Practical Machine Learning Course Project

Nick Franciose

Background

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement - a group of enthusiasts who take measurements about themselves regularly to improve their health and find patterns in their behavior. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har) (see the section on the Weight Lifting Exercise Dataset).

```
#Load relevant packages
library(AppliedPredictiveModeling)
library(caret)
```

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

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

```
library(rpart)
library(rattle)
```

```
## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 3.2.5
```

```
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(knitr)
```

```
## Warning: package 'knitr' was built under R version 3.2.5
```

```
library(RColorBrewer)
library(ggplot2)
library(plyr)
#Read files and replace zeros with NA
train <- read.csv("pml-training.csv", na.strings=c("NA","#DIV/0!", ""), header=TRUE)
test <- read.csv("pml-testing.csv", na.strings=c("NA","#DIV/0!", ""), header=TRUE)
```

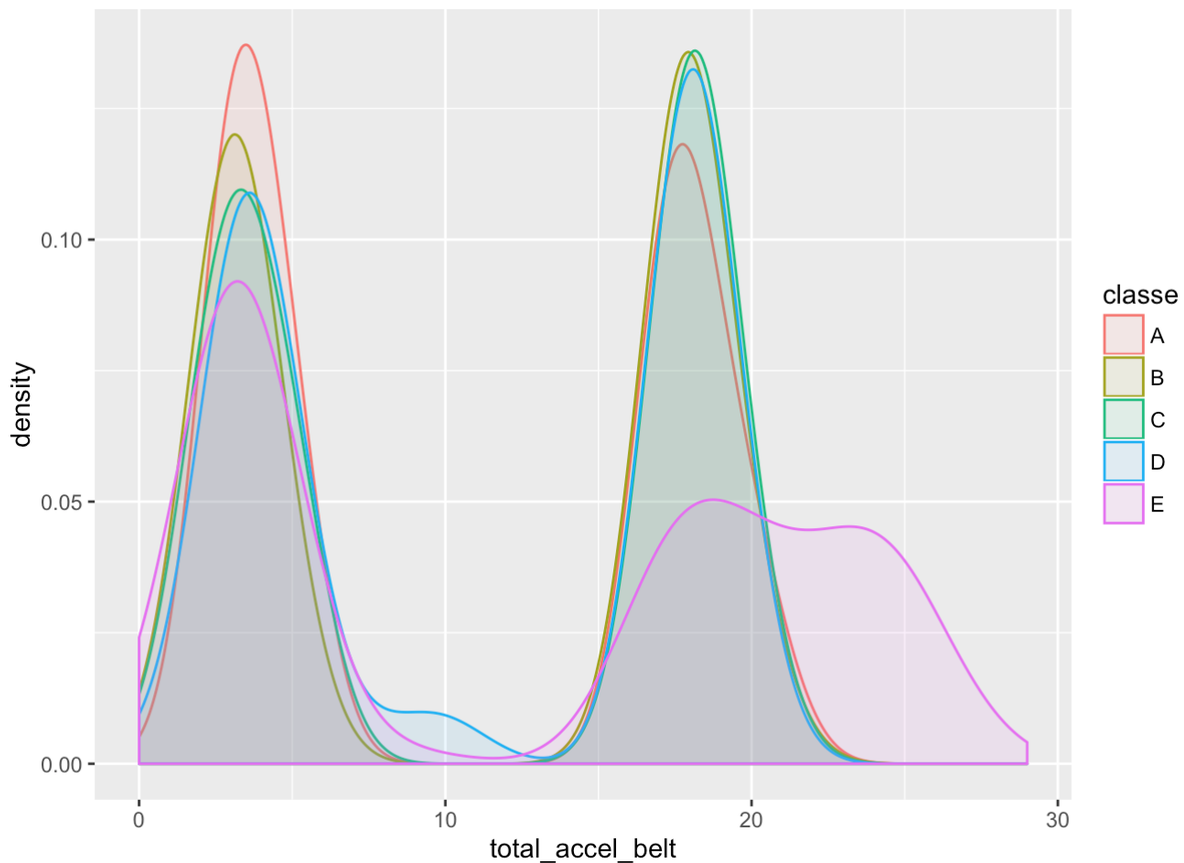
Clean the data

First we need to clean the training dataset to include only the variables we want to use as predictors (sensor data) of our outcomes (classe).

