Introduction

This homework's goal is to get more familiar with linear regression. There are multiple kinds of linear regression, and this assignment focuses on 7 of them – linear regression with all predictors, linear regression with the best subset of predictors, linear regression with stepwise selection using AIC, ridge regression, LASSO, principal component regression, and partial least squares. Each kind of regression's training, testing, average cross validated errors and variances will be looked at and compared.

The data set being used for this assignment is the "fat" data set. This data set shows the results of a test that found the percentage of the body fat of men using an underwater weighing technique. Two different kinds of equations were used to find the percent body fat, Brozek's equation and Siri's equation. For this assignment, Brozek's equation will be used as the response variable.

Exploratory Data Analysis

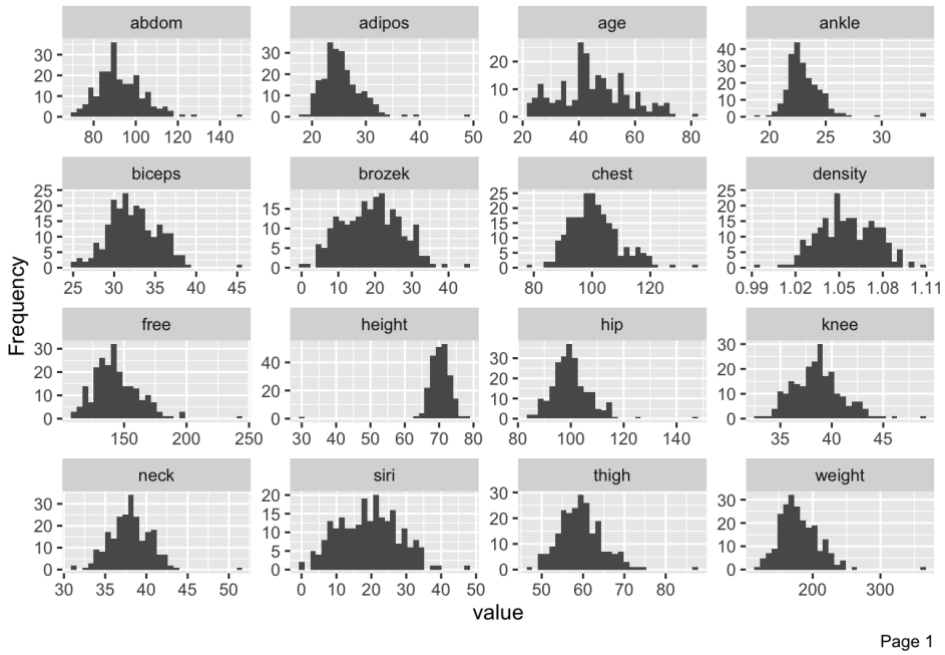
The original data set had 18 variables, each with 252 observations for each one. The variables are (brozek: percent body fat using Brozek's equation [457/Density – 414.2], siri: percent body fat using Siri's equation [495/Density – 450], density: density (gm/cm^3), age: age (yrs), weight: weight (lbs), height: height (inches), adipos: adiposity index = Weight/Height^2 (kg/m^2), free: fat free weight = (1 - fraction of body fat) * weight - using Brozek's formula (lbs), neck: neck circumference (cm), chest: chest circumference (cm), abdom: abdomen circumference (cm) at the umbilicus and level with the iliac crest, hip: hip circumference (cm), thigh: thigh circumference (cm), knee: knee circumference (cm), ankle: ankle circumference (cm), biceps: extended biceps circumference (cm), forearm: forearm circumference (cm), and wrist: wrist circumference (cm) distal to the styloid processes). As told, brozek is the response variable going to be used, and the rest of the 17 other variables will be the independent variables.

The "fat" data set was split up into training and testing data. The testing data set consists of 25 rows and the training data set consists of 227 rows. Both data sets contain the same variables as the original data set. The following is some basic analysis on the training data set:

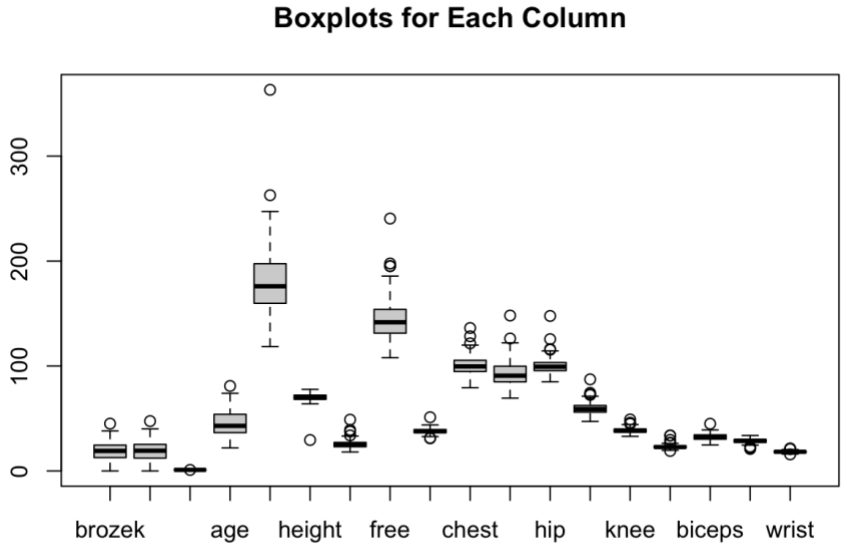
	Mean	Median	Mode
brozek	18.993392	19.1000	21.7000
siri	19.210573	19.3000	25.3000

