Introduction:

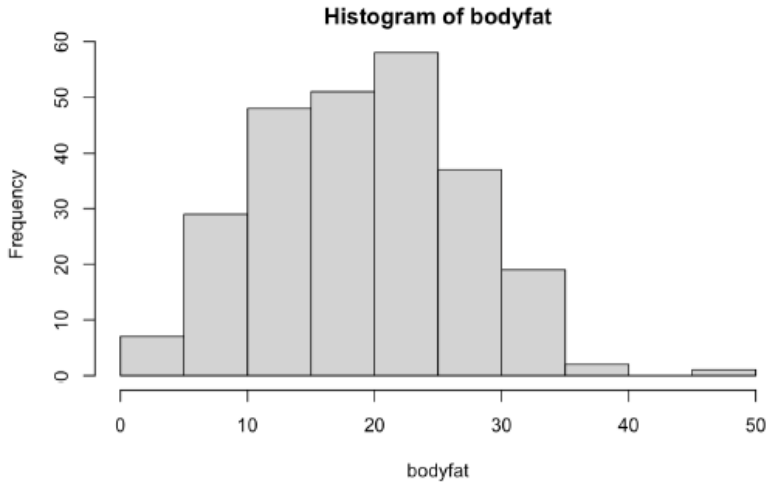
This report builds various linear and regularized models to estimate and predict men's body fat percentage based on some related variables using Brozek equation. Body fat percentage is the proportional mass of essential body fat and storage body fat to total body mass.

The dataset contains 252 rows of observations. It has 1 outcome variable of body fat percentage and 17 other independent variables containing various body measurements such as age, weight, height, etc. Due to the small size of the dataset, it was split into training (90%) and testing (10%) datasets. The splitting of the dataset was performed by manually picking 25 observations as test dataset for exploratory analysis, model fitting and calculating training and testing errors. For Monte Carlo cross validation process however, datasets were split randomly.

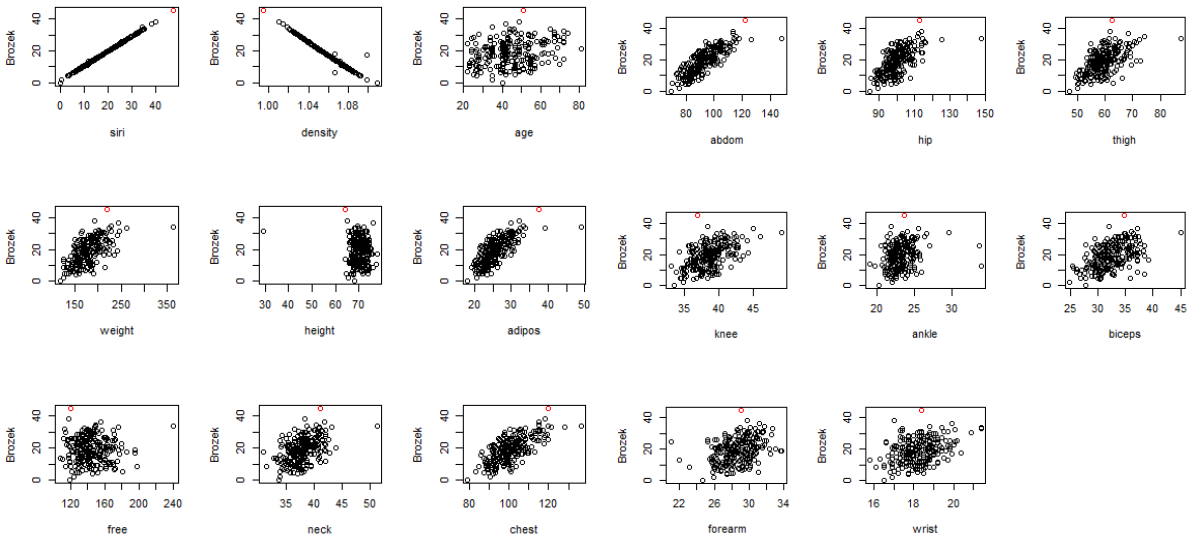
Data Source Johnson R. Journal of Statistics Education v.4, n.1 (1996)

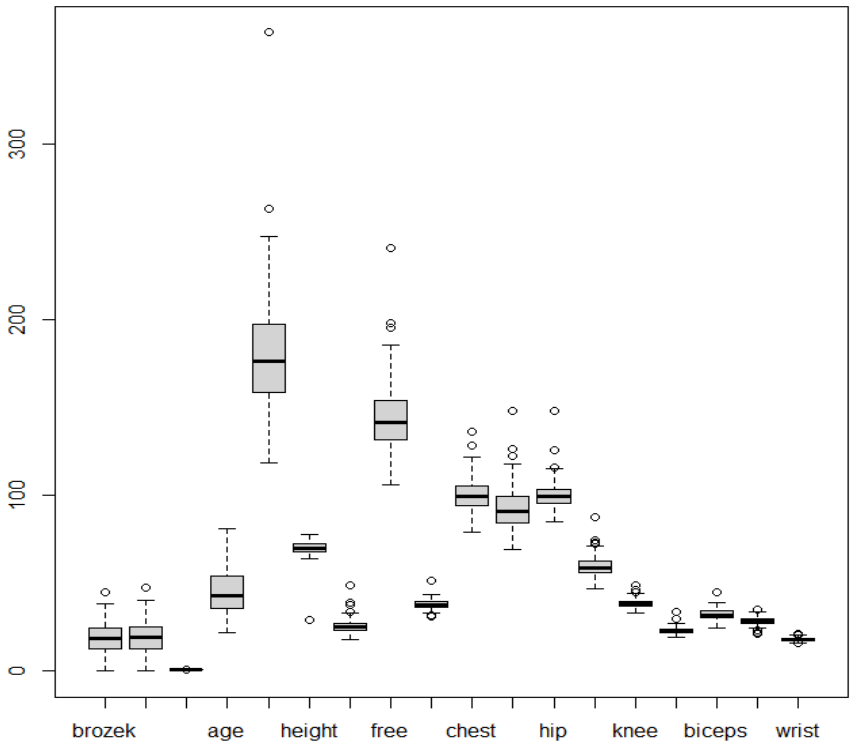
Exploratory Analysis:

The minimum value in our response variable is 0.0% and the maximum is 45.1%, with a mean of 18.9% and a median of 19.0%. We can plot a histogram of the response variable and observe it is an approximately normal distribution with a slight right skew.

