Code_SPAM Example_Practical Machine Learning

Yato

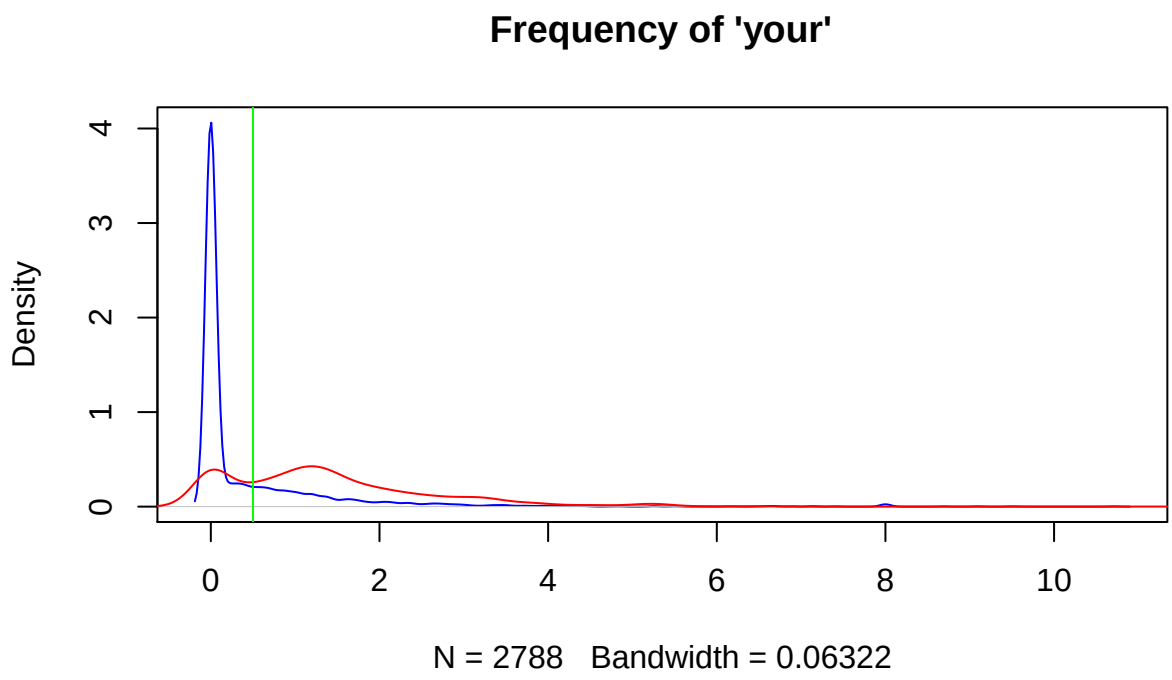
March 24, 2017

Part 1: Check the SPAM data

```
library(kernlab); data(spam); dim(spam)
```

```
## [1] 4601   58
```

```
plot(density(spam$your[spam$type=="nonspam"]), col="blue", main = "Frequency of 'your'")
lines(density(spam$your[spam$type=="spam"]), col="red")
abline(v = 0.5, col="green")
```




```
prediction <- ifelse(spam$your > 0.5, "spam", "nonspm")
table(prediction, spam$type)/length(spam$type)
```

```
##
## prediction   nonspam      spam
##     nonspm 0.4590306 0.1017170
##     spam   0.1469246 0.2923278
```

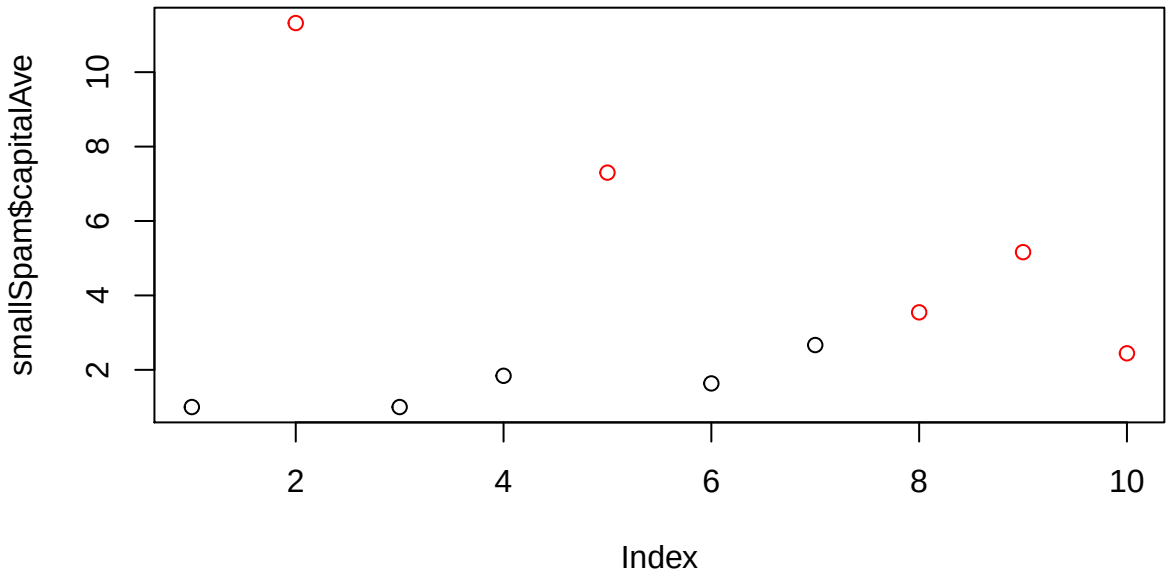
```
#or
mytable <- table(prediction, spam$type)
prop.table(mytable)
```

```
##
```

```
## prediction   nonspam      spam
##     nonspm 0.4590306 0.1017170
##     spam   0.1469246 0.2923278
```

Part 2: In sample and Out of sample

```
# subset a small data
set.seed(333)
smallSpam <- spam[sample(dim(spam)[1], size = 10),]
spamLabel <- (smallSpam$type == "spam") * 1 + 1
plot(smallSpam$capitalAve, col=spamLabel)
```




```
# prediction rule 1 to samllSpam
rule1 <- function(x){
        prediction <- rep(NA,length(x))
        prediction[x > 2.7] <- "spam"
        prediction[x < 2.40] <- "nonspam"
        prediction[(x >= 2.40 & x <= 2.45)] <- "spam"
        prediction[(x > 2.45 & x <= 2.70)] <- "nonspam"
        return(prediction)
}
table(rule1(smallSpam$capitalAve),smallSpam$type)
```

```
##
##            nonspam spam
##   nonspam        5    0
##   spam           0    5
```

```r
# prediction rule 2 to samllSpam
rule2 <- function(x){
        prediction <- rep(NA,length(x))
        prediction[x > 2.8] <- "spam"
        prediction[x <= 2.8] <- "nonspam"
        return(prediction)
}

# Apply to complete spam data
table(rule1(spam$capitalAve),spam$type)
```

```
##
##           nonspam spam
##   nonspam    2141  588
##   spam        647 1225
```

```r
table(rule2(spam$capitalAve),spam$type)
```

```
##
##           nonspam spam
##   nonspam    2224  642
##   spam        564 1171
```

```r
mean(rule1(spam$capitalAve)==spam$type)
```

```
## [1] 0.7315801
```

```r
mean(rule2(spam$capitalAve)==spam$type)
```

```
## [1] 0.7378831
```

```r
table(rule2(smallSpam$capitalAve),smallSpam$type)
```

```
##
##           nonspam spam
##   nonspam       5    1
##   spam          0    4
```

```r
# Look at accuracy
sum(rule1(spam$capitalAve)==spam$type)
```

```
## [1] 3366
```

```r
sum(rule2(spam$capitalAve)==spam$type)
```

```
## [1] 3395
```

```r
# Why the simplified rule actually does better than the more complicated rule? The reason why is over f
```

Part 3: The prediction design

```r
# Step 1: Data splitting
library(caret);
library(kernlab);
data(spam)
inTrain <- createDataPartition(y=spam$type,p=0.75, list=FALSE)
# split it based on the type and split into the 75% data to train and 25% to test the model
```

```r
training <- spam[inTrain,]
testing <- spam[-inTrain,]
dim(training)
```

```
## [1] 3451   58
```

```r
# Step 2: Fit a model
set.seed(32343)
modelFit <- train(type ~.,data=training, method="glm")
modelFit
```

```
## Generalized Linear Model
##
## 3451 samples
##   57 predictor
##    2 classes: 'nonspam', 'spam'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 3451, 3451, 3451, 3451, 3451, 3451, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9160009  0.8231214
##
##
```

```r
# Step 3: Final model
modelFit <- train(type ~.,data=training, method="glm")
modelFit$finalModel
```

```
##
## Call:  NULL
##
## Coefficients:
##       (Intercept)               make            address
##         -1.438e+00         -1.839e-01         -1.489e-01
##               all              num3d                our
##          1.175e-01          2.480e+00          5.420e-01
##              over             remove           internet
##          9.038e-01          2.457e+00          4.973e-01
##             order               mail            receive
##          5.102e-01          1.053e-01         -5.919e-01
##              will             people             report
##         -1.932e-01         -2.159e-01          2.981e-01
##         addresses               free           business
##          8.834e-01          8.883e-01          9.754e-01
##             email                you             credit
##          1.735e-01          7.511e-02          9.960e-01
##              your               font             num000
##          2.553e-01          1.483e-01          1.911e+00
##             money                 hp                hpl
##          6.091e-01         -1.837e+00         -8.798e-01
##            george             num650                lab
##         -1.179e+01          4.687e-01         -2.362e+00
##              labs              telnet             num857
```

```
##        -3.192e-01         -1.544e-01          1.026e+00
##              data             num415              num85
##        -8.858e-01          5.891e-01         -2.025e+00
##        technology            num1999              parts
##         9.146e-01          3.811e-02          4.856e-01
##                pm             direct                 cs
##        -8.030e-01         -4.246e-01         -5.553e+02
##           meeting           original            project
##        -2.624e+00         -1.211e+00         -2.089e+00
##                re                edu              table
##        -7.711e-01         -1.383e+00         -2.202e+00
##        conference      charSemicolon   charRoundbracket
##        -3.981e+00         -1.174e+00         -1.180e-01
## charSquarebracket    charExclamation         charDollar
##        -4.938e-01          2.642e-01          5.037e+00
##          charHash         capitalAve        capitalLong
##         2.437e+00          3.563e-03          1.021e-02
##      capitalTotal
##         8.545e-04
##
## Degrees of Freedom: 3450 Total (i.e. Null);  3393 Residual
## Null Deviance:       4628
## Residual Deviance: 1408  AIC: 1524
```

```
# Step 4: Prediction
predictions <- predict(modelFit,newdata=testing)
head(predictions, 10)
```

```
##  [1] spam    spam    spam    spam    nonspam spam    spam    spam
##  [9] nonspam spam
## Levels: nonspam spam
```

```
# Step 5: Confusion Matrix
confusionMatrix(predictions,testing$type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction nonspam spam
##    nonspam     659   36
##    spam         38  417
##
##                Accuracy : 0.9357
##                  95% CI : (0.9199, 0.9491)
##     No Information Rate : 0.6061
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8653
##  Mcnemar's Test P-Value : 0.9075
##
##             Sensitivity : 0.9455
##             Specificity : 0.9205
##          Pos Pred Value : 0.9482
##          Neg Pred Value : 0.9165
##              Prevalence : 0.6061
##          Detection Rate : 0.5730
```

```
##    Detection Prevalence : 0.6043
##       Balanced Accuracy : 0.9330
##
##        'Positive' Class : nonspam
##
```

Part 4: Data Slicing

```
# Data splitting
library(caret); library(kernlab); data(spam)
inTrain <- createDataPartition(y=spam$type,p=0.75, list=FALSE)
training <- spam[inTrain,]
testing <- spam[-inTrain,]
dim(training)
```

```
## [1] 3451   58
```

```
# K-fold
# Return the training set by setting returnTrain=TRUE
set.seed(32323)
folds <- createFolds(y=spam$type,k=10,list=TRUE,returnTrain=TRUE)
# k=10 is the number of folds that we'd like to create.
# list=TRUE means it will return each set of imbecies corresponding to a particular fold
# as a set of as a list(list/vector/matrix).
sapply(folds,length)
```

```
## Fold01 Fold02 Fold03 Fold04 Fold05 Fold06 Fold07 Fold08 Fold09 Fold10
##   4141   4140   4141   4142   4140   4142   4141   4141   4140   4141
```

```
folds[[1]][1:10]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
# Return the test set samples by setting returnTrain=FALSE
set.seed(32323)
folds <- createFolds(y=spam$type,k=10,list=TRUE,returnTrain=FALSE)
sapply(folds,length)
```

```
## Fold01 Fold02 Fold03 Fold04 Fold05 Fold06 Fold07 Fold08 Fold09 Fold10
##    460    461    460    459    461    459    460    460    461    460
```

```
folds[[1]][1:10]
```

```
##  [1] 24 27 32 40 41 43 55 58 63 68
```

```
## Resampling
set.seed(32323)
folds <- createResample(y=spam$type,times=10,list=TRUE)
sapply(folds,length)
```

```
## Resample01 Resample02 Resample03 Resample04 Resample05 Resample06
##       4601       4601       4601       4601       4601       4601
## Resample07 Resample08 Resample09 Resample10
##       4601       4601       4601       4601
```

```
folds[[1]][1:10]
```

```
##  [1]  1  2  3  3  3  5  5  7  8 12
```

```
# Time Slices
set.seed(32323)
tme <- 1:1000
folds <- createTimeSlices(y=tme,initialWindow=20,horizon=10)
# create slices that have a window of about 20 samples in them and
# predict the next 10 samples out after i take the initial window of 20.
names(folds)
```

```
## [1] "train" "test"
```

```
folds$train[[1]]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
folds$test[[1]] # there would be 10 samples and there would be next 10 samples, 21 to 30
```

```
##  [1] 21 22 23 24 25 26 27 28 29 30
```

Part 5: Train Options

```
library(caret); library(kernlab); data(spam)
inTrain <- createDataPartition(y=spam$type,p=0.75, list=FALSE)
training <- spam[inTrain,]
testing <- spam[-inTrain,]
modelFit <- train(type ~.,data=training, method="glm")
# Train options
args(train)
```

```
## function (x, ...)
## NULL
```

```
# Train control
args(trainControl)
```

```
## function (method = "boot", number = ifelse(grepl("cv", method),
##     10, 25), repeats = ifelse(grepl("cv", method), 1, number),
##     p = 0.75, search = "grid", initialWindow = NULL, horizon = 1,
##     fixedWindow = TRUE, skip = 0, verboseIter = FALSE, returnData = TRUE,
##     returnResamp = "final", savePredictions = FALSE, classProbs = FALSE,
##     summaryFunction = defaultSummary, selectionFunction = "best",
##     preProcOptions = list(thresh = 0.95, ICAcomp = 3, k = 5,
##         freqCut = 95/5, uniqueCut = 10, cutoff = 0.9), sampling = NULL,
##     index = NULL, indexOut = NULL, indexFinal = NULL, timingSamps = 0,
##     predictionBounds = rep(FALSE, 2), seeds = NA, adaptive = list(min = 5,
##         alpha = 0.05, method = "gls", complete = TRUE), trim = FALSE,
##     allowParallel = TRUE)
## NULL
```

```
# Setting the seed
# 1.it's useful to set an overall seed; 2.you can also set a seed for each resample;
# 3. seeding each resample is useful for parallel fits
set.seed(1235)
modelFit2 <- train(type ~.,data=training, method="glm")
set.seed(1235)
modelFit3 <- train(type ~.,data=training, method="glm")
```

```
modelFit3
```

```
## Generalized Linear Model
##
## 3451 samples
##   57 predictor
##    2 classes: 'nonspam', 'spam'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 3451, 3451, 3451, 3451, 3451, 3451, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9152739  0.8213541
##
##
```

