Principal Components and Biased Regression Methods

Alexander McLain

September 7, 2023

```
library(printr)
library(formatR)
library(tidyverse)
library(pls)
library(glmnet)
library(psych)
library(FactoMineR)
```

Principal components analysis (PCA)

Olympic Example

Here we'll further look at some examples of PCA. First, we'll look at the olympic dataset which is a data frame with 33 rows and 10 columns events of the decathlon: 100 meters (100), long jump (long), shotput (poid), high jump (haut), 400 meters (400), 110-meter hurdles (110), discus throw (disq), pole vault (perc), javelin (jave) and 1500 meters (1500).

```
library(ade4)
```

```
##
## Attaching package: 'ade4'
```

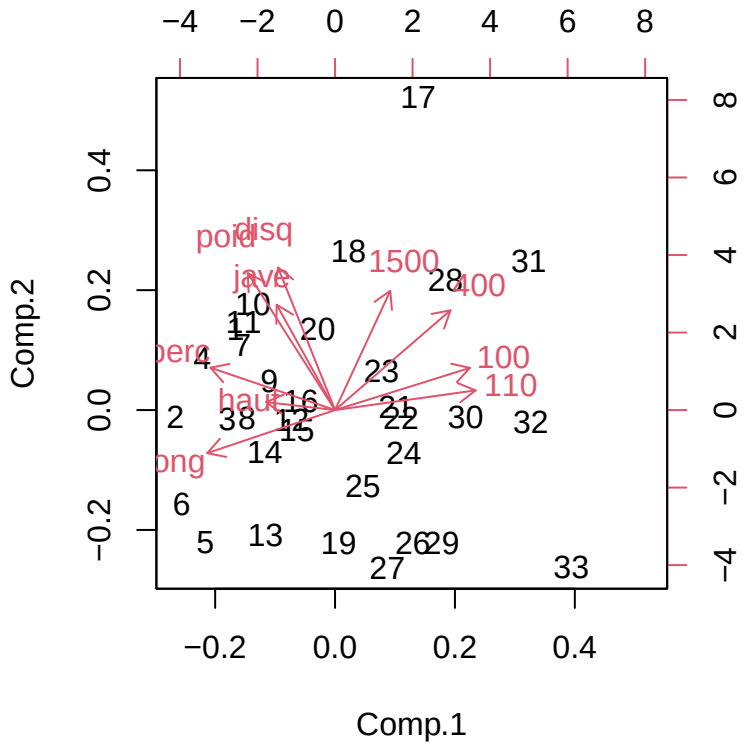
```
## The following object is masked from 'package:FactoMineR':
##
##     reconst
```

```
data(olympic)
head(olympic$tab)
```

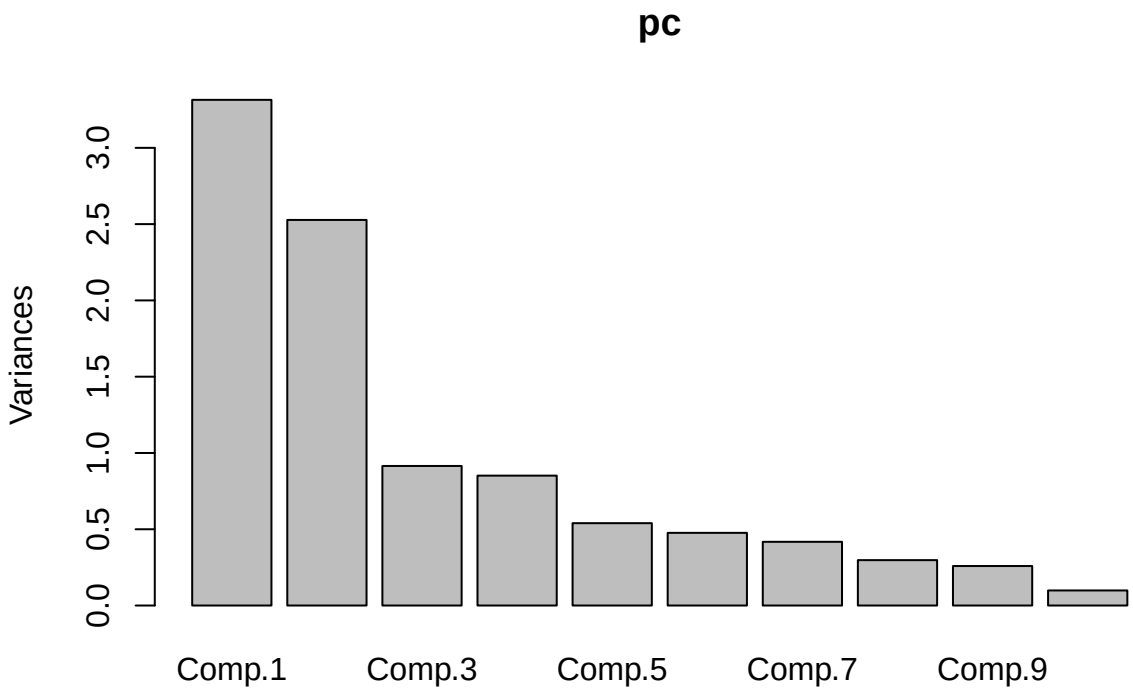
100	long	poid	haut	400	110	disq	perc	jave	1500
11.25	7.43	15.48	2.27	48.90	15.13	49.28	4.7	61.32	268.95
10.87	7.45	14.97	1.97	47.71	14.46	44.36	5.1	61.76	273.02
11.18	7.44	14.20	1.97	48.29	14.81	43.66	5.2	64.16	263.20
10.62	7.38	15.02	2.03	49.06	14.72	44.80	4.9	64.04	285.11
11.02	7.43	12.92	1.97	47.44	14.40	41.20	5.2	57.46	256.64
10.83	7.72	13.58	2.12	48.34	14.18	43.06	4.9	52.18	274.07

```
olym_Z <- scale(olympic$tab)

pc = princomp(olym_Z) #default - does NOT center and scale

#default R plots with princomp
```

```
biplot(pc)
```




```
screeplot(pc)
```



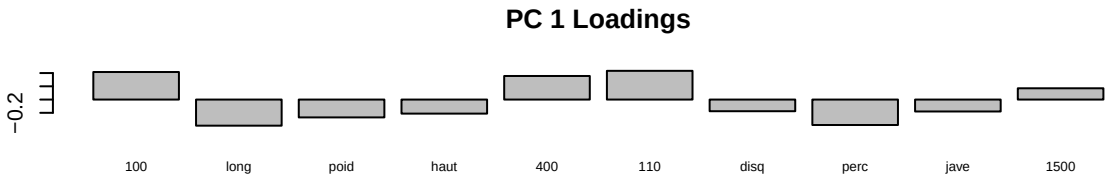

```
#scatter plots - patterns among observations
round(pc$scores[ c(2, 17, 33), c(1:2)], 3)
```

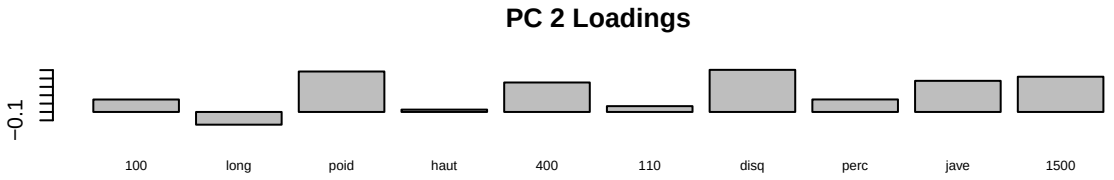
	Comp.1	Comp.2
2	-2.787	-0.101
17	1.452	4.771
33	4.124	-2.396

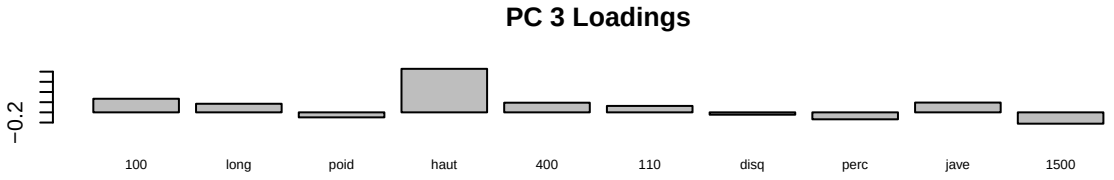
```
round(olym_Z[ c(2, 17, 33),], 3)
```

	100	long	poid	haut	400	110	disq	perc	jave	1500
2	-1.341	1.041	0.746	-0.135	-1.465	-1.162	0.539	1.078	0.422	-0.221
17	1.083	-1.260	1.572	0.184	1.873	1.995	2.233	0.181	2.395	1.932
33	1.536	0.186	-2.783	-0.774	1.340	2.272	-2.149	-1.912	-0.819	-0.444

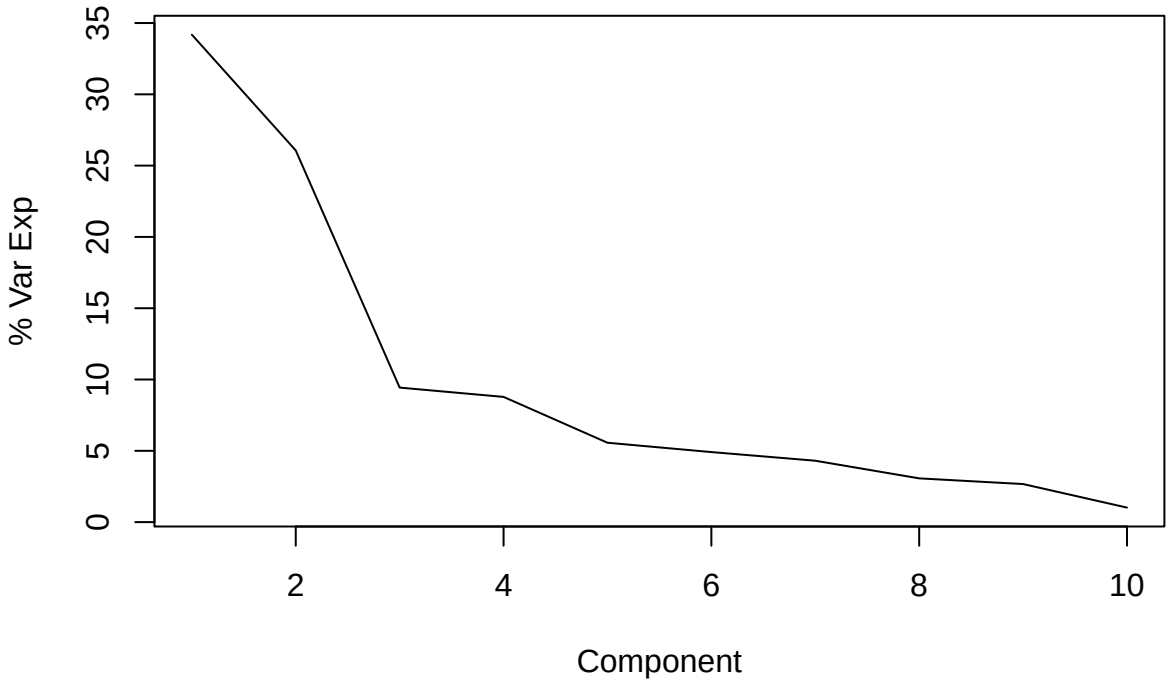
```
#loadings - variables that contribute to these patterns
par(mfrow=c(3,1))
barplot(pc$loadings[,1],cex.names=.6,main="PC 1 Loadings")
barplot(pc$loadings[,2],cex.names=.6,main="PC 2 Loadings")
barplot(pc$loadings[,3],cex.names=.6,main="PC 3 Loadings")
```