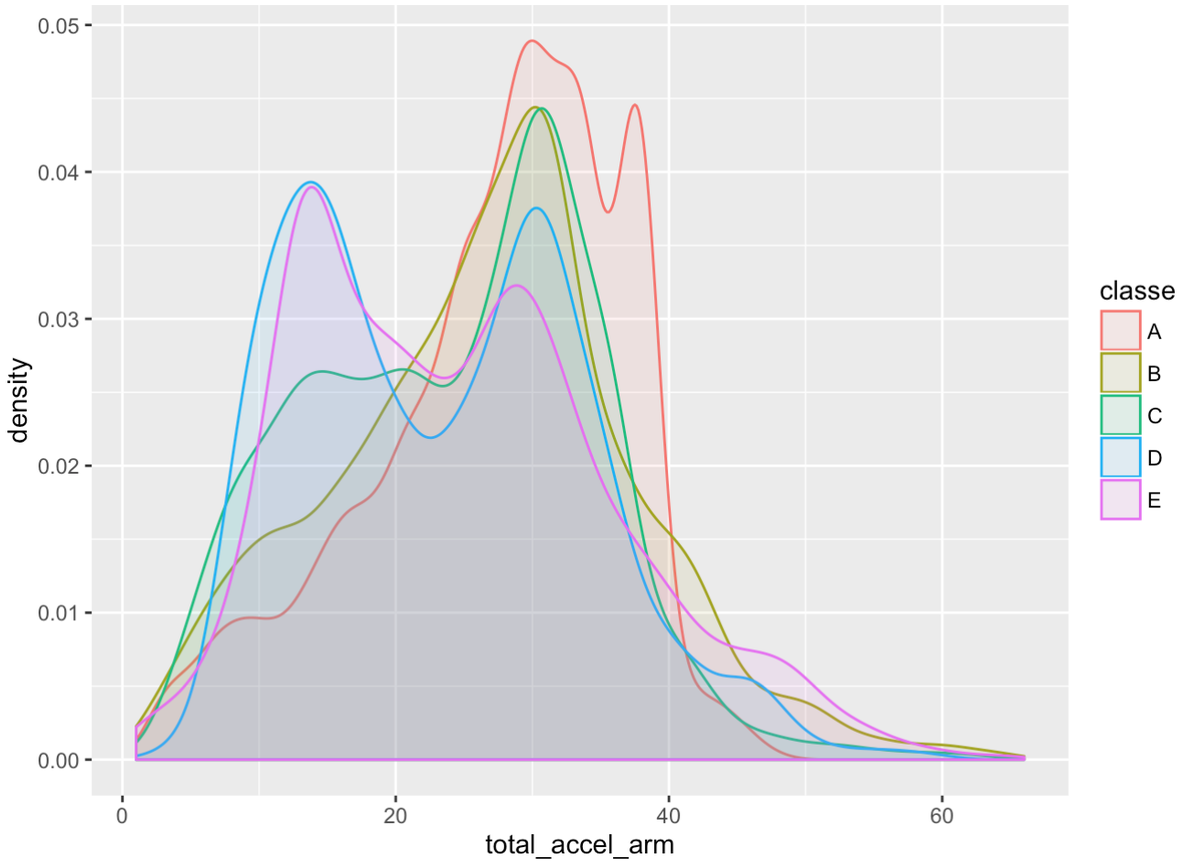
```
#Remove the first seven columns
train <- train[-c(1:7)]

#Clean variables with more than 70% NA

cleanedtrain<- train
for(i in 1:length(train)) {
    if( sum( is.na( train[, i] ) ) /nrow(train) >= .7) {
        for(j in 1:length(cleanedtrain)) {
            if( length( grep(names(train[i]), names(cleanedtrain)[j]) ) == 1)  {
                cleanedtrain <- cleanedtrain[ , -j]
            }
        }
    }
}

# Set back to the original variable name
train <- cleanedtrain
rm(cleanedtrain); rm(i); rm(j)
```

Now that we have cleaned the dataset to exclude extraneous variables, let's look at a few of the predictors we have to work with:

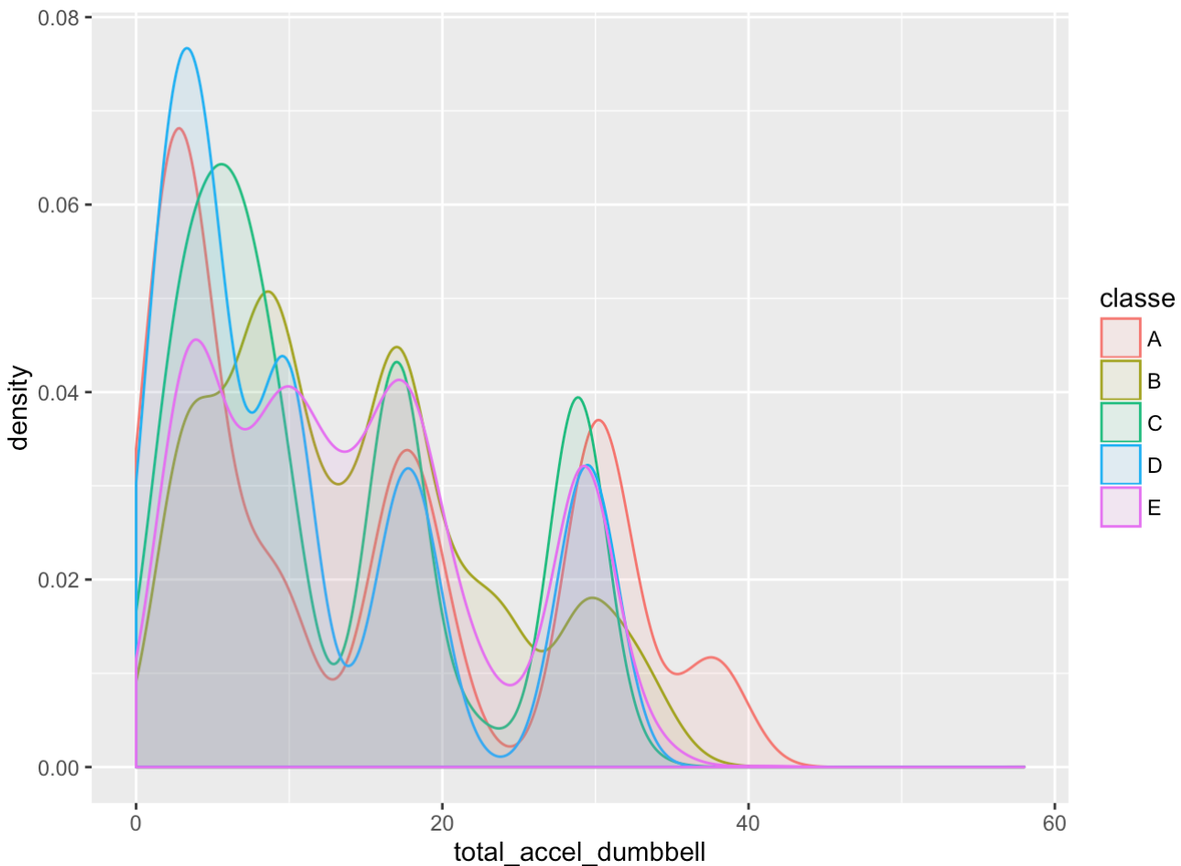
```
#Explore the Test Data
ggplot(train, aes(total_accel_belt, colour = classe, fill = classe)) +geom_density(alpha
 = 0.1)
```