	Mean	Median	Mode
density	1.055467	1.0547	1.0414
age	44.977974	43.0000	40.0000
weight	179.255286	176.0000	184.2500
height	70.159692	70.0000	71.5000
adipos	25.451101	24.9000	23.4000
free	143.777533	141.7000	133.1000
neck	37.991630	37.9000	38.5000
chest	100.867401	99.6000	99.1000
abdom	92.646256	90.8000	88.7000
hip	99.999559	99.3000	98.3000
thigh	59.482379	59.0000	58.9000
knee	38.604846	38.5000	39.0000
ankle	23.108811	22.8000	22.6000
biceps	32.342291	32.1000	30.5000
forearm	28.648458	28.7000	29.8000
wrist	18.238767	18.3000	18.8000

For the most part, the mean and median of each column are pretty similar to each other, meaning there is likely not too many outliers skewing the mean one way or another. To get a closer look at the distribution of the values of each variable, I looked at a histogram of each variable:

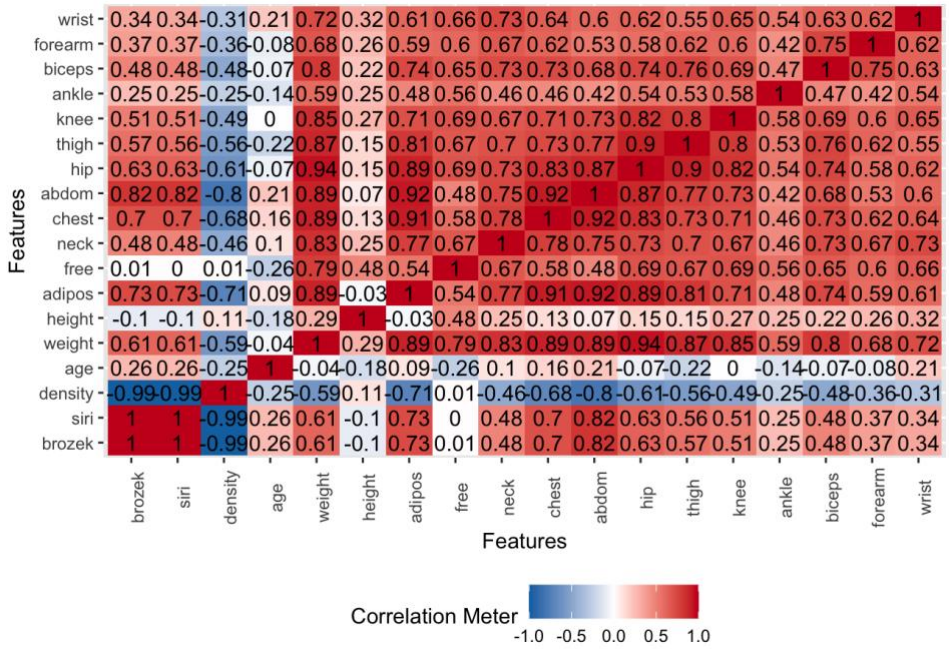


From the look of each histogram, each variable has a pretty normal distribution. As the graphs' show, almost each variable has one or more value that does not fall under that distribution. These values are likely outliers and are the cause for the mean and the median for some of the variables, for example age, abdom, and weight, to be different. To see if these were outliers, I looked at boxplots for each variable:



An open circle on the boxplot means the data point is considered an outlier, so it appears that each variable has at least one outlier in the training data set. Some of the outliers are

much larger outliers than others, so that will be something to look out for when doing the regression variations as I did not remove any of them from the data. The last thing I wanted to look at was the correlation matrix of the variables, meaning I wanted to see how related each variable was to the others. The following is the matrix:



If two variables are closely related to each other then they will be red (1 = perfect positive correlation), if they are not related to each other, they will be white (0 = no correlation), and if they are negatively related to each other, they will be blue (-1 = perfect negative correlation). The darker the color, the stronger the relation. This matrix also has the correlation values on it. The diagonal 1's come from each variable correlating to itself, so it will be a perfect correlation. Looking at this matrix, it shows that there are some variables that are very correlated with each other. Abdom and chest almost have a perfect positive correlation, weight is highly positively correlated with a couple of the other variables, age and height don't seem to be very correlated with any of the other variables, and density is either almost perfectly negatively correlated, negatively correlated, or 0 correlated with all the variables except itself.

