StatR 101: Winter 2013

Homework 01

Rod Doe

Thursday, January 24, 2012

Assumptions, Properties, and Behavior of Linear Regression

- Assume that data are “dirty”, badly-formed, or contain some errors.
- Do the data seem reasonable?
    - Is there any evidence of instrument errors or data entry errors?
- Outliers should be investigated, as they will skew your results.
    - It would be a shame to skew your model because somebody entered 123 when they meant 1.23.
- If you decide to reject data:
    - Identify the rejected data
    - Clearly explain the rationale for rejection.
- While some say that this is cheating, I like to run a cor(dataset).
    - It shows the r values of each covariate against every other covariate.
    - Sometimes it just helps me to confirm my comprehension of the data.
        - For example, in the Boston data set, there was a large-ish r value for (jobdist ~ biglots). That made sense to me – live farther away to have a big lot.
- Fear interactions as they can prove difficult to explain.
    - Do not use interactions between highly correlated covariates.
    - This makes me wonder when to use any interactions.

R Trick/Annoyance of the Week – Replace the histograms in pairsPlus with Density plots.

I added function panel.density:

```
panel.density <- function(x,right=FALSE,diagCol=5,linefun=mean, ...)
{
    # Get the current value of the par(usr) configuration.
      # This is a vector of the form c(x1, x2, y1, y2) giving the extremes of
      # the user coordinates of the plotting region.
      # Upon function exit, restore the value of par to the initial value.
      usr <- par("usr"); on.exit(par(usr))

      # Set the value of the usr vector.
      par(usr = c(usr[1:2], 0, 1.5)
```

```
	# Get the density return value.
	d = density(x)
	y = d$y / max(d$y)
	# Plot the density function x versus y values.
	lines(d$x, y)

	# Draw a vertical line at the mean(x).
	abline(v=linefun(x),col=2)
}
```

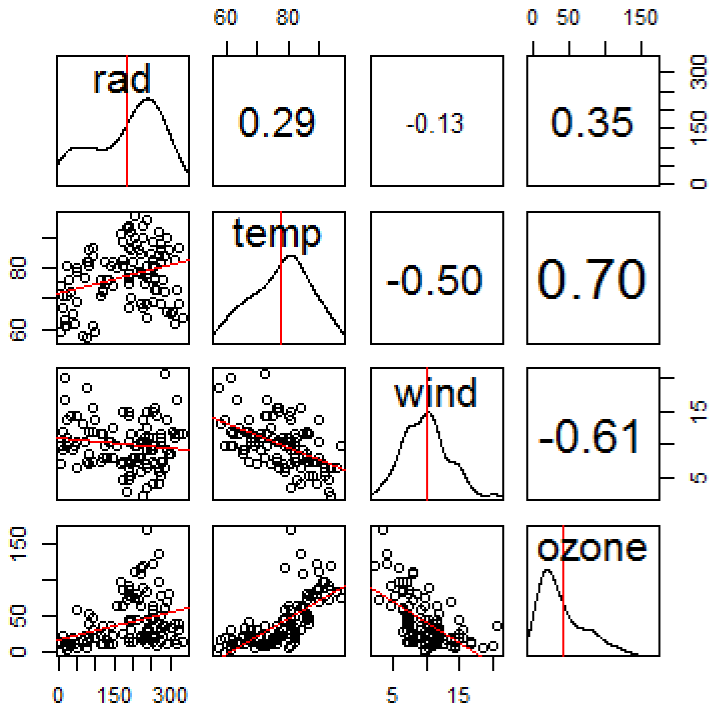
and added a new pairsPlusPlus function to augment pairsPlus (which augments pairs):

```
pairsPlusPlus<-function(x, diag.panel=panel.density, diagCol=4,
fitcurve='linear',...)
{
	pairs(x, diag.panel=diag.panel, upper.panel=panel.cor,
lower.panel=panel.scatter,...)
}
```

To test it, I used a simple, clean data set from Michael Crawley at Imperial College London. He has a good explanation of what's interesting in the data set. It's nice to have an informed opinion, even if it's borrowed.

```
setwd("C:/Users/Rod/SkyDrive/R/Crawley/data")
ozone.pollution = read.table("ozone.data.txt", header=T)
str(ozone.pollution)
pairsPlusPlus(ozone.pollution, panel=panel.smooth)
```

