Chapter 9

High-dimensional data

9.1 PCA

Optional reading: Ch. 16 of Shalizi's Advanced Data Analysis (http://www.stat.cmu.edu/~cshalizi/ADAfaEPoV/) is mathy but correct and useful. I don't know of a good treatment that doesn't have a bunch of linear algebra yet isn't a horrendous over-simplification; if you've seen one, let me know.

In **principal components analysis (PCA)**, we project a high-dimensional set of data on to the k-dimensional (hyper-) plane that retains the most of the original data's variance. This is done via a singular value decomposition. Choosing k can be a hard problem, but not in EDA: we use $k = 2$ because two is the best number of dimensions for graphs. So in other words, we're displaying a data set of lots of variables on a plane.

My preferred function for displaying PCA is `ggbiplot()` in the package of the same name. This isn't on CRAN, so you'll need to install it from Github:

```
library(devtools)
install_github("vqv/ggbiplot")
```

9.1.1 Wine

The `ggbiplot` package contains a data set on 13 chemical measurements on the composition of a bunch of Piemonte red wines. Details are at:

https://archive.ics.uci.edu/ml/datasets/Wine

```
library(ggbiplot)
data(wine)
nrow(wine)
```

```
## [1] 178
```

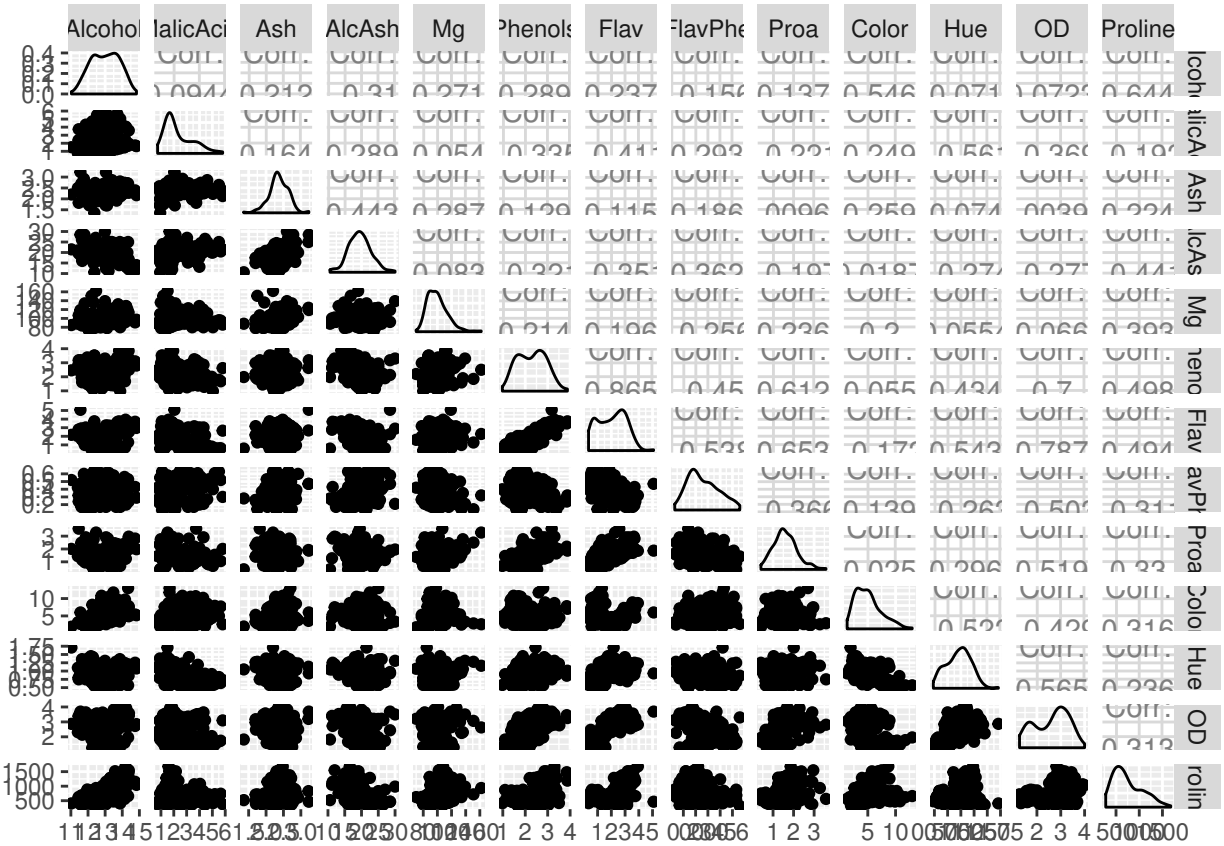
```
summary(wine)
```

```
##     Alcohol        MalicAcid          Ash            AlcAsh
##  Min.   :11.03   Min.   :0.740   Min.   :1.360   Min.   :10.60
##  1st Qu.:12.36   1st Qu.:1.603   1st Qu.:2.210   1st Qu.:17.20
##  Median :13.05   Median :1.865   Median :2.360   Median :19.50
##  Mean   :13.00   Mean   :2.336   Mean   :2.367   Mean   :19.49
##  3rd Qu.:13.68   3rd Qu.:3.083   3rd Qu.:2.558   3rd Qu.:21.50
##  Max.   :14.83   Max.   :5.800   Max.   :3.230   Max.   :30.00
```

```
##        Mg              Phenols          Flav        NonFlavPhenols
##  Min.   : 70.00   Min.   :0.980   Min.   :0.340   Min.   :0.1300
##  1st Qu.: 88.00   1st Qu.:1.742   1st Qu.:1.205   1st Qu.:0.2700
##  Median : 98.00   Median :2.355   Median :2.135   Median :0.3400
##  Mean   : 99.74   Mean   :2.295   Mean   :2.029   Mean   :0.3619
##  3rd Qu.:107.00   3rd Qu.:2.800   3rd Qu.:2.875   3rd Qu.:0.4375
##  Max.   :162.00   Max.   :3.880   Max.   :5.080   Max.   :0.6600
##       Proa            Color             Hue              OD
##  Min.   :0.410   Min.   : 1.280   Min.   :0.4800   Min.   :1.270
##  1st Qu.:1.250   1st Qu.: 3.220   1st Qu.:0.7825   1st Qu.:1.938
##  Median :1.555   Median : 4.690   Median :0.9650   Median :2.780
##  Mean   :1.591   Mean   : 5.058   Mean   :0.9574   Mean   :2.612
##  3rd Qu.:1.950   3rd Qu.: 6.200   3rd Qu.:1.1200   3rd Qu.:3.170
##  Max.   :3.580   Max.   :13.000   Max.   :1.7100   Max.   :4.000
##     Proline
##  Min.   : 278.0
##  1st Qu.: 500.5
##  Median : 673.5
##  Mean   : 746.9
##  3rd Qu.: 985.0
##  Max.   :1680.0
```

We'd expect most of the variables to be associated with most of the other variables. 13 is a pretty large number of variable. We can still draw a pairs plot:

```
library(GGally)
ggpairs(wine)
```



but it's pretty crowded. If I knew about the chemistry of wine I might try to choose variables I thought were

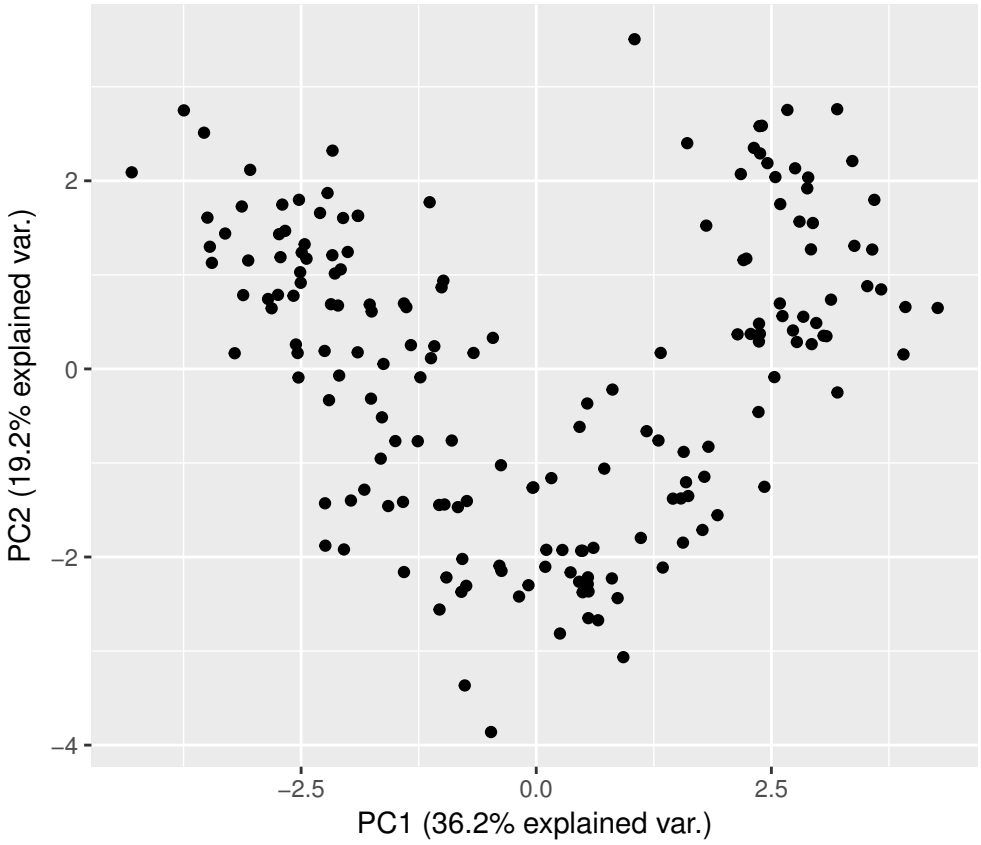
important, but I don't, so let's do dimension reduction via PCA.

