Chapter 1 R Instructions
Randomization Tests: Schistosomiasis

We suggest that you install the following packages before running the following code:

```
install.packages('mosaic')
require(mosaic)
install.packages('dplyr')
require(dplyr)
```

Activities

2. Read in the Mice data and attach it to your session:

```
C1.Mice <- read.csv("~/MAT310/Clean csv data/C1 Mice.csv")
View(`C1.Mice`)
```

To create a dotpot of the Mice data:

```
stripchart(C1.Mice, vertical = TRUE, method = "jitter", col= c("red",
"blue", "green", "purple"))
```

To obtain the summary statistics for each of the groups of mice:

Female Treatment Group:

```
c(summary(C1.Mice$Female.Trt),"Stand Dev"=sd(C1.Mice$Female.Trt))
```

Female Control Group

```
c(summary(C1.Mice$Female.Ctl),"Stand Dev"=sd(C1.Mice$Female.Ctl))
```

Male Treatment Group

```
c(summary(C1.Mice$Male.Trt),"Stand Dev"=sd(C1.Mice$Male.Trt))
```

Male Control Group

```
c(summary(C1.Mice$Male.Ctl),"Stand Dev"=sd(C1.Mice$Male.Ctl))
```

7-10. The R code below will simulate 10,000 random allocations.

Note that R code can be entered one line at a time into R at the prompt symbol, >. Alternatively, you can enter the code into an R script (click File >> New File >> R Script) and then cut and paste all the code into R at once to have R execute it. The cut-and-paste approach has advantages, primarily that you can try the code with a small number of iterations and then run it for real with a large number.

The characters from the # symbol to the end of a line are not interpreted by R; they form comments to help us remember what a particular line does; comments are optional, but often useful.

```
reps <- 10000
ctl.fem <- c(16,10,10,7,17) # directly typing in the female control data
trt.fem <- c(1,2,2,10,7) # directly typing in the female treatment data
results <- numeric(reps) # establish a vector of the right length
# this create a vector of 10,000 ''0''s initially
x <- c(trt.fem, ctl.fem) #combines both the trt and ctl vectors
for (i in 1:reps) {
temp <- sample(x)
results[i] <- mean(temp[1:5])-mean(temp[6:10])}
```

Note: Often minus signs or parentheses do not copy and paste into Rstudio. If you obtain an error, you can open an R script [File>>New File>>R Script], copy and paste the code into the R script, then retype the parentheses.

9.

```
p.value <- sum(results >= 7.6) / reps
p.value
```

10.

```
hist(results,xlab="Control Mean - Treatment Mean")
```

```
abline(v = 7.6)   # adds a line on the graph to represent the test statistic (7.6 in this example)
```

Note: You can also determine the location of the breaks in the histogram, type ?hist to learn more.

13.
```
p.value <- (sum(results <= -7.6)+sum(results >= 7.6)) / reps
p.value
```

14. Number of simulations resulting in a difference greater than or equal to 7.6:
```
sum(results >= 7.6)
```
Number of simulations resulting in a difference less than or equal to -7.6:
```
sum(results <= -7.6)
```

Using R Functions

When you want to use a string of R code repeatedly, making slight modifications each time, it is most efficient to use an R function. The code below illustrates the idea in the context of randomization tests. You can find more general information about writing R functions in the documentation available at http://cran.us.r-project.org/.

Below is R code for defining a function that calculates a randomization test on two vectors of quantitative data from a randomly-allocated, two-group experiment. The first line is the crucial line; it assigns the name `randtest` to a new function. The general syntax for defining a function: `function.name <- function( )`

- `Open an R script` (click File >> New File >> R Script)
- `Copy and paste the following code into the script:`
- `Select Source to save the code:`

```
randtest <- function(x,y,fun=mean,reps=10000) {
n <- length(x)
m <- length(y)
data <- c(x,y)
results <- numeric(reps)
for (i in 1:reps) {
  simtemp <- sample(data)
  results[i] <- fun(simtemp[1:n])-fun(simtemp[(n+1):(n+m)])
}
greater.p <- sum(results >= (fun(x)-fun(y)))/reps
less.p <- sum(results <= (fun(x)-fun(y)))/reps
test.stat <- abs(fun(x)-fun(y))
two.sided.p <- sum(abs(results)>=test.stat)/reps
p.values <- c(greater.p, less.p, two.sided.p)
names(simtemp) <- c("p.greater.than", "p.less.than",
                    "two.sided.p")
return(list(results,p.values,test.stat))
}
```

This R code creates a function to perform a randomization test on two vectors of data, x and y. The output of the function is a list of length three items:

1. A vector of length reps representing the approximate randomization distribution;
2. Three p-values: “less than” and “greater than” one-sided p-values, as well as a “two-sided” p-value; and
3. The test statistic, which by default, is the difference in means between x and y

To run this `randtest` function and view a histogram of only the results vector
```
newtest = randtest(ctl.fem,trt.fem)
hist(newtest[[1]])
abline(v = 7.6)  # adds a line on the graph to represent the test statistic (7.6 in this example)
```

Extended Activities

17. Read the data set Age in your R session:

```
reps <- 10000
layoff.no <- c(25,33,35,38,48,55,56)
layoff.yes <- c(55,55,64)
results <- numeric(reps)
x <- c(layoff.no, layoff.yes)

for (i in 1:reps) {
temp <- sample(x)
results[i] <- mean(temp[1:3])-mean(temp[4:9])
}

p.value<-sum(results>=(16.5714))/reps
p.value
```

Question 17. Option 2:

```
C1.Age <- read.table("FILE PATH/Age Discrim.txt",header=TRUE)
```

Create two vectors of ages corresponding to those who were and were not laid off:

```
layoff.yes <- subset(C1.Age, C1.Age$layoff == "yes") [,1]
layoff.no <- subset(C1.Age, C1.Age$layoff == "no") [,1]
```

The [,1] command is used to select only the first column of data.

Use the randtest function to conduct the permutation test:

```
age.perm <- randtest(layoff.yes,layoff.no,fun=mean,reps=10000)
```

To obtain the empirical p-value of the permutation test:

```
age.perm[[2]]
```

Select the p-value under the `"p.greater.than"` label, since we are trying to determine whether the mean age of people who were laid off is higher than the mean age of people who were not laid off.

18. Compare the median age of those who were and were not laid off:

```
age.perm <- randtest(layoff.yes,layoff.no,fun=median,reps=10000)
age.perm[[2]]
```

Select the p-value under the `"p.greater.than"` label, since we are trying to determine whether the median age of people who were laid off is higher than the median age of people who were not laid off.

20. The following R code defines a function, called `randtest.paired` to perform a randomization test on two vectors of data, x and y, from a matched pairs design. The output of the function is a list of three p-values, plus the vector, `results`, that represents the empirical randomization distribution.

To use the function, use syntax like: `output <- randtest.paired(x,y).` You can also change the default number of simulations of 10000 by specifying "`reps =`" in the function. Then the variable `output` contains a list with two elements: the first is a length-3 vector of three p-values; the second is a vector of length `reps.` To get only the vector of results, use the `output[[2]]command.`

```
randtest.paired <- function(x,y,fun=mean,reps=10000){
data <- dats <- cbind(x,y)
observed <-fun(x)-fun(y)
results<-numeric(reps)
n<-length(data[,1])
for(i in 1:reps){
for(j in 1:n){
dats[j,]<-sample(c(-1,1),1)*data[j,]
}
results[i]<-(fun(dats[,1]))-(fun(dats[,2]))
}
greater.p <- sum(results>=observed)/reps
less.p <- sum(results<=observed)/reps
two.sided.p <- sum(abs(results)>=abs(observed))/reps
temp <- c(greater.p,less.p,two.sided.p)
names(temp)<- c("p.greater.than","p.less.than","two.sided.p")
return(list(temp,results))
```

```
}
```

Read in the `Music` data and then attach it to your R session:

```
music <- read.table("FILE PATH/Music.txt",header=TRUE)
attach(music)
```

We'll use the `randtest.paired` and 10000 simulations:

```
music.rand.pair <- randtest.paired(Fastdiff,Slowdiff,fun=mean,reps=10000)
```

a) To construct a histogram of the mean differences:

```
hist(music.rand.pair[[2]],xlab="Fastdiff-Slowdiff (beats per minute)")
```

b) To obtain the empirical p-value for the upper tailed test, observe the p-value under the `"p.greater.than"` label: `music.rand.pair[[1]]`

21. Read in the `ChiSq` data:

```
chisq <- read.table("FILE PATH/ChiSq.txt",header=FALSE)
```

Construct and observe the histogram of the sampling distribution of X :

```
reps <- 1000
means <- numeric(reps)
for (i in 1:reps) {
means[i] <- mean(sample(chisq[,1],40))
}
hist(means,main="Sampling Distribution of X-bar",xlab="Sample Mean")
```

Mean and standard deviation of the sampling distribution of X :

```
mean(means)
sd(means)
```

22. Construct the bootstrap distribution of X :

```
samp.chisq <- sample(chisq[,1],40)
reps <- 1000
means1 <- numeric(reps)
for (i in 1:reps) {
means1[i] <- mean(sample(samp.chisq,40,replace=TRUE))
}
hist(means1,main="Bootstrap Distribution of X-bar",xlab="Resample Mean")
```

23. Construct the sampling distribution and bootstrap distribution of the sample standard deviation:

```
par(mfrow=c(1,2))
reps <- 1000
sds <- numeric(reps)
for (i in 1:reps) {
sds[i] <- sd(sample(chisq[,1],40))
}
hist(sds,main="Sampling Distribution of the Sample SD",xlab="Sample SD")
samp.chisq <- sample(chisq[,1],40)
reps <- 1000
sds1 <- numeric(reps)
for (i in 1:reps) {
sds1[i] <- sd(sample(samp.chisq,40,replace=TRUE))
}
hist(sds1,main="Bootstrap Distribution of the Sample SD",xlab="Resample
SD")
```

Note: You may need to zoom in to see the labels of the graphs.

24. Read in the `MedSalaries` data:

```
med <- read.table("FILE PATH/MedSalaries.txt",header=FALSE)
```

a) Construct the bootstrap distribution of the sample mean:

```
samp.size <- 100
samp.med <- sample(med[,1],samp.size)
reps <- 1000
means <- numeric(reps)
for (i in 1:reps) {
means[i] <- mean(sample(samp.med,samp.size,replace=TRUE))
}
hist(means,main="Bootstrap Distribution of X-bar",xlab="Resample Mean")
```

95% Bootstrap t Confidence Interval for the population mean:
Lower Limit: `mean(means)-qt(.975,samp.size-1)*sd(means)`
Upper Limit: `mean(means)+qt(.975,samp.size-1)*sd(means)`

95% Bootstrap Percentile Confidence Interval for the population mean:
Lower Limit: `quantile(means,.025)`
Upper Limit: `quantile(means,.975)`

b) Construct the bootstrap distribution of the sample standard deviation

```
samp.med <- sample(med[,1],samp.size)
reps <- 1000
sds <- numeric(reps)
for (i in 1:reps) {
sds[i] <- sd(sample(samp.med,samp.size,replace=TRUE))
}
hist(sds,main="Bootstrap Distribution of the Sample
SD",xlab="ResampleSD")
```

95% Bootstrap t Confidence Interval for the population standard deviation:
Lower Limit: `mean(sds)-qt(.975,samp.size-1)*sd(sds)`
Upper Limit: `mean(sds)+qt(.975,samp.size-1)*sd(sds)`

95% Bootstrap Percentile Confidence Interval for the population standard deviation:
Lower Limit: `quantile(sds,.025)`
Upper Limit: `quantile(sds,.975)`

25. Read in the `Baseball` data set:

```
baseball <- read.csv("~/MAT310/Clean csv data/C1 NLBB Salaries.csv")
```

Create two vectors containing the salaries of pitchers and first basemen:

```
pitcher <- baseball[baseball$Position=="Pitcher",]$Salary
firstbase <- baseball[baseball$Position=="First Baseman",]$Salary
```

Perform the Wilcoxon Rank Sum Test:

```
wilcox.test(pitcher,firstbase,exact=FALSE)
```

Note that the test statistic is different from the one given in the chapter (W = 18). R uses a variant of the rank sum test statistic; however, the p-value matches that from Minitab.

26. `2*pnorm(18,27.5,4.787)`

27. Two-sample t-test to compare the average salaries of pitchers and first basemen, not assuming equal variances:

```
t.test(pitcher,firstbase)#The default is var.equal = FALSE
```

28. Kruskal-Wallis test to determine if the distribution of salaries differs by position:

```
kruskal.test(Salary~Position,data=baseball)
```

29. Before conducting this test, you may prefer to subset the data into the different makes of the cars.

```
Pontiac <- subset(C1.Car1, C1.Car1$Make == "Pontiac")
```

To get a vector of only the car prices:

```
Pontiac <- subset(C1.Car1, C1.Car1$Make == "Pontiac")[[1]]
```

Chapter 2 R Instructions
The Two-Sample t-test, Regression, and ANOVA: Making Connections

We suggest that you install the following packages before running the following code:

```
install.packages('mosaic')
require(mosaic)
```

Activities

4. Read data set `Games1` into your R session and name it "`games1`":

```
games1 <- read.csv("~/MAT310/C2 Games1.csv")
```

Create a boxplot to compare the times for the color and standard grames:

```
boxplot(Time~Type,data=games1,ylab="Seconds")
```

Create two vectors of Time observations for the Standard and Color groups:

```
Standard <- games1[games1$Type=="Standard",]$Time
Color <- games1[games1$Type=="Color",]$Time
```

Mean and standard deviations for the time to win the game for standard game:

```
mean(Standard)
sd(Standard)
```

Mean and standard deviations for the time to win the game for color game:

```
mean(Color)
sd(Color)
```

6. Sample size, 12th observation, and 12th residual for the `Standard` group:

```
length(Standard)
Standard[12]
Standard[12]-mean(Standard)
```

Sample size, 12th observation, and 12th residual for the `Color` group:

```
length(Color)
Color[12]
Color[12]-mean(Color)
```

7. Create a vector of residuals for all observations:

```
residuals <- c(Standard-mean(Standard),Color-mean(Color))
```

Create a histogram of the residuals:

```
hist(residuals)
```

8. Compute the ratio of the maximum of the sample standard deviations for the `Standard` and `Color` groups to the minimum of sample standard deviations for the `Standard` and `Color` groups:

```
max(c(sd(Standard),sd(Color)))/min(c(sd(Standard),sd(Color)))
```

9. *Note: for efficiency, the approach to this Question is identical to the approach in Question 11.*
Create a dummy variable where a value of 1 corresponds to the color distracter game:

```
D1 <- (games1$Type=="Color")*1
# this creates a new column 1 for Color and 0 for Standard
```

Regress `Time` on the dummy variable `D1` and name the regression object `games.reg1`:

```
games.reg1 <- lm(Time~D1,data=games1)
summary(games.reg1 # to see the regression output
```

Obtain the vector of residuals:

```
residuals <- games.reg1$res
```

Plot of the residuals versus the order in which the data were collected.

```
plot(games1$studentID,residuals,type="l",xlab="Order")
points(games1$studentID,residuals)
abline(h = 0)
```

