Week2_Quiz

Anyi Guo

25/12/2018

Week 2 Quiz

Q1: Load the Alzheimer's disease data using the commands:

```r
library(AppliedPredictiveModeling)
data(AlzheimerDisease)
```

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

Answer:

```r
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

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

Q2: Load the cement data using the commands:

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

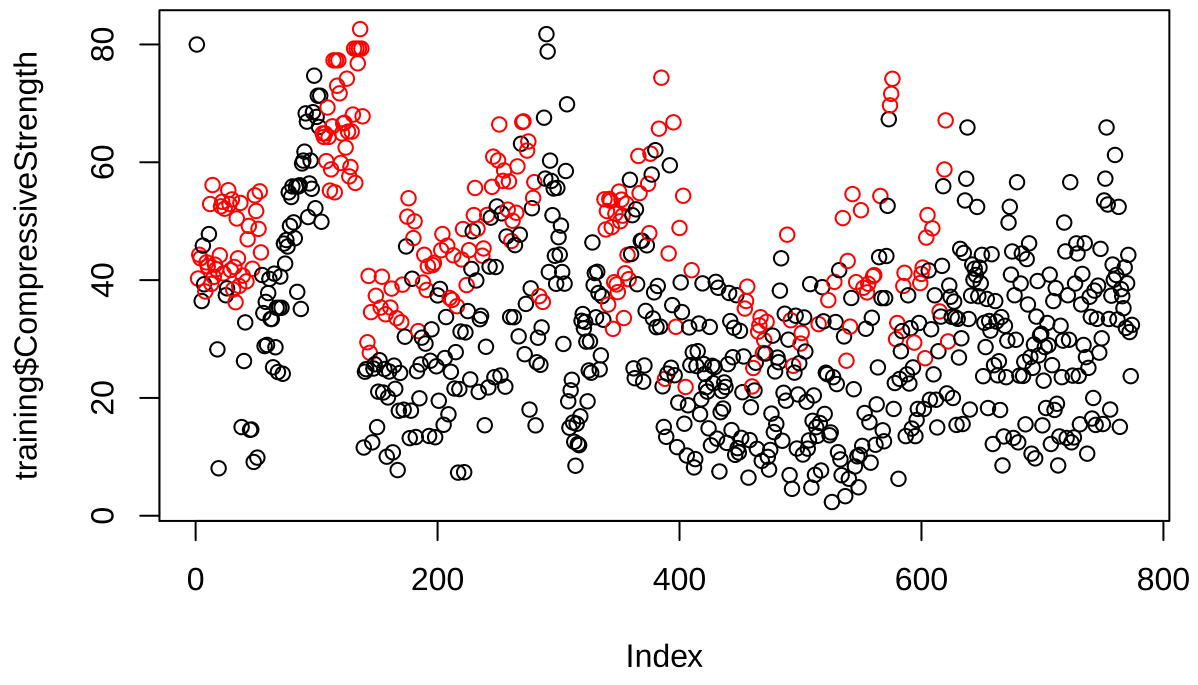
Make a plot of the outcome (CompressiveStrength) versus the index of the samples. Color by each of the variables in the data set (you may find the cut2() function in the Hmisc package useful for turning continuous covariates into factors). What do you notice in these plots?

