Practical Machine Learning Homework #2

Load required libraries

```
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
library(AppliedPredictiveModeling)
library(Hmisc)
```

```
## Loading required package: grid
## Loading required package: survival
## Loading required package: splines
##
## Attaching package: 'survival'
##
## The following object is masked from 'package:caret':
##
##     cluster
##
## Loading required package: Formula
##
## Attaching package: 'Hmisc'
##
## The following objects are masked from 'package:base':
##
##     format.pval, round.POSIXt, trunc.POSIXt, units
```

Question 1

Load the Alzheimer's disease data using the commands:

```
data(AlzheimerDisease)
```

It is clear that this option is the correct one since it loads both the diagnosis and the predictors data and has the correct command for subsetting the training and testing data sets

```
adData = data.frame(diagnosis, predictors)
trainIndex = createDataPartition(diagnosis, p = 0.5, list = FALSE)
training = adData[trainIndex, ]
testing = adData[-trainIndex, ]
```

Question 2

Load the cement data using the commands:

```
library(AppliedPredictiveModeling)
data(concrete)
library(caret)
set.seed(975)
inTrain = createDataPartition(mixtures$CompressiveStrength, p =
3/4)[[1]]
training = mixtures[inTrain, ]
testing = mixtures[-inTrain, ]
```

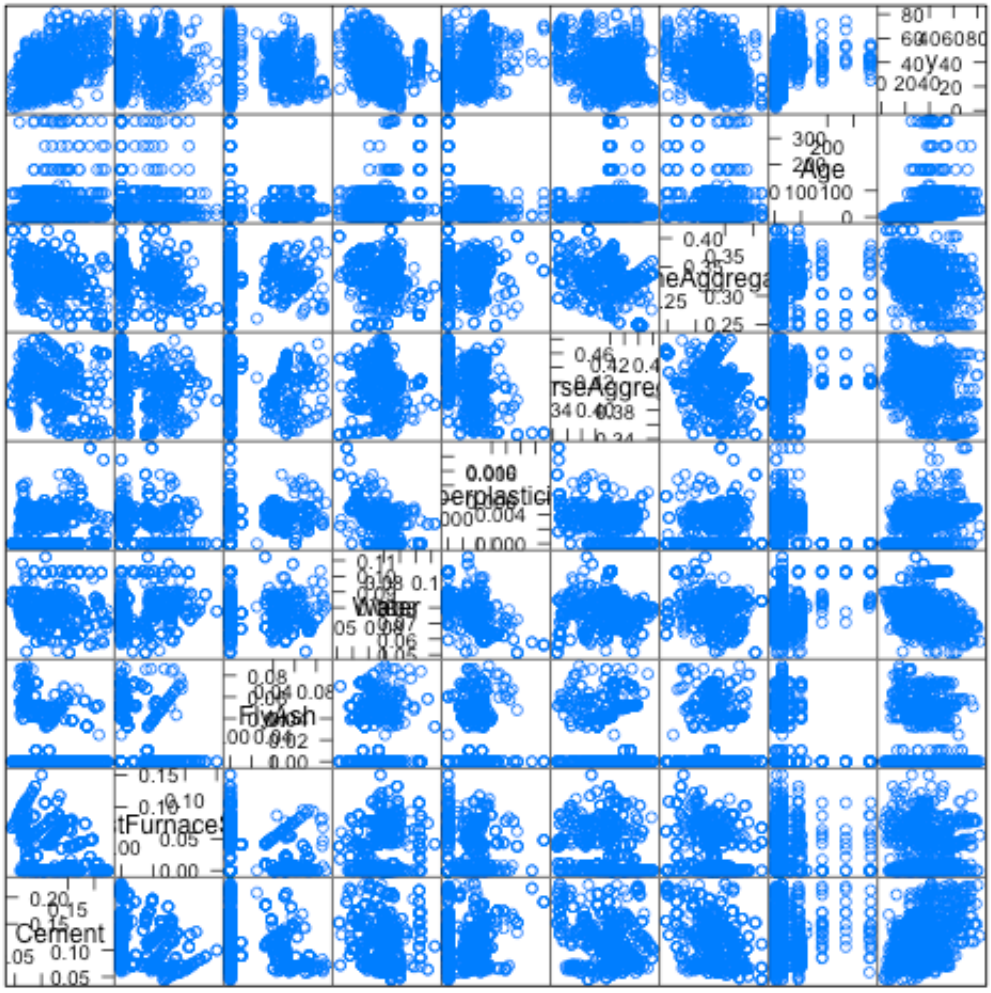
Make a plot of the outcome (CompressiveStrength) versus the index of the samples. Color by each of the variables in the data set (you may find the cut2() function in the Hmisc package useful for turning continuous covariates into factors). What do you notice in these plots?