Methods/Methodology

There were multiple methods used in this testing. They all revolved around a form of linear regression. The first one used was basic linear regression. This model minimizes errors and predicts the dependent variable based on the independent variables. The second method used was linear regression based on the best subset model. Using the leaps package in R, the best subset of independent variables for a given k (in this case k = 5) number of

predictors was found based on minimizing the residual sum of squares. The third regression used was using stepwise variable selection that minimizes the Akaike Information Criterion (AIC). Stepwise regression looks at both addition and deletion of one independent variable and chooses the option that in this case produces a smaller AIC. Ridge regression was the fourth regression used. This form of regression reduces overfitting by shrinking the coefficients of the independent variables, especially for ones that do little for the prediction. When there are some highly correlated variables, which in this case there are a couple, this regression model reduces their impact. The fifth regression used was LASSO regression. Unlike ridge regression, LASSO regression adds L1 regularization to a model which can shrink some coefficients to 0, which is a way of variable selection. Principal component regression (PCR) was the sixth regression model used. PCR is a combination of principal component analysis (PCA) and regression. PCA ranks each variable by how much they capture the variance of the model (the higher the principal component the more it captures – for example PCA1 captures more than PCA3) and then the top k principal components are chosen as the variables for linear regression. The last form of regression used was partial least squares (PLS). Like PCR, PLS selects components that explain the variance in the independent variables. Where they differ is that PLS also considers the components that explain the variance in the dependent variable as well.

The training, testing, and average cross validated errors and variances were found for each regression model. The Monte Carlo Cross Validation algorithm was used to cross validate with a repetition value of 100.

Results

The errors were not as I expected they would turn out to be. The testing error was by far the lowest, with all of them being almost perfect fits. The training and average cross validated errors were larger for each model. I believe this has to do with how correlated the data in this data set is.

Linear regression with all predictors:

Training Error:	0.02930823
Testing Error:	0.0003502392
Average Cross Validation Error:	0.002224738
Average Cross Validation Variance:	1.029697e-05

These numbers are the baseline for each of the models created from it. Due to the data being correlated, the error may be low due to overfitting because of multicollinearity.

Linear regression with the best k = 5 subset model:

Training Error:	0.03146801
Testing Error:	0.0001114487
Average Cross Validation Error:	0.001494487
Average Cross Validation Variance:	6.837376e-06

This model produced the lowest testing and average cross validation error. The best 5 k's selected to predict brozek in the testing data were siri, density, thigh, knee, and wrist. Looking back at the correlation matrix, this made sense. Siri was perfectly positively correlated to brozek and density was almost perfectly negatively correlated to it, so it makes sense that those two were selected as good predictors of the response. The remaining three were both also positively correlated, but there were others that were more correlated. My deduction is that either siri or density already explained the other ones that were higher, so they were not included. By reducing the number of predictors, the model had a lower prediction error likely because it got rid of some correlated predictors that were causing the coefficients to be unstable.

Linear regression with stepwise variable selection that minimizes AIC:

Training Error:	0.02945827
Testing Error:	0.0003582388
Average Cross Validation Error:	0.002329013
Average Cross Validation Variance:	1.237407e-05

Stepwise variable selection almost had the exact same error outputs as linear regression with all variables. This made sense when looking at which coefficients it chose to keep/leave out because it only got rid of one of them which was wrist. I found this surprising because it was selected as one of the predictors for to best k subset model. The reason stepwise almost replicated the full linear regression is because it removes predictors based on statistical significance, in this case minimizing AIC, and does not directly address multicollinearity.

Ridge regression:

Training Error:	0.0293089
Testing Error:	0.0003543693
Average Cross Validation Error:	0.002255873
Average Cross Validation Variance:	1.061705e-05

I found ridge regression having similar values to the linear regression with all variables surprising. Ridge regression is supposed to reduce the effect of positive predictors, but in

looking at the predictors it did not really reduce any of them. When looking at the testing data, the only one that had a large change was the density coefficient. I would have thought that some of the coefficients that are also correlated to siri would have been shrunk, but they all remained around that same value as in all variable linear regression. This implies that correlation is still influencing the model.

LASSO regression:

Training Error:	0.03085618
Testing Error:	0.0001263241
Average Cross Validation Error:	0.052160156
Average Cross Validation Variance:	5.425468e-03

LASSO regression produced the worst cross validation error value out of all the tests. I believe this happened because it got rid of relevant predictors due to their correlation with other predictors. On the testing data, the predictors that became 0 were weight, height, adipos, free, neck, chest, abdom, hip, and ankle. Predictors that became very close to 0 were age, thigh, knee, bicpes, forearm, and wrist. So, this regression almost based the entire regression on the predictors siri and density. The testing error looked good when done just one time, but when averaged out 100 times it was by far the worst value.

PCR:

Training Error:	0.02930823
Testing Error:	0.0003502392
Average Cross Validation Error:	0.002266986
Average Cross Validation Variance:	1.104264e-05

PCR produced the same training and testing errors as the full linear regression model and almost the same cross validation error value. This is because the PCR model ended up being the full linear regression model. It decided that to best explain the variance of the model all 17 principal components should be used. I think that the cross validation error is a little different because during one or more of the iterations it found that less principal components were needed than all 17.