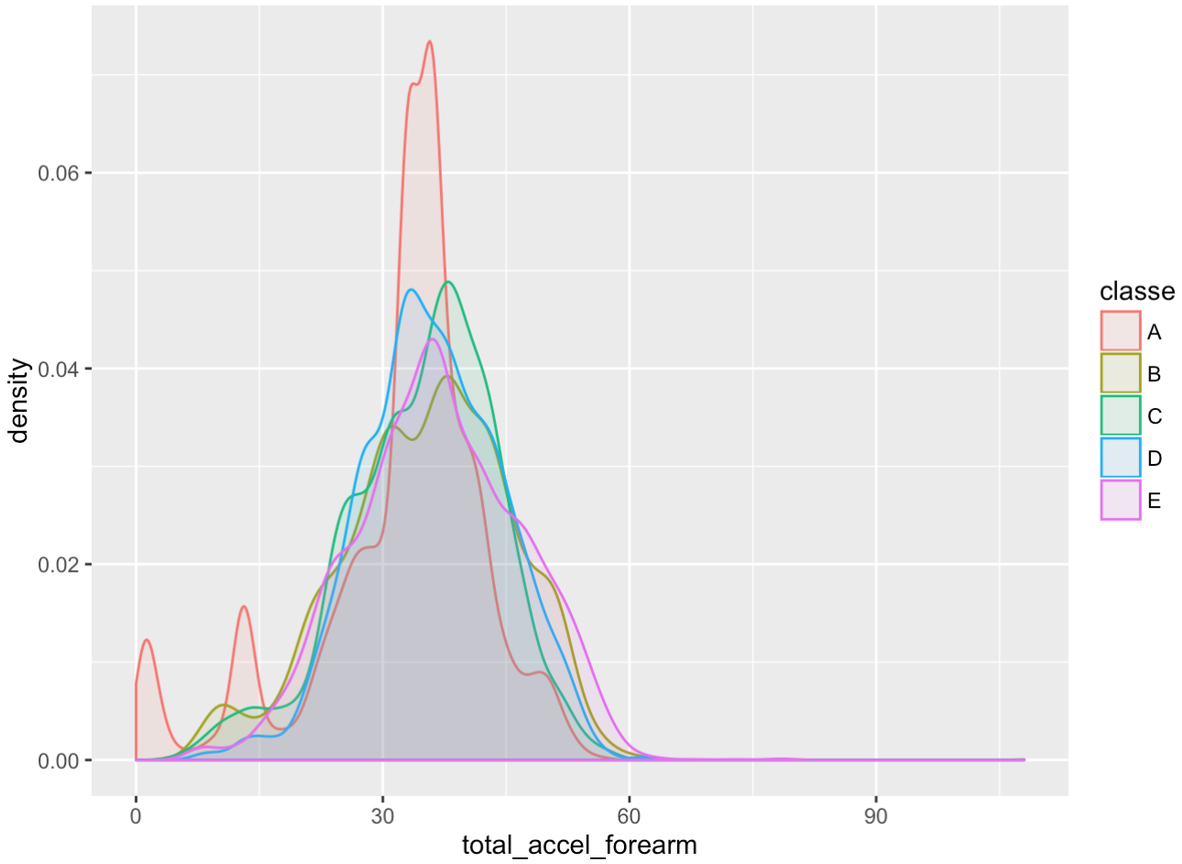
```
ggplot(train, aes(total_accel_arm, colour = classe, fill = classe)) +geom_density(alpha
= 0.1)
```




```
ggplot(train, aes(total_accel_dumbbell, colour = classe, fill = classe)) +geom_density(a
lpha = 0.1)
```




```
ggplot(train, aes(total_accel_forearm, colour = classe, fill = classe)) +geom_density(al
pha = 0.1)
```



In order to use the 52 predictor distributions to predict the classe variable, let's partition our training set into a smaller training set and new test set.

```
#Partioning the training set into two
trainPart <- createDataPartition(train$classe, p=0.6, list=FALSE)
myTrain <- train[trainPart, ]
myTest <- train[-trainPart, ]
dim(myTrain); dim(myTest)
```

```
## [1] 11776    53
```

```
## [1] 7846   53
```