Here is the result:



Variance Inflation Factor

This function was tricky to write. The math is simple enough, but creation of a dynamic string to express the various linear regression models was, well, interesting. This implementation works nicely:

```
# Returns the Variance Inflation Factor for each covariate
# in a linear regression model.
# Values over 5 merit concern.
vif <- function(df) { # df is of type data.frame.
	# Get the count of columns in the data frame.
	covariate = names(df)
	nCols = length(covariate)
	colIndices = 1:nCols

	# Results go here.
	vif = numeric(nCols)

	for (i in 1:nCols) {
		# The i'th column in the data frame is the outcome.
		outcomeName = covariate[i]

		# All other columns are covariates.
		covariateNames = covariate[which(colIndices != i)]
```

```
            # Construct a string that will be the lm formula.
            formulaText = paste(outcomeName,  " ~ ", paste(covariateNames,
collapse="+"))

            # Generate the linear model with the formula string.
            # The special part here is the as.formula function.
            # Capture the result.
            smry = summary( lm(as.formula(formulaText), data=df) )

            # Capture r-squared from from the model.
            rsq = smry$r.squared

            # Calculate and store the variance inflation factor
            # for this outcome.
            vif[i] =  1 / (1 - rsq)
      }

      result = cbind(covariate, vif)

      result
}
```

And here is what it produces with Crawley's ozone data set:

```
> vif(df)
  covariate vif
[1,] "rad"    "1.16337640696268"
[2,] "temp"   "1.99908916725127"
[3,] "wind"   "1.6526121902541"
[4,] "ozone"  "2.53943811529808"
```

Here are the variance inflation factors for the Boston data set:

```
> vif(boston)
      covariate    vif
 [1,] "crime"      "1.65195760315649"
 [2,] "biglots"    "2.31050874422489"
 [3,] "indust"     "3.68906987901752"
 [4,] "river"      "1.08841331702525"
 [5,] "nox"        "4.4255812321449"
 [6,] "rooms"      "2.19134388508719"
 [7,] "old"        "3.06811296335649"
 [8,] "jobdist"    "4.38275313108336"
 [9,] "tax"        "3.35203717389818"
[10,] "teachratio" "1.87275903617881"
[11,] "lowSES"     "3.50551499118306"
[12,] "homeval"    "3.62613212105399"
```

```
Here are the vif values from the HH library:
> library(HH)
Loading required package: lattice
.
.
Warning messages:
1: package ‘HH’ was built under R version 2.15.2
.
.
6: package ‘latticeExtra’ was built under R version 2.15.2
> vif(boston)
     crime    biglots     indust      river        nox      rooms        old 
  1.651958   2.310509   3.689070   1.088413   4.425581   2.191344   3.068113 
   jobdist        tax teachratio     lowSES    homeval 
  4.382753   3.352037   1.872759   3.505515   3.626132
```

The HH library results and my results match. Sweet!

I then found a version of vif in the car package. Unlike my vif and the HH vif, which take a data frame argument, the car implementation takes a model object as an argument. Here are its results:

```
package ‘car’ was built under R version 2.15.2
> vif(boston)
Error in UseMethod("vif") : 
  no applicable method for 'vif' applied to an object of class "data.frame"
> vif
function (mod, ...) 
{
    UseMethod("vif")
}
<bytecode: 0x0000000006c14448>
<environment: namespace:car>
> full.model = lm(formula = homeval ~ crime + biglots + indust + river + nox 
+ 
+     rooms + old + jobdist + tax + teachratio + lowSES, data = boston)
> 
> vif(full.model)
     crime    biglots     indust      river        nox      rooms        old 
  1.630791   2.272178   3.681575   1.059516   4.272901   1.859188   3.068076 
   jobdist        tax teachratio     lowSES 
  3.953227   3.351894   1.734159   2.864280
```

