Test2

Mao Soldevilla

1/10/2020

Loading Libraries

```
library(caret)
```

```
FALSE Warning: package 'caret' was built under R version 4.0.2

FALSE Loading required package: lattice

FALSE Loading required package: ggplot2

FALSE Warning: package 'ggplot2' was built under R version 4.0.2
```

```
library(ggplot2)
library(AppliedPredictiveModeling)
```

```
FALSE Warning: package 'AppliedPredictiveModeling' was built under R version 4.0.2
```

1. Load the Alzheimer's disease data using the commands:

```
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?

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

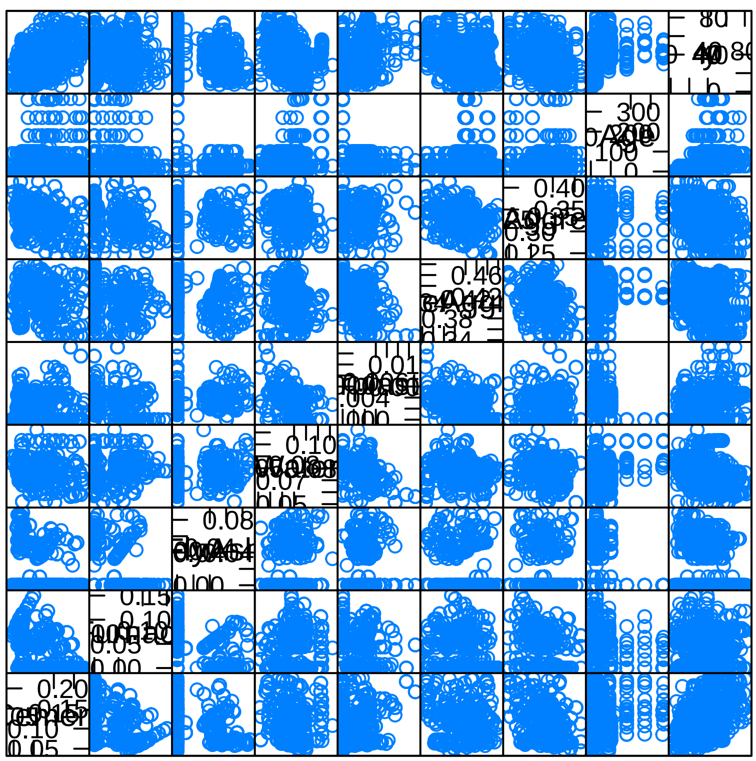
2. Load the cement data using the commands:

```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?

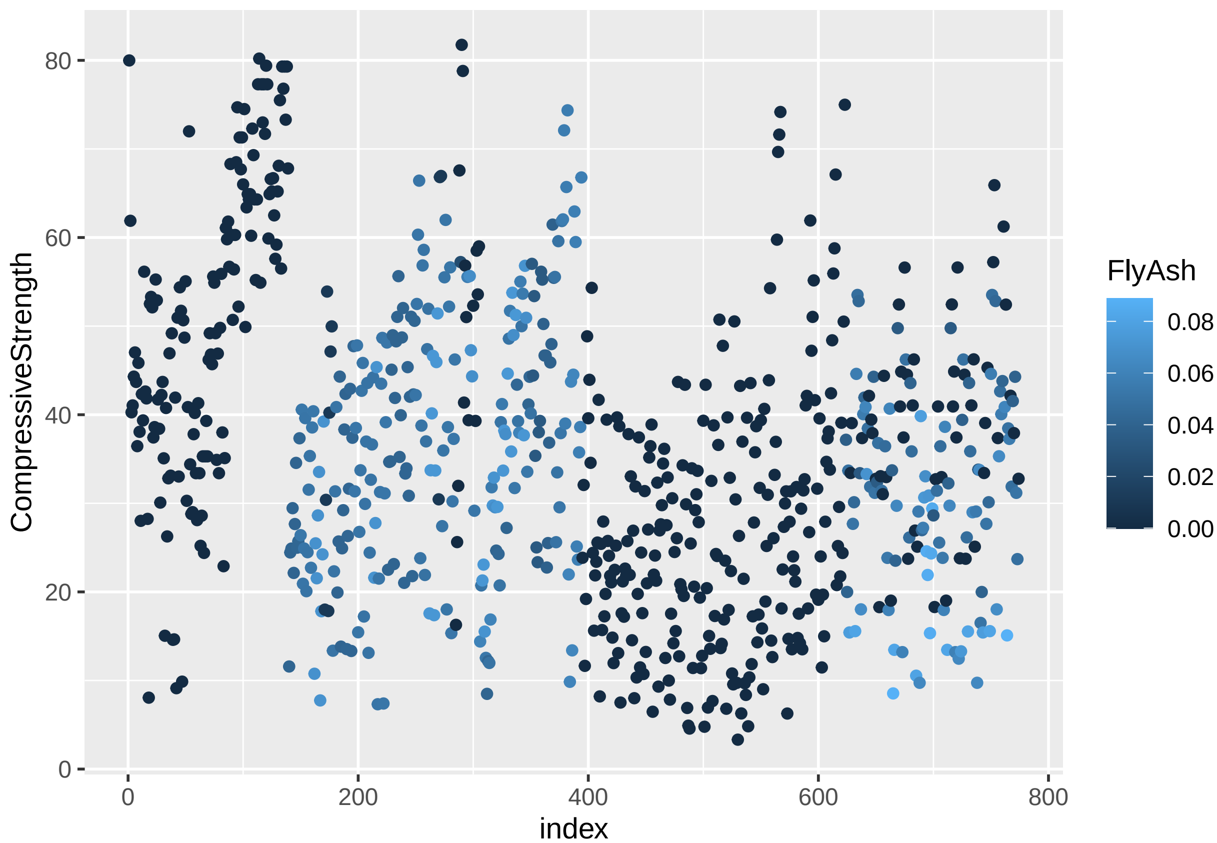
```r
# indexing
index <- seq_along(1:nrow(training))

# Checking the correlation with vars
featurePlot(x = training[, 1:8], y = training$CompressiveStrength, plot = "pairs")
```