First let's get the names of the columns to subset on them later

```
names <- colnames(concrete)
names <- names[-length(names)]
```

Now let's make a quick feature plot to see if there is any relation between the outcome CompressiveStrength and the rest of the parameters in the data:

```
featurePlot(x = training[, names], y =
training$CompressiveStrength, plot = "pairs")
```

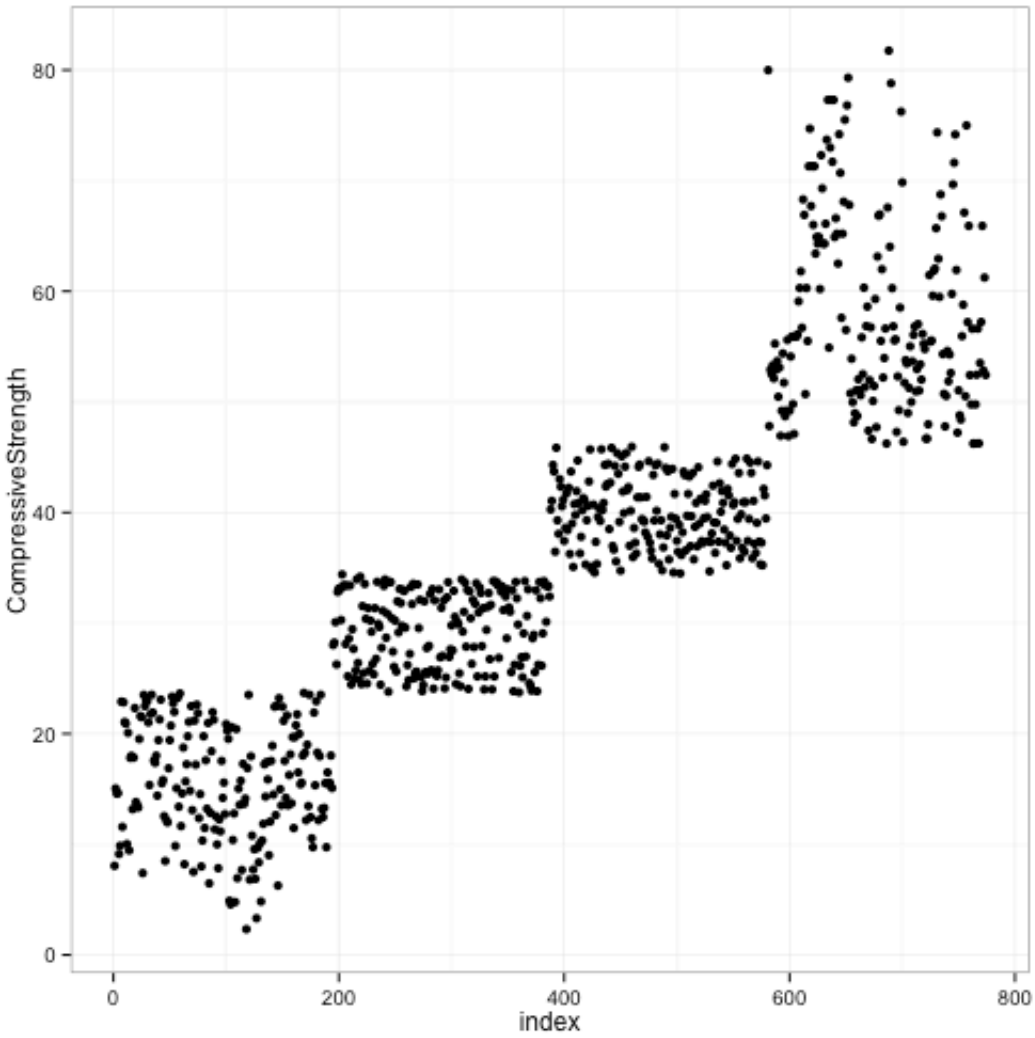


Scatter Plot Matrix

It is clear from this plot that there is no relation between the outcome and any of the other variables int he data set

Now we'll make a plot of the outcome as a function of the index

```
index <- seq_along(1:nrow(training))
ggplot(data = training, aes(x = index, y = CompressiveStrength)) +
geom_point() +
    theme_bw()
```