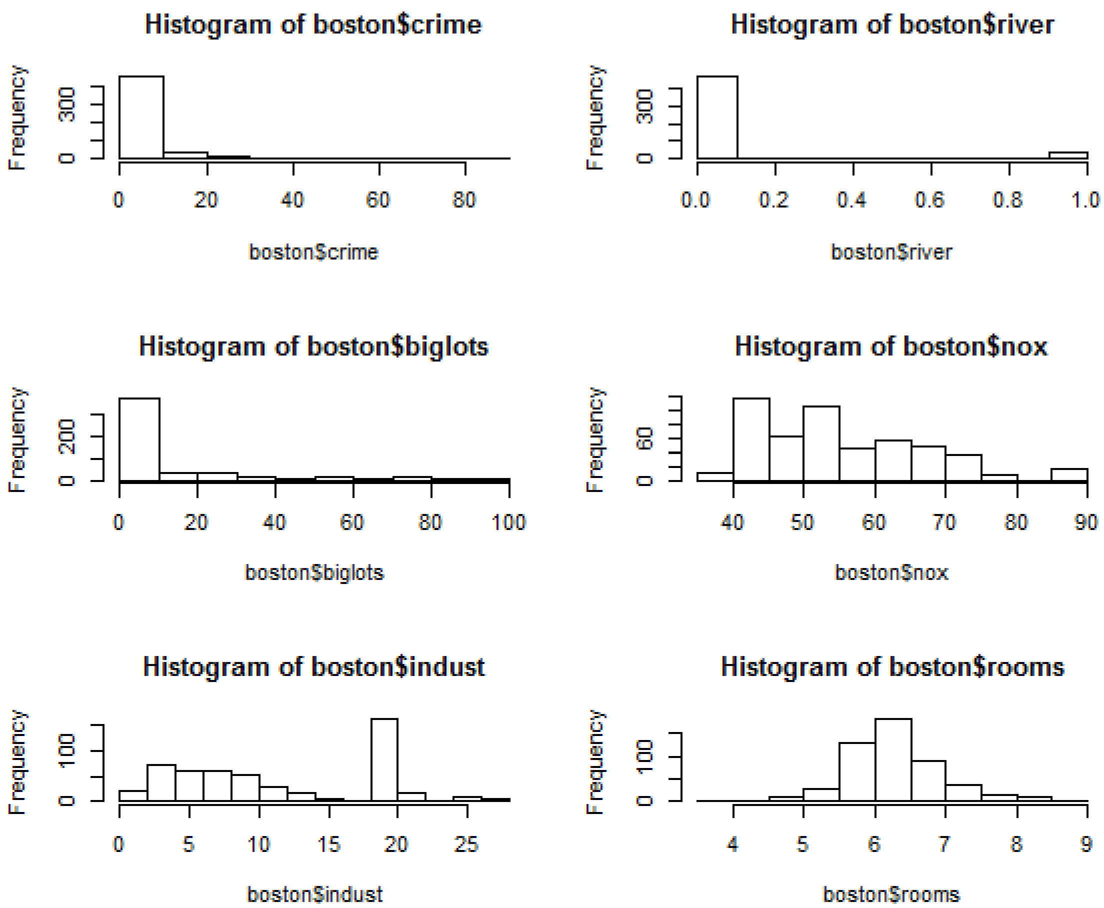
The car version of v produced results that differed slightly from the HH library results and my results. It was less intuitive to invoke it with a model object argument. The authors probably had a good reason.

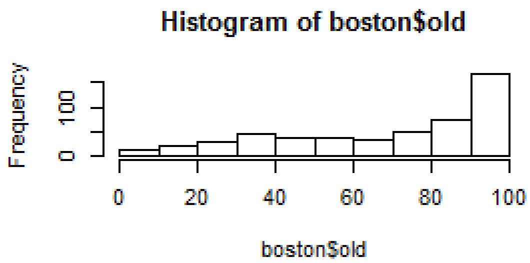
Boston

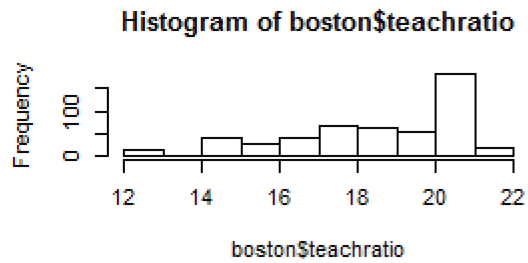
My first step was to do a visual examination of the data. I noted no missing data and nothing that looked, at first glance, to be like the bad guy in the movie Fargo, “funny-lookin’”.