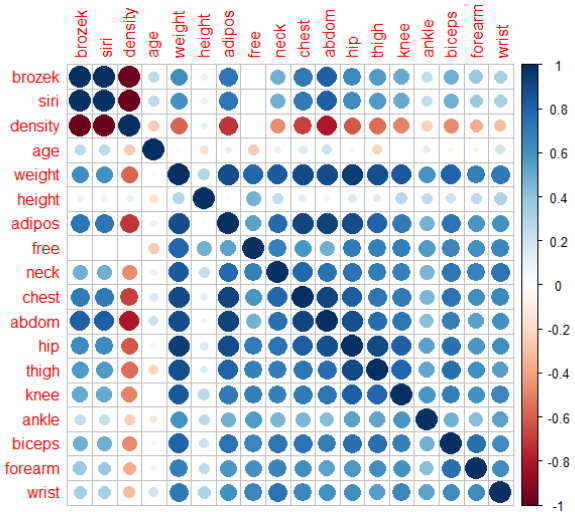


Body fat percentage seems to have a strong positive linear relationship with siri and a strong negative linear relationship with density as indicated by the first two scatterplots below. Besides height and age, all other variables seem to have a moderate positive linear relationship with body fat percentage. Height seems to have a negative relationship and age does not seem to have any apparent relationship with body fat percentage. We can also see some evidence of a few outliers, but we will include them for this analysis.


Brozek
siri
density
age
abdom
hip
thigh
weight
height
adipos
knee
ankle
biceps
free
neck
chest
forearm
wrist


0
100
200
300
brozek
age
height
free
chest
hip
knee
biceps
wrist

Correlation values above 0.80 and 0.90 seem to indicate a strong positive correlation among many variables. For example, weight seems to be correlated with chest, abdomen, and hip. Similarly, hip seems to be correlated with chest and abdomen. This indicates an issue of multicollinearity which could result into a complex model. Thus, variable selection models can be a perfect candidate in such cases.



Methods:

1) **Linear regression with all predictors (full model)**
   A multiple linear regression model using all 17 variables was fitted to the training dataset using lm() function in R. Testing responses were predicted using the predict() function, and testing errors were calculated through analyzing MSE.
2) **Linear regression with the best subset model k=5**
   Although the purpose of the best subset regression model is to run through all possible combination of predictive variables and selecting the best model, the model was only fit to return up to the best 5-variables model using the smallest RSS value using resubjects() function within the leaps package.
3) **Linear regression with variables (stepwise) selected using Akaike information criterion (AIC)**
   I applied stepwise variable selection method of direction "both" to the full model which simultaneously uses forward and backward search to minimize AIC using step() function.
4) **Ridge regression**
   This model was fitted using lm.ridge() regression function for various lambda values that changed the beta coefficients accordingly (Appendix, 4). The coefficients were derived using the auto selected optimal lambda of 0.03. The coefficients calculated using optimal lambda on the scaled data were converted to the original raw data which was then used for calculating the training and testing errors.
5) **Lasso**
   Lasso model was fitted using the lars() algorithm in R while choosing the optimal penalty parameter lambda (Appendix, 5) that minimizes Mellon's Cp criterion. I used the beta coefficients from the lars algorithm to calculate training and testing errors.
6) **Principal Component Regression (PCR)**
   Pcr function was used to run the linear regression on all possible number of principal components via cross validation (Appendix, 6). It auto selected optimal number of components via optimization based on default k-fold cross validation. The training and testing errors were calculated based on the optimal choice of principal components.
7) **Partial Least Squares (PLS) Regression**
   PLSR was fitted similarly using plsr() function.

Results:

All three linear regression models (Linear regression with all predictors, Linear regression with the best subset model and Linear regression with the best subset model k=5) produced a very high adjusted R-squared value of 0.995 indicating that most of the variability in the body fat percentage could be explained by the model (Appendix, 1-3). Based on a very small p-value from F-statistic (<0.05), we can reject the null hypothesis that all of regression coefficients are equal to zero. In other words, adding some of the variables in the models did improve their predictive capability.

PCR, PLSR, and full model are not variable selection models, thus coefficients will not be reduced to zeroes. These three models resulted with same coefficients since both PCR and PLSR identified 17 as the optimal number of components, which is the full dimension of the original data and thus the PCR and PLSR both reduced to the full model. Based on the coefficients, some variables have a moderate positive association with body fat percentage whereas some have a moderate negative association with body fat percentage. Siri and density seem to have strong effects on body fat percentage. Ridge regression decreases the complexity of a model but does not reduce the number of variables since it never leads to a coefficient being zero but only minimizes it. Lasso tends to make coefficients to absolute zero, thus eliminating those variables from the model.