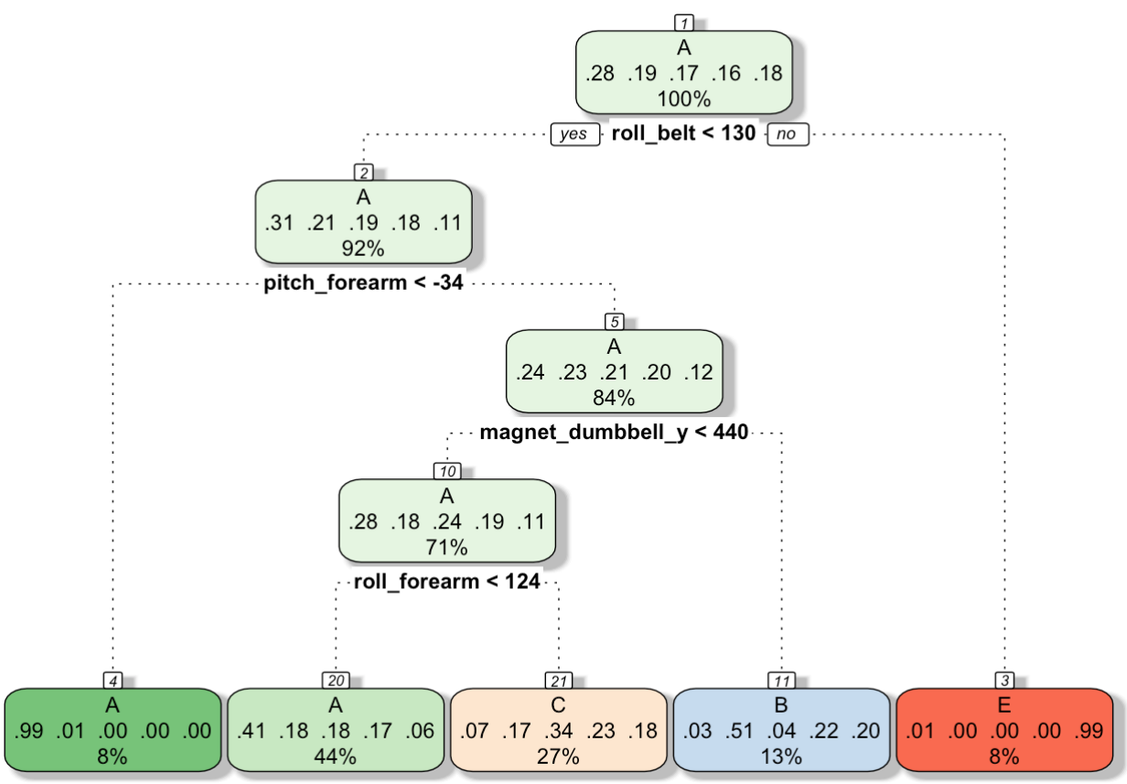
Training and Testing Predictive Models

Let's train and test a few different predictive models on the newly partitioned training and test sets. Let's start with a simple Classification Tree.

```
#Classification Tree

set.seed(345)
ctFit <- train(myTrain$classe ~ ., data = myTrain, method="rpart")
print(ctFit, digits=3)
```

```
## CART
##
## 11776 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 11776, 11776, 11776, 11776, 11776, 11776, ...
## Resampling results across tuning parameters:
##
##   cp      Accuracy  Kappa   Accuracy SD  Kappa SD
##   0.0388  0.509     0.3627  0.0203       0.0338
##   0.0612  0.402     0.1854  0.0594       0.0987
##   0.1149  0.332     0.0741  0.0421       0.0618
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.0388.
```

```
fancyRpartPlot(ctFit$finalModel)
```



Rattle 2016-Oct-22 11:21:20 nickfranciose

Now let's test that classification tree model on our test set.

```
#Apply Classification tree to myTest
ctPredictions <- predict(ctFit, newdata=myTest)
print(confusionMatrix(ctPredictions, myTest$classe), digits=4)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2023  632  655  591  203
##          B   37  514   45  234  187
##          C  168  372  668  461  399
##          D    0    0    0    0    0
##          E    4    0    0    0  653
##
## Overall Statistics
##
##                Accuracy : 0.4917
##                  95% CI : (0.4806, 0.5028)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3354
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9064  0.33860  0.48830   0.0000  0.45284
## Specificity            0.6293  0.92051  0.78388   1.0000  0.99938
## Pos Pred Value         0.4929  0.50541  0.32302      NaN  0.99391
## Neg Pred Value         0.9441  0.85298  0.87885   0.8361  0.89025
## Prevalence             0.2845  0.19347  0.17436   0.1639  0.18379
## Detection Rate         0.2578  0.06551  0.08514   0.0000  0.08323
## Detection Prevalence   0.5231  0.12962  0.26357   0.0000  0.08374
## Balanced Accuracy      0.7678  0.62956  0.63609   0.5000  0.72611
```

The accuracy of this model is fairly low. Let's see if we can improve on this by boosting with trees.

```
#Create a prediction model using boosting with Trees

#TRAIN
gbmFit <- train(classe ~ ., data = train, method = "gbm", verbose = FALSE)
```

```
## Loading required package: gbm
```

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

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.1
```

```
#Test Boosting with Trees model
gbmPrediction <- predict(gbmFit, myTest)
confusionMatrix(gbmPrediction, myTest$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2206   29    0    1    2
##          B   17 1457   33    3    9
##          C    6   32 1320   36    7
##          D    2    0   14 1239   18
##          E    1    0    1    7 1406
##
## Overall Statistics
##
##                Accuracy : 0.9722
##                  95% CI : (0.9683, 0.9757)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9649
##  Mcnemar's Test P-Value : 1.208e-05
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9884   0.9598   0.9649   0.9635   0.9750
## Specificity            0.9943   0.9902   0.9875   0.9948   0.9986
## Pos Pred Value         0.9857   0.9592   0.9422   0.9733   0.9936
## Neg Pred Value         0.9954   0.9904   0.9926   0.9928   0.9944
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2812   0.1857   0.1682   0.1579   0.1792
## Detection Prevalence   0.2852   0.1936   0.1786   0.1622   0.1803
## Balanced Accuracy      0.9913   0.9750   0.9762   0.9791   0.9868
```

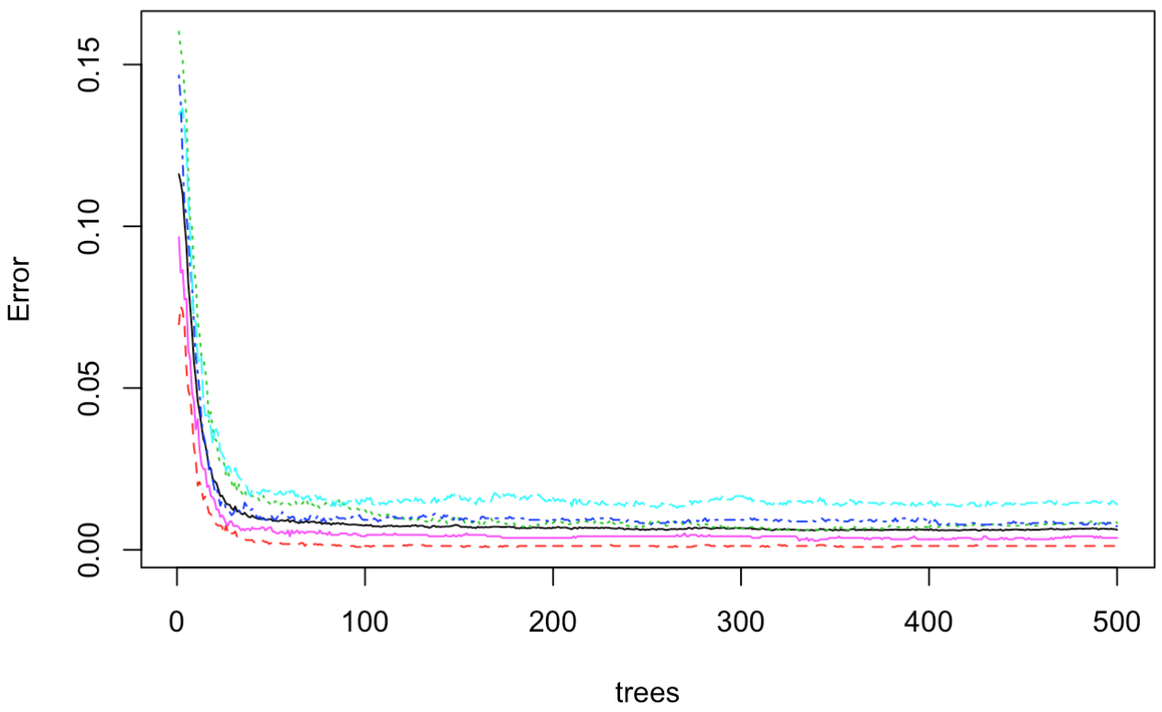
This model performed much better than a simple classification model, with accuracy of over 97%. Let's see if a random forest model can deliver improved performance still.

