Statistical Software Camp: Introduction to R

Day 3

August 31, 2009

1 Multivariate Data

1.1 Two-Way Tables

- The function `table(X, Y)` will create a two-way table using two variables `X` and `Y`.

```
> admit <- read.csv("admit.csv", header=T) # Data saved off of Blackboard
> table(admit$female)

 0  1
71 35

> table(admit$score)

 1  2  3  4  5
23 24  2 37 20

> gre.scores <- table(admit$female, admit$score)
> gre.scores # Look at the data again

     1  2  3  4  5
  0 16 16  0 27 12
  1  7  8  2 10  8
```

- The function `prop.table()` will convert a table to a table with proportions.

```
> prop.table(gre.scores)

              1          2          3          4          5
  0 0.15094340 0.15094340 0.00000000 0.25471698 0.11320755
  1 0.06603774 0.07547170 0.01886792 0.09433962 0.07547170
```

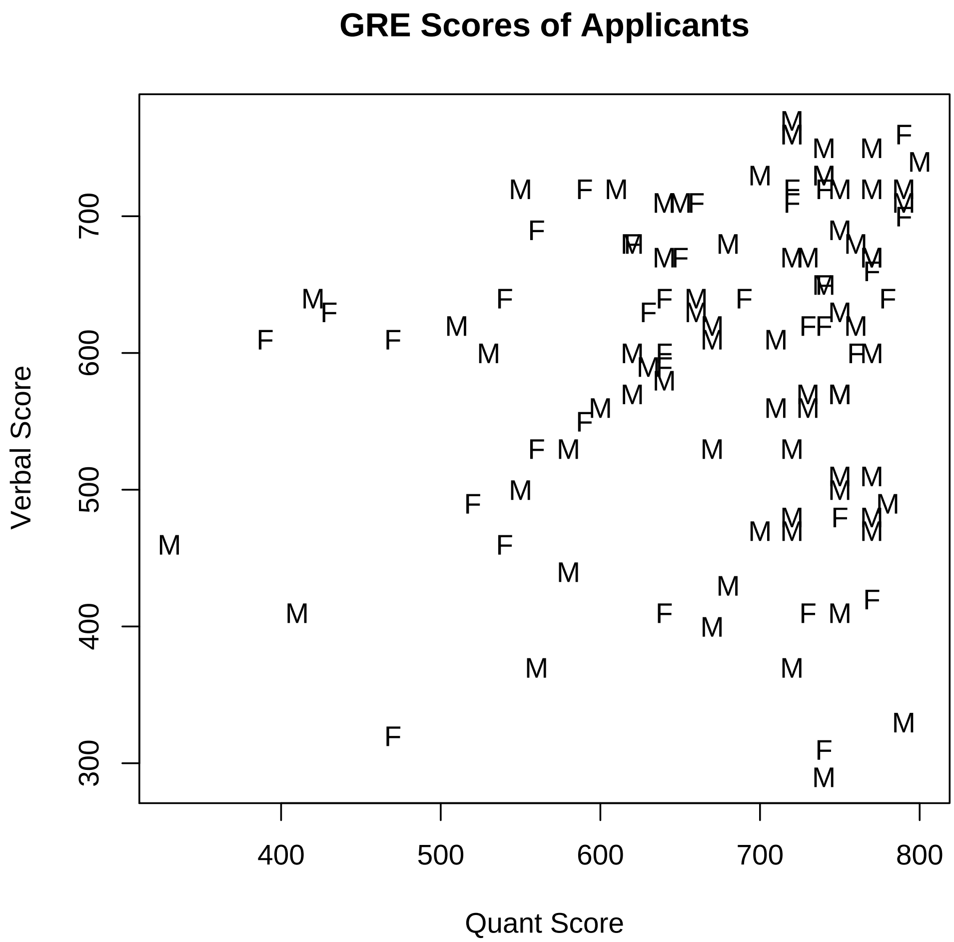
- The function `addmargins()` will append the sums for both rows and columns onto our table

```
> addmargins(gre.scores) # Append the sums for both rows and columns onto our table

        1   2   3   4   5 Sum
  0    16  16   0  27  12  71
  1     7   8   2  10   8  35
  Sum  23  24   2  37  20 106
```

