midterm final

Aditya Nanduri

11/5/2020

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

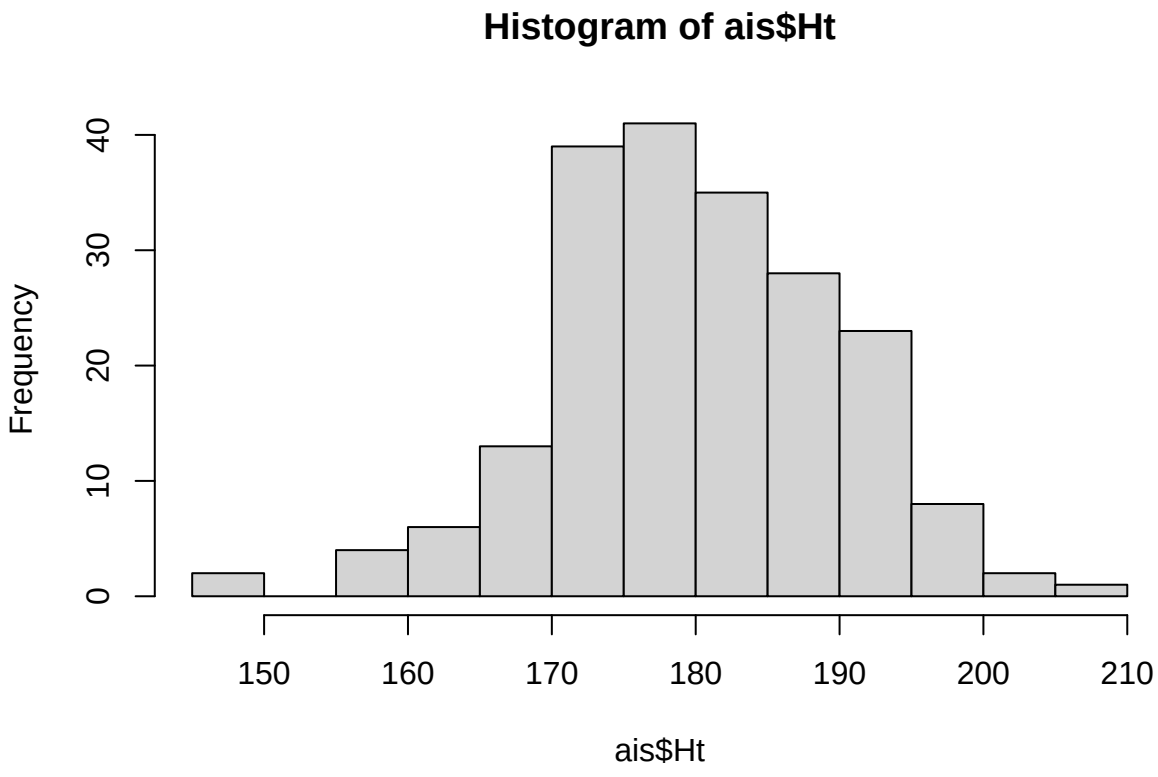
Data loading

```
ais <- read.csv("ais.csv")
```

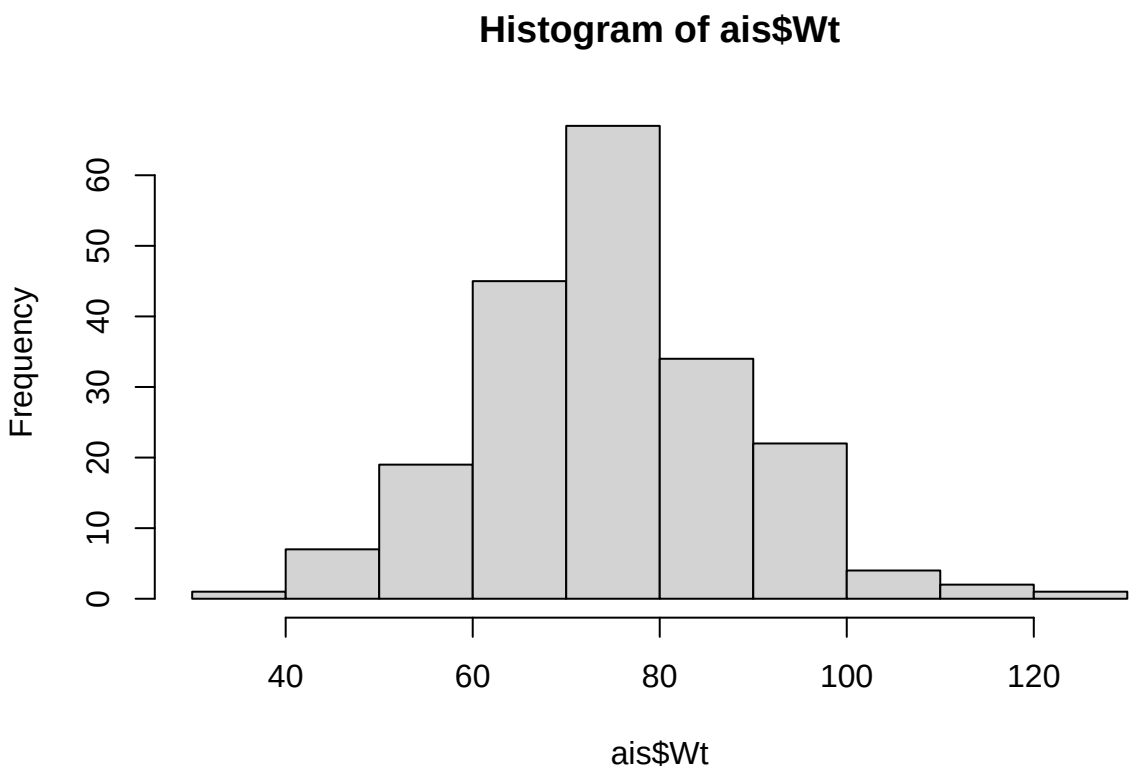
Data exploration

Analyse independent variables distribution Convert variable “sex” to factor variable