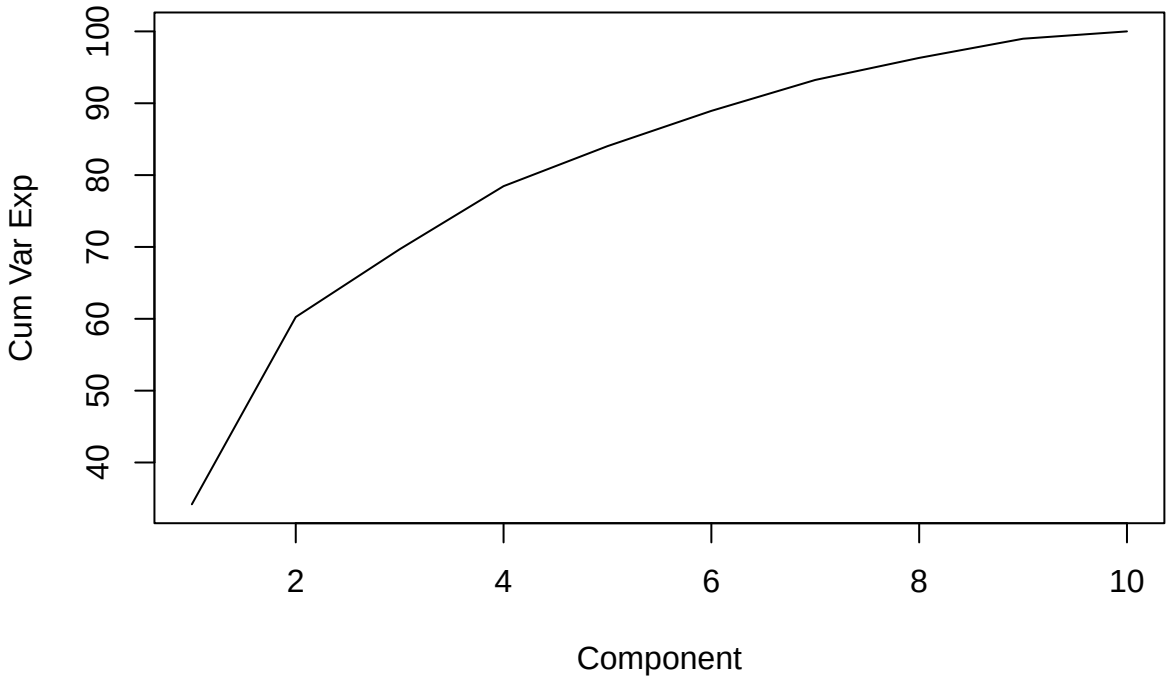





```
par(mfrow=c(1,1))
varex = 100*pc$sdev^2/sum(pc$sdev^2)
plot(varex,type="l",ylab="% Var Exp",xlab="Component")
```




```
#cumulative variance explained
cvarex = NULL
for(i in 1:ncol(olym_Z)){
  cvarex[i] = sum(varex[1:i])
}
plot(cvarex,type="l",ylab="Cum Var Exp",xlab="Component")
```



Now we'll rotate the PCs using the `varimax` rotation.

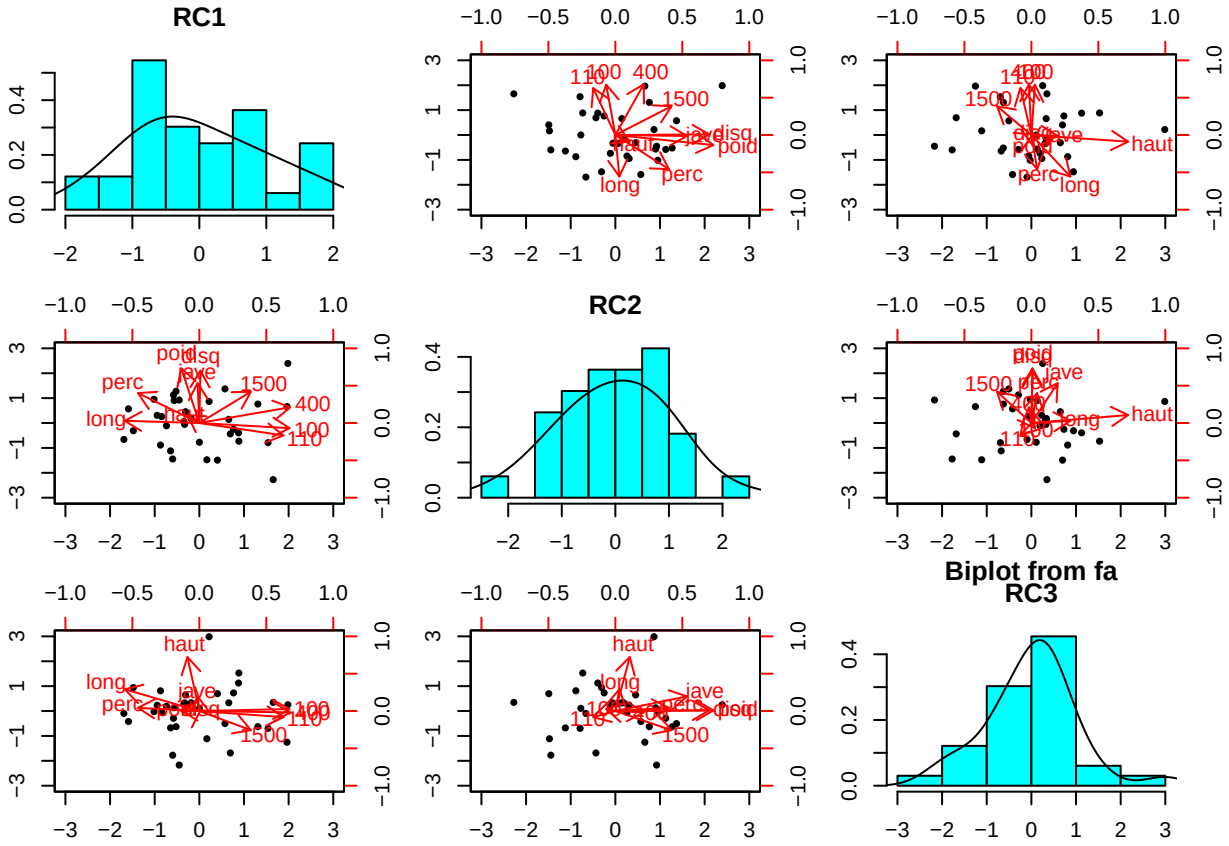
```
`?`(principal)
```

```
Principal components analysis (PCA)

Description:

     Does an eigen value decomposition and returns eigen values,
     loadings, and degree of fit for a specified number of components.
     Basically it is just doing a principal components analysis (PCA)
     for n principal components of either a correlation or covariance
     matrix.  Can show the residual correlations as well. The quality
     of reduction in the squared correlations is reported by comparing
     residual correlations to original correlations. Unlike princomp,
     this returns a subset of just the best nfactors. The eigen vectors
     are rescaled by the sqrt of the eigen values to produce the
     component loadings more typical in factor analysis.

Usage:

     principal(r, nfactors = 1, residuals = FALSE,rotate="varimax",n.obs=NA, covar=FALSE,
      scores=TRUE,missing=FALSE,impute="median",oblique.scores=TRUE,method="regression",
      use ="pairwise",cor="cor",correct=.5,weight=NULL,...)
```

```
pca_iris_rotated <- principal(olym_Z, rotate = "varimax", nfactors = 3, scores = TRUE)
pca_iris_rotated$loadings  # Loadings returned by principal()
```

```
Loadings:
     RC1    RC2    RC3
100   0.841
long -0.696         0.364
poid -0.168  0.915
haut -0.111  0.134  0.903
400   0.848  0.264
110   0.785 -0.207 -0.103
disq         0.876
perc -0.572  0.500
jave         0.668  0.245
1500  0.481  0.523 -0.320

                 RC1   RC2   RC3
SS loadings    3.128 2.715 1.126
Proportion Var 0.313 0.271 0.113
Cumulative Var 0.313 0.584 0.697
```

```
biplot(pca_iris_rotated)
```



Food Example

Here is a principal component analysis of some food data

```
food_df <- read.csv("food.data.csv")
food_df <- as.matrix(food_df[,1:7])
dim(food_df)
```

```
## [1] 961   7
```

```
head(food_df[,1:5])
```

Fat.grams	Food.energy.calories	Carbohydrates.grams	Protein.grams	Cholesterol.mg
2	25	2	0	2
6	60	2	0	4
1	90	22	4	0
0	90	22	3	0
0	10	1	1	0
1	70	21	4	0

```
apply(food_df, 2, sd)
```

```
##            Fat.grams Food.energy.calories  Carbohydrates.grams
##              29.11286             542.91736              78.49855
##        Protein.grams       Cholesterol.mg         weight.grams
##              10.12285             119.96059             175.90936
##  Saturated.fat.grams
```

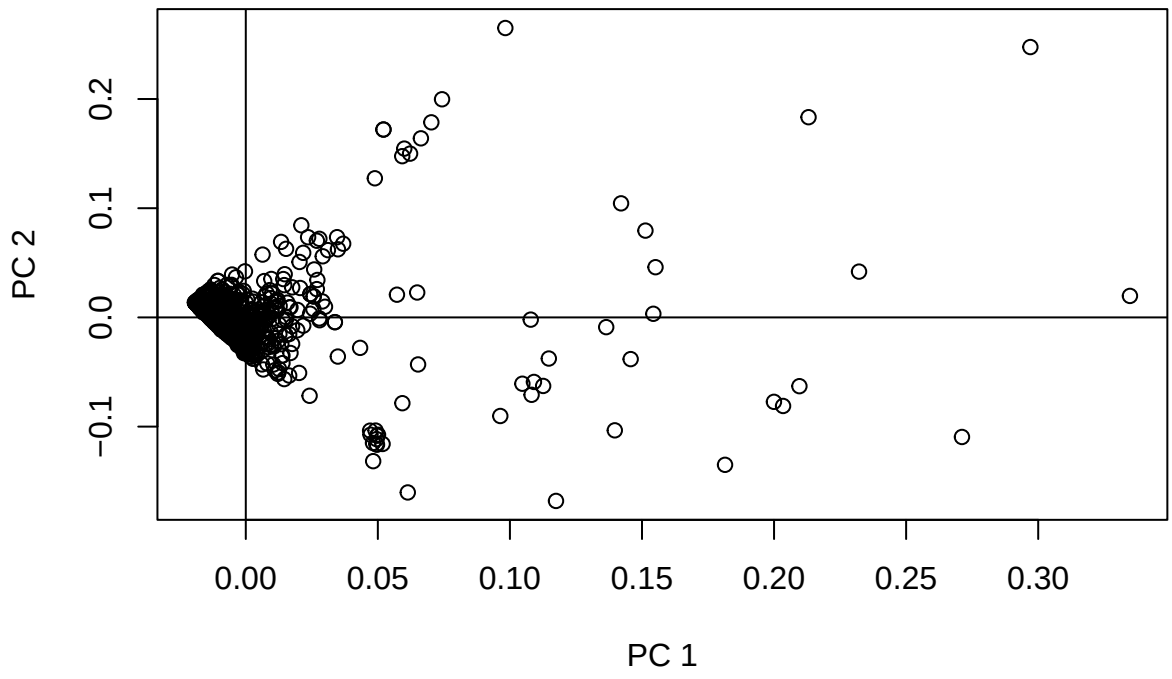
```
##              10.74436
```

Now we'll do some principal components on the food data

```
#First we'll standardize the data
c_food_df <- scale(food_df)
#PCA - take SVD to get solution
svdd = svd(c_food_df)
U = svdd$u
V = svdd$v #PC loadings
D = svdd$d
Z = c_food_df%*%V #PC "scores"
round(V,3)
```

0.381	0.446	0.159	-0.438	-0.165	-0.576	-0.284
0.434	-0.039	0.223	-0.159	-0.285	0.127	0.799
0.387	-0.427	0.316	0.140	-0.464	0.286	-0.502
0.317	-0.403	-0.697	-0.478	0.121	0.063	-0.060
0.352	0.232	-0.505	0.685	-0.232	-0.204	0.046
0.383	-0.363	0.289	0.257	0.667	-0.354	0.039
0.381	0.518	0.025	-0.018	0.403	0.633	-0.149

```
#PC scatterplot
i = 1; j = 2;
plot(U[,i],U[,j],type="p",xlab="PC 1",ylab="PC 2")
abline(h=0,v=0)
```