	OLR with all/PCR/PLS	LR with k=5	LR with AIC	Ridge (λ = 0.003)	LASSO
intercept	12.3905	11.2034	12.5728	12.6095	11.27
siri	0.8834	0.9044	0.8842	0.8826	0.904
density	-9.9981	-9.2427	-10.2111	-10.2017	-9.406
age	-0.0007	-	-	-0.0007	-4E-04
weight	0.0115	-	0.011	0.0117	-
height	-0.0008	-	-	-0.0008	-
adipos	-0.0189	-	-0.016	-0.0191	-
free	-0.0133	-	-0.0125	-0.0135	-
neck	-0.0006	-	-	-0.0006	-
chest	0.0025	-	-	0.0026	-
abdom	0.0007	-	-	0.0008	-

hip	-0.0036	-	-	-0.0036	-
thigh	0.0146	0.0099	0.0131	0.0146	0.007
knee	-0.0262	-0.0245	-0.0274	-0.0261	-0.014
ankle	0.0033	-	-	0.0032	-
biceps	-0.0172	-	-0.0172	-0.0172	-0.009
forearm	0.0238	-	0.0256	0.0239	0.013
wrist	0.0327	0.0289	0.0295	0.0328	0.017

Table above compares coefficients for PCR, PLSR, Ordinary Least Square (full model), Ridge and LASSO without intercepts, all on the original data scale.

Training and Testing Errors:

	test error
Linear regression with all predictors	0.0087560
Linear regression with the best subset of k = 5 predictors variables	0.0027862
Linear regression with variables (stepwise) selected using AIC	0.0089560
Ridge regression	0.0088592
LASSO	0.0032000
Principal component regression	0.0087560
Partial least sq	0.0087560

The models built using training dataset were used to predict body fat percentage on testing dataset respectively. The performance of these models was evaluated using Mean Squared Errors (MSE) as training and testing errors evaluated on training and testing datasets respectively to measure variation from the actual values. Linear regression with the best subset of k = 5 predictors variables has the smallest testing error of 0.0027862 followed by Lasso of 0.0032000.

Cross validation training and testing errors and variances:

	test error	test variance
Linear regression with all predictors	0.054500	0.005900
Linear regression with the best subset of k = 5 predictors variables	0.047400	0.006400

Linear regression with variables (stepwise) selected using AIC	0.056900	0.006400
Ridge regression	0.055100	0.006100
LASSO	0.045700	0.005400
Principal component regression	0.054100	0.005900
Partial least sq	0.053900	0.006000

In the report, I re-split the same dataset randomly into training (90%) and testing dataset (10%) and ran each model 100 times using Monte Carlo Cross-Validation algorithm and calculated training and testing errors. This resulted into 100 testing errors which were averaged to compare the robustness of the models. Based on the average testing error and variance, the Linear regression with the best subset of k = 5 and Lasso seem to be performing the bes. The testing errors seems to be larger than the training errors for all models as expected. The conclusions based on testing errors calculated without performing cross-validation could be misleading.

Findings:

Overall, all models seem to be performing well in terms of explaining variability in the outcome variable based on the adjusted R squares and low errors. In terms of MSE, Linear regression with the best subset of k = 5 and Lasso consistently outperformed other. Both models are known to avoid multicollinearity and overfitting. Stepwise linear regression model used more variables than the subset model k=5 but did not use all variables like in the full model. The full model used all variables but not all were significant based on the p-values. PCR and PLS reduced to the full model as well. While ridge minimized some coefficients, lasso eliminated some variables. Based on the results of all models, these variables seem to be significant in predicting the body fat percentage: siri, density, thigh, knee, biceps, forearm and wrist.

The unusual higher testing errors than training errors could be a result of the biased data splitting method provided in the sample code for the first part of modelling. Manually selecting testing data points and the nature of the small dataset could have caused overfitting. This issue was eliminated in cross-validation as the data splitting method was made random. We can rerun the linear regression model only using the significant variables only on a larger dataset to further improve the model.

Appendix:

1. Linear Regression

```
> summary(model1)

Call:
lm(formula = brozek ~ ., data = fat1train)

Residuals:
     Min       1Q   Median       3Q      Max
-1.12406 -0.04543 -0.00043  0.04959  1.44629

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 12.3904670  4.3993444   2.816  0.00532 **
siri         0.8833923  0.0121994  72.412  < 2e-16 ***
density     -9.9981331  3.9393673  -2.538  0.01188 *
age         -0.0006612  0.0014479  -0.457  0.64839
weight       0.0115250  0.0044435   2.594  0.01017 *
height      -0.0008222  0.0046718  -0.176  0.86048
adipos      -0.0188876  0.0137436  -1.374  0.17083
free        -0.0132586  0.0052230  -2.538  0.01186 *
neck        -0.0006210  0.0100988  -0.061  0.95102
chest        0.0025162  0.0047545   0.529  0.59721
abdom        0.0007461  0.0049150   0.152  0.87949
hip         -0.0035967  0.0064599  -0.557  0.57828
thigh        0.0146022  0.0064676   2.258  0.02499 *
knee        -0.0261689  0.0106002  -2.469  0.01436 *
ankle        0.0032552  0.0095854   0.340  0.73450
biceps      -0.0171959  0.0078168  -2.200  0.02891 *
forearm      0.0237699  0.0105532   2.252  0.02534 *
wrist        0.0326568  0.0238623   1.369  0.17261
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1784 on 209 degrees of freedom
Multiple R-squared:  0.9995,    Adjusted R-squared:  0.9995
F-statistic: 2.622e+04 on 17 and 209 DF,  p-value: < 2.2e-16
```

