Principal Components

James D. Wilson

In this lab, we will explore how to use two methods:

1) Principal component analysis (PCA)
2) Principal component regression (PCR)

Much of this lab is a replication of Sections 6.7.1, and 10.4 in the "Introduction to Statistical Learning" textbook by James, Witten, Hastie, and Tibshirani (with my own commentary throughout).

Part I: Principal Component Analysis

Recall that principal component analysis (PCA) is an unsupervised method that will be applied to a matrix X whose columns represent variables and rows represent observations.

There are several available functions to use to conduct PCA in R. Perhaps one of the simplest to use is the *prcomp()* function, which comes with the base package in R. Pay close attention to the arguments and output values of *prcomp()* by looking at the help file.

```
?prcomp
```

Example 1: USArrests Data

In this first example, we will be using the *USArrests* data set in R, which carries information about the arrests in all 50 states of the U.S. in 1973.

Let's load and pre-process the data

```
#read a description of the data
?USArrests

#create a variable for the name of each state
states <- row.names(USArrests)

#look at the variables in this data set
names(USArrests)
```

```
## [1] "Murder"   "Assault"  "UrbanPop" "Rape"
```

```
#look at the mean of each variable
apply(USArrests, 2, mean)
```

```
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
```

```
#look at the variance of each variable
apply(USArrests, 2, var)
```

```
##     Murder    Assault   UrbanPop       Rape
##   18.97047 6945.16571  209.51878   87.72916
```

Notably, the variables have vastly different means and variances. Recall that for PCA, we need to first column center our data matrix X. In other words, we need to center each of the variables in X.

It is useful to further standardize the variables to ensure they each have variance 1. In this way, all of the variables are on the same scale. Thankfully, using the *prcomp()* function, we can take care of this automatically by setting the argument *scale = TRUE*.

Let's perform PCA on the scaled variables of the *USArrests* data set.

```r
pr.out <- prcomp(USArrests, scale = TRUE)

#look at the names of the results of applying PCA
names(pr.out)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

Output of ***prcomp***

- *sdev*: the standard deviation of X explained by each PC. Recall, the ith value of this is the square root of the ith largest eigenvalue of $n^{-1}X^TX$
- *rotation*: a matrix whose columns contain the PC loadings for each PC
- *center*: the pre-adjusted mean of each variable
- *scale*: the pre-adjusted standard deviation of each variable
- *x*: a matrix whose (i,j)th entry is the jth PC score of observation i. These are the PCs.

Let's look at the PC loadings associated with each PC:

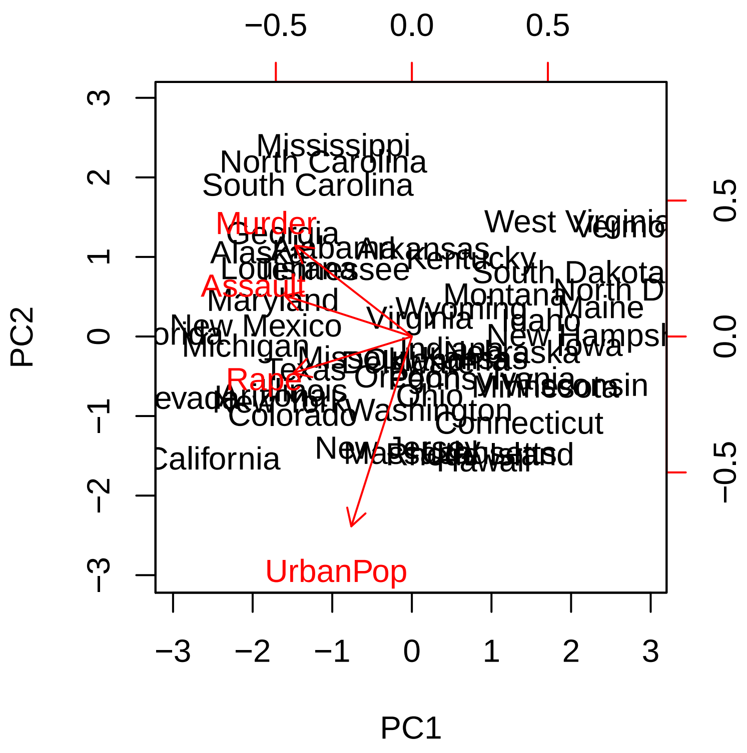
```r
pr.out$rotation
```