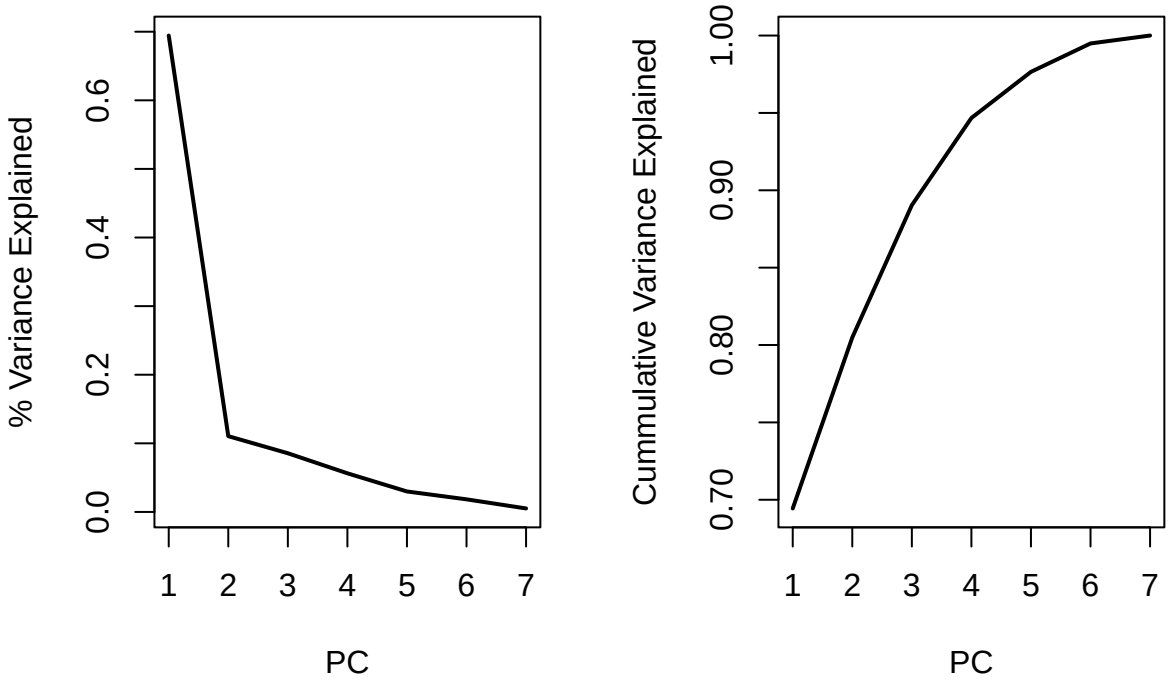
Now let's look at the variance explained and the scree plot

```
varex = 0
cumvar = 0
denom = sum(D^2)
for(i in 1:7){
```

```
  varex[i] = D[i]^2/denom
  cumvar[i] = sum(D[1:i]^2)/denom
}

#Percentage of variance explained by the first r variables.
round(100*varex,1)
```

```
## [1] 69.4 11.0  8.6  5.6  3.0  1.8  0.5
```

```
#screeplot
par(mfrow=c(1,2))
plot(1:7,varex,type="l",lwd=2,xlab="PC",ylab="% Variance Explained")
plot(1:7,cumvar,type="l",lwd=2,xlab="PC",ylab="Cummulative Variance Explained")
```

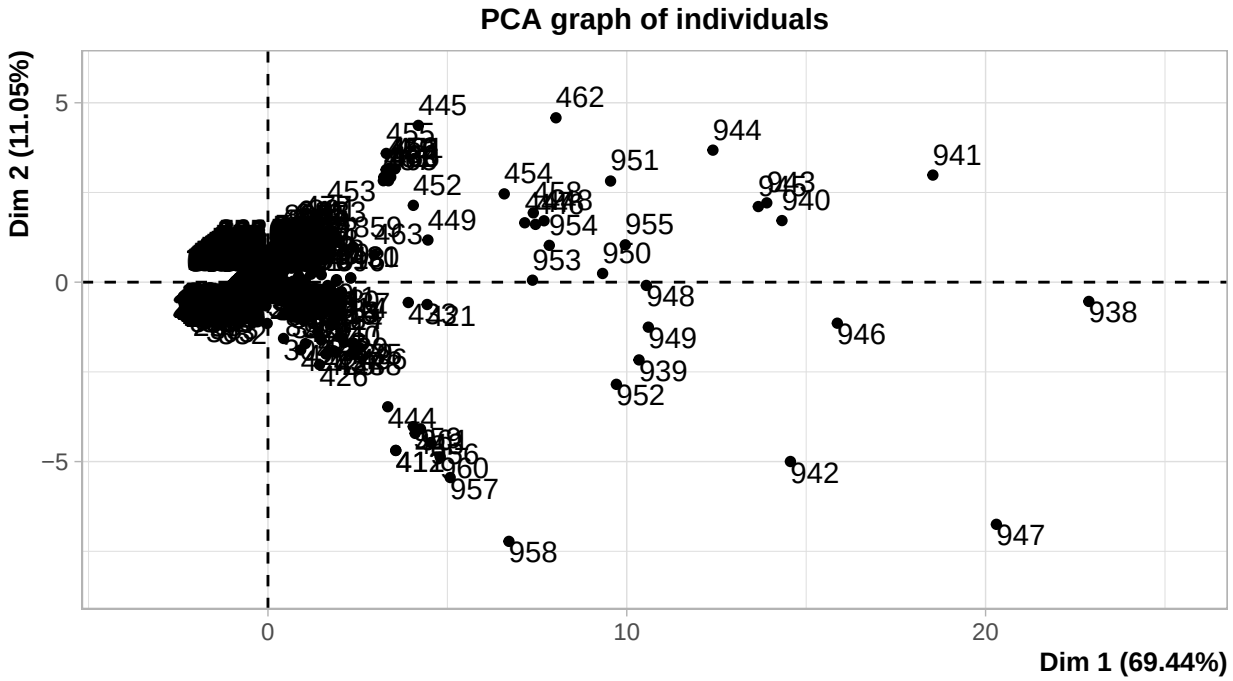


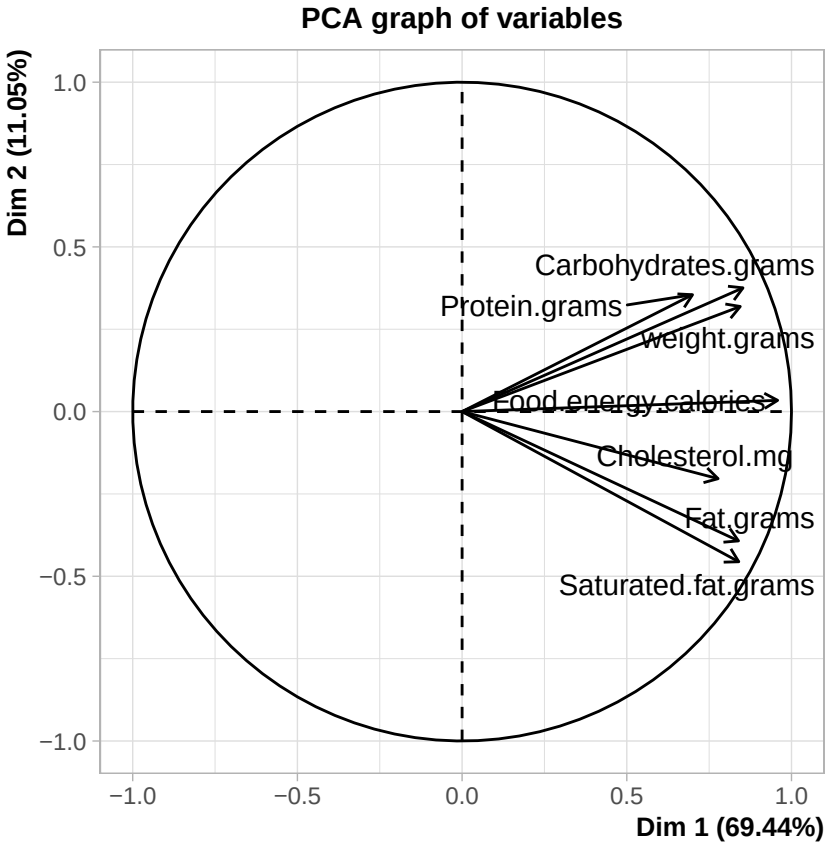
Here we'll refit using the `FactoMineR` function `PCA`

```
`?`(PCA)
```

```
Principal Component Analysis (PCA)

Description:

     Performs Principal Component Analysis (PCA) with supplementary
     individuals, supplementary quantitative variables and
     supplementary categorical variables.
     Missing values are replaced by the column mean.

Usage:

     PCA(X, scale.unit = TRUE, ncp = 5, ind.sup = NULL,
         quanti.sup = NULL, quali.sup = NULL, row.w = NULL,
         col.w = NULL, graph = TRUE, axes = c(1,2))
```

```
food_PCA <- PCA(food_df, ncp = 7)
```






```
round(food_PCA$var$coord, 3)
```

	Dim.1	Dim.2	Dim.3	Dim.4	Dim.5	Dim.6	Dim.7
Fat.grams	0.839	-0.393	-0.123	0.275	-0.075	0.206	0.054
Food.energy.calories	0.958	0.034	-0.172	0.100	-0.130	-0.045	-0.151
Carbohydrates.grams	0.853	0.376	-0.244	-0.088	-0.212	-0.102	0.095
Protein.grams	0.700	0.355	0.540	0.300	0.055	-0.023	0.011
Cholesterol.mg	0.777	-0.204	0.391	-0.430	-0.106	0.073	-0.009
weight.grams	0.844	0.319	-0.223	-0.161	0.305	0.127	-0.007
Saturated.fat.grams	0.840	-0.456	-0.020	0.011	0.184	-0.227	0.028

```
summary(food_PCA)
```

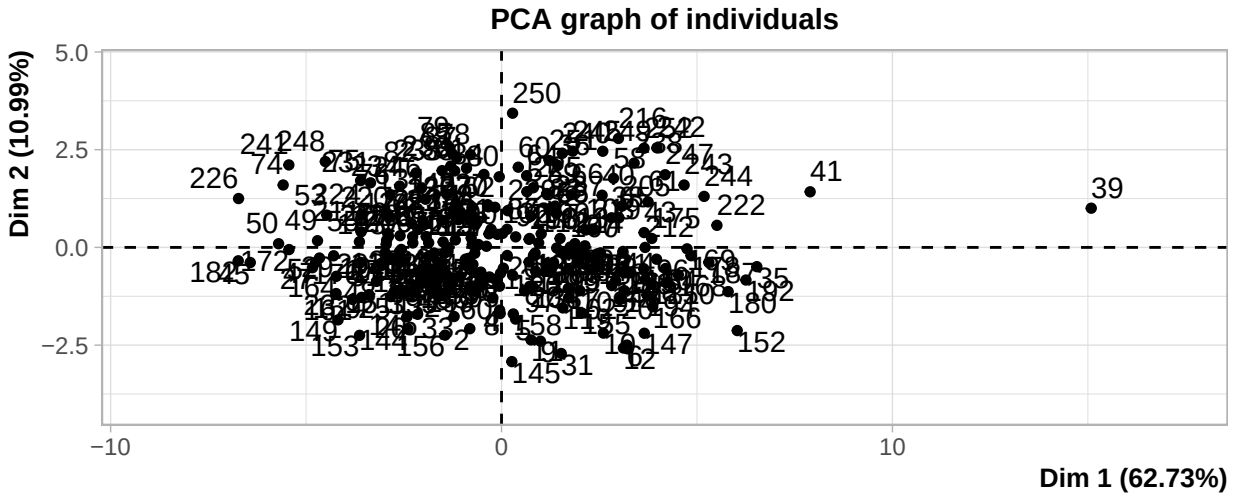
```
Call:
PCA(X = food_df, ncp = 7)


Eigenvalues
                       Dim.1   Dim.2   Dim.3   Dim.4   Dim.5   Dim.6   Dim.7
Variance               4.861   0.773   0.599   0.394   0.209   0.129   0.036
% of var.             69.439  11.046   8.554   5.632   2.983   1.836   0.510
Cumulative % of var.  69.439  80.485  89.039  94.671  97.654  99.490 100.000

Individuals (the 10 first)
                        Dist    Dim.1    ctr   cos2    Dim.2    ctr   cos2
1                    |  1.324 | -1.217  0.032  0.845 | -0.376  0.019  0.081 |
2                    |  1.252 | -1.100  0.026  0.772 | -0.476  0.030  0.144 |
3                    |  1.017 | -0.934  0.019  0.843 | -0.052  0.000  0.003 |
4                    |  1.064 | -0.979  0.021  0.846 | -0.076  0.001  0.005 |
5                    |  1.258 | -1.202  0.031  0.914 | -0.262  0.009  0.043 |
6                    |  1.032 | -0.955  0.020  0.857 | -0.059  0.000  0.003 |
7                    |  0.953 | -0.866  0.016  0.826 |  0.020  0.000  0.000 |
8                    |  1.014 | -0.817  0.014  0.649 |  0.036  0.000  0.001 |
9                    |  1.120 | -1.095  0.026  0.956 | -0.160  0.003  0.020 |
10                   |  1.123 | -1.098  0.026  0.957 | -0.155  0.003  0.019 |
                      Dim.3    ctr   cos2
1                    -0.119  0.002  0.008 |
2                    -0.150  0.004  0.014 |
3                     0.025  0.000  0.001 |
4                    -0.038  0.000  0.001 |
5                    -0.066  0.001  0.003 |
6                     0.038  0.000  0.001 |
7                    -0.121  0.003  0.016 |
8                    -0.341  0.020  0.113 |
9                    -0.052  0.000  0.002 |
10                   -0.052  0.000  0.002 |

Variables
                         Dim.1    ctr   cos2    Dim.2    ctr   cos2    Dim.3
Fat.grams             |  0.839 14.487  0.704 | -0.393 19.928  0.154 | -0.123
Food.energy.calories  |  0.958 18.876  0.918 |  0.034  0.150  0.001 | -0.172
Carbohydrates.grams   |  0.853 14.960  0.727 |  0.376 18.242  0.141 | -0.244
Protein.grams         |  0.700 10.068  0.489 |  0.355 16.275  0.126 |  0.540
Cholesterol.mg        |  0.777 12.425  0.604 | -0.204  5.363  0.041 |  0.391
```