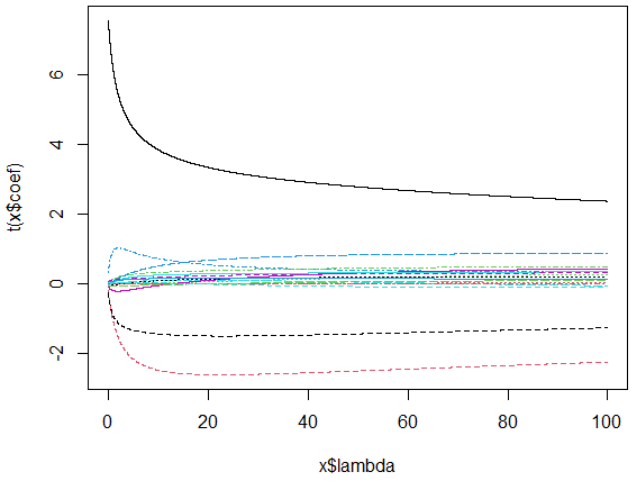
2. Subset model (Nbest=100 and nvmax=5 (k=5))

```
> summary(model2)

Call:
lm(formula = as.formula(mod2form), data = fat1train)

Residuals:
     Min       1Q   Median       3Q      Max
-1.07829 -0.04453 -0.00062  0.03778  1.59811

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 11.203419   4.227800   2.650  0.00863 **
siri         0.904414   0.008867 101.996  < 2e-16 ***
density     -9.242731   3.868070  -2.389  0.01771 *
thigh        0.009904   0.003942   2.512  0.01271 *
knee        -0.024479   0.009137  -2.679  0.00794 **
wrist        0.028891   0.017252   1.675  0.09541 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1798 on 221 degrees of freedom
Multiple R-squared:  0.9995,    Adjusted R-squared:  0.9995
F-statistic: 8.778e+04 on 5 and 221 DF,  p-value: < 2.2e-16
```

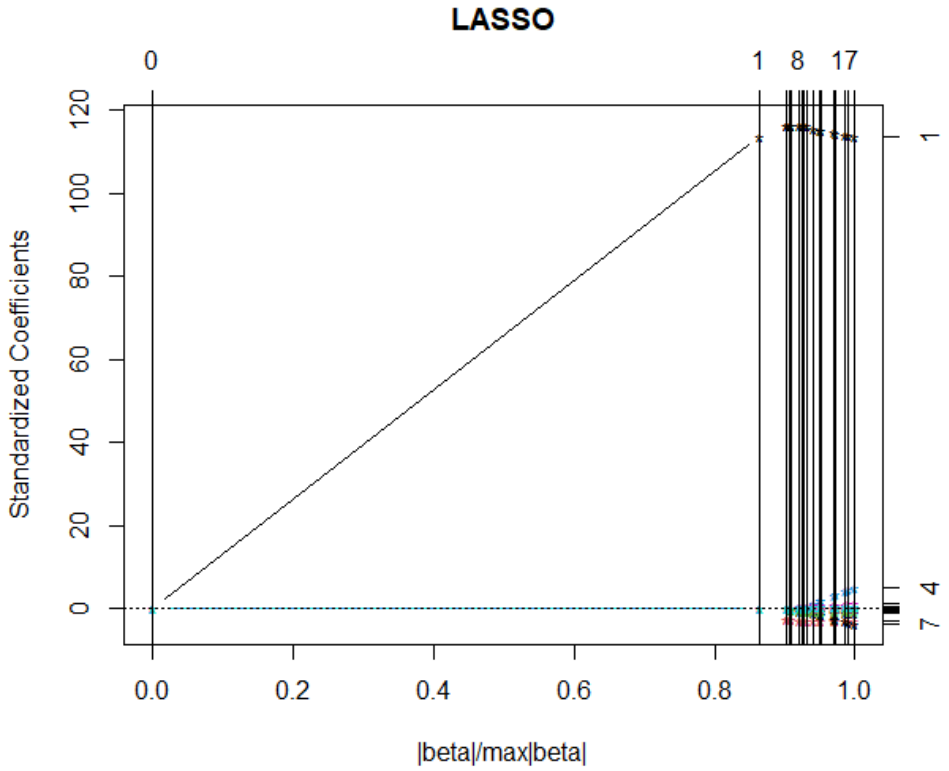
3. Linear regression with variables (stepwise) selected using AIC

```
> summary(model3)

Call:
lm(formula = brozek ~ siri + density + weight + adipos + free +
    thigh + knee + biceps + forearm + wrist, data = fat1train)

Residuals:
     Min       1Q   Median       3Q      Max
-1.12873 -0.04496 -0.00286  0.05366  1.46273

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  12.572781   4.222127   2.978  0.00323 **
siri          0.884216   0.011610  76.157  < 2e-16 ***
density     -10.211124   3.820851  -2.672  0.00810 **
weight        0.011021   0.004086   2.697  0.00755 **
adipos       -0.016046   0.009373  -1.712  0.08833 .
free         -0.012479   0.004971  -2.510  0.01280 *
thigh         0.013146   0.005091   2.582  0.01048 *
knee         -0.027377   0.009921  -2.760  0.00629 **
biceps       -0.017234   0.007604  -2.266  0.02442 *
forearm       0.025573   0.009837   2.600  0.00998 **
wrist         0.029532   0.020327   1.453  0.14771
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.176 on 216 degrees of freedom
Multiple R-squared:  0.9995,    Adjusted R-squared:  0.9995
F-statistic: 4.582e+04 on 10 and 216 DF,  p-value: < 2.2e-16
```