PLS:

Training Error:	0.02930823
Testing Error:	0.0003502392
Average Cross Validation Error:	0.002272353
Average Cross Validation Variance:	1.062744e-05

Just like the PCR, the PLS also produced the full linear regression model with the same training and testing errors, and a slightly different cross validation error. Both the variance of the predictors and the response were explained best by using all 17 components.

Findings

Examining all the results, if I were to pick a model to use based on error values it would be the linear regression with the best k = 5 subset model. It provided the lowest testing and average cross validated errors. It was also the model with the lowest variance, meaning it provided the most consistent results out of all of them. In a real-world setting, I would also consider using PCR, PLS, or ridge regression. If I wanted to generalize correlated data, I would likely use PCR or PLS. If I wanted to have a simple, yet effective, model I would lean more towards ridge regression with possibly a higher lambda value than used or the best subset model.

I found a couple of things interesting during this data investigation. I found the PCR and PLS being the exact equation as the full linear regression model to be surprising. In all my time using them, they have not told me to use the full model, so that took me back a little. I also have not worked with data that kept in two perfectly correlated predictors, so it made sense why they did say to use all the predictors. Another thing I found interesting was that the ridge regression did not make much of a difference either. This made me look back at the model building and realize that the optimal lambda selected was very small, meaning the penalty applied to the coefficients was small and the shrinkage was not very strong, which would result in such a case.

If I were to do this again, I would do it without the siri or density variables. Siri was another way the people who collected this data tested for percent body fat, so most of what explains brozek explains siri. That is shown in the sense that they are perfectly correlated with each other. I would also get rid of the density variable. Brozek's equation is defined as 457/Density - 414.2, so density being a part of the formula for the equation means that it being a predictor would introduce very strong correlation. This was confirmed in the correlation matrix which shows that brozek and density have almost perfect negative correlation, which the equation suggests. I believe the error values for the tests would be very different if these two variables were removed.

