UWEO StatR201 – Winter 2013: Homework 1 Key

Assaf Oron, assaf@uw.edu

1. ***Assumptions, Properties, Behavior of multiple linear regression.***

Perhaps this was overkill as your first HW question. But this class being a foray into deeper and deeper "black box" methods, it might be useful as a tool to keep one's head straight, and realize that most methods have conditions and circumstances when they work nicely – and others when they don't

Assumptions might be explicit and implicit. They establish the model's world, and within it they do not require proof. However, in the real world, we want to know how to check the assumptions' validity, and to have an idea what happens when they fail.

Properties can be proven based on the assumptions. This distinguishes them from **behavior**, which is how the method plays out in practice – although the line is often blurred. The quirks and limitations of methods usually fall under behavior.

It is not very important to classify everything about a method into these three rubrics. It is more important to understand how the interplay between them works in general, and to be able to choose the right tool for the job, and make sure to use it on its best terms. So... here goes, for linear regression, a list that by definition is never complete.

<u>Assumptions</u>

- $E[y]=X\beta$, i.e., the mean response is exactly a linear additive function of the predictor covariates *(which might themselves be nonlinear transformations of the original variables)*
- *(a corollary)* There is no covariate missing from the model, especially not one that is correlated with the ones in the model.
- **The residuals' basic three:** the randomness in y around this mean trend, is normal, i.i.d. and constant.
- The covariate values, in their range and distribution, represent our region of interest.

<u>Properties</u>

- The MLE for the statistical regression problem as defined above, is also the least-squares minimizer.
- The solution ("beta-hat" vector) is linear in y, and includes a matrix inverse (the inverse of X^tX).
- This makes the solution essentially a weighted average of observations, and therefore it shares the behavior/properties of averages (asymptotic Normality regardless of the Normality of errors; precision vs. lack of robustness). Its variance is also a function of the same matrix inverse.
- The solution is also the Best Linear Unbiased Estimator (BLUE) even if the Normality

assumption is violated, as long as the noise is still i.i.d. constant-variance, and meets the conditions of the Central Limit Theorem (finite expectation and variance).

- If the Normality assumption holds, but a certain covariate has no net effect, then its estimate will be t-distributed with n-p degrees of freedom (with p being the # of terms in the model).
- If the assumptions hold, the effect estimates ("beta-hat") and the estimate of residual variance ("sigma-square-hat") are statistically independent.
- If all assumptions hold, then the *Studentized* residuals are i.i.d t-distributed, with n-p-1 d.f.
- The fitted (predicted) values at the observations, are linear combinations of the observations.
- The asymptotic *variance* of the predicted regression mean increases quadratically, as a function of the covariate values' distance from the mean of X.
- As one adds terms to the model, the maximized likelihood always increases while the least-square error sum decreases. Conversely, the sum of squares of the covariates' SEs increases.

Behavior

- ***Association does NOT imply Causation:*** if an effect estimate is significantly different from zero, this does *not* mean that the x in question "causes y." Rather, it provides *evidence* that the two are *linearly associated*, in the presence of all other variables in the model.
- The model (X) matrix has a column of 1's signifying the intercept (unless the intercept is specifically suppressed).
- Multi-category covariates with k>2 levels require k-1 predictor terms. The most common coding (although not the only possible one) uses one column per level, with 1 when the data point has this level and 0 otherwise. One level known as the *base* or *reference* level, is *not* coded – it is subsumed in the intercept.
- A continuous covariate's effect estimate is of the associated average increase in y per unit increase in that specific x – *while holding all other covariates constant.*
- A multi-category covariate's effect estimate is of the associated average difference in y compared to that covariate's base level – *while holding all other covariates constant.*
- If a two-way *interaction* is included in the model, its effect estimates *the change in the effect* of one covariate, per unit change in the other – or vice versa. Meanwhile, the separate estimate of each of the participating covariates' effect, should now be interpreted while holding the other covariate *constant at zero.*
- The intercept is interpreted as the average of y when all terms in the model are zero.
- *Linear* transformations of y or the x's, don't affect the predictions/fits, or the essence of the solution (besides its units). *Hat-tip Ben!*
- The numerical summaries alone do not necessarily tell us when assumptions such as linearity or Normality are violated.

- Including highly collinear covariates tends to inflate the SEs of all effects, but especially of the collinear ones.
- Because of the least-squares nature of the solution, outliers in y tend to exert a disproportionate influence on estimates and predictions.
- Outliers in x might affect the solution even more strongly, since they provide the only information in their region of the covariate space, and since they have more *leverage*.
-

Questions 2 and 3 demonstrate typical ways in which experienced users of R interact with existing resources. In #2, we take a standard off-the-shelf function and tweak it. My implementation is admittedly limited (i.e., in order to fully control symbols and colors in all sub-plot types, a deeper dig into the original code might be needed). However, since this is only a quick-and-dirty visualization tool, industrial-grade coding is not required. On the other hand, as the examples below show, with very modest efforts the sky is literally the limit, and you emerge with a scatterplot-matrix function as good as anything to be found out there.

2. ***a. In the*** **`'pairsPlus'`** ***function, add an option that will replace the histograms on the diagonal with density plots.***