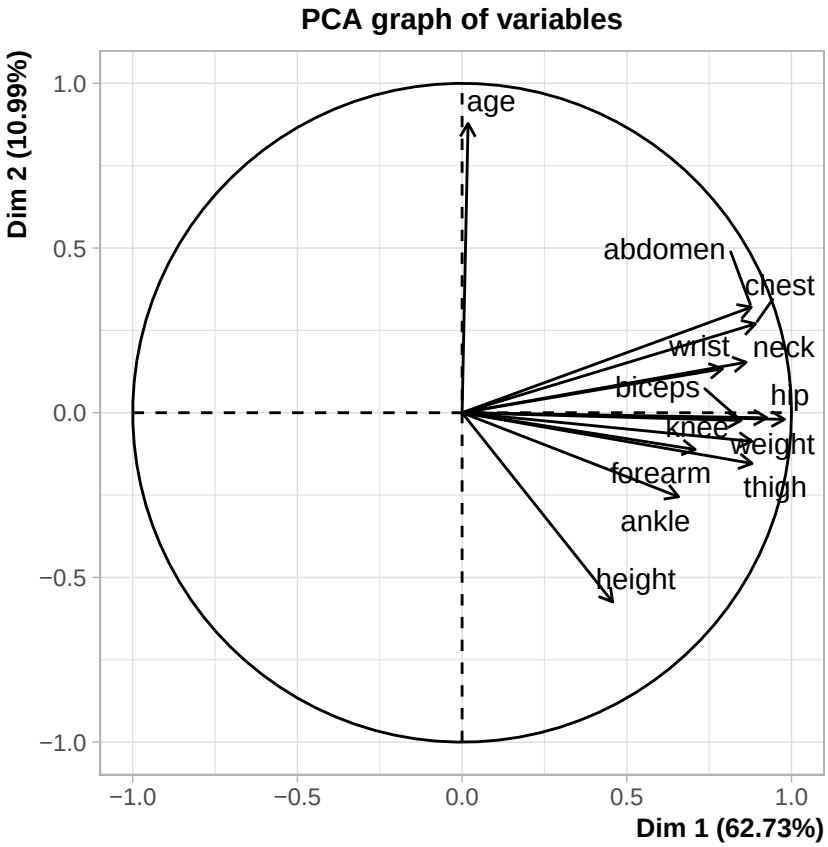
```
weight.grams          |  0.844 14.671  0.713 |  0.319 13.167  0.102 | -0.223
Saturated.fat.grams   |  0.840 14.514  0.705 | -0.456 26.875  0.208 | -0.020
                         ctr   cos2
Fat.grams              2.515  0.015 |
Food.energy.calories   4.952  0.030 |
Carbohydrates.grams    9.962  0.060 |
Protein.grams         48.619  0.291 |
Cholesterol.mg        25.549  0.153 |
weight.grams           8.338  0.050 |
Saturated.fat.grams    0.065  0.000 |
```

Principal components regression (PCR)

Now we'll do a PCR using the body fat data

```
bf_dat <- read.csv("bodyfat2.csv")
bf_df <- data.frame(bf_dat)
X_only <- bf_df[,-c(1,2)]
bf_PCA <- PCA(X_only)
```

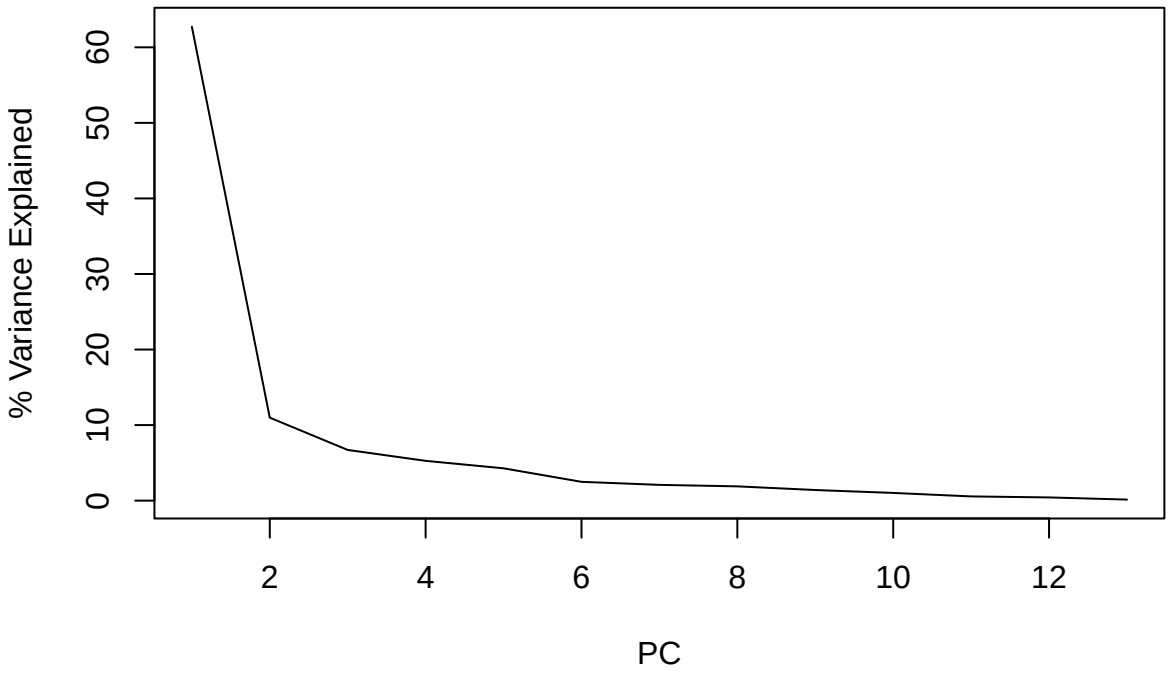




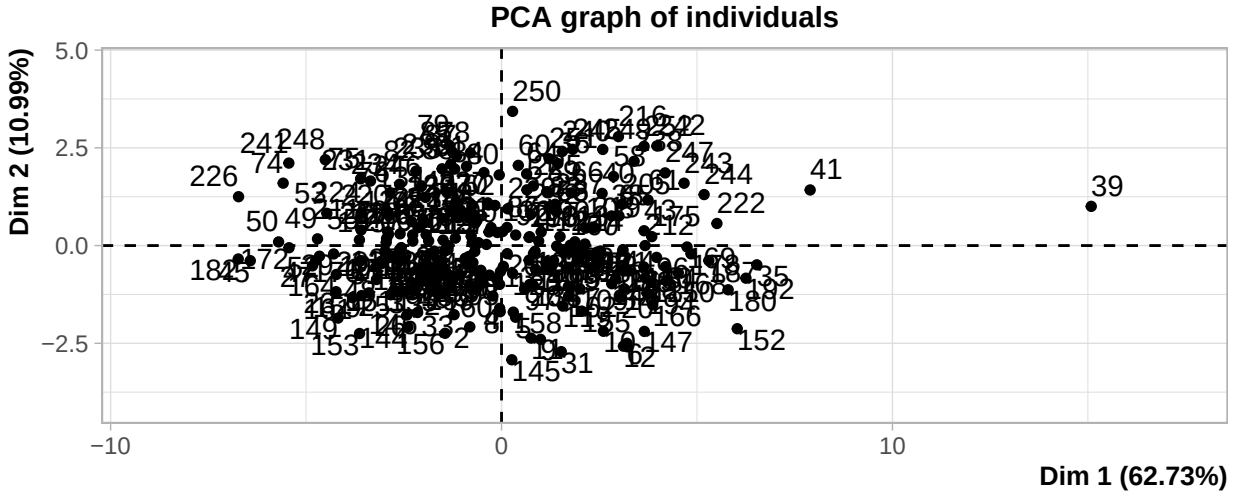

```
round(t(bf_PCA$eig),3)
```

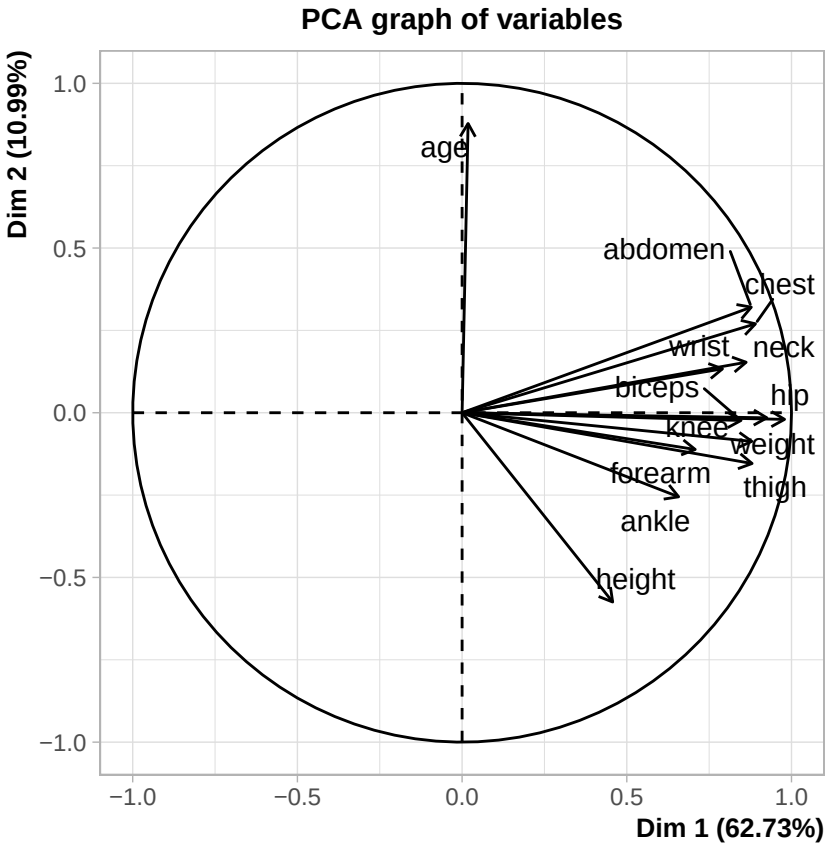
	comp 1	comp 2	comp 3	comp 4	comp 5	comp 6	comp 7	comp 8	comp 9	comp 10	comp 11	comp 12	comp 13
eigenvalue	8.155	1.428	0.873	0.685	0.556	0.324	0.271	0.246	0.183	0.133	0.073	0.055	0.018
percentage of variance	62.732	10.985	6.712	5.267	4.281	2.490	2.088	1.892	1.407	1.022	0.560	0.425	0.140
cumulative percentage of variance	62.732	73.717	80.429	85.696	89.976	92.466	94.554	96.446	97.852	98.874	99.435	99.860	100.000

```
plot(bf_PCA$eig[,2], type = "l", xlab="PC",ylab="% Variance Explained")
```




```
bf_PCA <- PCA(X_only,ncp=3)
```






```
round(bf_PCA$var$cor,3)
```

	Dim.1	Dim.2	Dim.3
age	0.018	0.878	0.402
weight	0.979	-0.020	-0.042
height	0.457	-0.575	0.511
neck	0.862	0.153	0.043
chest	0.891	0.270	-0.129
abdomen	0.878	0.321	-0.177
hip	0.925	-0.015	-0.199
thigh	0.880	-0.154	-0.306
knee	0.880	-0.087	0.087
ankle	0.657	-0.255	0.277
biceps	0.847	-0.023	-0.131
forearm	0.708	-0.112	0.030
wrist	0.790	0.133	0.402

```
head(bf_PCA$ind$coord,5)
```

Dim.1	Dim.2	Dim.3
-2.2915979	-1.142118	-1.6301345
-0.8110527	-2.079607	0.0859599
-2.4786332	-1.173773	-1.9774339
-0.0481556	-1.604541	-0.3578868
0.3591336	-1.833739	-0.9223473