Appendix

```
####
#### R code for Week #3: Linear Regression
##
## First, save the dataset "prostate.csv" in your laptop, say,
##       in the local folder "C:/temp".
##
## Data Set
fat <- read.table("fat.csv", header= TRUE, sep = ",")

##This dataset is from the textbook ESL, where the authors
##  have split the data into the training and testing subsets
##  here we use their split to produce similar results

n = dim(fat)[1];     ### total number of observations
n1 = round(n/10);    ### number of observations randomly selected for testing data
## To fix our ideas, let the following 25 rows of data as the testing subset:
flag = c(1,  21,  22,  57,  70,  88,  91,  94, 121, 127, 149, 151, 159, 162,
      164, 177, 179, 194, 206, 214, 215, 221, 240, 241, 243);
fat1train = fat[-flag,];
fat1test  = fat[flag,];
## The true Y response for the testing subset
ytrue    <- fat1test$brozek;

#Exploratory Data Analysis
library(DataExplorer)
```

```
length(fat1train$brozek)
get_mode <- function(v) {
  uniq_vals <- unique(v[!is.na(v)]) # Remove NAs and get unique values
  uniq_vals[which.max(tabulate(match(v, uniq_vals)))]
}
sapply(fat1train, get_mode)
mmm <- data.frame(
  Mean = sapply(fat1train, function(col) mean(col, na.rm = TRUE)),
  Median = sapply(fat1train, function(col) median(col, na.rm = TRUE)),
  Mode = sapply(fat1train, get_mode)
)
print(mmm)
plot_histogram(fat1train)
boxplot(fat1train, main = "Boxplots for Each Column")
plot_correlation(fat1train, maxcat = 5L)

### Below we consider seven (7) linear regression related models
### (1) Full model;  (2) The best subset model; (3) Stepwise variable selection with AIC
### (4) Ridge Regression; (5) LASSO;
##  (6) Principal Component Regression, and (7) Parital Least Squares (PLS) Regression
##
###   For each of these 7 models or methods, we fit to the training subset,
###    and then compute its training and testing errors.
##
##    Let us prepare to save all training and testing errors
MSEtrain <- NULL;
```

```
MSEtest  <- NULL;


###
### (1) Linear regression with all predictors (Full Model)
###    This fits a full linear regression model on the training data
model1 <- lm( brozek ~ ., data = fat1train);


## Model 1: Training error
MSEmod1train <-   mean( (resid(model1) )^2);
MSEtrain <- c(MSEtrain, MSEmod1train);
MSEmod1train
# Model 1: testing error
pred1a <- predict(model1, fat1test[,2:18]);
MSEmod1test <-   (1/n1) * mean((pred1a - ytrue)^2);
MSEmod1test;
MSEtest <- c(MSEtest, MSEmod1test);
#[1] 0.521274


### (2) Linear regression with the best subset model
###  YOu need to first install the package "leaps"
library(leaps);
fat.leaps <- regsubsets(brozek ~ ., data= fat1train, nbest= 100, really.big= TRUE);


## Record useful information from the output
fat.models <- summary(fat.leaps)$which;
fat.models.size <- as.numeric(attr(fat.models, "dimnames")[[1]]);
```

```
fat.models.rss <- summary(fat.leaps)$rss;

## 2A:  The following are to show the plots of all subset models
##   and the best subset model for each subset size k
plot(fat.models.size, fat.models.rss);
## find the smallest RSS values for each subset size
fat.models.best.rss <- tapply(fat.models.rss, fat.models.size, min);
## Also add the results for the only intercept model
fat.model0 <- lm( brozek ~ 1, data = fat1train);
fat.models.best.rss <- c( sum(resid(fat.model0)^2), fat.models.best.rss);
## plot all RSS for all subset models and highlight the smallest values
plot( 0:8, fat.models.best.rss, type = "b", col= "red", xlab="Subset Size k", ylab="Residual Sum-of-Square")
points(fat.models.size, fat.models.rss)

# 2B: What is the best subset with k=5
op2 <- which(fat.models.size == 5);
flag2 <- op2[which.min(fat.models.rss[op2])];



## 2B We can auto-find the best subset with k=3
##   this way will be useful when doing cross-validation
mod2selectedmodel <- fat.models[flag2,];
mod2Xname <- paste(names(mod2selectedmodel)[mod2selectedmodel][-1], collapse="+");
mod2form <- paste ("brozek ~", mod2Xname);
```

```
## To auto-fit the best subset model with k=3 to the data
model2 <- lm( as.formula(mod2form), data= fat1train);
# Model 2: training error
MSEmod2train <- mean(resid(model2)^2);
MSEmod2train
## save this training error to the overall training error vector
MSEtrain <- c(MSEtrain, MSEmod2train);
MSEtrain;
## Model 2:  testing error
pred2 <- predict(model2, fat1test[,2:18]);
MSEmod2test <-   (1/n1) * mean((pred2 - ytrue)^2);
MSEmod2test
MSEtest <- c(MSEtest, MSEmod2test);
MSEtest;
## Check the answer
##[1] 0.5212740 0.4005308

## As compared to the full model #1, the best subset model with K=3
##   has a larger training eror (0.521 vs 0.439),
##   but has a smaller testing error (0.400 vs 0.521).


### (3) Linear regression with the stepwise variable selection
###    that minimizes the AIC criterion
##    This can done by using the "step()" function in R,
##      but we need to build the full model first
```