The `prcomp()` function runs PCA. The main decision to make is whether to center and scale the data: the function's default is to center but not scale. However, rescaling is almost the right thing to do when the variables are not all measuring the same thing.

```
wine.pca = prcomp(wine, scale. = TRUE)
```

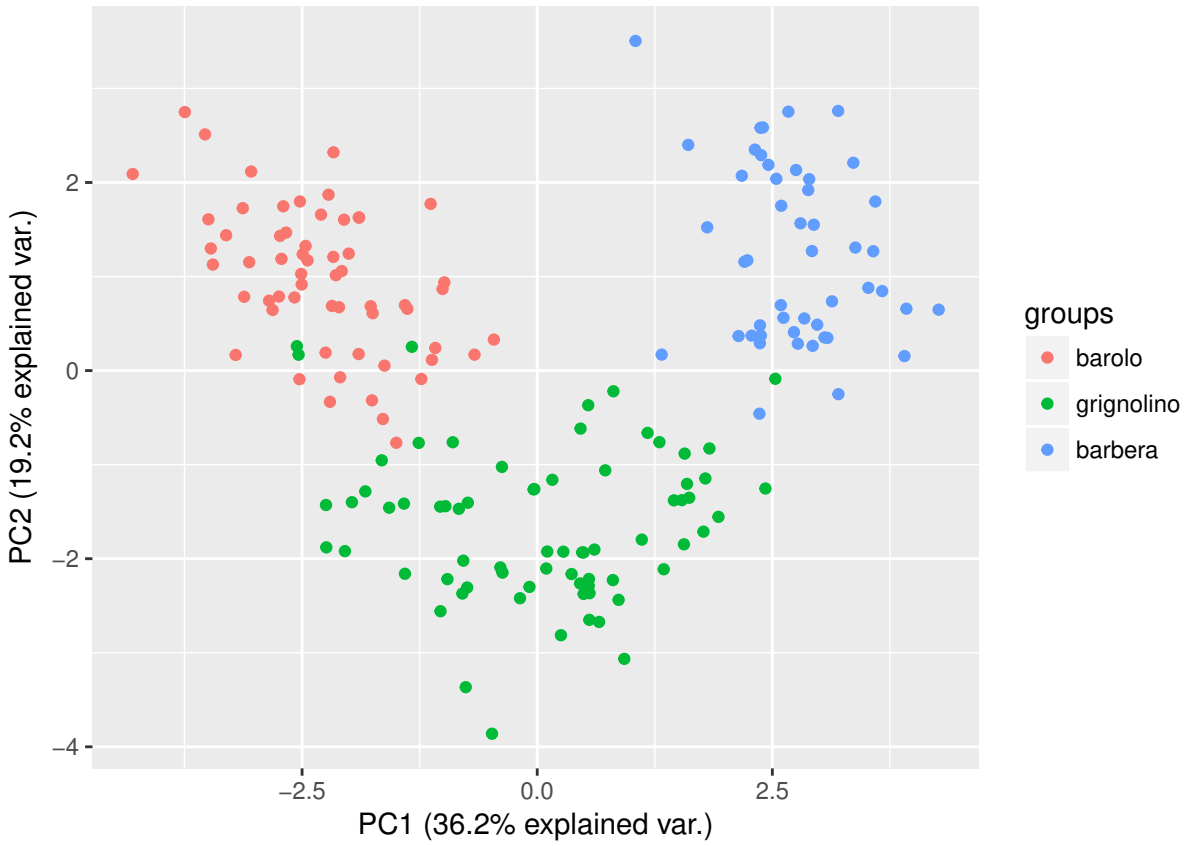
(Note the dot after scale.) Now use `ggbiplot()` to visualize the first two principal components. We first leave out the "biplot" part and just plot the points.

```
library(ggbiplot)
ggbiplot(wine.pca, obs.scale = 1, var.axes = FALSE)
```



`obs.scale = 1` prevent the rescaling of the principal components and `var.axes = FALSE` suppresses the biplot. We see that on this two-dimensional representation, there's a U-shape. For this to be useful, we need to relate this back to the wines. There's a data variable called `wine.class` that gives the varietal of the wines. We can group according to this variable:

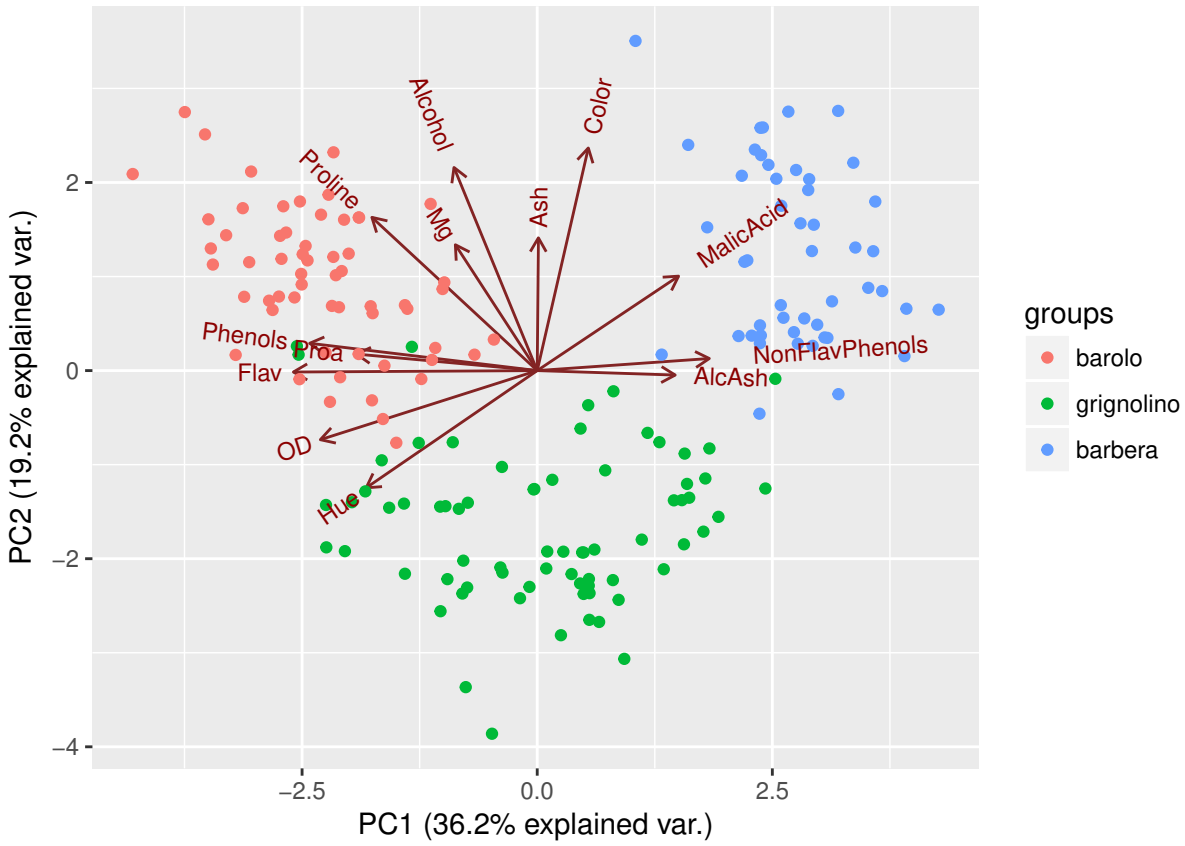
```
ggbiplot(wine.pca, obs.scale = 1, var.axes = FALSE, groups = wine.class)
```



We can now see an ordering: Barolo, Grignolino, Barbera. Barolo is a full-bodied red wine and Barbera is medium-bodied. I don't know anything about Grignolino but it looks like it's not simply in between the other two. The separation is fairly clear, with only a small amount of overlap between Barolo and Grignolino and between Grignolino and Barbera. (Note that if our main goal was classification, *linear discriminant analysis (LDA)* might be a better approach than PCA.)

Now add the **biplot**:

```
ggbiplot(wine.pca, obs.scale = 1, groups = wine.class)
```

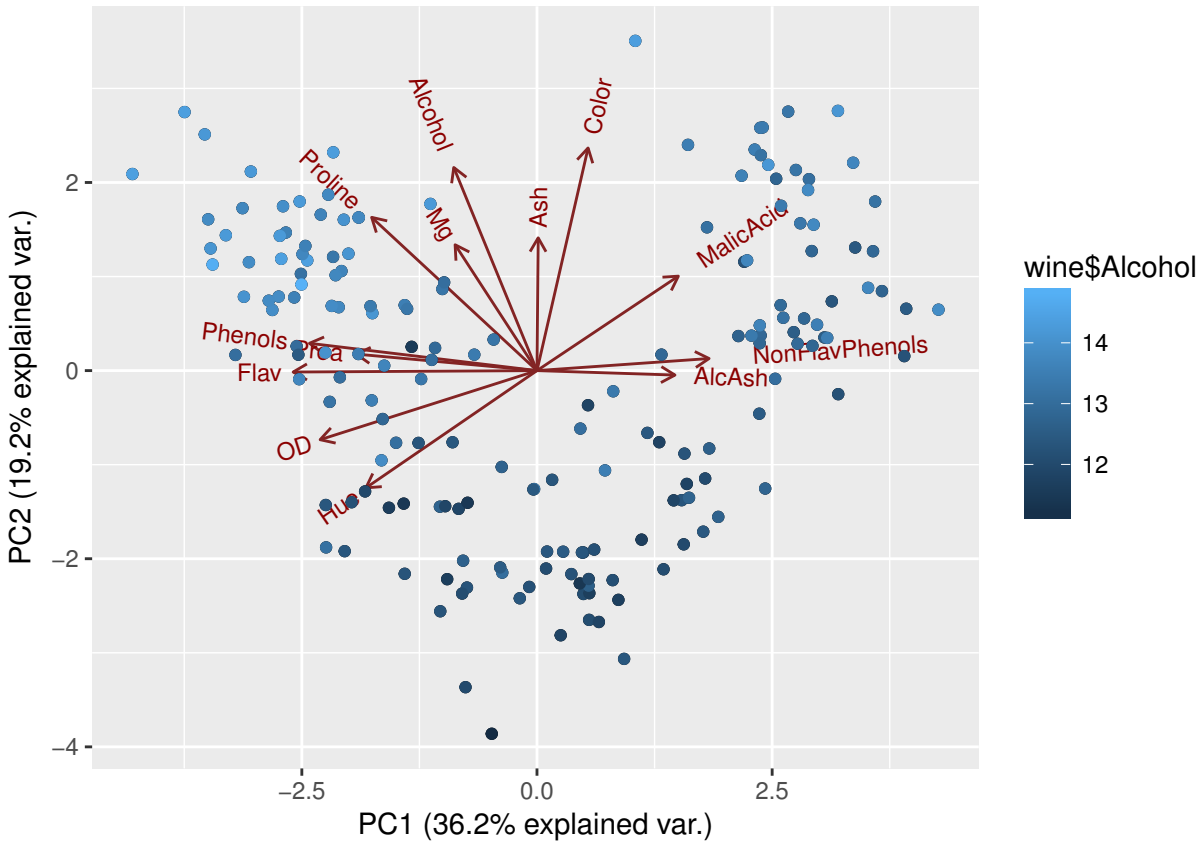


The direction of the arrows tells you in which direction values of that variable are generally increasing. So `AlcAsh` (the alkalinity of the ash in the wine) increases as you go right. `Flav` (flavanoids) increase as you go left, `Ash` increases as you go up, and so on.

The length of the arrows gives the correlation of the variables with the two principal components. So `Flav` has a relatively strong relationship with the two principal components, compared to, say, `Ash`.