1.2 Graphing Multivariate Data

- The function `plot(x, y, ...)` will create a simple scatterplot, in which the vector `x` is plotted against `y`

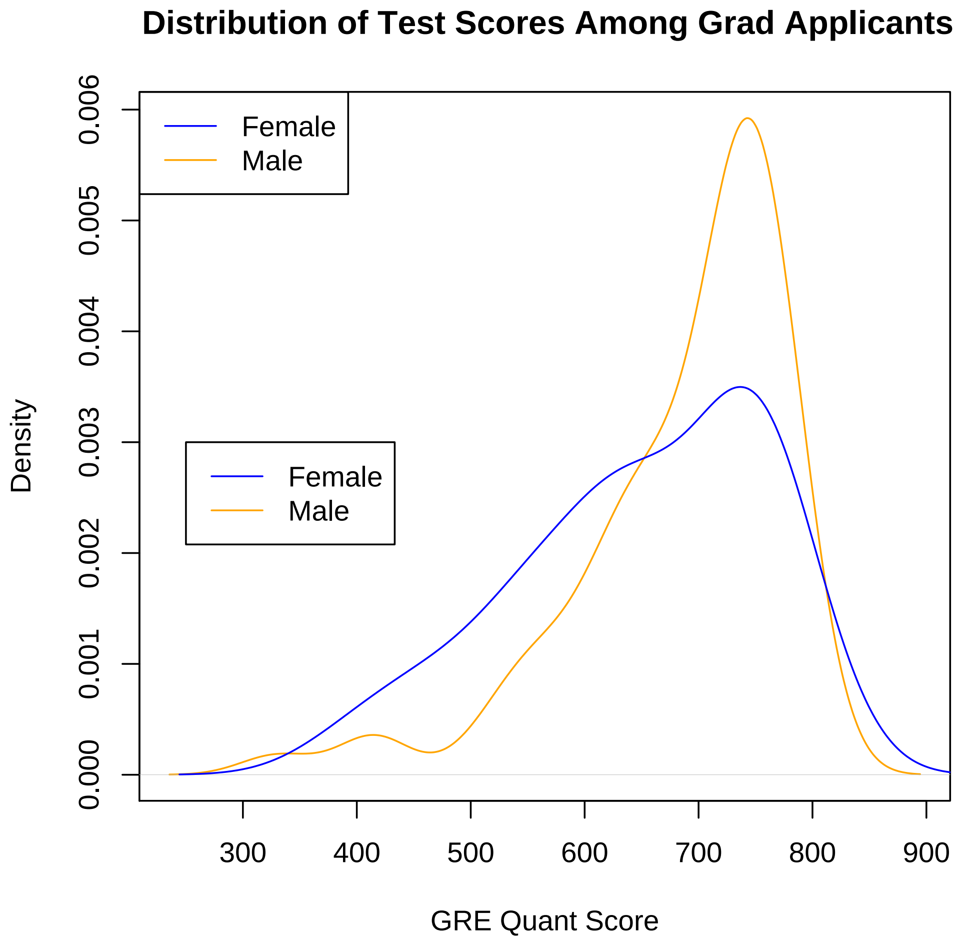
```
> admit$gender <- ifelse(admit$female==1,"F","M") # Creates a new gender variable
> plot(admit$gre.quant, admit$gre.verbal, pch = admit$gender,
+     xlab="Quant Score", ylab="Verbal Score", main="GRE Scores of Applicants")
```



- The function `legend(X, Y, Z)` will add a legend to an existing plot where `X` is the x-coordinate, `Y` is the y-coordinate, and `Z` is a vector of text.

