Coursera - Practical Machine Learning – Quiz 2

Question 1

Load the Alzheimer's disease data using the commands:

```
suppressWarnings(library(AppliedPredictiveModeling))
data(AlzheimerDisease)
suppressWarnings(library(caret))
```

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

Which of the following commands will create non-overlapping training and test sets with about 50% of the observations assigned to each?

-

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

-

```
adData = data.frame(diagnosis,predictors)
trainIndex = createDataPartition(diagnosis,p=0.50)
training = adData[trainIndex,]
testing = adData[-trainIndex,]
```

-

```
adData = data.frame(diagnosis,predictors)
trainIndex = createDataPartition(diagnosis, p = 0.50,list=FALSE)
training = adData[trainIndex,]
testing = adData[-trainIndex,]
```

-

```
adData = data.frame(diagnosis,predictors)
```

```
train = createDataPartition(diagnosis, p = 0.50,list=FALSE)
test = createDataPartition(diagnosis, p = 0.50,list=FALSE)
```

Answer : The ***The correct answer is 3.***. The source dataset needs to contain both *diagnosis* and *predictors*, and the result of CreateDataPartition cannot be a list, otherwise the use in adData produces an error.

Question 2

Load the cement data using the commands:

```
suppressWarnings(library(AppliedPredictiveModeling))
data(concrete)
suppressWarnings(library(caret))
set.seed(1000)
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?

- There is a non-random pattern in the plot of the outcome versus index that is perfectly explained by the FlyAsh variable.
- There is a non-random pattern in the plot of the outcome versus index that is perfectly explained by the Age variable.
- There is a non-random pattern in the plot of the outcome versus index.
- There is a non-random pattern in the plot of the outcome versus index that does not appear to be perfectly explained by any predictor suggesting a variable may be missing.

```
suppressWarnings(library(Hmisc))
```
```
## Loading required package: survival
##
## 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
```
```
# Define vector containing line numbers
index <- seq_along(1:nrow(training))
```

```
# Number of cut groups
nbGroups <- 10

# colour by FlyAsh
FlyAshCut <- cut2(training$FlyAsh,g=nbGroups)
ggplot(data=training,aes(x=index,y=CompressiveStrength,color=FlyAshCut))+ geom_point()
```


```
# colour by Age
AgeCut <- cut2(training$Age,g=nbGroups)
ggplot(data=training,aes(x=index,y=CompressiveStrength,color=AgeCut))+ geom_point()
```


```
# colour by Cement
CementCut <- cut2(training$Age,g=nbGroups)
ggplot(data=training,aes(x=index,y=CompressiveStrength,color=CementCut))+ geom_point()
```


```
# colour by Water
WaterCut <- cut2(training$Age,g=nbGroups)
ggplot(data=training,aes(x=index,y=CompressiveStrength,color=WaterCut))+ geom_point()
```


```
# colour by BlastFurnaceSlag

BlastFurnaceSlagCut <- cut2(training$Age,g=nbGroups)

ggplot(data=training,aes(x=index,y=CompressiveStrength,color=BlastFurnaceSlagCut))+ geom_
point()
```


```
# colour by BlastFurnaceSlag

SuperplasticizerCut <- cut2(training$Age,g=nbGroups)

ggplot(data=training,aes(x=index,y=CompressiveStrength,color=SuperplasticizerCut))+ geom_
point()
```


```
# colour by CoarseAggregate

CoarseAggregateCut <- cut2(training$Age,g=nbGroups)

ggplot(data=training,aes(x=index,y=CompressiveStrength,color=CoarseAggregateCut))+ geom_p
oint()
```


```
# colour by FineAggregate

FineAggregateCut <- cut2(training$Age,g=nbGroups)

ggplot(data=training,aes(x=index,y=CompressiveStrength,color=FineAggregateCut))+ geom_poi
nt()
```