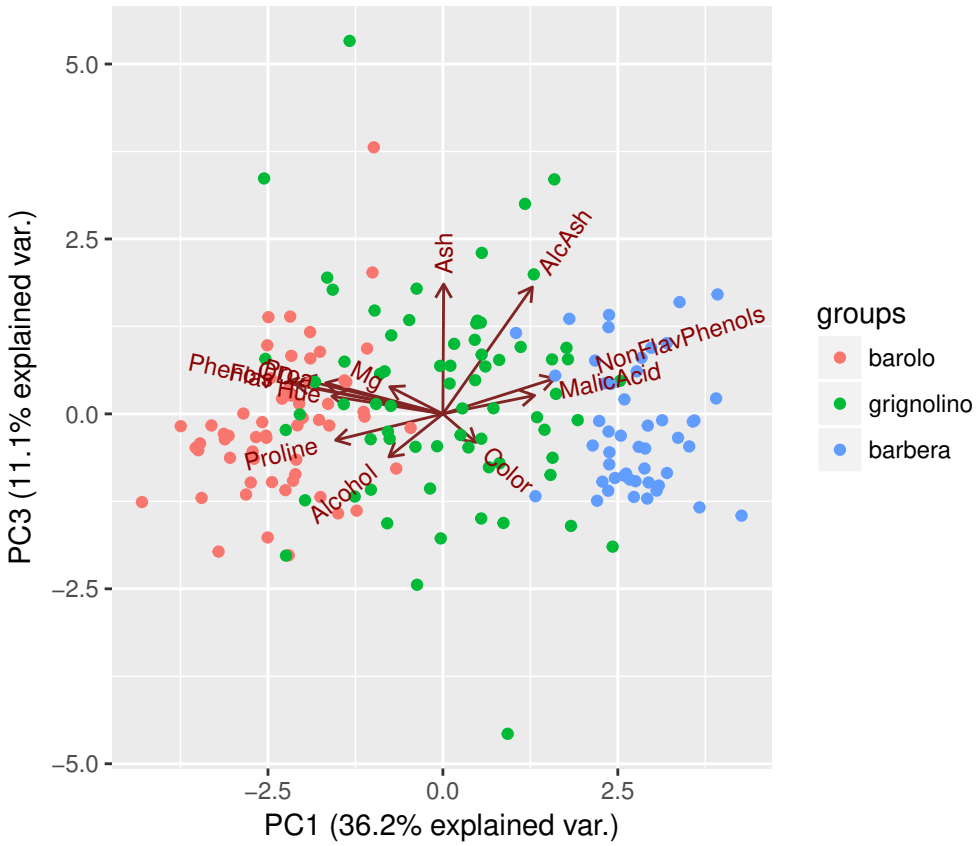
You can also use `geom_point()` and so on to control the appearance of the graph:

```
ggbiplot(wine.pca, obs.scale = 1) + geom_point(aes(color = wine$Alcohol))
```



Finally, you could also plot principal components other than the first two:

```
ggbiplot(wine.pca, obs.scale = 1, groups = wine.class, choices = c(1, 3))
```



We see that using the third principal component doesn't give much help with separation (over and above what the first component gives.)

9.1.2 The 88th Congress

The 88th Congress, which met in 1963–1964, passed many important pieces of legislation, including the Civil Rights Act, the Economic Opportunity Act, the Food Stamp Act, the Clean Air Act, and the Tonkin Gulf Resolution. It's especially interesting because many of these key votes were not along party lines. Data on all of the roll call votes held is on the site

http://www.voteview.com/house88.htm

We can dowload the Stata file, import it, and reformat it as a tibble:

```
library(rio)
house88 = import("hou88kh.dta")
# If you like tibbles: library(tibble) house88 = as_tibble(house88) house88
```

There were 232 roll call votes, coded 0 to 9 (see the webpage for details.) We can recode as follows:

- Let 1 mean some kind of Yes;
- Let −1 mean some kind of No;
- Everything else is a 0.

(This could be a bit misleading, as the zeroes don't distinguish between people who abstained from a vote and people who couldn't vote because they weren't in Congress or were dead, but we'll keep it simple.) Let's create a new matrix called `votes6364` to contain this new encoding.

```
votes6364 = matrix(0, nrow = 445, ncol = 232)
votes6364[house88[, 10:232] == 1] = 1
```

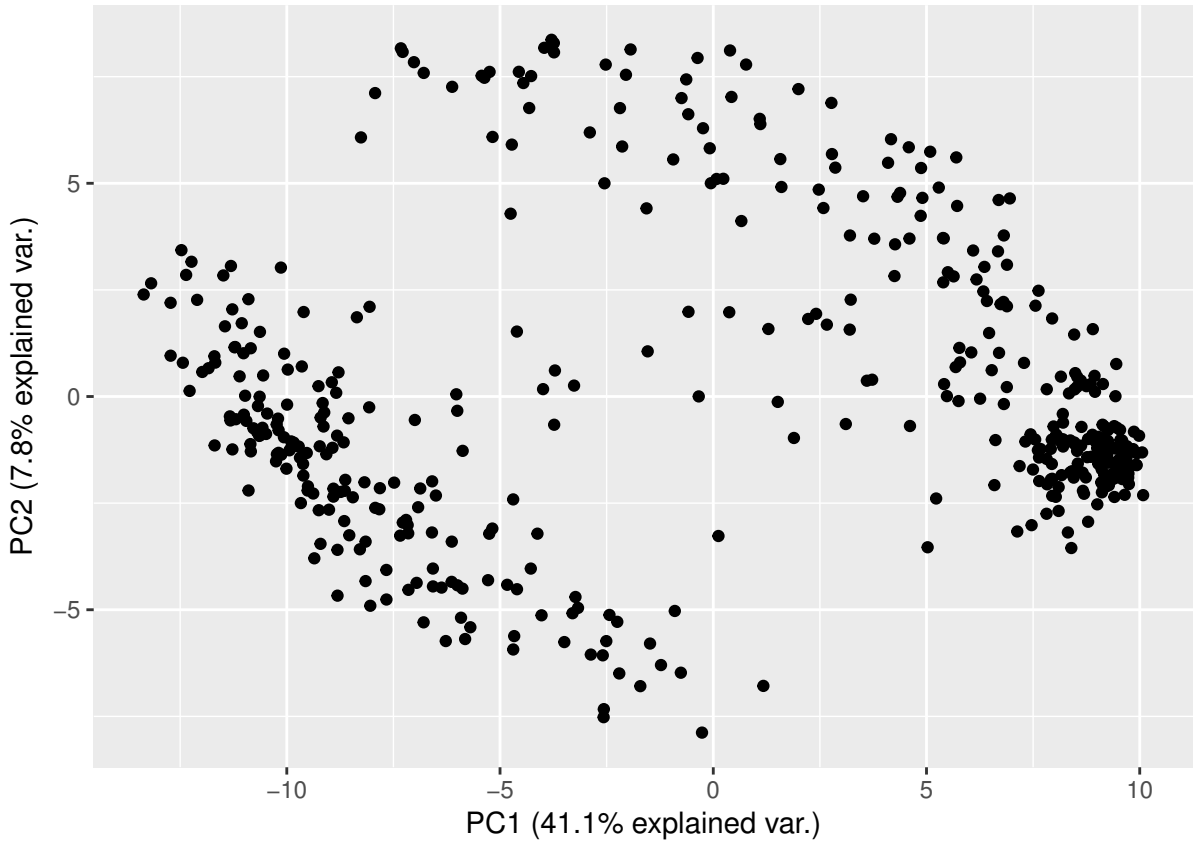
```
votes6364[house88[, 10:232] == 2] = 1
votes6364[house88[, 10:232] == 3] = 1
votes6364[house88[, 10:232] == 4] = -1
votes6364[house88[, 10:232] == 5] = -1
votes6364[house88[, 10:232] == 6] = -1
```

Everything's now on the same scale, so rescaling isn't necessary. Run PCA:

```
votes.pca = prcomp(votes6364)
```

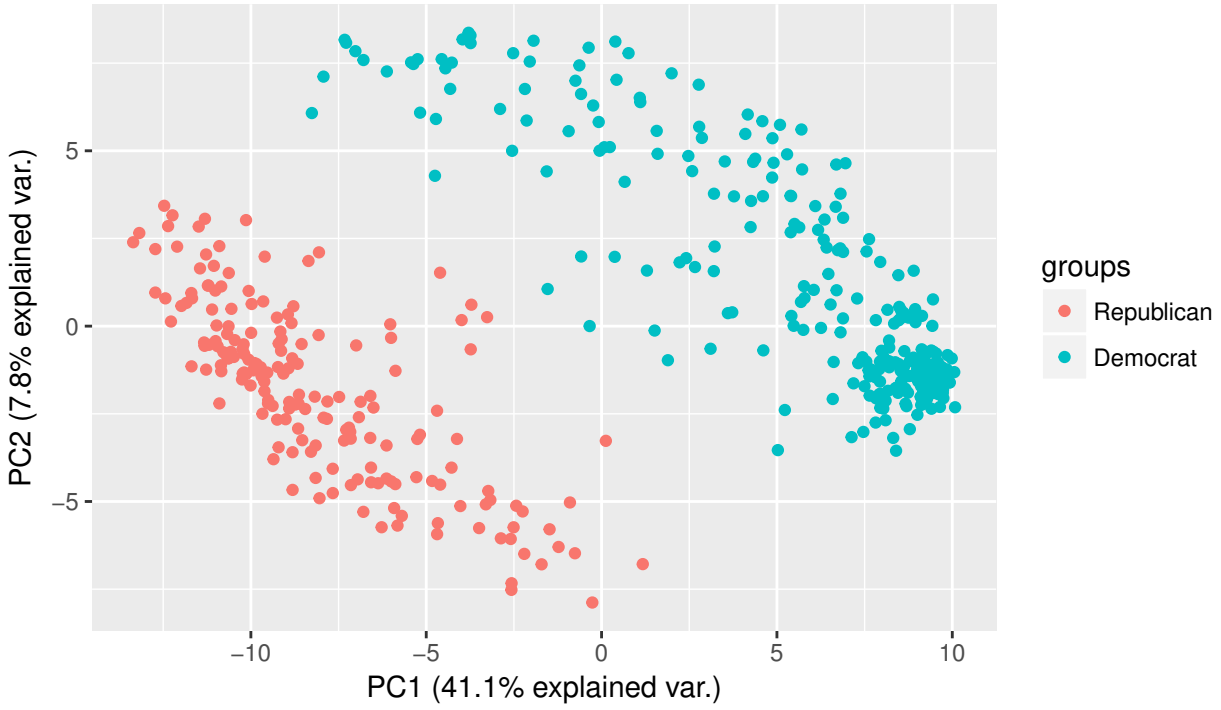
Now display the results, suppressing the biplot since we don't want 232 arrows cluttering our graph.

```
ggbiplot(votes.pca, obs.scale = 1, var.axes = FALSE)
```



