Recitation: Midterm Review

We've already covered a lot of material this semester. Let's consider what we'll do with R in a typical application:

- Read in some data – maybe something nicely formatted, maybe something we have to parse.
- Identify interesting subsets of the data (sometimes)
- Fit a model to the data
- And/Or do something over and over again
- Make our results presentable ot others (usually with a plot)

You're asked to do all of these things on the midterm. We'll take a short look at each of them now.

Reading in Data: The Easy Version

Sometimes we have nicely formatted data, in which case we can use `read.table()` or `read.csv()`. These give us a data frame.

```
pros = read.table('http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/prostate.data')
class(pros)
```

```
## [1] "data.frame"
```

Let's compare the top of the file and the resulting data frame:

From the file:

```
    lcavol  lweight age lbph    svi lcp gleason pgg45   lpsa    train
1   -0.579818495    2.769459    50  -1.38629436 0   -1.38629436 6     0 -0.4307829  T
2   -0.994252273    3.319626    58  -1.38629436 0   -1.38629436 6     0 -0.1625189  T
3   -0.510825624    2.691243    74  -1.38629436 0   -1.38629436 7    20 -0.1625189  T
4   -1.203972804    3.282789    58  -1.38629436 0   -1.38629436 6     0 -0.1625189  T
5    0.751416089    3.432373    62  -1.38629436 0   -1.38629436 6     0  0.3715636  T
```

Our data frame:

```
head(pros,5)
```

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
##   train
```

```
## 1  TRUE
## 2  TRUE
## 3  TRUE
## 4  TRUE
## 5  TRUE
```

Reading in Data: Parsing with Regular Expressions

We looked at a trickier example in Homework 4. One entry from the file looked like this:

```
<tr><td colspan="1" rowspan="1" class="date"><div style="display:block;" class="skedStartDateSite">Wed (
NBCSN</td><td colspan="1" rowspan="1" class="skedLinks"><!-- Button Links --><a class="btn" shape="rect
```

We read it in as a raw text file:

```
nhl1516 = readLines('http://www.stat.cmu.edu/~ryantibs/statcomp/homework/NHL1516.html')
length(nhl1516)
```

```
## [1] 3083
```

As a brief example, let's remember how we extracted the date. The relevant part of each entry looked like this:

```
<div style="display:block;" class="skedStartDateSite">Wed Oct 7, 2015</div><div style="display:none;" cl
```

We used regular expressions to isolate the parts we were interested in. What does `.*` do? Why do we match all the way out to `skedStartDateLocal`?

```
datematch = regexpr("DateSite\">.*</div>.*skedStartDateLocal",nhl1516)
dates = regmatches(x=nhl1516,m=datematch)
head(dates)
```

```
## [1] "DateSite\">Wed Oct 7, 2015</div><div style=\"display:none;\" class=\"skedStartDateLocal"
## [2] "DateSite\">Wed Oct 7, 2015</div><div style=\"display:none;\" class=\"skedStartDateLocal"
## [3] "DateSite\">Wed Oct 7, 2015</div><div style=\"display:none;\" class=\"skedStartDateLocal"
## [4] "DateSite\">Wed Oct 7, 2015</div><div style=\"display:none;\" class=\"skedStartDateLocal"
## [5] "DateSite\">Thu Oct 8, 2015</div><div style=\"display:none;\" class=\"skedStartDateLocal"
## [6] "DateSite\">Thu Oct 8, 2015</div><div style=\"display:none;\" class=\"skedStartDateLocal"
```

We can then pull out just the part of the string we want:

```
date = substr(dates,start=11,stop=25)
head(date)
```

```
## [1] "Wed Oct 7, 2015" "Wed Oct 7, 2015" "Wed Oct 7, 2015" "Wed Oct 7, 2015"
## [5] "Thu Oct 8, 2015" "Thu Oct 8, 2015"
```

Interesting Subsets of the Data

It's straightforward to generate vectors of `TRUE` and `FALSE` in R. We can then use those vectors to extract interesting portions of the data.

For example, going back to the prostate data set, suppose we wanted to compare young vs. old patients:

```
young_patients = (pros$age <= 55)
old_patients = (pros$age > 55)
head(young_patients)
```

```
## [1]  TRUE FALSE FALSE FALSE FALSE  TRUE
```

We could then compare the younger and older patients:

```
mean( pros$lweight[young_patients] )
```

```
## [1] 3.349398
```

```
mean( pros$lweight[old_patients] )
```

```
## [1] 3.661074
```

We can also look directly at the data frame, restricted to those values:

```
head(pros[young_patients,])
```

```
##          lcavol  lweight age      lbph svi       lcp gleason pgg45
## 1  -0.579818495 2.769459  50 -1.386294   0 -1.386294       6     0
## 6  -1.049822124 3.228826  50 -1.386294   0 -1.386294       6     0
## 9  -0.776528789 3.539509  47 -1.386294   0 -1.386294       6     0
## 19 -0.562118918 3.267666  41 -1.386294   0 -1.386294       6     0
## 34  0.009950331 3.267666  54 -1.386294   0 -1.386294       6     0
## 49  1.745715531 3.498022  43 -1.386294   0 -1.386294       6     0
##          lpsa train
## 1  -0.4307829  TRUE
## 6   0.7654678  TRUE
## 9   1.0473190 FALSE
## 19  1.5581446  TRUE
## 34  2.0215476 FALSE
## 49  2.5915164 FALSE
```

Why do we need the comma?

Fit a Model

Let's revisit an example from the lecture. Using the prostate cancer data, we had a proposed model:

$$\text{lpsa} = a\text{lcavol} + b\text{lweight}$$

We used `lm` to fit this model to the data:

```
pros.lm.1 = lm(lpsa ~ lcavol + lweight, data=pros)
summary(pros.lm.1)
```

```
##
## Call:
## lm(formula = lpsa ~ lcavol + lweight, data = pros)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.61051 -0.44135 -0.04666  0.53542  1.90424
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.81344    0.65309  -1.246 0.216033
## lcavol       0.65154    0.06693   9.734 6.75e-16 ***
## lweight      0.66472    0.18414   3.610 0.000494 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7419 on 94 degrees of freedom
## Multiple R-squared:  0.5955, Adjusted R-squared:  0.5869
## F-statistic: 69.19 on 2 and 94 DF,  p-value: < 2.2e-16
```

We specified the model with `lpsa ~ lcavol + lweight`.

We could extract the coefficients:

```
coef(pros.lm.1)
```

```
## (Intercept)      lcavol     lweight
##  -0.8134373   0.6515421   0.6647215
```

We could also extract the fitted values for each data point:

```
head(fitted(pros.lm.1))
```

```
##         1         2         3         4         5         6
## 0.6497055 0.7453923 0.6426654 0.5842642 1.9577140 0.6488295
```

Doing Things Over and Over Again

This is one of the reasons we use computers – they're good at doing the same thing over and over again without making mistakes or getting bored. In R, we have three main tools:

- `for` and `while` loops
- Functions from the `apply` family
- Defining our own functions

Loops

`for` loops execute a set of commands a fixed number of times, `while` loops execute a set of commands until a condition is satisfied. Let's look at a simple for loop:

```
for(i in 1:ncol(pros)){
  print( colnames(pros)[i] )
  print( mean(pros[,i]) )
}
```

```
## [1] "lcavol"
## [1] 1.35001
## [1] "lweight"
## [1] 3.628943
## [1] "age"
## [1] 63.86598
## [1] "lbph"
## [1] 0.1003556
## [1] "svi"
## [1] 0.2164948
## [1] "lcp"
## [1] -0.1793656
## [1] "gleason"
## [1] 6.752577
## [1] "pgg45"
## [1] 24.38144
## [1] "lpsa"
## [1] 2.478387
## [1] "train"
## [1] 0.6907216
```

In this case, we could have accomplished a similar result using `lapply`:

```
lapply(pros,mean)
```

```
## $lcavol
## [1] 1.35001
##
## $lweight
## [1] 3.628943
##
## $age
## [1] 63.86598
##
## $lbph
## [1] 0.1003556
##
## $svi
## [1] 0.2164948
##
## $lcp
## [1] -0.1793656
##
```

```
## $gleason
## [1] 6.752577
##
## $pgg45
## [1] 24.38144
##
## $lpsa
## [1] 2.478387
##
## $train
## [1] 0.6907216
```

Functions

Functions are a tool to break big problems into a series of small problems. As an added bonus, you may find that the solutions to the small problems are something you can use over and over again.

Basic syntax:

```
first_col_summary = function(df){
  this.name = colnames(df)[1]
  this.mean = mean(df[,1])
  this.sd   = sd(df[,1])
  return( list(name=this.name,mean = this.mean,sd=this.sd) )
}
first_col_summary(pros)
```

```
## $name
## [1] "lcavol"
##
## $mean
## [1] 1.35001
##
## $sd
## [1] 1.178625
```

A pesky detail

If you create a variable within the function, it won't "survive" outside the function.