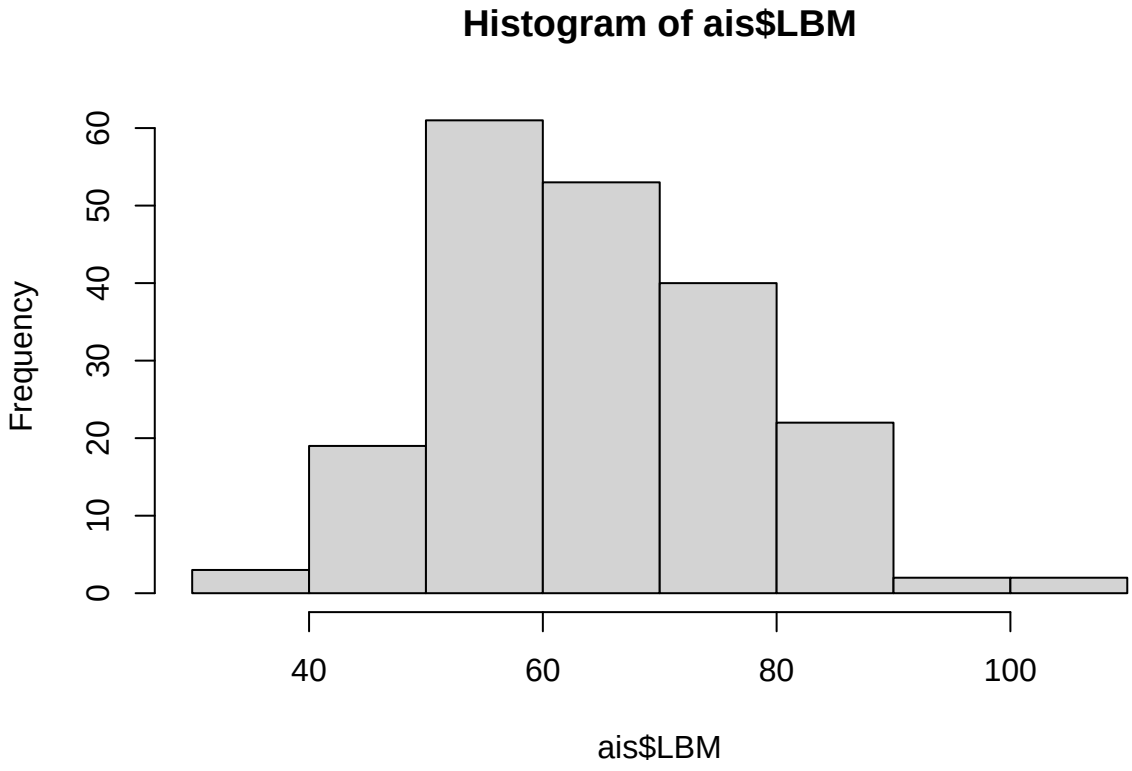
```
ais$Sex <- as.factor(ais$Sex)
ais$Sport <- as.factor(ais$Sport)
hist(ais$Ht)
```