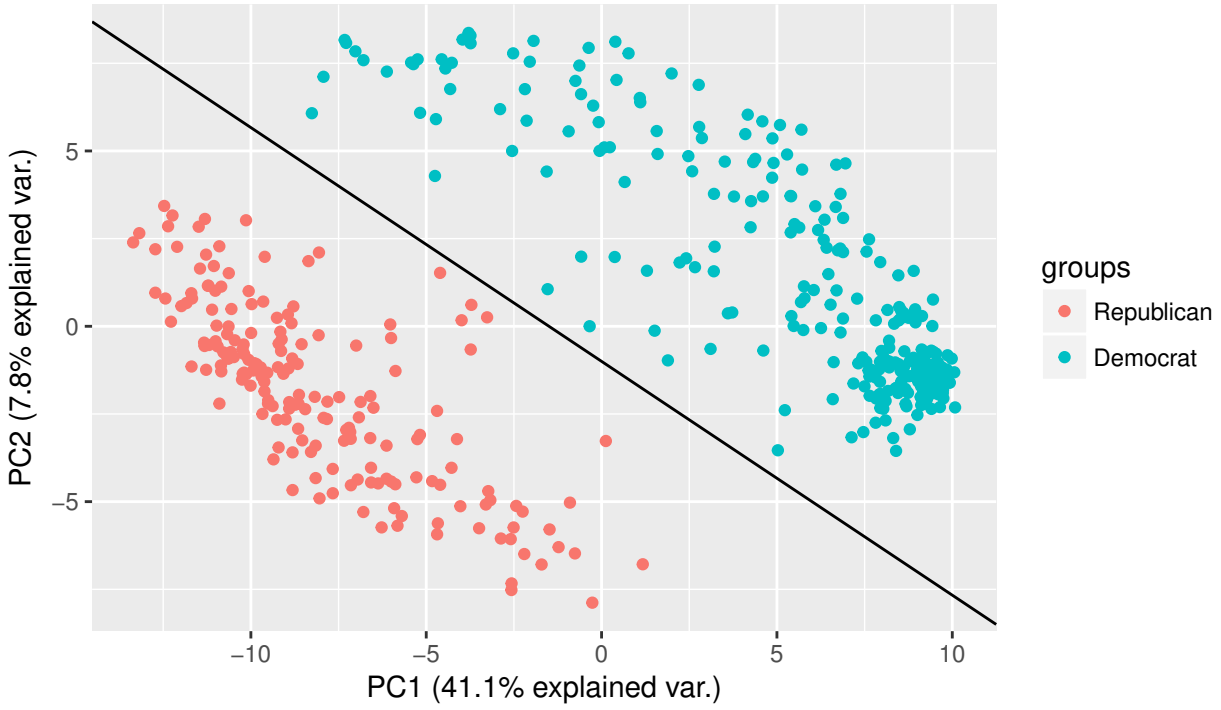
Recode the `party` variable to use as group labels:

```
party = house88$party
party[party == 100] = "Democrat"
party[party == 200] = "Republican"
party = factor(party, levels = c("Republican", "Democrat"))
ggbiplot(votes.pca, obs.scale = 1, var.axes = FALSE, group = party)
```



We see pretty clear separation between Republicans and Democrats. Eyeball a line:

```
ggbiplot(votes.pca, obs.scale = 1, var.axes = FALSE, group = party) + geom_abline(intercept = -1,
    slope = -2/3)
```



We can add names:

```
ggbiplot(votes.pca, obs.scale = 1, var.axes = FALSE, group = party, labels = house88$name,
    labels.size = 2) + geom_abline(intercept = -1, slope = -2/3)
```

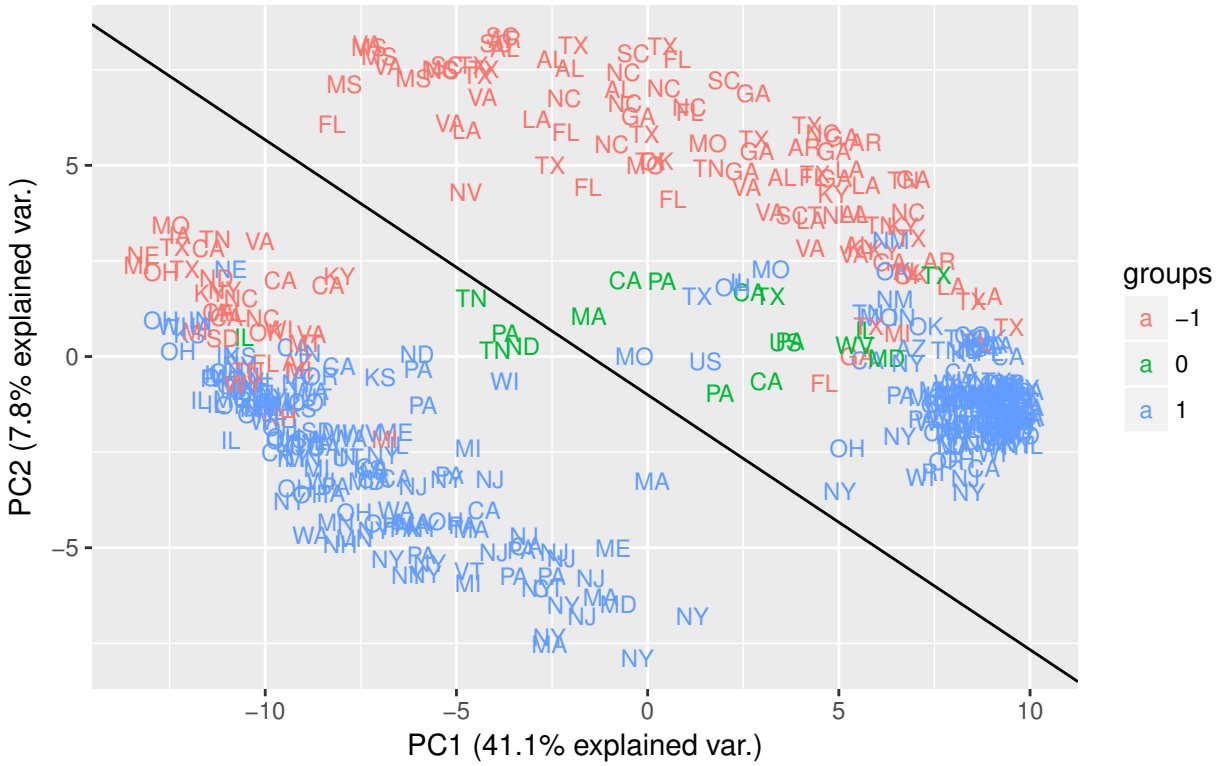
but these don't mean much to me and I'm supposed to know about these things. Instead, we find the state codes and use those as labels:

```
statecodes = read.table("statecodes.txt")
state = rep(NA, length(house88$state))
# Too lazy to work out how to do this efficiently so use a for loop
for (J in statecodes$V1) {
    code = statecodes$V2[statecodes$V1 == J]
    state[house88$state == J] = as.character(code)
}
ggbiplot(votes.pca, obs.scale = 1, var.axes = FALSE, group = party, labels = state,
    labels.size = 3) + geom_abline(intercept = -1, slope = -2/3)
```

In general, northeastern states are toward the bottom-right, while southern and Midwestern states tend to be toward the top left.

We can now look at the results of the key votes we noted at the beginning of the section. Start with the Civil Rights Act. According to ftp://k7moa.com/dtl/88.dtl, this was vote number 128, so we make our group label column 128 of `votes6364`.

```
ggbiplot(votes.pca, obs.scale = 1, var.axes = FALSE, group = factor(votes6364[,
    128]), labels = state, labels.size = 3) + geom_abline(intercept = -1, slope = -2/3)
```



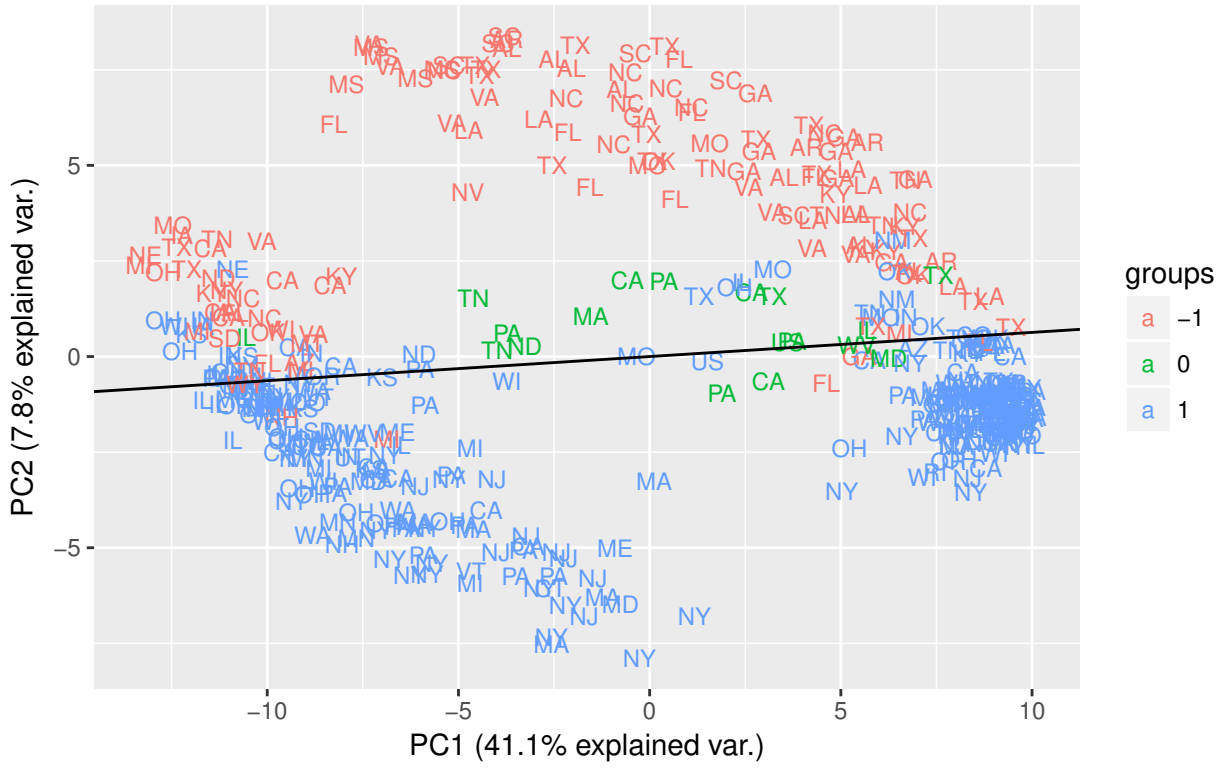
The vote here is almost entirely explained by the second component. The direct of the biplot arrow for this vote is given by the first two numbers of row 128 of the `rotation` matrix produced by the PCA:

```
votes.pca$rotation[128, 1:2]
```

```
##          PC1          PC2
##  0.01627373 -0.21041825
```