```r
library(Hmisc)
```

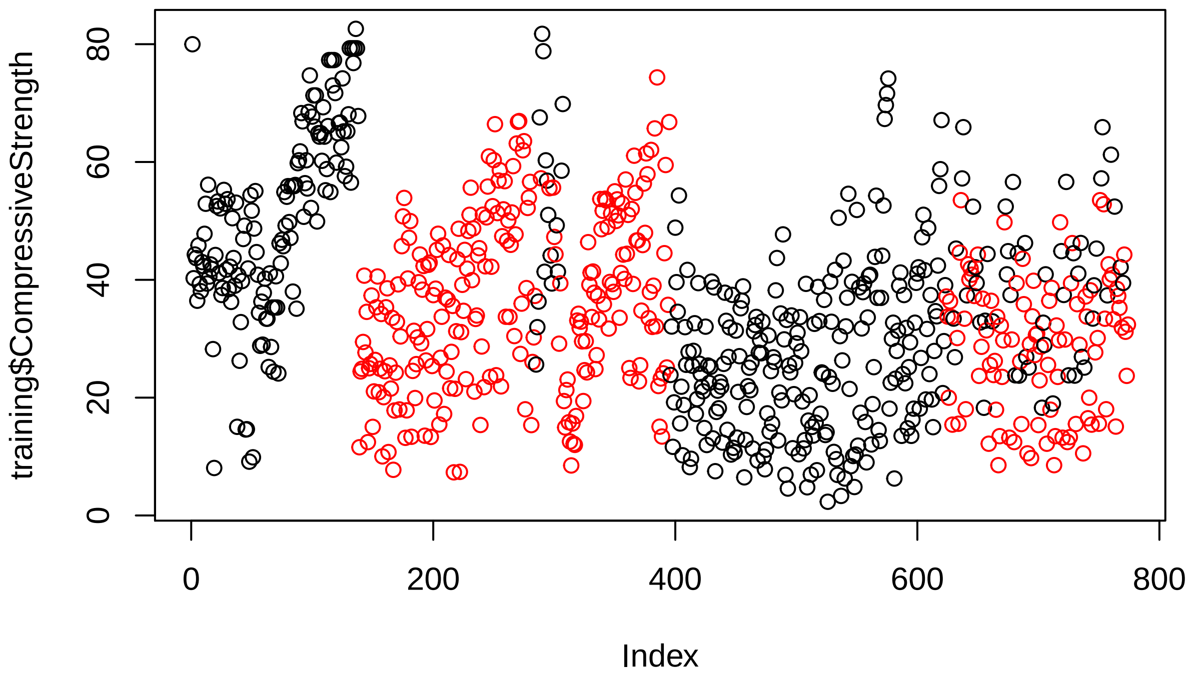
```
## Loading required package: survival
```

```
##
## 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, units
```

```
# Age
cutAge<-cut2(training$Age,g=2)
plot(training$CompressiveStrength,col=cutAge)
```



```
# FlyAsh
cutFly<-cut2(training$FlyAsh,g=2)
plot(training$CompressiveStrength,col=cutFly)
```



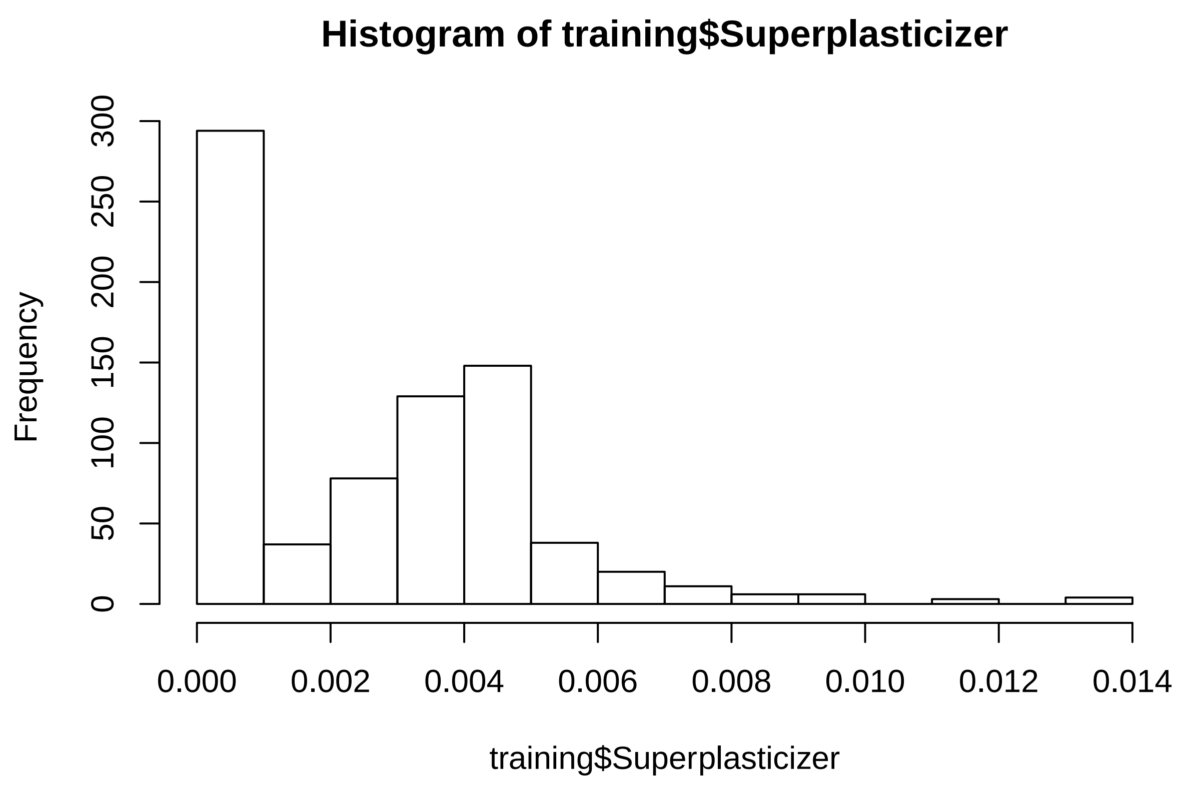
Answer: There is a non-random pattern in the plot of the outcome versus index that does not appear to be perfectly explained by any predictor suggesting a variable may be missing.

Q3: Load the cement data using the commands:

```
library(AppliedPredictiveModeling)
data(concrete)
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?

```r
hist(training$Superplasticizer)
```



```r
t<-log10(training$Superplasticizer)
t
```

```
##   [1] -2.985965      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
##   [8]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
##  [15]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
##  [22]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
##  [29]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
##  [36]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
##  [43]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
##  [50]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf -2.380605
##  [57] -2.449012 -2.118055 -2.022337 -2.433959 -1.881097 -2.169995 -2.169995
##  [64] -2.230949 -2.333005 -2.185957 -2.320898 -2.340196 -2.320898 -2.046265
##  [71] -2.320898 -2.404618 -2.380605 -2.449012 -2.118055 -2.022337 -1.881097
##  [78] -2.169995 -2.333005 -2.320898 -2.367953 -2.185957 -2.320898 -2.340196
##  [85] -2.320898 -2.335955 -2.404618 -2.449012 -2.169995 -2.118055 -2.433959
##  [92] -1.881097 -2.303446 -1.938403 -2.169995 -2.230949 -2.333005 -2.367953
##  [99] -2.185957 -2.340196 -2.320898 -2.335955 -2.320898 -2.404618 -2.449012
## [106] -2.169995 -2.118055 -2.022337 -1.881097 -2.169995 -2.303446 -1.938403
## [113] -2.230949 -2.320898 -2.185957 -2.320898 -2.340196 -2.320898 -2.335955
## [120] -2.404618 -2.380605 -2.449012 -2.118055 -2.022337 -2.433959 -2.169995
## [127] -2.303446 -1.938403 -2.169995 -2.230949 -2.320898 -2.185957 -2.320898
## [134] -2.340196 -2.320898 -2.046265 -2.320898 -2.404618 -2.717911 -2.717911
## [141] -2.717911 -2.717911 -2.717911 -2.704617 -2.704617 -2.501625 -2.501625
## [148] -2.501625 -2.501625 -2.484482 -2.484482 -2.484482 -2.484482 -2.484482
## [155] -2.616411 -2.616411 -2.616411 -2.705789 -2.705789 -2.705789 -2.705789
## [162] -2.380577 -2.717689 -2.717689 -2.717689 -2.717689 -2.613378 -2.613378
## [169] -2.613378 -2.613378 -2.613378 -2.537016 -2.537016 -2.537016 -2.537016
## [176] -2.537016 -2.549826 -2.549826 -2.549826 -2.549826 -2.549826 -2.582938
## [183] -2.582938 -2.582938 -2.520914 -2.520914 -2.520914 -2.520914 -2.520914
## [190] -2.631942 -2.631942 -2.631942 -2.439296 -2.439296 -2.439296 -2.439296
## [197] -2.439296 -2.361437 -2.361437 -2.361437 -2.486031 -2.486031 -2.486031
## [204] -2.486031 -2.548490 -2.548490 -2.548490 -2.548490 -2.607658 -2.607658
## [211] -2.565017 -2.565017 -2.565017 -2.491914 -2.491914 -2.491914 -2.460513
## [218] -2.460513 -2.460513 -2.460513 -2.460513 -2.343427 -2.343427 -2.343427
## [225] -2.405325 -2.405325 -2.307677 -2.307677 -2.307677 -2.307677 -2.307677
## [232] -2.428280 -2.428280 -2.289675 -2.289675 -2.289675 -2.289675 -2.289675
## [239] -2.272204 -2.272204 -2.272204 -2.225819 -2.225819 -2.225819 -2.225819
## [246] -2.225819 -2.303718 -2.303718 -2.303718 -2.303718 -2.303718 -2.384033
## [253] -2.384033 -2.384033 -2.384033 -2.397995 -2.397995 -2.397995 -2.397995
## [260] -2.368579 -2.368579 -2.368579 -2.308719 -2.308719 -2.308719 -2.308719
## [267] -2.333807 -2.333807 -2.333807 -2.333807 -2.333807 -2.307855 -2.307855
## [274] -2.307855 -2.307855 -2.388926 -2.388926 -2.388926 -2.388926 -2.388926
## [281] -2.324665 -2.324665 -2.324665 -2.324665      -Inf      -Inf      -Inf
## [288] -2.789404 -3.104505 -2.623232 -2.327549 -2.445870 -2.665148 -2.534045
## [295] -2.659697 -2.073718 -2.910393 -2.910393 -2.054849 -2.117549 -2.179653
## [302]      -Inf -2.440625 -2.602574 -2.555812 -2.393806 -2.416456 -2.464648
## [309] -2.481204 -2.764158 -2.561487 -2.386674 -2.379095 -2.646589 -2.307313
## [316] -2.500182 -2.610642      -Inf -2.464648 -2.817345 -2.481204 -2.475046
## [323] -2.386674 -2.379095 -2.646589 -2.307313 -2.500182 -2.464648 -2.817345
## [330] -2.481204 -2.764158 -2.475046 -2.386674 -2.379095 -2.646589 -2.500182
## [337] -2.610642 -2.464648 -2.817345 -2.481204 -2.764158 -2.561487 -2.475046
## [344] -2.386674 -2.379095 -2.646589 -2.307313 -2.610642      -Inf -2.464648
## [351] -2.817345 -2.481204 -2.386674 -2.379095 -2.646589 -2.307313 -2.500182
## [358] -2.610642 -2.316043 -2.316043 -2.316043 -2.316043 -2.316043 -2.316043
## [365] -2.316043 -2.316043 -2.367428 -2.247019 -2.325169 -2.234769 -2.247019
## [372] -2.325169 -2.247019 -2.234769 -2.247019 -2.325169 -2.782932 -2.795417
## [379] -2.782932 -2.439147 -2.450159 -2.439147 -2.439147 -2.417388 -2.417388
## [386] -3.132165 -3.132165 -3.132165 -2.615185 -2.615185 -2.615185 -2.347625
## [393] -2.347625 -2.347625 -2.347625      -Inf      -Inf      -Inf      -Inf
## [400]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [407]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [414]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [421]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [428]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [435]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [442]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [449]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [456]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [463]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [470]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [477]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [484]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [491]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [498]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [505]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [512]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [519]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [526]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [533]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [540]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [547]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [554]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [561]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [568]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [575]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [582]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [589]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [596]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [603]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [610]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [617]      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
## [624] -2.400442 -2.078609 -2.398133 -2.105321 -2.022878 -2.103043 -2.579974
## [631] -2.580545 -2.370883 -2.356981 -2.278182 -2.318822 -2.580164 -2.667266
## [638] -2.522631 -2.455416 -2.275119 -2.322219 -2.581874 -2.575573 -2.407296
## [645] -2.316733 -2.463333 -2.318822 -2.522258 -2.287615 -2.409181 -2.577492
## [652] -2.322595 -2.564469 -2.459204 -2.404454 -2.880814 -2.307302 -2.465197
## [659] -2.451786 -2.456366 -2.507470 -2.250420 -2.351796 -2.456366 -2.155905
## [666] -2.660296 -2.300042 -2.152097 -2.313867 -2.513218 -2.451786 -2.365301
## [673] -2.368845 -2.524674 -2.889115 -2.313100 -2.285745 -2.452170 -2.461836
## [680] -2.396780 -2.404644 -2.363236 -2.479827 -2.885926 -2.513978 -2.175898
## [687] -2.515685 -2.251357 -2.585085 -3.065206 -2.444630 -2.462585 -2.360404
## [694] -2.396974 -2.582442 -2.347720 -2.451403 -2.352568 -2.507856 -2.577109
## [701] -2.569179 -2.564528 -2.399788 -2.826665 -2.867866 -2.465383 -2.457192
## [708] -2.456404 -2.483510 -2.153161 -2.322181 -2.459656 -2.299963 -2.163054
## [715] -2.329762 -2.507057 -2.430808 -2.348174 -2.834832 -2.309035 -2.289304
## [722] -2.425841 -2.456329 -2.392001 -2.399940 -2.353435 -2.346297 -2.462155
## [729] -2.479755 -2.831418 -2.176053 -2.258128 -2.465271 -2.390973 -2.622741
## [736] -3.023963 -2.455528 -2.462529 -2.400269 -2.083262 -2.342305 -2.407816
## [743] -2.020890 -2.105463 -2.565772 -2.538439 -2.183099 -2.370883 -2.365717
## [750] -2.292791 -2.580336 -2.703590 -2.548203 -2.387952 -2.439580 -2.286019
## [757] -2.334191 -2.574847 -2.597637 -2.421031 -2.316847 -2.618771 -2.437172
## [764] -2.252468 -2.360155 -2.299725 -2.578600 -2.522481 -2.306821 -2.280604
## [771] -2.414147 -2.340044 -2.570256 -2.422464
```