Part 6: Plotting Predictors

```
# Example:predicting wages from ISLR package from book [Introduction to statistical learning]
library(ISLR); library(ggplot2); library(caret); library(gridExtra);
data(Wage)
summary(Wage)
```

```
##       year           age               sex                    maritl
##  Min.   :2003   Min.   :18.00   1. Male  :3000   1. Never Married: 648
##  1st Qu.:2004   1st Qu.:33.75   2. Female:   0   2. Married      :2074
##  Median :2006   Median :42.00                    3. Widowed      :  19
##  Mean   :2006   Mean   :42.41                    4. Divorced     : 204
##  3rd Qu.:2008   3rd Qu.:51.00                    5. Separated    :  55
##  Max.   :2009   Max.   :80.00
##
##        race                   education                     region
##  1. White:2480   1. < HS Grad      :268   2. Middle Atlantic   :3000
##  2. Black: 293   2. HS Grad        :971   1. New England       :   0
##  3. Asian: 190   3. Some College   :650   3. East North Central:   0
##  4. Other:  37   4. College Grad   :685   4. West North Central:   0
##                  5. Advanced Degree:426   5. South Atlantic    :   0
##                                           6. East South Central:   0
##                                           (Other)              :   0
##            jobclass               health      health_ins      logwage
##  1. Industrial :1544   1. <=Good     : 858   1. Yes:2083   Min.   :3.000
##  2. Information:1456   2. >=Very Good:2142   2. No : 917   1st Qu.:4.447
##                                                            Median :4.653
##                                                            Mean   :4.654
##                                                            3rd Qu.:4.857
##                                                            Max.   :5.763
##
##       wage
##  Min.   : 20.09
##  1st Qu.: 85.38
##  Median :104.92
```

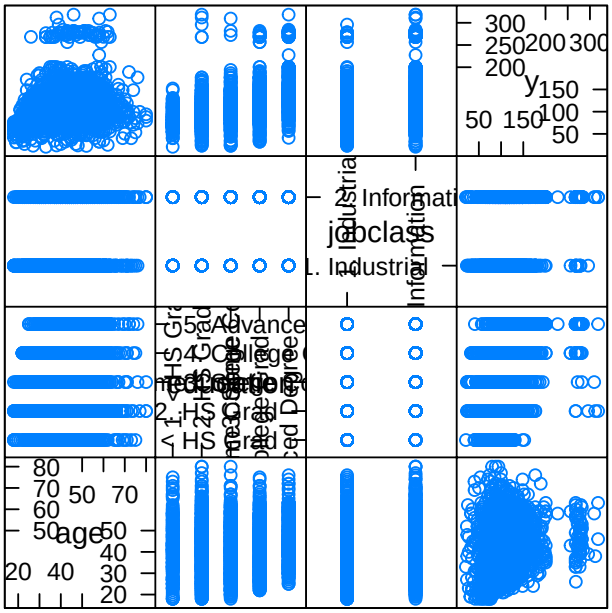
```
##  Mean   :111.70
##  3rd Qu.:128.68
##  Max.   :318.34
##
```

```
# Get training/test sets
inTrain <- createDataPartition(y=Wage$wage, p=0.7, list=FALSE)
training <- Wage[inTrain,]
testing <- Wage[-inTrain,]
dim(training); dim(testing)
```

```
## [1] 2102   12
```

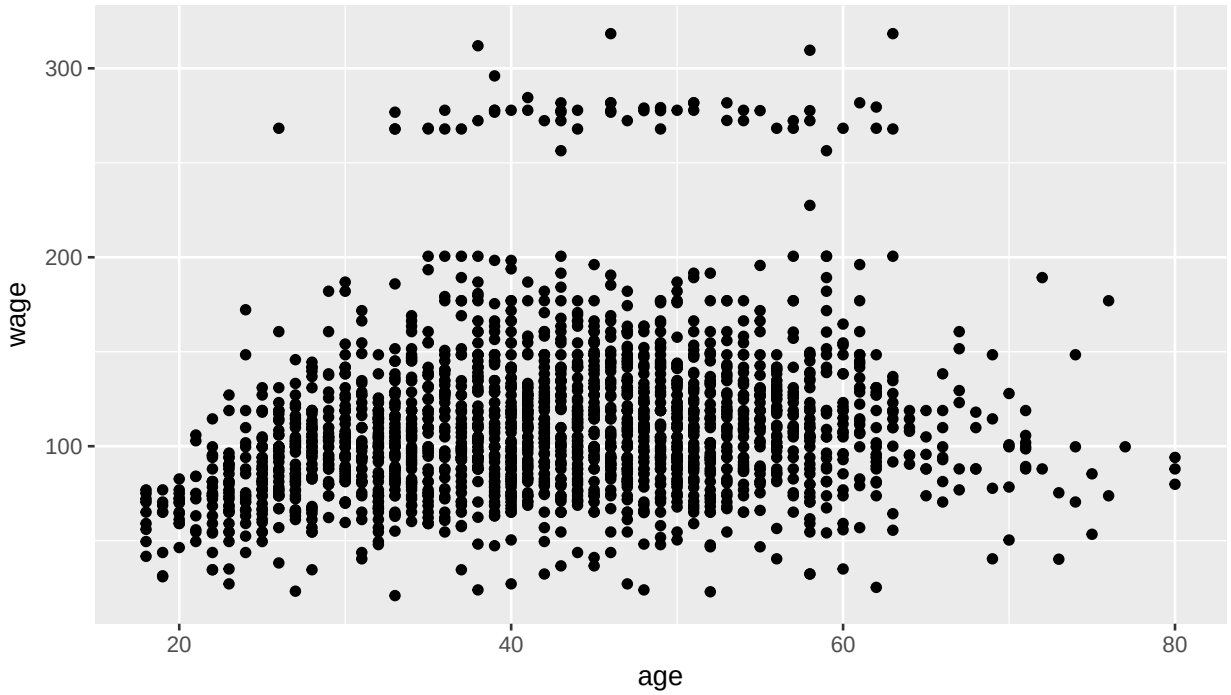
```
## [1] 898  12
```

```
# Even before we do exploration, we're going to set aside the testing set and we're not
# going to use it for anything until we actually look at the data at the end of the model
# building experience, and apply it just one time.

# Feature plot (caret package)
featurePlot(x=training[,c("age","education","jobclass")], y = training$wage, plot="pairs")
```

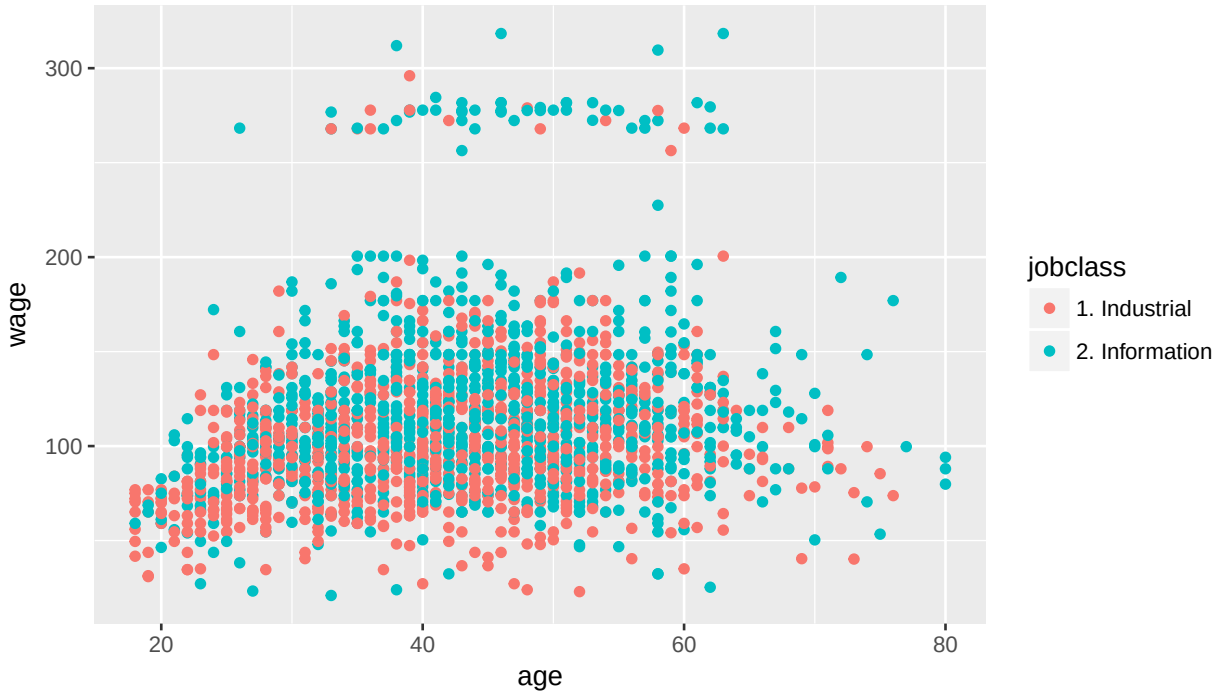


Scatter Plot Matrix

```
## y is the outcome; xs are variables.

# Qplot (ggplot2 package)
qplot(age,wage,data=training)
```




```
# Qplot with color (ggplot2 package)
qplot(age,wage,colour=jobclass,data=training)
```

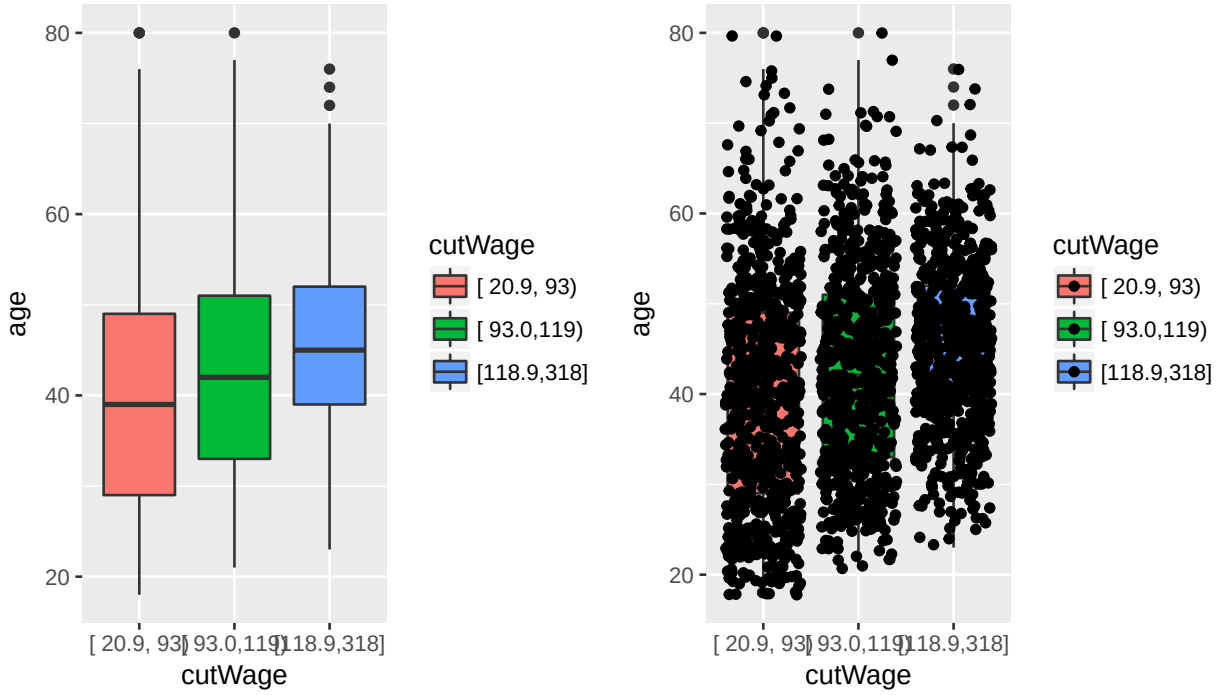



```
# Add regression smoothers (ggplot2 package)
qq <- qplot(age,wage,colour=education,data=training) + geom_smooth(method='lm',formula=y~x)

# cut2, making factors (Hmisc package)
cutWage <- cut2(training$wage,g=3) # g=3 means it will break the data set up into factors based on quan
```

```
table(cutWage)
```

```
## cutWage
## [ 20.9, 93) [ 93.0,119) [118.9,318]
##         713         715         674
```

```
# Boxplots with cut2
p1 <- qplot(cutWage,age, data=training,fill=cutWage, geom=c("boxplot"))
# Boxplots with points overlayed
p2 <- qplot(cutWage,age, data=training,fill=cutWage, geom=c("boxplot","jitter"))
# add the points on top of the box plots.This is because sometimes box plots can obscure how many point
grid.arrange(p1,p2,ncol=2)
```



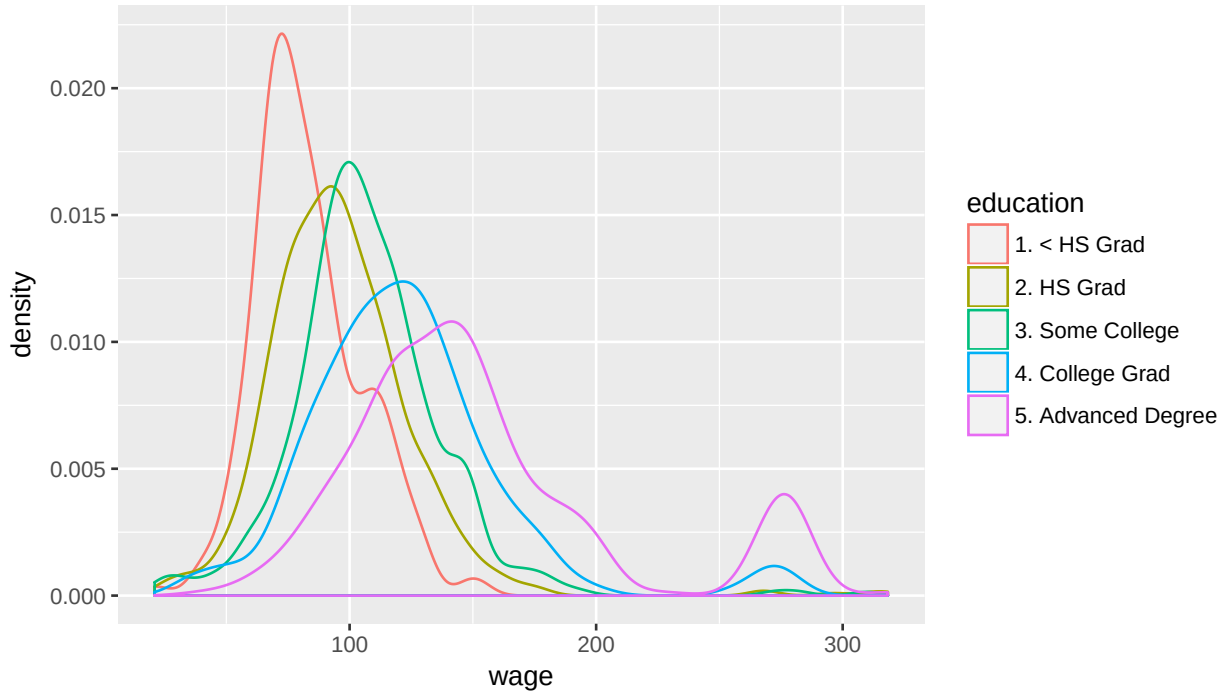

```
# Tables
t1 <- table(cutWage,training$jobclass)
t1
```

```
##
## cutWage       1. Industrial 2. Information
##   [ 20.9, 93)           446            267
##   [ 93.0,119)           358            357
##   [118.9,318]           265            409
```

```
prop.table(t1,1)
```

```
##
## cutWage       1. Industrial 2. Information
##   [ 20.9, 93)     0.6255259      0.3744741
##   [ 93.0,119)     0.5006993      0.4993007
##   [118.9,318]     0.3931751      0.6068249
```