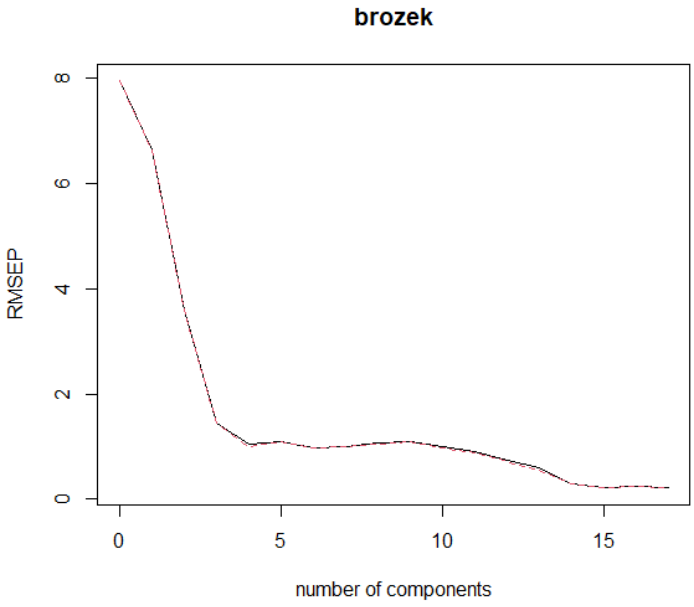
4. Ridge regression



5. LASSO



6. Choosing PCs automatically using r package pls:



Code:

```
#read data

fat <- read.table(file="fat.csv", header= TRUE, sep = ",")

head(fat,5) #first 5 rows

### Split the data as in Part (a)
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,];


#########exploratory analysis##############

#scatterplots


for (i in c(2:18)){
  col_name = names(fat1train[i])
  plot(fat1train[,i], fat1train$brozek, xlab= col_name, ylab = "Brozek",
```

```
       col=ifelse(fat1train$brozek== max(fat1train$brozek), 'red',
'black'))
}



# Correlation matrix
round(cor(fat1train[-2]),4)


#install.packages("corrplot")
library(corrplot)


corrplot(cor(fat1train))


boxplot(fat) #boxplot
summary(fat) #summary table


############################################################build
models


## The true Y response for the testing subset
ytrue    <- fat1test$brozek;



#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);

summary(model1)


## Model 1: Training error

MSEmod1train <-   mean((resid(model1) )^2);

MSEtrain <- c(MSEtrain, MSEmod1train);



# Model 1: testing error

pred1a <- predict(model1, fat1test[,2:18]);

MSEmod1test <-  mean((pred1a - ytrue)^2);

MSEmod1test;

MSEtest <- c(MSEtest, MSEmod1test);


# obtain coefficients

round(model1$coefficients,4)


#########################################################################
####################################


### (2) Linear regression with the best subset model

#playing with nbest and nvmax=5--play with k=5 and diff # of
nbest..should be higher than # of vars
```

```
library(leaps);

fat.leaps <- regsubsets(brozek ~ ., data= fat1train, nbest= 100,
really.big= TRUE, nvmax = 5); #model of size 100

#fat.leaps <- regsubsets(brozek ~ ., data= fat1train, nbest= 100,
really.big= TRUE); #By default, regsubsets() only reports results up
to the best eight-variable model. But the nvmax option can be used in
order to return as many variables as are desired. Here we fit up to a
19-variable model:


## Recording useful information from the output

fat.models <- summary(fat.leaps)$which; #summary() command outputs the
best set of variables for each model size.

fat.models.size <- as.numeric(attr(fat.models, "dimnames")[[1]]);

fat.models.rss <- summary(fat.leaps)$rss;#adjusted  R2


##he 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);

##the smallest RSS values for each subset size

fat.models.best.rss <- tapply(fat.models.rss, fat.models.size, min);

## Also adding 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);

## ploting all RSS for all subset models and highlighting the smallest
values

plot(0:5,fat.models.best.rss, type = "b", col= "red", xlab="Subset
Size k", ylab="Residual Sum-of-Square")

points(fat.models.size, fat.models.rss)


# What is the best subset with k=5

op2 <- which(fat.models.size == 5);

flag2 <- op2[which.min(fat.models.rss[op2])];
```

```
## There are two ways to fit this best subset model with k=5.


## First, we can manually look at the selected model and fit it.
##      It will not be able to be implemented in cross-validation
fat.models[flag2,]
model2a <- lm( brozek ~ siri+density+thigh+knee+wrist, data =
fat1train);
summary(model2a);


## Second, we can auto-find the best subset with k=5
##   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=5 to the data
model2 <- lm( as.formula(mod2form), data= fat1train);


# obtain coefficients
round(model2$coefficients,4)



# Model 2: training error
MSEmod2train <- mean(resid(model2)^2);
## saving 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 <-   mean((pred2 - ytrue)^2);
MSEtest <- c(MSEtest, MSEmod2test);
```

```
MSEtest;


##########################################################################
##############################################


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

#install.packages("stargazer")
library(stargazer)

model1 <- lm( brozek ~ ., data = fat1train); #full model
model3  <- step(model1, direction= "both");

##coefficents of model3
round(coef(model3),3)
summary(model3)

stargazer(model1, model3, type = "text")


## Model 3: training  and  testing errors
MSEmod3train <- mean(resid(model3)^2);
pred3 <- predict(model3, fat1test[,2:18]);
MSEmod3test <-   mean((pred3 - ytrue)^2);
MSEtrain <- c(MSEtrain, MSEmod3train);
MSEtrain;
```

```
##  0.02930823 0.03146801 0.02945827

MSEtest <- c(MSEtest, MSEmod3test);

## Check your answer

MSEtest;


#########################################################################
#############################################

### (4) Ridge regression (MASS: lm.ridge, mda: gen.ridge)


library(MASS);


## The following R code gives the ridge regression for all penality
function lamdba


fat.ridge <- lm.ridge( brozek ~ ., data = fat1train, lambda=
seq(0,100,0.001));


##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)))


## We need to select the ridge regression model

##        with the optimal lambda value

##     There are two ways to do so


##(i) manually finding the optimal lambda value
```

```
##    but this is infeasible for cross-validation
select(fat.ridge)
##
#modified HKB estimator is 0.00836436
#modified L-W estimator is 0.007640264
#smallest value of GCV  at 0.003