```
# colour by FineAggregate

FineAggregateCut <- cut2(training$Age,g=nbGroups)

ggplot(data=training,aes(x=index,y=CompressiveStrength,color=FineAggregateCut))+ geom_poi
nt()
```

Question 3

Load the cement data using the commands:

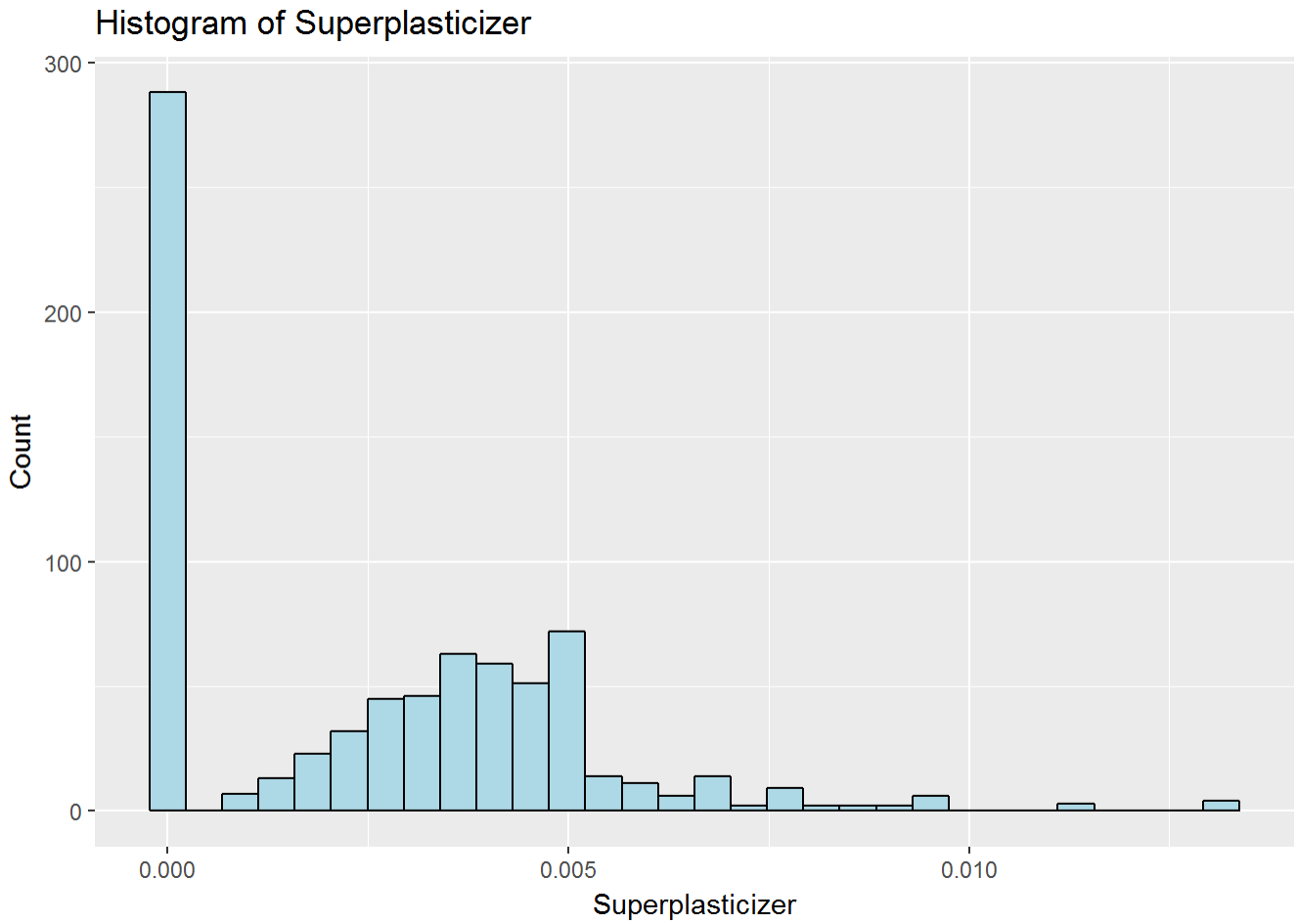
```
suppressWarnings(library(AppliedPredictiveModeling))
data(concrete)
suppressWarnings(library(caret))
set.seed(1000)
inTrain = createDataPartition(mixtures$CompressiveStrength, p = 3/4)[[1]]
training = mixtures[ inTrain,]
testing = mixtures[-inTrain,]
```

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?

- There are values of zero so when you take the log() transform those values will be -Inf.
- The SuperPlasticizer data include negative values so the log transform can not be performed.
- The log transform does not reduce the skewness of the non-zero values of SuperPlasticizer
- The log transform produces negative values which can not be used by some classifiers.

```
ggplot(training, aes(x=training$Superplasticizer))+ geom_histogram(color="black",fill="li
ghtblue") + labs(title="Histogram of Superplasticizer",
```

```
        x ="Superplasticizer", y = "Count")
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We can notice that some values of Superplasticizer are 0, which is confirmed by calling the summary on *training* dataset. So ***There are values of zero so when you take the log() transform those values will be -Inf***.