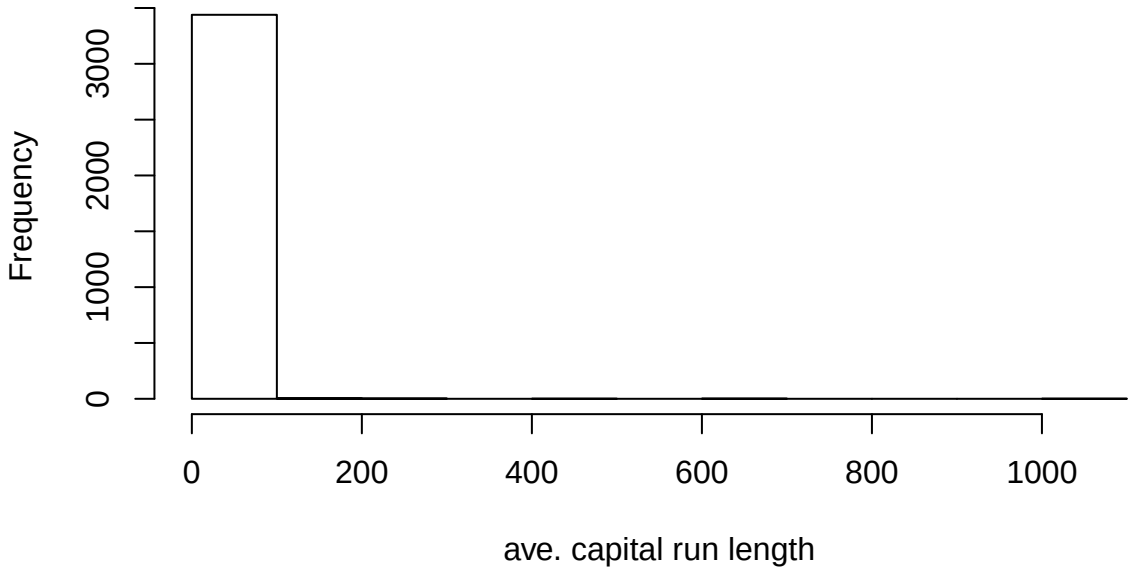
```
# Density plots
qplot(wage,colour=education,data=training,geom="density")
```




```
# Notes:
# 1. Make your plots only in the training set (don't use the test test for exploration)
# 2. Things you should be looking for: `Imbalance in outcomes/predictors; outliers;
#    Groups of points not explained by a predictor; skewed variable.
```

Part 7: Basic Preprocessing

```
# Why preprocess?
library(caret); library(kernlab); data(spam)
inTrain <- createDataPartition(y=spam$type, p=0.75, list=FALSE)
training <- spam[inTrain,]
testing <- spam[-inTrain,]
hist(training$capitalAve,main="",xlab="ave. capital run length")
```




```
mean(training$capitalAve)
```

```
## [1] 4.716994
```

```
sd(training$capitalAve) # it's skewed and highly variable
```

```
## [1] 26.82555
```

```
# Method 1: Standardizing -  training set
trainCapAve <- training$capitalAve
trainCapAveS <- (trainCapAve  - mean(trainCapAve))/sd(trainCapAve)
mean(trainCapAveS)
```

```
## [1] -6.935532e-18
```

```
sd(trainCapAveS)
```

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

```
# Method 1: Standardizing - test set
testCapAve <- testing$capitalAve
testCapAveS <- (testCapAve  - mean(trainCapAve))/sd(trainCapAve)
mean(testCapAveS)
```

```
## [1] 0.07077199
```

```
sd(testCapAveS)
```

```
## [1] 1.610786
```

```
# One thing to keep in mind is when we apply a prediction algorithm to the test set,
# we have to be aware that we can only use parameters that we estimated in the training set,
# so we use the mean and sd from the training set to standardize the test set.
```

```r
# Method 2: Standardizing training set - preProcess function
preObj <- preProcess(training[,-58],method=c("center","scale"))
trainCapAveS <- predict(preObj, training[,-58])$capitalAve
mean(trainCapAveS)
```

```
## [1] -6.935532e-18
```

```r
sd(trainCapAveS)
```

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

```r
# Method 2: Standardizing testing set - preProcess function
testCapAveS <- predict(preObj, testing[,-58])$capitalAve
mean(testCapAveS)
```

```
## [1] 0.07077199
```

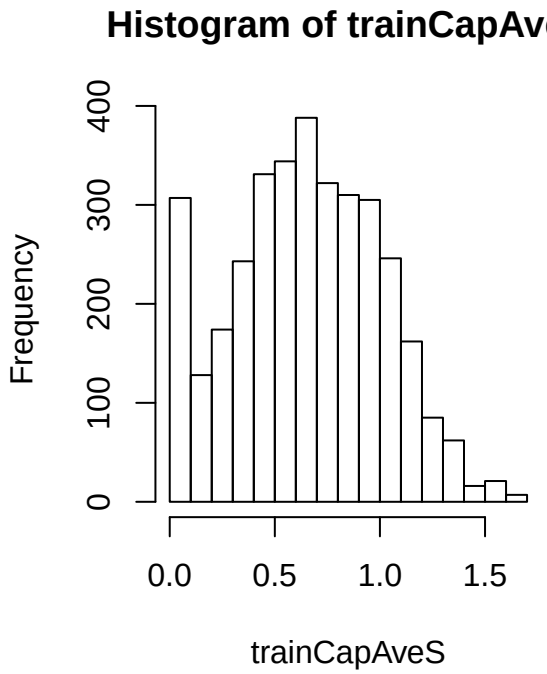
```r
sd(testCapAveS)
```

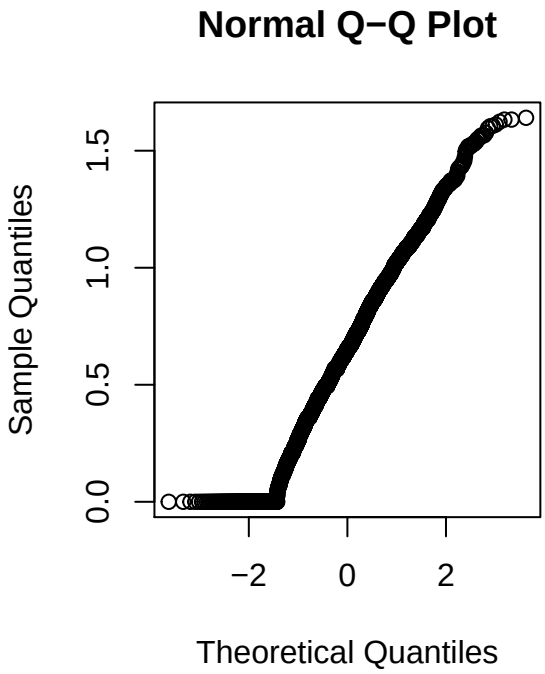
```
## [1] 1.610786
```

```r
# Method 3: Standardizing directly in train function by the preProcess argument
set.seed(32343)
modelFit <- train(type ~.,data=training, preProcess=c("center","scale"),method="glm")
modelFit
```

```
## Generalized Linear Model
##
## 3451 samples
##   57 predictor
##    2 classes: 'nonspam', 'spam'
##
## Pre-processing: centered (57), scaled (57)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 3451, 3451, 3451, 3451, 3451, 3451, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9208656  0.8332911
##
##
```

```r
# Method 4: Standardizing - Box-Cox transforms
# Centering and scaling is one approach, that you can remove very strongly
# biased predictors or preditors that have super high variability.
# The other thing you can do is use other different kinds of transformations.
# One example is the box-cox transforms, which are a set of transformations that take continuous data,
# and try to make them look like normal data by estimating a specific set of parameters using maximum l
preObj <- preProcess(training[,-58],method=c("BoxCox"))
trainCapAveS <- predict(preObj,training[,-58])$capitalAve
par(mfrow=c(1,2)); hist(trainCapAveS); qqnorm(trainCapAveS)
```






```
# Method 5: Standardizing - Imputing data, very common to have missing data
# when you using missing data in the data sets, the prediction algorithm oftern fail,
# prediction algorithm are built not to handle missing data in most cases.
# If you have missing data, you can impute them using sth called k-nearest neighbor's imputation.
set.seed(13343)

# Make some values NA
training$capAve <- training$capitalAve
selectNA <- rbinom(dim(training)[1],size=1,prob=0.05)==1
training$capAve[selectNA] <- NA

# Impute and standardize
preObj <- preProcess(training[,-58],method="knnImpute")
# k-nearest neighbors imputation finds the k. So if the k=10, then 10 nearest data vectors that
# look most like data vector with the missing value, and average the values of the variable
# that's missing and compute them at that position. Then we can predict on training set.
library(RANN)
capAve <- predict(preObj, training[,-58])$capAve

# Standardize true values
capAveTruth <- training$capitalAve
capAveTruth <- (capAveTruth-mean(capAveTruth))/sd(capAveTruth)

# Standardizing - Imputing data
quantile(capAve - capAveTruth)
```

```
##            0%           25%           50%           75%          100%
## -1.1884293998 -0.0008836469  0.0005853112  0.0012905270  1.0145601207
```

```
quantile((capAve - capAveTruth)[selectNA])
```

```
##           0%          25%          50%          75%         100%
## -1.188429400 -0.011925252  0.003908074  0.025463933  1.014560121
```

```
quantile((capAve - capAveTruth)[!selectNA])
```

```
##            0%           25%           50%           75%          100%
## -0.9757406838 -0.0008010270  0.0005776479  0.0012414810  0.0018085649
```

```
# Notes:
# 1. Training and test must be processed in the same way.
# 2. Test transformations will likely be imperfect,
# especially if the test/training sets collected at different times.
# 3. Careful when transforming factor variables.
```

Part 8: Covariate creation

```
# Load example data
library(ISLR); library(caret); data(Wage);
inTrain <- createDataPartition(y=Wage$wage, p=0.7, list=FALSE)
training <- Wage[inTrain,]; testing <- Wage[-inTrain,]

# Common covariates to add, dummy variables
# So one idea is that's very common when building machine learning algorithms is to
# turn covariates that are qualitative, or factor variables, into what are called dummy variables
table(training$jobclass)
```

```
##
##  1. Industrial 2. Information
##           1051           1051
```

```
dummies <- dummyVars(wage ~ jobclass,data=training)
head(predict(dummies,newdata=training))
```

```
##        jobclass.1. Industrial jobclass.2. Information
## 86582                       0                       1
## 161300                      1                       0
## 155159                      0                       1
## 11443                       0                       1
## 376662                      0                       1
## 450601                      1                       0
```

```
# Removing zero covariates
nsv <- nearZeroVar(training,saveMetrics=TRUE); nsv
```

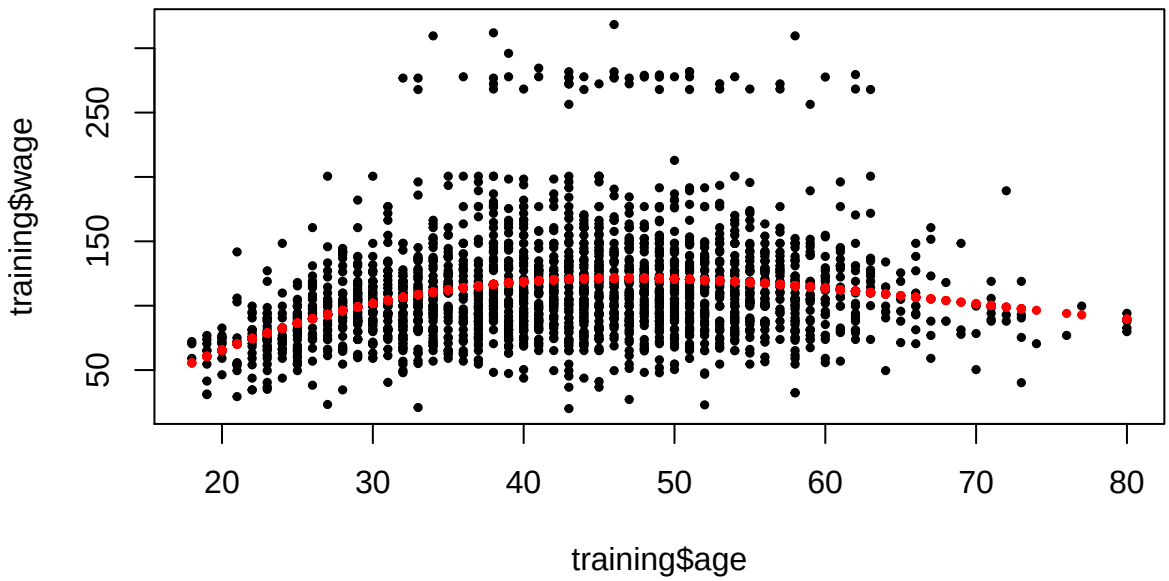
```
##            freqRatio percentUnique zeroVar   nzv
## year        1.037356    0.33301618   FALSE FALSE
## age         1.027027    2.85442436   FALSE FALSE
## sex         0.000000    0.04757374    TRUE  TRUE
## maritl      3.272931    0.23786870   FALSE FALSE
## race        8.938776    0.19029496   FALSE FALSE
## education   1.389002    0.23786870   FALSE FALSE
## region      0.000000    0.04757374    TRUE  TRUE
## jobclass    1.000000    0.09514748   FALSE FALSE
```

```
## health      2.468647    0.09514748   FALSE FALSE
## health_ins  2.352472    0.09514748   FALSE FALSE
## logwage     1.061728   19.17221694   FALSE FALSE
## wage        1.061728   19.17221694   FALSE FALSE
```

```
# Spline basis
library(splines)
bsBasis <- bs(training$age,df=3)
# create a ploynomial variable including age, age squared and age cubic.
head(bsBasis)
```

```
##              1          2           3
## [1,] 0.2368501 0.02537679 0.000906314
## [2,] 0.4163380 0.32117502 0.082587862
## [3,] 0.4308138 0.29109043 0.065560908
## [4,] 0.3625256 0.38669397 0.137491189
## [5,] 0.3063341 0.42415495 0.195763821
## [6,] 0.4241549 0.30633413 0.073747105
```

```
# Fitting curves with splines
lm1 <- lm(wage ~ bsBasis,data=training)
plot(training$age,training$wage,pch=19,cex=0.5)
points(training$age,predict(lm1,newdata=training),col="red",pch=19,cex=0.5)
```




```
# Splines on the test set
head(predict(bsBasis,age=testing$age))
```

```
##              1          2           3
## [1,] 0.2368501 0.02537679 0.000906314
## [2,] 0.4163380 0.32117502 0.082587862
## [3,] 0.4308138 0.29109043 0.065560908
```

```
## [4,] 0.3625256 0.38669397 0.137491189
## [5,] 0.3063341 0.42415495 0.195763821
## [6,] 0.4241549 0.30633413 0.073747105
```

Part 9: preProcessing PCA

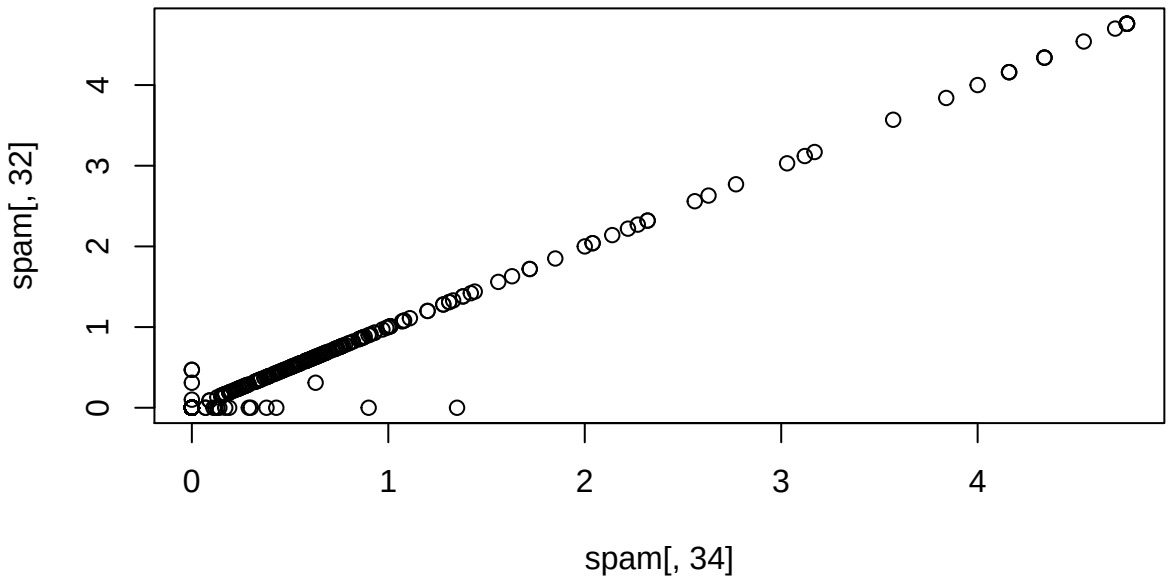
```
# Correlated predictors
library(caret); library(kernlab); data(spam)
inTrain <- createDataPartition(y=spam$type, p=0.75, list=FALSE)
training <- spam[inTrain,]
testing <- spam[-inTrain,]

M <- abs(cor(training[,-58]))
diag(M) <- 0
which(M > 0.8,arr.ind=T)
```

```
##        row col
## num415  34  32
## direct  40  32
## num857  32  34
## direct  40  34
## num857  32  40
## num415  34  40
```

