,2,3,4)
```

This creates a vector called `x` that contains the desired values. The command `c()` tells R to create a vector of whatever values are in the parentheses. Remember that each value must be separated by a comma. We use vectors to contain the observations for one particular variable.



```
R Console
> x=c(5,5,6,5,4,4,4,6,2,6)
> y=c(3,4,5,6,7,8,9)
> colors=c('red','blue','green')
>
> x
[1] 5 5 6 5 4 4 4 6 2 6
> y
[1] 3 4 5 6 7 8 9
> colors
[1] "red" "blue" "green"
> |
```

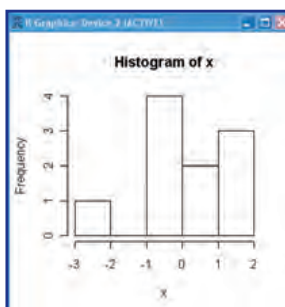
Creating Vectors from Files

Instead of entering every value, we can import the data from an external file (typically provided by your instructor). To import data from a file and save it as a vector we use the command:

```
> x=read.table(file.choose(), header=TRUE)
```

A window will pop up that lets you select the file that contains the data.

Histogram



In order to create a histogram of our observations, assuming they are stored in the vector `x`, we use the command:

```
> hist(x)
```

R will automatically decide how many bins to use for the histogram. If you would like to suggest how many bins to use, you can use the command:

```
> hist(x,breaks=a)
```

where `a` is the number of breaks to be used. The number of bins is equal to the number of breaks plus one. Note that R will take `a` as a suggestion and will sometimes add more bins than you asked for. Also, if we desire a histogram with a total area of one (useful for probability densities), then we can add in `prob=TRUE`:

```
> hist(x,breaks=a,prob=TRUE)
```



```
R Console
> x=c(1.09,.30,-.23,-2.49,1.99,.01,1.56,-.57,-.33,-.10)
> hist(x)
> |
```

ISBN-13: 978-0-321-59283-5
ISBN-10: 0-321-59283-2



Bar Chart

To create a bar chart, we first need to create a vector, `counts`, which is used for each bar. We can then use the command

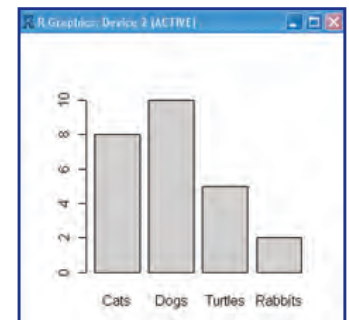
```
> barplot(counts)
```

This will create a very basic bar chart. If we would like to add a label to each bar, then we can use

```
> barplot(counts,
names.arg=c('bar1_name','bar2_name'))
```

```
R Console
> counts<-c(8,10,5,2)
> barplot(counts,names.arg=c('Cats','Dogs','Turtles','Rabbits'))
> |
```

where `bar1_name` and `bar2_name` should be replaced with the desired labels. Make sure labels are in single quotes so that R knows that they are names, not R objects. If there are more than two bars, we can include more labels by typing more names after `bar2_name`.

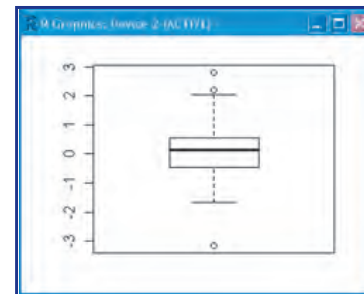


Boxplot

Assume we have a vector of our observations, called `x`. To construct a boxplot of `x`, we use the command

```
> boxplot(x)
```

```
R Console
> #We can generate data from a standard normal distribution
> x=rnorm(100,mean=0,sd=1)
>
> #And then we can create a boxplot of the data
> boxplot(x)
> |
```



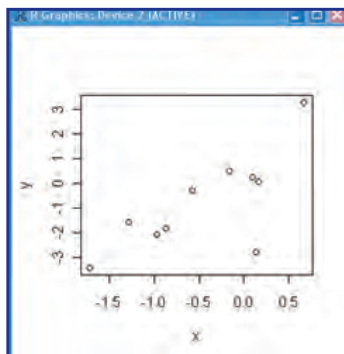
Scatterplot

To construct a scatterplot of paired data, we need two vectors for our observations. Call the first one `x` and the second one `y`. We can then use the command

```
> plot(x,y)
```

Remember that whichever variable is listed first will be plotted on the horizontal `x`-axis and the second variable will be on the vertical `y`-axis.