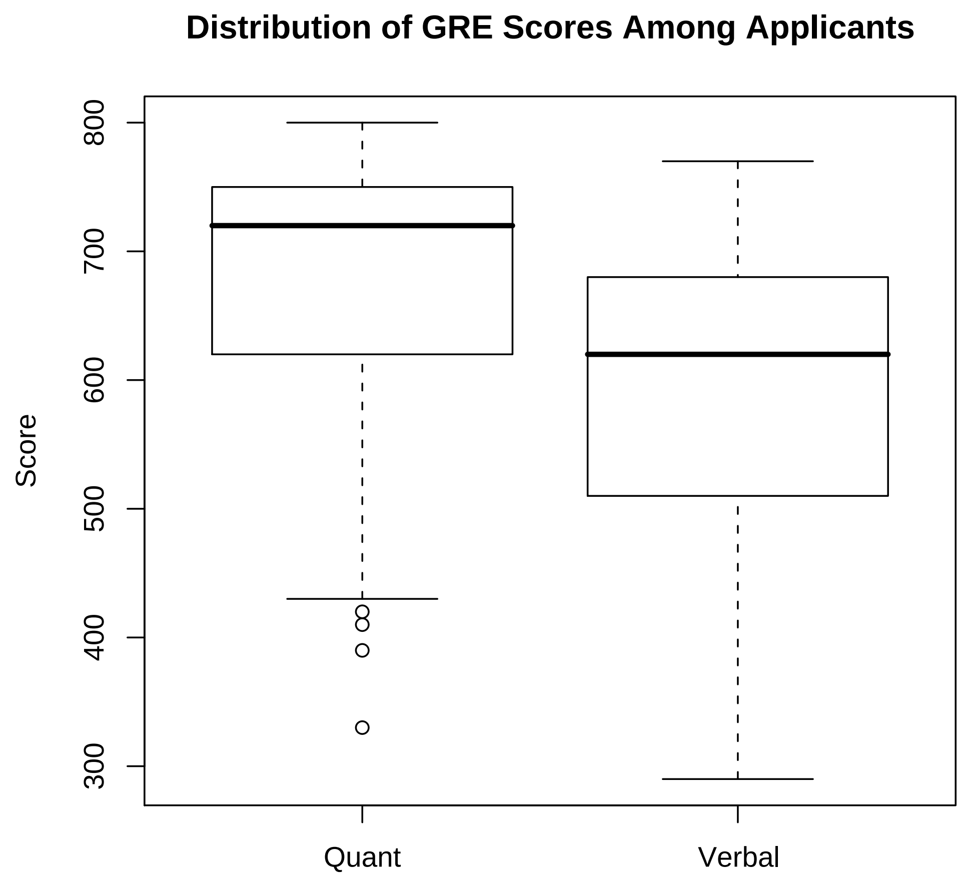
- The `X,Y` argument can be replaced with a keyword indicating location, such as `"topleft"`, `"bottomright"`, etc.

```
> plot(density(admit$gre.quant[admit$female==0]),
+     xlab="GRE Quant Score", ylab="Density",
+     main="Distribution of Test Scores Among Grad Applicants", col="orange")
> lines(density(admit$gre.quant[admit$female==1]), col="blue")
> legend("topleft", c("Female","Male"), lty = c(1,1), col = c("blue","orange"))
> legend(250, 0.003, c("Female","Male"), lty = c(1,1), col = c("blue","orange"))
```



- The function `boxplot(a, b, ...)` will create a side-by-side boxplot for the variables `a` and `b`

```
> # Side-by-side Boxplots
> boxplot(admit$gre.quant, admit$gre.verbal, names=c("Quant", "Verbal"),
+     ylab="Score", main="Distribution of GRE Scores Among Applicants")
```



1.3 Correlation

- Correlation is a measure of the strength and direction of two variables.
- The function `cor(X, Y)` takes in two vectors (`X` and `Y`) and returns their correlation

```
> cor(admit$gre.verbal, admit$gre.quant)

[1] 0.1599913
```

1.4 Linear Regression

- Simple linear regression is a procedure to find the best-fitting line to bivariate data (you will learn the details in Quant 1).
- The function `lm(Y ~ X, data = Z)` regresses a variable `Y` on a variable `X` taken from the data frame `Z`.

```
> # load in the union dataset, downloaded from Blackboard
> union <- read.csv("union.csv", header=T)
> # union: percentage of workers who belong to a union
> # left: extent to which parties of the left have controlled government
```