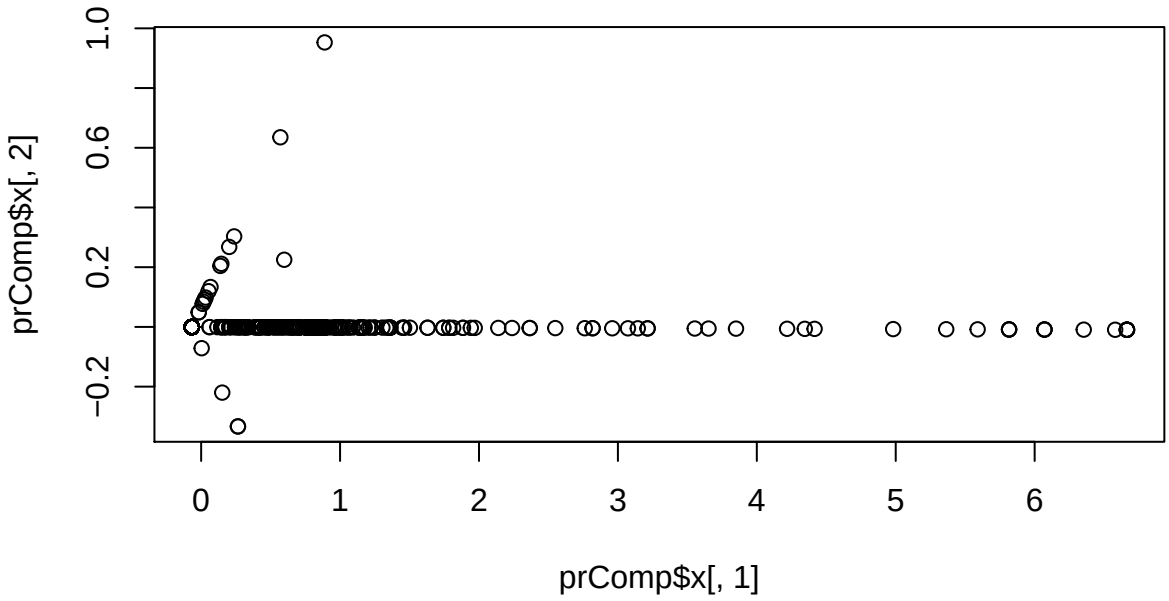
```
names(spam)[c(34,32)]
```

```
## [1] "num415" "num857"
```

```
plot(spam[,34],spam[,32])
```



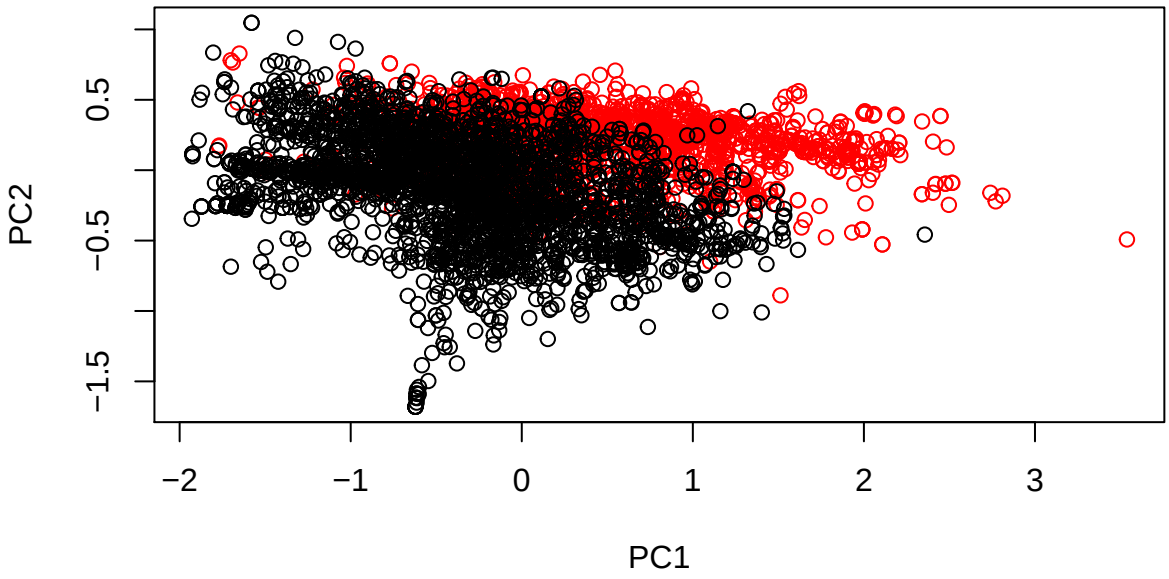
```r
# Principal components in R - prcomp
smallSpam <- spam[,c(34,32)]
prComp <- prcomp(smallSpam)
plot(prComp$x[,1],prComp$x[,2])
```



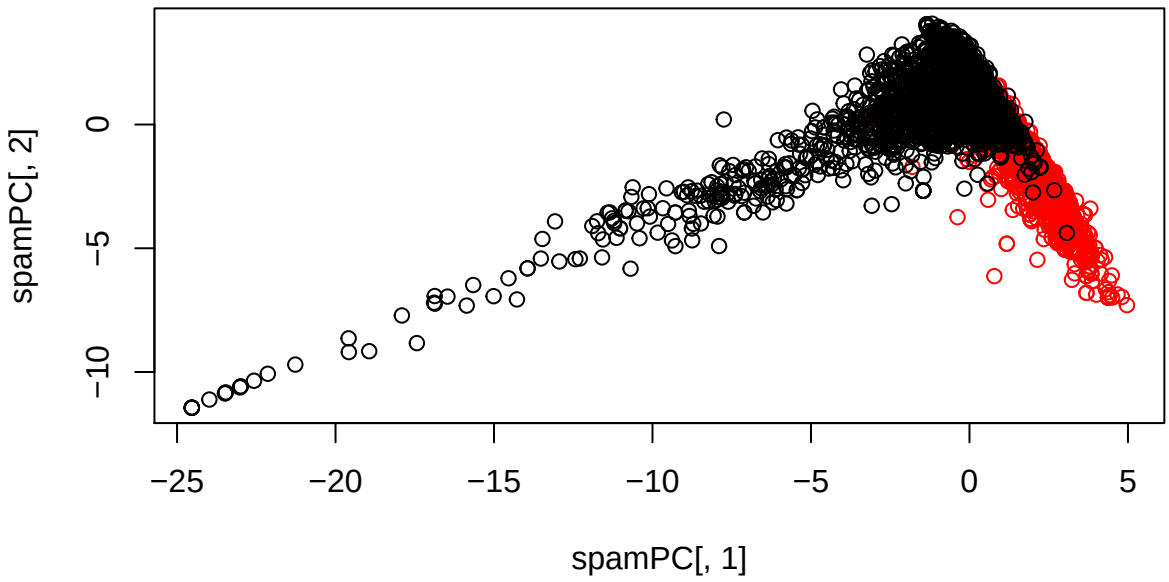

```r
prComp$rotation
```

```
##              PC1        PC2
## num415 0.7080625  0.7061498
## num857 0.7061498 -0.7080625
```

```r
# rotation is basically how it's summing up the two variables to get each of the principal components.

# PCA on SPAM data
typeColor <- ((spam$type=="spam")*1 + 1)
prComp <- prcomp(log10(spam[,-58]+1))
# log10 transformation to make the data look a little bit more Gaussian.
plot(prComp$x[,1],prComp$x[,2],col=typeColor,xlab="PC1",ylab="PC2")
```




```
# PCA with caret
preProc <- preProcess(log10(spam[,-58]+1),method="pca",pcaComp=2)
spamPC <- predict(preProc,log10(spam[,-58]+1))
plot(spamPC[,1],spamPC[,2],col=typeColor)
```



```
# Preprocessing with PCA for training set
preProc <- preProcess(log10(training[,-58]+1),method="pca",pcaComp=2)
trainPC <- predict(preProc,log10(training[,-58]+1))
# modelFit <- train(type ~ ., data=trainPC, method="glm")

# Preprocessing with PCA fot test set
# testPC <- predict(preProc,log10(testing[,-58]+1))
# confusionMatrix(testing$type,predict(modelFit,testPC))

# Alternative (sets # of PCs)
modelFit <- train(type ~ .,method="glm", preProcess="pca",data=training)
confusionMatrix(testing$type,predict(modelFit,testing))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction nonspam spam
##    nonspam     658   39
##    spam         50  403
##
##                Accuracy : 0.9226
##                  95% CI : (0.9056, 0.9374)
##     No Information Rate : 0.6157
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8372
##  Mcnemar's Test P-Value : 0.2891
##
##             Sensitivity : 0.9294
##             Specificity : 0.9118
##          Pos Pred Value : 0.9440
##          Neg Pred Value : 0.8896
##              Prevalence : 0.6157
##          Detection Rate : 0.5722
##    Detection Prevalence : 0.6061
##       Balanced Accuracy : 0.9206
##
##        'Positive' Class : nonspam
##
```

Part 10: Predicting with Regression

```
# Example: Old faithful eruptions
library(caret);data(faithful); dim(faithful); set.seed(333)
```

```
## [1] 272   2
```

```
inTrain <- createDataPartition(y=faithful$waiting, p=0.5, list=FALSE)
trainFaith <- faithful[inTrain,]; testFaith <- faithful[-inTrain,]
head(trainFaith)
```

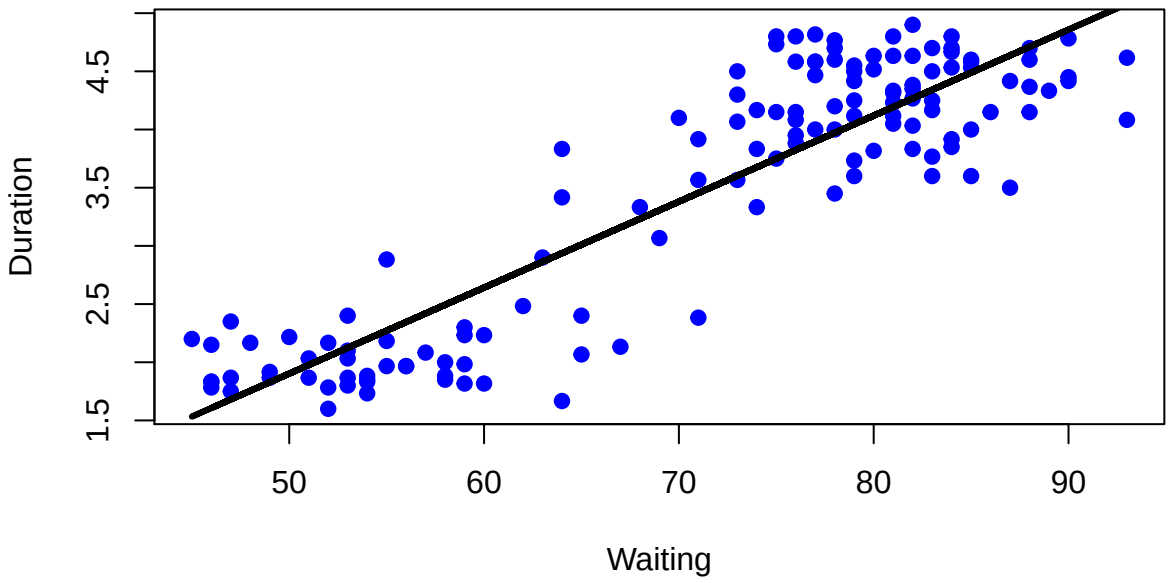
```
##   eruptions waiting
## 1     3.600      79
```

```
## 3     3.333      74
## 5     4.533      85
## 6     2.883      55
## 7     4.700      88
## 8     3.600      85
```

```
# Eruption duration versus waiting time
plot(trainFaith$waiting,trainFaith$eruptions,pch=19,col="blue",xlab="Waiting",ylab="Duration")

# Fit a linear model
lm1 <- lm(eruptions ~ waiting,data=trainFaith)
summary(lm1)
```

```
##
## Call:
## lm(formula = eruptions ~ waiting, data = trainFaith)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.26990 -0.34789  0.03979  0.36589  1.05020
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.792739   0.227869  -7.867 1.04e-12 ***
## waiting      0.073901   0.003148  23.474  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.495 on 135 degrees of freedom
## Multiple R-squared:  0.8032, Adjusted R-squared:  0.8018
## F-statistic:   551 on 1 and 135 DF,  p-value: < 2.2e-16
```

```
# Model fit
plot(trainFaith$waiting,trainFaith$eruptions,pch=19,col="blue",xlab="Waiting",ylab="Duration")
lines(trainFaith$waiting,lm1$fitted,lwd=3)
```



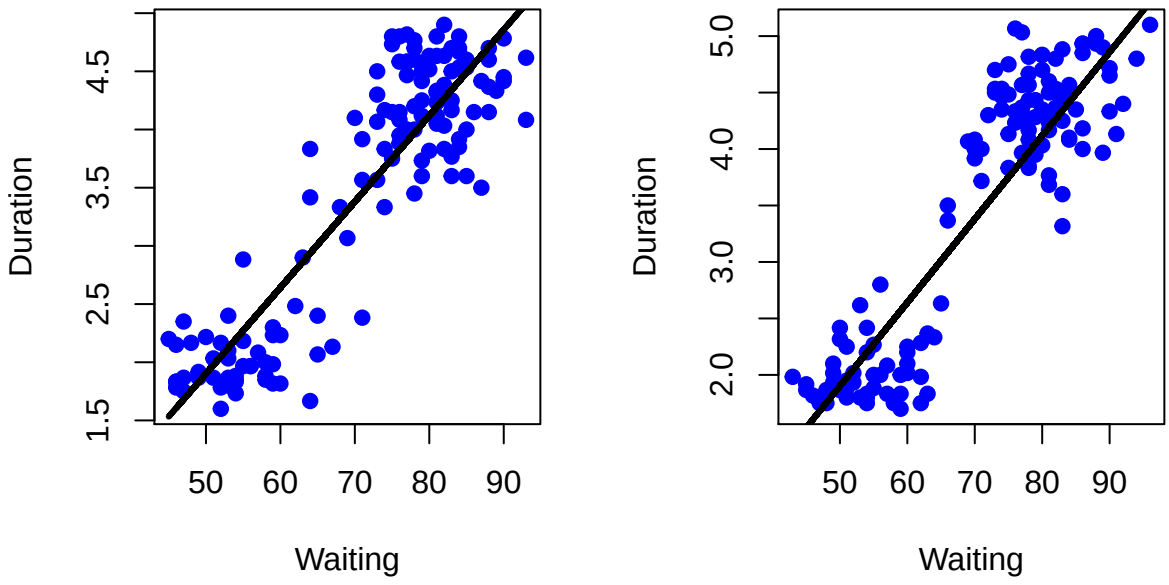

```
# Predict a new value
coef(lm1)[1] + coef(lm1)[2]*80
```

```
## (Intercept)
##    4.119307
```

```
newdata <- data.frame(waiting=80)
predict(lm1,newdata)
```

```
##        1
## 4.119307
```

```
# Plot predictions - training and test
par(mfrow=c(1,2))
plot(trainFaith$waiting,trainFaith$eruptions,pch=19,col="blue",xlab="Waiting",ylab="Duration")
lines(trainFaith$waiting,predict(lm1),lwd=3)
plot(testFaith$waiting,testFaith$eruptions,pch=19,col="blue",xlab="Waiting",ylab="Duration")
lines(testFaith$waiting,predict(lm1,newdata=testFaith),lwd=3)
```




```
# Get training set/test set errors
# 1. Calculate RMSE on training
sqrt(sum((lm1$fitted-trainFaith$eruptions)^2))
```

```
## [1] 5.75186
```