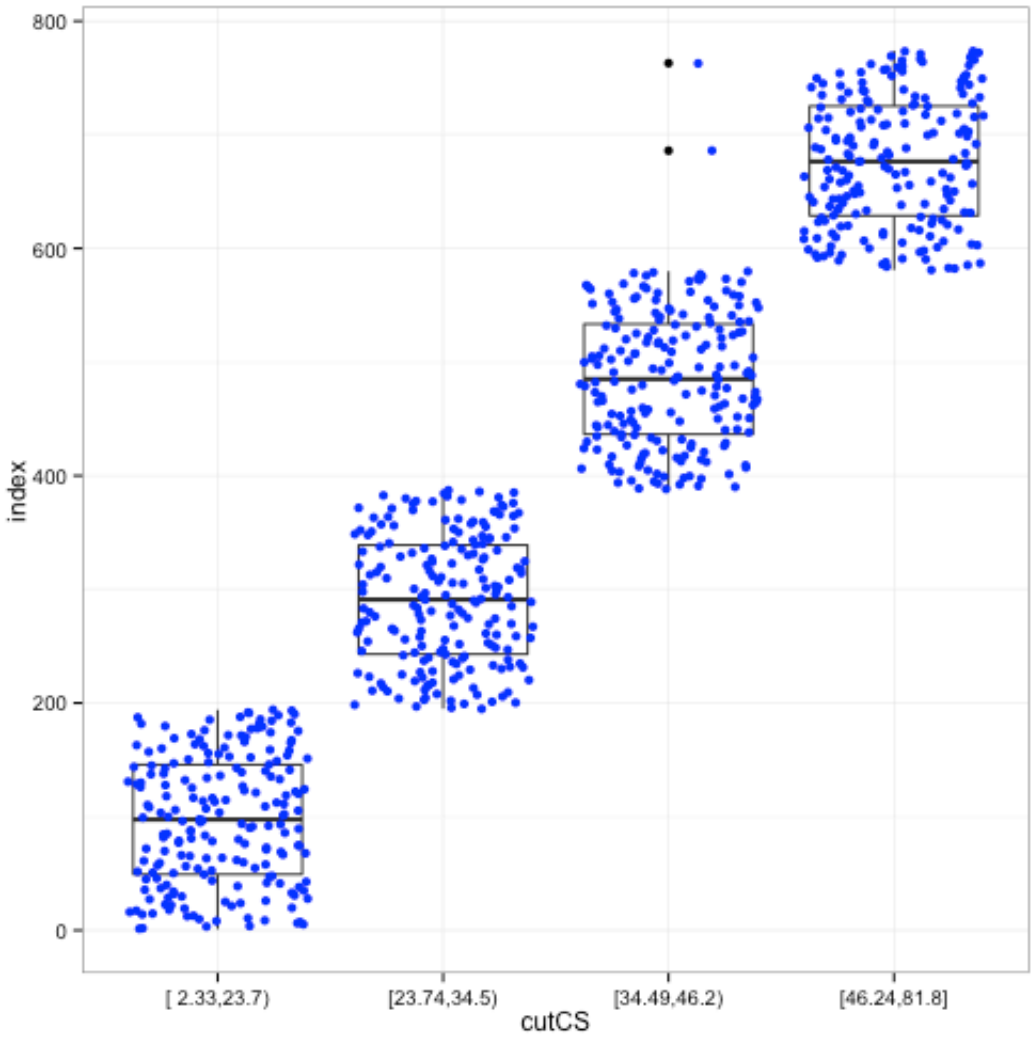
It is clear from this figure that there is a step-like pattern in the data that could be explained by one or more variable in the data.
From this plot we should probably cut the outcome in 4 categories

```
cutCS <- cut2(training$CompressiveStrength, g = 4)
summary(cutCS)
```

```
## [ 2.33,23.7) [23.74,34.5) [34.49,46.2) [46.24,81.8]
##          194          193          195          192
```