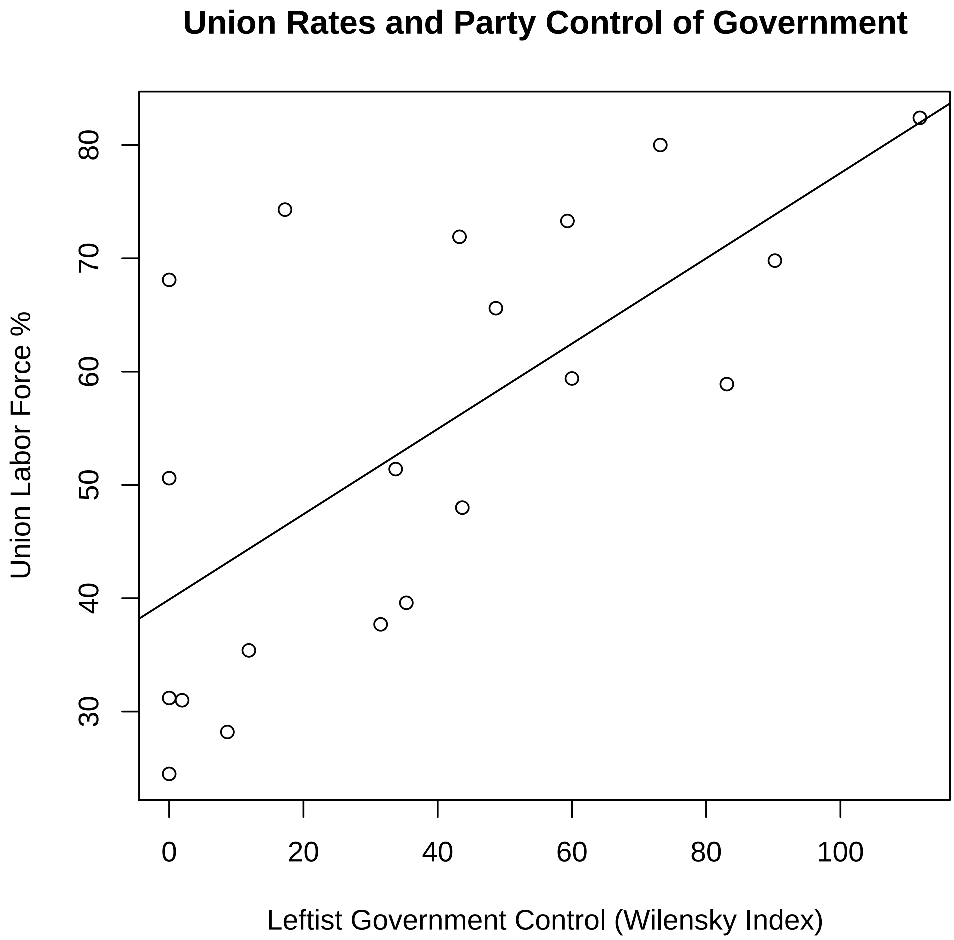
```
> # size: size of the labor force
> # concen: measure of economic concentration in top-4 industries
>
> fit.1 <- lm(union ~ left, data=union)  # Our linear regression
```

- Applying `summary()` to the regression output will produce a summary.
- Applying the function `coef()` to your linear model will output just the coefficient estimates for your regression.

```
> summary(fit.1)

Call:
lm(formula = union ~ left, data = union)

Residuals:
    Min      1Q  Median      3Q     Max
-15.384 -10.269  -3.558  10.808  28.216

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 39.88406    4.81269   8.287 1.48e-07 ***
left         0.37639    0.09619   3.913  0.00102 **
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

Residual standard error: 14.16 on 18 degrees of freedom
Multiple R-squared: 0.4597,       Adjusted R-squared: 0.4296
F-statistic: 15.31 on 1 and 18 DF,  p-value: 0.001019

> coef(fit.1)

(Intercept)        left
 39.8840609   0.3763868
```

- You can add the fitted line to the scatter plot through `abline()`.