Answer: There are values of zero so when you take the log() transform those values will be -Inf.

Q4: Load the Alzheimer's disease data using the commands:

```r
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,]
```

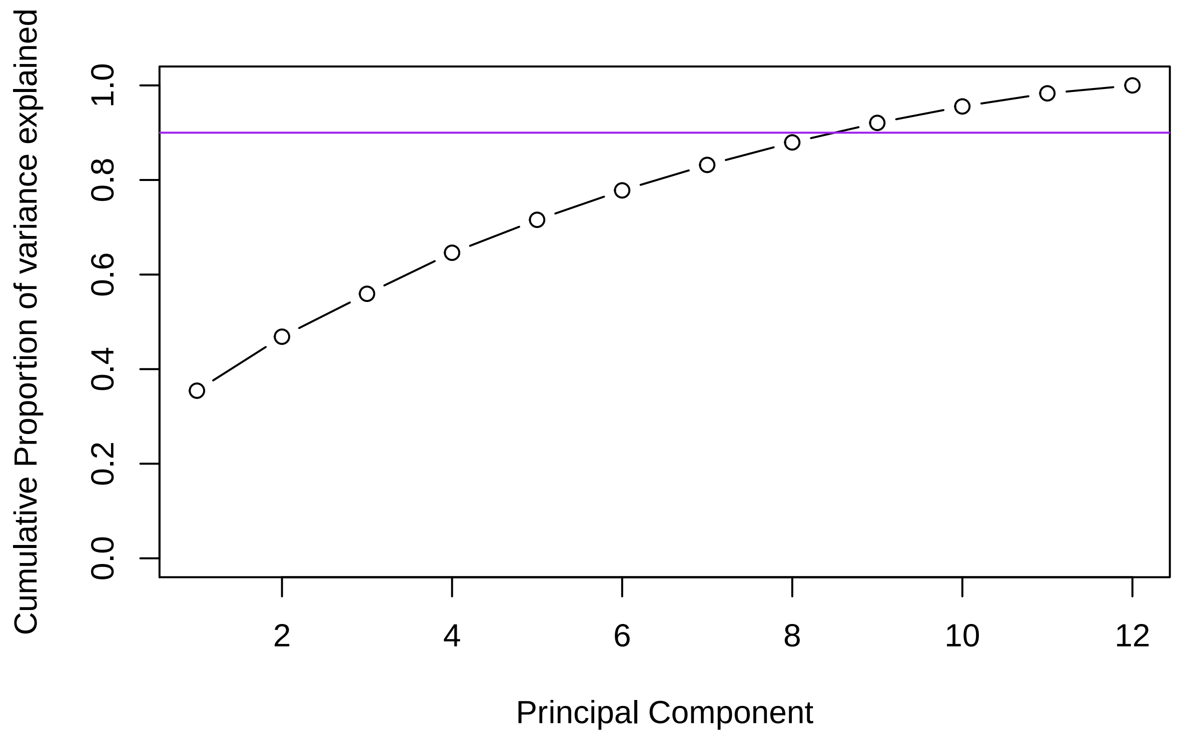
Find all the predictor variables in the training set that begin with IL.

```r
l<-as.character(colnames(training))
IL<-grep("^IL",l,value=TRUE)
IL
```

```
##  [1] "IL_11"         "IL_13"         "IL_16"         "IL_17E"
##  [5] "IL_1alpha"     "IL_3"          "IL_4"          "IL_5"
##  [9] "IL_6"          "IL_6_Receptor" "IL_7"          "IL_8"
```