#
# The output suggests that a good choice is lambda = 0.003,
abline(v=0.003)
# Compare the coefficients of ridge regression with lambda= 0.003
##  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)]


##(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]); #isolating intercept - was not scaled, thus

##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);

MSEtrain <- c(MSEtrain, MSEmod4train);

MSEtrain



## Model 4 (Ridge):  testing errors in the subset "test"

pred4test <- as.matrix( fat1test[,2:18]) %*% as.vector(ridge.coeffs) +
intercept;

MSEmod4test <-  mean((pred4test - ytrue)^2);

MSEtest <- c(MSEtest, MSEmod4test);

MSEtest;


######################################################################
################################


## Model (5): LASSO


library(lars)

fat.lars <- lars( as.matrix(fat1train[,2:18]), fat1train[,1], type=
"lasso", trace= TRUE);
```

```
## ome useful plots for LASSO for all penalty parameters \lambda
plot(fat.lars)

##the optimal \lambda value that minimizes Mellon's Cp criterion
Cp1  <- summary(fat.lars)$Cp;
index1 <- which.min(Cp1);

##to see the beta coefficient values (except the intercepts)
##   There are three equivalent ways
##    the first two are directly from the lars algorithm
lasso.lambda2 <- coef(fat.lars)[index1,]
lasso.lambda3 <- fat.lars$beta[index1,]

##   the third way is to get the coefficients via prediction function
lasso.lambda <- fat.lars$lambda[index1]
coef.lars1 <- predict(fat.lars,
as.matrix(rbind(rep.int(0,17),rep.int(1,17))),
                      s=lasso.lambda, type="fit", mode="lambda")
round(c(coef.lars1$fit[1],coef(fat.lars)[index1,]),4)

## 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);

MSEtrain <- c(MSEtrain, MSEmod5train);

MSEtrain



## Model 5:  testing error for lasso

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

yhat5test <- pred5test$fit;

MSEmod5test <- mean( (yhat5test - fat1test$brozek)^2);

MSEtest <- c(MSEtest, MSEmod5test);

MSEtest;


##########################################################################
###############################################


#### 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

##

##manual run of PCR


library(ggfortify)


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

autoplot(trainpca)

##
```

```
## (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)

### (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="")

##

## (iv) Choose a number beyond which all e. values are relatively
small

plot(trainpca$sdev,type="l", ylab="SD of PC", xlab="PC number")

##

## (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[,1:17], data = fat1train)

##

## (vi) note 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$rot[,1:17] %*% modelpca$coef[-1];

##

## (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)
```

```
##

###(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));

##

## (iX) New testing data X on the four PCs

xtestPC <-  xtesttransform %*% trainpca$rot[,1:4];

##

## (X) the Predicted Y

ypred6 <- cbind(1, xtestPC) %*% modelpca$coef;

##

## In practice, one must choose the number of PC carefully.

##   Using 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; dividing the data into
K=10 parts

##




##################################################auto-run PCR

##

## You need to first install the R package "pls" below

##

library(pls)

## (i): call the pcr function to run the linear regression on all
possible # of PCs.
```

```
##
fat.pca <- pcr(brozek~., data=fat1train, validation="CV");
##
## (ii) plots to see the effects on the number of PCs
validationplot(fat.pca);
summary(fat.pca);

## The minimum occurs at 17 components
## so for this dataset, maybe we should use full data
##
###(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

#beta coefficients
B <- coef(fat.pca, ncomp = ncompopt , intercept = FALSE)
B



# to obtain coefficients
coef2 <- predict(fat.pca, ncomp = ncompopt,
                 newdata =
as.matrix(rbind(rep.int(0,17),rep.int(1,17))));
c(coef2[1],coef(fat.pca, ncomp = 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);

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 <- mean( (ypred6test - fat1test$brozek)^2);

MSEtest <- c(MSEtest, MSEmod6test);

MSEtest;


## Fo this specific example, the optimal # of PC

##         ncompopt = 17, 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

### need to call the fuction "plsr"

#  library(pls)

fat.pls <- plsr(brozek ~ ., data = fat1train, validation="CV");


### (i) auto-select the optimal # of components of PLS

## choose the optimal # of components

mod7ncompopt <- which.min(fat.pls$validation$adj);

mod7ncompopt

## The opt # of components, it turns out to be 17 for this dataset,
```

```
##       and thus PLS also reduces to the full model!!!


# to obtain coefficients

coef2 <- predict(fat.pls, ncomp =  mod7ncompopt,

                 newdata =
as.matrix(rbind(rep.int(0,17),rep.int(1,17))));

c(coef2[1],coef(fat.pls, ncomp = mod7ncompopt))




# (ii) Training Error with the optimal choice of "mod7ncompopt"

# note that the prediction is from "fat.pls" with "mod7ncompopt"

ypred7train <- predict(fat.pls, ncomp = mod7ncompopt, newdata =
fat1train[2:18]);

MSEmod7train <- mean( (ypred7train - fat1train$brozek)^2);