```
#Create Random Forest model
set.seed(1500)
random_forest=randomForest(classe~.,data=myTrain,ntree=500,importance=TRUE)
random_forest
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = myTrain, ntree = 500,      importance = TR
UE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.62%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3344    3    0    0    1 0.001194743
## B   14 2260    5    0    0 0.008336990
## C    0   11 2039    4    0 0.007302824
## D    0    0   26 1903    1 0.013989637
## E    0    0    2    6 2157 0.003695150
```

It's important to evaluate the error rate of our model. We can see that by increasing the number of trees in our model, we decrease our error rate. In the case of random forest models, we don't need to guard against overfitting with cross validation. There is no need for cross validation because our "out of bag" prediction (represented by the black line on the error rate plot) is an internally calculated, unbiased estimate of the test set error.

```
plot(random_forest,main="Error Rate vs Number of Trees")
```



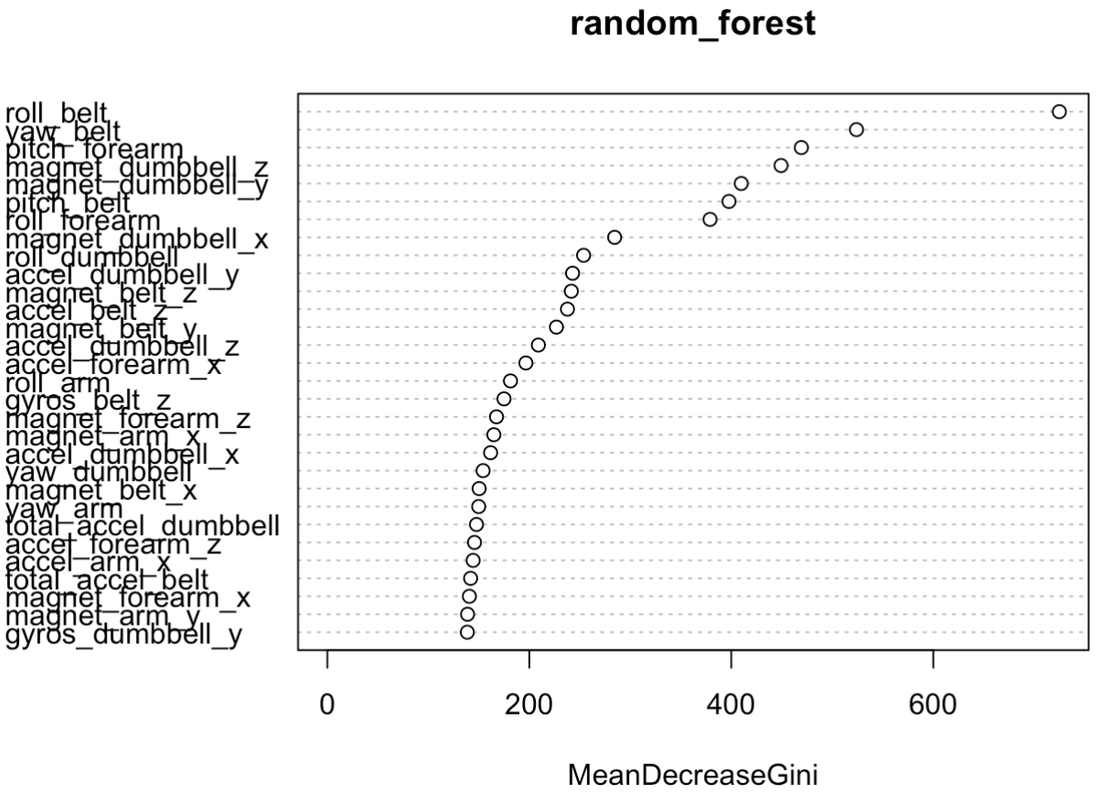
Now let's apply this random forest model to our test set to evaluate performance.

```
rfPredictions = predict(random_forest, newdata=myTest)
confusionMatrix(rfPredictions,myTest$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2225    8    0    0    0
##          B    7 1509   12    0    0
##          C    0    1 1356   18    0
##          D    0    0    0 1268    6
##          E    0    0    0    0 1436
##
## Overall Statistics
##
##                Accuracy : 0.9934
##                  95% CI : (0.9913, 0.995)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9916
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9969   0.9941   0.9912   0.9860   0.9958
## Specificity            0.9986   0.9970   0.9971   0.9991   1.0000
## Pos Pred Value         0.9964   0.9876   0.9862   0.9953   1.0000
## Neg Pred Value         0.9988   0.9986   0.9981   0.9973   0.9991
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2836   0.1923   0.1728   0.1616   0.1830
## Detection Prevalence   0.2846   0.1947   0.1752   0.1624   0.1830
## Balanced Accuracy      0.9977   0.9955   0.9941   0.9925   0.9979
```

This model is better than the boosted trees model, with an error rate less than one percent. We can inspect which variables are most significant to the model:

```
#Explore Variable importance to Random Forest model
varImpPlot(random_forest,type=2)
```



We can see that the five variables containing the most predictive signal are roll belt, yaw belt, pitch forearm, magnet dumbell z, and pitch belt. Distributions of these variables can be viewed in the appendix.

Final Prediction with Random Forest

The random forest is the most accurate model we have used, so we will apply this model to generate our final prediction of our test set.

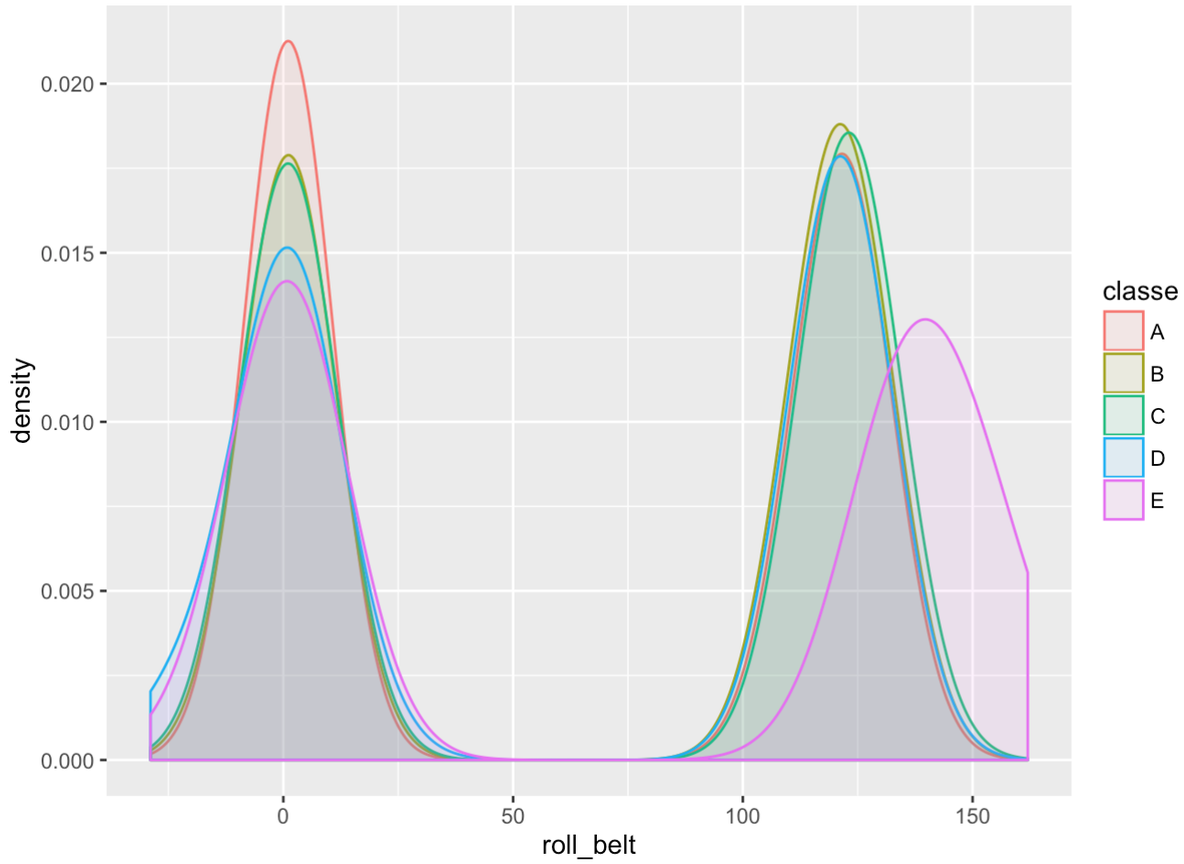
```
#Remove all unneeded columns