10. Perform a two-sample t-test to compare average winning time between the `Standard` and `Color` groups (assuming equal variances):

```
t.test(Time~Type,data=games1,var.equal = TRUE)

# In this chapter we are assuming equal variances. Using ''var.equal = FALSE''
will give the more common t-test with unequal variances.
```

11. Create a dummy variable where a value of 1 corresponds to the color distracter game:

```
D1 <- (games1$Type=="Color")*1
```

Regress `Time` on the dummy variable `D1` and name the regression object `games.reg1`:

```
games.reg1 <- lm(Time~D1,data=games1)
```

Estimated coefficients of the model of `Time` regressed on `D1`:

```
games.reg1$coef
```

12. The partial output from the following code contains the t-statistic and p-value for the hypothesis test that $\beta_1 = 0$ versus the alternative $\beta_1 \neq 0$:

```
summary(games.reg1)
```

The 95% confidence interval for β_1 is computed by:
Lower Limit: `2.55 - qt(.975,nrow(games1)-2)*1.1154`
Upper Limit: `2.55 + qt(.975,nrow(games1)-2)*1.1154`

```
# 2.55 is the slope coefficient and 1.1154 is the corresponding standard
deviation.
```

13. Now create a dummy variable where a value of 1 corresponds to the Standard game:

```
D2 <- (games1$Type=="Standard")*1
```

Regress `Time` on the dummy variable `D1` and name the regression object `games.reg2`:

```
games.reg2 <- lm(Time~D2,data=games1)
```

Estimated coefficients of the model of Time regressed on D1: `games.reg2$coef`

The partial output from the following code contains the t-statistic and p-value for the hypothesis test that $\beta_1 = 0$ versus the alternative $\beta_1 \neq 0$:

```
summary(games.reg2)
```

The 95% confidence interval for β_1 is computed by:
Lower Limit: `-2.55 - qt(.975,nrow(games1)-2)*1.1154`
Upper Limit: `-2.55 + qt(.975,nrow(games1)-2)*1.1154`

14. Obtain the residuals from the regression equation obtained in Question 11. First, create a dummy variable where a value of 1 corresponds to the color distracter game:

```
D1 <- (games1$Type=="Color")*1
```

Now obtain residuals from the regression model using D1 as the explanatory variable:

```
residuals <- lm(Time~D1,data=games1)$res
```

Construct a histogram of the residuals and a residual versus order plot:

```
par(mfrow=c(1,2))
hist(residuals)
plot(games1$studentID,residuals,xlab="Order",ylab="Residuals")
lines(games1$studentID,residuals)
abline(1,0)
```

15. Create scatterplot of `Time` versus `Type` of game with a superimposed regression line:

```
plot(D1,games1$Time)
abline(lm(Time~D1,data=games1)$coef)
```

19. Compute $y_{..}$, $y_{1.}$, and $y_{2.}$, respectively:

```
mean(games1$Time)
mean(Standard)
mean(Color)
```

20. Estimate the effect size for the color distracter game:

```
mean(Color)-mean(games1$Time)
```

Estimate the effect size for the standard game:

```
mean(Standard)-mean(games1$Time)
```

21. Construct the main effects plot of game type:

```
par(mfrow=c(1,1))
plot(c(0,1),c(mean(Color),mean(Standard)),xlim=c(-.1,1.1),ylim=c(35.2,38.5),
ylab="Mean Time",pch=16,xlab="Type",main="Main Effects Plot")


lines(c(0,1),c(mean(Color),mean(Standard)))
text(c(0,1),c(35.2,35.2),c("Color","Standard"))
```

22. Residual for the 20th observation from the standard group:

```
Standard[20]-mean(Standard)
```

23. Obtain the results (F-statistic and p-value) of an analysis of variance:

```
summary(aov(Time~Type,data=games1))
```

25. sqrt(5.2268)

26. Create a vector of residuals from the analysis of variance model:

```
residuals <- aov(Time~Type,data=games1)$res
```

Construct a histogram of the residuals, a plot of the residuals versus type of game, and a plot of the residuals versus order (all in one graphics window):

```
par(mfrow=c(1,3))
hist(residuals,xlab="Residuals")
plot(games1$Type,residuals,ylab="Residuals")
plot(games1$studentID,residuals,xlab="Order",ylab="Residuals")
lines(games1$studentID,residuals)
abline(h = 0)
```

Extended Activities

28. c) First, create a vector of the observations given in the question:

```
obs <- c(14,11,17,15,13)
```

Construct a normal probability plot of `obs`:

```
par(mfrow=c(1,1))
qqnorm(obs)
```

Note that in R, the normal probability plot is constructed so that the percentiles of the standard normal distribution are on the horizontal axis, and the values of the sample are on the vertical axis.

29. Read data set `Normal` into your session and rename it `normal`:

```
normal <- read.table("FILE PATH/Normal.txt",header=TRUE)
```

a) Create a histogram and normal probability plot for the first column of the `normal`:

```
par(mfrow=c(1,2))
hist(normal[,1])
qqnorm(normal[,1])
qqline(normal[,1])
```

b) Column two of `normal` contains the same values in column one, except the five largest values have been doubled. Create a histogram and normal probability plot for the second column of `normal`:

```
par(mfrow=c(1,2))
hist(normal[,2])
qqnorm(normal[,2])
qqline(normal[,2])
```

c) Column three of normal contains the same values in column one, except the five smallest values have been doubled. Create a histogram and normal probability plot for the third column of normal:

```
par(mfrow=c(1,2))
hist(normal[,3])
qqnorm(normal[,3])
qqline(normal[,3])
```

30. a) Create a normal probability plot of the residuals from the `games1` data in Question 26.

```
residuals <- aov(Time~Type,data=games1)$res
qqnorm(residuals)
qqline(residuals)
```

b-c) The following code can be run to construct normal probability plots for 9 random samples of size 40 observations taken from a standard normal distribution:

```
par(mfrow=c(3,3))
for (i in 1:9) {
samp <- rnorm(40)
qqnorm(samp)
}

# In Rstudio, you may need to enlarge the plot area before you can create 9
plots in one window. Or you can create 9 individual plots by using
par(mfrow=c(1,1)).
```

31. Read in data set `Emission`:

```
Emission <- read.csv("~/MAT310/Clean csv data/C2 Emission.csv")
```

a) Side-by-side boxplots of Emission level by year:

```
par(mfrow=c(1,1))
boxplot(emission~Year,data=Emission)
```

Mean and standard deviation of hydrocarbon Emission levels by year:

Pre63:

```
mean(Emission[Emission$Year=="Pre63",]$emission)
sd(Emission[Emission$Year=="Pre63",]$emission)
```

Yr63to67:

```
mean(Emission[Emission$Year=="Yr63to67",]$emission)
sd(Emission[Emission$Year=="Yr63to67",]$emission)
```

Yr68to69:

```
mean(Emission[Emission$Year=="Yr68to69",]$emission)
```

```
sd(Emission[Emission$Year=="Yr68to69",]$emission)
```
Yr70to71:
```
mean(Emission[Emission$Year=="Yr70to71",]$emission)
sd(Emission[Emission$Year=="Yr70to71",]$emission)
```
Yr72to74:
```
mean(Emission[Emission$Year=="Yr72to74",]$emission)
sd(Emission[Emission$Year=="Yr72to74",]$emission)
```

b) Take a log transformation of the Emission variable, and append it to the original Emission data set:
```
log.em <- log(Emission$emission)
Emission <- cbind(Emission,log.em)
```

Side-by-side boxplots of log-Emission level by year:
```
par(mfrow=c(1,1))
boxplot(log.em~Year,data=Emission)
```
Mean and standard deviation of log-Emission levels by year:
Pre63:
```
mean(Emission[Emission$Year=="Pre63",]$log.em)
sd(Emission[Emission$Year=="Pre63",]$log.em)
```
Yr63to67:
```
mean(Emission[Emission$Year=="Yr63to67",]$log.em)
sd(Emission[Emission$Year=="Yr63to67",]$log.em)
```
Yr68to69:
```
mean(Emission[Emission$Year=="Yr68to69",]$log.em)
sd(Emission[Emission$Year=="Yr68to69",]$log.em)
```
Yr70to71:
```
mean(Emission[Emission$Year=="Yr70to71",]$log.em)
sd(Emission[Emission$Year=="Yr70to71",]$log.em)
```
Yr72to74:
```
mean(Emission[Emission$Year=="Yr72to74",]$log.em)
sd(Emission[Emission$Year=="Yr72to74",]$log.em)
```

c) ANOVA summary table with F-test statistic and p-value:
```
summary(aov(log.em~Year,data=Emission))
```

32. Read in the Weight data:
```
weight <- read.table("FILE PATH/Weight.txt",header=TRUE,sep="\t")
```
Follow the instructions for Question 33 to complete Parts a & b for Question 32.

33. Read in the `RegTrans` data and rename it `regtrans`:
```
regtrans <- read.table("FILE PATH/RegTrans.txt",header=TRUE)
```

Parts a and b are discussed for the first data set in the `regtrans` file. These parts can easily be replicated for the remaining data sets in the `regtrans` file.

Fit the regression model of `Y1` on `X1`:
```
reg1 <- lm(regtrans[,2]~regtrans[,1])
```

a) Scatterplot of `X1` versus `Y1` with a superimposed regression line:
```
plot(regtrans[,1],regtrans[,2])
abline(reg1$coef)
```

Create vector of residuals from fitted regression model:
```
residuals <- reg1$res
```

Plot of residuals versus explanatory variable:
```
plot(regtrans[,1],residuals,xlab="X")
```

Plot of the residuals versus fitted values:
```
plot(reg1$fit,residuals,xlab="Fitted Values")
```

Construct a normal probability plot of the residuals:

```
residuals <- lm(regtrans[,2]~regtrans[,1])$res
qqnorm(residuals)
```

c) Based on the plot of the residuals versus the explanatory variable, we'll try a quadratic transformation of the explanatory variable:

```
x.sq <- regtrans[,1]^2
```

Fit a quadratic regression model that includes the lower order and quadratic terms:

```
quad1 <- lm(regtrans[,2]~regtrans[,1]+x.sq)
```

Now let's examine residual plots of the quadratic model. Obtain vector of residuals from the quadratic model: `residuals <- quad1$res`

Plot of the residuals versus fitted values:

```
plot(quad1$fit,residuals,xlab="Fitted Values")
```

Construct a normal probability plot of the residuals:

```
qqnorm(residuals)
qqline(residuals)
```

34. Compute the p-value using the t-distribution in R, assuming a test statistic equal to 2.2862 with 38 degrees of freedom:

```
2*(1-pt(2.2862,38))
```

35. Follow the instructions provided for Question 34 to find the p-value.

37. To obtain the ANOVA summary table and extract the various sums of squares and mean squares:

```
summary(aov(Time~Type,data=games1))
```

38. (Sample) variance of the completion times:

```
var(games1$Time)
```

Then the total sum of squares is:

```
(40-1)*var(games1$Time)
```

39. Use R to find the p-value of the ANOVA F-test, assuming a test statistic of 5.2268, *Type df* = 1, and *Error df* = 38:

```
1-pf(5.2268,1,38)
```

40. Based on the t-test output in R, find the standard error of the regression coefficient. First, create an R object containing the t-test output:

```
t.test.games <- t.test(Time~Type,data=games1,var.equal=TRUE)
```

Then the standard error of the regression coefficient can be found by:

```
(t.test.games$estimate[1]-t.test.games$estimate[2])/t.test.games$statistic
```

Exercises

E.12. Read in data set Hodgkins:

```
hodgkins <- read.table("FILE PATH/Hodgkins.txt",header=TRUE)
```

a) Side-by-side boxplots of bradykininogen levels by subject type:

```
par(mfrow=c(1,1))
boxplot(Blevel~Type,data=hodgkins)
```

Mean and standard deviation of bradykininogen levels by subject type:

Normal Subject:

```
mean(hodgkins[hodgkins$Type=="Normal",]$Blevel)
sd(hodgkins[hodgkins$Type=="Normal",]$Blevel)
```

Active Subject:

```
mean(hodgkins[hodgkins$Type=="Active",]$Blevel)
sd(hodgkins[hodgkins$Type=="Active",]$Blevel)
```

Inactive Subject:

```
mean(hodgkins[hodgkins$Type=="Inactive",]$Blevel)
sd(hodgkins[hodgkins$Type=="Inactive",]$Blevel)
```

b) Normal probability plot of the residuals:

```
residuals <- aov(Blevel~Type,data=hodgkins)$res
qqnorm(residuals)
qqline(residuals)
```

c) Take a log transformation of the `Blevel` variable, and append it to the original `Emission` data set:

```
log.blevel <- log(hodgkins$Blevel)
hodgkins <- cbind(hodgkins,log.blevel)
```

Side-by-side boxplots of log-Blevel by subject type:

```
par(mfrow=c(1,1))
boxplot(log.blevel~Type,data=hodgkins)
```

Mean and standard deviations of the log-transformed levels can be created by following the code of previous questions.

d) Normal probability plot of the residuals from the ANOVA with transformed response:

```
residuals <- aov(log.blevel~Type,data=hodgkins)$res
qqnorm(residuals)
```

e) ANOVA summary results of model using log-transformed response variable:

```
summary(aov(log.blevel~Type,data=hodgkins)
```

Chapter 3
Multiple Regression: How Much is Your Car Worth?

```
install.packages('mosaic')
require(mosaic)
install.packages('leaps')
require(leaps)
install.packages('graphics')
require(graphics)

##Packages lattice and dplyr are automatically added with mosaic
```

Activities

1. Read in the `cars` data set, and attach the data to your R session:

```
C3.Cars <- read.table("FILE PATH/C3.Cars.txt",header=TRUE)
attach(C3.Cars)
```

Create a scatterplot between `Mileage` and `Price`:

```
mplot(C3.Cars)# then select Mileage as the x variable and Price as the y variable.
```

Or

```
xyplot( Price ~ Mileage, data=C3.Cars, main="")
```

2. Create a "regression" object, named `cars.lm`, that will contain summary information about the fitted regression model of `Price` on `Mileage`:

```
cars.lm = lm(C3.Cars$Price~C3.Cars$Mileage)
```

The estimated coefficients, R^2, t-statistics, and other results about the model can be extracted from `cars.lm` using the `summary()`command:

```
summary(cars.lm)
anova(cars.lm) # anova table also is useful
```

3. Create a vector of residuals from the `cars.lm` model:

```
resid1 <- cars.lm$res
```

residuals can be found using cars.lm$res or residuals(cars.lm)
fitted/predicted values can be found using fitted(cars.lm)

Residual for the first car (Buick Century):

```
resid1[1]  # to show the first element of the vector of residuals
```

It is often helpful to attach the residuals to the cars data frame.

```
Cars = mutate(C3.Cars, resid1)
Cars[1,] #to show the first row of the Cars data frame
```

4. a-b) Follow the instructions provided for Question 2. If you want to have two variables in your model you can use a command similar to regr.model = lm(y ~ a + b)

c) Note that the stepwise regression method implemented in R does not use the improvement in R^2 as the basis for adding a new variable to the model. It uses a criterion called AIC. Models with smaller AIC values are preferred. To perform stepwise regression for the cars data in R, first create a regression object that contains all relevant explanatory variables:

```
full.model= lm(Price~Cyl+Liter+Doors+Cruise+Sound+Leather+Mileage, data =
Cars)
```

Now use the step function, and store the results in the object `step.model`:

```
step.model <- step(full.model)
```

To see the suggested final model, apply the summary command to `step.model`:

```
summary(step.model)
```

Note that the final model left out one of the original seven variables.