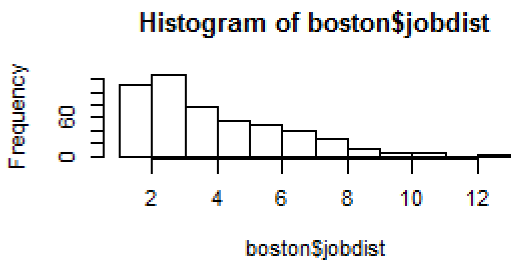
I also downloaded and skimmed an interesting text “Practical Regression and Anova using R” from here: http://csyue.nccu.edu.tw/Practical%20Regression%20and%20Anova%20using%20R.pdf

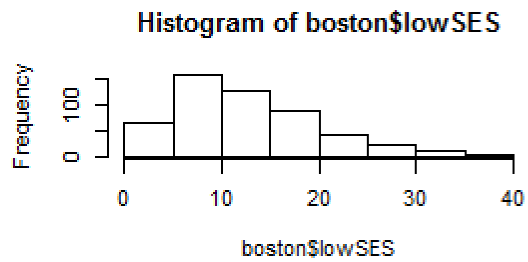
Next, I plotted some histograms of the data, just to see what’s what.

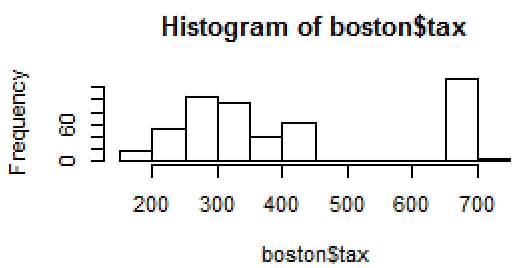


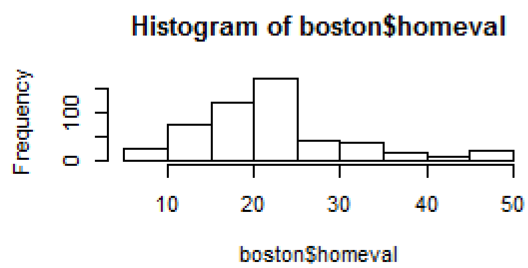








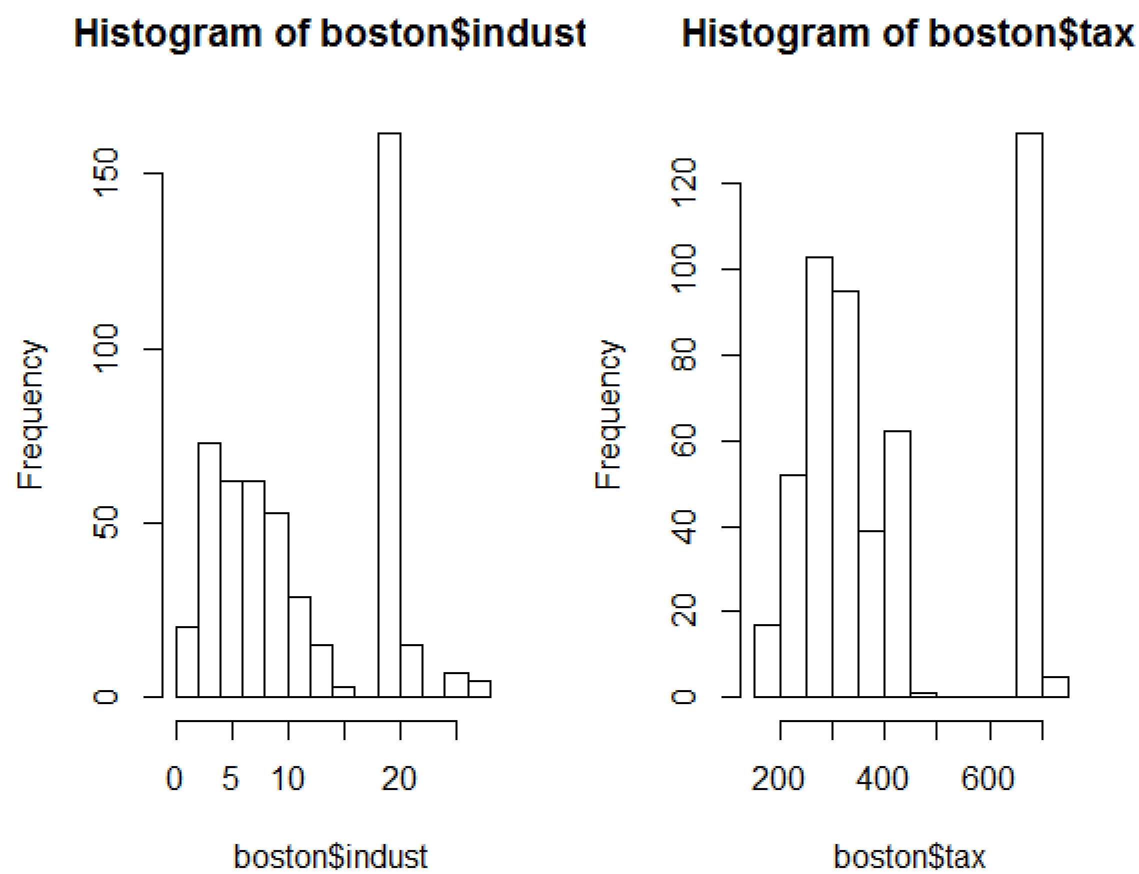




Some of these histograms were troubling, and I wish I had more information about the descriptions and units of the covariates. The gap in the tax histogram and the peak in the “indust” histogram were causes for concern. I would like to talk to the client to get more information. Absent other information, I have no reason to reject data at this point. The tax rate could be real, and the indust could be the effect of an industrial site that provides jobs for the area.

Anecdote

Once, while visiting Boston, I encountered a rusty, sprawling industrial site on the waterfront that looked like something from a Dickens novel. It was the New England Confectionary Company, the source of those tasty-but-probably-bad-for-you Necco wafers. Based upon its appearance, I could not imagine that plant producing anything more edible than a brake pad. I haven’t touched Necco wafers since. I ate them all the time as a kid. I might be doomed.