Make a plot of the categorized outcome outcome

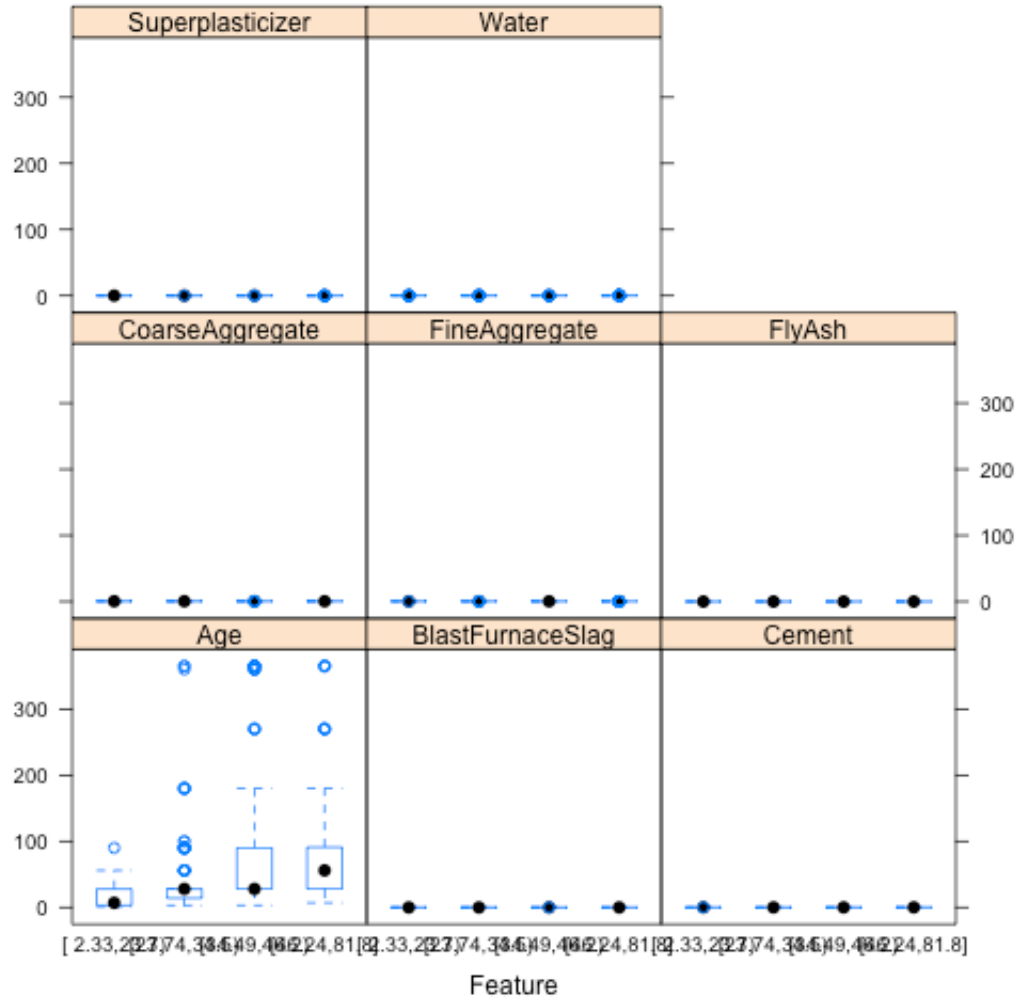
```
ggplot(data = training, aes(y = index, x = cutCS)) + geom_boxplot()
+ geom_jitter(col = "blue") +
    theme_bw()
```



Now the step is better seen in the above plot. As we can see this plot the step-like pattern is more clear now.

Now we'll make a plot of the categorized income as function of the rest of the variables

```
featurePlot(x = training[, names], y = cutCS, plot = "box")
```



Again, none of the variables in the data can explaing the step-like behaviour in the outcome.