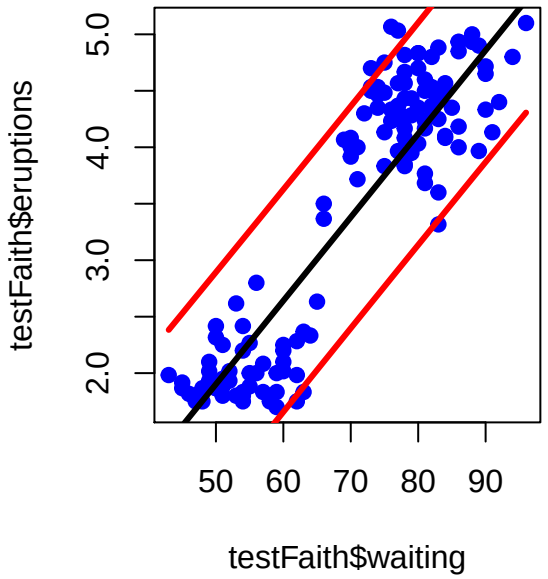
```
# 2. Calculate RMSE on test
sqrt(sum((predict(lm1,newdata=testFaith)-testFaith$eruptions)^2))
```

```
## [1] 5.838559
```

```
# Prediction intervals
pred1 <- predict(lm1,newdata=testFaith,interval="prediction")
ord <- order(testFaith$waiting)
plot(testFaith$waiting,testFaith$eruptions,pch=19,col="blue")
matlines(testFaith$waiting[ord],pred1[ord,],type="l", col=c(1,2,2),lty = c(1,1,1), lwd=3)

# Same process with caret
modFit <- train(eruptions ~ waiting,data=trainFaith,method="lm")
summary(modFit$finalModel)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.26990 -0.34789  0.03979  0.36589  1.05020
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept) -1.792739   0.227869  -7.867 1.04e-12 ***
## waiting      0.073901   0.003148  23.474  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.495 on 135 degrees of freedom
## Multiple R-squared:  0.8032, Adjusted R-squared:  0.8018
## F-statistic:   551 on 1 and 135 DF,  p-value: < 2.2e-16
```



Part 11: Predicting with Regression multiple covariate

```
# Example: Wage data
library(ISLR); library(ggplot2); library(caret);
data(Wage); Wage <- subset(Wage,select=-c(logwage))
summary(Wage)
```

```
##       year           age               sex                    maritl
##  Min.   :2003   Min.   :18.00   1. Male  :3000   1. Never Married: 648
##  1st Qu.:2004   1st Qu.:33.75   2. Female:   0   2. Married      :2074
##  Median :2006   Median :42.00                    3. Widowed      :  19
##  Mean   :2006   Mean   :42.41                    4. Divorced     : 204
##  3rd Qu.:2008   3rd Qu.:51.00                    5. Separated    :  55
##  Max.   :2009   Max.   :80.00
##
##        race                     education                     region
##  1. White:2480   1. < HS Grad      :268   2. Middle Atlantic   :3000
##  2. Black: 293   2. HS Grad        :971   1. New England       :   0
##  3. Asian: 190   3. Some College   :650   3. East North Central:   0
```

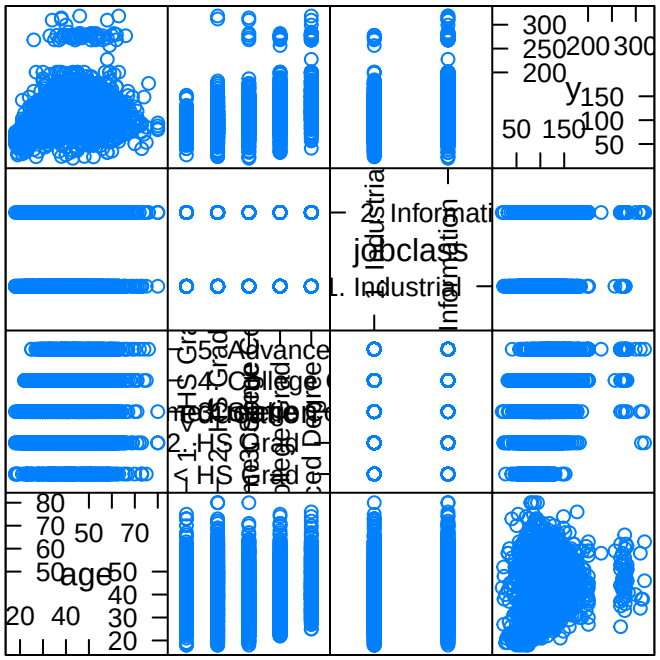
```
##  4. Other:  37   4. College Grad   :685   4. West North Central:   0
##                  5. Advanced Degree:426   5. South Atlantic    :   0
##                                           6. East South Central:   0
##                                           (Other)              :   0
##            jobclass               health      health_ins
##  1. Industrial :1544   1. <=Good     : 858   1. Yes:2083
##  2. Information:1456   2. >=Very Good:2142   2. No : 917
##
##
##
##
##
##       wage
##  Min.   : 20.09
##  1st Qu.: 85.38
##  Median :104.92
##  Mean   :111.70
##  3rd Qu.:128.68
##  Max.   :318.34
##
```

```
# Get training/test sets
inTrain <- createDataPartition(y=Wage$wage, p=0.7, list=FALSE)
training <- Wage[inTrain,]; testing <- Wage[-inTrain,]
dim(training); dim(testing)
```

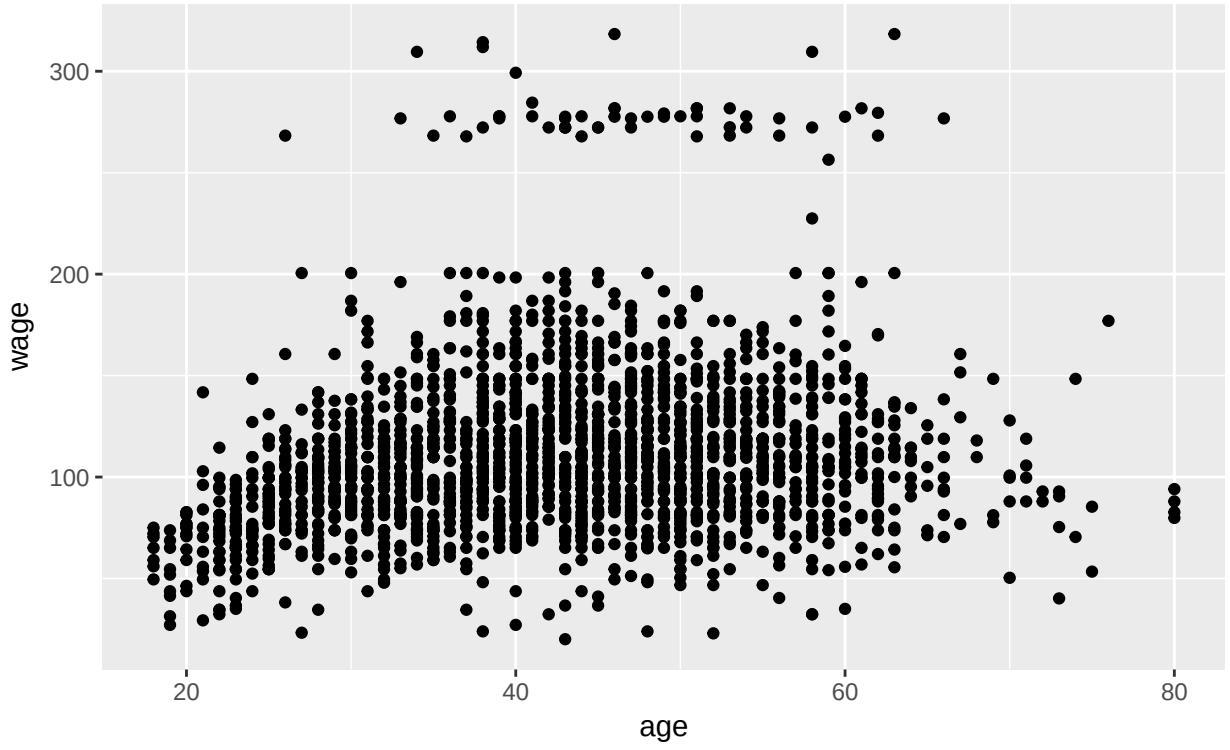
```
## [1] 2102   11
```

```
## [1] 898  11
```

```
# Feature plot
featurePlot(x=training[,c("age","education","jobclass")], y = training$wage,
            plot="pairs")
```

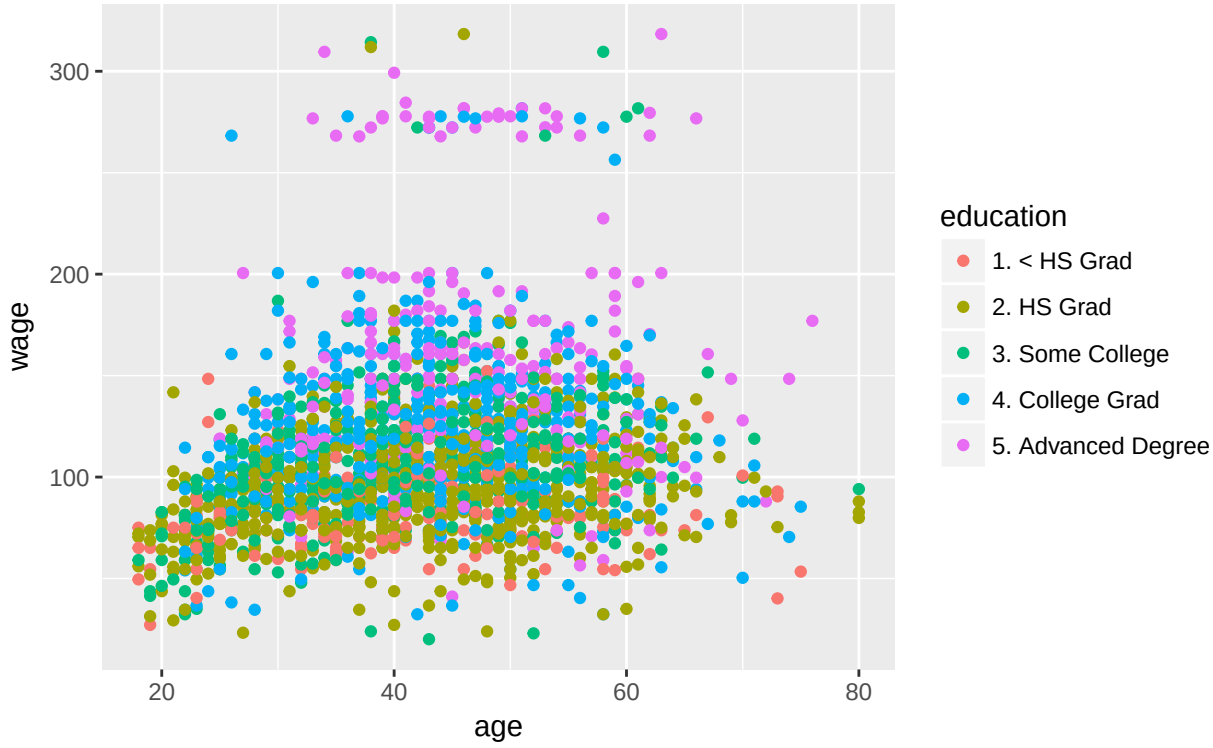



```
# Plot age versus wage
qplot(age,wage,data=training)
```




```
# Plot age versus wage colour by jobclass
qplot(age,wage,colour=jobclass,data=training)
```


```
# Plot age versus wage colour by education
qplot(age,wage,colour=education,data=training)
```

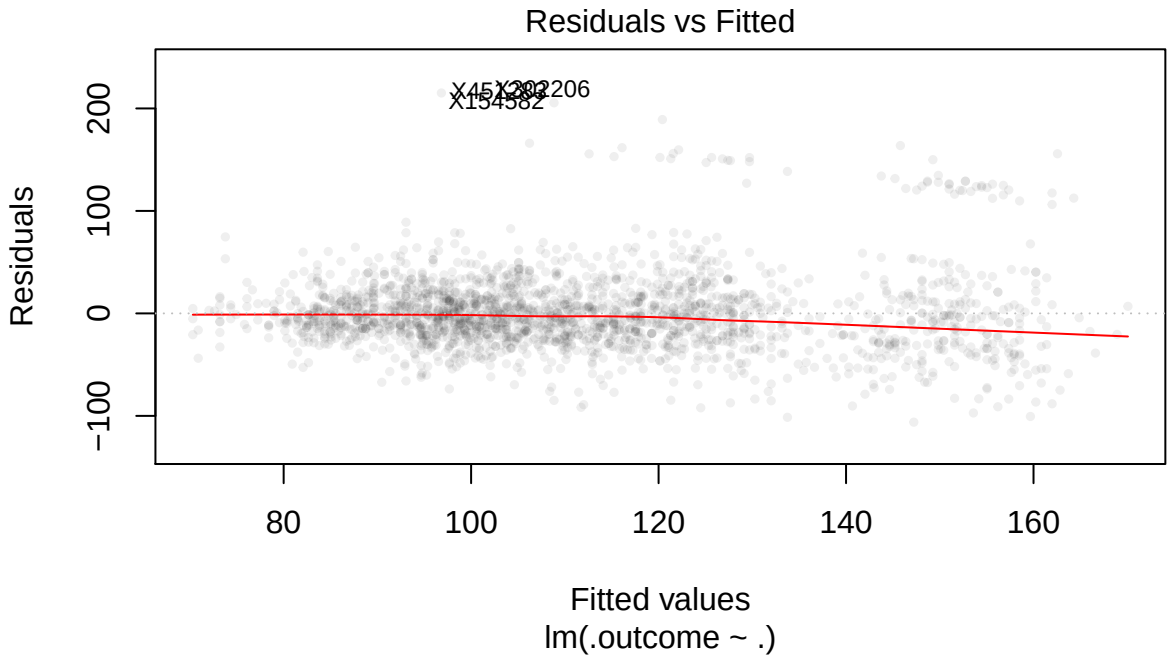



```
# Fit a linear model
# Education levels: 1 = HS Grad, 2 = Some College, 3 = College Grad, 4 = Advanced Degree
```

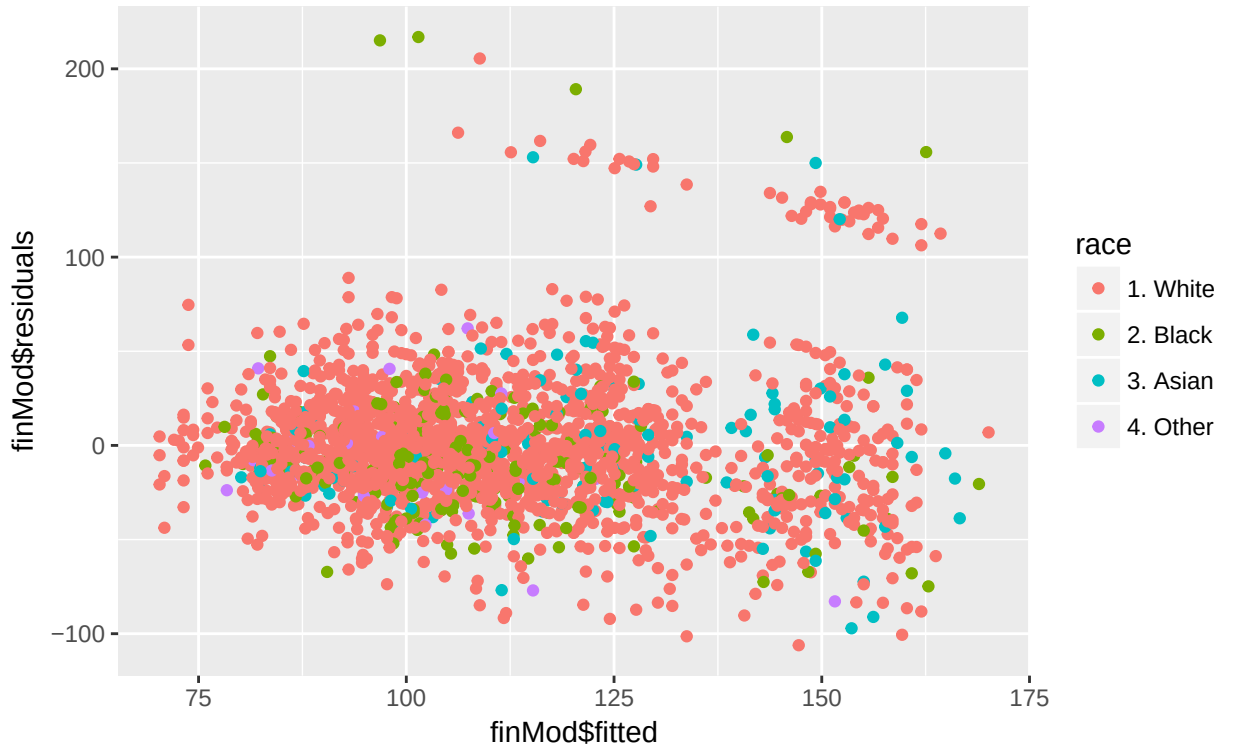
```r
modFit<- train(wage ~ age + jobclass + education, method = "lm",data=training)
finMod <- modFit$finalModel
print(modFit)
```

```
## Linear Regression
##
## 2102 samples
##    3 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 2102, 2102, 2102, 2102, 2102, 2102, ...
## Resampling results:
##
##   RMSE      Rsquared
##   36.06666  0.2517055
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
##
```