```
R Console
> x=c(-.58,-.97,-1.72,-1.29,.09,-.16,.67,-.87,.16,.13)
> y=c(-.26,-2.07,-3.42,-1.56,.27,.50,3.31,-1.81,.07,-2.78)
> plot(x,y)
> |
```



Descriptive Statistics

Here we list various R functions for summarizing quantitative variables. Again, let `x` be a vector containing our observations.

Five Number Summary (also calculates the mean):

```
> summary(x)
```

Quantiles (Percentiles): `> quantile(x,prob=a)`

Replace `a` with the desired quantile (in decimal form).

Mean: `> mean(x)`

Standard Deviation: `> sd(x)`

Variance: `> var(x)`

If in addition to `x`, we have a second set of observations (with the same length as `x`) in a vector called `y`, we can compute a correlation and covariance.

Correlation: `> cor(x,y)`

Covariance: `> cov(x,y)`

```
R Console
> x=c(-.58,-.97,-1.72,-1.29,.09,-.16,.67,-.87,.16,.13)
> y=c(-.26,-2.07,-3.42,-1.56,.27,.50,3.31,-1.81,.07,-2.78)
> summary(x)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.720  -0.945  -0.370  -0.454  0.120   0.670
> quantile(x,prob=.6) #60th quantile
60%
-0.06
> mean(x)
[1] -0.454
> sd(x)
[1] 0.7542944
> var(x)
[1] 0.56896
> cor(x,y)
[1] 0.7436118
> cov(x,y)
[1] 1.102389
> |
```

Normal Probabilities

Suppose we wish to compute $P(X < x)$ where X is a normal random variable with mean a and standard deviation b and x is some number we are interested in. This probability can be computed using the command

```
> pnorm(x, mean=a, sd=b)
```

```
R Console
> pnorm(0, mean=0, sd=1)
[1] 0.5
> pnorm(3, mean=0, sd=1)
[1] 0.9986501
> pnorm(3, mean=2, sd=10)
[1] 0.5398278
> pnorm(300, mean=400, sd=75)
[1] 0.09121122
> |
```

Normal Quantiles (Inverse Probability)

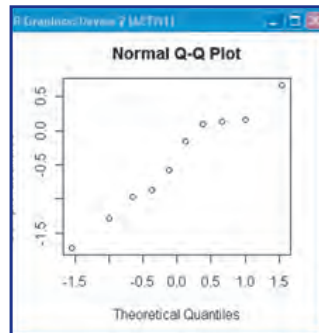
Suppose we have a probability, called q , and we would like to know what number x gives $P(X < x) = q$. Again, X is a normal random variable with mean a and standard deviation b . To find the quantile x , we can use the command

```
R Console
> qnorm(.5, mean=0, sd=1)
[1] 0
> qnorm(.99865, mean=0, sd=1)
[1] 2.999977
> qnorm(.53982, mean=2, sd=10)
[1] 2.999803
> qnorm(.091211, mean=400, sd=75)
[1] 299.9999
> |
```

Q-Q Plot

To assess normality, we can use a q-q plot. If the vector of observations is called x , we can generate a q-q plot of these observations by using the command

```
> qqnorm(x)
```



```
R Console
> x=c(-.58, -.97, -1.72, -1.29, .09, -.16, .67, -.87, .16, .13)
> qqnorm(x)
> |
```

Simple Linear Regression

Suppose that we have two vectors of observations, x and y . If we would like to calculate a regression line where y is the dependent or response variable and x is the independent or explanatory variable, we can use the command

```
> lm(y ~ x)$coef
```

The above command will produce two numbers: the first will be the intercept and the second will be the slope.