MSEtrain <- c(MSEtrain, MSEmod7train);

MSEtrain;

## (iii) Testing Error with the optimal choice of "mod7ncompopt"

ypred7test <- predict(fat.pls, ncomp = mod7ncompopt, newdata =
fat1test[2:18]);

MSEmod7test <- mean( (ypred7test - fat1test$brozek)^2);

MSEtest <- c(MSEtest, MSEmod7test);

MSEtest;


MSEtrain


MSEtest


## For this specific dataset, PCR and PLS reduce to the full model!!!


########################################################################
#######################################
```

```
####Monte Carlo Cross-Validation algorithm that repeats the above
computation B = 100 times, compute and compare the “average”
performances of each model mentioned in



### saving the TE values for all models in all $B=100$ loops



##combine the train and test dataset


fatfull = rbind(fat1train, fat1test) ### combine to a full data set


### combine to a full data set
n = dim(fatfull)[1]; ## the total sample size
n1 = round(n/10); # training set sample size

B= 100; ### number of loops
TRERR = NULL; #Final training errors
TEALL = NULL; ### Final TE values
set.seed(7805); ### 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));
  fatfulltrain <- fatfull[-flag,];
  fatfulltest <- fatfull[flag,];

  ## The true Y response for the testing subset
  tempytrue    <- fatfulltest$brozek;
```

```
tempMSEtrain <- NULL;
tempMSEtest <- NULL;



### (1) Linear regression with all predictors (Full Model)
tempmodel1 <- lm(brozek ~ ., data = fatfulltrain);

## Model 1: Training error
tempMSEmod1train <-   mean( (resid(tempmodel1) )^2);
tempMSEtrain <- c(tempMSEtrain,  tempMSEmod1train);
tempMSEtrain;

# Model 1: testing error
temppred1a <- predict( tempmodel1, fatfulltest[,2:18]);
tempMSEmod1test <-   mean((temppred1a - tempytrue )^2);
tempMSEtest <- c(tempMSEtest, tempMSEmod1test);
tempMSEtest;

#####

### (2) Linear regression with the best subset model

library(leaps);
fullfat.leaps <- regsubsets(brozek ~ ., data= fatfulltrain, nbest=
100, really.big= TRUE, nvmax = 5); #model of size 100

## Record useful information from the output
```

```
  fullfat.models <- summary(fullfat.leaps)$which; #summary() command
outputs the best set of variables for each model size.

  fullfat.models.size <- as.numeric(attr(fullfat.models,
"dimnames")[[1]]);

  fullfat.models.rss <- summary(fullfat.leaps)$rss;#adjusted  R2


  # 2B: What is the best subset with k=5

  tempop2 <- which(fullfat.models.size == 5);

  tempflag2 <- tempop2[which.min(fullfat.models.rss[tempop2])];


  ## 2B(ii) Second, we can auto-find the best subset with k=5

  ##   this way will be useful when doing cross-validation

  tempmod2selectedmodel <- fullfat.models[tempflag2,];

  tempmod2Xname <-
paste(names(tempmod2selectedmodel)[tempmod2selectedmodel][-1],
collapse="+");

  tempmod2form <- paste ("brozek ~", tempmod2Xname);

  ## To auto-fit the best subset model with k=5 to the data

  tempmodel2 <- lm( as.formula(tempmod2form), data= fatfulltrain);



  # Model 2: training error

  tempMSEmod2train <- mean(resid(tempmodel2)^2);

  ## save this training error to the overall training error vector

  tempMSEtrain <- c(tempMSEtrain, tempMSEmod2train);

  tempMSEtrain;

  ## Model 2:  testing error

  temppred2 <- predict(tempmodel2, fatfulltest[,2:18]);

  tempMSEmod2test <-   mean((temppred2 - tempytrue)^2);

  tempMSEtest <- c(tempMSEtest, tempMSEmod2test);

  tempMSEtest;
```

```
#####

### (3) Linear regression with the stepwise variable selection

#install.packages("stargazer")
library(stargazer)

tempmodel1 <- lm( brozek ~ ., data = fatfulltrain);
tempmodel3  <- step(tempmodel1, direction= "both");

## Model 3: training  and  testing errors
tempMSEmod3train <- mean(resid(tempmodel3)^2);
tempMSEtrain <- c(tempMSEtrain, tempMSEmod3train);
tempMSEtrain;



temppred3 <- predict(tempmodel3, fatfulltest[,2:18]);
tempMSEmod3test <-   mean((temppred3 - tempytrue)^2);
tempMSEtest <- c(tempMSEtest, tempMSEmod3test);
tempMSEtest;

#####

### (4) Ridge regression (MASS: lm.ridge, mda: gen.ridge)
library(MASS);

fullfat.ridge <- lm.ridge( brozek ~ ., data = fatfulltrain, lambda=
seq(0,100,0.001));
```

```
  ## 4B(ii) Auto-find the "index" for the optimal lambda value for
Ridge regression

  ##        and auto-compute the corresponding testing and testing
error

  tempindexopt <-  which.min(fullfat.ridge$GCV);




  #converting scaled data to original numbers

  tempridge.coeffs = fullfat.ridge$coef[,tempindexopt]/
fullfat.ridge$scales;

  tempintercept = -sum(tempridge.coeffs  *
colMeans(fatfulltrain[,2:18] ))+ mean(fatfulltrain[,1]); #isolating
intercept - was not scaled, thus

  ## If you want to see the coefficients estimated from the Ridge
Regression

  ##   on the original data scale

  #c(tempintercept, tempridge.coeffs);



  ## Model 4 (Ridge): training errors

  tempyhat4train <- as.matrix(fatfulltrain[,2:18]) %*%
as.vector(tempridge.coeffs) + tempintercept; #%*% is matrix
multiplication. For matrix multiplication, you need an m x n matrix
times an n x p matrix.

  tempMSEmod4train <- mean((tempyhat4train - fatfulltrain$brozek)^2);

  tempMSEtrain <- c(tempMSEtrain, tempMSEmod4train);

  tempMSEtrain



  ## Model 4 (Ridge):  testing errors in the subset "test"

  temppred4test <- as.matrix(fatfulltest[,2:18]) %*%