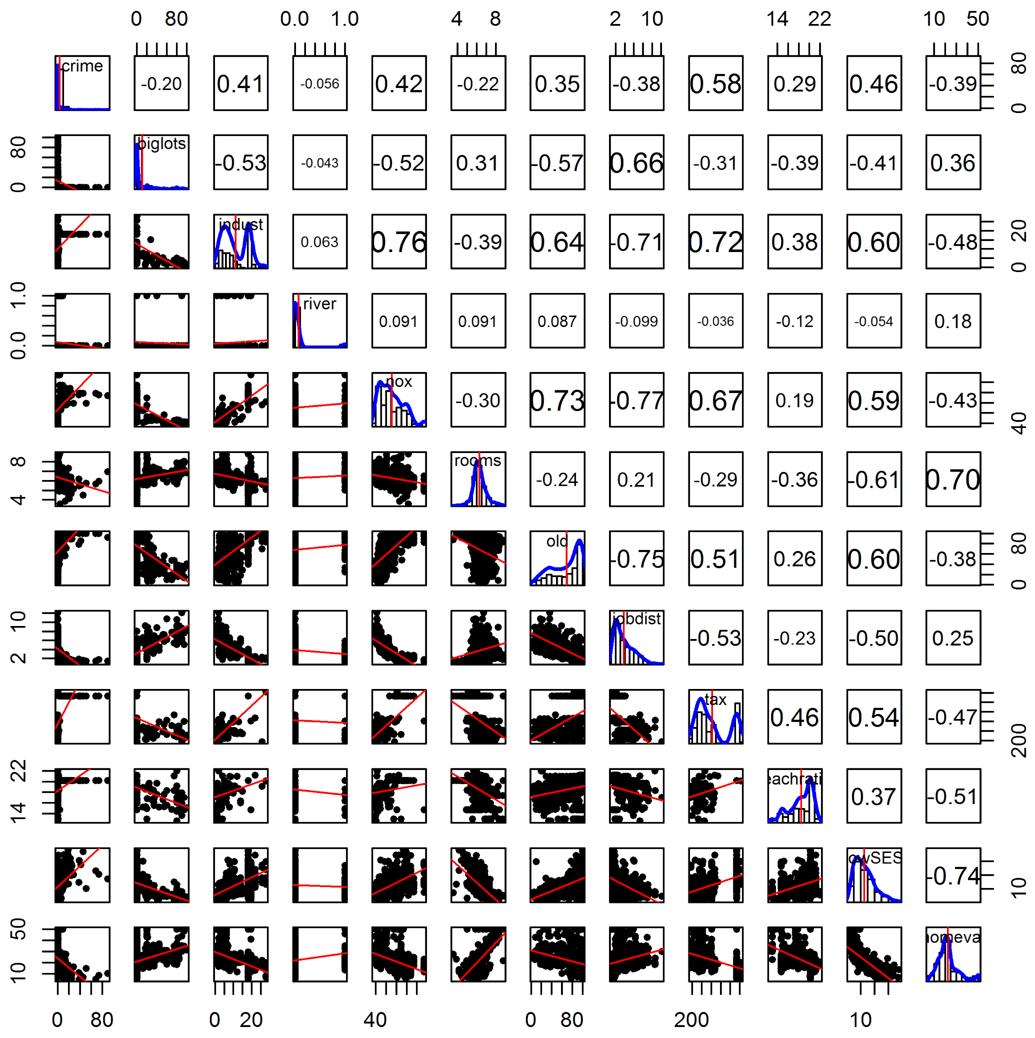
I liked Abderrazak's solution of keeping the histogram while overlaying it with the density curve. Helps show gaps in the data and other features. Here's my version based on his idea.

```
panel.dens <- function(x,diagCol=4,linefun=mean,...)
{
    usr <- par("usr"); on.exit(par(usr))
    par(usr = c(usr[1:2], 0, 1.2))

    h <- hist(x,plot = FALSE)
    breaks <- h$breaks; nB <- length(breaks)
    y <- h$counts; y <- 0.9*y/max(y)
    rect(breaks[-nB], 0, breaks[-1], y,col=0,...)

    d <- density(x)
    lines(d$x,d$y/max(d$y),lwd=2,col=diagCol)

    abline(v=linefun(x),col=2)
}
```

Notes:

i. The overall function has only one call to a high-level plotting function (i.e., one that creates the entire plotting window, etc.). Inside the diagonal, we are only adding and controlling graphics within a single diagonal window. Hence, we cannot plot directly via `hist` or `density.` Rather, we call these functions with plotting disabled, and add the calculated elements ourselves "by hand."

ii. `'usr'`, one of the parameters in the amazingly diverse `par`, controls the overall plotting boundaries in units of the actual x and y variables. Calling `par('usr')`, as the function does shortly after entry, returns the current values. The `on.exit()` bit makes sure that after drawing the diagonal panes, the function resumes the previous boundaries. This enables the scatterplots on the off-diagonal to use different boundaries. The safest way to get the diagonal boundaries right is to normalize both boundaries and plotting in compatible manner.



b*. Add an option to draw '`qqnorm`' curves on the diagonal, instead of density plots.

Rita solved this one, which is fairly straightforward. Again, here's my slightly different version.

```
panel.qq <- function(x,right=FALSE,diagCol=4,...)
{
      usr <- par("usr"); on.exit(par(usr))

      qq.value <- qqnorm(x, plot=FALSE)
      par(usr = c(min(qq.value$x), max(qq.value$x), min(qq.value$y), max(qq.value$y)) )

      points(qq.value$x, qq.value$y, col=diagCol,cex=0.7,pch=19)
}
```

c*. The 'boston' dataset has 12 variables, making the panes rather small. Add an option to split the display into several pages, with user input ('press any key...' etc.). for advancing a page. However, make sure that the y variable (or modeled outcome – in the 'boston' case, home values) still shows as the last variable in every page.

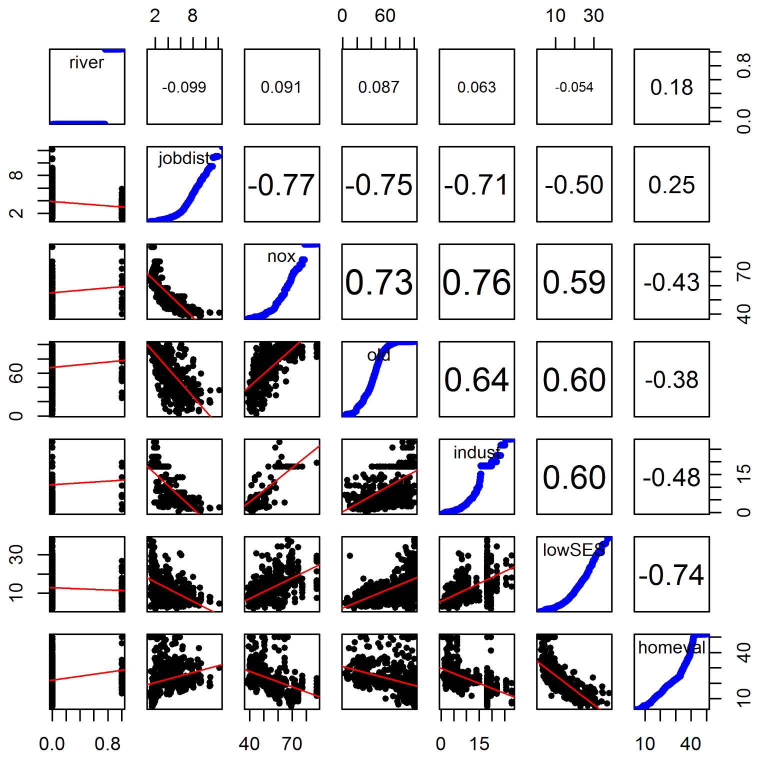
Noah was brave enough to try this one – and succeed. Halting between plots is controlled via setting `par(ask=TRUE)`. The version below adds the following knobs:

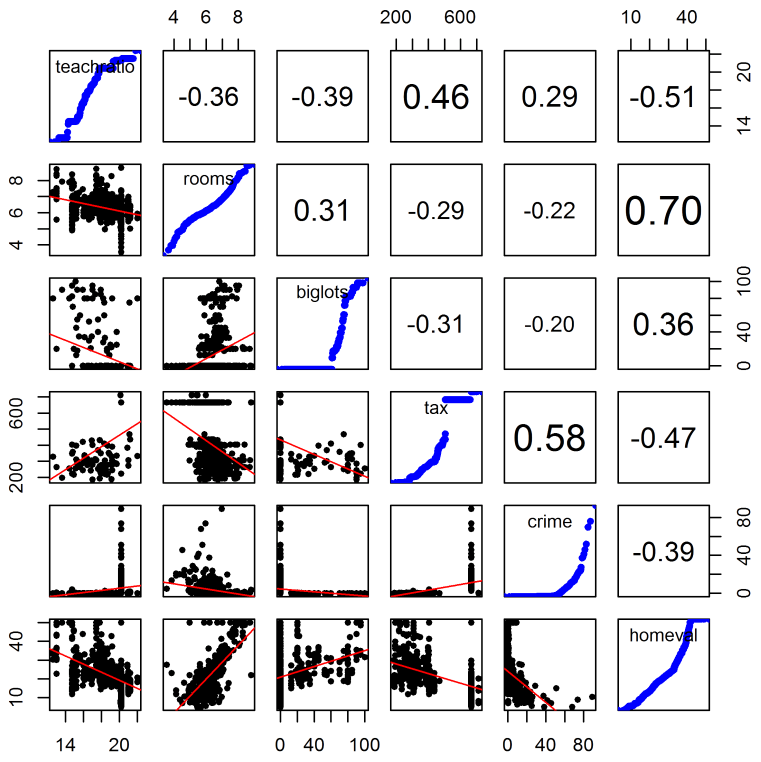
i. Controlling the number of variables per page (it is `pagesize+1`);
ii. Specifying the index of the y variable (`yind`, defaults to the last variable);
iii. Randomly scrambling the order, to see the pairwise relationship between non-adjacent variables.

```
pairsPlus<-function(x,diag.panel=panel.dens,diagCol=4,
      pagesize=dim(x)[2]-1,scramble=FALSE,yind=NA,pch=19,cex=0.6,...)
{
par(ask=FALSE)
nvars=dim(x)[2]
if(is.na(yind)) yind=nvars

### The number of pages; note the '%/%' operator for integer division
npages=1 + (nvars-2) %/% (pagesize)

varind=c((1:nvars)[-yind],yind)
if (scramble) varind=c(sample((1:nvars)[-yind]),yind)
### To show variables in scrambled order

for (a in 1:npages)
{
      currind=varind[c(((a-1)*pagesize+1):min(a*pagesize,nvars-1),nvars)]
      pairs(x[,currind],diag.panel=diag.panel,upper.panel=panel.cor,
             lower.panel=panel.scatter,pch=pch,cex=cex,...)
      if (a<npages) par(ask=TRUE)
}
par(ask=FALSE)
}
```

Here's an example combining all 3 added elements. (note: `lattice` has an analogous function called `splom`, as in “scatterplot-matrix”. I think our kludged version here gets more bang for the buck, but of course you can tweak the lattice version as well. The `psych` package (!) has a function more similar to our own version).

```
pairsPlus(boston,pagesize=6,diag.panel=panel.qq,scramble=T)
```


river
jobdist
nox
old
indus
lowSES
homeval
-0.099
0.091
0.087
0.063
-0.054
0.18
-0.77
-0.75
-0.71
-0.50
0.25
0.73
0.76
0.59
-0.43
0.64
0.60
-0.38
0.60
-0.48
-0.74


teachratio
rooms
biglots
tax
crime
homeval
-0.36
-0.39
0.46
0.29
-0.51
0.31
-0.29
-0.22
0.70
-0.31
-0.20
0.36
0.58
-0.47
-0.39

Another thing R users needs to do on a constant basis, is look for a tool that might be available, sometimes in more than one way. Relevant criteria might be user-friendliness and control options – but of course, first and foremost, statistical accuracy. When all else fails, you must code the tool on your own... Question 3 gave us a first taste of this experience.

3. The Variance Inflation Factor (VIF) – by hand, and evaluating available functions.

a. By hand. Here's a simple stripped-down version, that requires only a matrix as input. Notes:

i. When calculating VIFs, **leave the response (y) variable out of the matrix.** VIF calculates the collinearity in X, without regards to y.

ii. The VIFs are just another descriptive. There is no rule set in stone regarding how to act upon them. **In general, problems with better signal-to-noise ratio can tolerate higher VIFs (or pairwise correlations in X), and vice versa.**

iii. `lm` can take vectors and matrices directly as input – no need for full formulae with variable names. Each column in the matrix will be perceived as a term in the model, and an intercept will be automatically added. This is used to keep the home-brewed function short.

```
myvif=function(xmat)
{
xmat=as.matrix(xmat) ### just in case
nvar=dim(xmat)[2]

vout=rep(NA,nvar)
names(vout)=colnames(xmat)

for (a in 1:nvar)
{
	tmp=summary(lm(xmat[,a]~xmat[,-a]))
	vout[a]=1/(1-tmp$r.squared)
}
return(vout)
}

### output for the Boston dataset, untransformed:

     crime    biglots     indust      river        nox      rooms
  1.630791   2.272178   3.681575   1.059516   4.272901   1.859188
       old    jobdist        tax teachratio     lowSES
  3.068076   3.953227   3.351894   1.734159   2.864280
```

b. Evaluate and compare VIF functions from existing packages.

As we discovered together over the help threads, the most authoritative-yet-straightforward way to get the source code for any function is by downloading the package source code from CRAN, then unzipping it. If you are in a Unix-like system, you can connect to R's Subversion repository, and if the package is included in it (rather than be pre-bundled by the author) then you can get the latest version of the code (sometimes before the next official release).

http://cran.r-project.org/doc/manuals/R-admin.html#Using-Subversion-and-rsync

A shortcut discovered by some students, is to invoke the debugger via, e.g. `debug(HH:vif)`, then call the function. Its code will be spitted out on the console. In any case, here we will skip the evaluation of the raw code (*VIF is not that important*), and look at features and interface.