```
model1 <- lm( brozek ~ ., data = fat1train);
model3  <- step(model1);

## If you want, you can see the coefficents of model3
round(coef(model3),3)
summary(model3)

## Model 3: training  and  testing errors
MSEmod3train <- mean(resid(model3)^2);
MSEmod3train
pred3 <- predict(model3, fat1test[,2:18]);
MSEmod3test <-  (1/n1) * mean((pred3 - ytrue)^2);
MSEmod3test
MSEtrain <- c(MSEtrain, MSEmod3train);
MSEtrain;
## [1] 0.4391998 0.5210112 0.4393627
MSEtest <- c(MSEtest, MSEmod3test);
## Check your answer
MSEtest;
## [1] 0.5212740 0.4005308 0.5165135


### (4) Ridge regression (MASS: lm.ridge, mda: gen.ridge)
### We need to call the "MASS" library in R
###
```

```
library(MASS);

## The following R code gives the ridge regression for all penality function lamdba
##  Note that you can change lambda value to other different range/stepwise
fat.ridge <- lm.ridge( brozek ~ ., data = fat1train, lambda= seq(0,100,0.001));

## 4A. Ridge Regression plot how the \beta coefficients change with \lambda values
##   Two equivalent ways to plot
plot(fat.ridge)
### Or "matplot" to plot the columns of one matrix against the columns of another
matplot(fat.ridge$lambda, t(fat.ridge$coef), type="l", lty=1,
      xlab=expression(lambda), ylab=expression(hat(beta)))

## 4B: We need to select the ridge regression model
##       with the optimal lambda value
##    There are two ways to do so

## 4B(i) manually find the optimal lambda value
##    but this is infeasible for cross-validation
select(fat.ridge)
##
#modified HKB estimator is 3.355691
#modified L-W estimator is 3.050708
# smallest value of GCV  at 4.92
#
# The output suggests that a good choice is lambda = 4.92,
```

```
abline(v=0.003)
# Compare the ceofficients of ridge regression with lambda= 4.92
##  versus the full linear regression model #1 (i.e., with lambda = 0)
fat.ridge$coef[, which(fat.ridge$lambda == 0.003)]
fat.ridge$coef[, which(fat.ridge$lambda == 0)]

## 4B(ii) Auto-find the "index" for the optimal lambda value for Ridge regression
##       and auto-compute the corresponding testing and testing error
indexopt <-  which.min(fat.ridge$GCV);

## If you want, the corresponding coefficients with respect to the optimal "index"
##  it is okay not to check it!
fat.ridge$coef[,indexopt]
## However, this coefficeints are for the the scaled/normalized data
##     instead of original raw data
## We need to transfer to the original data
## Y = X \beta + \epsilon, and find the estimated \beta value
##       for this "optimal" Ridge Regression Model
## For the estimated \beta, we need to sparate \beta_0 (intercept) with other \beta's
ridge.coeffs = fat.ridge$coef[,indexopt]/ fat.ridge$scales;
intercept = -sum( ridge.coeffs  * colMeans(fat1train[,2:18] )  )+ mean(fat1train[,1]);
## If you want to see the coefficients estimated from the Ridge Regression
##   on the original data scale
c(intercept, ridge.coeffs);

## Model 4 (Ridge): training errors
```

```
yhat4train <- as.matrix( fat1train[,2:18]) %*% as.vector(ridge.coeffs) + intercept;
MSEmod4train <- mean((yhat4train - fat1train$brozek)^2);
MSEmod4train
MSEtrain <- c(MSEtrain, MSEmod4train);
MSEtrain
## [1]  0.4391998 0.5210112 0.4393627 0.4473617
## Model 4 (Ridge):  testing errors in the subset "test"
pred4test <- as.matrix( fat1test[,2:18]) %*% as.vector(ridge.coeffs) + intercept;
MSEmod4test <-  (1/n1) * mean((pred4test - ytrue)^2);
MSEmod4test
MSEtest <- c(MSEtest, MSEmod4test);
MSEtest;
## [1] 0.5212740 0.4005308 0.5165135 0.4943531



## Model (5): LASSO
## IMPORTANT: You need to install the R package "lars" beforehand
##
library(lars)
fat.lars <- lars( as.matrix(fat1train[,2:18]), fat1train[,1], type= "lasso", trace= TRUE);

## 5A: some useful plots for LASSO for all penalty parameters \lambda
plot(fat.lars)

## 5B: choose the optimal \lambda value that minimizes Mellon's Cp criterion
Cp1  <- summary(fat.lars)$Cp;
```

```
index1 <- which.min(Cp1);


## 5B(i) if you want to see the beta coefficient values (except the intercepts)
##  There are three equivalent ways
##   the first two are directly from the lars algorithm
coef(fat.lars)[index1,]
fat.lars$beta[index1,]
##  the third way is to get the coefficients via prediction function
lasso.lambda <- fat.lars$lambda[index1]
coef.lars1 <- predict(fat.lars, s=lasso.lambda, type="coef", mode="lambda")
coef.lars1$coef
## Can you get the intercept value?
##  \beta0 = mean(Y) - mean(X)*\beta of training data
##     for all linear models including LASSO
LASSOintercept = mean(fat1train[,1]) -sum( coef.lars1$coef  * colMeans(fat1train[,2:18] ));
c(LASSOintercept, coef.lars1$coef)


## Model 5:  training error for lasso
##
pred5train  <- predict(fat.lars, as.matrix(fat1train[,2:18]), s=lasso.lambda, type="fit",
mode="lambda");
yhat5train <- pred5train$fit;
MSEmod5train <- mean((yhat5train - fat1train$brozek)^2);
MSEmod5train
MSEtrain <- c(MSEtrain, MSEmod5train);