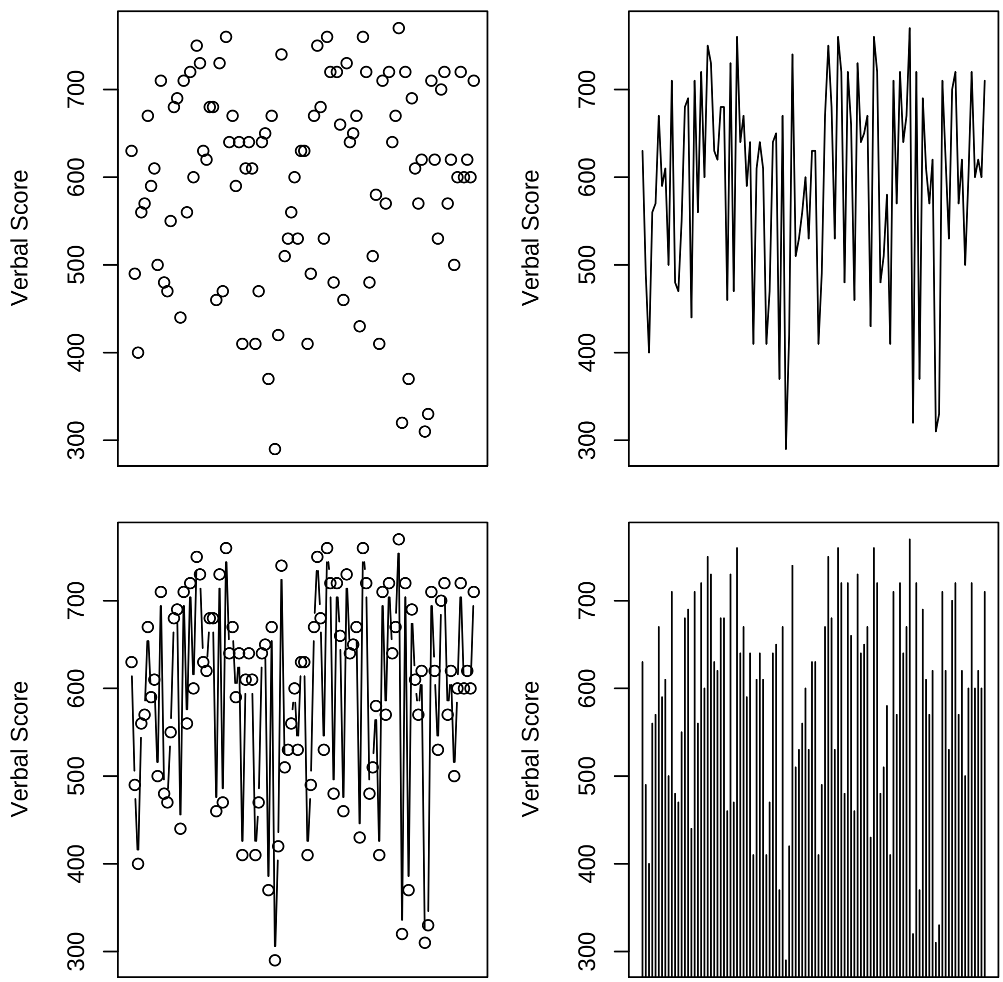
```
> plot(union$left, union$union,
+      xlab="Leftist Government Control (Wilensky Index)",
+      ylab="Union Labor Force %",
+      main="Union Rates and Party Control of Government")
>      # Creates a scatterplot
> abline(fit.1)  # Adds the best-fit line to our plot
> abline(coef(fit.1)) # Equivalent command
```



1.5 Tweaking Graphical Parameters

- To make your graphs look better, you might want to tweak **graphical parameters** via the `par()` function.
- (Review) Setting `mfrow=c(X, Y)` or `mfcol=c(X, Y)` in `par()` will allow you to place multiple (`X` $\times$ `Y`) plots in one graph
- The `cex` parameter changes the size of texts in a plot. The default is `cex = 1`.
- The `mar` parameter decides the size of margin around a plot. It takes a four-element vector `c(b,l,t,r)`; the elements correspond to the bottom, left, top and right margins, respectively, and default to `c(5.1,4.1,2.1,2.1)` where numbers indicate the number of lines.
- The `oma` parameter sets the size of outer margin (i.e. the space common for all the plots) for a graphical device.
- You can suppress the axes by setting `xaxt` and/or `yaxt` to `"n"`.
- Note that the `par()` function outputs the *old* values of the parameters which are overwritten by the call. This can be useful to restore the orginal setting later on.

```
> par(mfrow=c(2,2), mar=c(1,4,1,1), xaxt="n")
> old.par <- par(mfrow=c(2,2), mar=c(1,4,1,1), xaxt="n") # same but saves the original
> plot(admit$gre.verbal, type="p", xlab="", ylab="Verbal Score")
> plot(admit$gre.verbal, type="l", xlab="", ylab="Verbal Score")
> plot(admit$gre.verbal, type="b", xlab="", ylab="Verbal Score")
> plot(admit$gre.verbal, type="h", xlab="", ylab="Verbal Score")
> par(old.par) # restores the original setting
```