```
##                 PC1        PC2        PC3         PC4
## Murder   -0.5358995  0.4181809 -0.3412327  0.64922780
## Assault  -0.5831836  0.1879856 -0.2681484 -0.74340748
## UrbanPop -0.2781909 -0.8728062 -0.3780158  0.13387773
## Rape     -0.5434321 -0.1673186  0.8177779  0.08902432
```

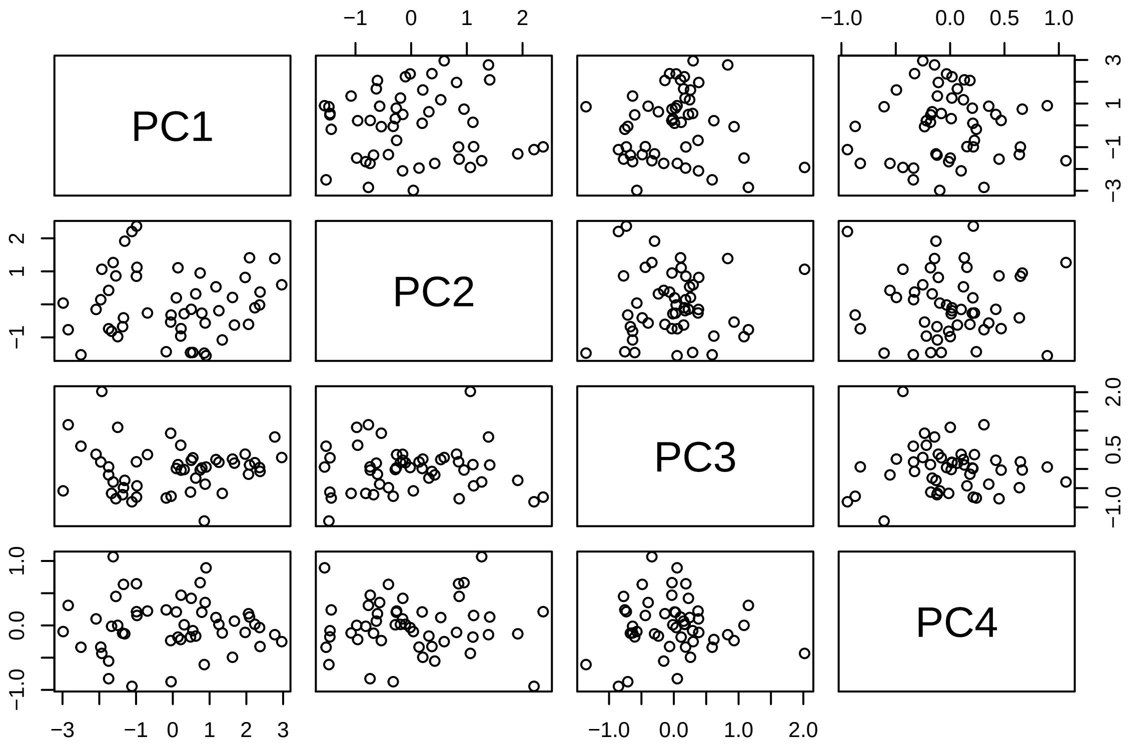
PC Scores plots

Often, the PC scores are used for further exploratory analysis (as done in class). By plotting scores against one another, we may witness clustering of the data that we wouldn't have noticed in the original data set. Let's plot the PC scores.

```r
#plot of PC1 against PC2 (the so-called bi-plot)
biplot(pr.out, scale = 0)
```



```r
#pairwise scatter plot of all pcs
pairs(pr.out$x)
```

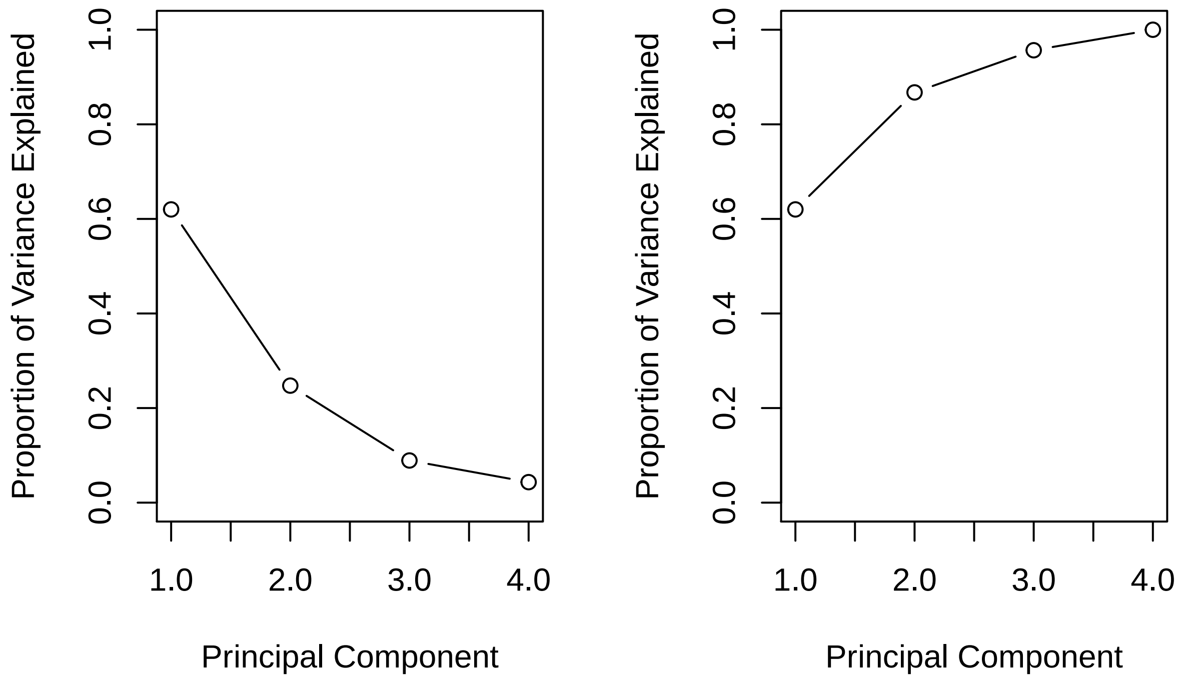


Of course, if we wanted to look more into the clustering of this data, we'd like to add colors according to external features of the data.

Proportion Variation Explained and Scree plots

To determine the number of PCs needed to capture the variability in the data X, we often use the *scree plot*, which display the proportion of variation explained (PVE) of each PC. Let's plot the PVE and the cumulative PVE.