The two "packaged" VIF functions I referred to were `HH:vif, car::vif`. Both are associated with textbooks covering material similar to the first part of our class (James Wing also found a similar function in a package called `bstats,` related to a statistics course in a certain university).

From a user-interface POV, `HH:vif` accepts either a data frame, a formula or a fitted model – whereas `car::vif` only accepts the latter.

However, in terms of functionality, `car::vif` treats multi-d.f. covariates (such as multilevel factors, or spline objects) as single entities and applies an approximate "generalized VIF" to them – whereas `HH:vif` sees each d.f. as a separate entity, which is clearly wrong.

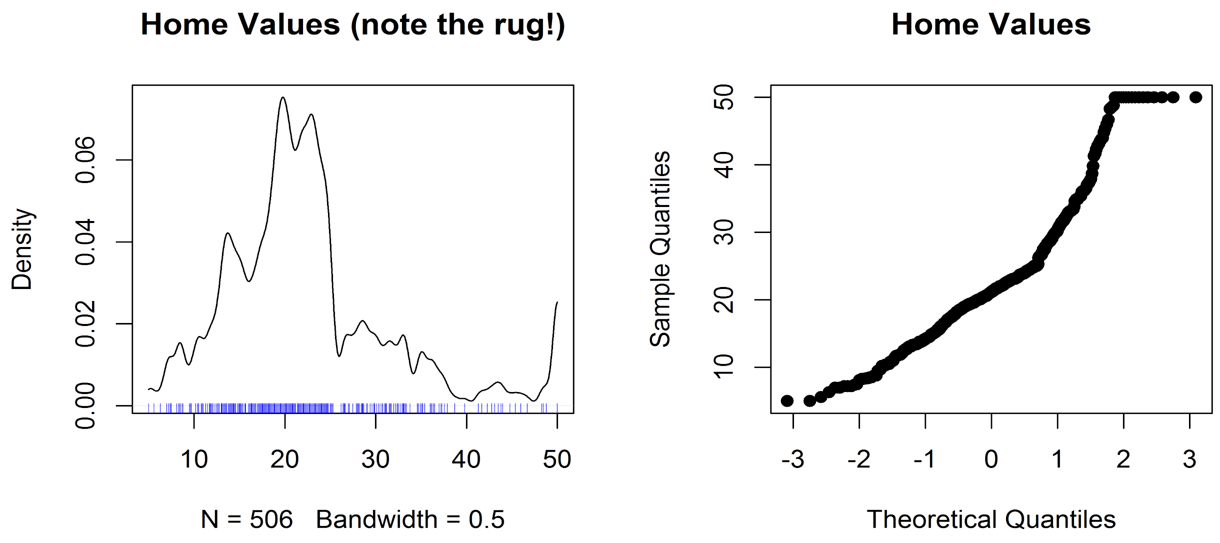
Overall, as a diagnostic VIFs are more important for **inference** than for **prediction.** In the former, the inflation of SEs increases the uncertainty of all effects and clouds the inferential picture. In the latter, it sometimes doesn't matter how much each individual covariate affects the prediction, as long as the end-result doesn't change much. In many high-dimensional cases VIFs may provide useful objective criteria for covariate pre-filtering. However, care must be taken – otherwise you might eliminate **all** covariates from a closely-correlated group and have none left in your model (the same warning applies to a pairwise-correlation filtering criterion).

4. Document and complete the work done in class on the 'boston' dataset.

The pairs/univariate/correlation plots were shown above in Question 2. We now look at potential data-quality and covariate-transformation issues.

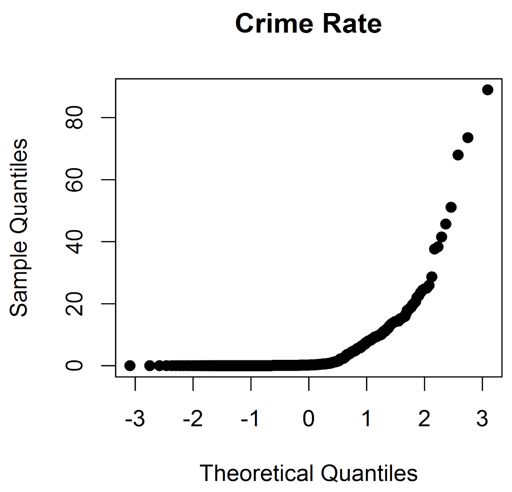
First, we discover to our horror that the response variable (`homeval`) is **censored** from above at $50k. This create a glut of >3% of the data (16 of 506) at the highest value, inducing a bias on both inference and prediction. In the olden days, when less data-manipulation tools were available, researchers often "massaged" the data in this and other ways to simplify problems and models – even though in this case there might have been a content reason for this censoring. This as well as other data features makes this dataset less attractive for sophisticated modern analysis (one thing one might do here, is to `cut(homeval)` into several pieces, thus turning the problem into a **classification** one.)

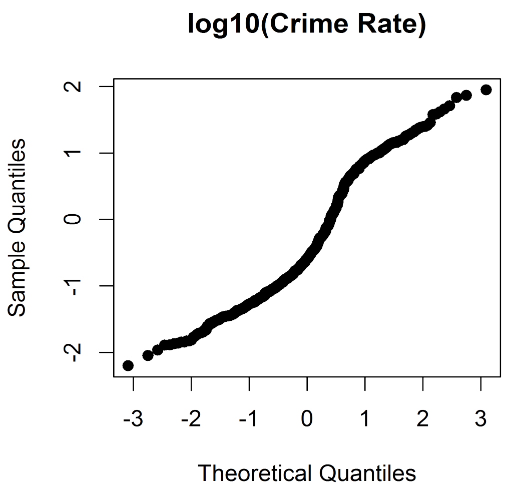
```
layout(t(1:2))
plot(density(boston$homeval,bw=.5,cut=0),main="Home Values (note the rug!)")
rug(boston$homeval,col=4)
qqnorm(boston$homeval,pch=19,main="Home Values")
```



Now a look at some of the problematic x variables. Easiest first: the crime rate is highly skewed. There are no zeros, and the variable lends it self to **a relatively painless log-transformation.** Reminder: whenever reasonably achievable, we prefer to have a covariate balanced and outlier-free, because then its estimated effect represents what the bulk of the dataset really "experiences".