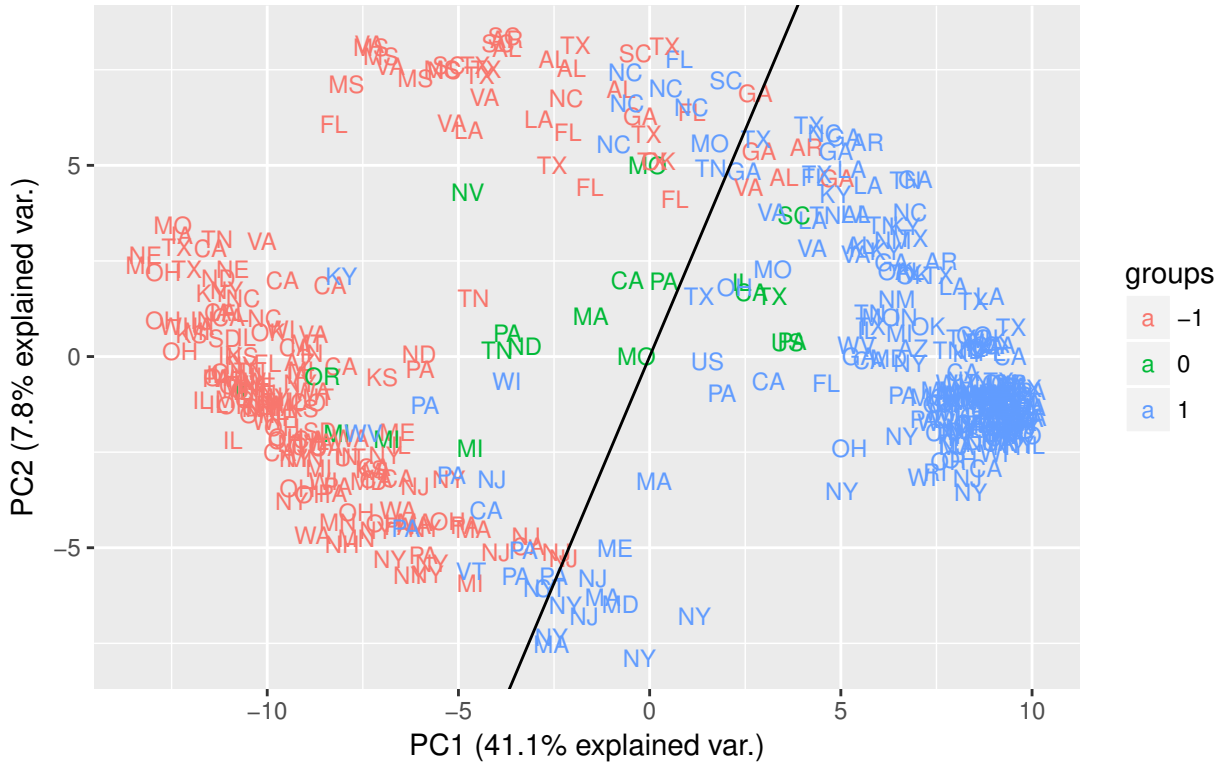
But it might be helpful to add a line *perpendicular* to this to separate ayes from nays:

```
ggbiplot(votes.pca, obs.scale = 1, var.axes = FALSE, group = factor(votes6364[,
    128]), labels = state, labels.size = 3) + geom_abline(slope = -votes.pca$rotation[1,
    128]/votes.pca$rotation[2, 128])
```



The Economic Opportunity Act (vote 201):

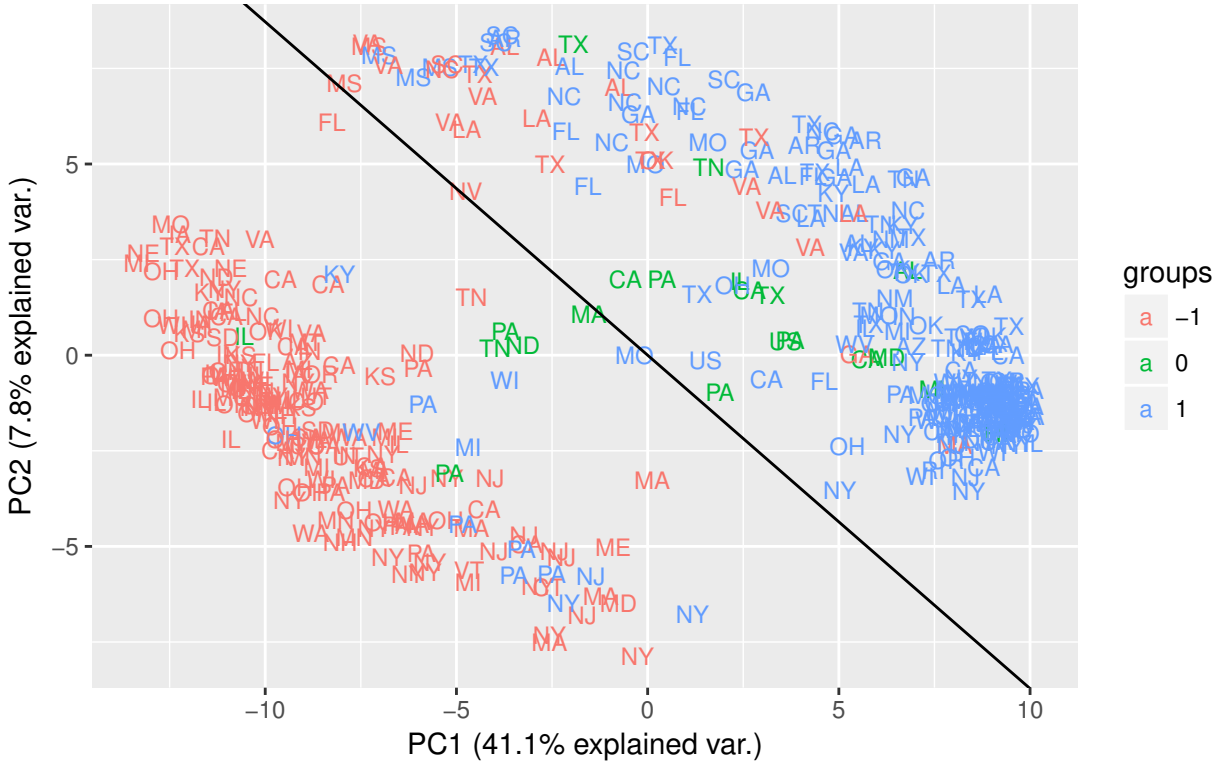
```
ggbiplot(votes.pca, obs.scale = 1, var.axes = FALSE, group = factor(votes6364[,
    201]), labels = state, labels.size = 3) + geom_abline(slope = -votes.pca$rotation[1,
    201]/votes.pca$rotation[2, 201])
```



This vote is better explained by the first component.

The Food Stamp Act (vote 149):

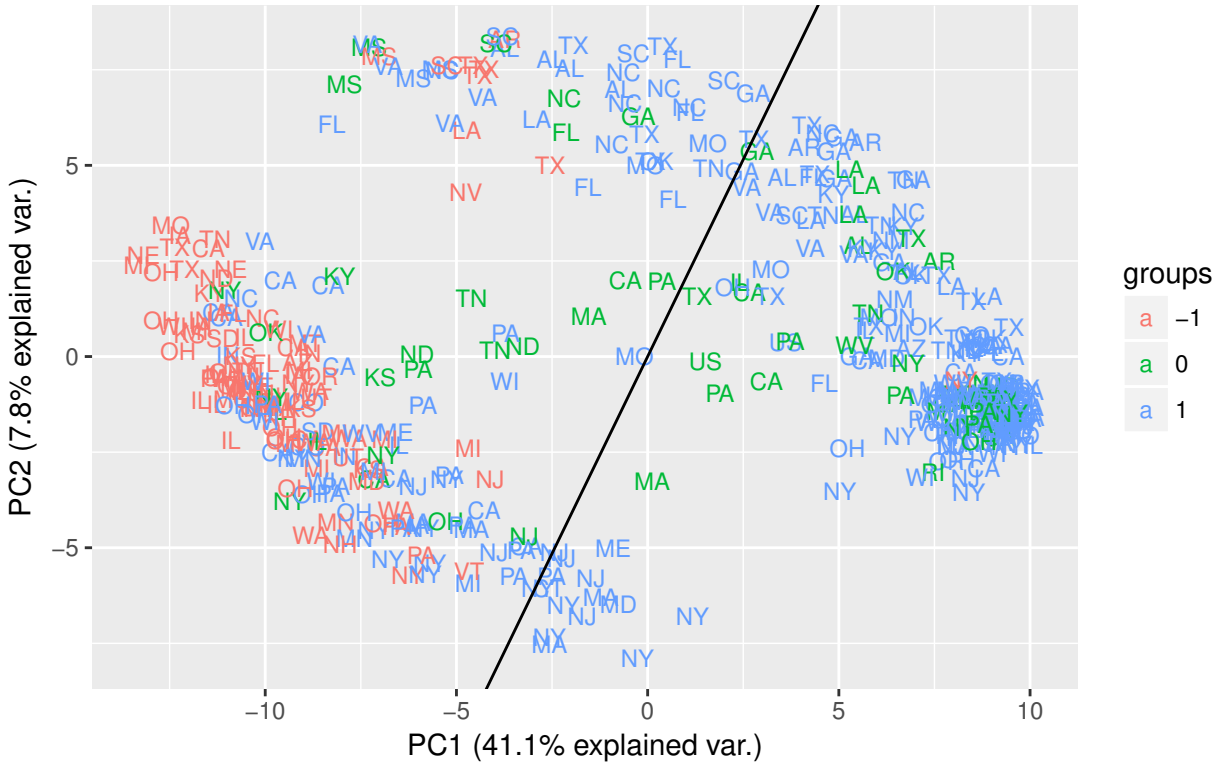
```
ggbiplot(votes.pca, obs.scale = 1, var.axes = FALSE, group = factor(votes6364[,
    149]), labels = state, labels.size = 3) + geom_abline(slope = -votes.pca$rotation[1,
    149]/votes.pca$rotation[2, 149])
```



This is a bit closer to being along party lines.

The Clean Air Act (vote 47):

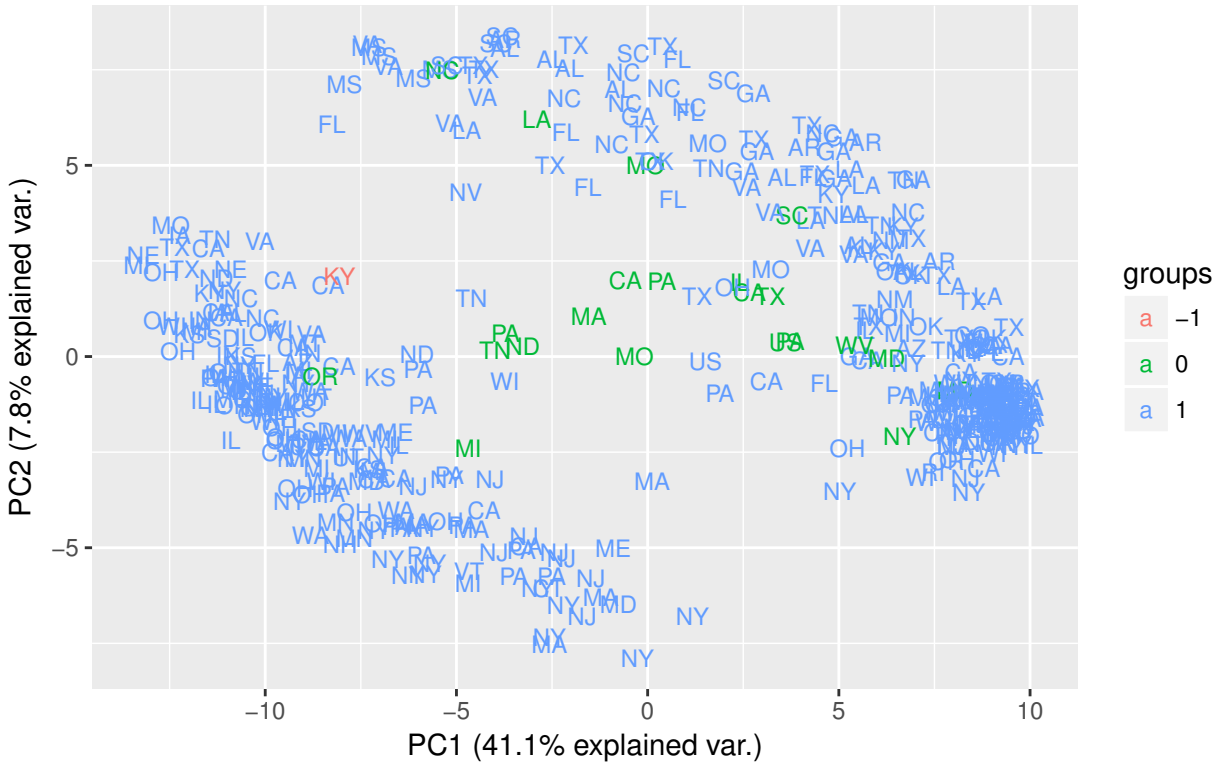
```
ggbiplot(votes.pca, obs.scale = 1, var.axes = FALSE, group = factor(votes6364[,
    47]), labels = state, labels.size = 3) + geom_abline(slope = -votes.pca$rotation[1,
    47]/votes.pca$rotation[2, 47])
```



The line is not doing a good job here. Most Democrats voted yes, but the behavior of Republicans is more complicated. We' might'd have to go to higher-order components if we wanted better separation.