```
pr.var <- pr.out$sdev^2
pve = pr.var / sum(pr.var)
par(mfrow = c(1,2))
plot(pve, xlab = "Principal Component", ylab = "Proportion of Variance Explained", ylim = c(0,1), type 

plot(cumsum(pve), xlab = "Principal Component", ylab = "Proportion of Variance Explained", ylim = c(0,1
```



Example 2: Image Compression

In this example, we will use PCA for compression of images. We will rely upon several R packages to allow us to import .png files. Let's install the needed packages here.

Install Required Image Libraries

```
install.packages("png", repos='http://cran.us.r-project.org')
```

```
## Installing package into '/Users/jdwilson4/Library/R/3.4/library'
## (as 'lib' is unspecified)

##
## The downloaded binary packages are in
##  /var/folders/hm/8gnvskgx0rb1c11fzmz7sgf82j1yqg/T//RtmpLMOFJn/downloaded_packages
```

```
install.packages("grid", repos='http://cran.us.r-project.org')
```

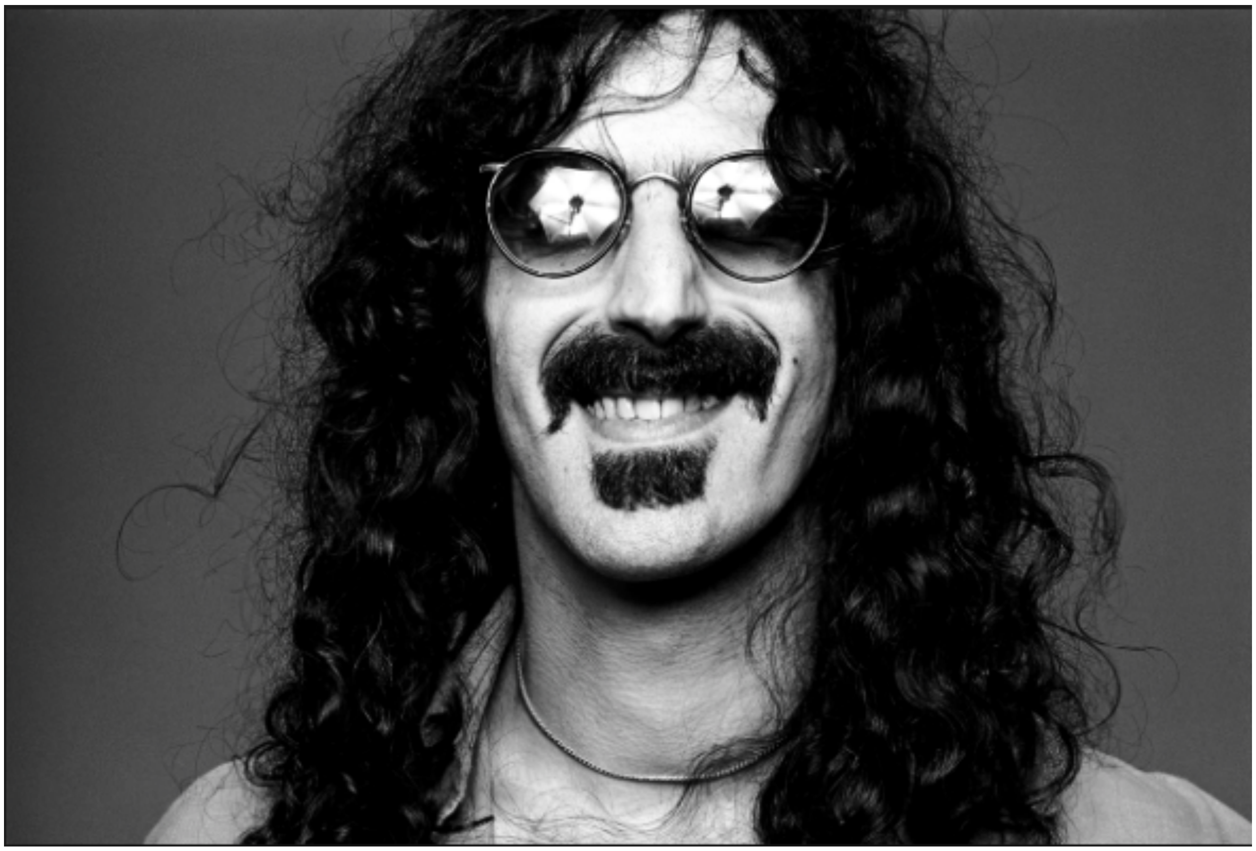
```
## Installing package into '/Users/jdwilson4/Library/R/3.4/library'
## (as 'lib' is unspecified)

## Warning: package 'grid' is not available (for R version 3.4.1)

## Warning: package 'grid' is a base package, and should not be updated
```

```r
library(png, quietly = TRUE)
library(grid, quietly = TRUE)
```

Now, download the image "Frank_Zappa.png" from the Files/Data folder on Canvas. Identify the location of this image and use this as the *directory* variable below. Importantly, your directory and mine will be different.

Let's load the image, convert it to an array and plot the image in R. This can be done with the following code

```r
directory <- "/Users/jdwilson4/Dropbox/Teaching/USF/Intro_to_Machine_Learning/github/Spring_2018/Data/F

#note - you'll need to download the data from Github and set the directory
#to be the location of the image to get this to work

#import the image as an array where each dimension of the array represents a different level of the RGB
Frank <- readPNG(directory)

#convert the image to a matrix whose entries are strings that represent color identifiers. This can be
Frank_plot <- as.raster(Frank)

#plot the image
grid.raster(Frank_plot)
```



Now, let's perform PCA on the image using the pixel intensities of the Red color. In other words, we will focus only on the *R* part of *RGB* in the *Frank* array. Namely, we will only analyze the first dimension of *Frank*. Note that you could simply take an average of the intensities here if you'd like.