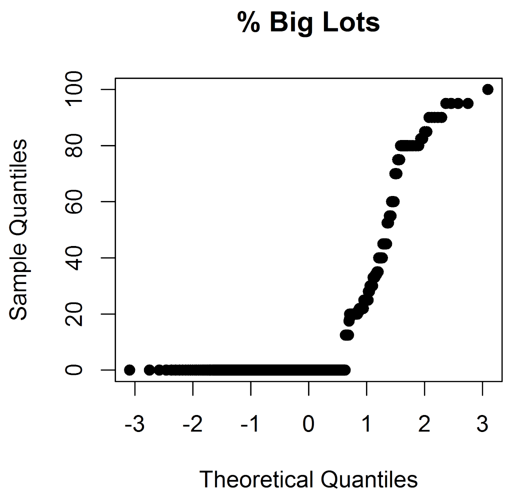
```
layout(t(1:2))
summary(boston$crime)

Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.01    0.08    0.26    3.61    3.68   89.00
qqnorm(boston$crime,pch=19,main="Crime Rate")
qqnorm(log10(boston$crime),pch=19,main="log10(Crime Rate)")
```

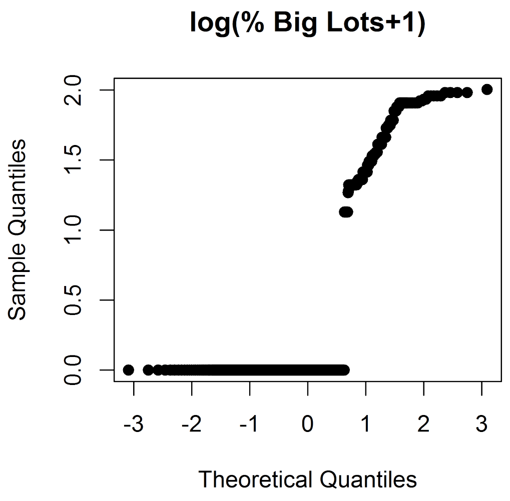




The next skewed covariate, `biglots`, is a bit of a trick question. At face value (e.g., looking only at histograms/density) it might look similar to `crime`. However, a closer look reveals a ton of zeros (nearly 3/4 of the data) – with the rest rather nicely spread out. As the plots show, transforming makes things worse rather than better. **So in this case, it's better not to transform or otherwise touch the covariate.** Remember Anscombe example 4? The model will just average the response over all the zeros, and combine it with the estimated effect over the remaining points. Given the distribution of the 134 nonzero points, we should get a reasonably balanced estimate.

```
layout(t(1:2))
summary(boston$biglots)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0     0.0     0.0    11.4    12.5   100.0
table(boston$biglots == 0)
## FALSE  TRUE
##   134   372
qqnorm(boston$biglots,pch=19,main="% Big Lots")
qqnorm(log10(boston$biglots+1),pch=19,main="log(% Big Lots+1)")
```



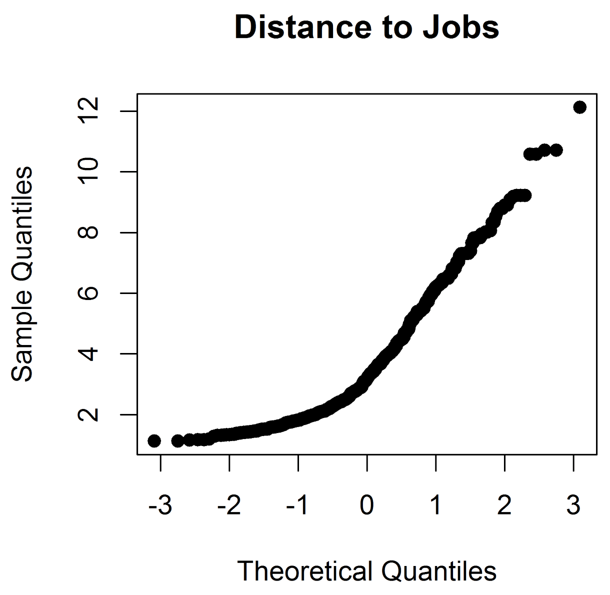


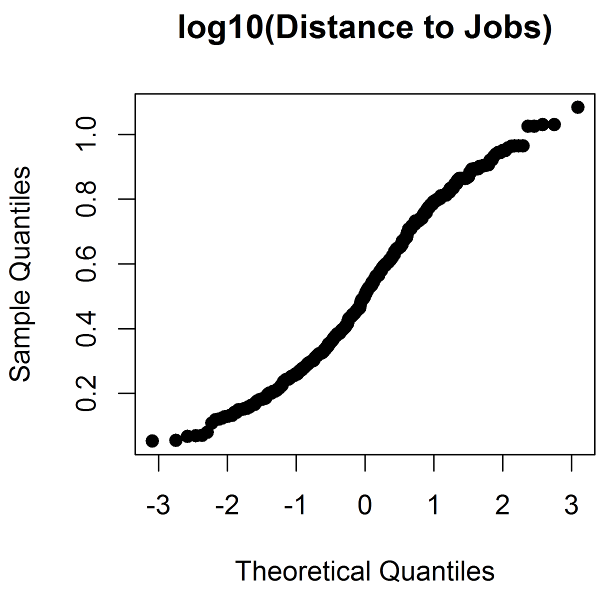
A related issue comes up with `river,` a two-level factor that is very unevenly distributed (93%:7%). Often, one would exclude covariates with so few nonzero entries. However, note (see pairs-plots above) that it is essentially uncorrelated with any of the 10 other covariates. This suggests that it provides information captured nowhere else in the dataset. We leave it be.

Next up is distance to job centers. As shown in class (Lecture 5), even though the skewness here is less extreme, this covariate is still better off log-transformed. A bit of content knowledge helps: very often in environmental or other spatial problems, the effect of a source decays either exponentially or in a power-law manner, so "distance-from" variables are usually (though not always) better off transformed. **Fundamentally, even if we're not sure of the form of decay, clearly the difference in effect between points, say, 1km and 2km from the source, and points 11km and 12km away from it, should not be the same (it is considered the same in the untransformed case) – and hence some transformation that emphasizes the near field over the far field is in order.**

A log-transformation of covariates usually takes care of things. If one looks for a power-law transformation, then the power needs to be chosen (that's another advantage of log-transforms: they are all alike, changing the base just multiplies the variable by a constant). A family known as **Box-Cox** power-law transformations, estimated from the data, is considered fairly standard. But here I just log-transformed.

```
layout(t(1:2))
summary(boston$jobdist)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.130   2.100   3.207   3.795   5.188  12.130
qqnorm(boston$jobdist,pch=19,main="Distance to Jobs")
qqnorm(log10(boston$jobdist),pch=19,main="log10(Distance to Jobs)")
```