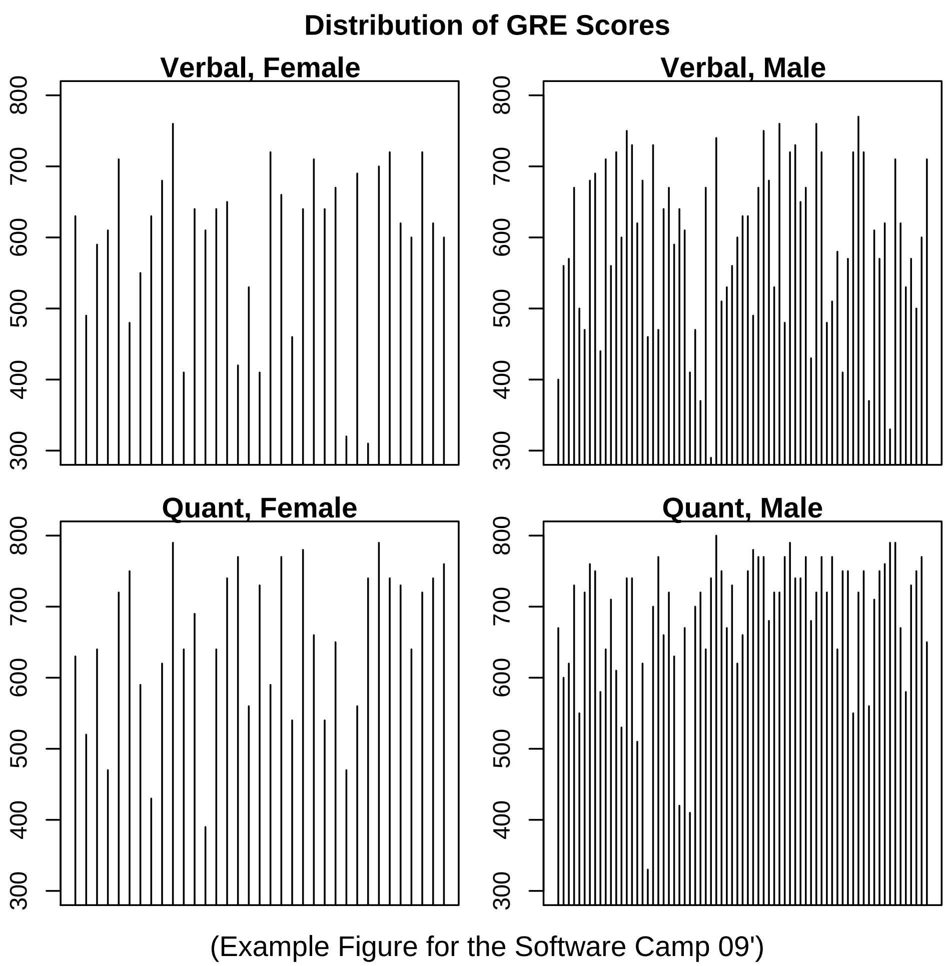


1.6 Adding Texts to a Graph

- (Review) The `text(x, y, yourtext,...)` function adds the text "`yourtext`" to an existing plot at the position specified by `x` and `y`.
- The `title()` function adds titles and/or axis labels to an existing plot with the `main=`, `sub=`, `xlab=` and `ylab=` arguments. Setting `outer = T` places the titles/labels in outer margins.
- Alternatively, the `mtext()` function can be used to add texts to the margins of an existing plot, with the `side=` and `line=` arguments specifying their exact positions. Setting `outer = T` place the text in outer margins.
- Use `title()` to add texts to standard positions; use `mtext()` if you need more flexibility.

