This project aims to examine the accelerometer data of six individuals who wore sensors on different body parts while performing both proper and improper barbell lifts.
To begin, we load the necessary data and libraries for running the model. We then proceed to clean up the data by removing any irrelevant or zero variance variables. Afterward, we divide the cleaned-up data into a training set and a testing set while keeping the original testing set, "pmltest", untouched for later use. Our next step involves creating and testing the model to obtain the most optimal outcome. To do this, we implement Decision Tree, Gradient Boosted Trees, and Random Forest, using k-folds cross-validation on the training set. Subsequently, we randomly select a test set from the training set to predict the accuracy and out-of-sample error rate. Based on the results, we choose the best model to predict the test case.

#LOADING THE LIBRARIES

```
library(caret)
library(ggplot2)
library(rattle)
library(corrplot)
```

#LOADING ALL THE DATA

```
pmltrain <- read.csv("pml-training.csv")
pmltest <- read.csv("pml-testing.csv")
set.seed(9)
dim(pmltrain)
dim(pmltest)
```

```
> dim(pmltrain)
[1] 19622   160
> dim(pmltest)
[1]  20 160
```

#CLEANING THE DATA

##Removing N/A variables

```
pmltrain <- pmltrain[, sapply(pmltrain, function(col) sum(is.na(col))/length(col) < 0.9)]
pmltrain <- pmltrain[,-c(1:7)]
```

##Removing near zero variance variables

```
nzvv <- nearZeroVar(pmltrain)
pmltrain <- pmltrain[,-nzvv]
dim(pmltrain)
```

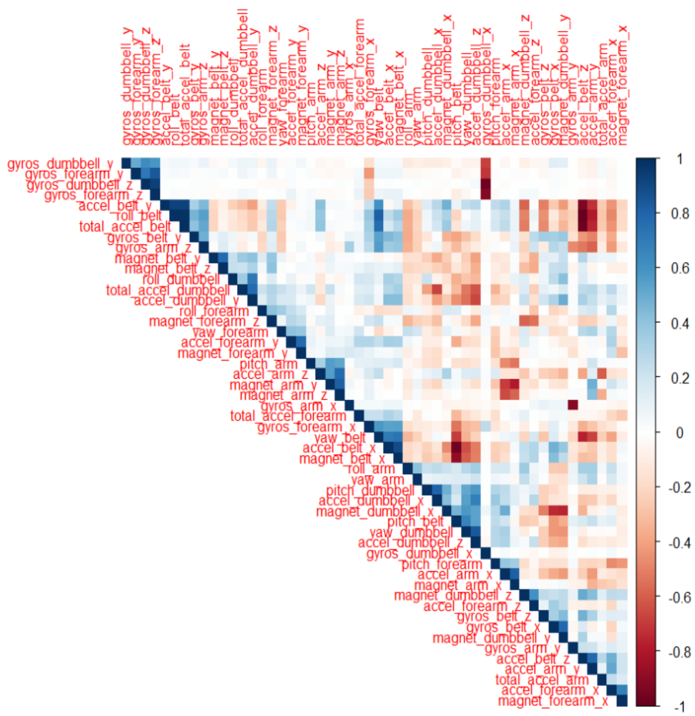
```
dim(pmltrain)
[1] 19622    53
```

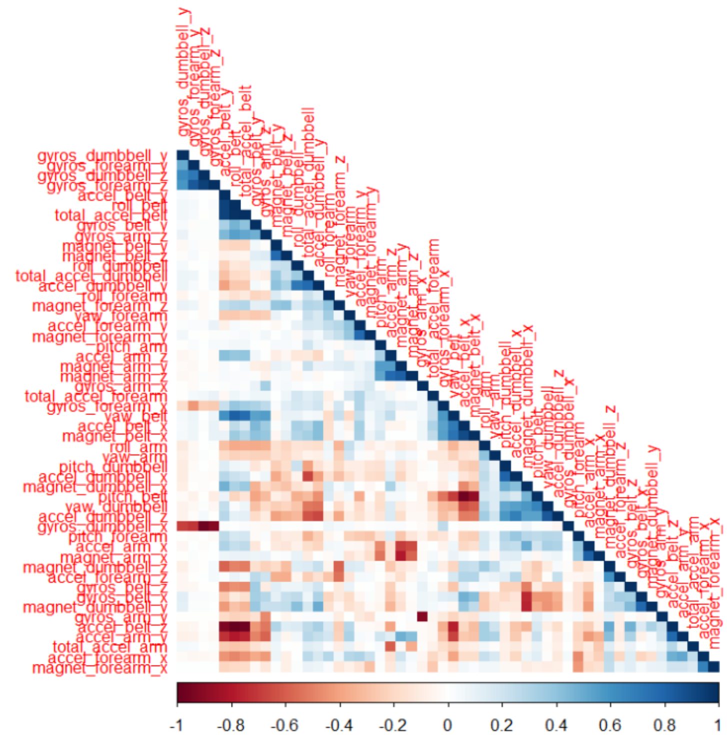
Calculate the correlation matrix

```
cor_matrix <- cor(pmltrain[, -ncol(pmltrain)])
```

Create a correlation plot

```
corrplot(cor_matrix, method = "color", type = "upper", order = "hclust", tl.cex = 0.8)
corrplot(cor_matrix, method = "color", type = "lower", order = "hclust", tl.cex = 0.8)
```