MSEtrain
```

```
# [1] 0.4391998 0.5210112 0.4393627 0.4473617 0.4398267

##

## Model 5:  training error for lasso

pred5test <- predict(fat.lars, as.matrix(fat1test[,2:18]), s=lasso.lambda, type="fit",
mode="lambda");

yhat5test <- pred5test$fit;

MSEmod5test <- (1/n1) * mean( (yhat5test - fat1test$brozek)^2);

MSEmod5test

MSEtest <- c(MSEtest, MSEmod5test);

MSEtest;

## Check your answer:

## [1] 0.5212740 0.4005308 0.5165135 0.4943531 0.5074249



#### Model 6: Principal Component Regression (PCR)

##

## We can either manually conduct PCR by ourselves

##   or use R package such as "pls" to auto-run PCR for us

##

## For purpose of learning, let us first conduct the manual run of PCR

##  6A: Manual PCR:

##  6A (i) some fun plots for PCA of training data

trainpca <- prcomp(fat1train[,2:18]);

##

## 6A(ii)  Examine the square root of eigenvalues

## Most variation in the predictors can be explained
```

```
## in the first a few dimensions
trainpca$sdev
round(trainpca$sdev,2)
### 6A (iii) Eigenvectors are in oj$rotation
### the dim of vectors is 8
###
matplot(2:18, trainpca$rot[,1:3], type ="l", xlab="", ylab="")
matplot(2:18, trainpca$rot[,1:5], type ="l", xlab="", ylab="")
##
## 6A (iv) Choose a number beyond which all e. values are relatively small
plot(trainpca$sdev,type="l", ylab="SD of PC", xlab="PC number")
##
## 6A (v) An an example, suppose we want to do Regression on the first 4 PCs
## Get Pcs from obj$x
modelpca <- lm(brozek ~ trainpca$x[,1:4], data = fat1train)
##
## 6A (vi) note that this is on the PC space (denote by Z), with model Y= Z\gamma + epsilon
## Since the PCs Z= X U for the original data, this yields to
## Y= X (U\gamma) + epsilon,
## which is the form Y=X\beta + epsilon in the original data space
##  with \beta = U \gamma.
beta.pca <- trainpca$rot[,1:4] %*% modelpca$coef[-1];
##
## 6A (vii) as a comparion of \beta for PCA, OLS, Ridge and LASSO
##   without intercepts, all on the original data scale
cbind(beta.pca, coef(model1)[-1], ridge.coeffs, coef.lars1$coef)
```

```
##
### 6A(viii) Prediciton for PCA
### To do so, we need to first standardize the training or testing data,
### For any new data X, we need to impose the center as in the training data
###  This requires us to subtract the column mean of training from the test data
xmean <- apply(fat1train[,2:18], 2, mean);
xtesttransform <- as.matrix(sweep(fat1test[,2:18], 2, xmean));
##
## 6A (iX) New testing data X on the four PCs
xtestPC <-  xtesttransform %*% trainpca$rot[,1:4];
##
## 6A (X) the Predicted Y
ypred6 <- cbind(1, xtestPC) %*% modelpca$coef;
##
## In practice, one must choose the number of PC carefully.
##  Use validation dataset to choose it. Or Use cross-Validation
##  This can be done use the R package, say "pls"
##  in the "pls", use the K-fold CV -- default; divide the data into K=10 parts
##
## 6B: auto-run PCR
##
## You need to first install the R package "pls" below
##
library(pls)
## 6B(i): call the pcr function to run the linear regression on all possible # of PCs.
##
```

```
fat.pca <- pcr(brozek~., data=fat1train, validation="CV");
##
## 6B(ii) You can have some plots to see the effects on the number of PCs
validationplot(fat.pca);
summary(fat.pca);
## The minimum occurs at 8 components
## so for this dataset, maybe we should use full data
##
### 6B(iii) How to auto-select # of components
##    automatically optimazation by PCR based on the cross-validation
##    It chooses the optimal # of components
ncompopt <- which.min(fat.pca$validation$adj);
ncompopt
##
## 6B(iv) Training Error with the optimal choice of PCs
ypred6train <- predict(fat.pca, ncomp = ncompopt, newdata = fat1train[2:18]);
MSEmod6train <- mean( (ypred6train - fat1train$brozek)^2);
MSEmod6train
MSEtrain <- c(MSEtrain, MSEmod6train);
MSEtrain;
## 6B(v) Testing Error with the optimal choice of PCs
ypred6test <- predict(fat.pca, ncomp = ncompopt, newdata = fat1test[2:18]);
MSEmod6test <- (1/n1) * mean( (ypred6test - fat1test$brozek)^2);
MSEmod6test
MSEtest <- c(MSEtest, MSEmod6test);
MSEtest;
```

```
## Check your answer:
## [1] 0.5212740 0.4005308 0.5165135 0.4943531 0.5074249 0.5212740
##
## Fo this specific example, the optimal # of PC
##       ncompopt = 8, which is the full dimension of the original data
##   and thus the PCR reduces to the full model!!!



### Model 7. Partial Least Squares (PLS) Regression
###
###  The idea is the same as the PCR and can be done by "pls" package
###  You need to call the fuction "plsr"  if you the code standalone
#  library(pls)
fat.pls <- plsr(brozek ~ ., data = fat1train, validation="CV");

### 7(i) auto-select the optimal # of components of PLS
## choose the optimal # of components
mod7ncompopt <- which.min(fat.pls$validation$adj);