```
numberchanger = function(){
  the_best_number = 4
  return(the_best_number)
}
numberchanger()
```

```
## [1] 4
```

```
#the_best_number
```

A pesky detail (cont.)

If a variable outside the function and inside the function have the same name, the function will ignore the "outside" copy of the variable. It won't modify it, and will work with its own copy.

```
the_best_number = 2
numberchanger = function(){
  the_best_number = 4
  return(the_best_number)
}
numberchanger()
```

```
## [1] 4
```

```
the_best_number
```

```
## [1] 2
```

A pesky detail (cont.)

If a variable isn't defined within the function, but is defined outside, the function will refer to the outside value:

```
the_best_number = 2
numberchanger = function(){
  return(the_best_number)
}
numberchanger()
```

```
## [1] 2
```

```
the_best_number
```

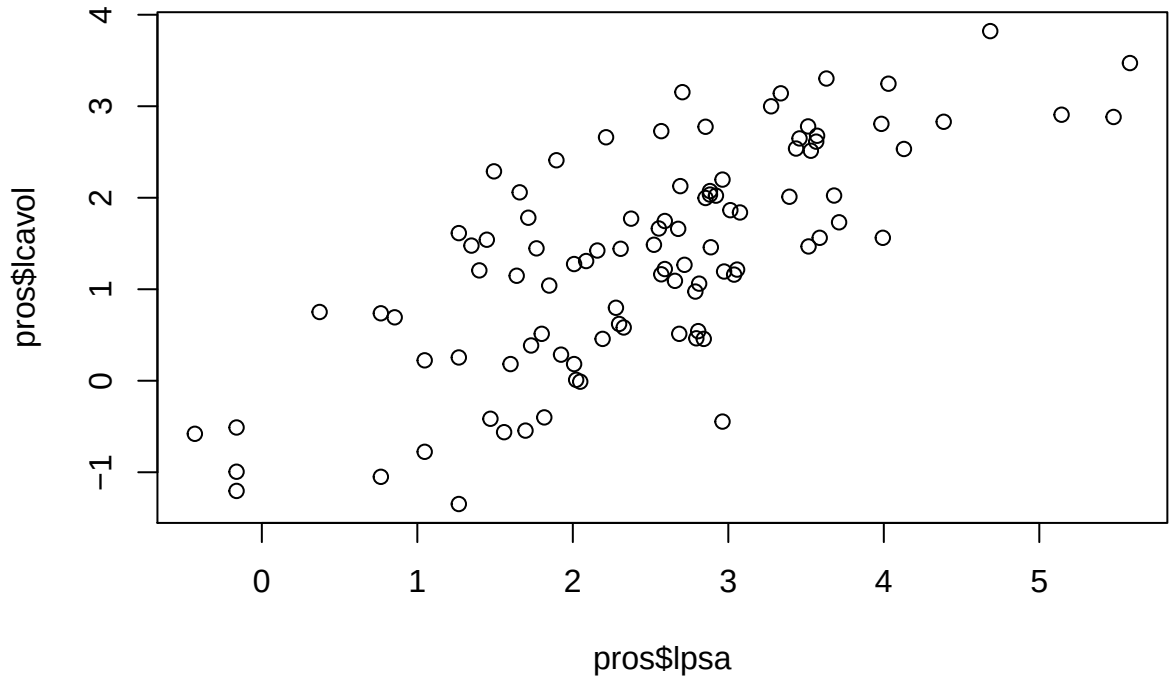
```
## [1] 2
```

But remember, if I try to modify `the_best_number` from inside my function, R will instead create a local copy of `the_best_number`. Generally speaking, getting variable values this way instead of as arguments is a poor programming practice.

Presenting Your Results

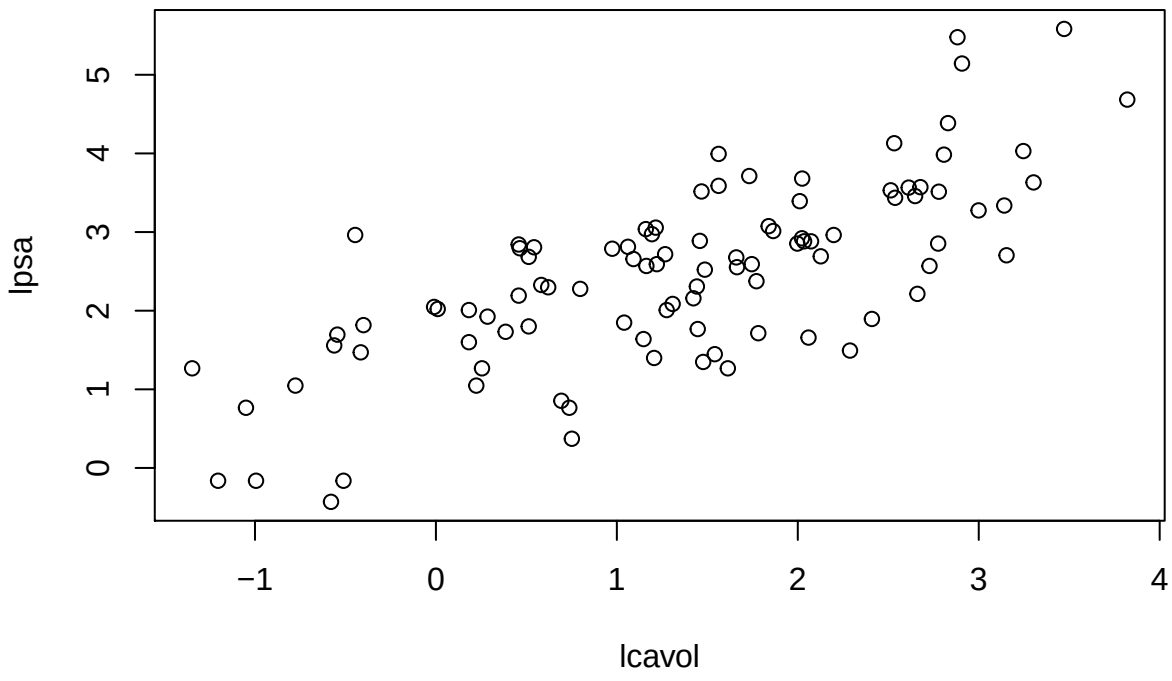
Usually, we ploy `x,y`:

```
plot(pros$lpsa,pros$lcavol)
```



When plotting columns from the same data frame, we can instead plot `y~x`:

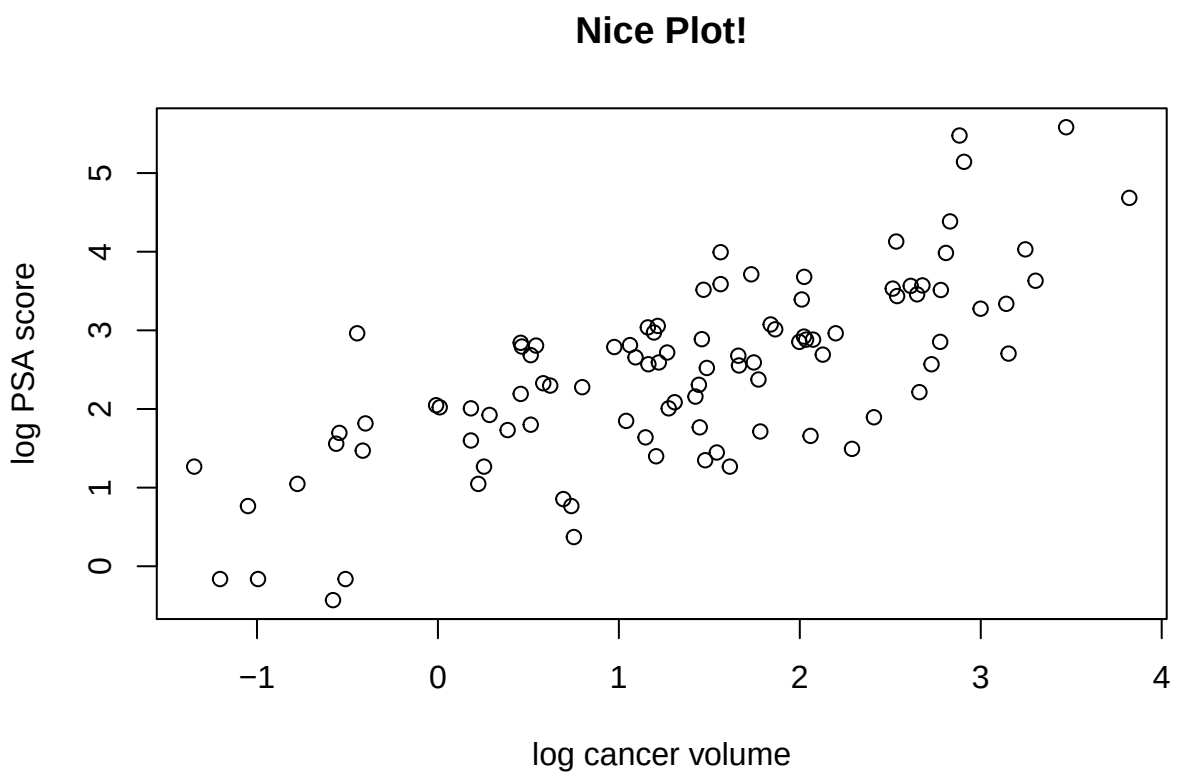
```
plot(lpsa~lcavol,data=pros)
```



Nicer Plots

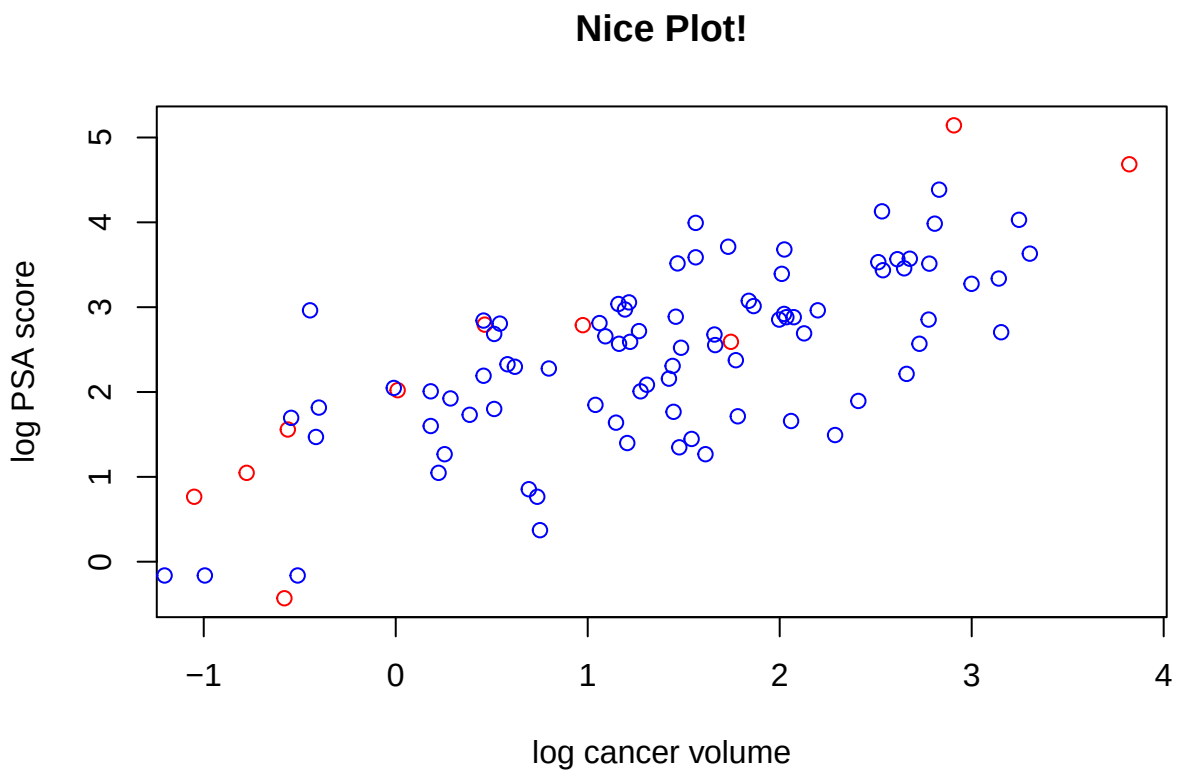
Label everything:

```
plot(lpsa~lcavol,data=pros,xlab='log cancer volume',ylab='log PSA score',main='Nice Plot!')
```



Use color to make your plots more informative:

```
plot(lpsa~lcavol,data=pros[young_patients,],xlab='log cancer volume',ylab='log PSA score',main='Nice Pl
points(pros$lcavol[old_patients],pros$lpsa[old_patients],col='blue')
```



And don't forget how to plot things side-by-side by setting `mfrow`:

```
par(mfrow=c(1,2))
plot(lpsa~lcavol,data=pros,xlab='log cancer volume',ylab='log PSA score',main='Super')
plot(lpsa~lweight,data=pros,xlab='log cancer weight',ylab='log PSA score',main='Sweet!')
```