The Tonkin Gulf Resolution (vote 197):

```
ggbiplot(votes.pca, obs.scale = 1, var.axes = FALSE, group = factor(votes6364[,
    197]), labels = state, labels.size = 3)
```



Just about everyone voted yes except Eugene Siler (who didn't show up for the vote but was a "paired no.")

It seems that the first component represents something like economic ideology and the second component represents something like social ideology. If you think there really are latent random variables called "economic ideology" and "social ideology", then **factor analysis** might be a good alternative to PCA. (Exploratory factor analysis has a questionable reputation among statisticians these days, but it's fine if you check your model and don't make causal claims.)

9.2 Regression with lots of predictors

Optional reading: Ch. 3.4 of the Elements of Statistical Learning.

In this section, we'll consider situations with "lots" of predictors, where lots might mean ten or twenty (but not the "$p > n$" situation with more predictors than observations, which presents a distinctive set of challenges.) In this situation, where prediction is not the goal (or at least not the only goal), there may be no single "best" model. Rather, different models might give you different insights about the data.

9.2.1 Prostate cancer

The `prostate` data set in the `ElemStatLearn` package contains measurements of 97 men suffering from prostate cancer.

```
# install.packages('ElemStatLearn')
library(ElemStatLearn)
data(prostate)
summary(prostate)
```

```
##      lcavol           lweight           age              lbph
##  Min.   :-1.3471   Min.   :2.375   Min.   :41.00   Min.   :-1.3863
```

```
##  1st Qu.: 0.5128   1st Qu.:3.376   1st Qu.:60.00   1st Qu.:-1.3863
##  Median : 1.4469   Median :3.623   Median :65.00   Median : 0.3001
##  Mean   : 1.3500   Mean   :3.629   Mean   :63.87   Mean   : 0.1004
##  3rd Qu.: 2.1270   3rd Qu.:3.876   3rd Qu.:68.00   3rd Qu.: 1.5581
##  Max.   : 3.8210   Max.   :4.780   Max.   :79.00   Max.   : 2.3263
##       svi              lcp              gleason          pgg45
##  Min.   :0.0000   Min.   :-1.3863   Min.   :6.000   Min.   :  0.00
##  1st Qu.:0.0000   1st Qu.:-1.3863   1st Qu.:6.000   1st Qu.:  0.00
##  Median :0.0000   Median :-0.7985   Median :7.000   Median : 15.00
##  Mean   :0.2165   Mean   :-0.1794   Mean   :6.753   Mean   : 24.38
##  3rd Qu.:0.0000   3rd Qu.: 1.1787   3rd Qu.:7.000   3rd Qu.: 40.00
##  Max.   :1.0000   Max.   : 2.9042   Max.   :9.000   Max.   :100.00
##       lpsa            train
##  Min.   :-0.4308   Mode :logical
##  1st Qu.: 1.7317   FALSE:30
##  Median : 2.5915   TRUE :67
##  Mean   : 2.4784
##  3rd Qu.: 3.0564
##  Max.   : 5.5829
```

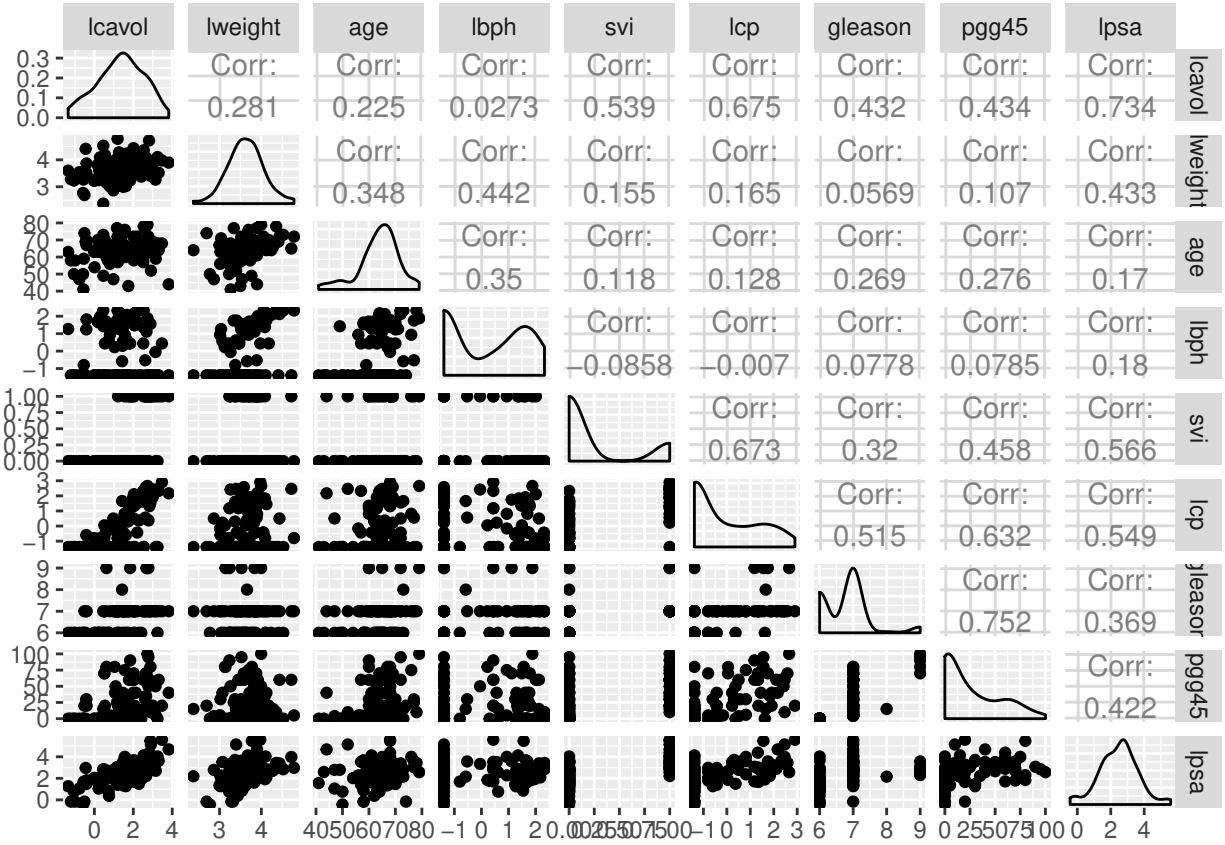
```
apply(prostate, 2, sd)
```

```
##     lcavol    lweight        age       lbph        svi        lcp
##  1.1786249  0.4284112  7.4451171  1.4508066  0.4139949  1.3982496
##    gleason      pgg45       lpsa      train
##  0.7221341 28.2040346  1.1543291  0.4645972
```

The response is `lpsa`, the log of "prostate specific antigen," a protein which is often at an elevated level in prostate cancer sufferers. There are also eight predictor variables (and an indicator `train` which we'll disregard.)

Draw a pairs plot (omitting `train`):

```
library(GGally)
ggpairs(prostate[, 1:9])
```



`lpsa` seems to be related to most of the other variables. It seems to have a reasonably symmetric distribution, so we won't transform further. (You could make arguments for transforming some of the predictors,but for simplicity, we won't.)

Eight predictors is getting toward the borderline where we need to use variable selection or regularization to avoid overfitting. But first, let's start by throwing all the predictors into a linear model.

9.2.2 Full linear model

Because there are lots of variables and I don't know much about prostate cancer, I'll rescale the variables so they all have mean 0 and SD 1, to allow easy comparison of their coefficients. (I prefer to not standardize the response.) Then we'll fit a full linear model, with all eight predictors.

```
predictors.scaled = scale(prostate[, 1:8])
lpsa = prostate$lpsa
prostate.lm = lm(lpsa ~ predictors.scaled)
```

As we've seen, the `summary()` function gives a bit of a mess, and that only gets worse as the number of predictors increases. Instead, we can use the `tidy()` function in `broom` to create a nice data frame of coefficients and their confidence intervals.

```
library(broom)
prostate.lm.tidy = tidy(prostate.lm, conf.int = TRUE)
data.frame(prostate.lm.tidy[, 1], round(prostate.lm.tidy[, -1], 2))
```