```
R Console
> x=c(-.58, -.97, -1.72, -1.29, .09, -.16, .67, -.87, .16, .13)
> y=c(-.26, -2.07, -3.42, -1.56, .27, .50, 3.31, -1.81, .07, -2.78)
> lm(y ~ x)$coef
      (Intercept)           x 
      0.1046481    1.9375508 
> |
```

Goodness of Fit for Regression

To assess the goodness of fit for the regression line between y and x , we can look at a plot of the residuals. If the fit is good, then the plot of the residuals should resemble random scatter about zero. The command to plot the residuals is

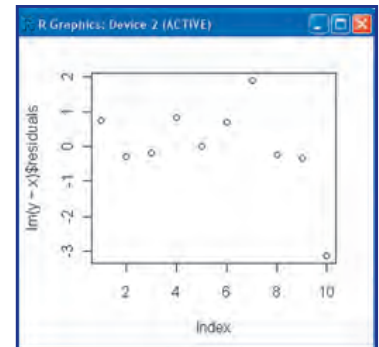
```
> plot(lm(y ~ x)$residuals)
```

Another plot command to test the goodness of the fit is

```
> plot(x, lm(y ~ x)$residuals)
```

```
R Console
> x=c(-.58, -.97, -1.72, -1.29, .09, -.16, .67, -.87, .16, .13)
> y=c(-.26, -2.07, -3.42, -1.56, .27, .50, 3.31, -1.81, .07, -2.78)
> plot(lm(y ~ x)$residuals)
> |
```

This plots the residuals against the explanatory variable, but again we are looking for random scatter about zero.



Sampling

Suppose we have a vector of values we would like to sample from; call that vector x . We can create a sample of size n from x using the command

```
> sample(x, size=n, replace=FALSE)
```

Remember to replace n with the desired sample size. The above will draw n objects from x without replacement. If replacement is desired, change `FALSE` to `TRUE`. Also, if you would like to store the sample in a new object, call it y , use the command

```
> y=sample(x, size=n, replace=FALSE)
```

```
R Console
> x=c('Tom', 'Emily', 'Bill', 'Mike', 'Susan', 'Jen', 'Tony')
> sample(x, size=3, replace=FALSE)
[1] "Jen" "Susan" "Mike"
> sample(x, size=3, replace=FALSE)
[1] "Emily" "Jen" "Tom"
> sample(x, size=3, replace=FALSE)
[1] "Emily" "Tom" "Jen"
> sample(x, size=3, replace=FALSE)
[1] "Susan" "Emily" "Bill"
> 
> sample(x, size=3, replace=TRUE)
[1] "Susan" "Bill" "Mike"
> sample(x, size=3, replace=TRUE)
[1] "Tom" "Tom" "Tom"
> sample(x, size=3, replace=TRUE)
[1] "Susan" "Susan" "Jen"
> sample(x, size=3, replace=TRUE)
[1] "Tom" "Tony" "Susan"
> |
```

Generating Random Data

We can generate random data according to any number of probability distributions. Suppose we want to generate n independent random observations following a normal distribution with mean a and standard deviation b and save them in the vector `x_normal`. We would use the command

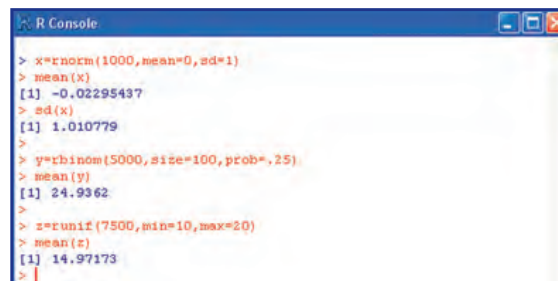
```
> x_normal = rnorm(n, mean=a, sd=b)
```

Next, suppose we want to generate n observations from a binomial distribution where the number of trials for each observation is a and the probability of success for each trial is b and we want to save them in the vector `x_binomial`. We would use the command

```
> x_binomial = rbinom(n, size=a, prob=b)
```

Finally, suppose we want to generate n observations from a uniform distribution on the interval $[a, b]$ and store them in the vector `x_unif`. We would use the command