Question 3

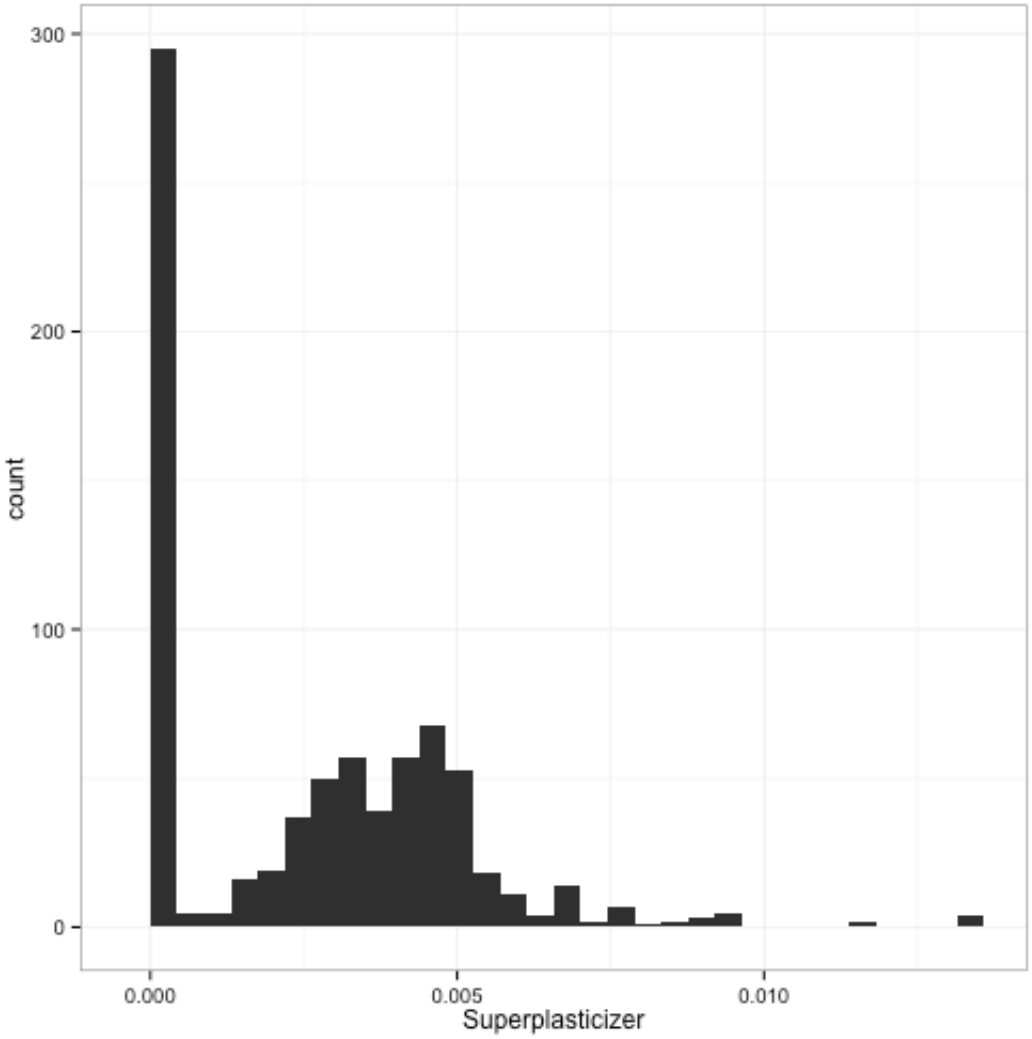
Load the cement data using the commands:

```
library(AppliedPredictiveModeling)
data(concrete)
library(caret)
set.seed(975)
inTrain = createDataPartition(mixtures$CompressiveStrength, p =
3/4)[[1]]
training = mixtures[inTrain, ]
testing = mixtures[-inTrain, ]
```

Make a histogram and confirm the Superplasticizer variable is skewed. Normally you might use the log transform to try to make the data more symmetric. Why would that be a poor choice for this variable?

```
ggplot(data = training, aes(x = Superplasticizer)) +
geom_histogram() + theme_bw()
```

```
## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to
adjust this.
```



It is clear that there are plenty of zeros in this parameter so taking the log base 10 would yield infinities.

Question 4

Load the Alzheimer's disease data using the commands:

```
set.seed(3433)
library(AppliedPredictiveModeling)
data(AlzheimerDisease)
adData = data.frame(diagnosis, predictors)
inTrain = createDataPartition(adData$diagnosis, p = 3/4)[[1]]
training = adData[inTrain, ]
testing = adData[-inTrain, ]
```

Find all the predictor variables in the training set that begin with IL. Perform principal components on these variables with the preProcess() function from the caret package. Calculate the number of principal components needed to capture 80% of the variance. How many are there?