Dim.1	Dim.2	Dim.3

```
bf_PCR <- lm(bf_dat$bodyfat~bf_PCA$ind$coord) # the full model had Multiple R-squared = 0.7486
summary(bf_PCR)
```

```
##
## Call:
## lm(formula = bf_dat$bodyfat ~ bf_PCA$ind$coord)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.1857  -3.8607  -0.0359   3.8626  11.9935
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)              19.1508     0.3343  57.289  < 2e-16 ***
## bf_PCA$ind$coordDim.1     1.7921     0.1171  15.310  < 2e-16 ***
## bf_PCA$ind$coordDim.2     2.7898     0.2797   9.973  < 2e-16 ***
## bf_PCA$ind$coordDim.3    -2.3302     0.3579  -6.511 4.11e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.307 on 248 degrees of freedom
## Multiple R-squared:  0.6027, Adjusted R-squared:  0.5979
## F-statistic: 125.4 on 3 and 248 DF,  p-value: < 2.2e-16
```

```
PC_coef = cbind(PC1_coef = coef(bf_PCR)[2]*bf_PCA$var$coord[,1],
                PC2_coef = coef(bf_PCR)[3]*bf_PCA$var$coord[,2],
                PC3_coef = coef(bf_PCR)[4]*bf_PCA$var$coord[,3])
round(PC_coef, 3)
```

	PC1_coef	PC2_coef	PC3_coef
age	0.033	2.450	-0.936
weight	1.754	-0.056	0.097
height	0.819	-1.603	-1.192
neck	1.546	0.427	-0.100
chest	1.596	0.753	0.301
abdomen	1.574	0.894	0.413
hip	1.658	-0.043	0.464
thigh	1.578	-0.430	0.714
knee	1.576	-0.242	-0.202
ankle	1.178	-0.712	-0.646
biceps	1.517	-0.066	0.305
forearm	1.268	-0.311	-0.070
wrist	1.417	0.372	-0.938

```
round(apply(PC_coef,1,sum), 3)
```

```
##     age  weight  height    neck   chest abdomen     hip   thigh    knee   ankle
##   1.546   1.795  -1.976   1.873   2.650   2.881   2.079   1.861   1.132  -0.180
##  biceps forearm   wrist
##   1.757   0.887   0.851
```

Least Squares for $p \gg n$

This example will use the same PET data from the MMST textbook.

These data were obtained from a calibration study of polyethylene terephthalate (PET) yarns, which are used for textile (e.g., clothing materials) and industrial purposes (e.g., tires, seat belts, and ropes). PET yarns are produced by a process of melt-spinning, whose settings largely determine the final semi-crystalline structure of the yarn (i.e., its physical structure), which, in turn, determines its thermo-mechanical properties. As a result, parameters that characterize the physical structure of PET yarns are important quality parameters for the end use of the yarn.

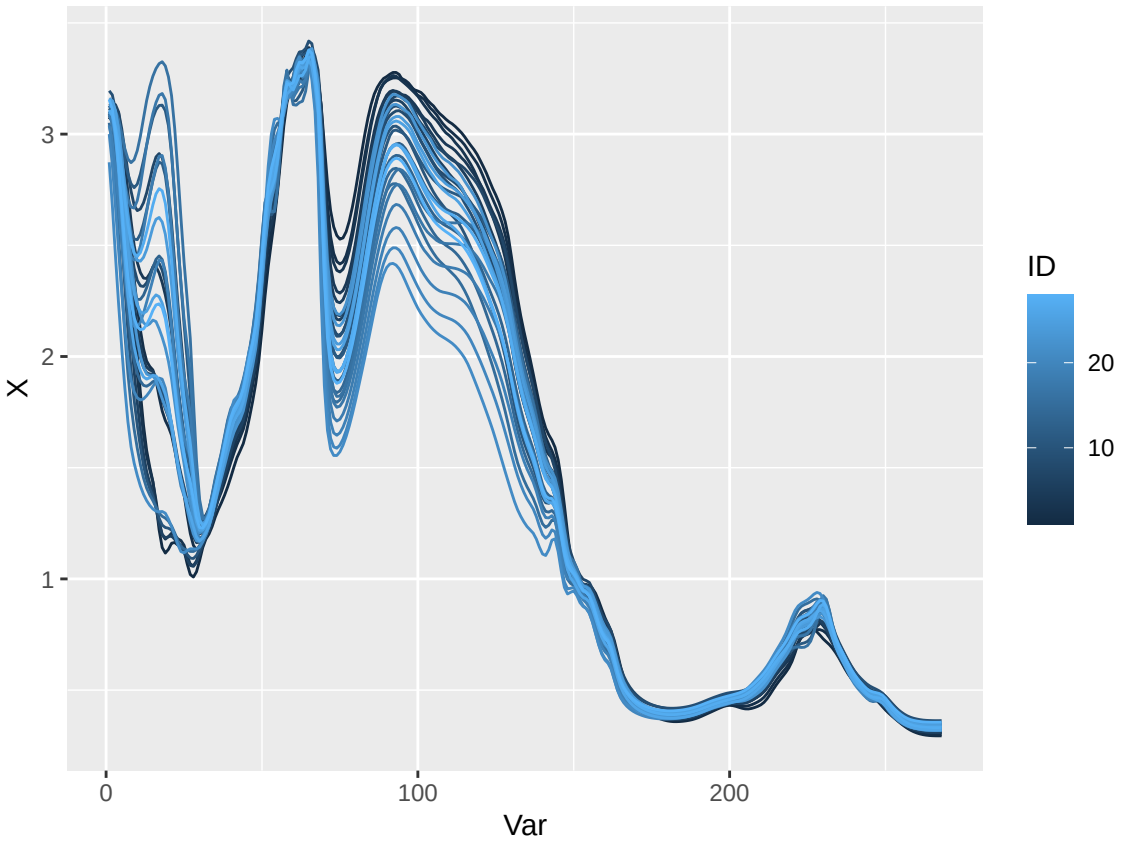
Raman near-infrared (NIR) spectroscopy has recently become an important tool in the pharmaceutical and semiconductor industries for investigating structural information on polymers; in particular, it is used to reveal information on the chemical nature, conformational order, state of the order, and orientation of polymers. Thus, Raman spectra are used to predict the physical structure parameters of polymers.

In this example, we study the relationship between the overall density of a PET yarn to its NIR spectrum. The data consist of a sample of $n = 21$ PET yarns having known mechanical and structural properties. For each PET yarn, the Y-variable is the density (measured in kg/m^3) of the yarn, and the $p = 268$ X-variables (measured at 268 frequencies in the range $598 - 1900cm^{-1}$) are selected from the NIR spectrum of that yarn. This example is quite representative of data sets in the chemometrics literature, in that $p \gg n$.

```
pet_df <- read.csv("PET.csv")
pet_mat <- as.matrix(pet_df)
dim(pet_df)
```

```
## [1]  28 269
```

```
pet_long <- NULL
for(i in 1:28){
  t_dat <- cbind(as.numeric(rep(i,268)), as.numeric(1:268),
                 as.numeric(rep(pet_df[i,269],268)),
                 as.numeric(pet_df[i,1:268]))
  pet_long <- rbind(pet_long,(t_dat))
}
colnames(pet_long) <- c("ID","Var","Y","X")
rownames(pet_long) <- 1:length(pet_long[,1])
pet_long <- data.frame(pet_long)
ggplot(pet_long,aes(y=X,x=Var,group=ID,color=ID)) + geom_line()
```



Now we'll try to fit a simple linear model to the data

```
pet_mod <- lm(pet_mat[,269]~ pet_mat[,1:30])
summary(pet_mod)
```

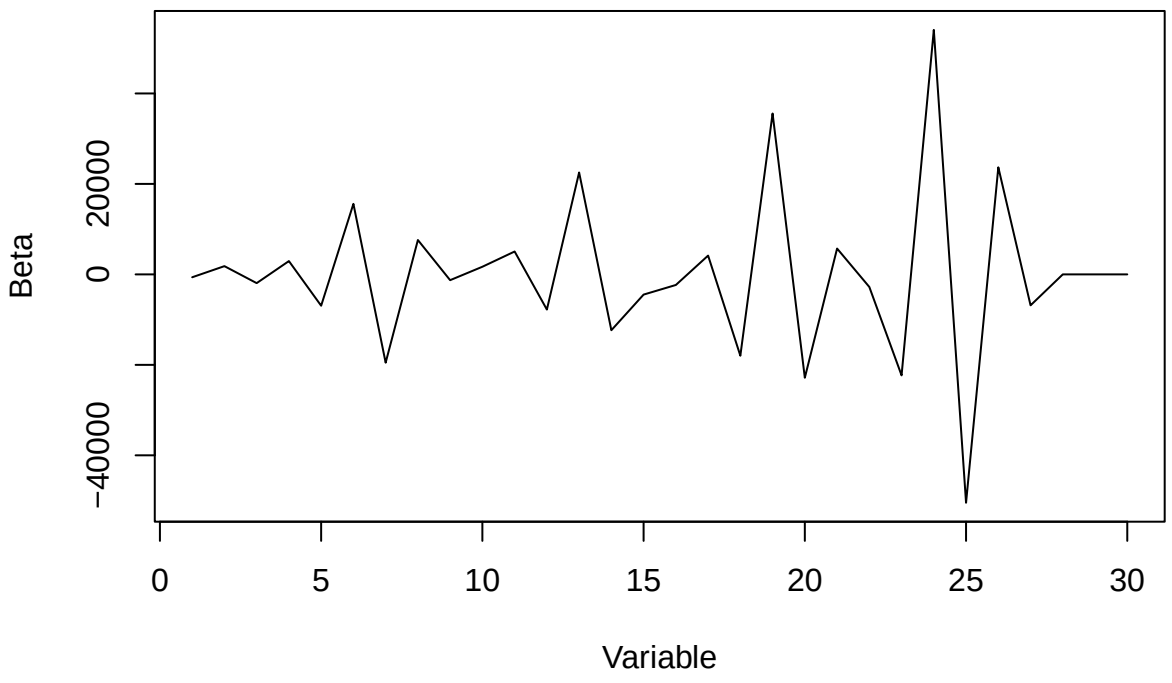
```
##
## Call:
## lm(formula = pet_mat[, 269] ~ pet_mat[, 1:30])
##
## Residuals:
## ALL 28 residuals are 0: no residual degrees of freedom!
##
## Coefficients: (3 not defined because of singularities)
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -859.7        NaN     NaN      NaN
## pet_mat[, 1:30]X.1    -634.7        NaN     NaN      NaN
## pet_mat[, 1:30]X.2    1823.8        NaN     NaN      NaN
## pet_mat[, 1:30]X.3   -1934.0        NaN     NaN      NaN
## pet_mat[, 1:30]X.4    2943.4        NaN     NaN      NaN
## pet_mat[, 1:30]X.5   -6923.8        NaN     NaN      NaN
## pet_mat[, 1:30]X.6   15592.6        NaN     NaN      NaN
## pet_mat[, 1:30]X.7  -19517.0        NaN     NaN      NaN
## pet_mat[, 1:30]X.8    7601.3        NaN     NaN      NaN
## pet_mat[, 1:30]X.9   -1279.5        NaN     NaN      NaN
## pet_mat[, 1:30]X.10   1685.3        NaN     NaN      NaN
## pet_mat[, 1:30]X.11   5057.9        NaN     NaN      NaN
## pet_mat[, 1:30]X.12  -7777.7        NaN     NaN      NaN
## pet_mat[, 1:30]X.13  22522.6        NaN     NaN      NaN
```

```
## pet_mat[, 1:30]X.14 -12342.9        NaN     NaN      NaN
## pet_mat[, 1:30]X.15  -4469.0        NaN     NaN      NaN
## pet_mat[, 1:30]X.16  -2338.2        NaN     NaN      NaN
## pet_mat[, 1:30]X.17   4153.6        NaN     NaN      NaN
## pet_mat[, 1:30]X.18 -17982.1        NaN     NaN      NaN
## pet_mat[, 1:30]X.19  35565.4        NaN     NaN      NaN
## pet_mat[, 1:30]X.20 -22850.4        NaN     NaN      NaN
## pet_mat[, 1:30]X.21   5711.8        NaN     NaN      NaN
## pet_mat[, 1:30]X.22  -2781.1        NaN     NaN      NaN
## pet_mat[, 1:30]X.23 -22323.0        NaN     NaN      NaN
## pet_mat[, 1:30]X.24  54050.3        NaN     NaN      NaN
## pet_mat[, 1:30]X.25 -50489.5        NaN     NaN      NaN
## pet_mat[, 1:30]X.26  23668.7        NaN     NaN      NaN
## pet_mat[, 1:30]X.27  -6842.9        NaN     NaN      NaN
## pet_mat[, 1:30]X.28       NA         NA      NA       NA
## pet_mat[, 1:30]X.29       NA         NA      NA       NA
## pet_mat[, 1:30]X.30       NA         NA      NA       NA
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 27 and 0 DF,  p-value: NA
```

```
pet_mod$residuals
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
##  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 27 28
##  0  0
```

```
coef_vals <- coef(pet_mod)
coef_vals[is.na(coef_vals)] <- 0
plot(coef_vals[-1],type = "l", xlab = "Variable", ylab = "Beta")
```