Perform principal components on these variables with the preProcess() function from the caret package.

```r
library(caret)
preProc<-preProcess(training[,58:69],method="pca")
pr.alz<-prcomp(training[,58:69],scale=TRUE)
pr.alz.var<-pr.alz$sdev^2
pve<-pr.alz.var/sum(pr.alz.var)
plot(cumsum(pve),xlab="Principal Component",ylab="Cumulative Proportion of variance explained",type="b"
```



```
## integer(0)
```

Calculate the number of principal components needed to capture 90% of the variance. How many are there?

Answer: 9. This is because when PC =9, the cumulative variance explained > 0.9

Q5: 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.

```
training2<-training[,c(1,58:69)]
testing2<-testing[,c(1,58:69)]
```

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.

First model: use the predictors as they are Accuracy for the first model (Non-PCA) is `0.6463`

```
modFit<-train(diagnosis~.,method="glm",data=training2)
confusionMatrix(testing$diagnosis,predict(modFit,testing))
```

```
## 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
##
```

Second model: use PCA. > 80% of the variance is explained when PC=7. Accuracy for the second model (PCA) is `0.7073`

```
modFit2<-train(diagnosis~.,method="glm",preProcess="pca",data=training2)

confusionMatrix(testing$diagnosis,predict(modFit2,testing2))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Impaired Control
##   Impaired        3      19
##   Control         5      55
##
##                Accuracy : 0.7073
##                  95% CI : (0.5965, 0.8026)
##     No Information Rate : 0.9024
##     P-Value [Acc > NIR] : 1.000000
##
##                   Kappa : 0.0664
##  Mcnemar's Test P-Value : 0.007963
##
##             Sensitivity : 0.37500
##             Specificity : 0.74324
##          Pos Pred Value : 0.13636
##          Neg Pred Value : 0.91667
##              Prevalence : 0.09756
##          Detection Rate : 0.03659
##    Detection Prevalence : 0.26829
##       Balanced Accuracy : 0.55912
##
##        'Positive' Class : Impaired
##
```

```
preProc<-preProcess(training2,method="pca",pcaComp=7)
trainPC<-predict(preProc,training2)
modelFit<-train(x=trainPC,y=training2$diagnosis,method="glm")
testPC<-predict(preProc,testing2)
confusionMatrix(testing$diagnosis,predict(modelFit,testPC))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Impaired Control
##   Impaired       22       0
##   Control         0      60
##
##                Accuracy : 1
##                  95% CI : (0.956, 1)
##     No Information Rate : 0.7317
##     P-Value [Acc > NIR] : 7.51e-12
##
##                   Kappa : 1
## Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.2683
##          Detection Rate : 0.2683
##    Detection Prevalence : 0.2683
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : Impaired
##
```

What is the accuracy of each method in the test set? * First model: `0.6463` * Second model: `0.7073`

Which is more accurate? The model with PCA is more accurate.

Exam answer: * Non-PCA Accuracy: 0.65 * PCA Accuracy: 0.72