5. Best subsets regression requires loading the R package `leaps`. The function that will be used to perform best subsets regression is also called `leaps`. The following code can be used to select the "best" models based on the lowest value of Cp. You can find additional information about various options for the leaps function by using the help function: `help(leaps)`.

First, create a "design matrix" whose columns contain the values of the explanatory variables of interest:

```
require(leaps)
full.data=cbind(Cars$Mileage,Cars$Cyl,Cars$Liter,Cars$Doors,Cars$Cruise,Cars$Sound,Cars$Leather)
head(full.data)
```

Now use the leaps function, and store the results in the object `best.lm`:

```
best.lm <- leaps(full.data,Cars$Price, method = "adjr2", names = c('Mileage','Cyl','Liter','Doors','Cruise','Sound','Leather'), nbest=3)

data1 = cbind(best.lm$which,best.lm$adjr2)
```

To identify the best models, examine the output of the output matrix we have called data1.
Each row of the matrix provides a value of `1` if the explanatory variable was included, and a value of `0` if it was not. For example, the best model with one explanatory variable contains the predictor `Cyl`, and has an adjusted R-squared value of 0.323.

```
> head(data1)
  Cyl Liter Doors Cruise Sound Leather Mileage
1   1     0     0      0     0       0       0 0.32301597
1   0     1     0      0     0       0       0 0.31066830
1   0     0     0      1     0       0       0 0.18461759
1   0     0     0      0     0       1       0 0.02349478
1   0     0     0      0     0       0       1 0.01924208
1   0     0     1      0     0       0       0 0.01802859
```

7. We will use the regression model that contains all explanatory variables except `Liter`:

```
cars.lm2 <- lm(Price~Cyl+Doors+Cruise+Sound+Leather+Mileage)
```

Create a vector of predicted (fitted) values of `Price`:

```
fit2 <- cars.lm2$fitted
```

Create a vector of residuals:

```
resid2 <- cars.lm2$res
```

Construct plots of residuals versus fitted values, and plots of residuals versus each of the explanatory variables in the model (each plot with a horizontal line through Y=0):

```
Cars = mutate(Cars, resid2, fit2)
mplot(Cars)
```

or you can make several plots using the following commands:

```
par(mfrow=c(2,4))
plot(fitted.value,residuals)
abline(h = 0)
plot(Cyl,residuals)
abline(h = 0)
plot(Doors,residuals)
abline(h = 0)
plot(Cruise,residuals)
```

```
abline(h = 0)
plot(Sound,residuals)
abline(h = 0)
plot(Leather,residuals)
abline(h = 0)
plot(Mileage,residuals)
abline(h = 0)
```

NOTE the difference in order for plots: xyplot(y~x) however plot(x,y)

8. Create regression models with the log-transformed (log base 10) and square root-transformed `Price`:

```
cars.log <- lm(log10(Price)~Cyl+Doors+Cruise+Sound+Leather+Mileage, data =
Cars)
cars.sqrt <- lm(sqrt(Price)~Cyl+Doors+Cruise+Sound+Leather+Mileage, data =
Cars)
```

Results of the new models can be obtained using the `summary()` command, and residual plots can be constructed following code similar to that provided in Question 7. Remember to name the vectors of residuals from the transformed-response models something different from "residuals" that was used in Question 7 (otherwise you will over-write the original vector of residuals).

9. Construct a simple linear regression model of log `Price (henceforth called TPrice)` on `Mileage:`

```
cars.log2 <- lm(log10(Price)~Mileage, data = Cars)
```

Create a vector of residuals for the transformed response model:

```
res.log2 <- cars.log2$res
```

Construct a plot of the residuals versus time/order (with a line connecting the residuals):

```
par(mfrow=c(1,1))
plot(seq(1:length(res.log2)),res.log2,xlab="Order")
lines(seq(1:length(res.log2)),res.log2)
```

10. After creating a regression model using the log-transformed response model that included the 6 explanatory variables selected from Question 5 (henceforth called the `TPrice` model), make a vector of residuals from

```
res.log <- cars.log$res
```

Construct a plot of the residuals versus time/order:

```
plot(seq(1:length(res.log)),res.log,xlab="Order")
lines(seq(1:length(res.log)),res.log)
```

11. Create vector of fitted values from the `TPrice` model:

```
cars.log.fit <- cars.log$fit
```

Plots of residuals versus fitted values and `Mileage`:

```
par(mfrow=c(1,2))
plot(cars.log.fit,res.log,xlab="Fitted Values")
# text(cars.log.fit,res.log,labels=seq(1:length(Mileage)))

# The labeling is not very readable


plot(Mileage,res.log,xlab="Mileage")
```

a) To identify the row numbers that correspond to the points on the plot of the residuals versus fitted values:

```
par(mfrow=c(1,1))
```

Create a dataframe with the appropriate residuals.

```
Cars2 = mutate(Cars, res.log)
Cars2 = arrange(Cars2, res.log)
tail(Cars2,10)  # shows the 10 rows with the largest residuals.
# which Make and Type correspond to these values?
```

```
Or use mplot() and color the data by Make and Type
```

12. Fit the `TPrice` model without the observations corresponding to the largest cluster of outliers, i.e. without the convertible Cadillacs (observations 151-160):
NewCars = subset(Cars,c(Make!="Cadillac" & Type!="Convertible"))

```
cars.log3 <- lm(log10(Price)~Cyl+Doors+Cruise+Sound+Leather+Mileage,data
= NewCars)
```

Use the `summary()`command to examine the estimated coefficients. Compare these estimated coefficients to those from the model with all the observations.

13. Construct a simple linear regression model of log `Price` on `Mileage`:

```
cars.log2 <- lm(log10(Price)~Mileage)
```

Create a vector of residuals for the transformed response model:

```
res.log2 <- cars.log2$res
```

Construct a histogram and normal probability plot of the residuals from the simple model of `TPrice` regressed on `Mileage` model:

```
par(mfrow=c(1,2))
hist(res.log2)
qqnorm(res.log2)
```

14. Results for the following models can be obtained using the `summary()`command.

a) Fit the regression model of `Price` on `Mileage` and `Liter`:

```
model1 <- lm(Price~Mileage+Liter, data = Cars)
```

b) Fit the regression model of `Price` on `Cyl`:

```
model2 <- lm(Price~Mileage + Cyl, data = Cars)
```

c) Fit the regression model of `Price` on `Mileage`, `Liter`, and `Cyl`:

```
model3 <- lm(Price~Mileage+Liter+Cyl, data = Cars)
```

16. Plot `Cyl` versus `Liter`:

```
par(mfrow=c(1,1))
plot(Cars$Liter,Cars$Cyl)
```

Correlation between `Cyl` and `Liter`:

```
cor(Cars$Liter,Cars$Cyl)
```

17. Define the log-transformed response variable, `TPrice`:

```
TPrice <- log10(Cars$Price)
```

Construct boxplots of `TPrice` versus the categorical explanatory variables `Make, Model, Trim,` and `Type`:

```
par(mfrow=c(2,2))
bwplot(Cars$TPrice~Cars$Make)
bwplot(Cars$TPrice~Cars$Model)
bwplot(Cars$TPrice~Cars$Trim)
bwplot(Cars$TPrice~Cars$Type)
```

18. Create indicator variables for the categorical variable Make:

```
It is possible to create categorical variables in R, but it is not needed in
regression models.
    DM1 <- (Make=="Buick")*1
    DM2 <- (Make=="Cadillac")*1
    DM3 <- (Make=="Chevrolet")*1
```

```
DM4 <- (Make=="Pontiac")*1
DM5 <- (Make=="SAAB")*1
DM6 <- (Make=="Saturn")*1
```

19. Construct a linear regression model with response variable `TPrice` and explanatory variables `Mileage,`
```
cars.log4 <- lm(TPrice~Mileage+Liter+Make, data = Cars)
```
or
```
cars.log4 <- lm(TPrice~Mileage+Liter+DM2+DM3+DM4+DM5+DM6, data = Cars)
```

Examine the results of the model using the `summary()` command.

20. Create indicator variables for the categorical variable `Type`:
```
cars.log5 <- lm(TPrice~Mileage+Liter+Make+Type, data = Cars)
```

Examine the results of the model using the `summary()` command.

Create a vector of residuals from this model:
```
res.log5 <- cars.log5$res
```

Residual plots can be constructed using instructions in previous problems.

21. Follow the instructions provided for previous activities.

22. Create a subset of the cars data set corresponding to the Chevrolet Cavalier Sedan models:
```
cav <- Cars[Model=="Cavalier",]
cav <- cav[cav$Type=="Sedan",]
```

Create a regression model to predict to Price from Mileage using the Cavalier data:
```
cav.lm <- lm(Price~Mileage,data=cav)
```

Obtain the sums of squares:
```
anova(cav.lm)
```
.

Extended Activities

23. Create a vector of values of y-y_hat:
```
res <- cav.lm$res
```

Create a vector of values of y_hat-y_bar:
```
reg <- cav.lm$fit-mean(cav$Price)
```

Examine the result of:
```
sum(res*reg)
```

30. For the Cavalier data, fit the regression model to predict `Price` from `Mileage` and `Cruise`:
```
cav.lm2 <- lm(Price~Mileage+Cruise,data=cav)
```

Obtain the F-statistic and p-value using the `summary()`command:
```
summary(cav.lm2)
```

31. Create indicator variables for the categorical variable `Trim` for the `cav` data set (leave out LS Sedan 4D):
```
DTr1 <- (cav$Trim=="LS Sport Sedan 4D")*1
DTr2 <- (cav$Trim=="Sedan 4D")*1
```

Construct the full regression model that includes `Mileage, Cruise,` and `Trim`:
```
cav.lm3 <- lm(Price~Mileage+Cruise+DTr1+DTr2,data=cav)
```

To test whether `Trim` is useful, use the `anova()` command to compare the full model `cav.lm3` to the reduced model `cav.lm2`:

```
anova(cav.lm3,cav.lm2)
```

35. Read the data set `4-8Cyl` into your R session:

```
cyl48 <- read.table("FILE PATH/4-8Cyl.txt",header=TRUE,sep="\t")
```

Construct regression model of `Price` on `Mileage` and `Cyl`:

```
cyl.lm1 <- lm(Price~Mileage+Cyl,data=cyl48)
```

Construct regression model of `Price` on `Mileage, Cyl`, and the interaction between `Mileage` and `Cyl`:

```
cyl.lm2 <- lm(Price~Mileage + Cyl +Mileage*Cyl,data=cyl48)
```

a) Use the `summary()` command for each model to determine whether the R^2_adj increased.

b)

```
fit <- makeFun(cyl.lm1)
fit(Mileage=10000, Cyl=4, data = cyl48)
```

```
repeat for cyl.lm2
```

c)

```
fit <- makeFun(cyl.lm1)
fit(Mileage=10000, Cyl=4, data = cyl48)
```

```
repeat for cyl.lm2
```

d) Use the `anova()` command and follow the code in Question 31 to determine whether the interaction term should be included in the model.

```
anova(cyl.lm1, cyl.lm2)
```

36. Create indicator variables for the categorical variable `Make` for the `cyl48` data set (leave out Pontiac):

```
cyl48 =mutate(cyl48, Cadillac = (cyl48$Make=="Cadillac")*1, SAAB =
(cyl48$Make=="SAAB")*1)
```

```
head(cyl48)
```

Construct regression model of `Price` on `Mileage, Cadillac,` and SAAB:

```
cyl.lm3 <- lm(Price~Mileage+Cadillac+SAAB,data=cyl48)
```

a) Scatterplot of `Price` versus `Mileage` by `Make`:

```
xyplot(cyl48$Price~cyl48$Mileage, groups = cyl48$Make,
xlab="Mileage",ylab="Price", main = "Additive Regression Model",
auto.key=list(space="right"))
fit <- makeFun(cyl.lm3)
plotFun( fit(Mileage, Cadillac=1, SAAB=0) ~ Mileage, add=TRUE, col=1)
plotFun( fit(Mileage, Cadillac=0, SAAB=0) ~ Mileage, add=TRUE, col=2)
plotFun( fit(Mileage, Cadillac=0, SAAB=1) ~ Mileage, add=TRUE, col=3)
```

b) Construct regression model of `Price` on `Mileage, Cadillac,` SAAB, and the interaction terms between `Mileage` and `Cadillac`, and `Mileage` and SAAB:

```
cyl.lm4 <- lm(Price~Mileage + Cadillac+ SAAB + Cadillac*Mileage +
SAAB*Mileage,data=cyl48)
```

Use the `anova()` command to compare models `cyl.lm3` and `cyl.lm4` to determine if the interaction terms are significant.

```
anova(cyl.lm3,cyl.lm4)
```

Scatterplot of `Price` versus `Mileage`:

```
xyplot(cyl48$Price~cyl48$Mileage, groups = cyl48$Make,
xlab="Mileage",ylab="Price", main = "Regression Model with Interaction Terms",
auto.key=list(space="right"))
```

Superimpose regression lines for the Cadillacs and SAAB's. First obtain the estimated coefficients from the interaction model:

```
fit <- makeFun(cyl.lm4)
plotFun( fit(Mileage, Cadillac=1, SAAB=0) ~ Mileage, add=TRUE, col=1)
plotFun( fit(Mileage, Cadillac=0, SAAB=0) ~ Mileage, add=TRUE, col=2)
plotFun( fit(Mileage, Cadillac=0, SAAB=1) ~ Mileage, add=TRUE, col=3)
```

37. Read the `MPG` data into your R session:

```
mpg <- read.table("FILE PATH/MPG.txt",header=TRUE)
```

Create the regression model to predict `mpg` from `speed` and `displacement`:

```
mpg.lm <- lm(mpg~speed+displacement,data=mpg)
```

a) Use the `summary()` command to examine results of the fitted model.

b) Create a vector of residuals from the model:

```
res.mpg <- mpg.lm$res
```

Construct the plots of residuals versus `speed` and residuals versus `displacement`:

```
par(mfrow=c(1,2))
plot(mpg$speed,res.mpg)
plot(mpg$displacement,res.mpg)
```

c) Normal probability plot of the residuals:

```
par(mfrow=c(1,1))
qqnorm(res.mpg)
```

38. Create the regression model to predict mpg from `speed`:

```
mpg.lm2 <- lm(mpg~speed,data=mpg)
```

a) Use the `summary()` command to get the coefficients and R^2 values.

b) Create a vector of residuals from the model `mpg.lm2`:

```
res.mpg <- mpg.lm2$res
```

Construct the plots of residuals versus `speed` and residuals versus `displacement`:

```
par(mfrow=c(1,2))
plot(mpg$speed,res.mpg)
plot(mpg$displacement,res.mpg)
```

c) Normal probability plot of the residuals:

```
par(mfrow=c(1,1))
qqnorm(res.mpg)
```

39. Create the regression model to predict mpg from `displacement:`

```
mpg.lm3 <- lm(mpg~displacement,data=mpg)
```

a) Use the `summary()` command to get the coefficients and R^2 values.

b) Create a vector of residuals from the model `mpg.lm3`:

```
res.mpg <- mpg.lm3$res
```

Construct the plots of residuals versus `speed` and residuals versus `displacement`:

```
par(mfrow=c(1,2))
```

```
plot(mpg$speed,res.mpg)
plot(mpg$displacement,res.mpg)
```

40. Create a quadratic term (`speedsq`) for `speed`:
```
speedsq <- mpg$speed^2
```

Create the quadratic regression model to predict `mpg` from `speed, displacement,` and `speedsq`:
```
mpg.lm4 <- lm(mpg~speed+displacement+speedsq,data=mpg)
```

a) Use the `summary()` command to get the coefficients and R^2 values.

b) Create a vector of residuals from the model `mpg.lm4`:
```
res.mpg <- mpg.lm4$res
```

Construct the plots of residuals versus `speed` and residuals versus `displacement`:
```
par(mfrow=c(1,2))
plot(mpg$speed,res.mpg)
plot(mpg$displacement,res.mpg)
```

c) Normal probability plot of the residuals:
```
par(mfrow=c(1,1))
qqnorm(res.mpg)
```

41. If you are working in a new R session, you will need to read in the `cars` data again; be sure to attach it to your session with the `attach()` command, so that the following code will work. You will also have to create the `TPrice` (log-transformed Price) variable again.
Create quadratic term, `Mileagesq`, for `Mileage`:
```
Mileagesq <- Mileage^2
```

Regression model to predict `TPrice` from `Mileage`:
```
cars.lm1 <- lm(TPrice~Mileage)
```

Regression model to predict `TPrice` from `Mileage` and `Mileagesq`:
```
cars.lm2 <- lm(TPrice~Mileage+Mileagesq)
```

a) Use the `summary()` command to determine the increase in R^2 between the two models.

b) Follow the instructions provided for previous activities to construct residual plots.