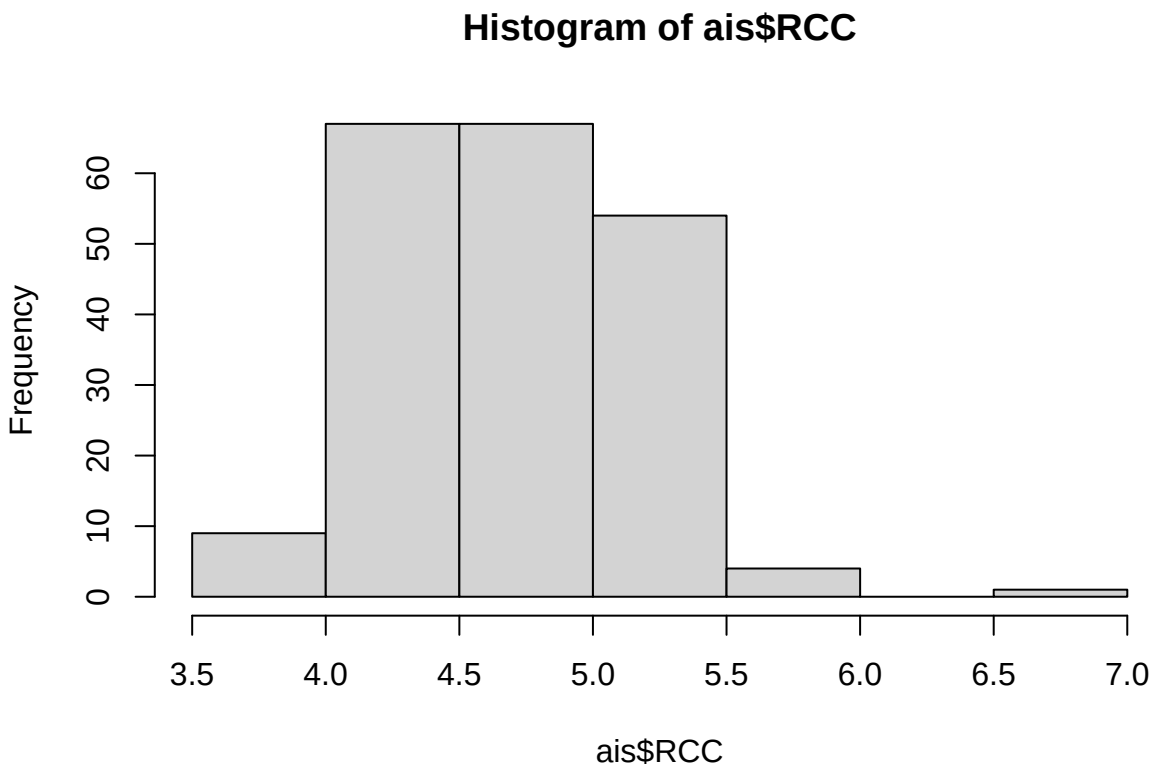
```r
hist(ais$Wt)
```



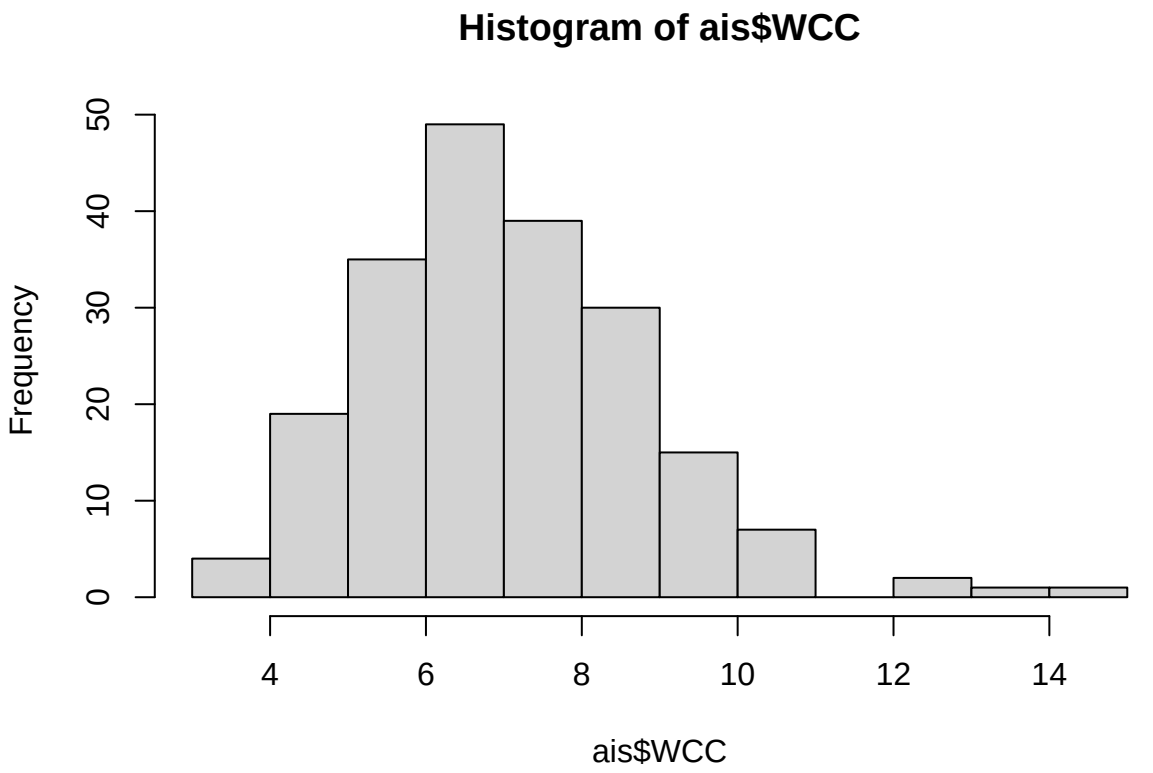

```r
hist(ais$LBM)
```



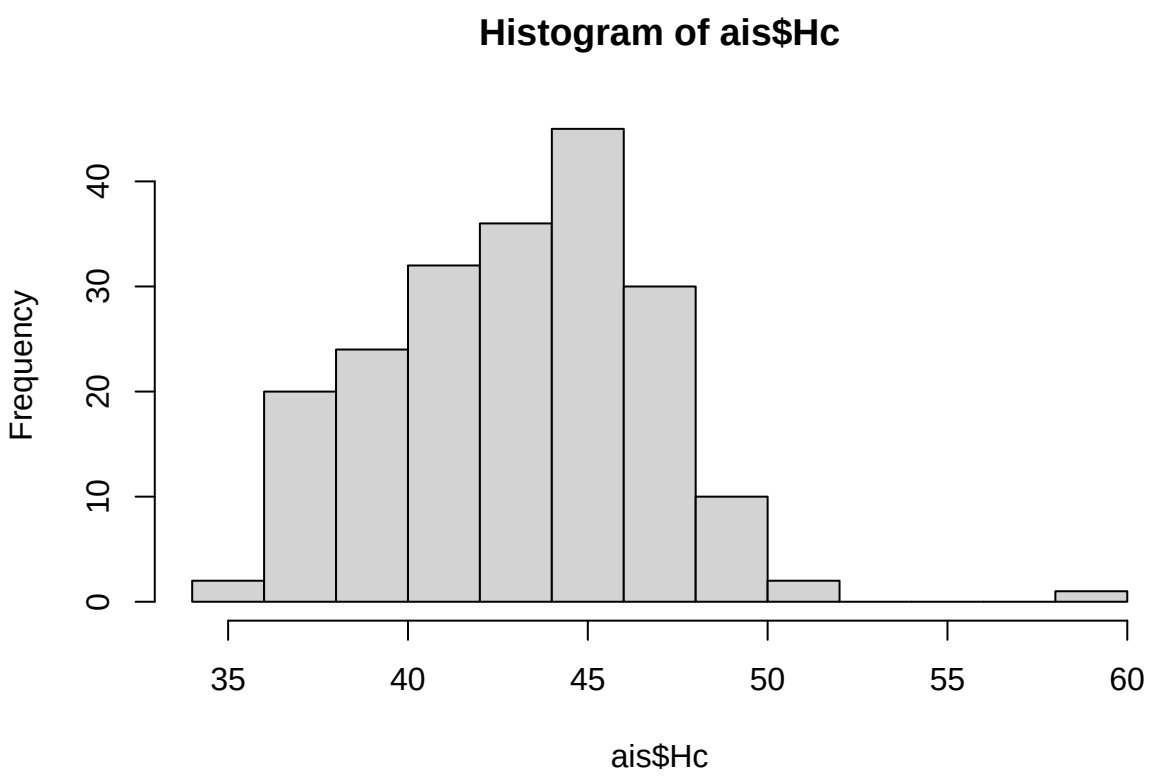
```r
hist(ais$RCC)
```



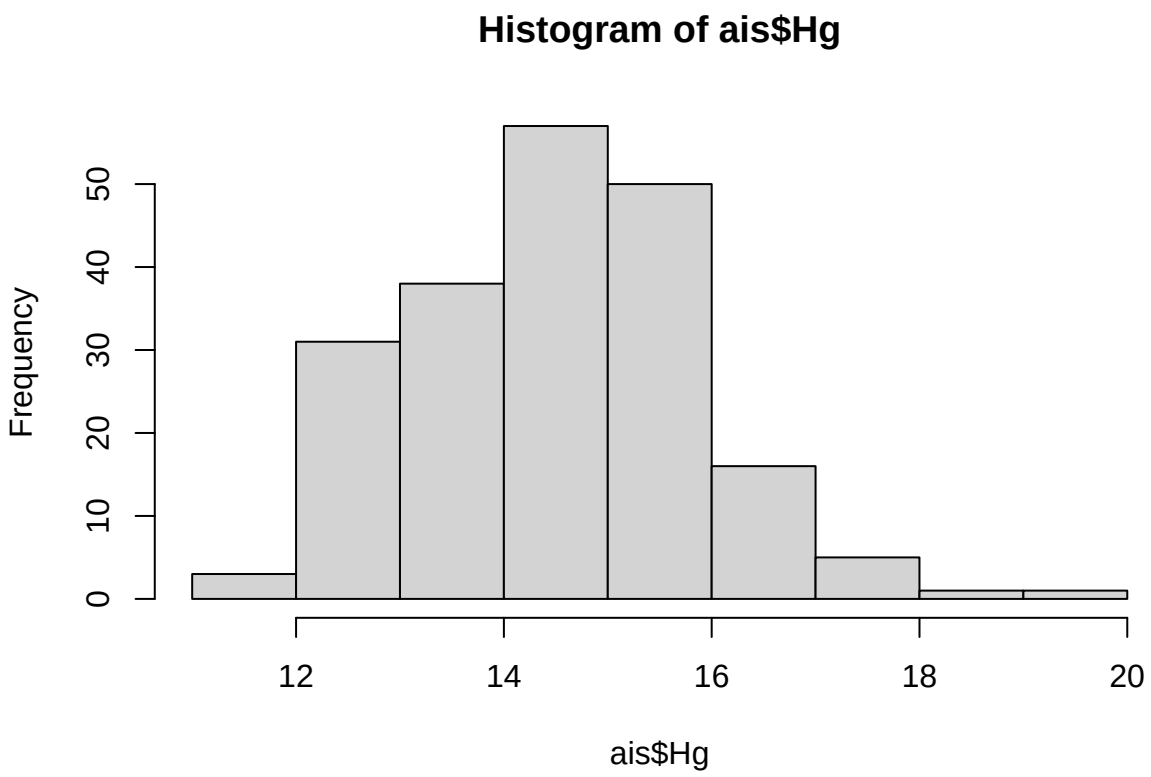

```r
hist(ais$WCC)
```



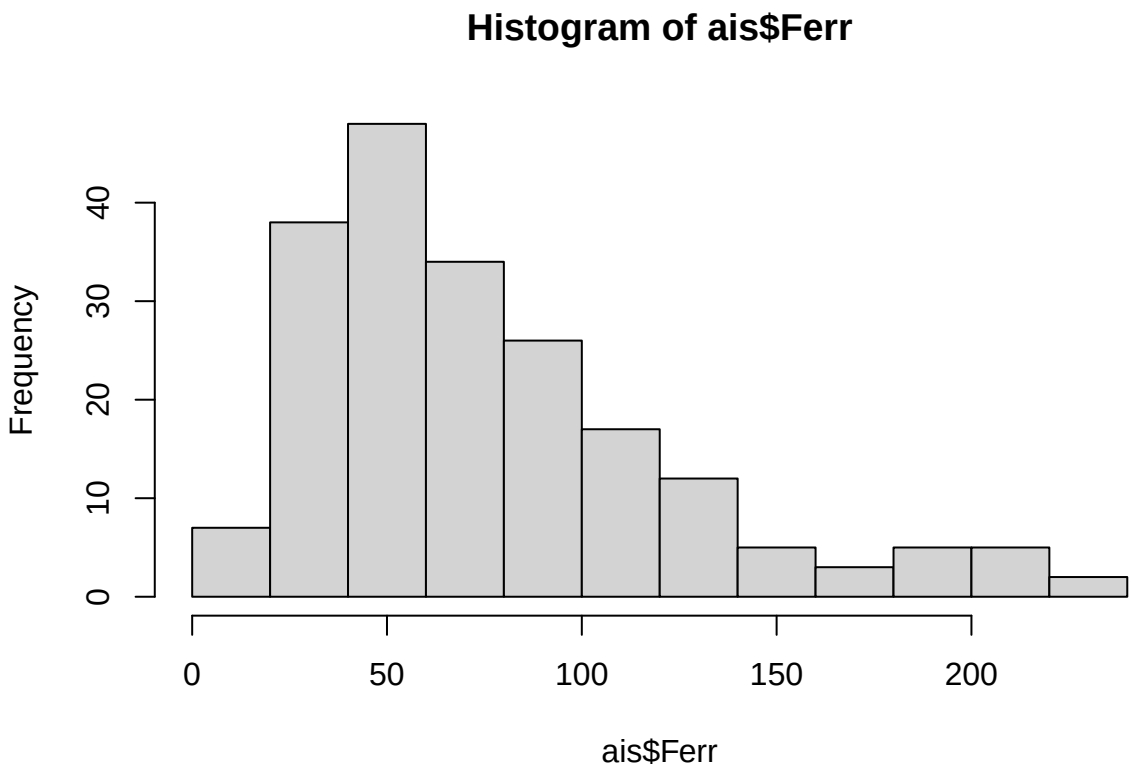
```
hist(ais$Hc)
```



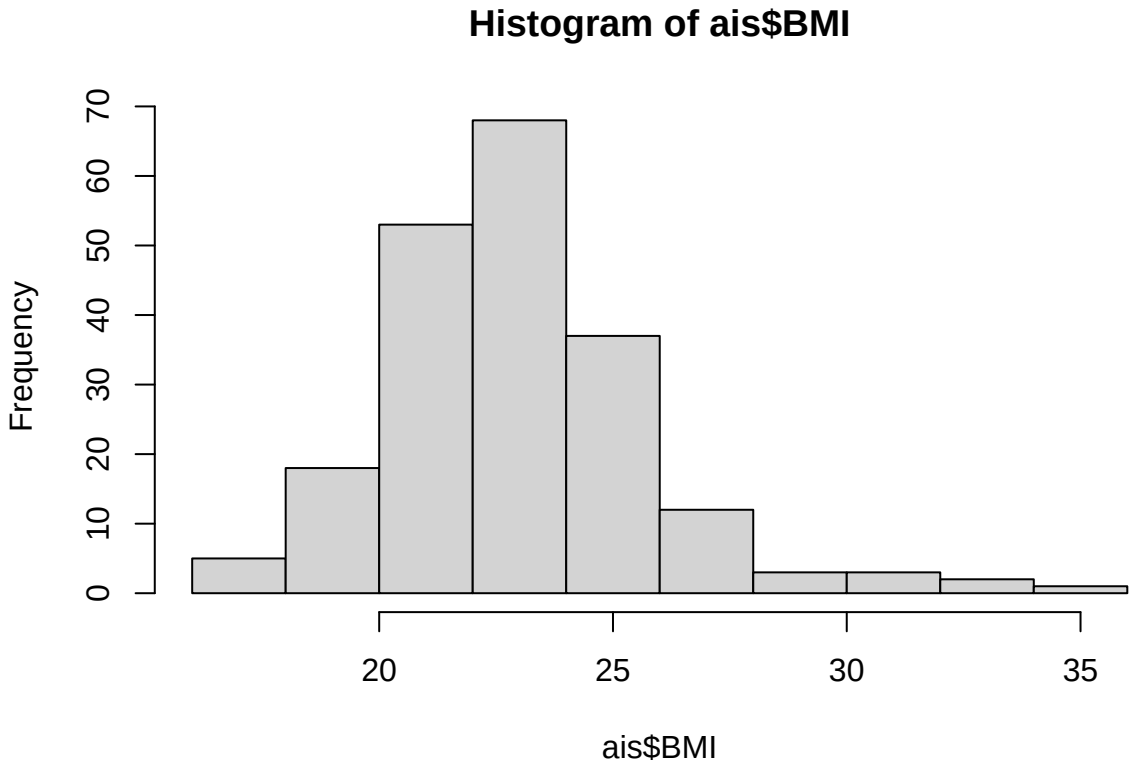

```
hist(ais$Hg)
```



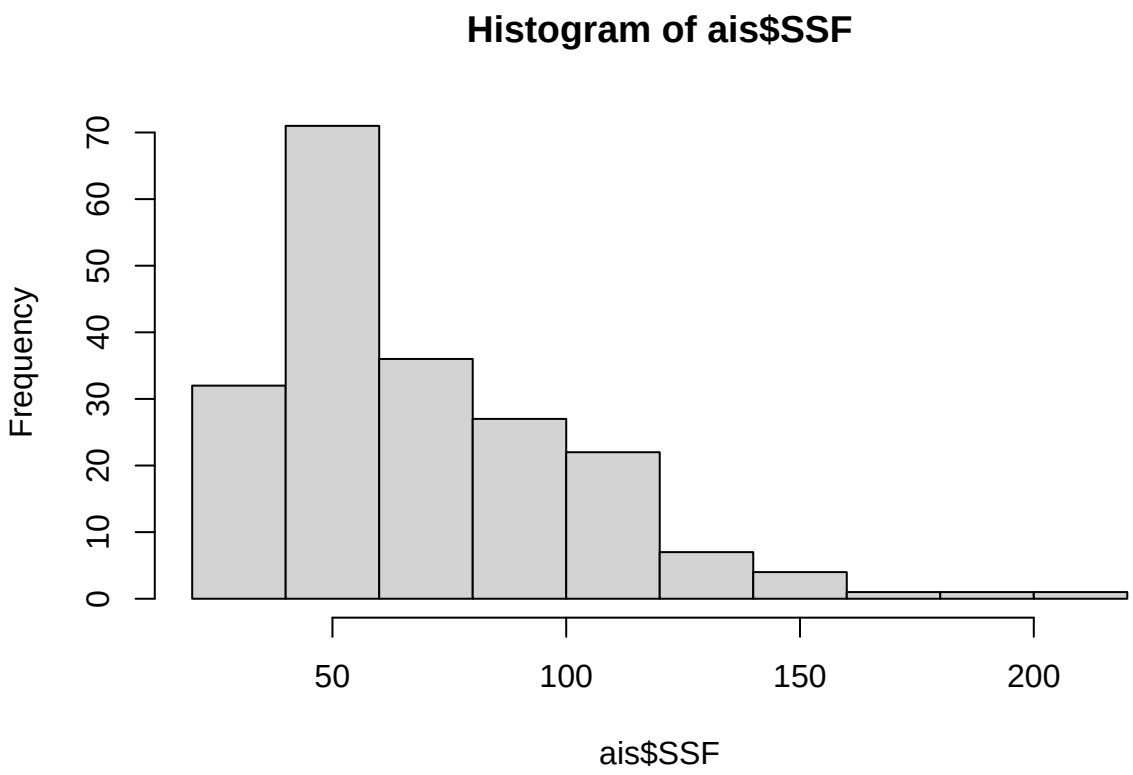
```r
hist(ais$Ferr)
```




```r
hist(ais$BMI)
```



```r
hist(ais$SSF)
```




```r
cor(ais[,-c(1,3)])
```

```
##               Bfat          Ht            Wt        LBM        RCC        WCC
## Bfat  1.0000000000 -0.18802168 -0.0001618851 -0.3618504 -0.4935123 0.1081136
## Ht   -0.1880216785  1.00000000  0.7809062893  0.8021192  0.3588540 0.0769577
## Wt   -0.0001618851  