```r
Frank_red_only <- Frank[, , 1]
pr.out <- prcomp(Frank_red_only, scale = TRUE)
```

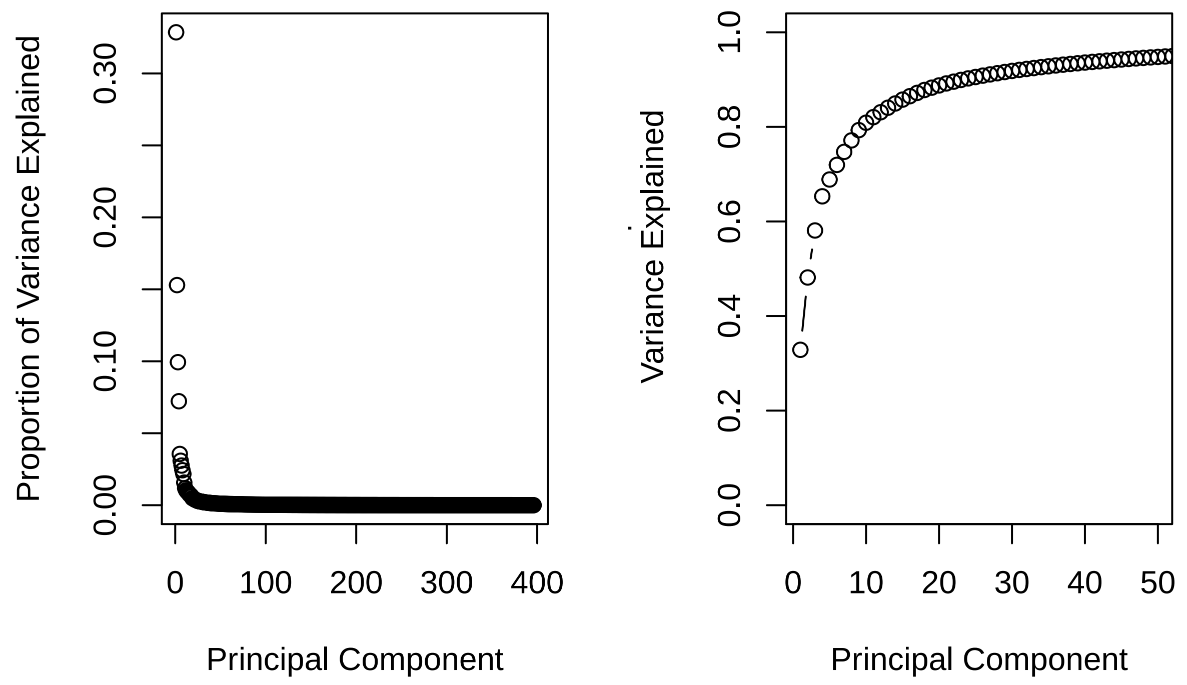
```
dim(pr.out$rotation)
```