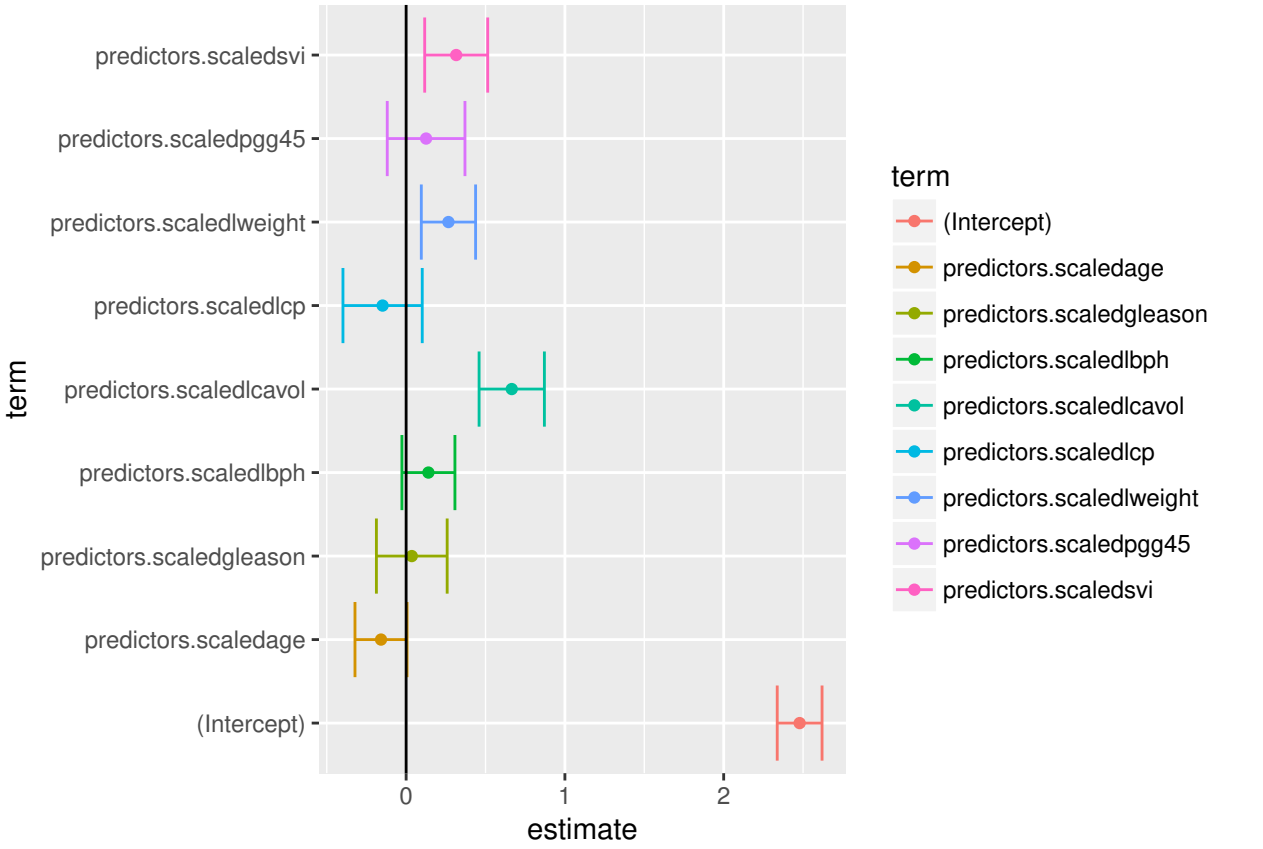
```
##      prostate.lm.tidy...1. estimate std.error statistic p.value conf.low
## 1              (Intercept)     2.48      0.07     34.90    0.00     2.34
## 2  predictors.scaledlcavol     0.67      0.10      6.43    0.00     0.46
## 3 predictors.scaledlweight     0.27      0.09      3.10    0.00     0.10
```

```
## 4     predictors.scaledage    -0.16      0.08     -1.92    0.06    -0.32
## 5    predictors.scaledlbph     0.14      0.08      1.67    0.10    -0.03
## 6     predictors.scaledsvi     0.32      0.10      3.16    0.00     0.12
## 7     predictors.scaledlcp    -0.15      0.13     -1.18    0.24    -0.40
## 8 predictors.scaledgleason     0.04      0.11      0.32    0.75    -0.19
## 9   predictors.scaledpgg45     0.13      0.12      1.02    0.31    -0.12
##   conf.high
## 1      2.62
## 2      0.87
## 3      0.44
## 4      0.01
## 5      0.31
## 6      0.51
## 7      0.10
## 8      0.26
## 9      0.37
```

The standard errors of the coefficients are of similar magnitude. The biggest coefficient is for `lcavol` (log cancer volume.) Since the predictors are scaled and response is a natural log, small coefficients (of the order of 0.1 or 0.2) can be interpreted as the proportion change in the prediction unlogged response if the predictor is one SD higher. So if age were one SD higher (about 7.5 years) and everything else was constant, the model's predicted PSA would be about 16% lower.

We can also draw a **TIE fighter plot** of the coefficients (that might be a name I just made up):

```
# library(tidyverse)
ggplot(prostate.lm.tidy, aes(x = estimate, y = term, xmin = conf.low, xmax = conf.high,
    color = term)) + geom_point() + geom_errorbarh() + geom_vline(xintercept = 0)
```



Without worrying too much about significance, we see that some of the coefficients are very small compared to their margins of error. We should thus consider a model with fewer predictors.

9.2.3 All subsets

There are lots of ways to do variable selection. An old-school method is to do **stepwise** selection, but this has the fairly major weakness that it doesn't always choose the best model of a given size, let alone the right model. In fact, given a half-decent laptop, there's no difficulty in looking at *all* subsets of eight predictor variables, and finding the best (i.e. lowest squared erorr) subset for each size. `regsubsets()` in the `leaps` package does this for you. We run it on the unscaled data (excluding the `train` variable.)

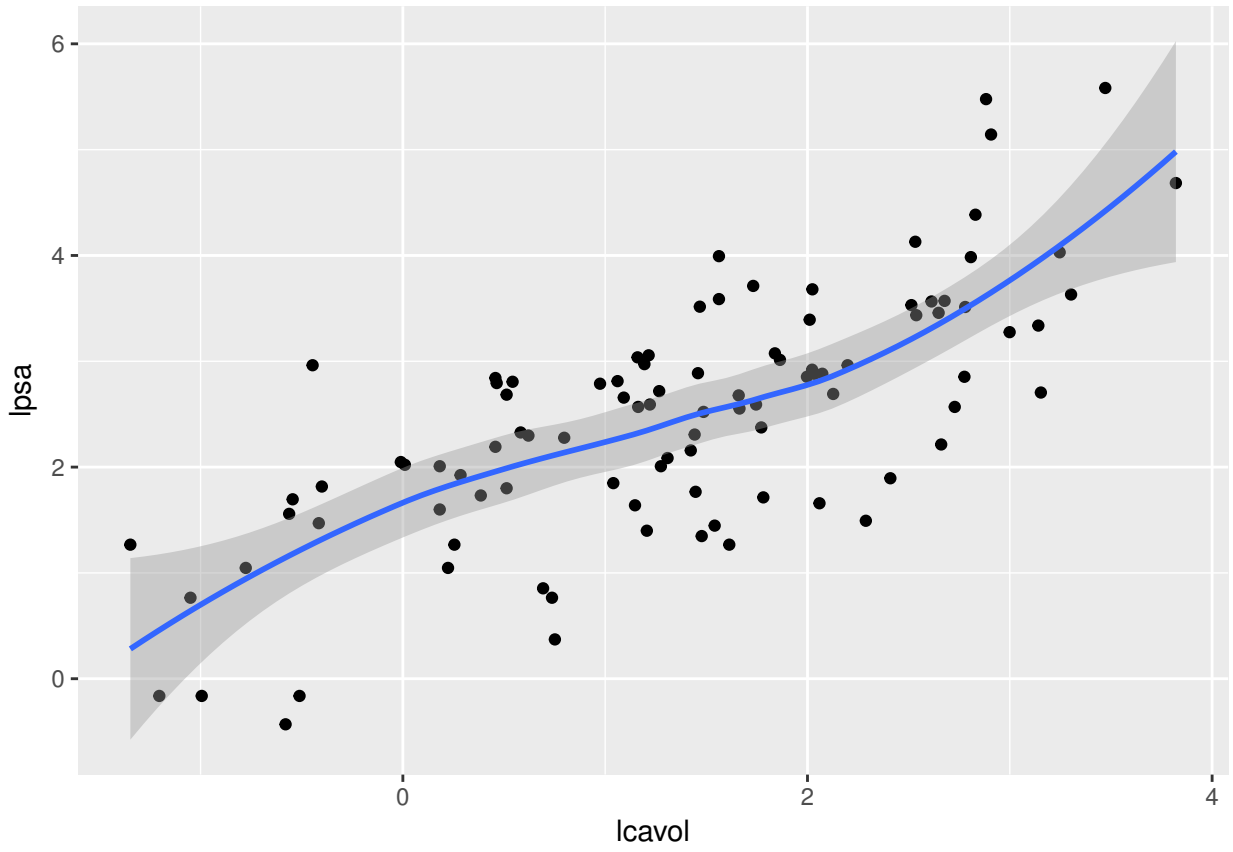
```
# install.packages('leaps')
library(leaps)
prostate.leaps = regsubsets(lpsa ~ . - train, data = prostate)
summary(prostate.leaps)$which
```

```
##   (Intercept) lcavol lweight   age  lbph   svi   lcp gleason pgg45
## 1        TRUE   TRUE   FALSE FALSE FALSE FALSE FALSE   FALSE FALSE
## 2        TRUE   TRUE    TRUE FALSE FALSE FALSE FALSE   FALSE FALSE
## 3        TRUE   TRUE    TRUE FALSE FALSE  TRUE FALSE   FALSE FALSE
## 4        TRUE   TRUE    TRUE FALSE  TRUE  TRUE FALSE   FALSE FALSE
## 5        TRUE   TRUE    TRUE  TRUE  TRUE  TRUE FALSE   FALSE FALSE
## 6        TRUE   TRUE    TRUE  TRUE  TRUE  TRUE FALSE   FALSE  TRUE
## 7        TRUE   TRUE    TRUE  TRUE  TRUE  TRUE  TRUE   FALSE  TRUE
## 8        TRUE   TRUE    TRUE  TRUE  TRUE  TRUE  TRUE    TRUE  TRUE
```

So we see that for example, the best 3-predictor model includes `lcavol`, `lweight`, and `svi`. (Note that if we only want to find the best models, it doesn't matter whether we run `regsubsets()` with the scaled or unscaled predictors.) We see that the best models for each size form a nested sequence, so a greedy algorithm would've worked here.

Now we can build up our understanding of the data by drawing plots, adding variables sequentially and checking for nonlinearity and interactions as we've done throughout the course. Start with log cancer volume as a single predictor:

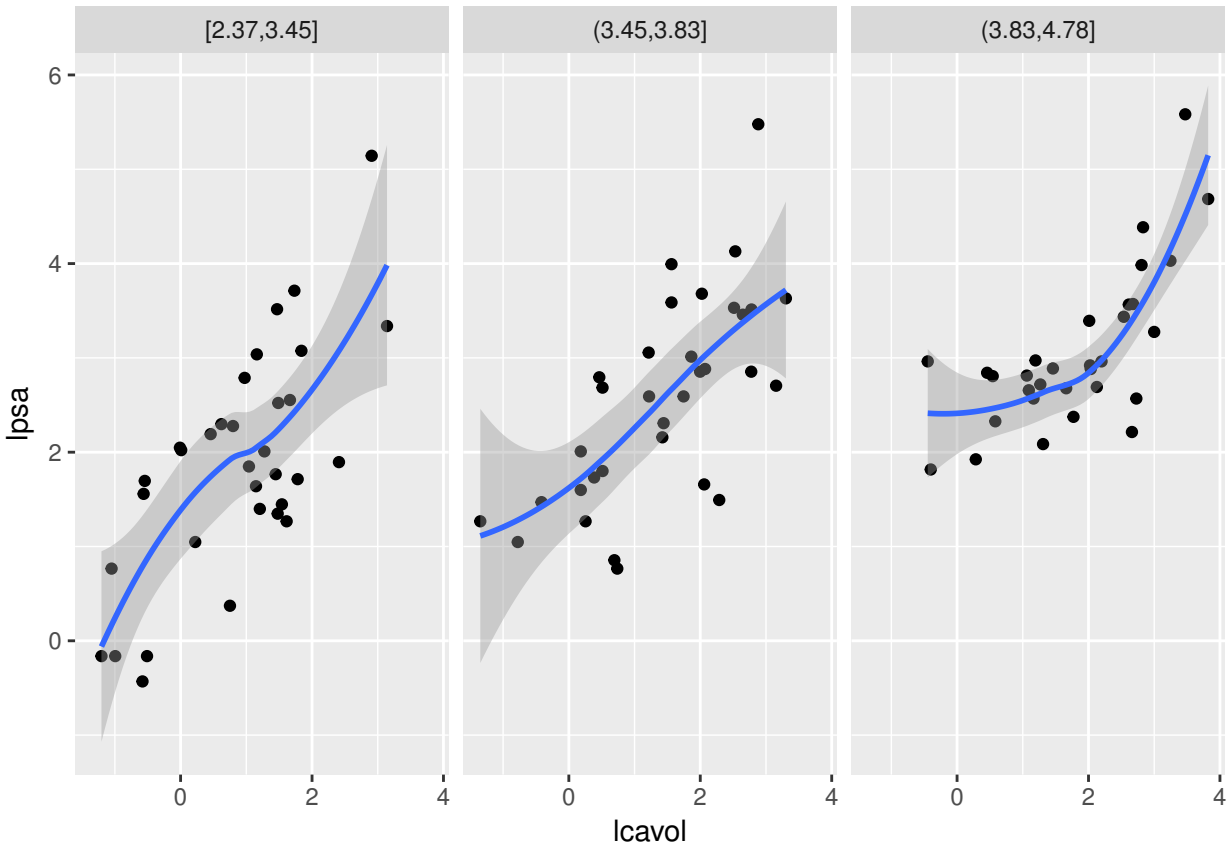
```
ggplot(prostate, aes(x = lcavol, y = lpsa)) + geom_point() + geom_smooth()
```



The relationship seems reasonably close to linear here – we could draw a straight line entirely within the confidence band. If there was more curvature, I'd throw the model out and start from scratch using a nonparametric approach.