42. Create the interaction between `Mileage` and `Cyl`:
```
MileCyl <- Mileage*Cyl
```

Regression model to predict `TPrice` from `Mileage, Cyl,` and `MileCyl`:
```
cars.lm3 <- lm(TPrice~Mileage+Cyl+MileCyl)
```

Follow the instructions provided for previous activities to determine if the interaction term improves the model, and to create residual plots.

43. Follow the instructions provided for previous activities

Chapter 4 R Instructions
Designing Factorial Experiments: Microwave Popcorn

We suggest that you install the following packages before running the following code:

```
install.packages('mosaic')
require(mosaic)
```

Activities

4. Read the `Popcorn` data set into your R session and attach it:

```
popcorn <- read.csv("~/MAT295/Clean csv data/C4 Popcorn.csv", header=TRUE)
attach(popcorn)
```

Compute descriptive statistics by group:

```
mean(PopRate[Brand=="Fastco"])
mean(PopRate[Brand=="Pop Secret"])
mean(PopRate[Time==105])
mean(PopRate[Time==135])
mean(PopRate)
```

If you are using the mosaic package:

```
mean(PopRate~Brand)
mean(PopRate~Time)
mean(PopRate)
mean(PopRate~Brand+Time)
```

You could also use:

```
favstats(PopRate~Brand+Time)
```

5. The average popping rates for each level of `Brand` and `Time`, as well as overall average `PopRate`:
If you are using the mosaic package:

```
favstats(PopRate~Brand+Time)
mean(PopRate~Brand+Time)
```

Find the difference between average PopRate for the cooking times for each brand:

```
mean(PopRate[(Brand=="Fastco") & (Time==135)])-
mean(PopRate[(Brand=="Fastco") & (Time==105)])

mean(PopRate[(Brand=="Pop Secret") & (Time==135)])-
mean(PopRate[(Brand=="Pop Secret") & (Time==105)])
```

13. The two-way ANOVA is conducted using the `aov()` command. We will store the ANOVA results in the object `pop.aov1`:

```
pop.aov1 <- aov(PopRate~Brand+Time+Brand:Time)
```

Use the `summary()` command to obtain the ANOVA table:

```
summary(pop.aov1)
```

16. Create individual value plots of the eight factor-level groups:

In order to do anova, we have to make Time into a categorical variable

#Thus, using as.factor we can transform the data, and we need to use Time1 in the next several questions

```
Time1 <- as.factor(Time)
xyplot(PopRate~Brand:Time1:Microwave)
```

Labels are difficult to see without zooming in. Instead you could use mplot to see individual groups

```
mplot(popcorn)
```

d) Create a vector of residuals from the 3-Way ANOVA model:

```
res <- aov(PopRate~Brand+Time+Microwave+Brand:Microwave+Time1:Microwave)$res
```

Now construct a normal probability plot of the residuals:

```
qqnorm(res)
qqline(res)
```

17. Calculate the eight group standard deviations:

We have to use Time1 to use tapply

```
tapply(PopRate, Brand:Time1:Microwave, sd)
```

To find the maximum and minimum standard deviations:

```
max(tapply(PopRate, Brand:Time1:Microwave, sd))
min(tapply(PopRate, Brand:Time1:Microwave, sd))
```

18. a) The following ANOVA object will contain the results of all six hypotheses:

```
pop.3way.aov <- aov(PopRate ~ Brand + Time + Microwave + Brand:Time + Brand:Microwave +
Time:Microwave)
```

Display the results of the tests using the summary() command:

```
summary(pop.3way.aov)
```

Extended Activities

25. Read in the PaperTowels data and attach it to your R session:

```
paper <- read.csv("~/MAT295/Clean csv data/C4 PaperTowels.csv",header=TRUE)
attach(paper)
```

Calculate means by factor level:

```
meanD <- mean(paper[Brand=="Decorator",]$Strength)
meanC <- mean(paper[Brand=="Comfort",]$Strength)
mean0 <- mean(paper[as.factor(Water)=="0",]$Strength)
mean5 <- mean(paper[as.factor(Water)=="5",]$Strength)
mean15 <- mean(paper[as.factor(Water)=="15",]$Strength)
```

Construct the main effects plot:

```
par(mfrow=c(1,2))
plot(c(1,2),c(0,3500),xaxt="n",type="n",xlab="Brand",
   ylab="Paper Towel Breaking Strength (grams)",
   main="Main Effects Plot: Brand")
axis(1,at=c(1:2),labels=c("Comfort","Decorator"))
points(c(1,2),c(meanC,meanD),pch=".",cex=3)
lines(c(1,2),c(meanC,meanD),lty=1)
```

```
plot(c(1,2,3),c(0,1500,3500),xaxt="n",type="n",xlab="Water",
   ylab="Paper Towel Breaking Strength (grams)",
   main="Main Effects Plot: Water")
axis(1,at=c(1:3),labels=c("0 Drops","5 Drops","15 Drops"))
points(c(1,2,3),c(mean0,mean5,mean15),pch=".",cex=3)
lines(c(1,2,3),c(mean0,mean5,mean15),lty=1)
```

28. Construct the interaction plot:

```
par(mfrow=c(1,1))
plot(c(1,2),c(0,3500),xaxt="n",type="n",xlab="Brand",
   ylab="Paper Towel Breaking Strength (grams)",main="Interaction Plot")
axis(1,at=c(1:2),labels=c("Comfort","Decorator"))
points(c(1,2),c(3205.7692,2219.2308),pch=".",cex=3)
lines(c(1,2),c(3205.7692,2219.2308),lty=1)
points(c(1,2),c(1711.5385,704.8077),pch=".",cex=3)
lines(c(1,2),c(1711.5385,704.8077),lty=2)
points(c(1,2),c(400.9615,446.1538),pch=".",cex=3)
lines(c(1,2),c(400.9615,446.1538),lty=3)
legend(1.6,3500,c("0 Drops","5 Drops","15 Drops"),lty=1:3)
```

35. b) Calculate the six group standard deviations:

```
tapply(Strength, Brand:as.factor(Water), sd)
```

c) Calculate the factor-level group means:

```
tapply(Strength, Brand:as.factor(Water), mean)
```

d) Transform the response variable using the natural log and square root transformations:

```
ln.strength <- log(Strength)
sqrt.strength <- sqrt(Strength)
```

Now check the equal variances assumption by first finding the standard deviations of the transformed response for the 6 factor-level groups:

```
tapply(ln.strength, Brand:as.factor(Water), sd)
tapply(sqrt.strength, Brand:as.factor(Water), sd)
```

36. Construct boxplots for the log and square root transformed responses for the 6 factor-level groups:

```
plot(Brand:as.factor(Water),ln.strength)
plot(Brand:as.factor(Water),sqrt.strength)
```

37. Perform the two-way ANOVA on the square root transformed paper towel data, and store results in the object, aov.paper:

```
aov.paper <- aov(sqrt.strength~Brand+as.factor(Water)+Brand:as.factor(Water))
```

Examine the ANOVA summary table to obtain the F-statistics and p-values for the appropriate hypothesis tests:

```
summary(aov.paper)
```

Create a vector of residuals from the model:

```
res <- aov.paper$res
```

Construct a normal probability plot of the residuals to investigate the normality assumption.

```
qqnorm(res)
qqline(res)
```

Chapter 5 R Instructions
Block, Split-Plot and Repeated Measure Designs: What Influences Memory?

Activities

1. Read in and attach the Memory data into your R session:

```
memory <- read.table("FILE PATH/Memory.txt",header=TRUE,sep="\t") attach(memory)
```

Analyze as if data were from a randomized factorial design:

```
wrongCRD <- aov(Score ~ Word.List*Distracter)
```

Examine the results using the summary() command:

```
summary(wrongCRD)
```

b) Create side-by-side boxplots:

```
boxplot(Score ~ reorder(Word.List:Distracter, Score, median), ylab="Avg
Score (Recalled Words out of 20)")
```

Examine the standard deviations of the groups to investigate the equal variances assumption:

```
tapply(Score, Word.List:Distracter, sd)
```

Note: When working with the tapply() function, the syntax Word.List:Distracter should be used when both Word.List and Distracter are categorical variables.

c) Create side-by-side boxplots:

```
boxplot(Score ~ reorder(Word.List:Distracter, Score, median), ylab="Avg
Score (Recalled Words out of 20)")
```

Examine the standard deviations of the groups to investigate the equal variances assumption:

```
tapply(Score, Word.List:Distracter, sd)
```

Note: When working with the tapply() function, the syntax Word.List:Distracter should be used when both Word.List and Distracter are categorical variables.

d) Create vector if residuals from the model:

```
res <- wrongCRD$res
```

Create normal probability plot of residuals:

```
qqnorm(res)
```

Create main effects plots:

```
par(mfrow=c(1,2))
plot(c(1:2),seq(7,9,length=2),xaxt="n",type="n",xlab="Word List",
ylab="Mean Number of Words Recalled")
axis(1,at=c(1:2),labels=c("Abstract","Concrete"))
points(c(1:2),tapply(Score, Word.List, mean))
lines(c(1:2),tapply(Score, Word.List, mean))

plot(c(1:2),seq(7,9,length=2),xaxt="n",type="n",xlab="Distracter",
ylab="Mean Number of Words Recalled") axis(1,at=c(1:2),labels=c("Math","Poetry"))
points(c(1:2),tapply(Score, Distracter, mean))
lines(c(1:2),tapply(Score, Distracter, mean))
```

Create interaction plot:

```
par(mfrow=c(1,1))
interaction.plot(Word.List,Distracter,Score,xlab="Word List",
ylab="Mean Number of Words Recalled")
```

2. Conduct an ANOVA assuming the data are from a block design:

```
block <- aov(Score ~ as.factor(Student) + Word.List*Distracter)
```

Examine the results using the summary() command:

```
summary(block)
```

c) Construct main effects plots:

```
par(mfrow=c(1,3))
plot(c(1:12),seq(5,11,length=12),xaxt="n",type="n",xlab="Student",
ylab="M an Number of Words Recalled")
axis(1,at=c(1:12),labels=c(1:12))
points(c(1:12),tapply(Score, as.factor(Student), mean))
lines(c(1:12),tapply(Score, as.factor(Student), mean))

plot(c(1:2),seq(5,11,length=2),xaxt="n",type="n",xlab="Word List",
ylab="Mean Number of Words Recalled")
axis(1,at=c(1:2),labels=c("Abstract","Concrete"))
points(c(1:2),tapply(Score, Word.List, mean))
lines(c(1:2),tapply(Score, Word.List, mean))

plot(c(1:2),seq(5,11,length=2),xaxt="n",type="n",xlab="Distracter",
ylab="Mean Number of Words Recalled")
axis(1,at=c(1:2),labels=c("Math","Poetry"))
points(c(1:2),tapply(Score, Distracter, mean))
lines(c(1:2),tapply(Score, Distracter, mean))
```

d) To create a normal probability plot of residuals, see Question 1d.

8. Conduct an ANOVA, treating the memory data as a split-plot design:

```
splitplot <- aov(Score ~ Major + Word.List*Distracter +
Error(Major/as.factor(Student2)))
```

Examine the results of the analysis:

```
summary(splitplot)
```

To test the significance of the whole-plot factor (Major), compute the F-statistic which is the ratio of the MS whole-plot to MS split-plot, and find the corresponding p-value:

```
F <- 3.1875/8.208
1-pf(F,3,8)
```

To construct a normal probability plot of the residuals, see Question 1d.

Construct main effects plots:

```
par(mfrow=c(1,3))
plot(c(1:4),seq(7,9,length=4),xaxt="n",type="n",xlab="Major",ylab="Mean
Number of Words Recalled")
axis(1,at=c(1:4),labels=levels(memory$Major))
points(c(1:4),tapply(Score, Major, mean))
```

```
lines(c(1:4),tapply(Score, Major, mean))

plot(c(1:2),seq(7,9,length=2),xaxt="n",type="n",xlab="Word List",
ylab="Mean Number of Words Recalled")
axis(1,at=c(1:2),labels=c("Abstract","Concrete"))
points(c(1:2),tapply(Score, Word.List, mean))
lines(c(1:2),tapply(Score, Word.List, mean))
plot(c(1:2),seq(7,9,length=2),xaxt="n",type="n",xlab="Distracter",
ylab="Mean Number of Words Recalled")
axis(1,at=c(1:2),labels=c("Math","Poetry"))
points(c(1:2),tapply(Score, Distracter, mean))
lines(c(1:2),tapply(Score, Distracter, mean))
```

To construct an interaction plot, follow the steps provided for Question 1d.

To construct a Normal probability plot of the residuals, follow the steps provided for Question 1d.