```
## [1] 588 396
```

Note that there are 396 total PCs (columns) since the number of PCs will be less than or equal to n and p.

Scree Plots of the image compression

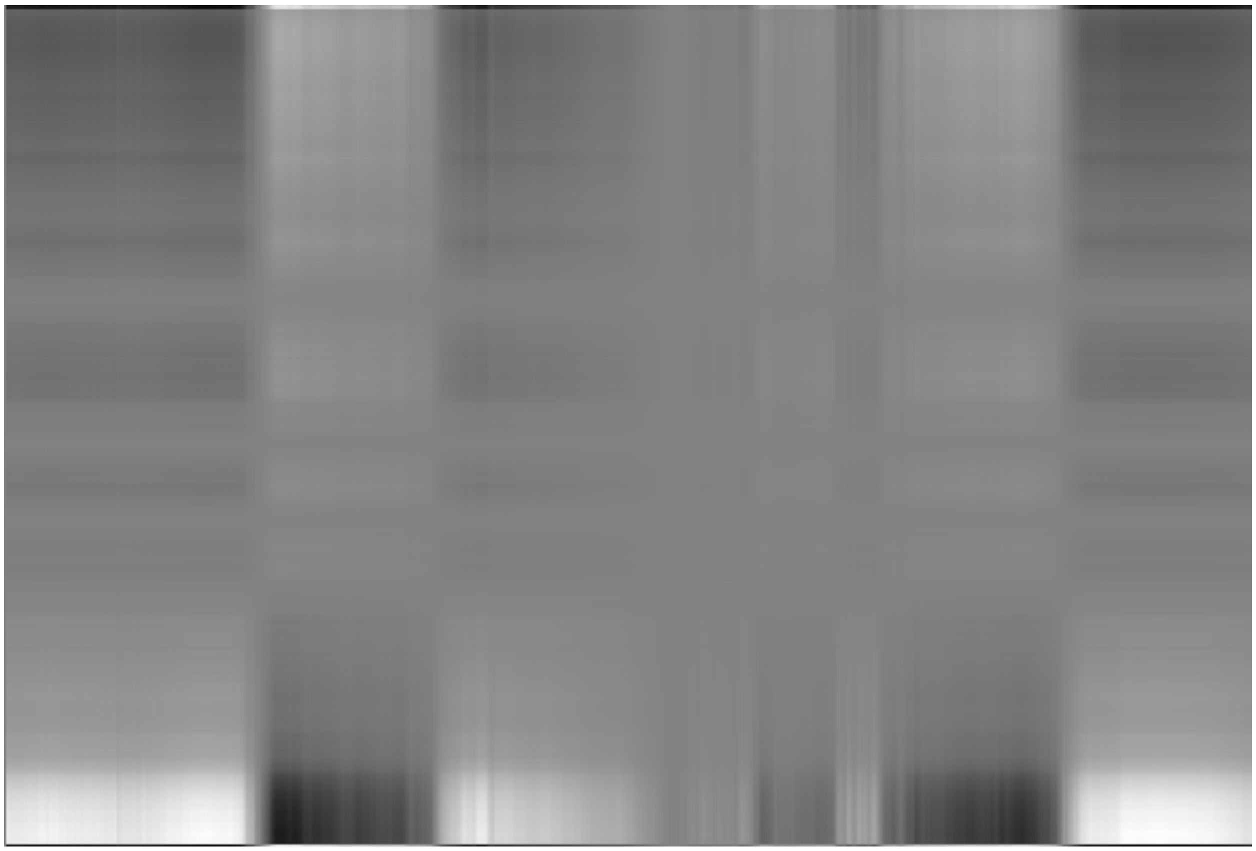
```
pr.var <- pr.out$sdev^2
pve <- pr.var / sum(pr.var)
par(mfrow = c(1,2))
plot(pve, xlab = "Principal Component", ylab = "Proportion of Variance Explained")
plot(cumsum(pve), xlab = "Principal Component", ylab = "Cumulative Proportion of
     Variance Explained", ylim = c(0, 1), type = "b", xlim = c(1, 50))
```



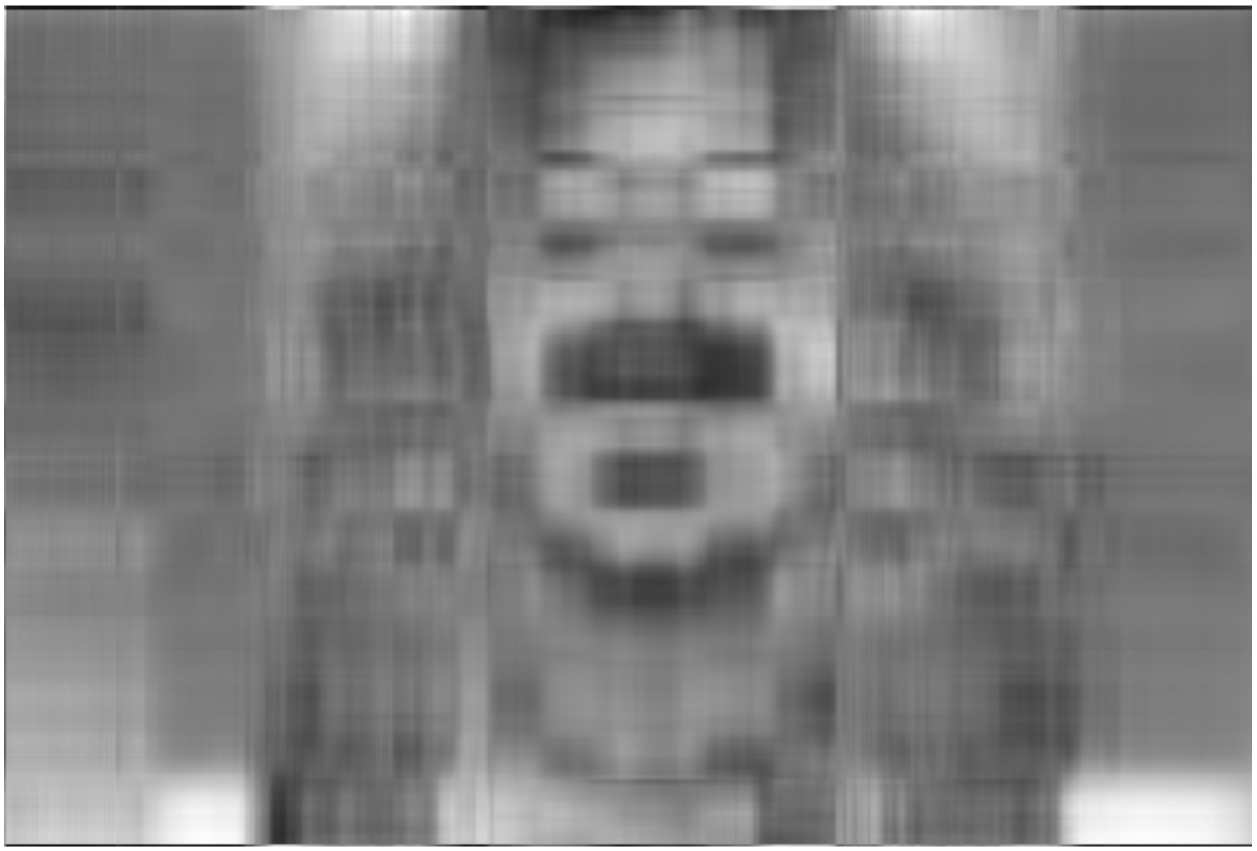
Projecting the Image to lower dimensional spaces Let's now take projections of this image onto lower dimensional spaces by projecting the original image onto the first d PC loading vectors. We'll do this for $d = 1, 5, 15, 50, and 200$ and view the output.