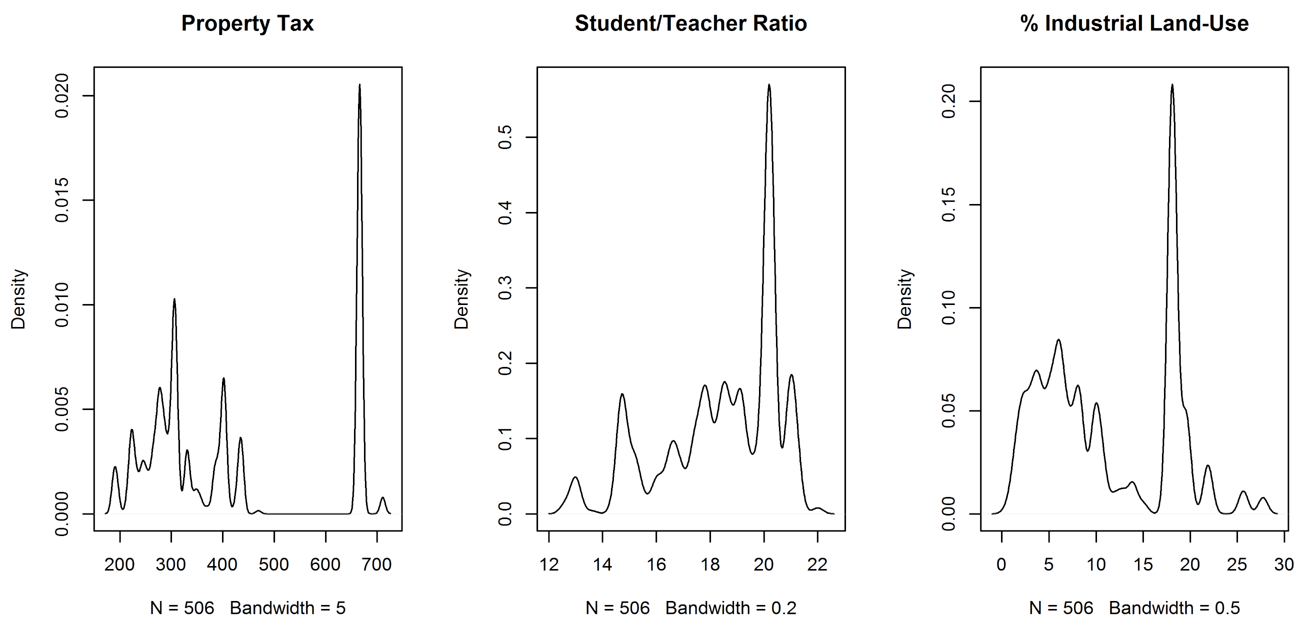




Now, to those variables with repeated identical values. This being an "ancient, massaged" dataset, it is hard to track down whether they are due to data error/manipulation, or simply because 132 of the 506

Census tracts really do fall into a homogeneous georgraphical/administrative region, about which we have no information on a finer scale for any of these 3 covariates (`tax, indust, teachratio`).

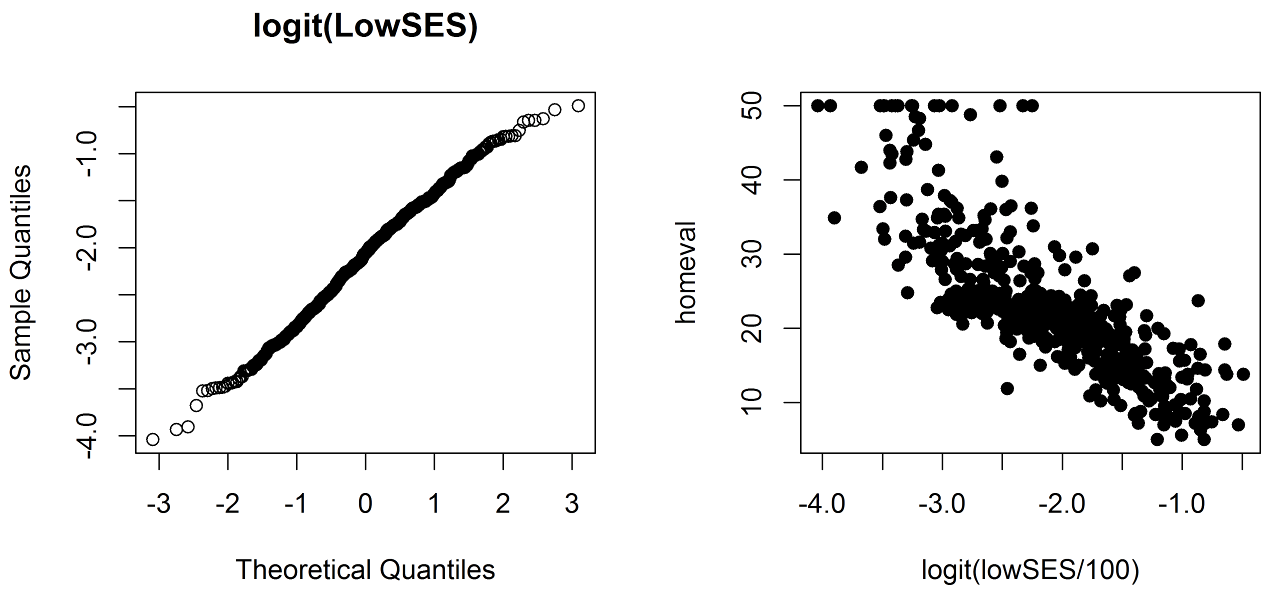
```
layout(t(1:3))
rev(table(table(boston$tax)))   ### The grouping of data by counts at individual values
132  40  30  15  14  12  11  10   9   8   7   6   5   4   3   2   1 
  1   1   1   1   1   2   2   2   3   4   6   1   6   5   5  12  13 

# So we have one instance of 132 identical points - vs., e.g., 13 data points which are unique

rev(table(table(boston$indust)))
 132  30  22  18  15  12  11  10   9   8   7   6   5   4   3   2   1 
  1   1   1   1   1   2   2   1   2   4   4   4   6   9   7  14  16 
rev(table(table(boston$teachratio)))
140  34  27  23  19  18  17  16  15  13  12  11   9   8   7   6   5   4   3   2   1 
  1   1   1   1   1   1   2   2   1   1   1   2   1   2   1   1   5   7   4   4   6 

plot(density(boston$tax,bw=5),pch=19,main="Property Tax")
plot(density(boston$teachratio,bw=0.2),pch=19,main="Student/Teacher Ratio")
plot(density(boston$indust,bw=0.5),pch=19,main="% Industrial Land-Use")
```