11. Model that incorrectly assumes Student2 is a fixed factor and Student2 is not nested within Major:

```
fixedStu2 <- aov(Score~Major+as.factor(Student2)+Word.List*Distracter)
```

Examine results of incorrect model:

```
summary(fixedStu2)
```

12. Model that incorrectly assumes Student2 is a fixed factor, but correctly assumes Student2 is nested within Major:

```
nestedStu2 <- aov(Score~Major+as.factor(Student2)+Word.List*Distracter+Error(Major/as.factor(Student2)))
```

Examine results of incorrect model:

```
summary(nestedStu2)
```

To test the significance of the Major, compute the F-statistic, and find the corresponding p-value:

```
F <- 3.1875/2.44
1-pf(F,3,33)
```

15. Analyze memory data using a split-plot design:

```
splitplot2 <- aov(Score ~ Major + Word.List*Distracter
Major:Word.List+Major:Distracter+ Error(Major/as.factor(Student2)))  summary(splitplot2)
```

To test the significance of the Major, compute the F-statistic, and find the corresponding p-value:

```
F <- 3.1875/8.208
1-pf(F,3,8)
```

19. To construct a Normal probability plot of the residuals, follow the steps provided for Question 1d.

20. Individual values plot:

```
allcombs <- as.integer(Major:Word.List:Distracter)
plot(allcombs,Score, pch=20, xaxt="n",ylab="Score",
xlab="Factor/Level Group", main="Individual Value Plot")
axis(1,at=c(1:16),labels=c("CS.ab.m","CS.ab.p","CS.c.m","CS.c.p",
"Eng.ab.m",
"Eng.ab.p","Eng.c.m","Eng.c.p","H.ab.m","H.ab.p","H.c.m","H.c.p",
```

```
"Ma.ab.m",
"Ma.ab.p","Ma.c.m","Ma.c.p"),cex.axis=.7,las=2)
```

21. Individual value plot of subject average score by major:

```
mean.student <- tapply(Score, as.factor(Student), mean)
plot(c(1,1,1,2,2,2,3,3,3,4,4,4),mean.student,pch=20,xaxt="n",
ylab="Student Average Score",
xlab="Major") axis(1,at=c(1:4), labels=c("Math","CS","Hist","Eng"))
```

Extended Activities

25. Read in and attach the Flowers data into your R session:

```
flowers <- read.table("FILE PATH/Flowers.txt",header=TRUE)
attach(flowers)
```

Analyze the block design:

```
flower.block <- aov(Days~as.factor(Store)+Water)
```

Examine the results of the model:

```
summary(flower.block)
```

To construct a Normal probability plot of the residuals, follow the steps provided for Question 1d.

Create an individual value plot:

```
allcombs <- as.integer(as.factor(Store):Water)
plot(allcombs,Days, pch=20, xaxt="n",ylab="Days",
xlab="Factor/Level  Group",
main="Individual Value Plot")
axis(1,at=c(1:9),labels=c("1.asp","1.floral","1.plain","2.asp","2.floral",
"2.plain","3.asp","3.floral","3.plain"),cex.axis=.7,las=2)
```

26. Grand mean: `mean(Days)`

Average for each Water level:

```
tapply(Days, as.factor(Water), mean)
```

Water effects can be computed using the grand mean and Water level means calculated above.

31. Read in and attach the Popcorn data into your R session:

```
popcorn <- read.table("FILE PATH/Popcorn.txt",header=TRUE)
attach(popcorn)
```

Conduct an ANOVA on the popcorn data:

```
pop.split <- aov(PopRate~Brand+Temp+Brand:Temp+Error
(Brand/as.factor(Box)))
```

Examine the results:

```
summary(pop.split)
```

To construct a Normal probability plot of the residuals, follow the steps provided for Question 1d.

Create an individual value plot:

```
allcombs <- as.integer(Brand:Temp)
```

```
plot(allcombs,PopRate, pch=20, xaxt="n",ylab="Percent Popped",  xlab="Factor/Level
Group",main="Individual Value Plot")
axis(1,at=c(1:4), labels=c("Exp Frig","Exp Room","Gen Frig","Gen Room"))
```

Create main effects plots:

```
par(mfrow=c(1,2))
plot(c(1:2),seq(83.5,84.5,length=2),xaxt="n",type="n",xlab="Brand",
ylab="Mean Percent Popped")
axis(1,at=c(1:2),labels=c("Expensive","Generic"))
points(c(1:2),tapply(PopRate, Brand, mean))
lines(c(1:2),tapply(PopRate, Brand, mean))
plot(c(1:2),seq(83.5,84.5,length=2),xaxt="n",type="n",
xlab="Temp",ylab="Mean Percent Popped")
axis(1,at=c(1:2),labels=c("Frig","Room"))
points(c(1:2),tapply(PopRate, Temp, mean))
lines(c(1:2),tapply(PopRate, Temp, mean))
```

Create interaction plot:

```
par(mfrow=c(1,1))
interaction.plot(Temp,Brand,PopRate,xlab="Temp",
ylab="Mean Percent  Popped")
```

32. Grand mean: `mean(PopRate)`

Average for each Brand level: `tapply(PopRate, Brand, mean)`

Brand effects can be computed using the grand mean and Brand level means calculated above.

33. Follow the instructions provided for Question 32.

34. Obtain the means for all Brand/Temp conditions:

```
tapply(PopRate,Brand:Temp,mean)
```

Use these means and the those from Questions 32 and 33 to estimate the interaction effects.

41. Read in and attach the Handwash data:

```
handwash <- read.table("FILE PATH/Handwash.txt",header=TRUE,sep="\t")
attach(handwash)
```

Scatterplot of before versus after CFU counts:

```
plot(CFU.Before,CFU.After,xlab="Before CFU Counts",ylab="After CFU  Counts")
```

42. Natural log transform of the explanatory and response variables:

```
ln.before <- log(CFU.Before)
ln.after <- log(CFU.After)
```

Perform the ANCOVA:

```
hand.ancova <- lm(ln.after~ln.before+Cleanser)
```

Examine ANCOVA results:

```
anova(hand.ancova)
```

43. Create a new blocking factor for Before CFU counts:

```
Before.block <- 1:30
for (i in 1:30)
{
if (CFU.Before[i]<52) (Before.block[i]="Low")  if (CFU.Before[i]>51 && CFU.Before[i]<150)
(Before.block[i]="Medium")  if (CFU.Before[i]>150) (Before.block[i]="High")
}
```

Perform ANOVA on the block design:

```
hand.block <- aov(ln.after~as.factor(Before.block)+Cleanser)
```

Examine ANOVA results:

```
anova(hand.block)
```

Chapter 6
Categorical Data Analysis: Is a Tumor Malignant or Benign?

R packages required: no new R packages are required for this chapter.

Activities

3. Create a segmented bar graph for the `tumor` data:

```
tumors <- cbind(c(7/16,9/16),c(17/21,4/21))
barplot(tumors,xlab="Shape",legend=c("Malignant","Benign"),
ylab="Proportion Malignant",names.arg=c("Round","Concave"),
xlim=c(0,3.8))
```

6. Simulation to randomly select 21 cancer cell nuclei and count the number of concave malignant:

```
cancer.cells <- c(rep("M",24),rep("B",13))
sum(sample(cancer.cells,21)=="M")
# This takes a random sample of size 21 and
# counts the number of "M"s
```

The second line of code provides the number of concave cells that are malignant.

7. The following code will provide the number of malignant cells in each of the 9 random sample of 21 cells:

```
nsims <- 9
malig <- 1:nsims
for (i in 1:nsims) {cancer.cells <- c(rep("M",24),rep("B",13))
malig[i] <- sum(sample(cancer.cells,21)=="M")}
malig
```

8. In the code provided for Question 7, set `nsims = 10000`, then construct a histogram of the 10,000 simulated counts:

```
hist(malig)
```

Estimate the p-value:

```
sum(malig>=17)/10000
```

9. Find P(X=17):

```
phyper(17,24,13,21)-phyper(16,24,13,21)
```

In the above code, 24 = number of malignant cells, 13 = number of benign cells, and 21 = number of cells selected at random. By itself, `phyper(17,24,13,21)` gives P(X<=17).

```
# You can also use dhyper to calculate probability directly
dhyper(17,24,13,21)
```

Find P(X=18):

```
phyper(18,24,13,21)-phyper(17,24,13,21)
# dhyper(18,24,13,21) # is an alternative way to find this probability
```

P(X=19), P(X=20), and P(X=21) can be found in a similar manner.

10. Construct a histogram for the hypergeometric probabilities:

```
barplot(phyper(seq(1,21),24,13,21)-phyper(seq(0,20),24,13,21),names.arg
=seq(1,21),ylab="Probability")
```

11. Exact p-value, P(X >=17):

```
1-phyper(16,24,13,21)
```

12. Assume M = 13 successes and take a sample of size 16. Then P(X>=9) is:

```
1-phyper(8,13,24,16)
```

13. Assume M = 13 successes and take a sample of size 21. Then P(X<=4) is:

```
phyper(4,13,24,21)
```

15. Estimated p-value for the two-sided hypothesis:

```
(sum(malig>=17)+sum(malig<=10))/10000
```

16. Find the exact p-value for the two-sided hypothesis using the hypergeometric distribution P(X<=10)+P(X >=17):

```
phyper(10,24,13,21)+(1-phyper(16,24,13,21))
```

19. Conduct the chi-square test for the cancer cell data:

```
cancer <- cbind(c(9,4),c(7,17))
chisq.test(cancer,correct=FALSE)
```

The test statistic and p-value are provided in the output.

Extended Activities

14. Randomly draw a vector of 10000 observations from a hypergeometric distribution:

```
ranvec <- rhyper(10000, 13, 24, 21)
```

Write a function to calculate the chisq statistics:

```
chisq.stat <- function(obs) {
(X.sq = (13 - (21 - obs) - 5.62)^2/5.62 + (24 - obs - 10.38)^2/10.38 + (21
- obs - 7.38)^2/7.38 + (obs - 13.62)^2/13.62)
return (X.sq)}
```

Calculate the chisq statistics for 10000 observations:

```
chisq.vec <- chisq.stat(ranvec)
```

Draw a histogram with specified breaks:

```
hist(chisq.vec, breaks = seq(0, 90,by = 1))
```

34. Conduct the chi-square goodness-of-fit test on the die experiment:

```
die.rolls <- c(2,5,3,3,8,9)
chisq.test(die.rolls,correct=FALSE)
```

E.14

```
cancer <- cbind(c(9,4),c(7,17))
chisq.test(cancer,correct=FALSE)
chisqdist = rchisq(10000,1)
    hist(sampledist, breaks = 25)
hist(sampledist, breaks = 60)
1-pchisq(5.5148,1)

nsims <- 10000
malig <- 1:nsims
for (i in 1:nsims) {
cancer.cells <- c(rep("M",24),rep("B",13))
cell4 = sum(sample(cancer.cells,21)=="M")
cell2 = 21 - cell4
cell3 = 24 - cell4
cell1 = 13 - cell2
cancer <- cbind(c(cell1,cell2),c(cell3,cell4))
malig[i] = chisq.test(cancer,correct=FALSE)$statistic }
hist(malig, breaks = 25)
```

Chapter 7
Logistic Regression: The Space Shuttle Challenger

R packages require for this chapter: Design and/or rms

```
install.packages("Design")
require(Design)
```

Warning: Pay attention to the quotation marks; if you copy and paste the code to the R prompt, it might distort the "" sign and the code may not work. If that is the case then manually typing in the code will fix it.

Some later versions of R will not accept package Design. In this case, install package rms instead:

```
install.packages("rms")
require(rms)
```

How to Perform Logistic Regression with R

The `glm` function is used to perform logistic regression analysis in R. Similar to standard linear regression, there are three main arguments that need to be provided to `glm`:

1) A formula with the binary response variable on the left side of the "~" and the explanatory variable on the right side of the "~" sign. If there is more than one explanatory variable to include in the model, then place a "+" between them. Interaction terms can be included by placing a ":" between the variables for which an interaction will be created.

2) The argument `family=binomial` indicates that logistic regression, rather than linear regression, will be performed.

3) An argument to indicate which data set will be used.

Other options are available, and you can type `help(glm)` to learn more about the function.

Activities

2. Read in and attach the `Shuttle` data into your R session:

```
shuttle <- read.csv("FILE PATH/ C7Shuttle.csv",header=TRUE)
attach(shuttle)
```

Create a scatterplot of `Temperature` versus `Success` (0 = not a successful launch, 1 = successful launch):

```
plot(Temperature,Success,xlab="Temperature(Fahrenheit)",ylab="Success (0/1)")
```

Warning: The names of 2 variables in the previous plot might not be exactly `"Temperature"` and `"Success".` In that case, use the function `names()` to check the variables in shuttle and use them for plotting.

```
names(shuttle)
```

Create a side-by-side boxplot of Temperature by Success:

```
boxplot(Temperature~Success)
```

3. Create a scatterplot of `Temperature` versus `Success` (0 = not a successful launch, 1 = successful launch) with a least squares regression line superimposed:

```
plot(Temperature,Success,xlab="Temperature (Fahrenheit)", ylab="Success
(0/1)")

abline(lm(Success~Temperature)$coef[1],lm(Success~Temperature)$coef[2])
# or use: abline(lm(Success~Temperature)
```

Obtain the least squares estimates of the slope and y-intercept:

```
lm(Success~Temperature)$coef
```

These estimates can be used to predict the response values when temperature is 60 degrees and 85 degrees.

5. To plot relationships between π and *X* using values of β_0 =-10, -5 and values of β_1 =.5, 1, and 1.5, type the following lines of code:

```
par(mfrow=c(2,3))
Beta0 <- c(-10,-5)
Beta1 <- c(.5,1,1.5)
X <- seq(0,30,by=.1)

for (i in 1:2) {
  for (j in 1:3) {
    pi <- exp(Beta0[i] + Beta1[j]*X)/(1+exp(Beta0[i] + Beta1[j]*X))
    plot(X,pi,type="l",ylim=c(0,1),ylab = expression(pi))
    title(main=bquote(paste(beta[0]==.(Beta0[i]),"
   ",beta[1]==.(Beta1[j]))),cex=.7)
  }
}
```

To plot relationships between π and *X* using values of β_0 =10, 5 and values of β_1 = -.5, -1, and -1.5, type the following lines of code:

```
par(mfrow=c(2,3))
Beta0 <- c(10,5)
Beta1 <- c(-.5,-1,-1.5) X <- seq(0,30,by=.1)
for (i in 1:2) {
  for (j in 1:3) {
    pi <- exp(Beta0[i] + Beta1[j]*X)/(1+exp(Beta0[i] + Beta1[j]*X))
    plot(X,pi,type="l",ylim=c(0,1),ylab = expression(pi))
    title(main=bquote(paste(beta[0]==.(Beta0[i]),"
   ",beta[1]==.(Beta1[j]))),cex=.7)
  }
}
```

6. To obtain maximum likelihood estimate for β_0 and β_1, we first create a logistic regression object in R which will be named shuttle.logistic.5that will contain several key aspects of the model, including parameter estimates, standard errors, and additional information that will be used later. Type in the following code:

```
shuttle.logistic1 <- glm(Success~Temperature,family=binomial)
```

Now we can examine the output in shuttle.logistic1to obtain the maximum likelihood estimates using the summary() function. Type:

```
summary(shuttle.logistic1)
```

In the output, the values in the column labeled Estimateare the MLE's for β_0 and β_1, i.e. they are the values for b_0 and b_1, respectively.