```
> x_unif = runif(n, min=a, max=b)
```

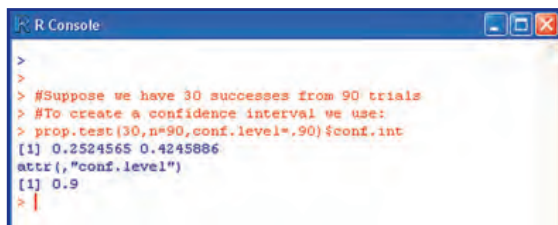


```
R Console
> x=rnorm(1000,mean=0,sd=1)
> mean(x)
[1] -0.02295437
> sd(x)
[1] 1.010779
>
> y=rbinom(5000,size=100,prob=.25)
> mean(y)
[1] 24.9362
>
> z=runif(7500,min=10,max=20)
> mean(z)
[1] 14.97173
>
```

Confidence Intervals for Proportions

Suppose we want to generate a confidence interval for a particular proportion. Let x be the number of successes, N be the number of trials, and a be the confidence level (in decimal form). The command to generate the interval is

```
> prop.test(x,n=N,conf.level=a)$conf.int
```



```
R Console
>
> #Suppose we have 30 successes from 90 trials
> #To create a confidence interval we use:
> prop.test(30,n=90,conf.level=.90)$conf.int
[1] 0.2524565 0.4245886
attr(,"conf.level")
[1] 0.9
>
```

Confidence Intervals for Means

Suppose we want to compute a confidence interval for the mean using our observations stored in the vector `x`. We use the command

```
> t.test(x,conf.level=a)$conf.int
```

Where a is the confidence level (in decimal form). The above will return the lower and upper bound of the interval as well as the confidence level used.



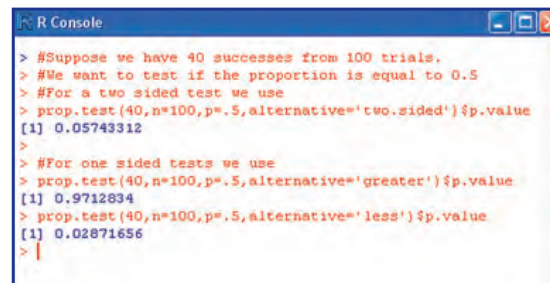
```
R Console
>
> x=c(-.58,-.97,-1.72,-1.29,.09,-.16,.67,-.87,.16,.13)
> t.test(x,conf.level=.95)$conf.int
[1] -0.99358969 0.08558969
attr(,"conf.level")
[1] 0.95
>
```

Hypothesis Tests for Proportions

Suppose we want to perform a hypothesis test for proportions. Let x be the number of successes, N be the number of trials, and p_{null} be the probability of success under the null hypothesis. The command is

```
> prop.test(x,n=N,p=p_null,alternative=
'two.sided')$p.value
```

The above command will give back the p -value for the test. If the test is not two sided, replace `two.sided` with `greater` or `less` depending on the desired test.



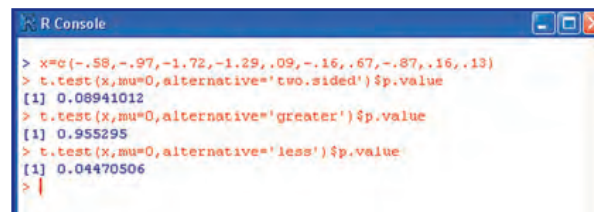
```
R Console
> #Suppose we have 40 successes from 100 trials.
> #We want to test if the proportion is equal to 0.5
> #For a two sided test we use
> prop.test(40,n=100,p=.5,alternative='two.sided')$p.value
[1] 0.05743312
>
> #For one sided tests we use
> prop.test(40,n=100,p=.5,alternative='greater')$p.value
[1] 0.9712834
> prop.test(40,n=100,p=.5,alternative='less')$p.value
[1] 0.02871656
>
```

Hypothesis Tests for Means

If we want to perform a hypothesis test, then we again use the command `t.test`. If x is the vector of our observations and μ_{null} is the value of the mean under the null hypothesis, command we use is

```
> t.test(x,mu=mu_null,alternative=
'two.sided')$p.value
```

The above will return the p -value of the test. Again, if a one-sided test is desired, replace `two.sided` with `greater` or `less`.