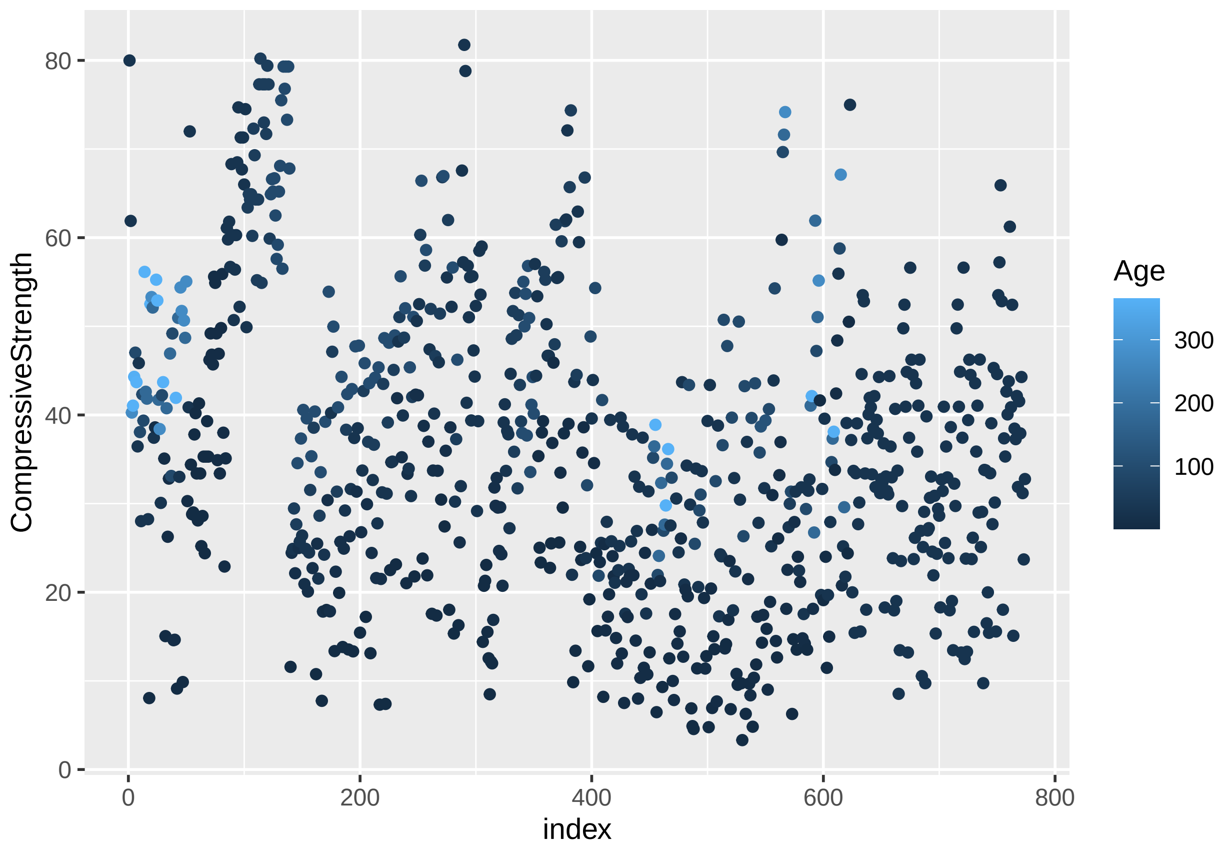


Scatter Plot Matrix

```r
# plotting by index and coloring with others vars according the question
g <- ggplot(training, aes(index, CompressiveStrength)) + geom_point(aes(color = FlyAsh))
g
```




```
g <- ggplot(training, aes(index, CompressiveStrength)) + geom_point(aes(color = Age))
g
```



There is no correlation the compressiveStrength vs FlyAsh or Age, maybe some variable is missing.

3. 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?

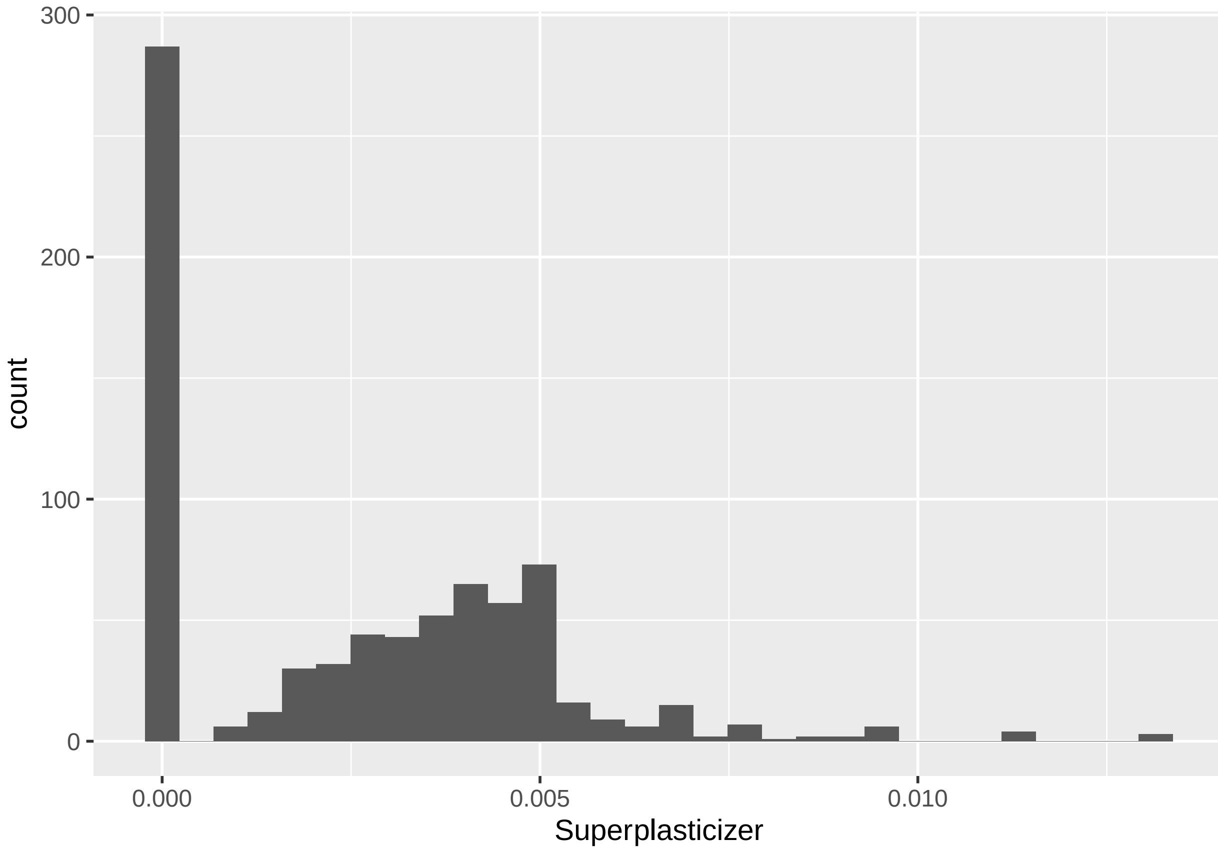
```
str(training)
```

```
## 'data.frame':    774 obs. of  9 variables:
##  $ Cement             : num  0.2231 0.2217 0.1492 0.1492 0.0853 ...
##  $ BlastFurnaceSlag   : num  0 0 0.0639 0.0639 0.0569 ...
##  $ FlyAsh             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Water              : num  0.0669 0.0665 0.1023 0.1023 0.0825 ...
##  $ Superplasticizer   : num  0.00103 0.00103 0 0 0 ...
##  $ CoarseAggregate    : num  0.43 0.433 0.418 0.418 0.42 ...
```

```
##  $ FineAggregate      : num  0.279 0.278 0.266 0.266 0.355 ...
##  $ Age                : int  28 28 270 365 360 90 365 28 28 90 ...
##  $ CompressiveStrength: num  80 61.9 40.3 41 44.3 ...
```

```
ggplot(training, aes(Superplasticizer)) + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



According this plot we can see that there are a lot of values in Zero.

Reviewing the data