7. After you find the coefficient for the logistic regression model in Question 6, there is a quick way to get the fitted values of the predicted probabilities for each value of X given:

```
shuttle.logistic1$fitted.values
```

11. Probabilities from a logistic regression model using the MLE's (from Questions 6):

```
prob.mle <- exp(-15.043+.232*Temperature)/(1+exp(-15.043+.232*Temperature))
```

Probabilities from a logistic regression model using the least squares estimates:

```
ls.coef <- lm(Success~Temperature)$coef
prob.ls <- exp(ls.coef[1]+ls.coef[2]*Temperature)/
   (1+exp(ls.coef[1]+ls.coef[2]*Temperature))
```

Create a plot of two logistic regression models:

```
par(mfrow=c(1,2))
plot(Temperature,prob.mle,xlab="Temperature",ylab="Probability",
main="Parameters Estimated by MLE", ylim = c(0,1), xlim = c(0,125))

plot(Temperature,prob.ls,xlab="Temperature",ylab="Probability",
main="Parameters Estimated by LS", ylim = c(0,1), xlim = c(0,125))
```

To fit the logistic regression model where a 1 indicates O-ring failure, we can use `(1-Success)` as the new response variable. The new logistic regression object will be called `shuttle.logistic2:`

```
shuttle.logistic2 <- glm((1-Success)~Temperature,family=binomial)
```

Now examine the output in `shuttle.logistic2` to observe the maximum likelihood estimates using the `summary()` function. Check the column labeled `Estimate` to find the MLE's.

```
summary(shuttle.logistic2)
```

13. We will use the summary output for `shuttle.logistic1` to conduct the likelihood ratio test (LRT). First, compute the test statistic for the LRT, which we'll name `G`:

```
G <- shuttle.logistic1$null.deviance-shuttle.logistic1$deviance
```

Next, we'll compute the degrees of freedom for the test statistic:

```
df <- shuttle.logistic1$df.null-shuttle.logistic1$df.residual
```

Note that gives us the number of extra parameters that need to be estimated in the full model vs. the null model. Finally, we can compute a *p*-value for the test use the following code:

```
1-pchisq(G,df)
```

Extended Activities

14. Read in and attach the `Cancer2` data into your R session:

```
cancer <- read.table("FILE PATH/Cancer2.TXT",header=TRUE,sep="\t")
attach(cancer)
```

The logistic regression object corresponding to the model with binary response variable `Malignant` and explanatory variables `radius` and `concavity` will be saved as `cancer.logistic1`:

```
cancer.logistic1 <- glm(Malignant~radius+concavity,family
=binomial,data=cancer)
```

Warning: The names of 2 variables in the previous plot might not be exactly `"radius"` and `"concavity".` In that case, use the function `names()` to check the variables before using them in `glm().`

The parameter estimates for the model is obtained using the `summary()` function. The residual deviance can be obtained with the code:

```
cancer.logistic1$deviance
```

The null deviance can be obtained with the code:

```
cancer.logistic1$null.deviance
```

The LRT Statistic and degrees of freedom are found using the code:

```
G <- cancer.logistic1$null.deviance-cancer.logistic1$deviance
df <- cancer.logistic1$df.null-cancer.logistic1$df.residual
```

The *p*-value for the LRT:

```
1-pchisq(G,df)
```

Probabilities of malignant cells can be computed by plugging in appropriate values of the explanatory variables into the estimated logistic regression function and solving for P(Malignant).

16. The logistic regression object corresponding to the model with binary response variable `Malignant` and explanatory variable `radius` will be saved as `cancer.logistic2`:

```
cancer.logistic2 <- glm(Malignant~radius,family=binomial)
```

The remainder of the question is identical to that of Questions 15; however, be sure to replace `cancer.logistic1` with `cancer.logistic2` when extracting relevant information from the model results.

18. To fit the logistic regression model where a 1 indicates a benign mass, we can use `(1-Malignant)` as the new response variable. The new logistic regression object will be called `cancer.logistic3`:

```
cancer.logistic3 <- glm((1-Malignant)~radius+concavity,family=binomial)
```

The logistic regression output can be examined with: `summary(cancer.logistic3)`

22. Goodman-Kruskal Gamma and Kendall's Tau-a can be obtained easily in R, but a separate R package must be loaded (and possibly downloaded first), and then the logistic model must be re-fit to the data using the function `lrm`. The output for `lrm` is slightly different from that obtained using the `glm` function.

Re-fit the logistic regression using the `lrm` function:

```
lrm(Success~Temperature)
```

Warning: Data for flight 4 is missing, so you will leave out this observations, and as a result the logistic regression model will only include 23 observations.

The output is as follows:

```
Logistic Regression Model
lrm(formula = Success ~ Temperature)
Frequencies of Responses
0  1
7 16

Frequencies of Missing Values Due to Each Variable
Success Temperature
1  0

Obs  Max Deriv Model L.R.       d.f.       P          C         Dxy
23       2e-04       7.95          1     0.0048      0.781     0.562

Gamma    Tau-a R2    Brier
0.589    0.249 0.413 0.139
```

	Coef	S.E.	Wald Z	P
Intercept	-15.0429	7.3786	-2.04	0.0415
Temperature	0.2322	0.1082	2.14	0.032

Goodman-Kruskal Gamma and Kendall's Tau-a are identified in bold above.

26. Read in and attach the Cancercellsdata into your R session:

```
cancer.group <- read.table("FILE PATH\Cancercells.txt",header
   =TRUE,sep="\t")
attach(cancer.group)
```

Fitting a logistic regression model to binomial counts is slightly different from what was done earlier. First, create a two-column matrix containing the counts of successes (cells that are malignant) in the first column and the counts of failures (cells that are benign) in the second column:

```
cells <- cbind(Malignant,Benign)
```

Now fit the logistic regression model:

```
cancer.group1 <- glm(cells~med.rad,family=binomial)
```

The results of the fit can be examined with the `summary()` function, and the LRT can be performed by following the instructions of Questions 15. You can check the fit by code:

```
plot(Median.Radius,Proportion.Malignant,xlab="R",ylab="Probability",
main="Parameters Estimated by MLE", ylim = c(0,1), xlim = c(1,7))
par(new=T)
M <- seq(1,7,by=0.01)
cancery <- exp(cancer.group1[1] + cancer.group1[2]*M)/(1+exp(cancer.group1[1] +
cancer.group1[2]*M))
plot(M,cancery,type="l",ylim=c(0,1),ylab = NULL)
par(new=F)
```

27. **a)** Create a vector of the Pearson residuals:

```
pearson.res <- residuals(cancer.group1,type="pearson")
```

b) Construct a normal probability plot of the Pearson residuals:

```
qqnorm(pearson.res)
```

c) Create a scatterplot of the mean radius versus Pearson residuals:

```
plot(med.rad,pearson.res)
```

28. **a)** Create a vector of the deviance residuals:

```
dev.res <- residuals(cancer.group1,type="deviance")
```

b) Construct a normal probability plot of the Pearson residuals:

```
qqnorm(dev.res)
```

c) Create a scatterplot of the mean radius versus Pearson residuals:

```
plot(med.rad,dev.res)
```

31. There is no easily accessible code for the Pearson or deviance goodness-of-fit tests in R for ungrouped data when there are duplicate values of the explanatory variable. As stated in the text, these tests are not reliable when there are few observations at each level of the explanatory variable.

There is available code to perform the Hosmer-Lemeshow test. The following function to perform the Hosmer-Lemeshow test was developed by Peter D. M. Macdonald, McMaster University:

```
hosmerlem <- function (y, yhat, g = 10)
{
cutyhat <- cut(yhat, breaks = quantile(yhat, probs = seq(0, 1, 1/g)),
include.lowest = T)
obs <- xtabs(cbind(1 - y, y) ~ cutyhat)
expect <- xtabs(cbind(1 - yhat, yhat) ~ cutyhat)
chisq <- sum((obs - expect)^2/expect)
P <- 1 - pchisq(chisq, g - 2)
c("X^2" = chisq, Df = g - 2, "P(>Chi)" = P)
}
```

In the code, `y` is the binary response variable, `yhat` corresponds to the fitted probabilities, and `g` is number of groups (default number is 10 groups).

We obtain the fitted probabilities from the logistic regression model, then apply the `hosmerlem` () function.

Warning: Before re-attaching cancer, you might want to clear your memory because there are more than 1 variables name `radius` in two data sets. The code to clear all memory is:

```
rm(list=ls())
```

Then re-attach `cancer2`:

```
cancer <- read.table("FILE PATH/Cancer2.TXT",header=TRUE,sep="\t")
attach(cancer)

cancer.glm.fits <- glm(Malignant~radius,family=binomial)$fit
hosmerlem(Malignant,cancer.glm.fits)
```

32. Adjust `g` in the previous question.

33. See previous questions for residual plots. There is no easily accessible code for delta and leverage graphs.

34. **a)** Plot the log-likelihood function:

```
pi <- seq(.01,.99,length=100)
loglik <- 5*log(pi)+7*log(1-pi)
plot(pi,loglik,xlab="Pi",ylab="Log-Likelihood Function")
```

Chapter 8 R Instructions
Poisson Log-Linear Regression: Detecting Cancer Clusters

How to Perform Log-Linear Regression with R

The `glm` function is used to perform log-linear regression analysis in R. There are either 3-4 main arguments that need to be provided to `glm`:

1) A formula with a count response variable on the left side of the "~" and the explanatory variable or variables on the right side of the "~" sign. If there is more than one explanatory variable to include in the model, then place a "+" between them. Interaction terms can be included by placing a ":" between the variables for which an interaction will be created.

2) The argument `family=poisson` to indicate that log-linear regression, rather than linear regression will be performed.

3) An argument to indicate which data set will be used.

4) In addition, an "offset" term may need to be specified if the response variable is a count per unit of "exposure," for example number of persons in a geographic region, or the number of persons in different age groups. For a given exposure variable, the offset is specified as `offset=log(exposure)` in the `glm()` function.

Other options are available, and you can type `help(glm)` to learn more about the function.

Activities

9. Here is example R code for simulating the cancer status of 1,138 individuals with an average of 25 years of residence. We will use `runif()` to draw a random number between zero and one for each person-year of exposure (and repeat 10,000 times).

Create 10,000 communities of 1,138 people, and let's assume that every person lived in Randolph for 25 years. Each community has a cancer rate equal to the national average of 326 cases/100,000 person-years:

```
n.sims <- 10000
n.people <- 1138
n.years <- 25
rate <- 326/100000
n.person.years <- n.people * n.years
```

Create an array to hold the 10,000 simulated cancer counts, and get a histogram of the simulation results to see how the Bartlett-Green Acres neighborhood compares:

```
cases <- rep(0, n.sims)
for (sim in 1:n.sims) {
x = runif(n.person.years)
cases[sim] <- sum(x < rate)
}
summary(cases)
hist(cases)
```

To obtain the simulated p-value, add up the number of simulated neighborhoods at least as extreme as Randolph, MA (at least 67 cases), and divide by number of simulations to get proportion:

```
pvalue <- sum(cases >= 67) / n.sims
pvalue
```

10. Repeat the steps outlined for the previous question, but change one line so that the exposure is 12.5 years per person on average:

```
n.years = 12.5
```

15. Read in the BGA_CTR data into your R session:

```
bga_ctr <- read.table("FILE PATH/bga_ctr.TXT",header=TRUE)
```

Create a subset of the data corresponding to the BGA location:

```
BGA <- subset(bga_ctr, location=="BGA")
```

Fit the Poisson regression model to the BGA cancer rates with median age as the covariate:

```
bga.glm <- glm(cases ~ median_age,
family=poisson,offset=log(exposure),data=BGA)
```

The offset term `offset=log(exposure)` take care of the different exposure levels (person-years) for the age groups. We can inspect the estimated coefficients by using the `summary()` command:

```
summary(bga.glm)
```

17. Create a subset of the data corresponding to the CTR location:

```
CTR <- subset(bga_ctr, location=="CTR")
```

Fit the Poisson regression model to the CTR cancer rates with median age as the covariate:

```
ctr.glm <- glm(cases ~ median_age, family=poisson, offset=log(exposure),
data=CTR)
```

Use the `summary()` command to view the results of the fitted model.

20. Fit the Poisson regression model to the BGA and CTR cancer rates with median age as the covariate:

```
bgactr.glm1 <- glm(cases ~ median_age, family=poisson,
offset=log(exposure), data=bga_ctr)
```

Use the `summary()` command to view the results of the fitted model.

21. Fit the Poisson regression model to the BGA and CTR cancer rates with location as the covariate:

```
bgactr.glm2 <- glm(cases ~ as.factor(location), family=poisson,
offset=log(exposure),data=bga_ctr)
```

22. Fit the Poisson regression model to the BGA and CTR data with age and location as the covariates:

```
bgactr.glm3 <- glm(cases ~ median_age+as.factor(location), family=poisson,
offset=log(exposure), data=bga_ctr)
```

Use the `summary()` command to view the results of the fitted model.

25. Create a graph of the logarithm of the incidence rates versus median age by location:

```
par(mfrow=c(1,1))
plot(CTR$median_age,log(CTR$rate),xlab="Median Age",
```

```
ylab="Log Incidence Rate",ylim=c(3,8))
points(BGA$median_age,log(BGA$rate),pch=19)
legend(30,7,c("BGA","CTR"),pch=c(19,1))
```

27. Fit the Poisson regression model to the BGA and CTR data with covariate age, location, and an interaction between age and location:

```
bgactr.glm4 <- glm(cases ~ median_age+as.factor(location)+
median_age:as.factor(location), family=poisson, offset=log(exposure),
data=bga_ctr)
```

31. Compute the LRT statistic:

```
LRT <- bgactr.glm3$deviance-bgactr.glm4$deviance
```

Obtain the p-value for the LRT:

```
1-pchisq(LRT,1)
```

33. To fit the Poisson model to the CTR data that includes a quadratic term for Age:

```
ctr.glm2 <- glm(cases ~ median_age + I(median_age^2), family=poisson,
offset=log(exposure),data=CTR)
```

Examine the summary output for the deviance statistic.