```
R Console
>
> x=c(-.58,-.97,-1.72,-1.29,.09,-.16,.67,-.87,.16,.13)
> t.test(x,mu=0,alternative='two.sided')$p.value
[1] 0.08941012
> t.test(x,mu=0,alternative='greater')$p.value
[1] 0.955295
> t.test(x,mu=0,alternative='less')$p.value
[1] 0.04470506
>
```

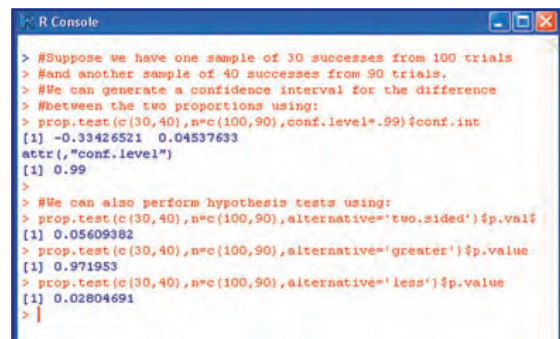

Two Sample Inference for Proportions

The commands for two sample inferences are very similar to the one sample cases. Let x be the number of successes for the first sample and let N be the sample size of the first sample. Let y be the number of successes for the second sample and let M be the sample size of the second sample. The command to construct a confidence interval for the difference between the two proportions is

```
> prop.test(c(x,y),n=c(N, M),conf.level= a)$conf.int
```

If we want to perform a hypothesis test that the two population proportions are equal, we use the command

```
> prop.test(c(x,y),n=c(N, M),alternative=
two.sided)$p.value
```



```
R Console
> #Suppose we have one sample of 30 successes from 100 trials
> #and another sample of 40 successes from 90 trials.
> #We can generate a confidence interval for the difference
> #between the two proportions using:
> prop.test(c(30,40),n=c(100,90),conf.level=.99)$conf.int
[1] -0.33426521 0.04537633
attr(,"conf.level")
[1] 0.99
>
> #We can also perform hypothesis tests using:
> prop.test(c(30,40),n=c(100,90),alternative='two.sided')$p.val#
[1] 0.05609382
> prop.test(c(30,40),n=c(100,90),alternative='greater')$p.value
[1] 0.971953
> prop.test(c(30,40),n=c(100,90),alternative='less')$p.value
[1] 0.02804691
> |
```

Two Sample Inference for Means

We can also use the `t.test` command to perform two sample tests comparing means. Let x be a vector containing the observations of the first sample and y the observations from the second. The command to construct a confidence interval of size a for the difference between the two means, assuming equal variances, is

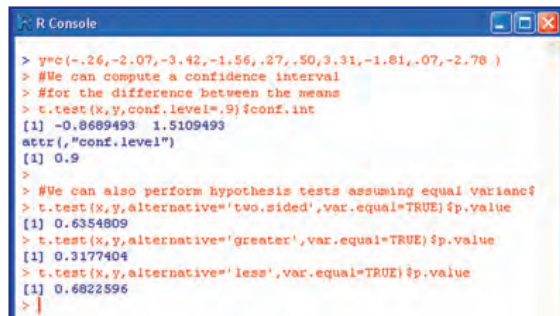
```
> t.test(x,y,conf.level=a,var.equal=TRUE)$conf.int
```

If we wish to test whether the population means are equal, assuming that the two samples come from populations with equal variances, we use the command

```
> t.test(x,y,alternative='two.sided',
var.equal=TRUE)
```

If we wish to perform a one-sided test, we replace `two.sided` with `greater` or `less` depending on the desired test. Notice that

`greater` will test if the mean for sample x is greater than the mean for sample y . Changing `TRUE` to `FALSE` will tell R not to assume that the variances are equal.