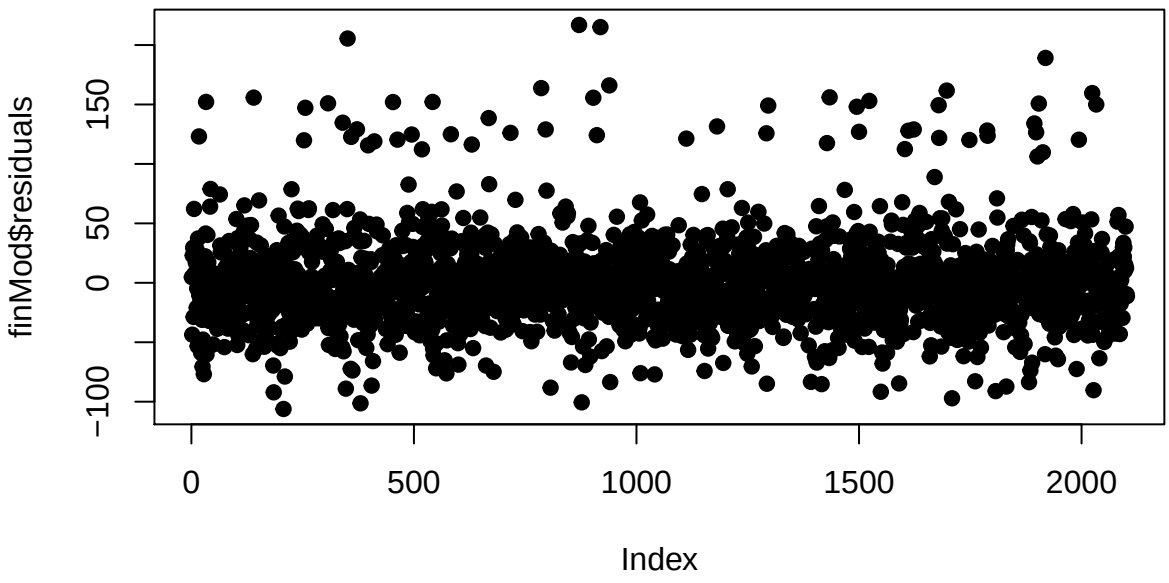
```r
# Diagnostics
plot(finMod,1,pch=19,cex=0.5,col="#00000010")
```




```r
# Color by variables not used in the model
qplot(finMod$fitted,finMod$residuals,colour=race,data=training)
```



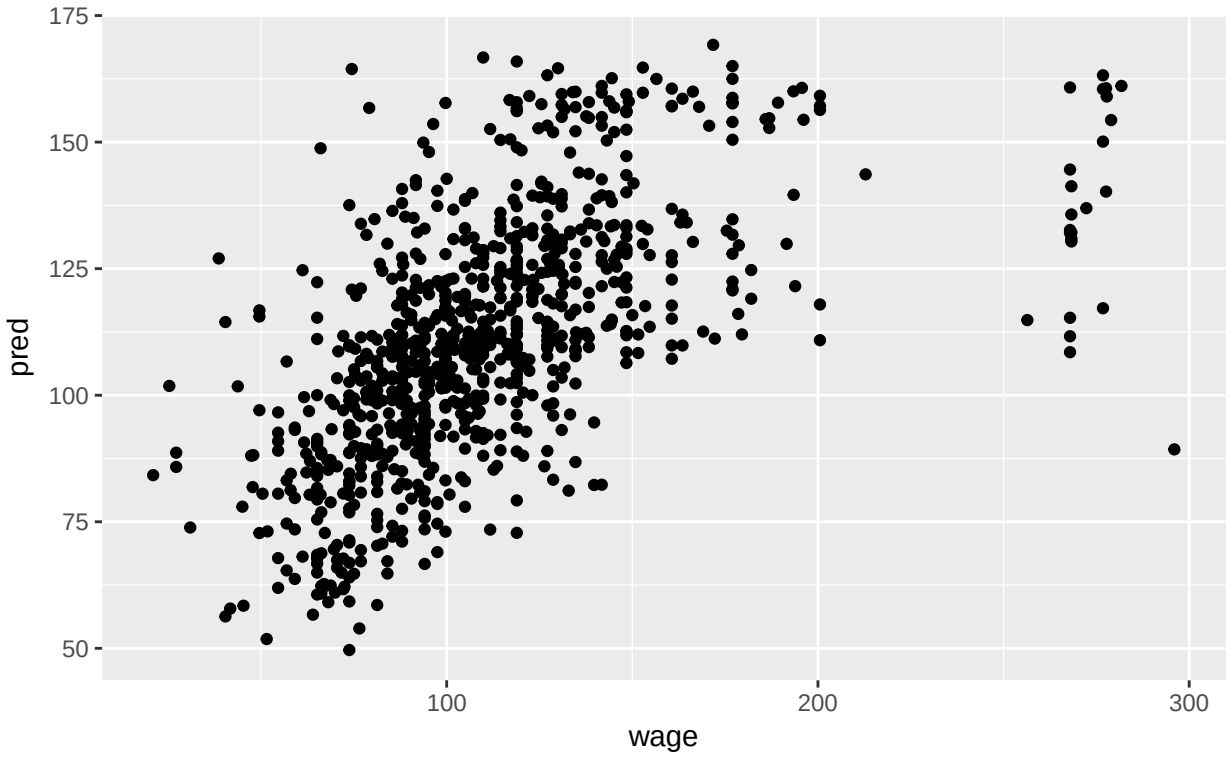

```
# Plot by index
plot(finMod$residuals,pch=19)
```




```
# Predicted versus truth in test set
pred <- predict(modFit, testing)
```

```
qplot(wage,pred,colour=year,data=testing)
```


```
# If you want to use all covariates
modFitAll<- train(wage ~ .,data=training,method="lm")
pred <- predict(modFitAll, testing)
qplot(wage,pred,data=testing)
```



Part 12: Predicting with Trees

```
# Example: Iris Data
data(iris); library(ggplot2)
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

```
table(iris$Species)
```

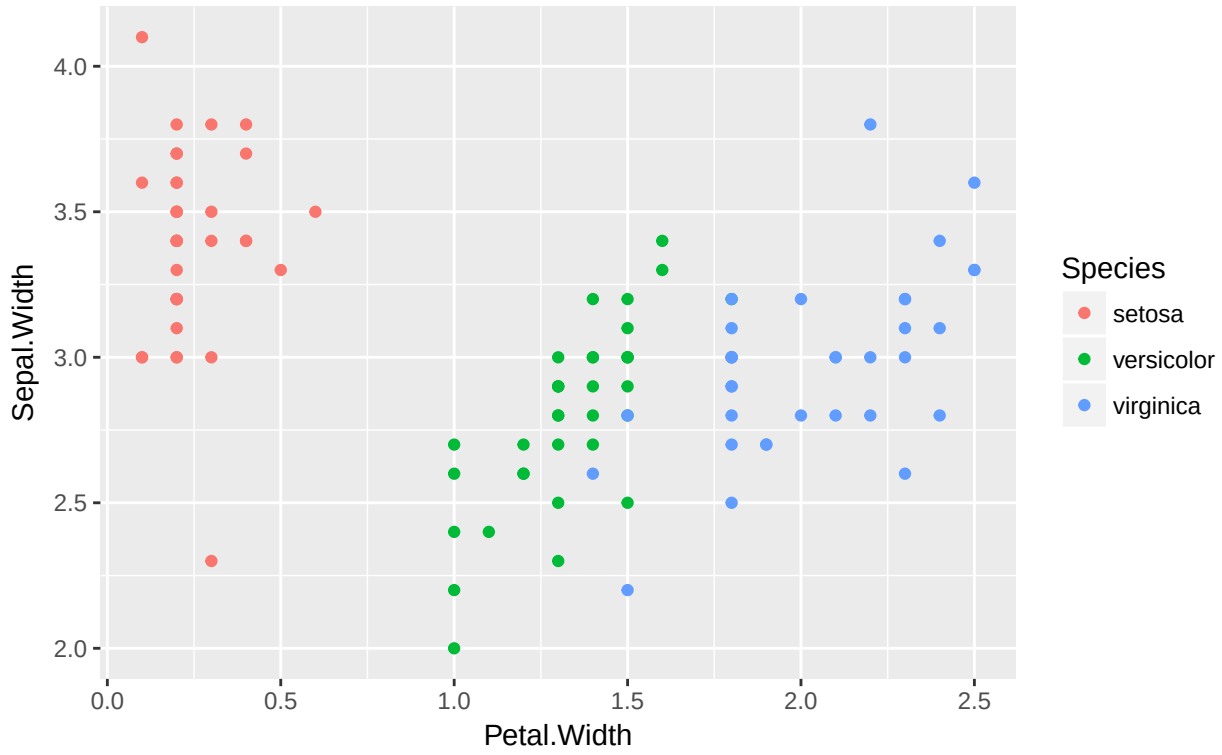
```
##
##     setosa versicolor  virginica
##         50         50         50
```

```
# Create training and test sets
inTrain <- createDataPartition(y=iris$Species,p=0.7, list=FALSE)
training <- iris[inTrain,]
testing <- iris[-inTrain,]
dim(training); dim(testing)
```

```
## [1] 105   5
```

```
## [1] 45  5
```

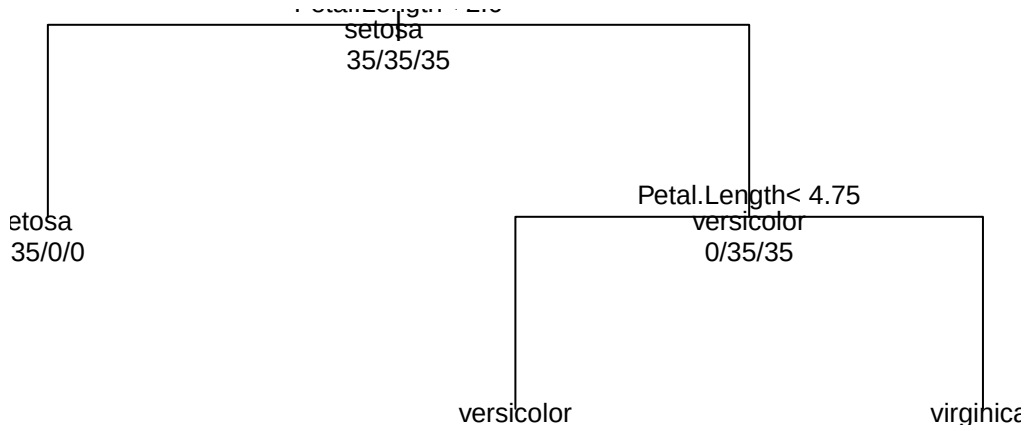
```
# Iris petal widths/sepal width
qplot(Petal.Width,Sepal.Width,colour=Species,data=training)
```



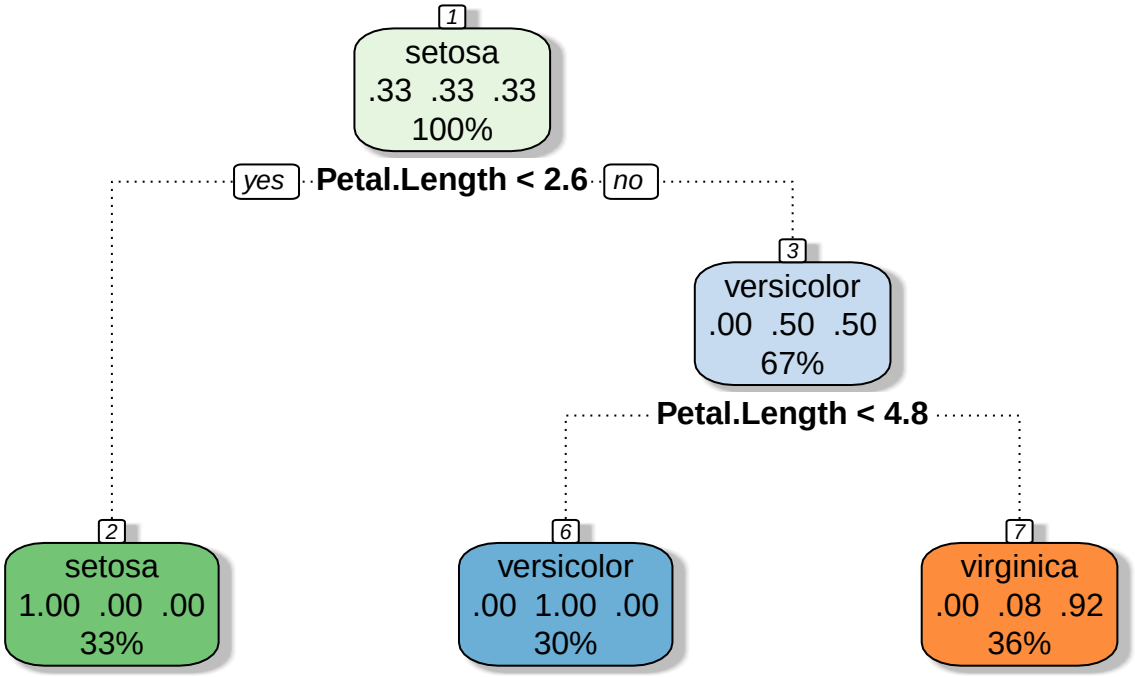

```
# Iris petal widths/sepal width
library(caret)
modFit <- train(Species ~ .,method="rpart",data=training)
print(modFit$finalModel)
```

```
## n= 105
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 105 70 setosa (0.33333333 0.33333333 0.33333333)
##   2) Petal.Length< 2.6 35  0 setosa (1.00000000 0.00000000 0.00000000) *
##   3) Petal.Length>=2.6 70 35 versicolor (0.00000000 0.50000000 0.50000000)
##     6) Petal.Length< 4.75 32  0 versicolor (0.00000000 1.00000000 0.00000000) *
##     7) Petal.Length>=4.75 38  3 virginica (0.00000000 0.07894737 0.92105263) *
```

```
# Plot tree
plot(modFit$finalModel, uniform=TRUE,
      main="Classification Tree")
text(modFit$finalModel, use.n=TRUE, all=TRUE, cex=.8)
```




```
# Prettier plots
library(rattle)
fancyRpartPlot(modFit$finalModel)
```



```
# Predicting new values
predict(modFit,newdata=testing)
```

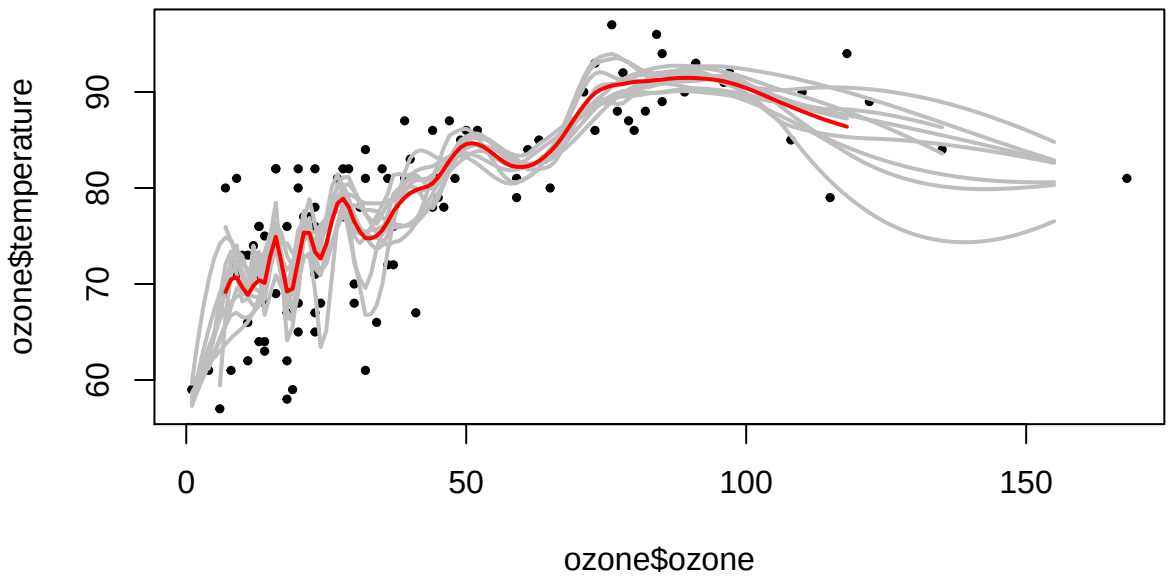
```
##  [1] setosa     setosa     setosa     setosa     setosa     setosa
##  [7] setosa     setosa     setosa     setosa     setosa     setosa
## [13] setosa     setosa     setosa     virginica  versicolor versicolor
## [19] versicolor versicolor versicolor versicolor virginica  versicolor
## [25] virginica  versicolor versicolor versicolor versicolor versicolor
## [31] versicolor virginica  virginica  virginica  virginica  virginica
## [37] virginica  virginica  virginica  virginica  virginica  virginica
## [43] virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