0.78090629  1.0000000000  0.9309040  0.4037426 0.1558443
## LBM  -0.3618504448  0.80211920  0.9309040500  1.0000000  0.5509751 0.1027337
## RCC  -0.4935122578  0.35885396  0.4037425515  0.5509751  1.0000000 0.1470978
## WCC   0.1081136062  0.07695770  0.1558443259  0.1027337  0.1470978 1.0000000
## Hc   -0.5324490851  0.37119150  0.4236989441  0.5833748  0.9249639 0.1533331
## Hg   -0.5315220600  0.35232222  0.4552549388  0.6109861  0.8887998 0.1347209
## Ferr -0.1833862133  0.12325468  0.2736859423  0.3175809  0.2508655 0.1319726
## BMI   0.1875577578  0.33709720  0.8459550779  0.7138581  0.2994711 0.1770323
## SSF   0.9630168490 -0.07125283  0.1542266552 -0.2077489 -0.4030039 0.1371308
##              Hc         Hg       Ferr       BMI         SSF
## Bfat -0.5324491 -0.5315221 -0.1833862 0.1875578  0.96301685
## Ht    0.3711915  0.3523222  0.1232547 0.3370972 -0.07125283
## Wt    0.4236989  0.4552549  0.2736859 0.8459551  0.15422666
## LBM   0.5833748  0.6109861  0.3175809 0.7138581 -0.20774895
## RCC   0.9249639  0.8887998  0.2508655 0.2994711 -0.40300393
## WCC   0.1533331  0.1347209  0.1319726 0.1770323  0.13713081
## Hc    1.0000000  0.9507567  0.2582395 0.3205271 -0.44913507
## Hg    0.9507567  1.0000000  0.3083911 0.3825241 -0.43542899
## Ferr  0.2582395  0.3083911  1.0000000 0.3025561 -0.10824311
## BMI   0.3205271  0.3825241  0.3025561 1.0000000  0.32111644
## SSF  -0.4491351 -0.4354290 -0.1082431 0.3211164  1.00000000
```

Data Preparation

Since Ht and Wt data are slightly skewed, applying log to make it normal

```
ais$Ht<-log(ais$Ht)
ais$Wt<-log(ais$Wt)
ais<-ais[,-c(1)]
```

MODELING SELECTION PROCESS:

Multiple Linear and Penalised regression

```
aispenaliseddata <-ais
n= dim(aispenaliseddata)[1]

names(aispenaliseddata)
```

```
##  [1] "Bfat" "Sex"  "Ht"   "Wt"   "LBM"  "RCC"  "WCC"  "Hc"   "Hg"   "Ferr"
## [11] "BMI"  "SSF"
```

```
# specify models to consider
#model list specification
LinModel1 = (Bfat ~ Ht)
LinModel2 = (Bfat ~ Ht+Wt)
LinModel3 = (Bfat ~ Ht+Wt+LBM)
LinModel4 = (Bfat ~ Ht+Wt+LBM+RCC)
LinModel5 = (Bfat ~ Ht+Wt+LBM+RCC+WCC)
LinModel6 = (Bfat ~ Ht+Wt+LBM+RCC+WCC+Hc)
LinModel7 = (Bfat ~ Ht+Wt+LBM+RCC+WCC+Hc+Hg)
LinModel8 = (Bfat ~ Ht+Wt+LBM+RCC+WCC+Hc+Hg+Ferr)
LinModel9 = (Bfat ~ Ht+Wt+LBM+RCC+WCC+Hc+Hg+Ferr+BMI)
LinModel10 = (Bfat ~ Ht+Wt+LBM+RCC+WCC+Hc+Hg+Ferr+BMI+SSF)


allLinModels = list(LinModel1,LinModel2,LinModel3,LinModel4,LinModel5,LinModel6,LinModel7,LinModel8,Lin
nLinmodels = length(allLinModels)

library(glmnet)  # use RR and LASSO modeling commands from package glmnet
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.0-2
```

```
# RR model specification and number
lambdalistRR = c(0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5)  # specifies RR models
nRRmodels = length(lambdalistRR)
# LASSO model specification and number
lambdalistLASSO = c(0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5)  # specifies LASSO m
nLASSOmodels = length(lambdalistLASSO)

# Elasticnet model specification and number
lambdalistENET = c(0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5)  # specifies LASSO mo
nENETmodels = length(lambdalistENET)

nmodels = nLinmodels+nRRmodels+nLASSOmodels+nENETmodels

#specify the data set used to perform the model selection
```

```r
fulldata.in = aispenaliseddata
# set seed for randomizing CV fold selection
set.seed(8, sample.kind = "Rounding")
```

```
## Warning in set.seed(8, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

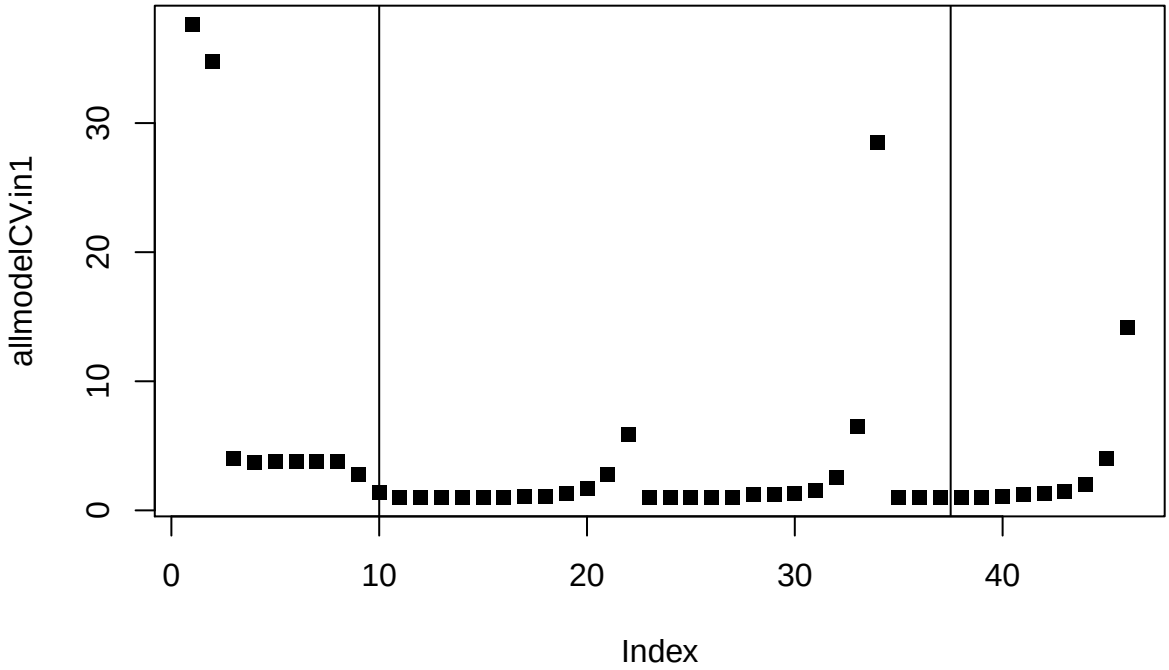
```r
############################
## Full modeling process ##
############################

# we begin setting up the model-fitting process to use notation that will be
# useful later, "in"side a validation
n.in = dim(fulldata.in)[1]
x.in = model.matrix(Bfat~.,data=fulldata.in)
y.in = fulldata.in[,1]
# number folds and groups for (inner) cross-validation for model-selection
k.in = 10
   #produce list of group labels
groups.in = c(rep(1:k.in,floor(n.in/k.in))); if(floor(n.in/k.in) != (n.in/k.in)) groups.in = c(groups.in
cvgroups.in = sample(groups.in,n.in)  #orders randomly, with seed (8)
# table(cvgroups.in)  # check correct distribution
allmodelCV.in1 = rep(NA,nmodels) #place-holder for results
```

```r
##### cross-validation for model selection ##### reference - Lesson 2

# since linear regression does not have any automatic CV output,
# set up storage for predicted values from the CV splits, across all linear models
allpredictedCV.in = matrix(rep(NA,n.in*nLinmodels),ncol=nLinmodels)

#cycle through all folds:  fit the model to training data, predict test data,
# and store the (cross-validated) predicted values
for (i in 1:k.in)  {
  train.in = (cvgroups.in != i)
  test.in = (cvgroups.in == i)
  #fit each of the linear regression models on training, and predict the test
  for (m in 1:nLinmodels) {
    lmfitCV.in = lm(formula = allLinModels[[m]],data=aispenaliseddata,subset=train.in)
    allpredictedCV.in[test.in,m] = predict.lm(lmfitCV.in,fulldata.in[test.in,])
  }
}
# compute and store the CV(10) values
for (m in 1:nLinmodels) {
  allmodelCV.in1[m] = mean((allpredictedCV.in[,m]-fulldata.in$Bfat)^2)
}
```

```r
##### cross-validation for model selection ##### reference - Lesson 5

#RR cross-validation - uses internal cross-validation function
cvRRglm.in = cv.glmnet(x.in, y.in, lambda=lambdalistRR, alpha = 0, nfolds=k.in, foldid=cvgroups.in)


#LASSO cross-validation - uses internal cross-validation function
cvLASSOglm.in = cv.glmnet(x.in, y.in, lambda=lambdalistLASSO, alpha = 1, nfolds=k.in, foldid=cvgroups.in
```

```r
#ENET  cross-validation - uses internal cross-validation function
cvENETglm.in = cv.glmnet(x.in, y.in, lambda=lambdalistENET, alpha = 0.5, nfolds=k.in, foldid=cvgroups.in

# store CV(10) values, in same numeric order as lambda, in storage spots for CV values
allmodelCV.in1[(1:nRRmodels)+nLinmodels] = cvRRglm.in$cvm[order(cvRRglm.in$lambda)]
# store CV(10) values, in same numeric order as lambda, in storage spots for CV values
allmodelCV.in1[(1:nLASSOmodels)+nRRmodels+nLinmodels] = cvLASSOglm.in$cvm[order(cvLASSOglm.in$lambda)]

# store CV(10) values, in same numeric order as lambda, in storage spots for CV values
allmodelCV.in1[(1:nENETmodels)+nLASSOmodels+nRRmodels+nLinmodels] = cvENETglm.in$cvm[order(cvENETglm.in$
```