34. Residuals of the interaction model:

```
round(residuals(bgactr.glm4,type="deviance"),4)
```

Plot of deviance residuals versus median age:

```
plot(bga_ctr$median_age,residuals(bgactr.glm4),xlab="Median Age",
ylab="Deviance Residual")
```

Fit the interaction model plus the quadratic term for age:

```
bgactr.glm5 <- glm(cases ~ median_age+as.factor(location)
+median_age:as.factor(location) +I(median_age^2),family=poisson,
offset=log(exposure),data=bga_ctr)
```

The LRT statistic is:

```
LRT <- bgactr.glm4$deviance-bgactr.glm5$deviance
```

The corresponding p-value of the LRT is:

```
1-pchisq(LRT,1)
```

Extended Activities

44. Read in the `Smoking` data:

```
smoking <- read.table("FILE PATH/Smoking.txt",header=TRUE)
```

Fit a Poisson regression model to the smoking data:

```
smoke.glm <- glm(Number ~ Location,family=poisson,data=smoking)
```

49. Pearson chi-square statistic:

```
sum(residuals(smoke.glm,type="Pearson")^2)
```

50. Read in the `gala` data:

```
gala <- read.table("FILE PATH/gala.txt",header=TRUE)
```

Create plots of the logarithm of the observed counts versus each of the potential covariates:

```
plot“gala$species”,“gala$area”,xlab="Area",
ylab="Log Species Count")
plot“gala$species”,“gala$elevation”,xlab="Elevation",
ylab="Log Species Count")
plot“gala$species”,“gala$nearest”,xlab="Nearest",
ylab="Log Species Count")
plot “gala$species”,“gala$scruz”,xlab="Scruz",
ylab="Log Species Count")
plot “gala$species”,“gala$adjacent”,xlab="Adjacent",
ylab="Log Species Count")
```

51. Plot the log-observed counts versus the logarithm of the covariates:

```
plot“gala$species”,“gala$area”,xlab="Log Area",
ylab="Log Species Count"
plot“gala$species”,“gala$elevation”,xlab="Log Elevation",
ylab="Log Species Count")
plot“gala$species”),“gala$nearest”,xlab="Log Nearest",
ylab="Log Species Count")
plot“gala$species”,“gala$scruz”,xlab="Log Scruz",
ylab="Log Species Count")
plot”gala$species”,”gala$adjacent”,xlab="Log Adjacent",
ylab="Log Species Count")
```

52. Fit a Poisson regression model to the total species count with covariates `log(area)` & `log(elevation)`:

```
gala.glm <-
glm(species~log(area)+log(elevation),family=poisson,data=gala)
```

The LRT statistic to determine if at least one of the covariates is significant:

```
LRT <- gala.glm$null.deviance-gala.glm$deviance
```

P-value for the LRT:

```
1-pchisq(LRT,2)
```

53. Fit a Poisson regression model to the total species count with covariates `log(area), log(elevation), nearest, scruz, and adjacent:`

```
gala.glm2 <-
glm(species~nearest+scruz+adjacent+log(area)+log(elevation),
family=poisson,data=gala)
```

The LRT statistic is:

```
LRT <- gala.glm$deviance - gala.glm2$deviance
```

P-value is:

```
1-pchisq(LRT,3)
```

55. Compute the correlations between all pairs of covariates (use the logarithm of age and elevation):

```
cor(cbind(gala$area, gala$elevation,gala$nearest,gala$scruz,
gala$adjacent))
```

58. To fit an overdispersed Poisson regression model, choose the option `family=quasipoisson`, and leave everything else the same. The parameter estimates will be identical to those in Problem 51, but the standard errors (and hence the Wald statistics) will be adjusted by the estimated overdispersion parameter:

```
gala.glm2 <-glm(species~nearest+scruz+adjacent+log(area)+log(elevation),
family=quasipoisson,data=gala)
```

Use the `summary()` command, and compare the results to those in Problem 53. You will notice that the estimated coefficients are the same, but the p-values are all different.

Chapter 9
Survival Analysis: Melting Chocolate Chips

How to Perform Survival Analysis with R

To analyze survival data in R, you will need to download the package `survival` and load the package using the command `library(survival)` in your R session. This package contains many of the necessary survival analysis routines you will need to solve the Activities and Extended Activities.

Unfortunately, there are currently no R functions in the `survival` package that compute and plot the estimated hazard function or the cumulative hazard function. So we have supplied two R functions that you can use to plot these functions. The function `plot.haz()` will plot the estimated hazard function and `plot.chaz()` will plot the estimated cumulative hazard function when supplied with a `survfit` object (see the instructions for the related Activities and Extended Activities below).

You will need to enable this code when you are in your R session. The easiest way to do this is to open the files `plot.haz.R` and `plot.chaz.R` in your R session, highlight it, and click the "Run line or selection" icon. This process effectively makes these functions available to use. To apply it to survival data, you will supply a `survfit` object (e.g. `KM.obj`) to either of these functions. See the related Extended Activities below.

For the questions that require class activity data, we will use the `MeltingChipsJS` data set.

Activities

install.packages("survival")
require(survival)

19. Read in the class data or `MeltingChipsJS` data into R session:

```
meltchips <- read.table("FILE PATH/MeltingChipsJS.txt",header=TRUE)
```

Construct the Kaplan-Meier curves for the white and milk chocolate chips and store them in the `survfit` object named `KM.obj`:

```
KM.obj <- survfit(Surv(Time,Censor)~Type,data=meltchips)
```

Plot the Kaplan-Meier curves on the same graph:

```
plot(KM.obj,lty=1:2,xlab="Time",ylab="Survival    Probability",ylim=c(0,1))
```

Add a legend to the plot to distinguish between the two groups:

```
legend(1,.2,c("Milk","White"),lty=1:2)
```

25. The estimated median survival time can be obtained using the `print()` command:

```
print(KM.obj)
```

To compute the estimated mean survival times, use the `print()` command and include the following options:

```
print(KM.obj,print.rmean=TRUE,rmean="individual")
```

29. First, create a dataset with milk chocolate chips only:

```
milkchips <- meltchips[meltchips$Type=="Milk",]
```

Construct the Kaplan-Meier curve for the milk chocolate chips and store it in `KM.obj`:

```
KM.obj <- survfit(Surv(Time,Censor)~Type,data=milkchips,conf.type="plain")
```

Plot the Kaplan-Meier curve with the pointwise confidence intervals for the survival probabilities on the same graph:

```
plot(KM.obj,xlab="Time",ylab="Survival Probability",ylim=c(0,1))
```

You can examine the values of the 95% confidence intervals using the `summary()` command:

```
summary(KM.obj)
```

Note that the upper limits of the confidence intervals are truncated at 1.

31. Construct the Kaplan-Meier curves for the white and milk chocolate chips and store them in `KM.obj`:

```
KM.obj <- survfit(Surv(Time,Censor)~Type,data=meltchips,conf.type="plain")
```

Plot the Kaplan-Meier curves on the same graph:

```
plot(KM.obj,lty=1:2,xlab="Time",ylab="Survival Probability",ylim=c(0,1),
conf.int=TRUE)
```

Add a legend to the plot to distinguish between the two groups:

```
legend(1,.2,c("Milk","White"),lty=1:2)
```

The values of the pointwise confidence intervals for the survival probabilities can also be examined using the `summary()` command.

36. Perform the log-rank test to compare the melting experiences of the white and milk chocolate chips:

```
survdiff(Surv(Time,Censor)~Type,data=meltchips)
```

The test statistic and p-value are provided at the bottom of the output.

Extended Activities

43. Read in the `Firstdrink` data:

```
firstdrink <- read.table("FILE PATH/Firstdrink.txt",header=TRUE)
```

Construct the Kaplan-Meier curve and store it in `KM.obj`:

```
KM.obj <- survfit(Surv(Age,Censor)~1,data=firstdrink,conf.type="plain")
```

44. Use the accompanying R function `plot.haz()` to compute the estimated hazard rates, and/or plot the estimated hazard function, `KM.obj` from Question 43:

```
plot.haz(KM.obj)
```

```
plot.haz <- function(KM.obj,plot="TRUE") {
 ti <- summary(KM.obj)$time
 di <- summary(KM.obj)$n.event
 ni <- summary(KM.obj)$n.risk

 #Est Hazard Function
 est.haz <- 1:(length(ti))
 for (i in 1:(length(ti)-1))
      est.haz[i] <- di[i]/(ni[i]*(ti[i+1]-ti[i]))
 est.haz[length(ti)] <- est.haz[length(ti)-1]

 if (plot=="TRUE") {
  plot(ti,est.haz,type="s",xlab="Time",
       ylab="Hazard Rate",
       main=expression(paste(hat(h),(t)[KM])))
 }
 return(list(est.haz=est.haz,time=ti))
}
```

45. Read in the Graduate data:

```
graduate <- read.table("FILE PATH/Graduate.txt",header=TRUE)
```

First construct the Kaplan-Meier curve and store it in KM.obj:

```
KM.obj <- survfit(Surv(Years,Censor)~1,data=graduate,conf.type="plain")
```

Now plot the estimated hazard function using the plot.haz() function:

```
plot.haz(KM.obj)
```

49. Read in the Rearrestdata:

```
rearrest <- read.table("FILE PATH/Rearrest.txt",header=TRUE)
```

First construct the Kaplan-Meier curve and store it in KM.obj:

```
KM.obj<-survfit(Surv(months,censor)~1,data=rearrest,conf.type="plain")
```

The estimated cumulative hazard function is plotted using the following plot.chaz() function:

```
plot.chaz <- function(KM.obj,plot="TRUE") {
ti <- summary(KM.obj)$time
di <- summary(KM.obj)$n.event
ni <- summary(KM.obj)$n.risk

#Est Cumulative Hazard Function
est.cum.haz <- 1:(length(ti))
for (i in 1:(length(ti)))
  est.cum.haz[i] <- sum(di[1:i]/ni[1:i])

plot.chaz <- 1:length(KM.obj$time)
for (i in 1:length(plot.chaz))
   plot.chaz[i] <- sum((KM.obj)$n.event[1:i]/(KM.obj)$n.risk[1:i])

if (plot=="TRUE") {
 plot((KM.obj)$time,plot.chaz,type="s",xlab="Time",
      ylab="Cumulative Hazard",main=expression(paste(hat(H),(t)["NA"])))
}
return(list(est.chaz=plot.chaz,time=(KM.obj)$time))
}
```

Now plot the estimated hazard and cumulative hazard functions using the plot.haz()and plot.chaz() functions:

```
par(mfrow=c(1,2))
plot.haz(KM.obj)
plot.chaz(KM.obj)
```

Chapter 10
Principal Component Analysis: Stock Market Values

```
install.packages("scatterplot3d")
library(scatterplot3d)
```

Activities

1. Open the `2006Stocks` dataset and attach it to your R session:

```
stocks <- read.table("FILE PATH/2006Stocks.txt", header=TRUE,sep="\t")
attach(stocks)
```

Create time series plots of the Dow and S&P. Both time series should be on the same graph.

```
> lablist <- as.vector(Date)[seq(1,251,by=45)]
> plot(Dow,axes=FALSE,type="l",ylab="Stock MarketValue", xlab="Time",ylim=c(1000,
13000),col=2)
> lines(S.P,lty=2,col=3)
> box()
> axis(1,at=seq(1,251,by=45),labels=FALSE)
> axis(2,at=seq(0,13000,by=1000),labels=TRUE)
> text(seq(1,251,by=45),par("usr")[3],pos=1,labels=lablist,xpd=TRUE)
> legend(175,7000,c("Dow Jones","S&P"),lty=1:2,col=2:3)
```

2. Standardize the Dow and S&P columns, (i.e. for each element in the Dow column, subtract the Dow mean and divide by the Dow standard deviation. Save the standardized Dow values in a new column labeled Z1.

   a) Create time series plots of the standardized Dow and S&P, Z1 and Z2. Both time series should be on the same graph.

```
Z1 <- scale(Dow) #Standardize DJ
Z2 <- scale(S.P) #Standardize SP

lablist <- as.vector(Date)[seq(1,251,by=45)]
plot(Z1,axes=FALSE,type="l",ylab="Stock Market Value",
xlab="Time",ylim=c(-3,3),col=2)
lines(Z2,lty=2,col=3)
box()
axis(1,at=seq(1,251,by=45),labels=FALSE)
axis(2,at=seq(-3,3,by=1),labels=TRUE)
text(seq(1,251,by=45),par("usr")[3],pos=1,labels=lablist,
xpd=TRUE)
legend(2,2,c("Z1 (Dow)","Z2 (S&P)"),lty=1:2,col=2:3)
```

3. Later sections will show that the first principal component is calculated as PC1 = Y 1 = 0.707Z1 + 0.707Z2.
   a) Calculate PC1 and submit a time series plot of the standardized Dow and S&P closing values (Z1, Z2) and the first principal component (PC1).

```
PC1 <- 0.707*Z1 + 0.707*Z2
lablist <- as.vector(Date)[seq(1,251,by=45)]
plot(Z1,axes=FALSE,type="l",ylab="Stock Market Value",
xlab="Time",ylim=c(-3,3),col=2,lwd=2)
lines(Z2,lty=2,col=3,lwd=2)
lines(PC1,lty=3,col=4,lwd=2)
box()
axis(1,at=seq(1,251,by=45),labels=FALSE)
axis(2,at=seq(-3,3,by=1),labels=TRUE)
text(seq(1,251,by=45),par("usr")[3],pos=1,labels=lablist,xpd=TRUE)
legend(2,2,c("Z1 (Dow)","Z2 (S&P)","PC1"),lty=1:3,col=2:4,lwd=2)
# or use: legend(0,3,c("Z1 (Dow)","Z2 (S&P)","PC1"),lty=1:3,col=2:4,lwd=2, cex=0.5)
```

You can use cex=0.5 to make legends smaller

4. First, create Z1, Z2, and Z3:

```
Z1 <- scale(Dow) #Standardize DJ
Z2 <- scale(SP) #Standardize SP
Z3 <- scale(Nasdaq) #Standardize Nasdaq
```

a)

```
C1 <- 1*Z1 + 0*Z2 + 0*Z3
lablist <- as.vector(Date)[seq(1,251,by=45)]
plot(Z1,axes=FALSE,type="l",ylab="Stock Market Value",xlab="Time", ylim=c(-
3,3),col=2,lwd=2)
lines(Z2,lty=2,col=3,lwd=2)
lines(Z3,lty=3,col=4,lwd=2)
lines(C1,lty=4,col=5,lwd=2) box()
axis(1,at=seq(1,251,by=45),labels=FALSE)
axis(2,at=seq(-3,3,by=1),labels=TRUE)
text(seq(1,251,by=45),par("usr")[3],pos=1,labels=lablist,xpd=TRUE)
legend(2,2.75,c("Z1 (Dow)","Z2 (S&P)","Z3
(Nasdaq)","C1"),lty=1:4,col=2:5,lwd=2)
# or use: legend(0,3,c("Z1 (Dow)","Z2 (S&P)","Z3 (Nasdaq)","C1"),
lty=1:4,col=2:5,lwd=2, cex=0.5)
```

b)

```
C2 <- 1*Z1 - 1*Z2 + 0*Z3
lablist <- as.vector(Date)[seq(1,251,by=45)]
plot(Z1,axes=FALSE,type="l",ylab="Stock Market Value",xlab="Time", ylim=c(-
3,3),col=2,lwd=2)
lines(Z2,lty=2,col=3,lwd=2)
lines(Z3,lty=3,col=4,lwd=2)
lines(C2,lty=4,col=5,lwd=2) box()
axis(1,at=seq(1,251,by=45),labels=FALSE)
axis(2,at=seq(-3,3,by=1),labels=TRUE)
text(seq(1,251,by=45),par("usr")[3],pos=1,labels=lablist,xpd=TRUE)
legend(2,2.75,c("Z1 (Dow)","Z2 (S&P)","Z3(Nasdaq)","C2"),
lty=1:4,col=2:5,lwd=2)
```

c)

```
C3 <- 1/3*Z1 + 1/3*Z2 + 1/3*Z3
lablist <- as.vector(Date)[seq(1,251,by=45)]
plot(Z1,axes=FALSE,type="l",ylab="Stock Market Value",xlab="Time", ylim=c(-
3,3),col=2,lwd=2)
lines(Z2,lty=2,col=3,lwd=2)
lines(Z3,lty=3,col=4,lwd=2)
lines(C3,lty=4,col=5,lwd=2) box()
axis(1,at=seq(1,251,by=45),labels=FALSE) axis(2,at=seq(-
3,3,by=1),labels=TRUE)
text(seq(1,251,by=45),par("usr")[3],pos=1,labels=lablist,xpd=TRUE)
legend(2,2.75,c("Z1 (Dow)","Z2 (S&P)","Z3(Nasdaq)","C3"),
lty=1:4,col=2:5,lwd=2)
```

d)

```
C4 <- 0*Z1 + 0*Z2 - 1*Z3
lablist <- as.vector(Date)[seq(1,251,by=45)]
plot(Z1,axes=FALSE,type="l",ylab="Stock Market Value",
xlab="Time",ylim=c(-3,3),col=2,lwd=2)
lines(Z2,lty=2,col=3,lwd=2)
lines(Z3,lty=3,col=4,lwd=2)
lines(C4,lty=4,col=5,lwd=2)
box()
axis(1,at=seq(1,251,by=45),labels=FALSE)
axis(2,at=seq(-3,3,by=1),labels=TRUE)
text(seq(1,251,by=45),par("usr")[3],pos=1,labels=lablist,xpd=TRUE)
```

```
legend(2,2.75,c("Z1 (Dow)","Z2 (S&P)","Z3(Nasdaq)",
"C4"),lty=1:4,col=2:5,lwd=2)
```

4.