as.vector(tempridge.coeffs) + tempintercept;

  tempMSEmod4test <-  mean((temppred4test - tempytrue)^2);

  tempMSEtest <- c(tempMSEtest, tempMSEmod4test);

  tempMSEtest;
```

```
   #####
  ## Model (5): LASSO


  library(lars)
  fullfat.lars <- lars( as.matrix(fatfulltrain[,2:18]),
fatfulltrain[,1], type= "lasso", trace= TRUE);


  ## 5B: choose the optimal \lambda value that minimizes Mellon's Cp
criterion
  Cp1  <- summary(fullfat.lars)$Cp;
  index1 <- which.min(Cp1);


  #lasso.lambda <- fullfat.lars$lambda[index1]
  lasso.lambda <- coef(fullfat.lars)[index1,]
  #tempcoef.lars1 <- predict(fullfat.lars, s=lasso.lambda,
type="coef", mode="lambda")


  ##   the third way is to get the coefficients via prediction
function


  lasso.lambda <-    fullfat.lars$beta[index1]
  coef.lars1 <- predict(fullfat.lars,
                        s=lasso.lambda, type="coeff", mode="lambda")



  #round(c(coef.lars1$fit[1],coef(fat.lars)[index1,]),4)




  ## Model 5:  training error for lasso
  ##
```

```
  temppred5train  <- predict(fullfat.lars,
as.matrix(fatfulltrain[,2:18]), s=lasso.lambda, type="fit",
mode="lambda");

  tempyhat5train <- temppred5train$fit;

  tempMSEmod5train <- mean((tempyhat5train - fatfulltrain$brozek)^2);

  tempMSEtrain <- c(tempMSEtrain, tempMSEmod5train);

  tempMSEtrain


  ##

  ## Model 5:  training error for lasso

  temppred5test <- predict(fullfat.lars,
as.matrix(fatfulltest[,2:18]), s=lasso.lambda, type="fit",
mode="lambda");

  tempyhat5test <- temppred5test$fit;

  tempMSEmod5test <- mean((tempyhat5test - fatfulltest$brozek)^2);

  tempMSEtest <- c(tempMSEtest, tempMSEmod5test);

  tempMSEtest;


  #####


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


  ##

  library(pls)

  ## 6B(i): call the pcr function to run the linear regression on all
possible # of PCs.

  ##

  fullfat.pca <- pcr(brozek~., data=fatfulltrain, validation="CV");

  ##

  ## 6B(ii) You can have some plots to see the effects on the number
of PCs

  #validationplot(fullfat.pca);
```

```
  #summary(fullfat.pca);

  #

  ### 6B(iii) How to auto-select # of components

  ##     automatically optimazation by PCR based on the cross-
validation

  ##     It chooses the optimal # of components

  tempncompopt <- which.min(fullfat.pca$validation$adj);

  ##

  ## 6B(iv) Training Error with the optimal choice of PCs

  tempypred6train <- predict(fullfat.pca, ncomp = tempncompopt,
newdata = fatfulltrain[2:18]);

  tempMSEmod6train <- mean( (tempypred6train -
fatfulltrain$brozek)^2);

  tempMSEtrain <- c(tempMSEtrain, tempMSEmod6train);

  tempMSEtrain;

  ## 6B(v) Testing Error with the optimal choice of PCs

  tempypred6test <- predict(fullfat.pca, ncomp = tempncompopt, newdata
= fatfulltest[2:18]);

  tempMSEmod6test <- mean( (tempypred6test - fatfulltest$brozek)^2);

  tempMSEtest <- c(tempMSEtest, tempMSEmod6test);

  tempMSEtest;


  ######


  ### Model 7. Partial Least Squares (PLS) Regression


  #  library(pls)

  fullfat.pls <- plsr(brozek ~ ., data = fatfulltrain,
validation="CV");


  ### 7(i) auto-select the optimal # of components of PLS

  ## choose the optimal # of components
```

```
  tempmod7ncompopt <- which.min(fullfat.pls$validation$adj);



  # 7(ii) Training Error with the optimal choice of "mod7ncompopt"

  # note that the prediction is from "fat.pls" with "mod7ncompopt"

  tempypred7train <- predict(fullfat.pls, ncomp =  tempmod7ncompopt,
newdata = fatfulltrain[2:18]);

  tempMSEmod7train <- mean( ( tempypred7train -
fatfulltrain$brozek)^2);

  tempMSEtrain <- c( tempMSEtrain,  tempMSEmod7train);

  tempMSEtrain;


  ## 7(iii) Testing Error with the optimal choice of "mod7ncompopt"

  tempypred7test <- predict(fullfat.pls, ncomp =  tempmod7ncompopt,
newdata = fatfulltest[2:18]);

  tempMSEmod7test <- mean( ( tempypred7test - fatfulltest$brozek)^2);

  tempMSEtest <- c( tempMSEtest,  tempMSEmod7test);

  tempMSEtest;



  TRERR = rbind (TRERR, tempMSEtrain);

  TEALL = rbind (TEALL, tempMSEtest);


}


dim(TEALL);

dim(TRERR);


### if you want, you can change the column name of TEALL

colnames(TRERR) <- c("OLS (all)", "Best Subset (k=5)", "AIC", "Ridge",
"LASSO", "PCR", "PLS");
```

```
colnames(TEALL) <- c("OLS (all)", "Best Subset (k=5)", "AIC", "Ridge",
"LASSO", "PCR", "PLS");

## You can report the sample mean and sample variances for the seven
models



##TRAINING ERROR AVG MEAN AND VARIANCES

apply(TRERR, 2, mean);

apply(TRERR, 2, var);


#TESTING ERROR AVG MEAN AND VARIANCES


apply(TEALL, 2, mean);

apply(TEALL, 2, var)




## statistical difference testing

T1=t.test(TEALL[,5],TEALL[,7],paired=T)

T2=t.test(TEALL[,5],TEALL[,6],paired=T)

T3=t.test(TEALL[,5],TEALL[,4],paired=T)

T4=t.test(TEALL[,5],TEALL[,3],paired=T)

T5=t.test(TEALL[,5],TEALL[,2],paired=T)

T6=t.test(TEALL[,5],TEALL[,1],paired=T)

c(T6$p.value,T5$p.value,T4$p.value,T3$p.value,T2$p.value,T1$p.value)



W1=wilcox.test(TEALL[,5],TEALL[,7],paired=T)

W2=wilcox.test(TEALL[,5],TEALL[,6],paired=T)

W3=wilcox.test(TEALL[,5],TEALL[,4],paired=T)

W4=wilcox.test(TEALL[,5],TEALL[,3],paired=T)
```

```
W5=wilcox.test(TEALL[,5],TEALL[,2],paired=T)
W6=wilcox.test(TEALL[,5],TEALL[,1],paired=T)
c(W6$p.value,W5$p.value,W4$p.value,W3$p.value,W2$p.value,W1$p.value)


### END ###
```