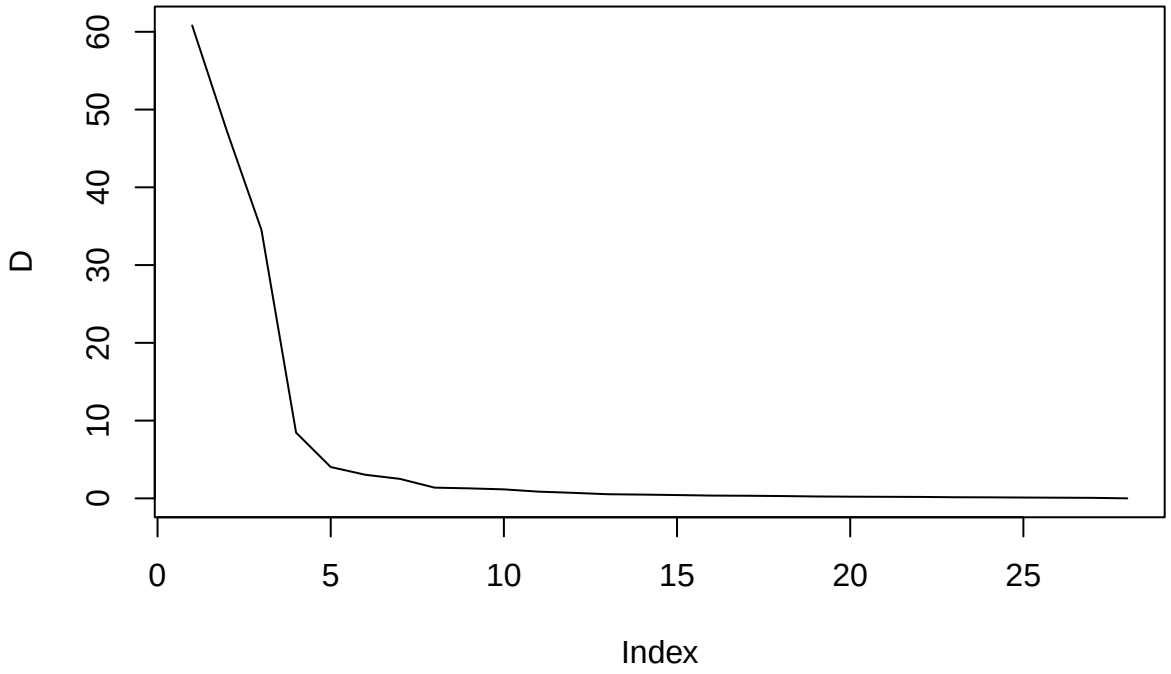
Principal Components Regression

Let's take a look at a PCA of the pet data:

```
##First let's scale.
pet_mat <- scale(pet_mat,scale = TRUE)
apply(pet_mat,2,mean)[1:10]
```

```
##           X.1           X.2           X.3           X.4           X.5
##  2.729587e-15 -1.752071e-15  1.442758e-15  1.093071e-15  1.385862e-16
##           X.6           X.7           X.8           X.9          X.10
##  4.989653e-16 -6.625211e-16 -1.119874e-16  3.968930e-16 -1.923016e-16
```

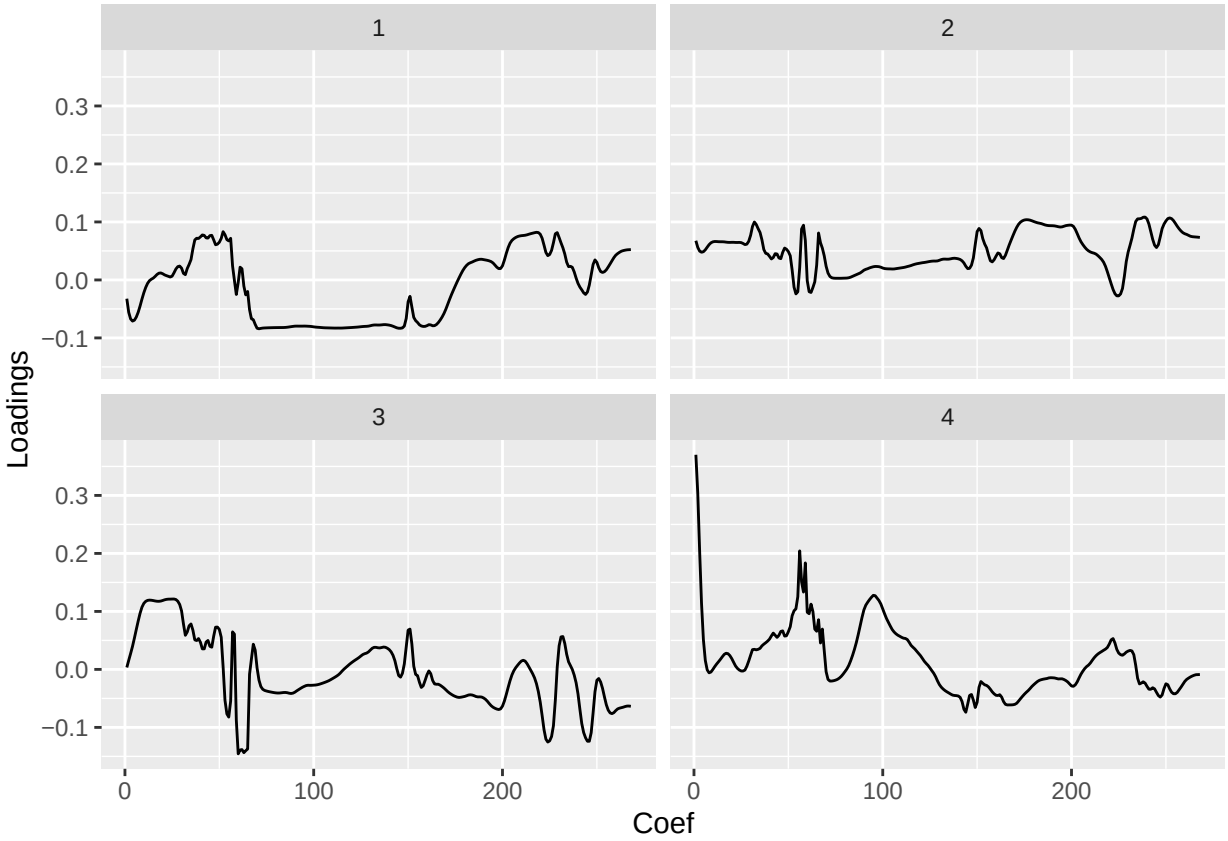
```
#PCA - take SVD to get solution
svdd = svd(pet_mat[,-269])
V = svdd$v #PC loadings
D = svdd$d
Z = pet_mat[,-269] %*% V #PC "scores"

plot(D, type = "l")
```




```
# Change V from wide to long
PCA_long <- data.frame(PC = 1:28, V=t(V)) %>%
  pivot_longer( !PC,
                names_to = "Coef",
                names_prefix = "V.",
                values_to = "Loadings")
PCA_long <- PCA_long %>%
  mutate(Coef = as.numeric(PCA_long$Coef)) %>%
  filter(PC < 5)
head(PCA_long)
```

PC	Coef	Loadings
1	1	-0.0323294
1	2	-0.0557750
1	3	-0.0672177
1	4	-0.0709571
1	5	-0.0694235
1	6	-0.0641731

```
ggplot(PCA_long,aes(y=Loadings,x=Coef)) +
  geom_line()  + facet_wrap(~PC)
```



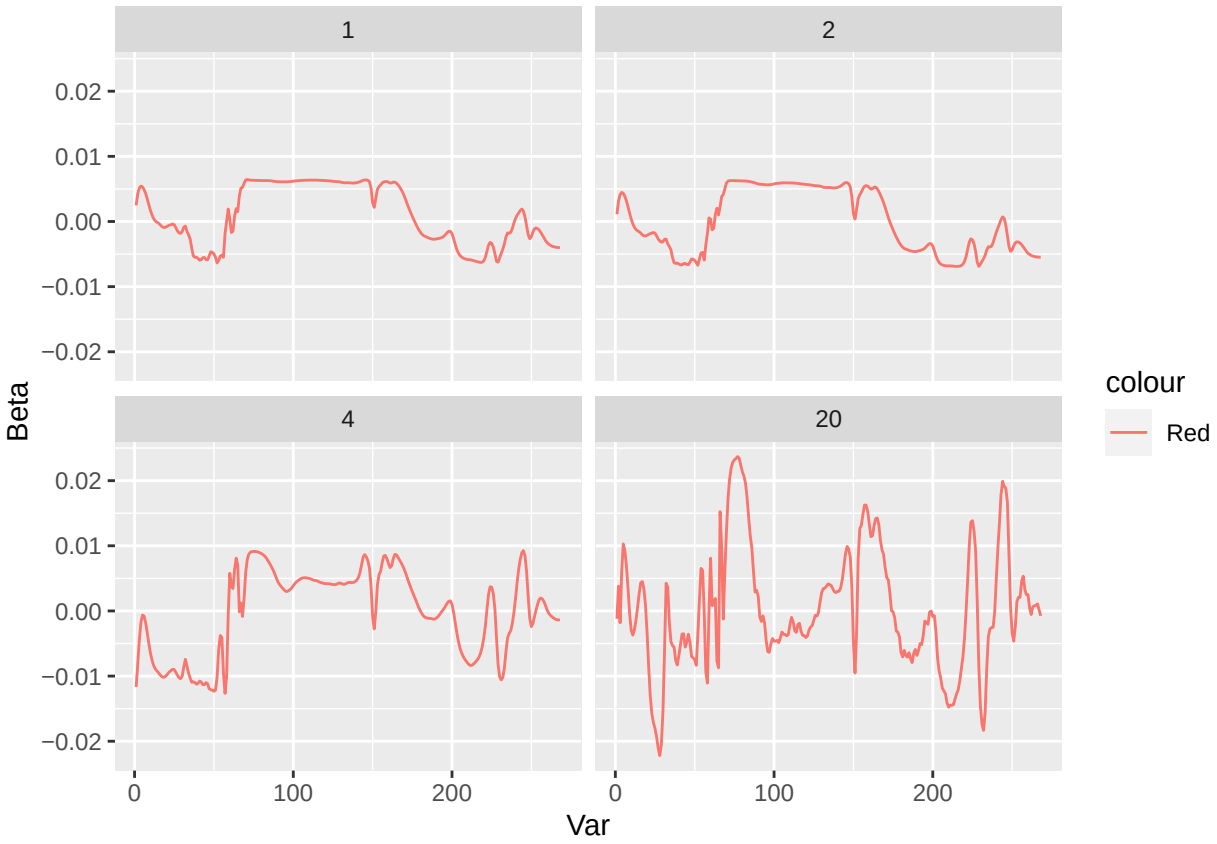
Now lets fit a principal components regression with different values of k.

```
`?`(mvr)
```

```
Partial Least Squares and Principal Component Regression

Description:

     Functions to perform partial least squares regression (PLSR),
     canonical powered partial least squares (CPPLS) or principal
     component regression (PCR), with a formula interface.
     Cross-validation can be used.  Prediction, model extraction, plot,
     print and summary methods exist.

Usage:
```

```
mvr(
  formula,
  ncomp,
  Y.add,
  data,
  subset,
  na.action,
  method = pls.options()$mvralg,
  scale = FALSE,
  center = TRUE,
  validation = c("none", "CV", "LOO"),
  model = TRUE,
  x = FALSE,
  y = FALSE,
  ...
)

plsr(..., method = pls.options()$plsralg)

pcr(..., method = pls.options()$pcralg)

cppls(..., Y.add, weights, method = pls.options()$cpplsalg)
```

```
K <- c(1, 2, 4, 20)
coef_mat_PCR <- NULL
for (k in K) {
    pet_pcr <- pcr(pet_mat[, 269] ~ pet_mat[, 1:268], k, method = "svdpc")
    B <- coef(pet_pcr, ncomp = k, intercept = FALSE)
    coef_mat_PCR <- rbind(coef_mat_PCR, cbind(rep(k, length(B)), 1:length(B), as.numeric(B)))
}

colnames(coef_mat_PCR) <- c("K", "Var", "Beta")
rownames(coef_mat_PCR) <- 1:length(coef_mat_PCR[, 1])
coef_long <- data.frame(coef_mat_PCR)
ggplot(coef_long, aes(y = Beta, x = Var, color = "Red")) + geom_line() + facet_wrap(~K)
```