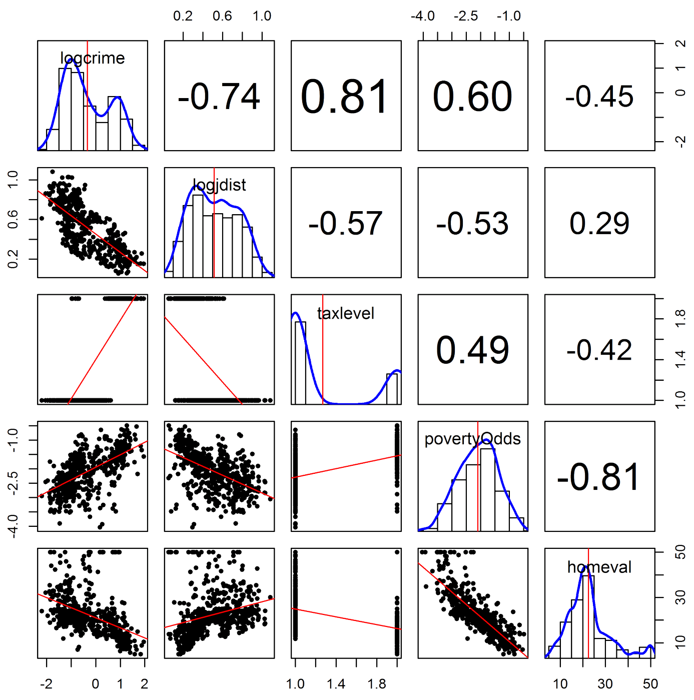
At least in my books, these covariates will face a stronger burden of proof to show their worth. The tax rate, specifically, I suggest to **dichotomize** (say at 500) **into two levels.** The contrast with the `biglots` variable is instructive:

i. In `tax`, the 132 points with the diabolical 666 rate act more as a de-balancing outlier than the zeros in `biglots`.
ii. Content-wise, `biglots` measures a less-controlled, quasi-natural process (human settlement patterns etc.), whilst the tax rate is more arbitrary and controlled – so there is more meaning in continuous values of the former vs. the latter (that being said, had `tax` been more well-behaved I'd leave it alone).

Finally, that high-impact, percentage-based covariate, `lowSES`. There are no zeros or 100's. A logit-transform makes this variable more symmetric, and the trend with `homeval` more linear and better correlated (compare with the Q2 pairs patterns). The intepretation of the transformed covariate is “the log-odds of a resident in this tract having a low SES”.



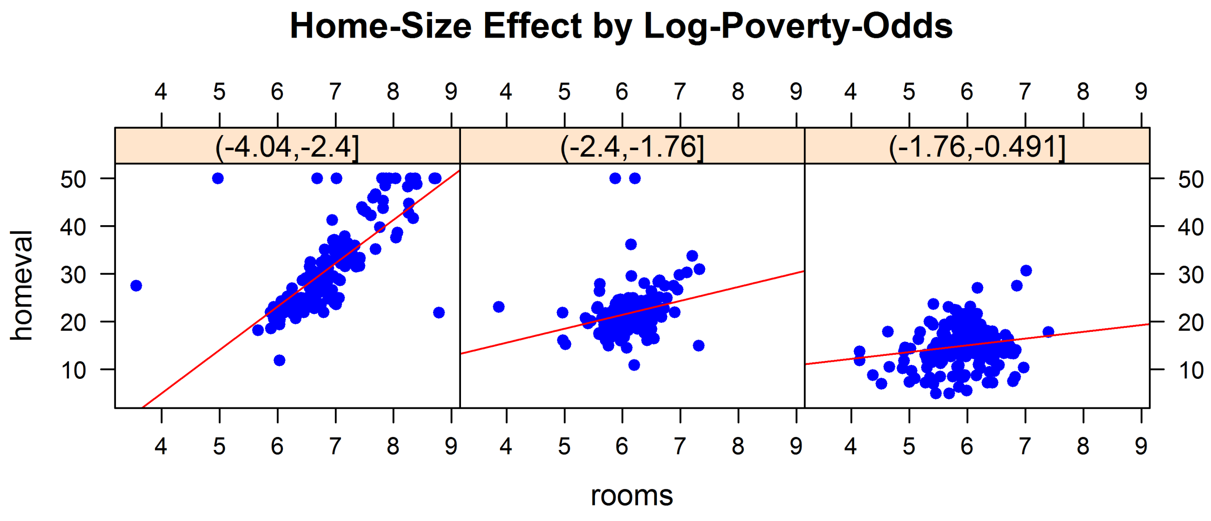
So all in all, I chose to transform/discretize 4 of the 11 covariates. Here's a scatterplot-matrix of the new covariates:

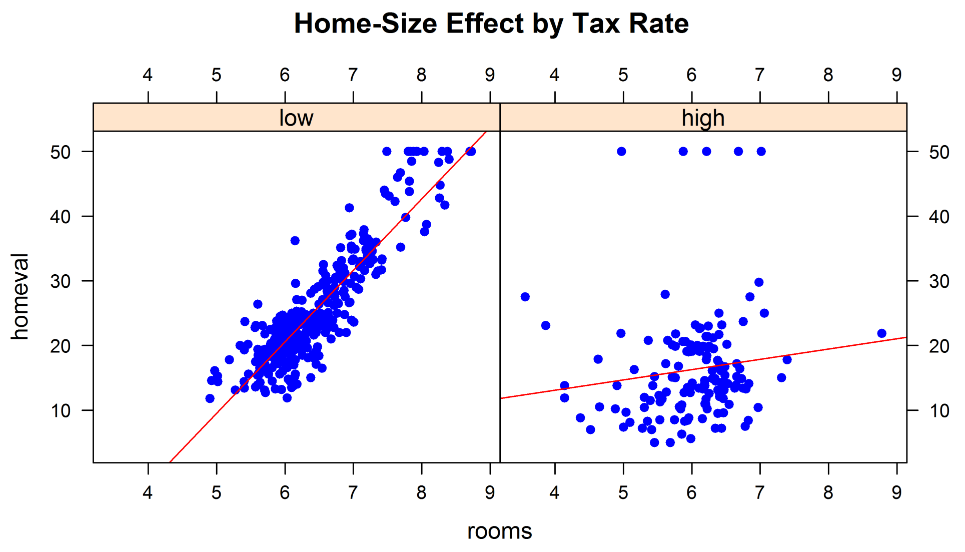


*5. Use the lattice command **`xyplot`**, to examine potential interactions between covariates*