```r
# visualize CV(10) values across all methods
plot(allmodelCV.in1,pch=15); abline(v=c(nLinmodels,nLASSOmodels+.5+nRRmodels+.5+nENETmodels+.5))
```




```r
bestmodel.in = (1:nmodels)[order(allmodelCV.in1)[1]]  # actual selection
# state which is best model and minimum CV(10) value
bestmodel.in; min(allmodelCV.in1)
```

```
## [1] 26
```

```
## [1] 0.9993979
```

```r
### finally, fit the best model to the full (available) data ###
if (bestmodel.in <= nLinmodels) {  # then best is one of linear models
  bestfit = lm(formula = allLinModels[[bestmodel.in]],data=fulldata.in)  # fit on all available data
  bestcoef = coef(bestfit)
} else if (bestmodel.in <= nRRmodels+nLinmodels) {  # then best is one of RR models
  bestlambdaRR = (lambdalistRR)[bestmodel.in-nLinmodels]
  bestfit = glmnet(x.in, y.in, alpha = 0,lambda=lambdalistRR)  # fit the model across possible lambda
  bestcoef = coef(bestfit, s = bestlambdaRR) # coefficients for the best model fit
} else if (bestmodel.in <= nRRmodels+nLinmodels+nLASSOmodels) {  # then best is one of LASSO models
  bestlambdaLASSO = (lambdalistLASSO)[bestmodel.in-nLinmodels-nRRmodels]
  bestfit = glmnet(x.in, y.in, alpha = 1,lambda=lambdalistLASSO)  # fit the model across possible lambd
```

```
  bestcoef = coef(bestfit, s = bestlambdaLASSO) # coefficients for the best model fit
}else {  # then best is one of Enet models
  bestlambdaEnet = (lambdalistENET)[bestmodel.in-nLinmodels-nRRmodels-nLASSOmodels]
  bestfit = glmnet(x.in, y.in, alpha = .5,lambda=lambdalistENET)  # fit the model across possible lambd
  bestcoef = coef(bestfit, s = lambdalistENET) # coefficients for the best model fit
}

##############################
## End of modeling process ##
##############################
```

```
# summary of best model selected
selectmodelsummary = list(selectmodel = bestmodel.in, selectfit = bestfit,
                          selectcoef = bestcoef)
selectmodelsummary  # in order to recall the final selected fit after any validation
```

```
## $selectmodel
## [1] 26
##
## $selectfit
##
## Call:  glmnet(x = x.in, y = y.in, alpha = 1, lambda = lambdalistLASSO)
##
##    Df  %Dev Lambda
## 1   1 27.17  5.000
## 2   2 83.27  2.000
## 3   2 93.44  1.000
## 4   2 95.98  0.500
## 5   2 96.70  0.200
## 6   3 96.80  0.100
## 7   5 96.91  0.050
## 8   8 97.56  0.020
## 9   8 97.66  0.010
## 10 11 97.71  0.005
## 11 11 97.73  0.002
## 12 11 97.73  0.001
##
## $selectcoef
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                            1
## (Intercept) -46.599627289
## (Intercept)   .
## Sex1          2.757998771
## Ht            .
## Wt           14.794193487
## LBM          -0.233027071
## RCC           .
## WCC           0.004120828
## Hc            0.013652405
## Hg            .
## Ferr          0.001195071
## BMI           0.038545629
## SSF           0.125325819
```

Among Linear and Penalised regression models (RR , Lasso and Elastic net) Lasso regression is the best

model with cross validation error of 0.9993979

Random Forest

we have choosen is Random forest as second method since it works best when variables have collinear relations and one of the variable has more inportance than others

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
set.seed(8, sample.kind = "Rounding")
```

```
## Warning in set.seed(8, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
n = 202
k = 10
groups = c(rep(1:k,floor(n/k)),1:(n-floor(n/k)*k))
cvgroups = sample(groups,n)
rf.predictmlrfitcv = rep(0,n)
for (i in 1:k){
  groupi = (cvgroups == i)

  ais.rf = randomForest(Bfat~., data=ais[!groupi,], mtry = 4, importance = T)
  rf.predictmlrfitcv[groupi] = predict(ais.rf,newdata = ais[groupi,])

}

rfcv1 = sum((rf.predictmlrfitcv-ais$Bfat)^2)/n

print(rfcv1)
```

```
## [1] 2.106099
```

Random forest cross validation error is 2.106099 vs best among regrssion cv error(Lasso) is 0.997455