```
summary(training$Superplasticizer)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.000000 0.000000 0.002723 0.002602 0.004396 0.013149
```

```
summary(concrete$Superplasticizer)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   6.400   6.205  10.200  32.200
```

```
log10(0)
```

```
## [1] -Inf
```

```
table(training[training$Superplasticizer == 0, 5])
```

```
##
##   0
## 287
```

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

4. Load the Alzheimer's disease data using the commands:

```
library(caret)
library(AppliedPredictiveModeling)
set.seed(3433)
data(AlzheimerDisease)
adData = data.frame(diagnosis,predictors)
# Adding this line in order to subset the variables
adData = adData[, c("diagnosis", grep(pattern = "^IL", names(adData), value = TRUE))]
# adData = adData[, grep(pattern = "^IL", names(adData), value = TRUE)]
# Continue
inTrain = createDataPartition(adData$diagnosis, p = 3/4)[[1]]
training = adData[ inTrain, ]
testing = adData[-inTrain, ]
str(training[, -1])
```

```
## 'data.frame':    251 obs. of  12 variables:
##  $ IL_11         : num  5.12 4.67 6.22 7.07 6.1 ...
##  $ IL_13         : num  1.28 1.27 1.31 1.31 1.28 ...
##  $ IL_16         : num  4.19 2.62 2.44 4.74 2.67 ...
##  $ IL_17E        : num  5.73 4.15 4.7 4.2 3.64 ...
##  $ IL_1alpha     : num  -6.57 -8.18 -7.6 -6.94 -8.18 ...
##  $ IL_3          : num  -3.24 -4.65 -4.27 -3 -3.86 ...
##  $ IL_4          : num  2.48 1.82 1.48 2.71 1.21 ...
##  $ IL_5          : num  1.099 -0.248 0.788 1.163 -0.4 ...
##  $ IL_6          : num  0.269 0.186 -0.371 -0.072 0.186 ...
##  $ IL_6_Receptor: num  0.6428 0.0967 0.5752 0.0967 -0.5173 ...
##  $ IL_7          : num  4.81 1.01 2.34 4.29 2.78 ...
##  $ IL_8          : num  1.71 1.69 1.72 1.76 1.71 ...
```

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?

Analyzing the data with Preprocess

```r
preProc <- preProcess(training[, -1], method = c("pca"), thresh = 0.9)
print(preProc)
```

```
## Created from 251 samples and 12 variables
##
## Pre-processing:
##   - centered (12)
##   - ignored (0)
##   - principal component signal extraction (12)
##   - scaled (12)
##
## PCA needed 9 components to capture 90 percent of the variance
```

```r
transformed <- predict(preProc, training)
summary(transformed)
```

```
##     diagnosis        PC1                PC2                 PC3
##  Impaired: 69   Min.   :-4.7101   Min.   :-3.63772   Min.   :-2.76407
##  Control :182   1st Qu.:-1.4912   1st Qu.:-0.78638   1st Qu.:-0.68983
##                 Median :-0.3192   Median : 0.00162   Median : 0.01856
##                 Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000
##                 3rd Qu.: 1.3631   3rd Qu.: 0.82856   3rd Qu.: 0.70217
##                 Max.   : 6.5025   Max.   : 3.02382   Max.   : 3.12305
##       PC4                PC5                PC6                PC7
##  Min.   :-2.63885   Min.   :-2.34826   Min.   :-2.59490   Min.   :-3.08962
##  1st Qu.:-0.62330   1st Qu.:-0.68812   1st Qu.:-0.54293   1st Qu.:-0.52825
##  Median :-0.05975   Median : 0.02678   Median :-0.01206   Median : 0.01575
##  Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.00000
##  3rd Qu.: 0.68440   3rd Qu.: 0.70531   3rd Qu.: 0.62528   3rd Qu.: 0.52299
##  Max.   : 2.44130   Max.   : 2.68526   Max.   : 3.61301   Max.   : 2.43413
##       PC8                PC9
##  Min.   :-1.96006   Min.   :-2.2663682
##  1st Qu.:-0.43349   1st Qu.:-0.4941838
##  Median :-0.04194   Median : 0.0004087
##  Mean   : 0.00000   Mean   : 0.0000000
##  3rd Qu.: 0.49517   3rd Qu.: 0.4660437
##  Max.   : 2.91236   Max.   : 1.7181586
```