Question 4

Load the Alzheimer's disease data using the commands:

```
suppressWarnings(library(caret))
suppressWarnings(library(AppliedPredictiveModeling))
set.seed(3433)
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 90% of the variance. How many are there?

- 10
- 8
- 7
- 9

```
IdxCol_IL <- grep("^IL", names(training))
train_IL <- training[,IdxCol_IL]
test_IL <- testing[,IdxCol_IL]
preproc <- preProcess(train_IL, method="pca", thresh=0.9)
preproc$numComp
```

```
## [1] 9
```

***Anwser :** **The number of principal components necesssary to explain 90% of the variance is 9**.*

Question 5

Load the Alzheimer's disease data using the commands:

```
library(caret)
library(AppliedPredictiveModeling)
set.seed(3433)
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?

- Non-PCA Accuracy: 0.91
  PCA Accuracy: 0.93
- Non-PCA Accuracy: 0.72
  PCA Accuracy: 0.65
- Non-PCA Accuracy: 0.72
  PCA Accuracy: 0.71
- Non-PCA Accuracy: 0.65
  PCA Accuracy: 0.72

```
set.seed(3430)
suppressMessages(library(dplyr))
IdxCol_IL <- grep("^IL", names(testing))
names_IL <- names(testing[,IdxCol_IL])
newcols <- c(names_IL,"diagnosis")
```

```
new_testing <- testing [,newcols]

new_training <- training[,newcols]


# Model 1 : predictors as they are, without PCA

model_without_PCA <- train(diagnosis~., data=new_training,   preProcess=c("center","scale
"),method="glm")

model_result_without_PCA <- confusionMatrix(new_testing$diagnosis,predict(model_without_P
CA,subset(new_testing, select = -c(diagnosis))))

model_result_without_PCA
```

```
## Confusion Matrix and Statistics

##

##           Reference

## Prediction Impaired Control

##   Impaired        2      20

##   Control         9      51

##

##                Accuracy : 0.6463

##                  95% CI : (0.533, 0.7488)

##     No Information Rate : 0.8659

##     P-Value [Acc > NIR] : 1.00000

##

##                   Kappa : -0.0702

##  Mcnemar's Test P-Value : 0.06332

##

##             Sensitivity : 0.18182

##             Specificity : 0.71831

##          Pos Pred Value : 0.09091

##          Neg Pred Value : 0.85000

##              Prevalence : 0.13415

##          Detection Rate : 0.02439

##    Detection Prevalence : 0.26829

##       Balanced Accuracy : 0.45006

##

##        'Positive' Class : Impaired

##
```

```
# Model 2 : predictors using PCA, with explanation of 80% of variance

preProc_pca <-  preProcess(subset(new_training, select = -c(diagnosis)), method="pca", th
resh=0.8)
```

```r
trainPC <- predict(preProc_pca,subset(new_training, select = -c(diagnosis)))

testPC <- predict(preProc_pca,subset(new_testing, select = -c(diagnosis)))

# Syntax to use to avoid "undefined columns selected" error message (by following the for
mula  defined in the slides.)

model_with_PCA<- train(x = trainPC, y = new_training$diagnosis,method="glm")

model_result_with_PCA <- confusionMatrix(new_testing$diagnosis,predict(model_with_PCA, ne
wdata=testPC))

model_result_with_PCA
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Impaired Control
##   Impaired        3      19
##   Control         4      56
##
##                Accuracy : 0.7195
##                  95% CI : (0.6094, 0.8132)
##     No Information Rate : 0.9146
##     P-Value [Acc > NIR] : 1.000000
##
##                   Kappa : 0.0889
##  Mcnemar's Test P-Value : 0.003509
##
##             Sensitivity : 0.42857
##             Specificity : 0.74667
##          Pos Pred Value : 0.13636
##          Neg Pred Value : 0.93333
##              Prevalence : 0.08537
##          Detection Rate : 0.03659
##    Detection Prevalence : 0.26829
##       Balanced Accuracy : 0.58762
##
##        'Positive' Class : Impaired
##
```

Answer : We can notice that the model using PCA is more accurate (71.95%) than the one using the original predictors (64.63%).

Non-PCA Accuracy: 0.65; PCA Accuracy: 0.72.