First we'll get the predictors with grep, then we'll use the preProcess function in caret to estimate the number of

```
IL_str <- grep("^IL", colnames(training), value = TRUE)
preProc <- preProcess(training[, IL_str], method = "pca", thresh =
0.8)
preProc$rotation
```

```
##                    PC1        PC2        PC3        PC4        PC5
PC6
## IL_11          -0.06530 -0.5555957 -0.2031318  0.050390 -0.735128
0.102015
## IL_13           0.27529 -0.3559427  0.0399011 -0.265077  0.257963
0.068928
## IL_16           0.42079 -0.0007225 -0.0832211  0.082097 -0.044359
0.007095
## IL_17E         -0.01126 -0.5635958 -0.3744707 -0.302512  0.389187
-0.221149
## IL_1alpha       0.25078  0.0687043  0.3008367 -0.330946 -0.169925
-0.742391
## IL_3            0.42026  0.0703353  0.1049647  0.065353 -0.023528
0.165588
## IL_4            0.33302 -0.0688496  0.1395450 -0.165632  0.142688
0.297421
## IL_5            0.38707  0.0039620 -0.0005616  0.224449 -0.084260
-0.153836
## IL_6            0.05398  0.4248426 -0.6090822 -0.417591  0.001651
0.166090
## IL_6_Receptor   0.21219 -0.1005338 -0.2920341  0.659953  0.296540
-0.138000
## IL_7            0.32949 -0.0806070  0.1966472 -0.165545 -0.113735
0.405698
## IL_8            0.29330  0.1883040 -0.4405255 -0.002811 -0.286086
-0.184321
##                    PC7
## IL_11          -0.20984
## IL_13          -0.58943
## IL_16           0.06582
## IL_17E          0.46463
## IL_1alpha      -0.12787
## IL_3            0.09007
## IL_4           -0.19661
## IL_5            0.16426
## IL_6           -0.21895
## IL_6_Receptor -0.22658
## IL_7            0.42066
## IL_8            0.14834
```

and we can see that there are 7 components required to achieve 80% of the variance

Question 5

Load the Alzheimer's disease data using the commands:

```
set.seed(3433)
library(AppliedPredictiveModeling)
data(AlzheimerDisease)
adData = data.frame(diagnosis, predictors)
inTrain = createDataPartition(adData$diagnosis, p = 3/4)[[1]]
training = adData[inTrain, ]
testing = adData[-inTrain, ]
```

Create a training data set consisting of only the predictors with variable names beginning with IL and the diagnosis. Build two predictive models, one using the predictors as they are and one using PCA with principal components explaining 80% of the variance in the predictors. Use method="glm" in the train function. What is the accuracy of each method in the test set? Which is more accurate?

```
set.seed(3433)
## grep the predictors starting with 'IL'
IL_str <- grep("^IL", colnames(training), value = TRUE)
## make a subset of these predictors
predictors_IL <- predictors[, IL_str]
df <- data.frame(diagnosis, predictors_IL)
inTrain = createDataPartition(df$diagnosis, p = 3/4)[[1]]
training = df[inTrain, ]
testing = df[-inTrain, ]

## train the data using the first method
modelFit <- train(diagnosis ~ ., method = "glm", data = training)
```

```
##
## Attaching package: 'e1071'
##
## The following object is masked from 'package:Hmisc':
##
##     impute
```

```
predictions <- predict(modelFit, newdata = testing)
## get the confustion matrix for the first method
C1 <- confusionMatrix(predictions, testing$diagnosis)
print(C1)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Impaired Control
##   Impaired        2       9
##   Control        20      51
##
##                Accuracy : 0.646
##                  95% CI : (0.533, 0.749)
##     No Information Rate : 0.732
##     P-Value [Acc > NIR] : 0.9664
##
##                   Kappa : -0.07
##  Mcnemar's Test P-Value : 0.0633
##
##             Sensitivity : 0.0909
##             Specificity : 0.8500
##          Pos Pred Value : 0.1818
##          Neg Pred Value : 0.7183
##              Prevalence : 0.2683
##          Detection Rate : 0.0244
##    Detection Prevalence : 0.1341
##       Balanced Accuracy : 0.4705
##
##        'Positive' Class : Impaired
##
```

```
A1 <- C1$overall[1]

## do similar steps with the caret package
modelFit <- train(training$diagnosis ~ ., method = "glm",
preProcess = "pca",
    data = training, trControl = trainControl(preProcOptions =
list(thresh = 0.8)))
C2 <- confusionMatrix(testing$diagnosis, predict(modelFit,
testing))
print(C2)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Impaired Control
##   Impaired        3      19
##   Control         4      56
##
##                Accuracy : 0.72
##                  95% CI : (0.609, 0.813)
##     No Information Rate : 0.915
##     P-Value [Acc > NIR] : 1.00000
##
##                   Kappa : 0.089
##  Mcnemar's Test P-Value : 0.00351
##
##             Sensitivity : 0.4286
##             Specificity : 0.7467
##          Pos Pred Value : 0.1364
##          Neg Pred Value : 0.9333
##              Prevalence : 0.0854
##          Detection Rate : 0.0366
##    Detection Prevalence : 0.2683
##       Balanced Accuracy : 0.5876
##
##        'Positive' Class : Impaired
##
```

```
A2 <- C2$overall[1]
```

The accuracies are 0.65 and 0.72 respectively.