Ridge Regression

To fit ridge and lasso regresssion models we'll use the `glmnet` function.

```
`?`(glmnet)
```

```
fit a GLM with lasso or elasticnet regularization

Description:

     Fit a generalized linear model via penalized maximum likelihood.
     The regularization path is computed for the lasso or elasticnet
     penalty at a grid of values for the regularization parameter
     lambda. Can deal with all shapes of data, including very large
     sparse data matrices. Fits linear, logistic and multinomial,
     poisson, and Cox regression models.

Usage:

     glmnet(
       x,
       y,
       family = c("gaussian", "binomial", "poisson", "multinomial", "cox", "mgaussian"),
       weights = NULL,
       offset = NULL,
       alpha = 1,
       nlambda = 100,
```

```
    lambda.min.ratio = ifelse(nobs < nvars, 0.01, 1e-04),
    lambda = NULL,
    standardize = TRUE,
    intercept = TRUE,
    thresh = 1e-07,
    dfmax = nvars + 1,
    pmax = min(dfmax * 2 + 20, nvars),
    exclude = NULL,
    penalty.factor = rep(1, nvars),
    lower.limits = -Inf,
    upper.limits = Inf,
    maxit = 1e+05,
    type.gaussian = ifelse(nvars < 500, "covariance", "naive"),
    type.logistic = c("Newton", "modified.Newton"),
    standardize.response = FALSE,
    type.multinomial = c("ungrouped", "grouped"),
    relax = FALSE,
    trace.it = 0,
    ...
)

relax.glmnet(fit, x, ..., maxp = n - 3, path = FALSE, check.args = TRUE)

Details:

The sequence of models implied by 'lambda' is fit by coordinate
descent. For 'family="gaussian"' this is the lasso sequence if
'alpha=1', else it is the elasticnet sequence.

The objective function for '"gaussian"' is

                  1/2 RSS/nobs +lambda*penalty,

and for the other models it is

                  -loglik/nobs +lambda*penalty.

Note also that for '"gaussian"', 'glmnet' standardizes y to have
unit variance (using 1/n rather than 1/(n-1) formula) before
computing its lambda sequence (and then unstandardizes the
resulting coefficients); if you wish to reproduce/compare results
with other software, best to supply a standardized y. The
coefficients for any predictor variables with zero variance are
set to zero for all values of lambda.

  Details on 'family' option:

    From version 4.0 onwards, glmnet supports both the original
    built-in families, as well as _any_ family object as used by
    'stats:glm()'. This opens the door to a wide variety of
    additional models. For example 'family=binomial(link=cloglog)'
    or 'family=negative.binomial(theta=1.5)' (from the MASS
    library). Note that the code runs faster for the built-in
    families.
```

```
      The built in families are specifed via a character string. For
      all families, the object produced is a lasso or elasticnet
      regularization path for fitting the generalized linear
      regression paths, by maximizing the appropriate penalized
      log-likelihood (partial likelihood for the "cox" model).
      Sometimes the sequence is truncated before 'nlambda' values of
      'lambda' have been used, because of instabilities in the inverse
      link functions near a saturated fit.
      'glmnet(...,family="binomial")' fits a traditional logistic
      regression model for the log-odds.
      'glmnet(...,family="multinomial")' fits a symmetric multinomial
      model, where each class is represented by a linear model (on the
      log-scale). The penalties take care of redundancies. A two-class
      '"multinomial"' model will produce the same fit as the
      corresponding '"binomial"' model, except the pair of coefficient
      matrices will be equal in magnitude and opposite in sign, and
      half the '"binomial"' values. Two useful additional families are
      the 'family="mgaussian"' family and the
      'type.multinomial="grouped"' option for multinomial fitting. The
      former allows a multi-response gaussian model to be fit, using a
      "group -lasso" penalty on the coefficients for each variable.
      Tying the responses together like this is called "multi-task"
      learning in some domains. The grouped multinomial allows the
      same penalty for the 'family="multinomial"' model, which is also
      multi-responsed. For both of these the penalty on the
      coefficient vector for variable j is

                (1-alpha)/2||beta_j||_2^2+alpha||beta_j||_2.

      When 'alpha=1' this is a group-lasso penalty, and otherwise it
      mixes with quadratic just like elasticnet. A small detail in the
      Cox model: if death times are tied with censored times, we
      assume the censored times occurred just _before_ the death times
      in computing the Breslow approximation; if users prefer the
      usual convention of _after_, they can add a small number to all
      censoring times to achieve this effect.


Details on response for 'family="cox"':

      For Cox models, the response should preferably be a 'Surv'
      object, created by the 'Surv()' function in 'survival' package.
      For right-censored data, this object should have type "right",
      and for (start, stop] data, it should have type "counting". To
      fit stratified Cox models, strata should be added to the
      response via the 'stratifySurv()' function before passing the
      response to 'glmnet()'. (For backward compatibility,
      right-censored data can also be passed as a two-column matrix
      with columns named 'time' and 'status'. The latter is a binary
      variable, with '1' indicating death, and '0' indicating right
      censored.)
```

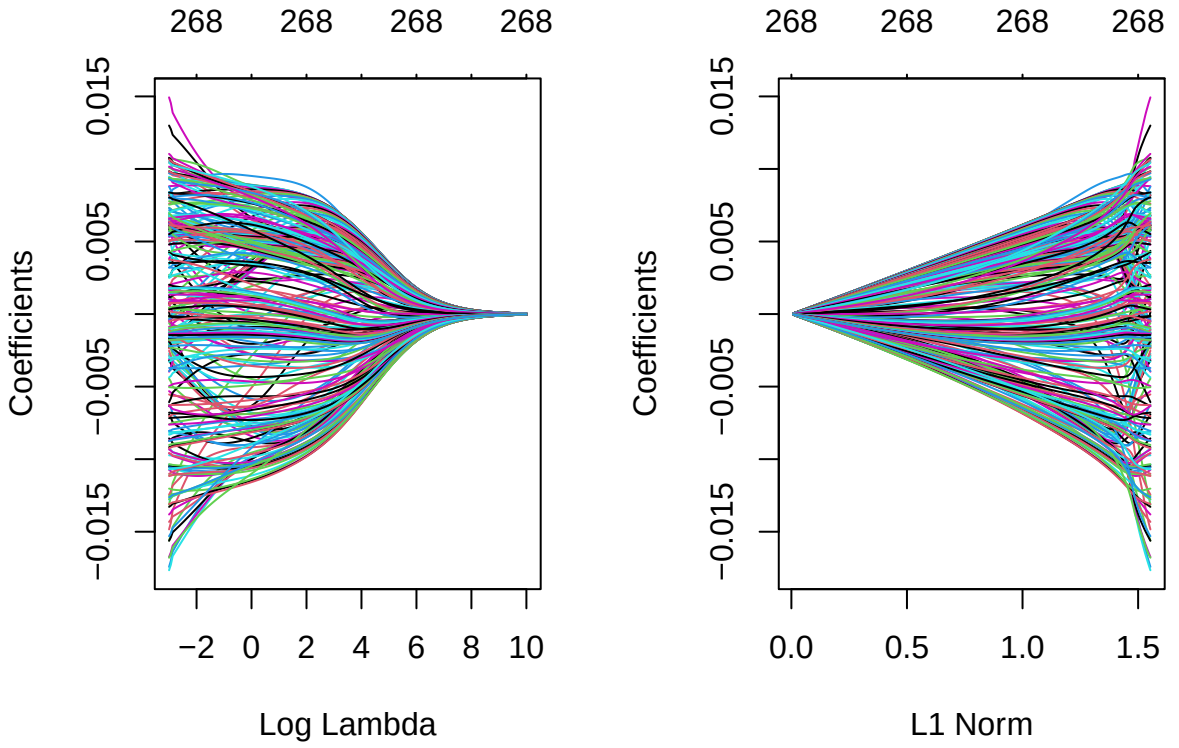
```
  Details on 'relax' option:

       If 'relax=TRUE' a duplicate sequence of models is produced,
       where each active set in the elastic-net path is refit without
       regularization. The result of this is a matching '"glmnet"'
       object which is stored on the original object in a component
       named '"relaxed"', and is part of the glmnet output. Generally
       users will not call 'relax.glmnet' directly, unless the original
       'glmnet' object took a long time to fit. But if they do, they
       must supply the fit, and all the original arguments used to
       create that fit. They can limit the length of the relaxed path
       via 'maxp'.
```

Now lets use `glmnet` to fit ridge regression (i.e., use `alpha=0`):

```
# Values of Lambda
lambda <- exp( c(-5, -3, -1, 1, 4, 7))
coef_mat <- NULL
for(l in lambda){
  lm_mod <- glmnet( pet_mat[,1:268], pet_mat[,269], family = "gaussian",
                    alpha = 0, lambda = l)
  B <- coef.glmnet(lm_mod)
  coef_mat <- rbind( coef_mat,
                     cbind(rep(log(l),length(B)),1:length(B),
                           as.numeric(B)) )
}

colnames(coef_mat) <- c("Lambda","Var","Beta")
rownames(coef_mat) <- 1:length(coef_mat[,1])
coef_long <- data.frame(coef_mat)
ggplot(coef_long,aes(y=Beta,x=Var,color="Red")) +
  geom_line() + facet_wrap(~Lambda, scales = "fixed")
```

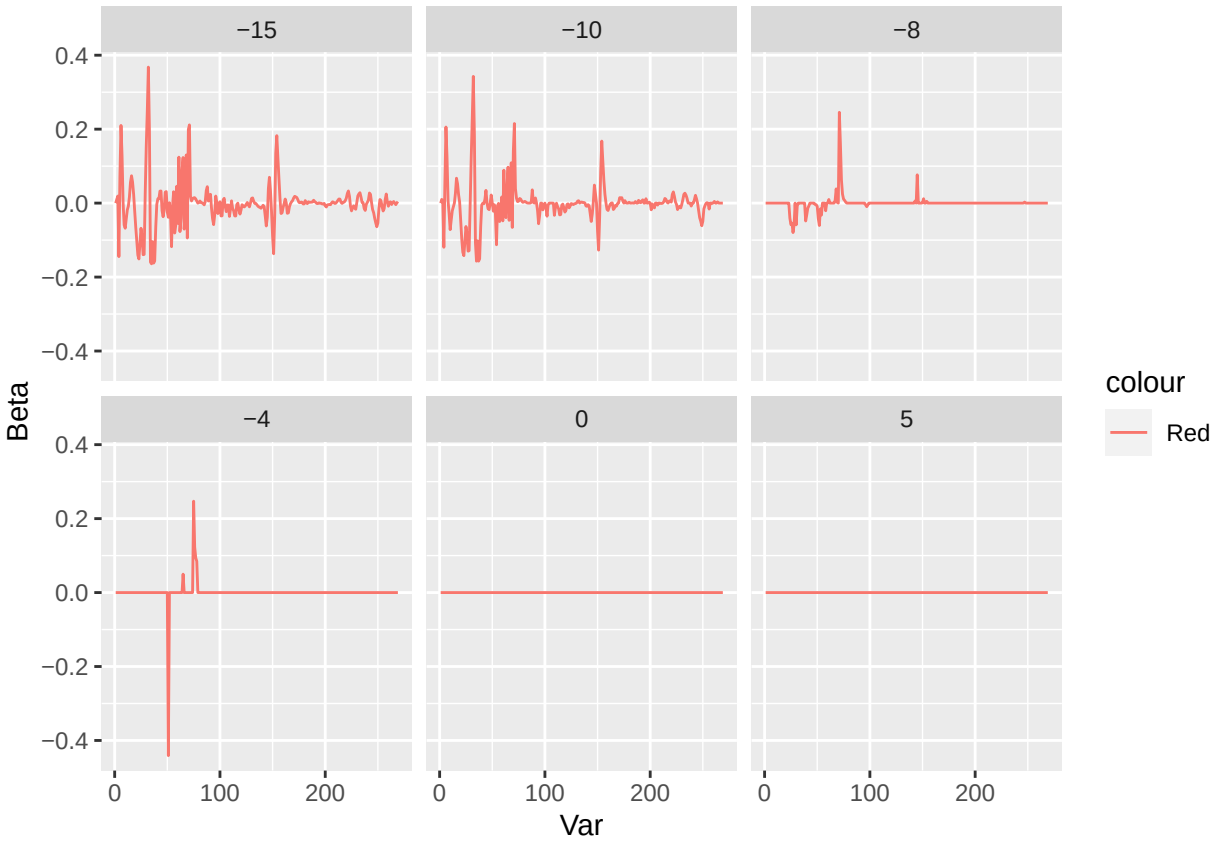
We can fit the ridge model for many different values of λ in one shot.

```
ridge_mod <- glmnet(pet_mat[, 1:268], pet_mat[, 269], family = "gaussian", alpha = 0,
    lambda = exp(seq(-3, 10, length.out = 200)))
par(mfrow = c(1, 2))
plot(ridge_mod, xvar = "lambda", label = TRUE)
plot(ridge_mod, xvar = "norm", label = TRUE)
```