## The opt # of components, it turns out to be 8 for this dataset,
##      and thus PLS also reduces to the full model!!!

# 7(ii) Training Error with the optimal choice of "mod7ncompopt"
# note that the prediction is from "prostate.pls" with "mod7ncompopt"
ypred7train <- predict(fat.pls, ncomp = mod7ncompopt, newdata = fat1train[2:18]);
MSEmod7train <- mean( (ypred7train - fat1train$brozek)^2);
MSEmod7train
```

```
MSEtrain <- c(MSEtrain, MSEmod7train);
MSEtrain;
## 7(iii) Testing Error with the optimal choice of "mod7ncompopt"
ypred7test <- predict(fat.pls, ncomp = mod7ncompopt, newdata = fat1test[2:18]);
MSEmod7test <- (1/n1) * mean( (ypred7test - fat1test$brozek)^2);
MSEmod7test
MSEtest <- c(MSEtest, MSEmod7test);
MSEtest;

## Check your answers
MSEtrain
## Training errors of these 7 models/methods
#[1] 0.4391998 0.5210112 0.4393627 0.4473617 0.4398267 0.4391998 0.4391998
MSEtest
## Testing errors of these 7 models/methods
#[1] 0.5212740 0.4005308 0.5165135 0.4943531 0.5074249 0.5212740 0.5212740
##
## For this specific dataset, PCR and PLS reduce to the full model!!!

### Part (e): the following R code might be useful, and feel free to modify it.
###    save the TE values for all models in all $B=100$ loops
B= 100;
TEALL = NULL;
set.seed(7406);
### number of loops
### Final TE values
```

```
### You might want to set the seed for randomization
for (b in 1:B){
 ### randomly select 25 observations as testing data in each loop
 flag <- sort(sample(1:n, n1));
 fattrain <- fat[-flag,];
 fattest  <- fat[flag,];
 truey <- fattest$brozek;

 ### you can write your own R code here to first fit each model to "fattrain"
 ### then get the testing error (TE) values on the testing data "fattest"
 ### Suppose that you save the TE values for these five models as
 ###   te1, te2, te3, te4, te5, te6, te7, respectively, within this loop
 ###   Then you can save these 5 Testing Error values by using the R code
 ###
 # Model 1: Linear regression with all predictors
 mod1 <- lm( brozek ~ ., data = fattrain);
 pred1 <- predict(mod1, fattest[,2:18]);
 te1 <-   (1/n1) * mean((pred1 - truey)^2);

 # Model 2: Linear regression with the best subset of k = 5 predictors
 library(leaps);
 fat.l <- regsubsets(brozek ~ ., data= fattrain, nbest= 100, really.big= TRUE);
 fat.m <- summary(fat.l)$which
 fat.m.size <- as.numeric(attr(fat.m, "dimnames")[[1]]);
 fat.m.rss <- summary(fat.l)$rss;
 op <- which(fat.m.size == 5);
```

```
flagk <- op2[which.min(fat.m.rss[op])];
mod2selectedmod <- fat.models[flagk,];
mod2Xn <- paste(names(mod2selectedmod)[mod2selectedmod][-1], collapse="+");
mod2f <- paste ("brozek ~", mod2Xn);
mod2 <- lm( as.formula(mod2f), data= fattrain);
pred2b <- predict(mod2, fattest[,2:18]);
te2 <-   (1/n1) * mean((pred2b - truey)^2);

# Model 3: Linear regression with stepwise selection using AIC
mod3  <- step(mod1);
pred3b <- predict(mod3, fattest[,2:18]);
te3 <-  (1/n1) * mean((pred3b - truey)^2);

# Model 4: Ridge regression

library(MASS);
fat.rid <- lm.ridge( brozek ~ ., data = fattrain, lambda= seq(0,100,0.001));
idxopt <-  which.min(fat.rid$GCV);
fat.rid$coef[,idxopt]
rid.coeffs = fat.rid$coef[,idxopt]/ fat.rid$scales;
inter = -sum( rid.coeffs  * colMeans(fattrain[,2:18] )  )+ mean(fattrain[,1]);
c(inter, rid.coeffs);
pred4testb <- as.matrix( fattest[,2:18]) %*% as.vector(rid.coeffs) + inter;
te4 <-  (1/n1) * mean((pred4testb - truey)^2);
```

```
# Model 5: Lasso regression

library(glmnet)
X_train <- as.matrix(fattrain[, 2:18]) # Predictor matrix for training
y_train <- fattrain[, 1]           # Response variable for training
X_test <- as.matrix(fattest[, 2:18])  # Predictor matrix for testing
y_test <- truey                    # True response values for the test set
cv_lasso <- cv.glmnet(X_train, y_train, alpha = 1)  # alpha = 1 for Lasso
optimal_lambda <- cv_lasso$lambda.min
pred5testb <- predict(cv_lasso, newx = X_test, s = optimal_lambda)
te5 <- mean((pred5testb - y_test)^2)

# Model 6: Principal Component Regression (PCR)

library(pls)
fat.pc <- pcr(brozek~., data=fattrain, validation="CV");
validationplot(fat.pc);
summary(fat.pc);
ncompop <- which.min(fat.pc$validation$adj);
ypred6testb <- predict(fat.pc, ncomp = ncompop, newdata = fattest[2:18]);
te6 <- (1/n1) * mean( (ypred6testb - truey)^2);

# Model 7: Partial Least Squares Regression (PLSR)
fat.pl <- plsr(brozek ~ ., data = fattrain, validation="CV");
mod7ncompop <- which.min(fat.pl$validation$adj);
```

```
  ## 7(iii) Testing Error with the optimal choice of "mod7ncompopt"
  ypred7testb <- predict(fat.pl, ncomp = mod7ncompop, newdata = fattest[2:18]);
  te7 <- (1/n1) * mean( (ypred7testb - truey)^2);



  TEALL = rbind( TEALL, cbind(te1, te2, te3, te4, te5, te6, te7) );
}
dim(TEALL);  ### This should be a Bx7 matrices
### if you want, you can change the column name of TEALL
colnames(TEALL) <- c("mod1", "mod2", "mod3", "mod4", "mod5", "mod6", "mod7");
## You can report the sample mean and sample variances for the seven models
apply(TEALL, 2, mean);
apply(TEALL, 2, var);
### END ###
```