```
W <- pr.out$rotation #the loading matrix
pc.image <- list()
num.pcs <- c(1, 5, 15, 50, 200)

#scale the original image
Image <- scale(Frank_red_only)
for(j in 1:length(num.pcs)){
  u.proj <- W
  #we will only use the first num.pcs PC loadings so set the remaining to 0
  u.proj[, (num.pcs[j] + 1) : 396] <- 0

  #Make the projection
  projection <- (Image%*%u.proj)%*%t(u.proj)

  #to draw an image, values need to be between 0 and 1
```

```r
  scaled <- (projection - min(as.numeric(projection)))
  scaled <- scaled / max(as.numeric(scaled))
  pc.image[[j]] <- as.raster(scaled)
}

#plot each of the images
  grid.raster(pc.image[[1]])
```

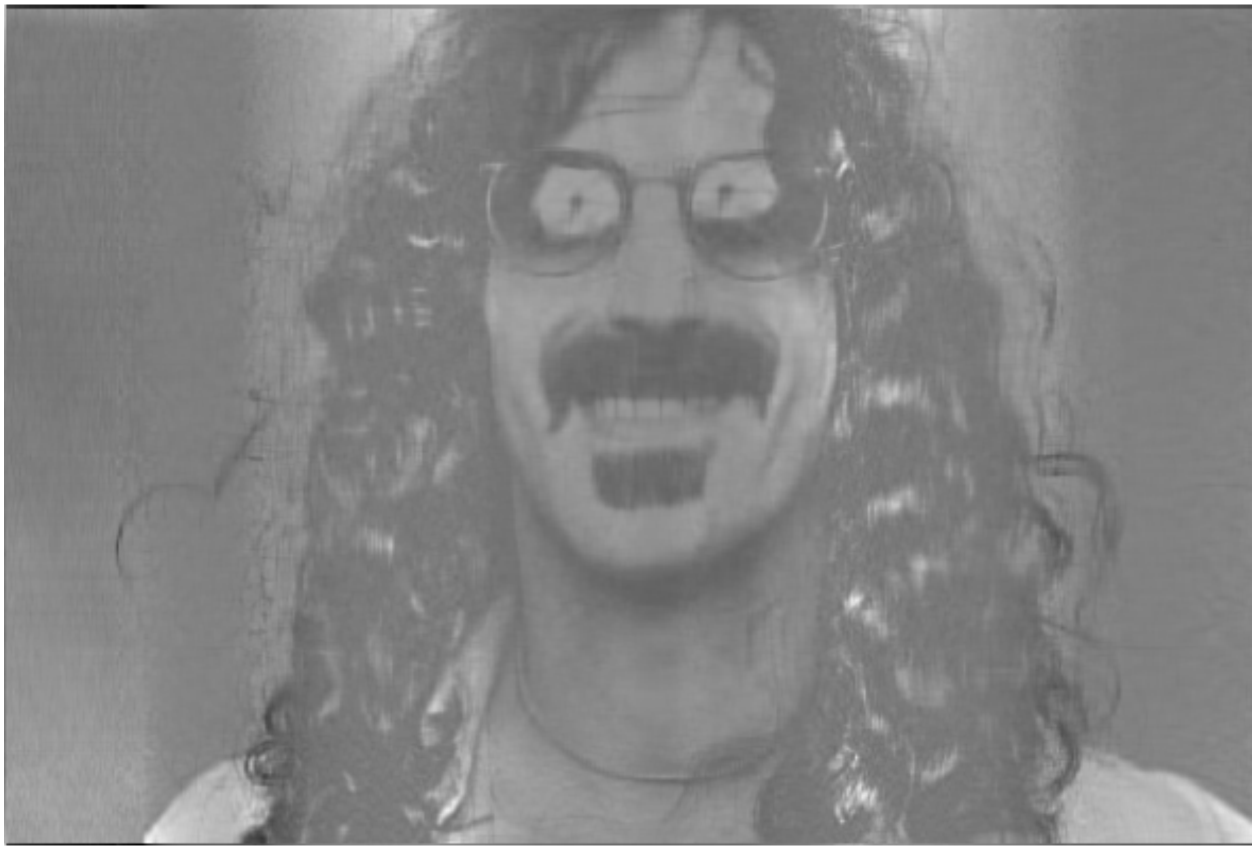


```r
  grid.raster(pc.image[[2]])
```



```
grid.raster(pc.image[[3]])
```

```r
grid.raster(pc.image[[4]])
```



```r
grid.raster(pc.image[[5]])
```

Part II: Principal Component Regression

Now, we will investigate how to fit a principal component regression (PCR). This requires the library *pls*. Let's install the needed library first.