5. Calculate the 2 by 2 correlation matrix for the Dow and S&P variables:

```
Corr1 <- cor(cbind(Dow,S.P))
Corr1
```

6. Calculate the 2 by 2 correlation matrix for the Dow and Nasdaq variables:

```
Corr2 <- cor(cbind(Dow,Nasdaq))
Corr2
```

9. Use R to calculate the eigenvectors and eigenvalues of the correlation matrix R (for the Dow and S&P variables).

```
#Eigenvalues of Corr1:
eigen(Corr1)$values

#(Normalized) Eigenvectors of Corr1:
eigen(Corr1)$vectors
```

Note that the signs of the coefficients of the eigenvectors may differ from those coefficients computed using Minitab:

10. Calculate the principal components:

```
PC1 <- .707*Z1+.707*Z2
PC2 <- -.707*Z1+.707*Z2
```

a) Create a scatterplot of PC1 versus PC2:

```
plot(PC1,PC2)
```

Calculate the correlation between PC1 and PC2:

```
cor(PC1,PC2)
```

b) Calculate the variances of PC1 and PC2:

```
var(PC1)
var(PC2)
```

11. Create a time series plot of Z1, Z2, PC1, and PC2:

```
lablist <- as.vector(Date)[seq(1,251,by=45)]
plot(Z1,axes=FALSE,type="l",ylab="Stock Market Value",xlab="Time",
ylim=c(-3,3),col=2,lwd=2)
lines(Z2,lty=2,col=3,lwd=2)
lines(PC1,lty=3,col=4,lwd=2)
lines(PC2,lty=4,col=5,lwd=2)
box()
axis(1,at=seq(1,251,by=45),labels=FALSE)
axis(2,at=seq(-3,3,by=1),labels=TRUE)
text(seq(1,251,by=45),par("usr")[3],pos=1,labels=lablist,xpd=TRUE)
legend(2,2.75,c("Z1 (Dow)","Z2 (S&P)","PC1","PC2"),lty=1:4,col=2:5,lwd=2)
```

12. Standardize the Nasdaq column into a new column, Z3 and conduct a principal component analysis on all three variables.

```
Z1 <- scale(Dow) #Standardize DJ
Z2 <- scale(SP) #Standardize SP
Z3 <- scale(Nasdaq) #Standardize Nasdaq
```

a) To perform a principal component analysis, we can either find the eigenvalues and eigenvectors of the correlation matrix R, or we can use the R function `princomp()`. For this problem, use the `princomp()` function:

```
pc.stocks <- princomp(cbind(Z1,Z2,Z3)) #Creates a PCA object
round(loadings(pc.stocks),3) #round the values to 3 decimal places
```

The columns corresponding to "loadings" in the output are the eigenvectors of the correlation matrix R. Compute the eigenvalues of the correlation matrix R:

```
R <- cor(cbind(Dow,S.P,Nasdaq)) eigen(R)$values
```

b) Calculate the values (scores) of the three principal components:

```
PC1<-pc.stocks$scores[,1]
PC2<-pc.stocks$scores[,2]
PC3 <- pc.stocks$scores[,3]
```

13. For this question, the R package `scatterplot3d` is required and must be downloaded. Create a 3-D plot of the data:

```
# here you need to use: library(scatterplot3d)
s3d <- scatterplot3d(Z1,Z2,Z3,angle=105)
#Plot 1st PC
s3d$points3d(c(-0.5821417,0), c(-0.6080418,0), c(-0.5398113,0),
type="l",col="blue",lwd=2)
#Plot 2nd PC
s3d$points3d(c(-0.5377453,0), c(-0.2100566,0), c(0.8165208,0),
type="l",col="red",lwd=2)
#Plot 3rd PC
s3d$points3d(c(0.6098697,0), c(-0.7656118,0), c(0.2046889,0),
type="l",col="green",lwd=2)
legend(1,0,c("PC1","PC2","PC3"),lty=1,col=c("blue","red","green"),lwd=2)


#Rotate the plot (highlight the following "for loop," and run)
for (i in 1:460) {
s3d <- scatterplot3d(Z1,Z2,Z3,angle=i)
s3d$points3d(c(-0.5821417,0), c(-0.6080418,0), c(-0.5398113,0),
type="l",col="blue",lwd=2)
s3d$points3d(c(-0.5377453,0), c(-0.2100566,0), c(0.8165208,0),
type="l",col="red",lwd=2)
s3d$points3d(c(0.6098697,0), c(-0.7656118,0), c(0.2046889,0),
type="l",col="green",lwd=2)
legend(1,0,c("PC1","PC2","PC3"),lty=1,col=c("blue","red","green"),lwd=2)
}
```

14. Create a time series plot with Z1, Z2, Z3, PC1, PC2, and PC3:

```
lablist <- as.vector(Date)[seq(1,251,by=45)]
plot(-PC1,axes=FALSE,type="l",ylab="Stock Market Value",
xlab="Time",ylim=c(-4,4),col=2,lwd=2)
lines(PC2,lty=2,col=3,lwd=2)
lines(PC3,lty=3,col=4,lwd=2)
lines(Z1,lty=4,col=5,lwd=2)
lines(Z2,lty=5,col=6,lwd=2)
lines(Z3,lty=6,col=7,lwd=2)
box() axis(1,at=seq(1,251,by=45),labels=FALSE)
axis(2,at=seq(-3,3,by=1),labels=TRUE)
text(seq(1,251,by=45),par("usr")[3],pos=1,labels=lablist,xpd=TRUE)
legend(2,4,c("PC1","PC2","PC3","Z1","Z2","Z3"),lty=1:6,col=2:7,lwd=2)
```

15. Percentage of the variability is explained by the first principal component:

```
var(PC1)/(var(PC1)+var(PC2)+var(PC3))
```

Percentage of the variability is explained by the first two principal components combined:

```
(var(PC1)+ var(PC2))/(var(PC1)+var(PC2)+var(PC3))
```

Extended Activities

16. For this question, we'll perform PCA by finding the eigenvectors and eigenvalues of the covariance matrix (i.e. using the unstandardized data). First, calculate the covariance matrix S:

```
S <- cov(cbind(Dow,SP,Nasdaq))
```

Eigenvalues and eigenvectors of S:

```
eigen(S)$values
eigen(S)$vector
```

17. First compute the values of PC1 (requires some matrix algebra):

```
PC1 <- cbind(Dow,SP,Nasdaq)%*%eigen(S)$vector[,1]
```

Create a time series plot of the original data (X1, X2, and X3) and the first principal component, PC1:

```
lablist <- as.vector(Date)[seq(1,251,by=45)]
plot(PC1,axes=FALSE,type="l",ylab="Stock Market Value",xlab="Time",col=2,
lwd=2,ylim=c(-700,12500))
lines(Dow,lty=2,col=3,lwd=2)
lines(SP,lty=3,col=4,lwd=2)
lines(Nasdaq,lty=4,col=5,lwd=2)
box()
axis(1,at=seq(1,251,by=45),labels=FALSE)
axis(2,at=seq(-700,12500,by=1000),labels=TRUE)
text(seq(1,251,by=45),par("usr")[3],pos=1,labels=lablist,xpd=TRUE)
legend(2,8000,c("PC1","Dow","SP","Nasdaq"),lty=1:4,col=2:5,lwd=2)
```

18. Multiply Nasdaq values by 5000:

```
NewNasdaq <- 5000*Nasdaq
```

a) Conduct PCA using the covariance matrix on the Dow, S&P, and NewNasdaq. First, calculate the new covariance matrix Snew:

```
Snew <- cov(cbind(Dow,SP,NewNasdaq))
```

Eigenvalues and eigenvectors of Snew:

```
eigen(Snew)$values
eigen(Snew)$vector
```

b) Conduct PCA using the correlation matrix on the Dow, S&P, and NewNasdaq. First, calculate the new covariance matrix, Rnew:

```
Rnew <- cor(cbind(Dow,SP,NewNasdaq))
```

Eigenvalues and eigenvectors of Rnew:

```
eigen(Rnew)$values
eigen(Rnew)$vector
```

19) Read dataset `Versicolor` into your R session:

```
versicolor <- read.table("FILE PATH/Versicolor.txt", header=TRUE,sep="\t")
or
versicolor <- read.csv("FILE PATH/C10 Versicolor.csv", header=TRUE)
```

Perform a PCA on versicolor data using the correltaion matrix. First create a PCA object using the `princomp()` function:

```
versi.pca <- princomp(versicolor,cor=TRUE)
```

a) The four eigenvalues can be obtained using the syntax:

```
evalues <- (versi.pca$sdev)^2
```

b) Percentage of variation explained by the first PC:

```
evalues[1]/sum(evalues)
```

c) Percentage of variation explained by the first two PC's combined:

```
sum(evalues[1:2])/sum(evalues)
```

d) Construct a screeplot:

```
screeplot(versi.pca)
```

20. Percentage of variation in the original data explained by the third component:

```
evalues[3]/sum(evalues)
```

21. Create a loadings plot:

```
plot(evectors[,3],-evectors[,4],xlab="PC3",ylab="PC4")
lines(c(0,evectors[,3][1]),c(0,-evectors[,4][1]),col=1)
lines(c(0,evectors[,3][2]),c(0,-evectors[,4][2]),col=2)
lines(c(0,evectors[,3][3]),c(0,-evectors[,4][3]),col=3)
lines(c(0,evectors[,3][4]),c(0,-evectors[,4][4]),col=4)
```

24. Read and attach Cars dataset into your R session:

```
cars <- read.table("FILE PATH/Cars.txt", header=TRUE,sep="\t")
attach(cars)
```

Perform PCA on liters (Liter) and cylinders (Cyl), and store results in cars.pca:

```
cars.pca <- princomp(cbind(Liter,Cyl),cor=TRUE)
```

a)Create a new variable (PC1) that is a linear combination of liters and cylinders:

```
PC1 <- cbind(Liter,Cyl)%*%loadings(cars.pca)[,1]
```

Use the first principal component, plus Mileage, Buick, Cadillac, Chevrolet, Pontiac and SAAB in a regression model to predict the natural log of retail price, LnPrice.

```
LnPrice <- log(cars$Price)
cars.lm1 <-
glm(LnPrice~PC1+cars$Mileage+cars$Buick+cars$Cadillac+cars$Chevrolet+cars$Pontiac+cars$SAAB
```

Examine the output of cars.lm1 with the syntax:

```
summary(cars.lm1)
```

b) Use Liter, Cyl, Mileage, Buick, Cadillac, Chevrolet, Pontiac and SAAB in a regression model to
predict the natural log of retail price, LnPrice.

```
cars.lm2 <-
glm(LnPrice~cars$Liter+cars$Cyl+cars$Mileage+cars$Buick+cars$Cadillac+cars$Chevrolet+cars$Pontiac+cars$SAAB)
```

Examine the output of cars.lm2 with the syntax:

```
summary(cars.lm2)
```

Chapter 11 R Instructions

Bayesian Data Analysis: What Colors Come in Your M&M's® Candy Bag?

Activities

2. If using the `MMs` dataset, then read the data into your R session:

```
mms <- read.table("FILE PATH/MMs.txt",header=TRUE,sep="\t")
```

Plot the relative frequency estimates:

```
plot(mms$Total.Number.MMs,mms$Proportion.B.or.O,type="l",
xlab="Sample Size (Number of M&M's)",
ylab="Proportion of Brown or Orange M&M's")
```

Extended Activities

34. To find the parameters of the beta prior distribution, use the R function `beta.param()`:

```
beta.param <- function(min,max) {
  n <- 100
  alpha <- round(seq(1,100,length=n),2)
  beta <-  round(seq(1,100,length=n),2)
  p1.min <- p2.max <- .1
  tol <- seq(0,.1,length=50)

for (k in 1:length(tol)) {
  for (i in 1:n) {
    for (j in 1:n)  {
        q1 <- qbeta(p1.min,alpha[i],beta[j])
        q2 <- qbeta((1-p2.max),alpha[i],beta[j])
   if ((round(q1,2)>(round(min,2)-tol[k]) &&
round(q1,2)<(round(min,2)+tol[k]))
           && (round(q2,2)>(round(max,2)-tol[k]) &&
round(q2,2)<(round(max,2)+tol[k])))
  return(list(alpha=alpha[i],beta=beta[j]))
    }
  }
}
}
```

Within the parentheses, enter the values of α_{min} and α_{max} separated by a comma. For the skeptic, α_{min} =.21 and α_{min} =.29, so the parameters for the Beta prior are found with the following syntax: `beta.param(.21,.29)`

The following output will appear, providing the parameters of the prior distribution:

```
> beta.param(.21,.29)
$alpha
[1] 27
$beta
[1] 81
```

36. For the open-minded individual, $\pi_{min} = .21$ and $\pi_{min} = .5$, so the parameters for the beta prior are found with the following syntax:

```
beta.param(.21,.5)
```

For the believer, $\pi_{min} = .3$ and $\pi_{min} = .7$, so the parameters for the beta prior are found with the following code:

```
beta.param(.3,.7)
```

38. Plot the prior and posterior distributions for the open-minded individual and believer:

```
par(mfrow=c(1,2))
x <- seq(0,1,length=500)
plot(x,dbeta(x,24,43),type="n",ylim=c(0,7),
xlab=expression(pi*" = Population Proportion of Hits"),
ylab="Probability Density Function",main="Open-Minded Individual")
lines(x,dbeta(x,6,11))
lines(x,dbeta(x,24,43),lty=2)
legend(.45,6,c(expression("Prior Distribution for "*pi),expression
("Posterior Distribution for "*pi)),lty=1:2,cex=.7)
plot(x,dbeta(x,23,37),type="n", ylim=c(0,7),
xlab=expression(pi*" = Population Proportion of Hits"),
ylab="Probability Density Function",main="Believer")
lines(x,dbeta(x,5,5))
lines(x,dbeta(x,23,37),lty=2)
legend(.45,6,c(expression("Prior Distribution for "*pi),expression
("Posterior Distribution for "*pi)),lty=1:2,cex=.7)
```

41. The `qbeta()` function is used to find quantiles of the Beta(α,β) distribution. To find the value of L on the Beta(24,33) posterior distribution such that .025 area is to the left of L:

```
qbeta(.025,24,33)
```

To find the value of U on the Beta(24,33) posterior distribution such that .025 area is to the right of U:

```
qbeta(.975,24,33)
```

To find the limits L and U for other beta distributions, simply follow the above syntax, and supply the appropriate values of α and β.

45. Find the posterior probability that $\pi < .5$:

```
pbeta(.5,24,33)
```

46. To find the 95% credible intervals, follow the syntax provided in Question #41.