LASSO

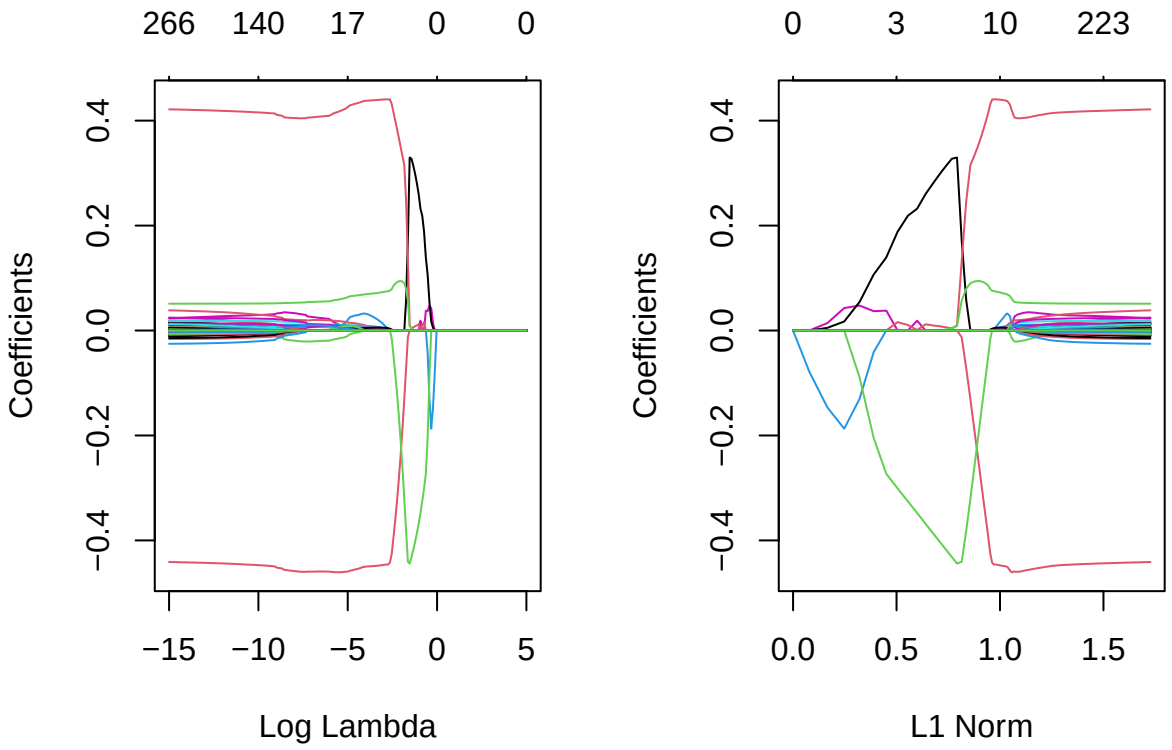
```
# Values of Lambda
lambda <- exp(c(-15, -10, -8, -4, 0, 5))
coef_mat <- NULL
for (l in lambda) {
    lm_mod <- glmnet(pet_mat[, 1:268], pet_mat[, 269], family = "gaussian", alpha = 1,
        lambda = l)
    B <- coef.glmnet(lm_mod)
    coef_mat <- rbind(coef_mat, cbind(rep(log(l), length(B)), 1:length(B), as.numeric(B)))
}

colnames(coef_mat) <- c("Lambda", "Var", "Beta")
rownames(coef_mat) <- 1:length(coef_mat[, 1])
coef_long <- data.frame(coef_mat)
ggplot(coef_long, aes(y = Beta, x = Var, color = "Red")) + geom_line() + facet_wrap(~Lambda,
    scales = "fixed")
```



We can fit the lasso model for many different values of λ in one shot.

```
lasso_mod <- glmnet(pet_mat[, 1:268], pet_mat[, 269], family = "gaussian", alpha = 1,
    lambda = exp(seq(-15, 5, length.out = 200)))
par(mfrow = c(1, 2))
plot(lasso_mod, xvar = "lambda", label = TRUE)
plot(lasso_mod, xvar = "norm", label = TRUE)
```

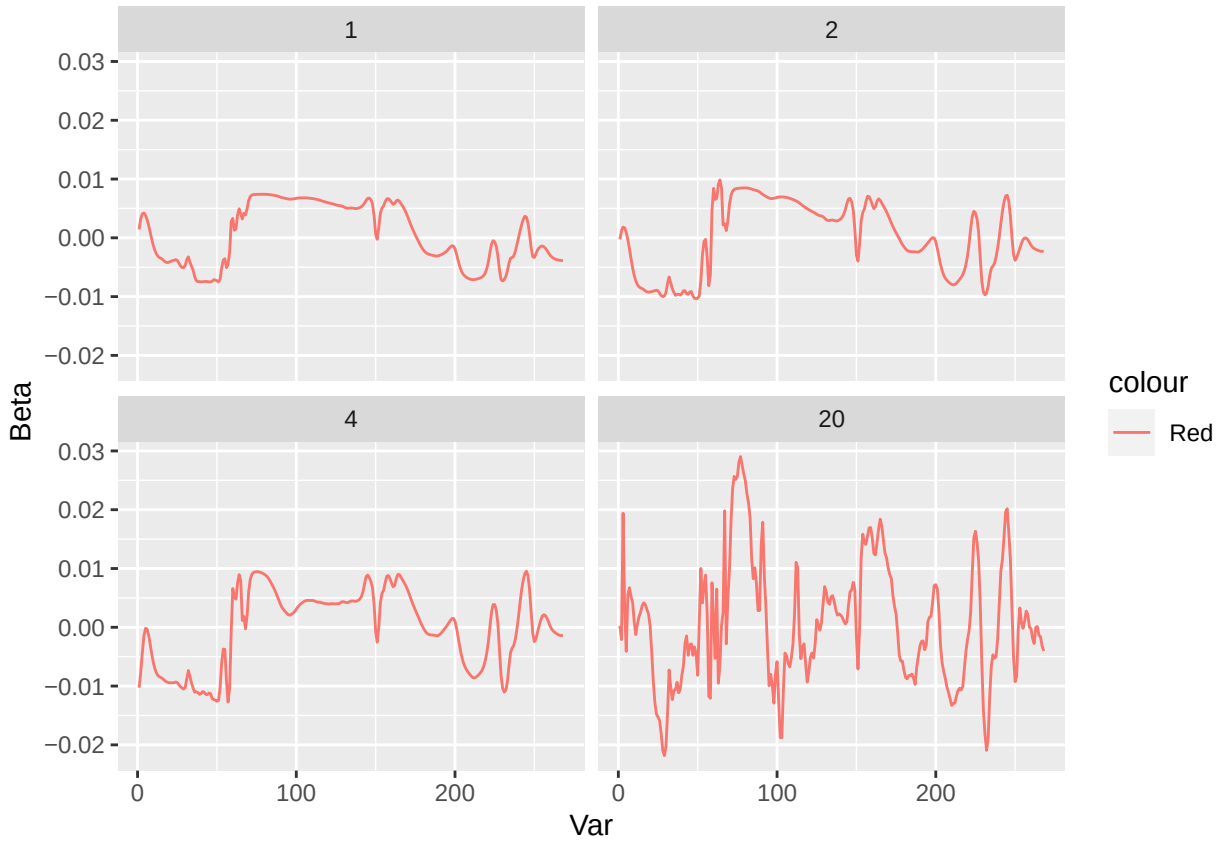
```
Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
collapsing to unique 'x' values
```



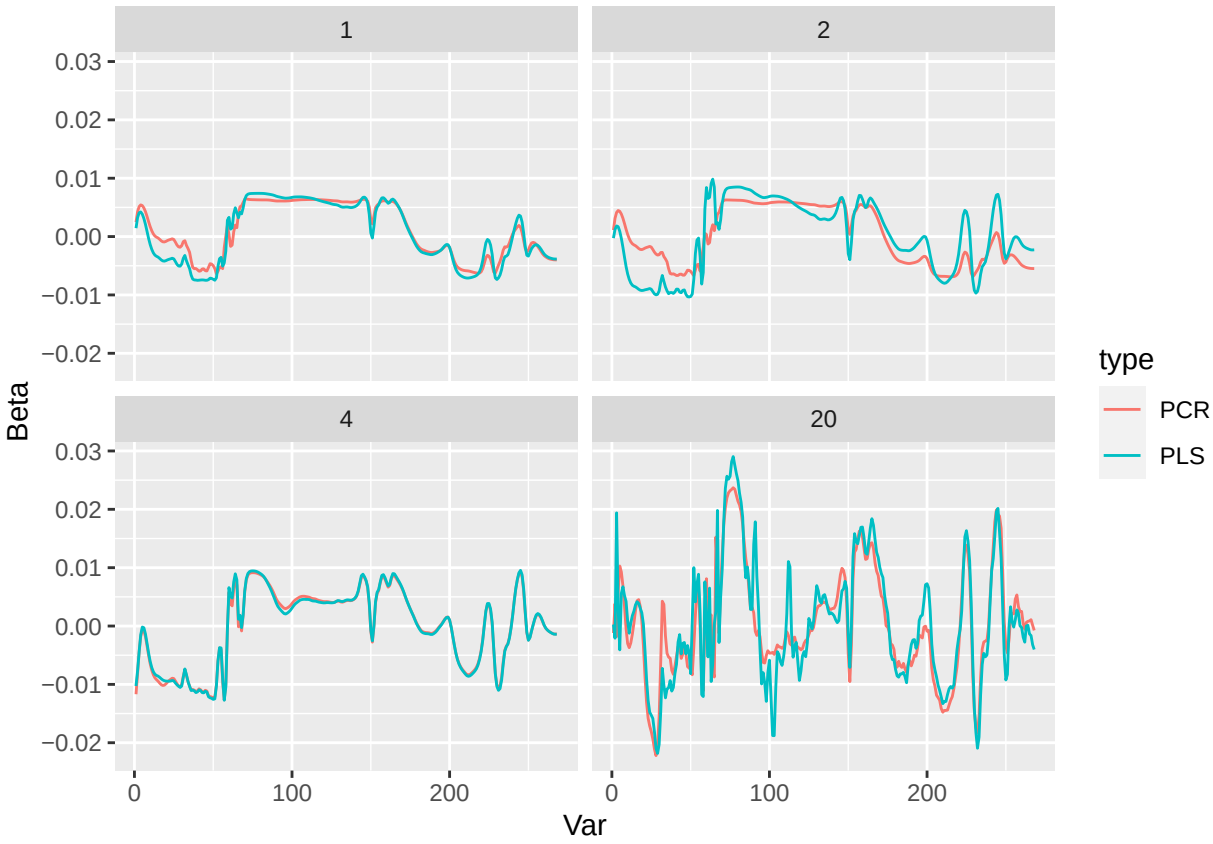
Partial Least Squares

Now we'll do this with partial least squares:

```
K <- c(1, 2, 4, 20)
coef_mat_PLS <- NULL
for (k in K) {
    pet_pcr <- plsr(pet_mat[, 269] ~ pet_mat[, 1:268], k)
    B <- coef(pet_pcr, ncomp = k, intercept = FALSE)
    coef_mat_PLS <- rbind(coef_mat_PLS, cbind(rep(k, length(B)), 1:length(B), as.numeric(B)))
}

colnames(coef_mat_PLS) <- c("K", "Var", "Beta")
rownames(coef_mat_PLS) <- 1:length(coef_mat_PLS[, 1])
coef_long_PLS <- data.frame(coef_mat_PLS)
ggplot(coef_long_PLS, aes(y = Beta, x = Var, color = "Red")) + geom_line() + facet_wrap(~K)
```




```
PLS_v_PCR <- rbind(coef_mat_PCR, coef_mat_PLS)
colnames(PLS_v_PCR) <- c("K", "Var", "Beta")
PLS_v_PCR <- data.frame(PLS_v_PCR)
PLS_v_PCR$type <- rep(c("PCR", "PLS"), each = length(coef_mat_PLS[, 1]))
ggplot(PLS_v_PCR, aes(y = Beta, x = Var, color = type)) + geom_line() + facet_wrap(~K)
```


1
2
4
20
0.03
0.02
0.01
0.00
−0.01
−0.02
Beta
0
100
200
Var
type
PCR
PLS