3.0 - MODEL ASSESSMENT FOR HONEST PREDICTION

As a final step in this analysis, we will preform double 10-fold cross-validation to assess an honest expectation of error rate for this model. We first split the enter dataset into 10 folds. One is held as an "outer" test, and the other 9 folds are sent into an "inner" modeling selection process where each of these models are also being fit and tested using a CV10 process. The best model from the "inner" fitting process is selected and used to predict the fold that was held back from the "outer" split. The predicted classes are stored, and once the "outer" CV10 process is complete, a confusion matrix is created and the overall misclassification rate is calculated.

```
aispenaliseddata <-ais
n= dim(aispenaliseddata)[1]

names(aispenaliseddata)
```

```
##  [1] "Bfat" "Sex"  "Ht"   "Wt"   "LBM"  "RCC"  "WCC"  "Hc"   "Hg"   "Ferr"
## [11] "BMI"  "SSF"
```

```
# specify models to consider
#model list specification
```

```r
library(glmnet)  # use RR and LASSO modeling commands from package glmnet

# LASSO model specification and number
lambdalistLASSO = c(0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5)  # specifies LASSO m
nLASSOmodels = length(lambdalistLASSO)

nrfmodels = 1

nmodels = nLASSOmodels + nrfmodels

#####################################################################
##### Double cross-validation for modeling-process assessment #####
#####################################################################

##### model assessment OUTER shell #####
fulldata.out = aispenaliseddata
k.out = 10
n.out = dim(fulldata.out)[1]
#define the cross-validation splits
groups.out = c(rep(1:k.out,floor(n.out/k.out))); if(floor(n.out/k.out) != (n.out/k.out)) groups.out = c
set.seed(8, sample.kind = "Rounding")
cvgroups.out = sample(groups.out,n.out)  #orders randomly, with seed (8)

# set up storage for predicted values from the double-cross-validation
allpredictedCV.out = rep(NA,n.out)
# set up storage to see what models are "best" on the inner loops
allbestmodels = rep(NA,k.out)

# loop through outer splits
for (j in 1:k.out)  {  #be careful not to re-use loop indices
  groupj.out = (cvgroups.out == j)
  traindata.out = aispenaliseddata[!groupj.out,]
  trainx.out = model.matrix(Bfat~.,data=traindata.out)
  trainy.out = traindata.out[,1]
  validdata.out = aispenaliseddata[groupj.out,]
  validx.out = model.matrix(Bfat~.,data=validdata.out)
  validy.out = validdata.out[,1]

#specify the data set used to perform the model selection
fulldata.in = traindata.out
# set seed for randomizing CV fold selection


x.out  = model.matrix(Bfat~.,data=validdata.out)
y.out = validdata.out[,1]

############################
## Full modeling process ##
############################

# we begin setting up the model-fitting process to use notation that will be
# useful later, "in"side a validation
n.in = dim(fulldata.in)[1]
```

```
x.in = model.matrix(Bfat~.,data=fulldata.in)
y.in = fulldata.in[,1]
# number folds and groups for (inner) cross-validation for model-selection
k.in = 10
   #produce list of group labels
groups.in = c(rep(1:k.in,floor(n.in/k.in))); if(floor(n.in/k.in) != (n.in/k.in)) groups.in = c(groups.i
cvgroups.in = sample(groups.in,n.in)  #orders randomly, with seed (7)
# table(cvgroups.in)  # check correct distribution
allmodelCV.in = rep(NA,nmodels) #place-holder for results
#LASSO cross-validation - uses internal cross-validation function
cvLASSOglm.in = cv.glmnet(x.in, y.in, lambda=lambdalistLASSO, alpha = 1, nfolds=k.in, foldid=cvgroups.i

rf.predictmlrfitcv = rep(0,n.in)
for (i in 1:k.in){
  groupi = (cvgroups.in == i)
  ais.rf = randomForest(Bfat~., data=aispenaliseddata[!groupi,], mtry = 4, importance = T)
  rf.predictmlrfitcv[groupi] = predict(ais.rf,newdata = aispenaliseddata[groupi,])
}

rfcv1 = sum((rf.predictmlrfitcv-aispenaliseddata$Bfat)^2)/n.in

# store CV(10) values, in same numeric order as lambda, in storage spots for CV values
allmodelCV.in[(1:nLASSOmodels)] = cvLASSOglm.in$cvm[order(cvLASSOglm.in$lambda)]
allmodelCV.in[(1:nrfmodels)+nLASSOmodels] = rfcv1
# store CV(10) values, in same numeric order as lambda, in storage spots for CV values
bestmodel.in = (1:nmodels)[order(allmodelCV.in)[1]]  # actual selection
# state which is best model and minimum CV(10) value
bestmodel.in; min(allmodelCV.in)
### finally, fit the best model to the full (available) data ###
if (bestmodel.in <= nLASSOmodels) {  # then best is one of linear models
  bestlambdaLASSO = (lambdalistLASSO)[bestmodel.in]
  bestfit = glmnet(x.in, y.in, alpha = 1,lambda=lambdalistLASSO)  # fit the model across possible lambd
  bestcoef = coef(bestfit, s = bestlambdaLASSO) # coefficients for the best model fit
} else{  # then best is one of RR models
  bestfit = randomForest(Bfat~., data=aispenaliseddata, mtry = 4, importance = T)  # fit on all availab
  bestcoef = coef(bestfit)
}

##############################
## End of modeling process ##
##############################
  ###   :   :   :   :   :   :  ###
  ### resulting in bestmodel.in ###

  allbestmodels[j] = bestmodel.in

  if (bestmodel.in <= nLASSOmodels) {  # then best is one of linear models
    LASSOfit = glmnet(x.out, y.out, alpha = 1,lambda=lambdalistLASSO)
    allpredictedCV.out[groupj.out] = predict(LASSOfit,newx=x.out,s=.5)


  } else  {  # then best is one of Random forest
    allpredictedCV.out[groupj.out] =  predict(ais.rf,newdata = validdata.out)
```

```
  }
}

# for curiosity, we can see the models that were "best" on each of the inner splits
allbestmodels
```

```
##  [1] 4 4 3 4 4 4 4 3 4 3
```

```
#assessment
y.out = fulldata.out$Bfat
CV.out = sum((allpredictedCV.out-y.out)^2)/n.out; CV.out
```

```
## [1] 1.321632
```

```
R2.out = 1-sum((allpredictedCV.out-y.out)^2)/sum((y.out-mean(y.out))^2); R2.out
```

```
## [1] 0.9653336
```

```
bestfit
```

```
##
## Call:  glmnet(x = x.in, y = y.in, alpha = 1, lambda = lambdalistLASSO)
##
##    Df  %Dev Lambda
## 1   1 28.96  5.000
## 2   2 83.93  2.000
## 3   2 93.78  1.000
## 4   2 96.24  0.500
## 5   2 96.93  0.200
## 6   2 97.03  0.100
## 7   7 97.09  0.050
## 8   8 97.69  0.020
## 9   9 97.80  0.010
## 10 10 97.85  0.005
## 11 11 97.86  0.002
## 12 11 97.87  0.001
```

```
allmodelCV.in
```

```
##  [1]  0.9300669  0.9290725  0.9290342  0.9368235  0.9716221  1.1932793
##  [7]  1.2057720  1.2376119  1.5115579  2.4882710  6.3911619 28.0923707
## [13] 39.9582946
```

```
cvRRglm.in$cvm[order(cvRRglm.in$lambda)]
```

```
##  [1] 1.006964 1.005876 1.003517 1.001638 1.002237 1.015735 1.044298 1.098988
##  [9] 1.287589 1.710899 2.741132 5.896746
```

```
cvENETglm.in$cvm[order(cvENETglm.in$lambda)]
```

```
##  [1]  1.014900  1.012456  1.008593  1.003765  1.009888  1.074494  1.209774
##  [8]  1.284239  1.441655  2.001501  3.986701 14.141569
```

```
(1:nmodels)[order(allmodelCV.in)[1]]
```

```
## [1] 3
```

```
min(allmodelCV.in)
```

```
## [1] 0.9290342
```

```
allmodelCV.in1
```

```
##  [1] 37.6308977 34.7988666  3.9917335  3.7465479  3.7830499  3.7649809
##  [7]  3.7809433  3.7895150  2.7973418  1.3534770  1.0069638  1.0058760
## [13]  1.0035174  1.0016381  1.0022373  1.0157350  1.0442976  1.0989885
## [19]  1.2875895  1.7108988  2.7411316  5.8967456  1.0166300  1.0121639
## [25]  1.0057860  0.9993979  1.0229991  1.2539408  1.2627617  1.2961130
## [31]  1.5635657  2.5399210  6.4817865 28.5190398  1.0148999  1.0124555
## [37]  1.0085928  1.0037655  1.0098876  1.0744941  1.2097737  1.2842389
## [43]  1.4416550  2.0015008  3.9867008 14.1415690
```

```
rfcv1
```

```
## [1] 39.95829
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.