```
install.packages("pls", repos='http://cran.us.r-project.org')
```

```
## Installing package into '/Users/jdwilson4/Library/R/3.4/library'
## (as 'lib' is unspecified)
```

```
##
## The downloaded binary packages are in
##  /var/folders/hm/8gnvskgx0rb1c11fzmz7sgf82j1yqg/T//RtmpLMOFJn/downloaded_packages
```

```
library(pls, quietly = TRUE)
```

```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

The primary function that we will be using is *pcr()*.

```
?pcr
```

Example: Hitters Data

Again, we will use the *Hitters* data from the *ISLR* package to implement PCR. Let's first remove missing values from this data set.

```r
library(ISLR, quietly = TRUE)
```

```
## Warning: package 'ISLR' was built under R version 3.4.2
```

```r
Hitters <- na.omit(Hitters)
```

Now, we will run PCR to fit a regression of the player *Salary* against all other variables in the *Hitters* data set. We will use cross-validation to determine the number of PCs to be used for the model fit. Again, we'd like to standardize the variabls in the data set, so we set *scale = TRUE*.

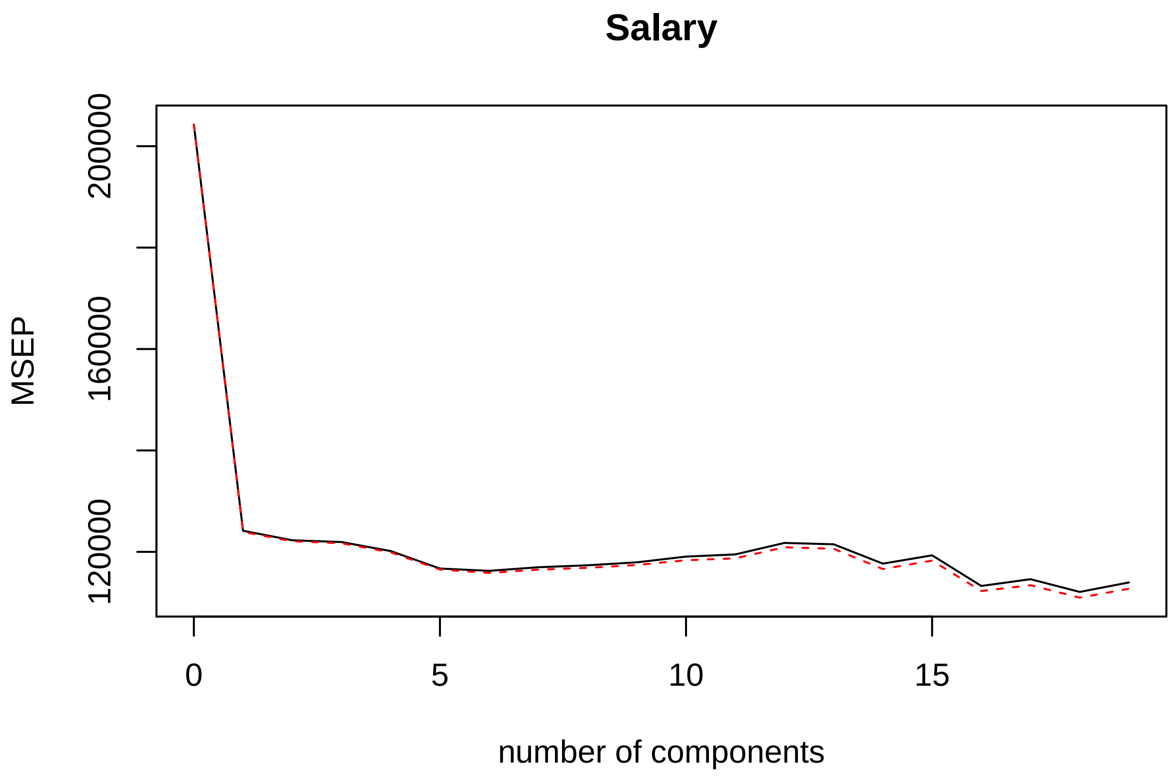
```r
#first set a seed for reproducability
set.seed(1)

#fit a PCR
pcr.fit <- pcr(Salary ~ ., data = Hitters, scale = TRUE, validation = "CV")

#look at a summary of the fit
summary(pcr.fit)
```

```
## Data:    X dimension: 263 19
##  Y dimension: 263 1
## Fit method: svdpc
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV             452    352.4    349.7    349.2    346.6    341.6    341.0
## adjCV          452    352.0    349.4    348.9    346.3    341.3    340.4
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       342.0    342.6    343.4     345.1     345.7     348.9     348.6
## adjCV    341.3    341.8    342.6     344.0     344.6     347.7     347.3
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        343.0     345.4     336.6     338.5     334.8     337.6
## adjCV     341.5     343.9     335.1     336.8     333.1     335.7
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X         38.31    60.16    70.84    79.03    84.29    88.63    92.26
## Salary    40.63    41.58    42.17    43.22    44.90    46.48    46.69
##         8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X         94.96    96.28     97.26     97.98     98.65     99.15     99.47
## Salary    46.75    46.86     47.76     47.82     47.85     48.10     50.40
##         15 comps  16 comps  17 comps  18 comps  19 comps
## X          99.75     99.89     99.97     99.99    100.00
## Salary     50.55     53.01     53.85     54.61     54.61
```

Note that the *summary()* function will provide the *root mean squared error*, so these will need to be squared to get the usual MSE.