```
R Console
> y=c(-1.26,-2.07,-3.42,-1.56,.27,.50,3.31,-1.81,.07,-2.78 )
> #We can compute a confidence interval
> #for the difference between the means
> t.test(x,y,conf.level=.9)$conf.int
[1] -0.8689493 1.5109493
attr(,"conf.level")
[1] 0.9
>
> #We can also perform hypothesis tests assuming equal variance
> t.test(x,y,alternative='two.sided',var.equal=TRUE)$p.value
[1] 0.6354809
> t.test(x,y,alternative='greater',var.equal=TRUE)$p.value
[1] 0.3177404
> t.test(x,y,alternative='less',var.equal=TRUE)$p.value
[1] 0.6822596
> |
```

ANOVA

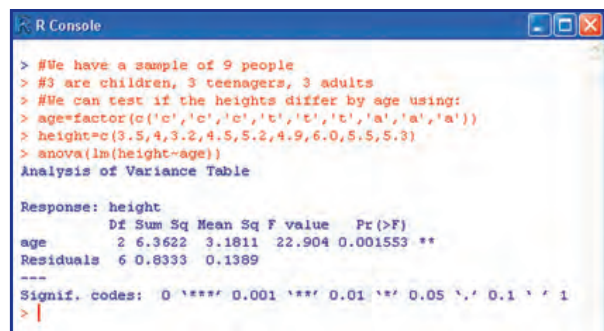
In a one-way analysis of variance, we wish to test if the means of multiple groups are equal. As before, let x be the vector of observations. Next, we need to create a `factor` object that shows which observations in x belong to which group. Suppose we have three groups A, B, and C. Also assume the first two observations in the vector x are from sample A, the second two from sample B, and the last two from sample C. We would then create a factor object using the command

```
> group = factor(c('A','A','B','B','C','C'))
```

If there are more observations, we just include more letters (one for each observation). Since the first two elements of `group` are A, R will know that the first two elements of x are from group A when we perform the ANOVA. Also there is no need to use A, B, or C; we could use any names we like to distinguish the groups, and we could have any number of groups by including more names. To perform a one-way ANOVA we use the command

```
> anova(lm(x~group))
```

Notice that there are actually two commands above. The `lm` command tells R that x is the dependent or response variable and that y is the independent factor. The `anova` command then tells R to carry out the one-way ANOVA on the observations x given that `group` is the factor. R will display several pieces of information when performing an ANOVA, but the value listed under `P(>F)` is the p -value of the test.



```
R Console
> #We have a sample of 9 people
> #3 are children, 3 teenagers, 3 adults
> #We can test if the heights differ by age using:
> age=factor(c('c','c','c','t','t','t','a','a','a'))
> height=c(3.5,4,3.2,4.5,5.2,4.9,6.0,5.5,5.3)
> anova(lm(height~age))
Analysis of Variance Table

Response: height
          Df Sum Sq Mean Sq F value    Pr(>F)
age         2  6.3622   3.1811  22.904 0.001553 **
Residuals   6  0.8333   0.1389
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> |
```

Side-by-Side Boxplots

Let x be the vector of observations and $group$ be the factor (as in the one way ANOVA). We can construct a boxplot of each group and plot them side by side using the command

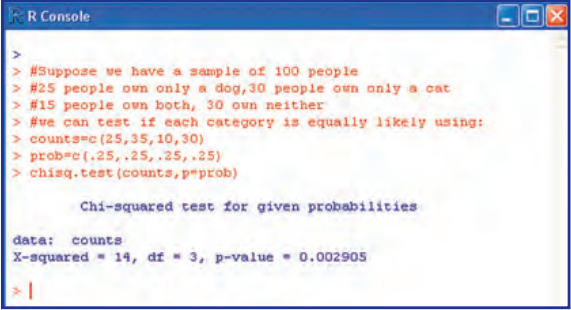
```
> boxplot(x~group)
```

Goodness of Fit

Suppose we have computed the counts (see Bar Chart for reminder) of our variable, saved them in a vector `counts`, and we wish to test if our variable follows a given probability distribution. Let `prob_x` be a vector of the probability distribution for the variable we wish to test. We can then perform a goodness-of-fit test using the command

```
> chisq.test(counts,p=prob_x)
```

The above command will compute the chi-squared value, the degrees of freedom, and the p -value for the test.