Now facet by log weight:

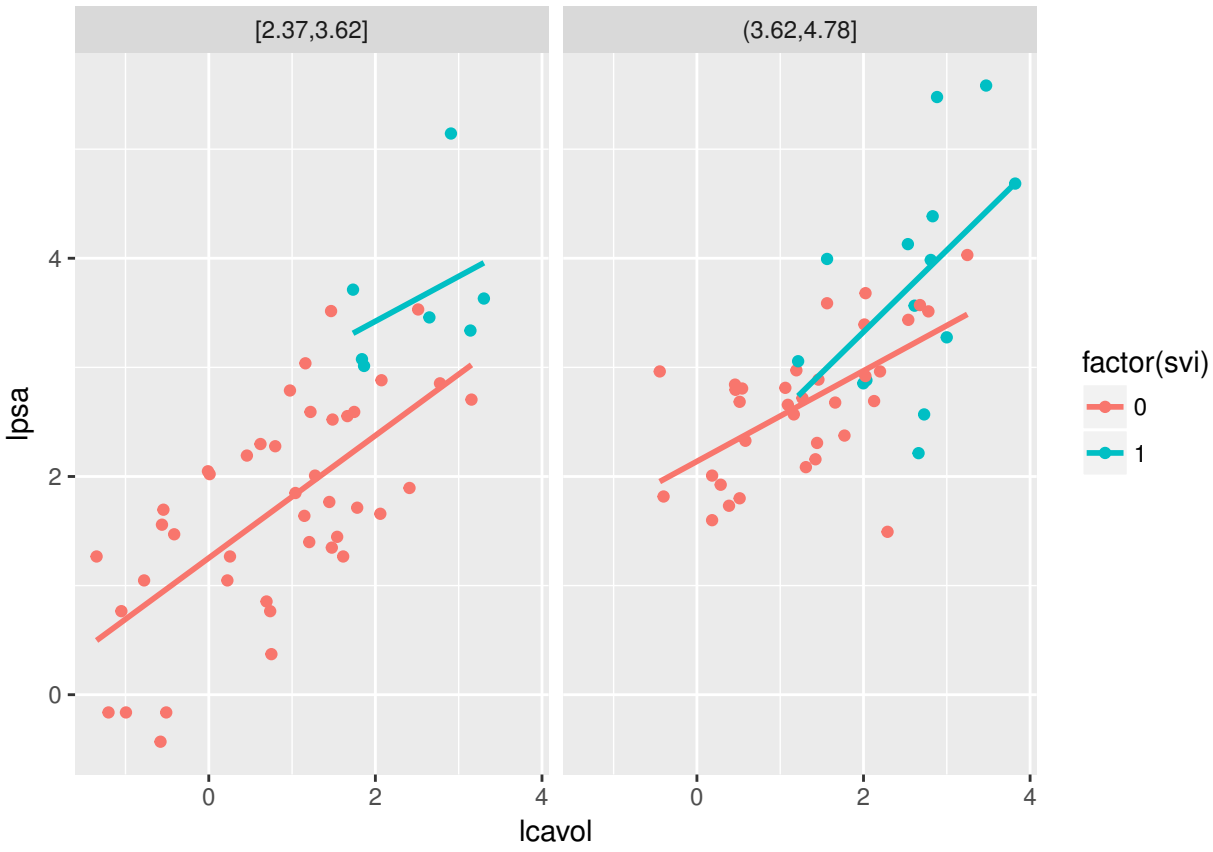
```
ggplot(prostate, aes(x = lcavol, y = lpsa)) + geom_point() + geom_smooth(span = 1) +
    facet_grid(~cut_number(lweight, n = 3))
```



There's an intriguing hint of nonlinearity here – the log PSA prediction increases more and more quickly with log cancer volume, but only for the heavy patients. With lots of data, we could switch to a multipredictor loess here, but since our sample is small, let's first see if this pattern shows up in other graphs.

Next, conditional `svi` (seminal vesicle invasion) which if present, indicates the cancer is advanced. Since this is a binary variable, we can distinguish it with color. However, as only 21 individuals in the data set have SVI, the number of facets we can show becomes limited, and it's hard to fit curves instead of lines.

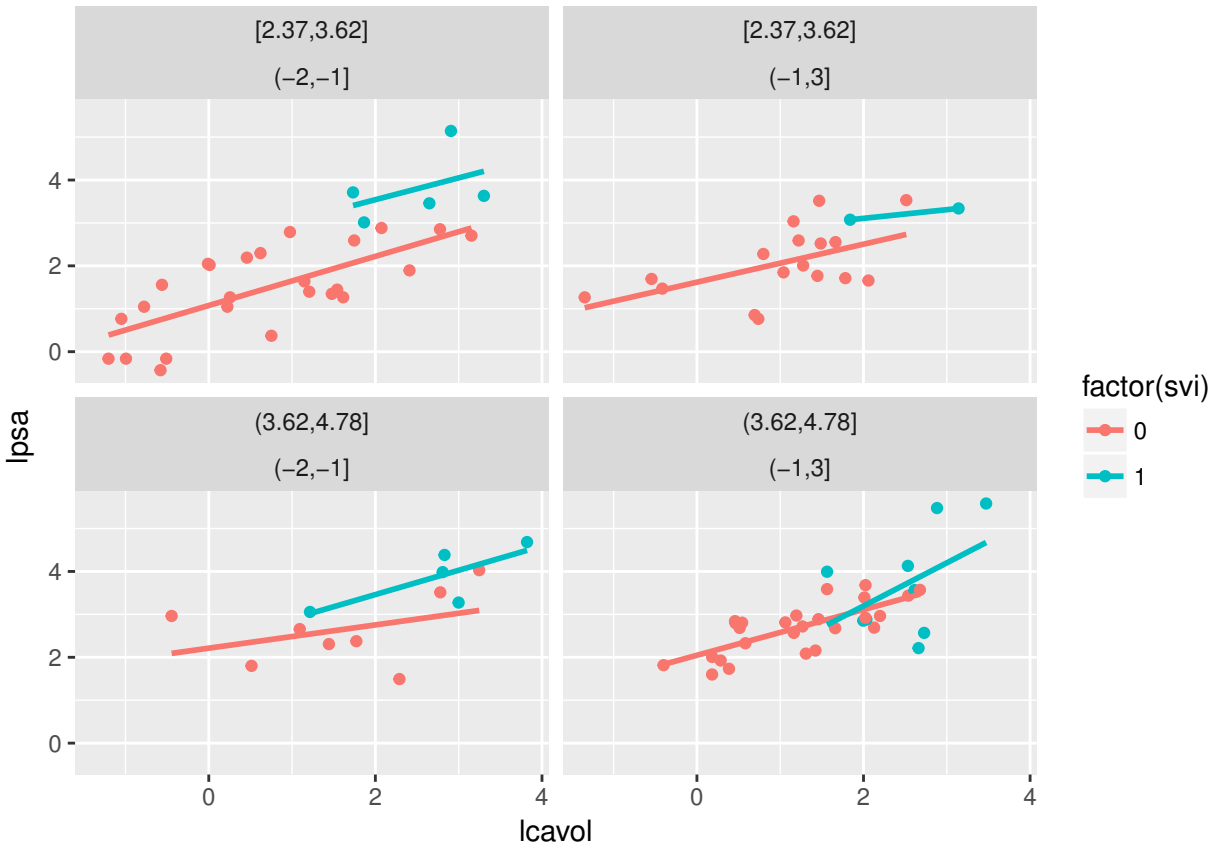
```
ggplot(prostate, aes(x = lcavol, y = lpsa, group = svi, color = factor(svi))) +
    geom_point() + geom_smooth(method = "lm", se = FALSE) + facet_wrap(~cut_number(lweight,
    n = 2))
```



There's not much hint of nonlinearity here. However, the lines on the right panel are far from parallel, so we should keep the possibility of an interaction in mind.

Now facet again on `lbph`, "log benign prostatic hyperplasia amount":

```
ggplot(prostate, aes(x = lcavol, y = lpsa, group = svi, color = factor(svi))) +
    geom_point() + geom_smooth(method = "lm", se = FALSE) + facet_wrap(~cut_number(lweight,
    n = 2) + cut(lbph, breaks = c(-2, -1, 3)))
```



It's getting harder to be confident about an interaction: the blue lines vary in slope, but there's based on very small samples. The red lines have different heights but are similar in slope.

In EDA we're not always required to find a "best" model, and even if we were we can decide on what best means subjectively. So if you wanted to fit a linear model with `lcavol`, `lweight`, and `svi` as predictors plus interactions, you're free to do so and then call that "best" because of the complexity you can get out of a relatively small number of variables. On the other hand, if you want a (somewhat) objective decision for "best", you can just find the model that optimizes your favorite criterion. For example, if you like Mallow's C_p:

```
summary(prostate.leaps)$cp
```

```
## [1] 27.406210 14.747299  6.173546  6.185065  5.816804  6.466493  7.100428
## [8]  9.000000
```

the best model is the five-predictor one:

```
tidy(lm(lpsa ~ lcavol + lweight + svi + lbph + age, data = prostate))
```

```
##          term    estimate  std.error  statistic      p.value
## 1 (Intercept)  0.49472926 0.87651892  0.5644251 5.738535e-01
## 2      lcavol  0.54399786 0.07463455  7.2888209 1.106003e-10
## 3     lweight  0.58821270 0.19789916  2.9722850 3.781885e-03
## 4         svi  0.71490398 0.20653188  3.4614703 8.201880e-04
## 5        lbph  0.10122333 0.05758677  1.7577532 8.215221e-02
## 6         age -0.01644485 0.01067524 -1.5404667 1.269169e-01
```

Note, however, that the standard errors and P-values are wrong – we selected the model *because* it fitted the data well, so the goodness-of-fit will be overestimated. If in doubt, bootstrap.

9.2.4 If prediction is a major goal

If prediction is a major goal (in addition to interpretation), there are lots of modern techniques to consider, such as principal components regression, ridge regressions, and Lasso. These are beyond the scope of this course; go read *Elements of Statistical Learning*.