Part 13: Bagging

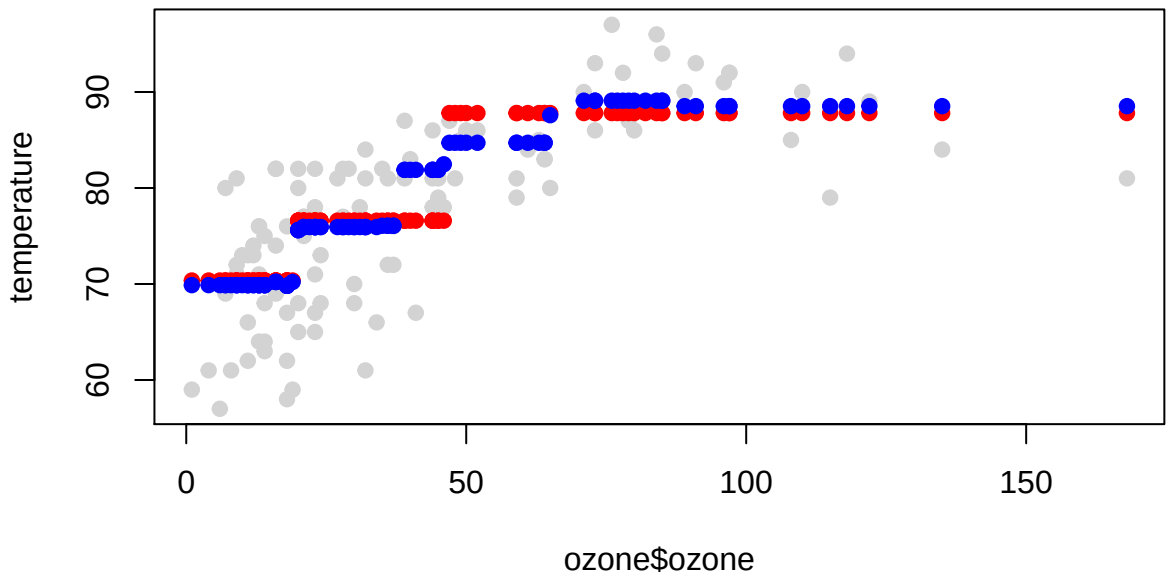
```
## Ozone data
library(ElemStatLearn); data(ozone,package="ElemStatLearn")
ozone <- ozone[order(ozone$ozone),]
head(ozone)
```

```
##     ozone radiation temperature wind
## 17      1         8          59  9.7
## 19      4        25          61  9.7
## 14      6        78          57 18.4
## 45      7        48          80 14.3
## 106     7        49          69 10.3
## 7       8        19          61 20.1
```

```
# Bagged loess
ll <- matrix(NA,nrow=10,ncol=155)
for(i in 1:10){
  ss <- sample(1:dim(ozone)[1],replace=T)
  ozone0 <- ozone[ss,]
  ozone0 <- ozone0[order(ozone0$ozone),]
  loess0 <- loess(temperature ~ ozone,data=ozone0,span=0.2)
  ll[i,] <- predict(loess0,newdata=data.frame(ozone=1:155))
}

# Bagged loess
plot(ozone$ozone,ozone$temperature,pch=19,cex=0.5)
for(i in 1:10){lines(1:155,ll[i,],col="grey",lwd=2)}
lines(1:155,apply(ll,2,mean),col="red",lwd=2)
```




```
# More bagging in caret
library(party)
predictors = data.frame(ozone=ozone$ozone)
temperature = ozone$temperature
treebag <- bag(predictors, temperature, B = 10,
               bagControl = bagControl(fit = ctreeBag$fit,
                                       predict = ctreeBag$pred,
                                       aggregate = ctreeBag$aggregate))

# Example of custom bagging (continued)
plot(ozone$ozone,temperature,col='lightgrey',pch=19)
points(ozone$ozone,predict(treebag$fits[[1]]$fit,predictors),pch=19,col="red")
points(ozone$ozone,predict(treebag,predictors),pch=19,col="blue")
```




```
# Parts of bagging
ctreeBag$fit
```

```
## function (x, y, ...)
## {
##     loadNamespace("party")
##     data <- as.data.frame(x)
##     data$y <- y
##     party::ctree(y ~ ., data = data)
## }
## <environment: namespace:caret>
```

```
# Parts of bagging
ctreeBag$pred
```

```
## function (object, x)
## {
##     if (!is.data.frame(x))
##         x <- as.data.frame(x)
##     obsLevels <- levels(object@data@get("response")[, 1])
##     if (!is.null(obsLevels)) {
##         rawProbs <- party::treeresponse(object, x)
##         probMatrix <- matrix(unlist(rawProbs), ncol = length(obsLevels),
##             byrow = TRUE)
##         out <- data.frame(probMatrix)
##         colnames(out) <- obsLevels
##         rownames(out) <- NULL
##     }
##     else out <- unlist(party::treeresponse(object, x))
##     out
```

```
## }
## <environment: namespace:caret>
```

```
# Parts of bagging
ctreeBag$aggregate
```

```
## function (x, type = "class")
## {
##     if (is.matrix(x[[1]]) | is.data.frame(x[[1]])) {
##         pooled <- x[[1]] & NA
##         classes <- colnames(pooled)
##         for (i in 1:ncol(pooled)) {
##             tmp <- lapply(x, function(y, col) y[, col], col = i)
##             tmp <- do.call("rbind", tmp)
##             pooled[, i] <- apply(tmp, 2, median)
##         }
##         if (type == "class") {
##             out <- factor(classes[apply(pooled, 1, which.max)],
##                 levels = classes)
##         }
##         else out <- as.data.frame(pooled)
##     }
##     else {
##         x <- matrix(unlist(x), ncol = length(x))
##         out <- apply(x, 1, median)
##     }
##     out
## }
## <environment: namespace:caret>
```

Part 14: Random Forests

```
# Iris data
data(iris); library(ggplot2)
inTrain <- createDataPartition(y=iris$Species, p=0.7, list=FALSE)
training <- iris[inTrain,]; testing <- iris[-inTrain,];
dim(training)
```

```
## [1] 105   5
```

```
dim(testing)
```

```
## [1] 45  5
```

```
# Random forests
library(randomForest); library(caret)
modFit <- train(Species ~ ., data=training,method="rf",prox=TRUE)
modFit
```

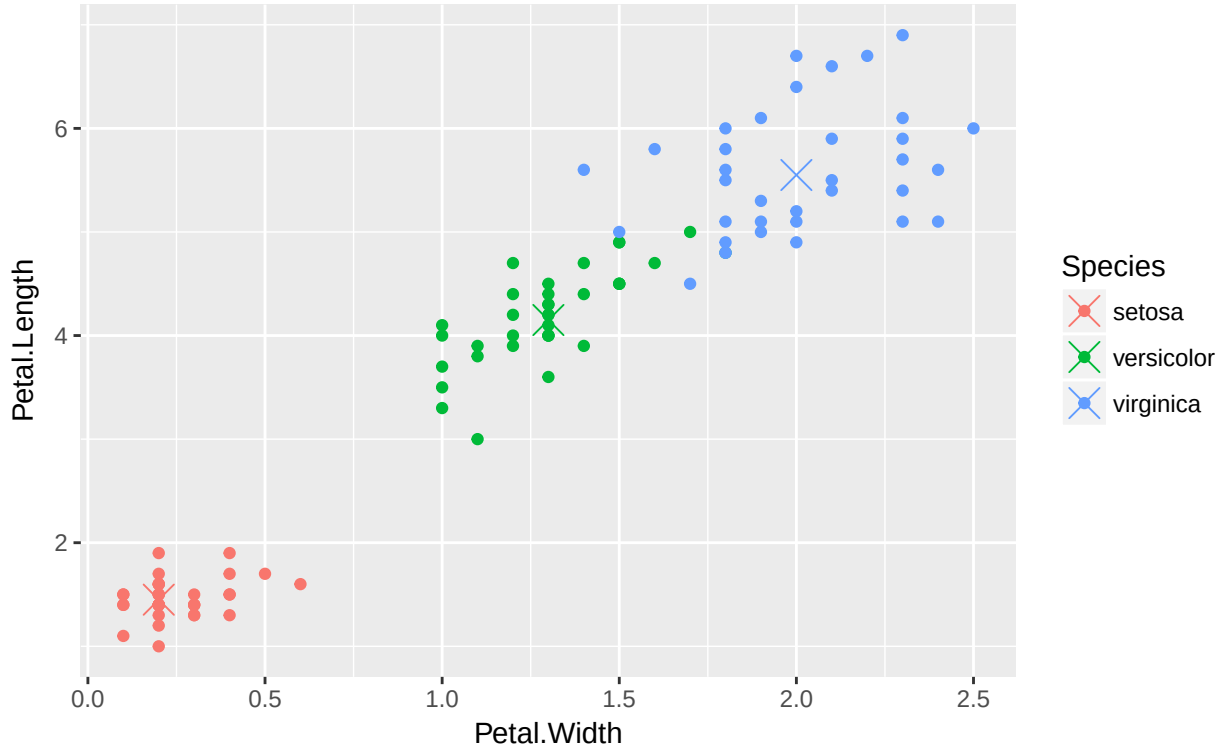
```
## Random Forest
##
## 105 samples
##   4 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
```

```
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 105, 105, 105, 105, 105, 105, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9445242  0.9154697
##   3     0.9422482  0.9120655
##   4     0.9476209  0.9202830
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 4.
```

```
# Getting a single tree
getTree(modFit$finalModel,k=2)
```

```
##   left daughter right daughter split var split point status prediction
## 1             2              3         4        0.70      1          0
## 2             0              0         0        0.00     -1          1
## 3             4              5         3        4.75      1          0
## 4             0              0         0        0.00     -1          2
## 5             6              7         4        1.55      1          0
## 6             8              9         1        6.20      1          0
## 7             0              0         0        0.00     -1          3
## 8             0              0         0        0.00     -1          3
## 9             0              0         0        0.00     -1          2
```

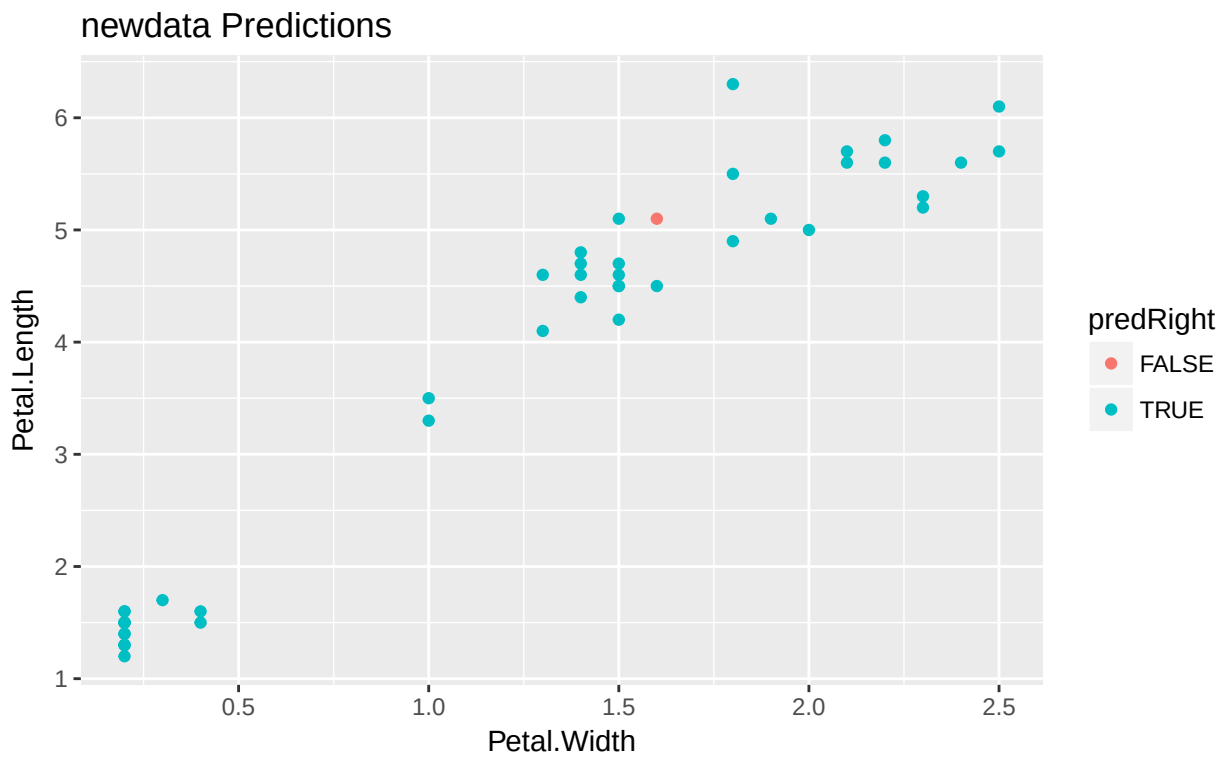
```
# Class "centers"
irisP <- classCenter(training[,c(3,4)], training$Species, modFit$finalModel$prox)
irisP <- as.data.frame(irisP); irisP$Species <- rownames(irisP)
p <- qplot(Petal.Width, Petal.Length, col=Species,data=training)
p + geom_point(aes(x=Petal.Width,y=Petal.Length,col=Species),size=5,shape=4,data=irisP)
```




```
# Predicting new values
pred <- predict(modFit,testing); testing$predRight <- pred==testing$Species
table(pred,testing$Species)
```

```
##
## pred         setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      0         14         0
##   virginica       0          1        15
```

```
# Predicting new values
qplot(Petal.Width,Petal.Length,colour=predRight,data=testing,main="newdata Predictions")
```



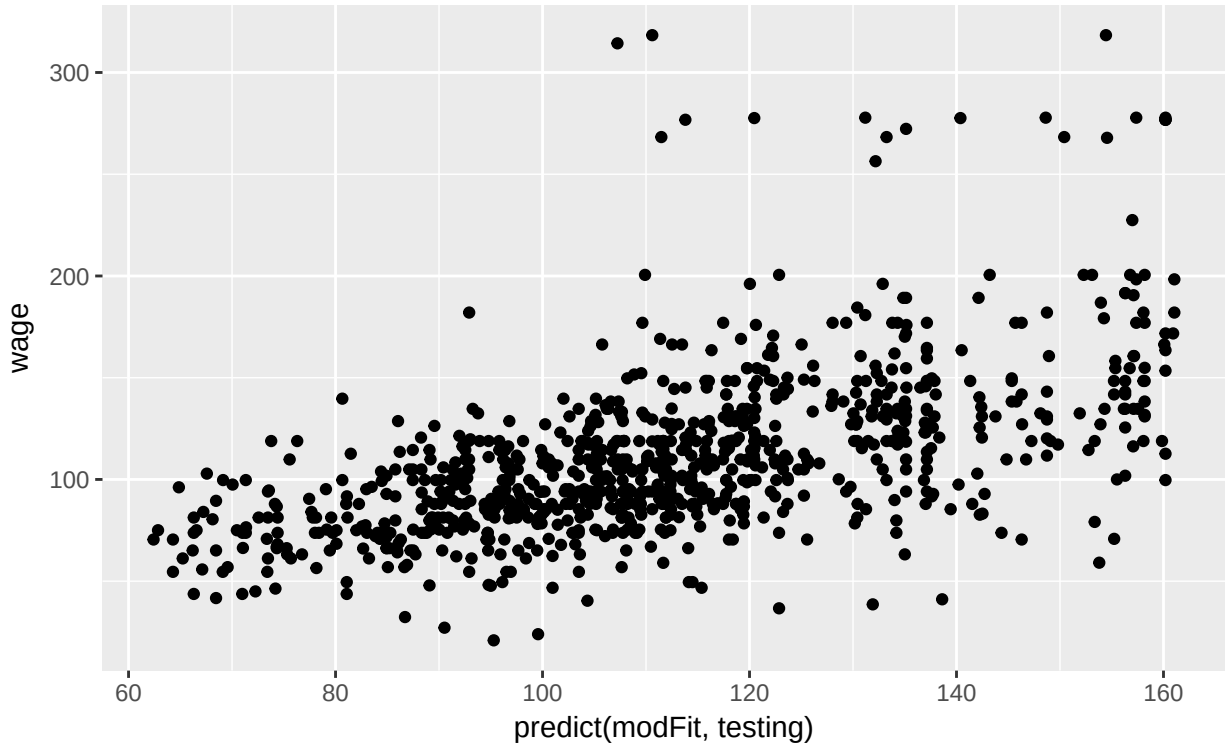
Part 15: Boosting

```
# Wage example
library(ISLR); data(Wage); library(ggplot2); library(caret);
Wage <- subset(Wage,select=-c(logwage))
inTrain <- createDataPartition(y=Wage$wage,p=0.7, list=FALSE)
training <- Wage[inTrain,]; testing <- Wage[-inTrain,]

# Fit the model
library(gbm)
modFit <- train(wage ~ ., method="gbm",data=training,verbose=FALSE)
print(modFit)
```

```
## Stochastic Gradient Boosting
##
## 2102 samples
##   10 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 2102, 2102, 2102, 2102, 2102, 2102, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  RMSE      Rsquared
##   1                   50      34.82022  0.3175575
##   1                  100      34.31440  0.3270464
##   1                  150      34.32044  0.3265703
```

```
##   2                   50      34.37653  0.3258330
##   2                  100      34.33878  0.3261820
##   2                  150      34.41707  0.3243136
##   3                   50      34.34068  0.3256270
##   3                  100      34.50665  0.3214110
##   3                  150      34.73799  0.3147820
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## RMSE was used to select the optimal model using  the smallest value.
## The final values used for the model were n.trees = 100,
##  interaction.depth = 1, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
# Plot the results
qplot(predict(modFit,testing),wage,data=testing)
```