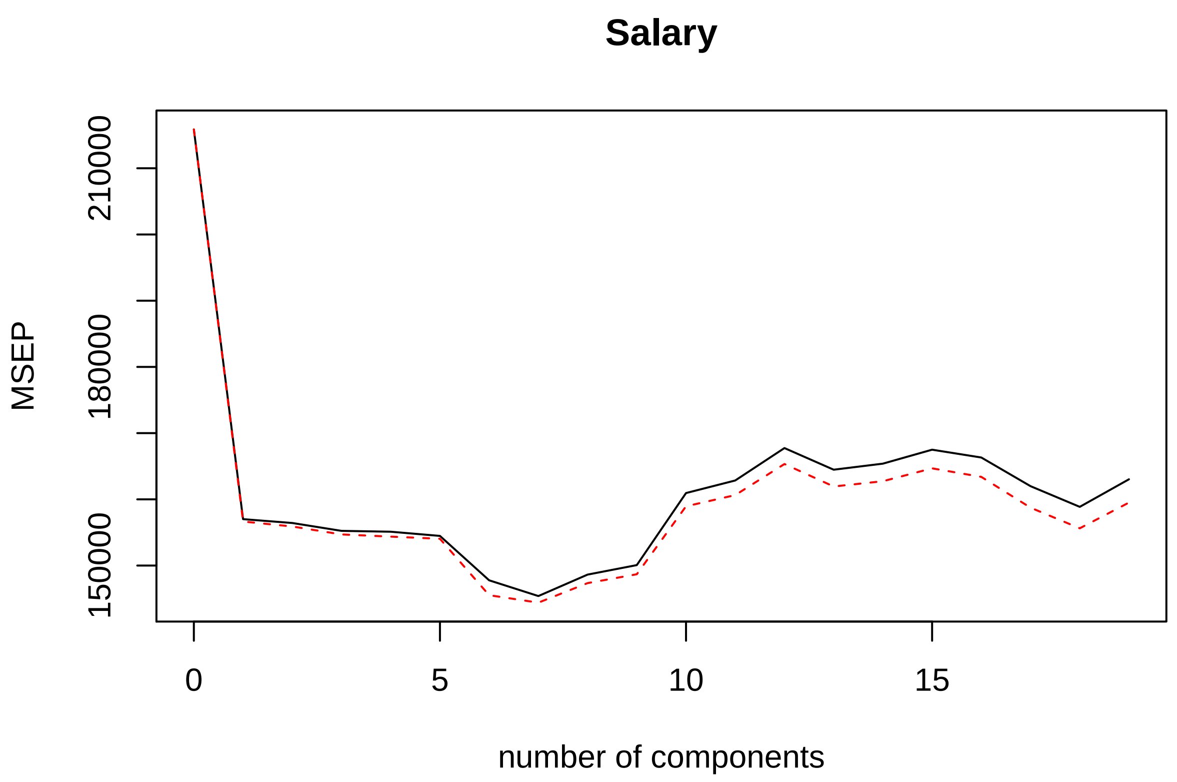
Now, let's look at a plot of cross-validation MSE across differing numbers of PCs. Note that in the following code, *val.type* can be set to any of *"MSEP", "RMSEP", or "R2"* for MSE, root MSE, or R^2 values.

```r
validationplot(pcr.fit, val.type = "MSEP")
```



Now, we run PCR on a randomly selected training set and check the MSPE on the held-out test set.

```r
set.seed(1)
#choose the training and test set randomly
x <- model.matrix(Salary ~., data = Hitters)[, -1]
y <- Hitters$Salary
train <- sample(1:nrow(x), nrow(x) / 2)
test <- (-train)
y.train <- y[train]
y.test <- y[test]

#fit PCR across a number of PCs
pcr.fit <- pcr(Salary ~ ., data = Hitters, subset = train, scale = TRUE, validation = "CV")

#look at the validation plot to choose the number of PCs
validationplot(pcr.fit, val.type = "MSEP")
```



```
##calculate the MSPE on the test set using 7 PCs from above
#predict the values on the test set
pcr.pred <- predict(pcr.fit, x[test, ], ncomp = 7)

#calculate the MSPE
mean((pcr.pred - y.test)^2)
```

```
## [1] 96556.22
```

Finally, we can fit PCR using 7 principal components as chosen from CV on the entire data set to use a model for future predictions.

```
pcr.final.model <- pcr(y ~ x, scale = TRUE, ncomp = 7)
summary(pcr.final.model)
```

```
## Data:    X dimension: 263 19
##  Y dimension: 263 1
## Fit method: svdpc
## Number of components considered: 7
## TRAINING: % variance explained
##    1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X    38.31    60.16    70.84    79.03    84.29    88.63    92.26
## y    40.63    41.58    42.17    43.22    44.90    46.48    46.69
```

```
coef(pcr.final.model)
```

```
## , , 7 comps
##
##                      y
## AtBat        27.005477
## Hits         28.531195
## HmRun         4.031036
## Runs        29.464202
## RBI         18.974255
## Walks       47.658639
## Years       24.125975
## CAtBat      30.831690
## CHits       32.111585
## CHmRun      21.811584
## CRuns       34.054133
## CRBI        28.901388
## CWalks      37.990794
## LeagueN      9.021954
## DivisionW  -66.069150
## PutOuts     74.483241
## Assists     -3.654576
## Errors      -6.004836
## NewLeagueN  11.401041
```