When modeling, I like to start with a full model, and then look at the summary report to see what I can take out of the model. Given the principle of Occam's Razor, if two models produce similar results, the simpler one should be preferred.

Here is the summary from the full model:

```
> summary(full.model)

Call:
lm(formula = homeval ~ crime + biglots + indust + river + nox +
    rooms + old + jobdist + tax + teachratio + lowSES, data = boston)

Residuals:
     Min       1Q   Median       3Q      Max
-15.4482  -2.9644  -0.6383   1.8092  27.7361

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 34.7760981  4.7588567   7.308 1.10e-12 ***
crime       -0.0816927  0.0322618  -2.532 0.011644 *
```

```
biglots      0.0405441  0.0140447   2.887 0.004062 **
indust      -0.0609507  0.0607766  -1.003 0.316417
river        3.2324756  0.8806352   3.671 0.000268 ***
nox         -0.1628622  0.0387639  -4.201 3.15e-05 ***
rooms        3.9616965  0.4217050   9.394  < 2e-16 ***
old         -0.0010472  0.0135219  -0.077 0.938302
jobdist     -1.5032294  0.2051838  -7.326 9.71e-13 ***
tax          0.0003432  0.0023606   0.145 0.884456
teachratio  -0.8305470  0.1321792  -6.283 7.27e-10 ***
lowSES      -0.5415955  0.0515004 -10.516  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.883 on 494 degrees of freedom
Multiple R-squared: 0.7242,     Adjusted R-squared: 0.7181
F-statistic: 117.9 on 11 and 494 DF,  p-value: < 2.2e-16
```

I was happy to see that my worry points (indust, tax) were not significant contributors to the model.