```r
#preProc$rotation
```

according the results, there are 10 Principal components

5. Load the Alzheimer's disease data using the commands:

```r
library(caret)
library(AppliedPredictiveModeling)
set.seed(3433)
data(AlzheimerDisease)
adData = data.frame(diagnosis,predictors)
# Adding this line in order to subset the variables
```

```r
adData = adData[, c("diagnosis", grep(pattern = "^IL", names(adData), value = TRUE))]
# Continue
inTrain = createDataPartition(adData$diagnosis, p = 3/4)[[1]]
training = adData[ inTrain,]
testing = adData[-inTrain,]
str(training)
```

```
## 'data.frame':    251 obs. of  13 variables:
##  $ diagnosis    : Factor w/ 2 levels "Impaired","Control": 2 2 2 2 1 2 2 2 1 1 ...
##  $ IL_11        : num  5.12 4.67 6.22 7.07 6.1 ...
##  $ IL_13        : num  1.28 1.27 1.31 1.31 1.28 ...
##  $ IL_16        : num  4.19 2.62 2.44 4.74 2.67 ...
##  $ IL_17E       : num  5.73 4.15 4.7 4.2 3.64 ...
##  $ IL_1alpha    : num  -6.57 -8.18 -7.6 -6.94 -8.18 ...
##  $ IL_3         : num  -3.24 -4.65 -4.27 -3 -3.86 ...
##  $ IL_4         : num  2.48 1.82 1.48 2.71 1.21 ...
##  $ IL_5         : num  1.099 -0.248 0.788 1.163 -0.4 ...
##  $ IL_6         : num  0.269 0.186 -0.371 -0.072 0.186 ...
##  $ IL_6_Receptor: num  0.6428 0.0967 0.5752 0.0967 -0.5173 ...
##  $ IL_7         : num  4.81 1.01 2.34 4.29 2.78 ...
##  $ IL_8         : num  1.71 1.69 1.72 1.76 1.71 ...
```

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?

Train the data with the first model

```r
modelFit1 <- train(diagnosis ~ ., method = "glm", data = training)
predictions <- predict(modelFit1, newdata = testing)
Acc1 <- confusionMatrix(predictions, testing$diagnosis)
Acc1
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Impaired Control
##   Impaired        4       2
##   Control        18      58
##
##                Accuracy : 0.7561
##                  95% CI : (0.6488, 0.8442)
##     No Information Rate : 0.7317
##     P-Value [Acc > NIR] : 0.3606488
##
##                   Kappa : 0.1929
##
##  Mcnemar's Test P-Value : 0.0007962
##
##             Sensitivity : 0.18182
```

```
##          Specificity : 0.96667
##       Pos Pred Value : 0.66667
##       Neg Pred Value : 0.76316
##           Prevalence : 0.26829
##       Detection Rate : 0.04878
## Detection Prevalence : 0.07317
##    Balanced Accuracy : 0.57424
##
##     'Positive' Class : Impaired
##
```

```
modelFit2 <- train(diagnosis ~ .,
                  method = "glm",
                  preProcess = "pca",
                  data = training,
                  trControl = trainControl(preProcOptions = list(thresh = 0.8)))
Acc2 <- confusionMatrix(testing$diagnosis, predict(modelFit2, testing))
Acc2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Impaired Control
##   Impaired        1      21
##   Control         2      58
##
##                Accuracy : 0.7195
##                  95% CI : (0.6094, 0.8132)
##     No Information Rate : 0.9634
##     P-Value [Acc > NIR] : 1.0000000
##
##                   Kappa : 0.0167
##
##  Mcnemar's Test P-Value : 0.0001746
##
##             Sensitivity : 0.33333
##             Specificity : 0.73418
##          Pos Pred Value : 0.04545
##          Neg Pred Value : 0.96667
##              Prevalence : 0.03659
##          Detection Rate : 0.01220
##    Detection Prevalence : 0.26829
##       Balanced Accuracy : 0.53376
##
##        'Positive' Class : Impaired
##
```

```
names(Acc2)
```

```
## [1] "positive" "table"    "overall"  "byClass"  "mode"     "dots"
```

The accuracy are 0.7560976 and 0.7195122 respectively.