Part 16: Model Based Prediction

```
# Example: Iris Data
data(iris); library(ggplot2)
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

```
table(iris$Species)
```

```
##
```

```
##     setosa versicolor  virginica
##         50         50         50
```

```
# Create training and test sets
inTrain <- createDataPartition(y=iris$Species, p=0.7, list=FALSE)
training <- iris[inTrain,]
testing <- iris[-inTrain,]
dim(training); dim(testing)
```

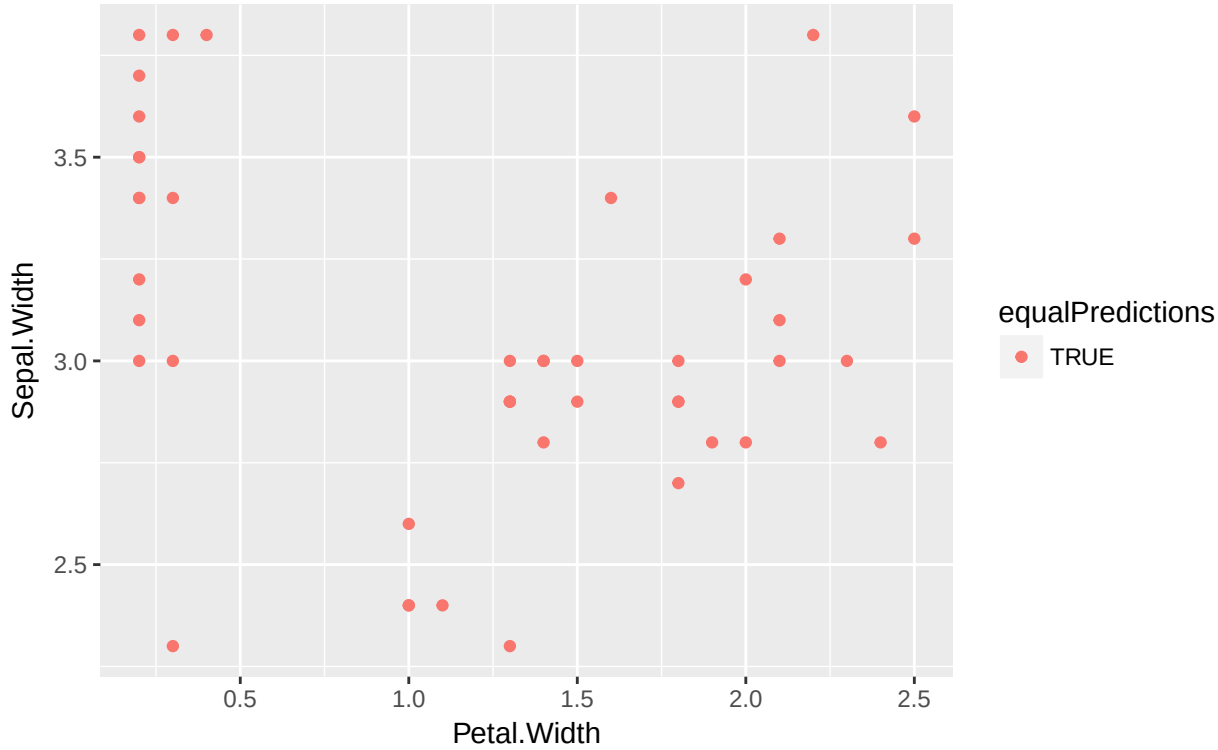
```
## [1] 105   5
```

```
## [1] 45  5
```

```
# Build predictions
modlda = train(Species ~ .,data=training,method="lda")
library(klaR)
modnb = train(Species ~ ., data=training,method="nb")
plda = predict(modlda,testing); pnb = predict(modnb,testing)
table(plda,pnb)
```

```
##             pnb
## plda         setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      0         15         0
##   virginica       0          0        15
```

```
# Comparison of results
equalPredictions = (plda==pnb)
qplot(Petal.Width,Sepal.Width,colour=equalPredictions,data=testing)
```



Part 17: Combining Predictors

```
# Example with Wage data--create training, test and validation sets
library(ISLR); data(Wage); library(ggplot2); library(caret);
Wage <- subset(Wage,select=-c(logwage))

# Create a building data set and validation set
inBuild <- createDataPartition(y=Wage$wage,p=0.7, list=FALSE)
validation <- Wage[-inBuild,]; buildData <- Wage[inBuild,]
inTrain <- createDataPartition(y=buildData$wage,p=0.7, list=FALSE)
training <- buildData[inTrain,]; testing <- buildData[-inTrain,]
dim(training)
```

```
## [1] 1474   11
```

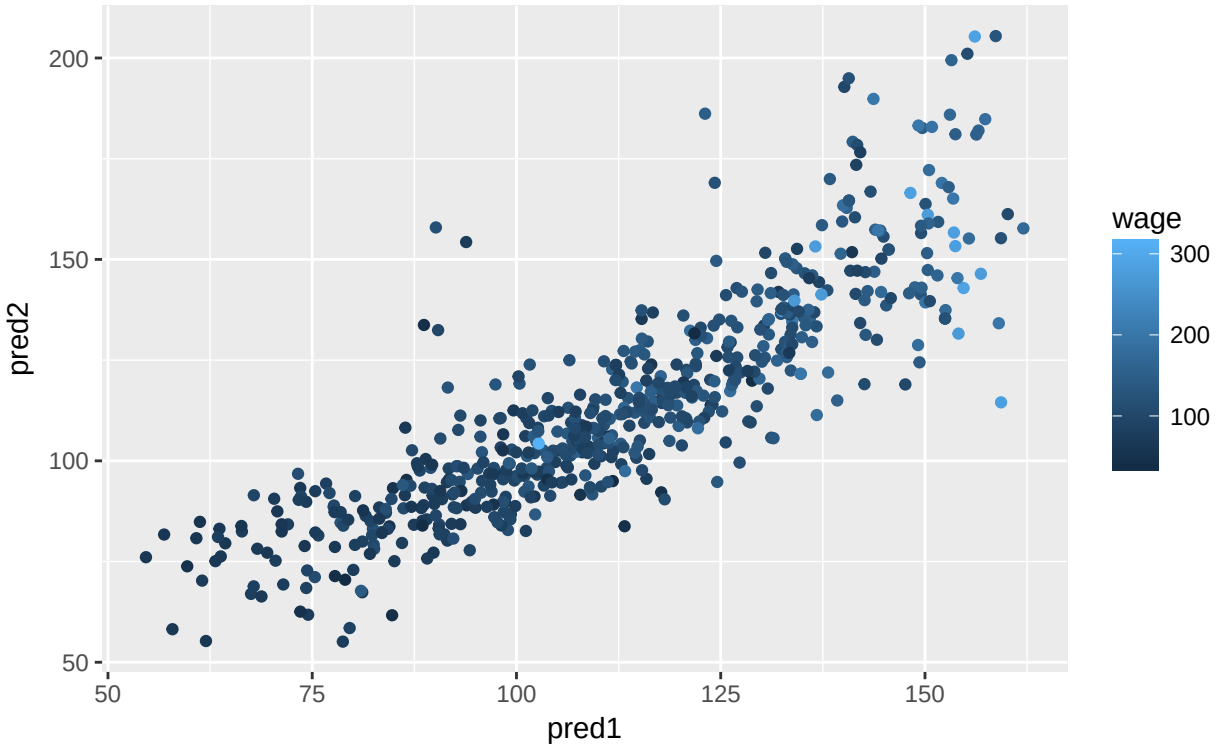
```
dim(testing)
```

```
## [1] 628  11
```

```
dim(validation)
```

```
## [1] 898  11
```

```
# Build two different models
mod1 <- train(wage ~.,method="glm",data=training)
mod2 <- train(wage ~.,method="rf", data=training,
              trControl = trainControl(method="cv"),number=3)

# Predict on the testing set
pred1 <- predict(mod1,testing); pred2 <- predict(mod2,testing)
qplot(pred1,pred2,colour=wage,data=testing)
```



```
# Fit a model that combines predictors
predDF <- data.frame(pred1,pred2,wage=testing$wage)
combModFit <- train(wage ~.,method="gam",data=predDF)
combPred <- predict(combModFit,predDF)

# Testing errors
sqrt(sum((pred1-testing$wage)^2))
```

```
## [1] 811.6074
```

```
sqrt(sum((pred2-testing$wage)^2))
```

```
## [1] 858.4664
```

```
sqrt(sum((combPred-testing$wage)^2))
```

```
## [1] 789.7957
```

```
# Predict on validation data set
pred1V <- predict(mod1,validation); pred2V <- predict(mod2,validation)
predVDF <- data.frame(pred1=pred1V,pred2=pred2V)
combPredV <- predict(combModFit,predVDF)

# Evaluate on validation
sqrt(sum((pred1V-validation$wage)^2))
```

```
## [1] 1053.328
```

```
sqrt(sum((pred2V-validation$wage)^2))
```

```
## [1] 1063.146
```

```
sqrt(sum((combPredV-validation$wage)^2))
```

```
## [1] 1036.111
```

Part 18: Unsuprevised Prediction

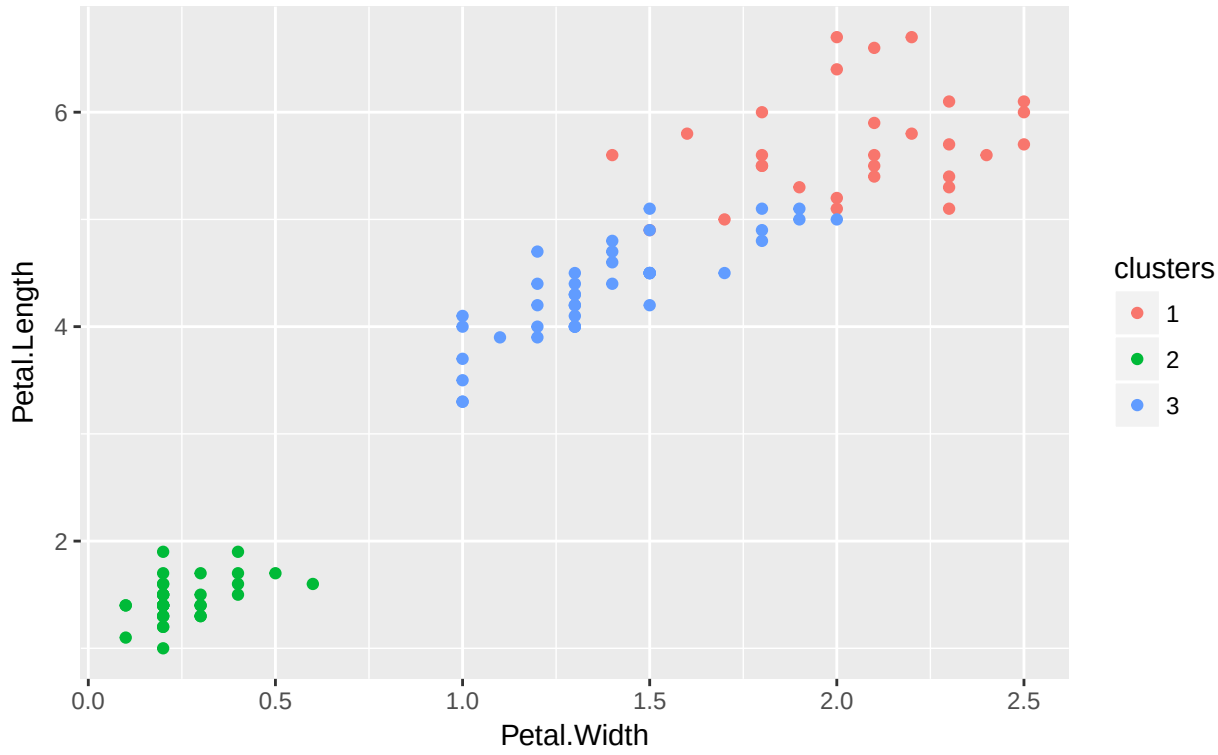
```
# Iris example ignoring species labels
data(iris); library(ggplot2)
inTrain <- createDataPartition(y=iris$Species, p=0.7, list=FALSE)
training <- iris[inTrain,]; testing <- iris[-inTrain,]
dim(training)
```

```
## [1] 105   5
```

```
dim(testing)
```

```
## [1] 45  5
```

```
# Cluster with k-means
kMeans1 <- kmeans(subset(training,select=-c(Species)),centers=3)
training$clusters <- as.factor(kMeans1$cluster)
qplot(Petal.Width,Petal.Length,colour=clusters,data=training)
```




```
# Compare to real labels
table(kMeans1$cluster,training$Species)
```

```
##
##     setosa versicolor virginica
##   1      0          2        27
##   2     35          0         0
##   3      0         33         8
```

```
# Build predictor
modFit <- train(clusters ~.,data=subset(training,select=-c(Species)),method="rpart")
table(predict(modFit,training),training$Species)
```

```
##
##     setosa versicolor virginica
##   1      0          0        25
##   2     35          0         0
##   3      0         35        10
```

```
# Apply on test
testClusterPred <- predict(modFit,testing)
table(testClusterPred, testing$Species)
```

```
##
## testClusterPred setosa versicolor virginica
##               1      0          0         9
##               2     15          0         0
##               3      0         15         6
```

Part19: Forcasting

```r
# Google data
library(quantmod)
from.dat <- as.Date("01/01/08", format="%m/%d/%y")
to.dat <- as.Date("12/31/13", format="%m/%d/%y")
getSymbols("GOOG", src="google", from = from.dat, to = to.dat)
head(GOOG)

# Summarize monthly and store as time series
mGoog <- to.monthly(GOOG)
googOpen <- Op(mGoog)
ts1 <- ts(googOpen,frequency=12)
plot(ts1,xlab="Years+1", ylab="GOOG")

# Decompose a time series into parts
# __Trend__  - Consistently increasing pattern over time
# __Seasonal__ -  When there is a pattern over a fixed period of time that recurs.
# __Cyclic__ -  When data rises and falls over non fixed periods
plot(decompose(ts1),xlab="Years+1")

# Training and test sets
ts1Train <- window(ts1,start=1,end=5)
ts1Test <- window(ts1,start=5,end=(7-0.01))
ts1Train

# Simple moving average
plot(ts1Train)
lines(ma(ts1Train,order=3),col="red")

# Exponential smoothing
ets1 <- ets(ts1Train,model="MMM")
fcast <- forecast(ets1)
plot(fcast); lines(ts1Test,col="red")

# Get the accuracy
accuracy(fcast,ts1Test)
```