```
R Console
> #Suppose we have a sample of 100 people
> #25 people own only a dog, 30 people own only a cat
> #15 people own both, 30 own neither
> #We can test if each category is equally likely using:
> counts=c(25,35,10,30)
> prob=c(.25,.25,.25,.25)
> chisq.test(counts,p=prob)

Chi-squared test for given probabilities

data: counts
X-squared = 14, df = 3, p-value = 0.002905
> |
```

Contingency Tables

Creating Tables Before performing any statistical tests associated with contingency tables, we first need to create a table in R. There are multiple methods of doing this, but we will show how to create tables using the `matrix` command. First, we list the table as one long vector. In this example, we will call it `x`. Create `x` by listing the element in the first row and first column of the table. Then list the element in the first row and the second column and continue to list the rest of the first row. Then move to the second and list the second row and continue until the entire table is listed as one long vector. Now we turn `x` into a matrix, calling it `x_matrix`, using the command

```
> x_matrix=matrix(x,nrow=3,ncol=3,byrow=T)
```

This will generate a basic table that can be used for our computations. If you would like to label the table, we need to create a `list` of the row and column names; in this example, we will call the list `var_names`. Suppose we have two rows and two columns. The row names are `r_name1` and `r_name2` and the column names are `c_name1` and `c_name2`. Then we create a `list` of our names with the command

```
> var_names=list(c('r_name1','r_name2'),c('c_name1','c_name2'))
```

Make sure to list the row names in the first `c()` and column names in the second `c()`. Of course, replace `r_name1` and so on with whatever labels you prefer. If you have more than two rows or columns, just continue to list additional row names in the first `c()` and additional column names in the second `c()`. Now we can create a labeled table with the command

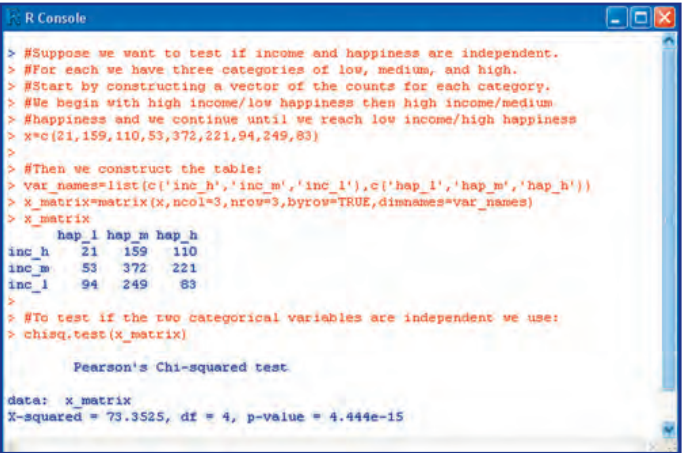
```
> x_matrix = matrix(x,nrow=3,ncol=3,byrow=T,dimnames=var_names)
```

Chi-Squared Test for Independence

Once a table is created, we can perform a chi-squared test for independence using the command

```
> chisq.test(x_matrix)
```

This command will compute the chi-squared value, the degrees of freedom, and the p -value of the test.



```
R Console
> #Suppose we want to test if income and happiness are independent.
> #For each we have three categories of low, medium, and high.
> #Start by constructing a vector of the counts for each category.
> #We begin with high income/low happiness then high income/medium
> #happiness and we continue until we reach low income/high happiness
> x=c(21,159,110,53,372,221,94,249,83)
>
> #Then we construct the table:
> var_names=list(c('inc_h','inc_m','inc_l'),c('hap_l','hap_m','hap_h'))
> x_matrix=matrix(x,ncol=3,nrow=3,byrow=TRUE,dimnames=var_names)
> x_matrix
      hap_l hap_m hap_h
inc_h    21   159   110
inc_m    53   372   221
inc_l    94   249    83
>
> #To test if the two categorical variables are independent we use:
> chisq.test(x_matrix)

Pearson's Chi-squared test

data: x_matrix
X-squared = 73.3525, df = 4, p-value = 4.444e-15
```