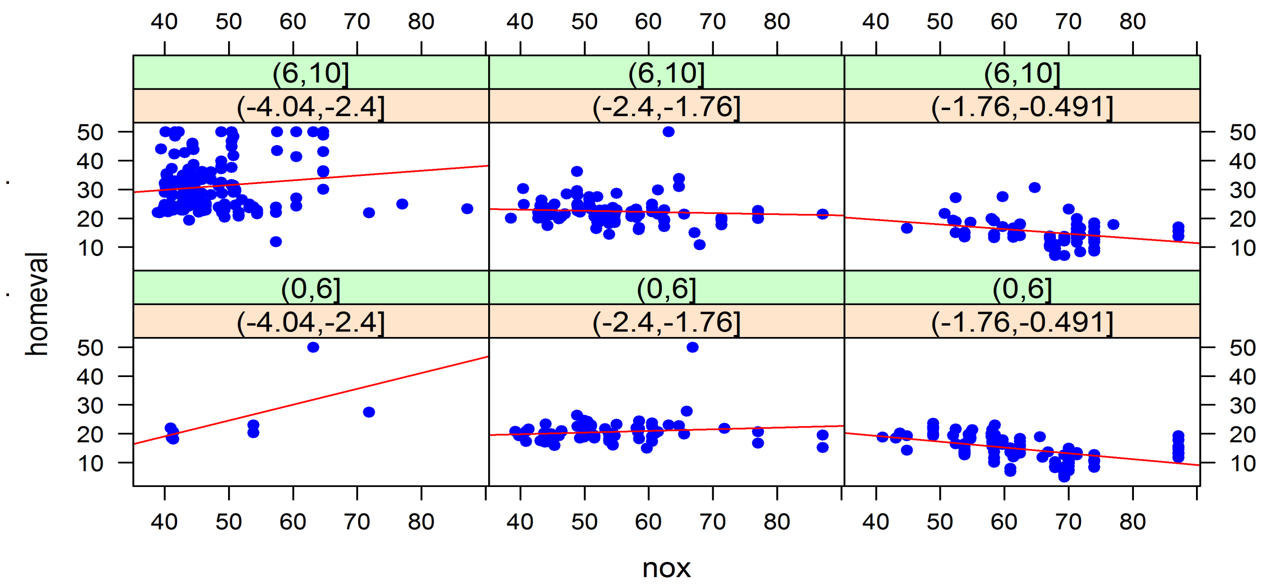
We use this to evaluate the effect of secondary covariates, after the two "big" ones (`rooms`, `povertyOdds`) are accounted for. Here's a sample, with the code shown for two. In general, it seems that most interactions with these covariates, especially `povertyOdds`, are worth considering.

```
lmpanel=function(x,y,...) {
  panel.xyplot(x,y,...)
  panel.abline(lm(y~x),col=2)
  }
xyplot(homeval~rooms |
cut(povertyOdds,quantile(povertyOdds,p=0:3/3)),data=boston,pch=19,panel=lmpanel,main="Home-
Size Effect by Log-Poverty-Odds")

xyplot(homeval~nox | cut(povertyOdds,quantile(povertyOdds,p=0:3/3))
+cut(rooms,c(0,6,10)),data=boston,pch=19,panel=lmpanel,main="Nox Effect by Main 2
Covariates")
```

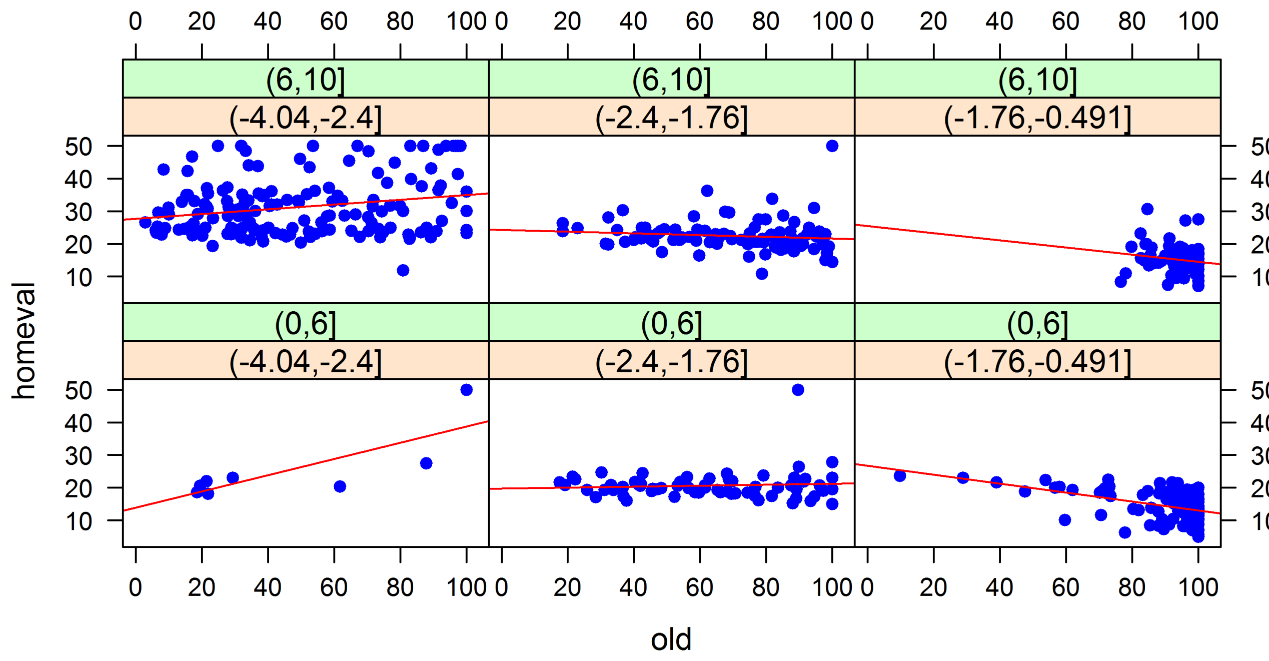




Nox Effect by Main 2 Covariates



Old-House Effect by Main 2 Covariates


(6,10]
(-4.04,-2.4]
(-2.4,-1.76]
(-1.76,-0.491]
(0,6]
homeval
old