test <-test[,c(8:11,37:49,60:68,84:86,102,113:124,140,151:160)]
print(predict(random_forest, newdata=test))
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  B  A  B  A  A  E  D  B  A  A  B  C  B  A  E  E  A  B  B  B
## Levels: A B C D E
```

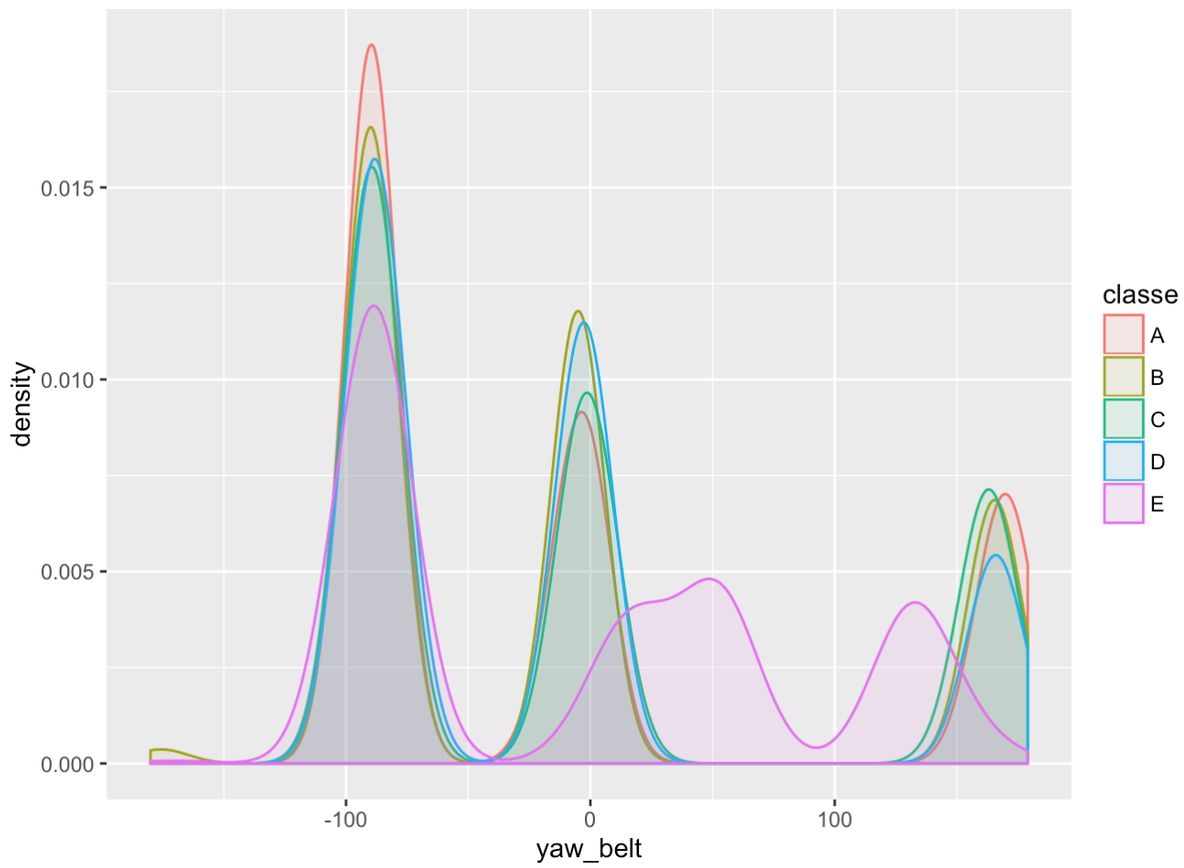
Appendix

Distributions of the five most predictive variables in our random forest model can be seen below.

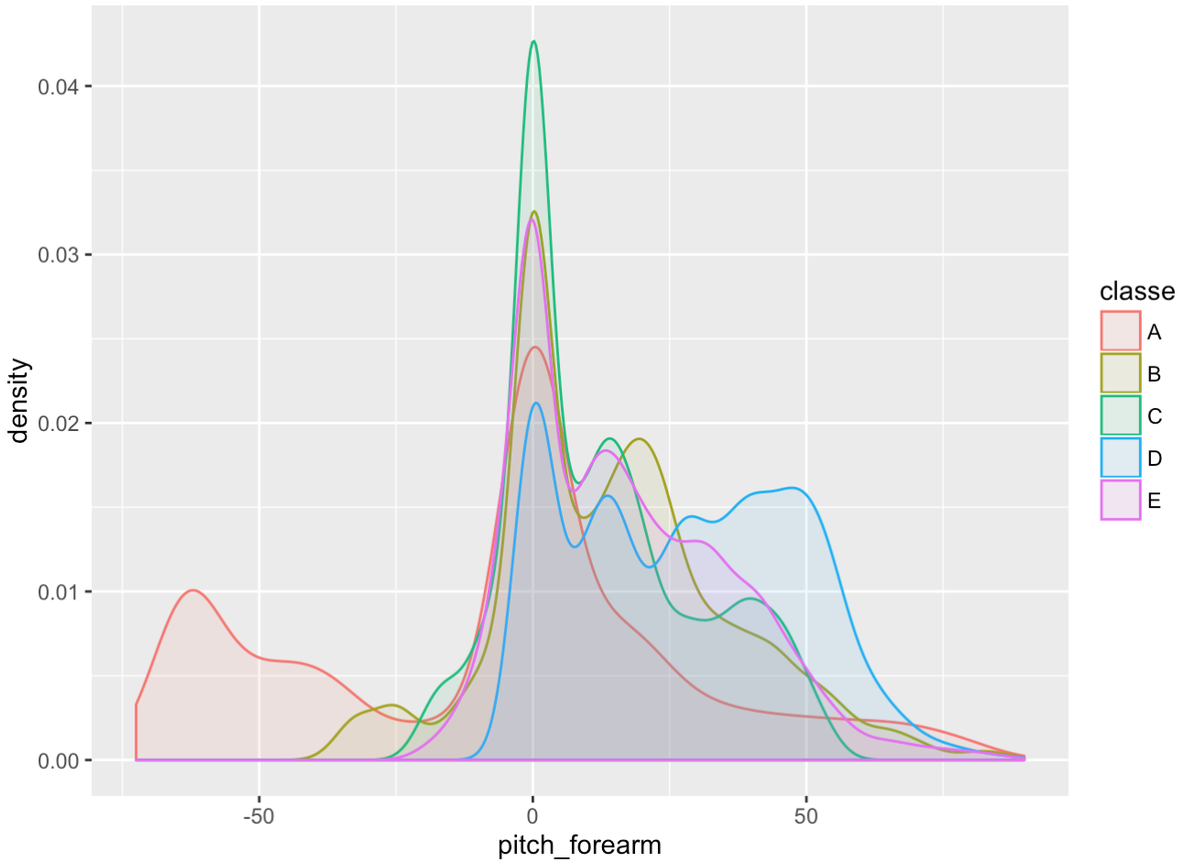
```
ggplot(train, aes(roll_belt, colour = classe, fill = classe)) +geom_density(alpha = 0.1)
```



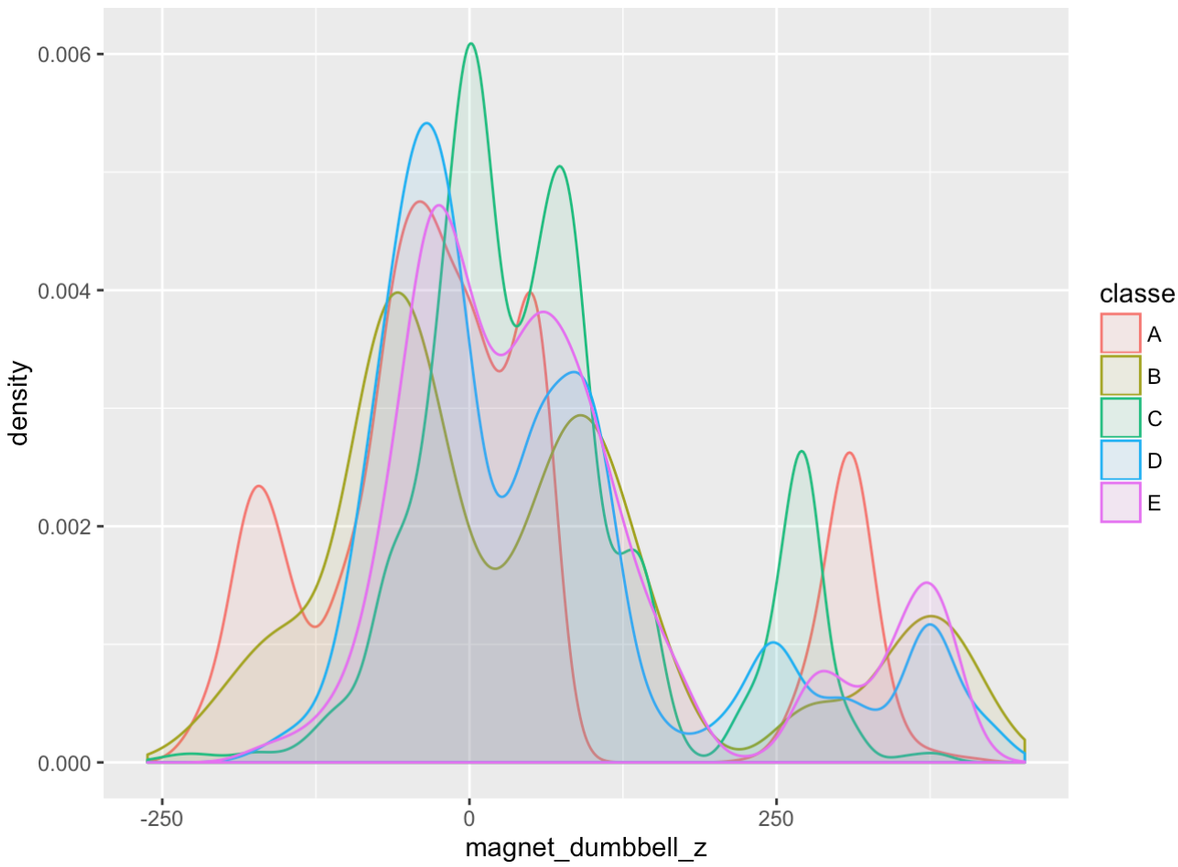