- To use mathematical expressions in the text-drawing functions, the `expression()` function is used. Type `help(plotmath)` to find the syntax (it is similar to LaTeX, but different).

```
> verbal.f <- admit$gre.verbal[admit$female==1]
> verbal.m <- admit$gre.verbal[admit$female==0]
> quant.f <- admit$gre.quant[admit$female==1]
> quant.m <- admit$gre.quant[admit$female==0]
> par(mfrow=c(2,2), mar=c(1,2,1,1), oma=c(3,1,2,1), xaxt="n")
> plot(verbal.f, type="h", xlab="", ylab="", ylim=c(300,800), main="Verbal, Female")
> plot(verbal.m, type="h", xlab="", ylab="", ylim=c(300,800), main="Verbal, Male")
> plot(quant.f, type="h", xlab="", ylab="", ylim=c(300,800), main="Quant, Female")
> plot(quant.m, type="h", xlab="", ylab="", ylim=c(300,800), main="Quant, Male")
> title(main="Distribution of GRE Scores", outer=T)
> mtext("(Example Figure for the Software Camp 09')", side=1, outer=T)
```



2 Control Structures

2.1 Loop

- The function `for(i in X)` will create a loop in your programming code where `i` is a counter and `X` is a vector for the counter. That is, the following syntax,

```
for (i in X) {
  blah1...
  blah2...
  ...
}
```

will execute the code chunk, `blah1... blah2... ...`, the same number of times as the length of `X` vector while setting the counter `i` to each element of `X`. You can have as many commands and lines in a loop as you like.

- Braces (`{}`) are used to denote the beginning and end of your loops. If your code chunk only contains one line, you can get away without using the braces. That is,

```
for (i in X)
  blah1...
```

works though it is generally a good idea to keep the braces.

- The function `rep(X,Y)` will create a vector of length Y with each item equal to X.
- The function `print()` will print a formatted object.
- The function `cat()` will concatenate (i.e. paste) a set of texts and/or objects together (each should be separated by a comma) and then print the information to the **R** console.
- Examples:

```
> for (i in 1:3){
+   print(i)
+ }

[1] 1
[1] 2
[1] 3

> x <- c("hey", "Hey", "HEY")
> for (i in x){
+   print(i)
+ }

[1] "hey"
[1] "Hey"
[1] "HEY"

> for (j in 3:5){
+   x <- j*2
+   cat(j, "times 2 is equal to", x, "\n") #\n changes a line
+ }

3 times 2 is equal to 6
4 times 2 is equal to 8
5 times 2 is equal to 10
```

```
> Z <- rep(NA, 10) # Create an empty vector to hold our answer in
> for (j in 1:10){
+   Z[j] <- j*2 # Store the value from each loop into the vector
+ }
> Z

 [1]  2  4  6  8 10 12 14 16 18 20
```

2.2 Conditional Statements

- The following syntax

  ```
  if (X) {
    blah1...
    blah2...
    ...
  }
  ```

  will execute the code chunk, `blah1...  blah2...` if the condition `X` is met. If the condition is not met, then it will not execute that code chunk.

- You can have as many lines in the code chunk as you like. Similar to a loop, if you only have one line in the code chunk, you can omit the braces though it is generally a good idea to have them for the sake of clarity. It is also a good idea to indent the code chunk so that the code is easy to read.

  ```
  > if (3>4) 3*12 #No action takes place because condition isn't met
  > if (5>4) 3*12 #Condition met and R proceeds with computation

  [1] 36

  > # you can use if() within a loop
  > x <- c(1, 5, 4, 2, 3)
  > y <- 0
  > for (i in 1:length(x)) {
  +   if (x[i] > 2) {
  +     y <- y + x[i]
  +   }
  + }
  > y

  [1] 12

  > ## this is the same as
  > sum(x[x > 2])

  [1] 12
  ```

- The following syntax

```
if (X) {
  blah1...
  blah2...
  ...
} else {
  blah3...
  ...
}
```

will execute the code chunk, `blah1... blah2... ...`, if the condition `X` is met. Otherwise, the code chunk, `blah3...`, will be executed.

- You can nest multiple conditional statements. For example,

```
if (X) {
  blah1...
  blah2...
  ...
} else if (Y) {
  blah3...
  ...
} else if (Z) {
  blah4...
  ...
} else {
  blah5...
  ...
}
```

```
> #Add in an else statement
> if (3 > 4){
+    x <- 3*12
+ } else {
+    x <- 3*20
+ }
> x
```

```
[1] 60
```

```
> #use if, else if, and else
> if (3 > 4){
+   a <- "Skip Election"
+ } else if (4 > 3) {
+   a <- "Obama Wins"
+ } else {
+   a <- "McCain Wins"
+ }
> a
```

```
[1] "Obama Wins"
```