Next, I used the step function in R to refine the full model. Here is the summary from the reduced model:

```
> summary(reduced.model)

Call:
lm(formula = homeval ~ crime + biglots + river + nox + rooms +
    jobdist + teachratio + lowSES, data = boston)

Residuals:
    Min      1Q  Median      3Q     Max
-15.5219  -2.9766  -0.6161   1.7256  27.6578

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 34.99981    4.57425   7.651 1.04e-13 ***
crime       -0.08042    0.02998  -2.682 0.007556 **
biglots      0.04109    0.01353   3.036 0.002522 **
river        3.17045    0.87455   3.625 0.000318 ***
nox         -0.17719    0.03216  -5.509 5.80e-08 ***
rooms        4.00998    0.40862   9.814  < 2e-16 ***
jobdist     -1.45011    0.19012  -7.628 1.23e-13 ***
teachratio  -0.85618    0.11820  -7.243 1.68e-12 ***
lowSES      -0.54777    0.04819 -11.366  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

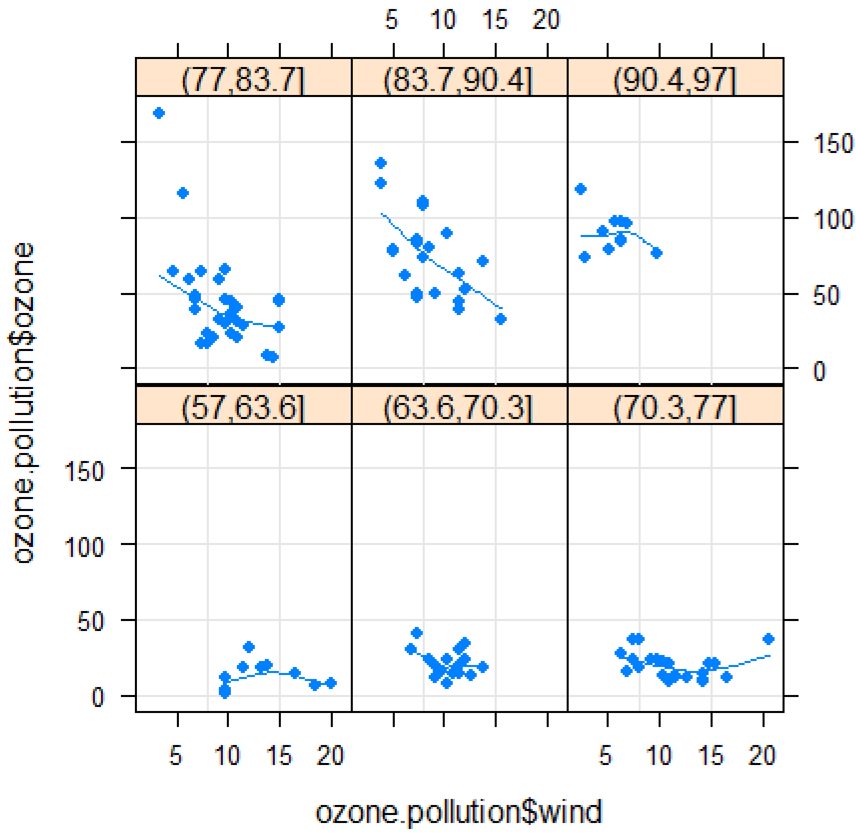
Residual standard error: 4.874 on 497 degrees of freedom
Multiple R-squared: 0.7236,   Adjusted R-squared: 0.7192
F-statistic: 162.7 on 8 and 497 DF, p-value: < 2.2e-16

The reduced model removes covariates (indust, old, tax) as low contributors.

Lattice: xyplot

I monkeyed with this for quite a while and finally got it to produce something. The encapsulation on this function is not exactly crisp.   I need to spend more time with it to understand what capability it brings that warrants its weirdness.

```
library(lattice)
xyplot(ozone.pollution$ozone ~ ozone.pollution$wind |
cut(ozone.pollution$temp, 6),
      panel = function(x, y)
      {
            panel.grid(h=-1, v=2)
            panel.xyplot(x, y, pch=16)
            panel.loess(x, y, span=1 )
      }
  )
```


(77,83.7]
(83.7,90.4]
(90.4,97]
(57,63.6]
(63.6,70.3]
(70.3,77]
ozone.pollution$ozone
ozone.pollution$wind
0
50
100
150
5
10
15
20