```
ggplot(train, aes(yaw_belt, colour = classe, fill = classe)) +geom_density(alpha = 0.1)
```



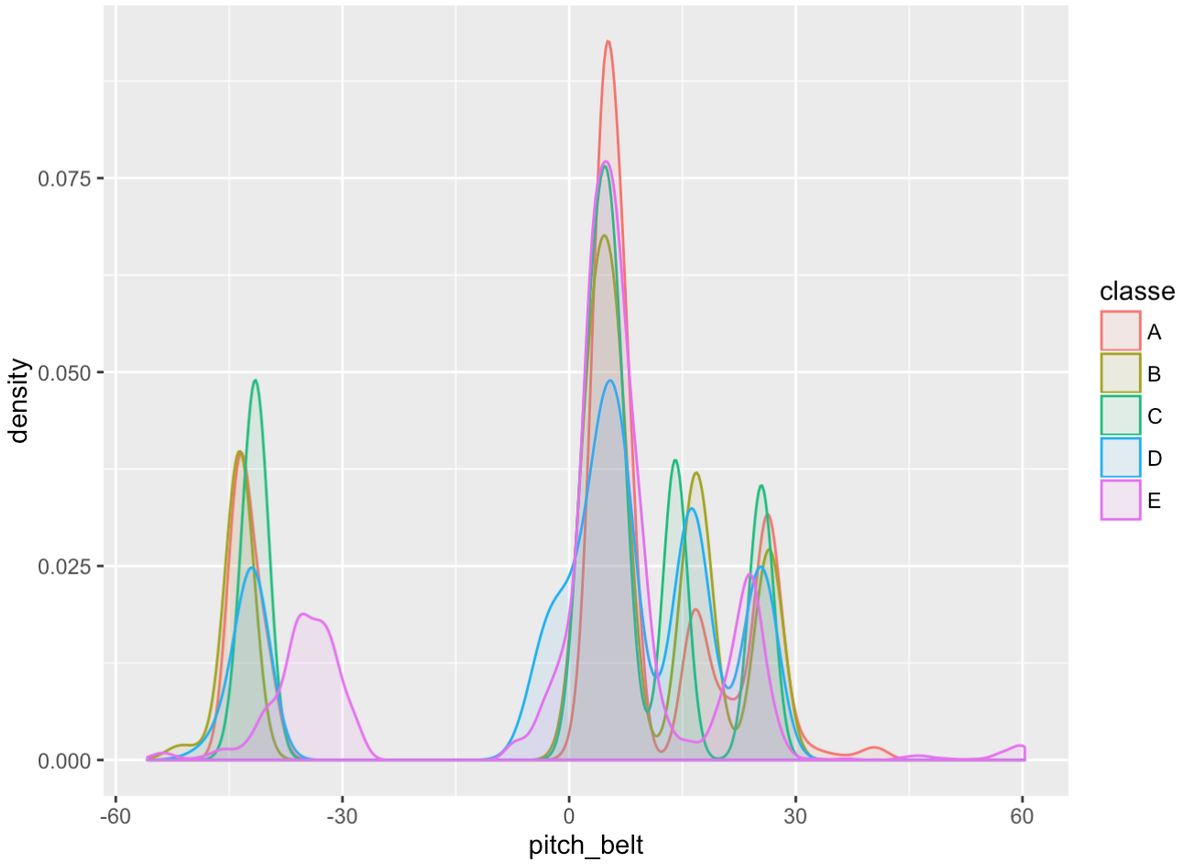

```
ggplot(train, aes(pitch_forearm, colour = classe, fill = classe)) +geom_density(alpha = 0.
```




```
ggplot(train, aes(magnet_dumbbell_z, colour = classe, fill = classe)) +geom_density(alph
a = 0.1)
```




```
ggplot(train, aes(pitch_belt, colour = classe, fill = classe)) +geom_density(alpha =
0.1)
```


density
0.075
0.050
0.025
0.000
-60
-30
0
30
60
pitch_belt
classe
A
B
C
D
E