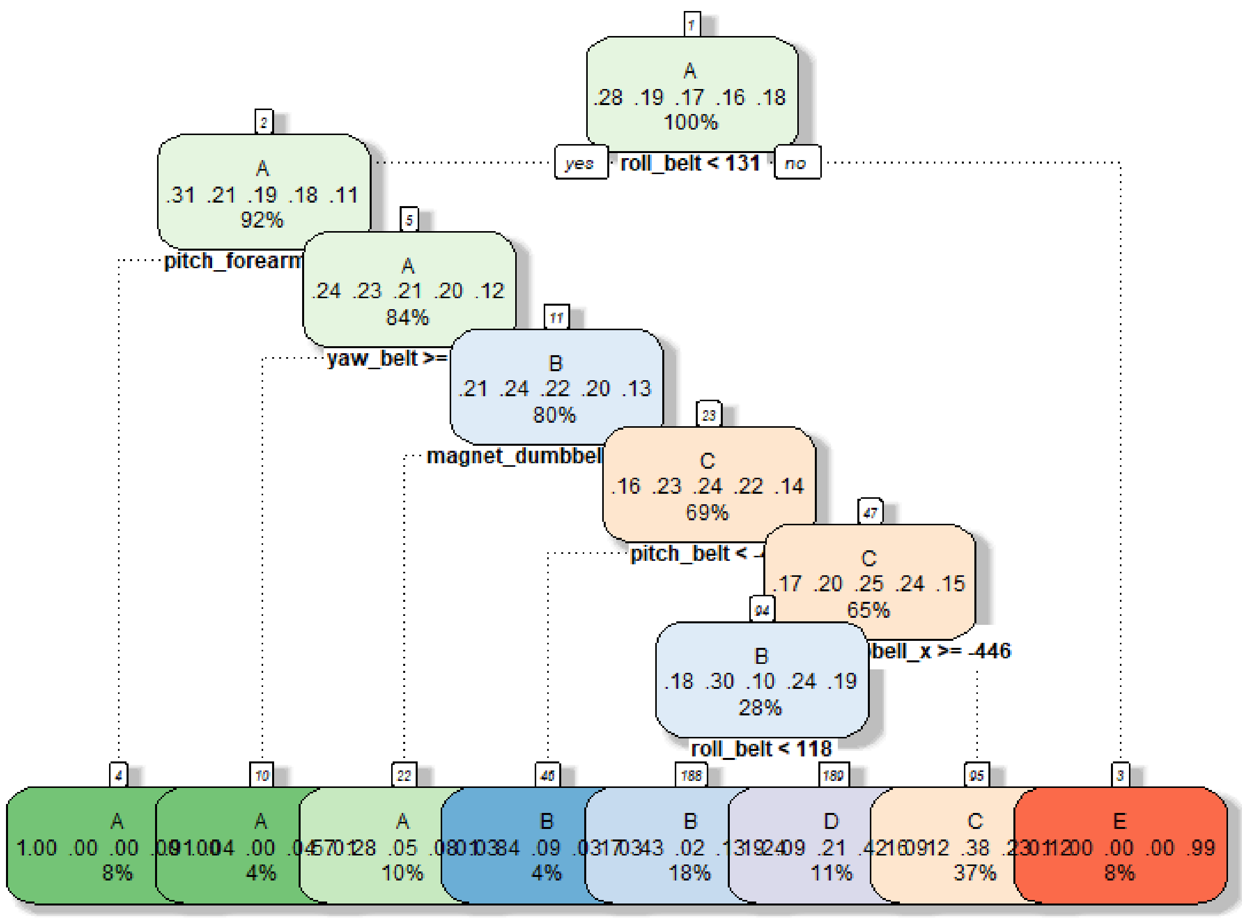

gyros_dumbbell_y
gyros_forearm_y
gyros_dumbbell_z
gyros_forearm_z
accel_belt_y
roll_belt
total_accel_belt
gyros_belt_y
gyros_arm_z
magnet_belt_y
magnet_belt_z
roll_dumbbell
total_accel_dumbbell
accel_dumbbell_y
roll_forearm
magnet_forearm_z
yaw_forearm
accel_forearm_y
magnet_forearm_y
pitch_arm
accel_arm_z
magnet_arm_y
magnet_arm_z
gyros_arm_x
total_accel_forearm
gyros_forearm_x
yaw_belt
accel_belt_x
magnet_belt_x
roll_arm
yaw_arm
pitch_dumbbell
accel_dumbbell_x
magnet_dumbbell_x
pitch_belt
yaw_dumbbell
accel_dumbbell_z
gyros_dumbbell_x
pitch_forearm
accel_arm_x
magnet_arm_x
magnet_dumbbell_z
accel_forearm_z
gyros_belt_z
gyros_belt_x
magnet_dumbbell_y
gyros_arm_y
accel_belt_z
accel_arm_y
total_accel_arm
accel_forearm_x
magnet_forearm_x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1


gyros_dumbbell_y
gyros_forearm_y
gyros_dumbbell_z
gyros_forearm_z
accel_belt_y
roll_belt
total_accel_belt
gyros_belt_y
gyros_arm_z
magnet_belt_y
magnet_belt_z
roll_dumbbell
total_accel_dumbbell
accel_dumbbell_y
roll_forearm
magnet_forearm_z
yaw_forearm
accel_forearm_y
magnet_forearm_y
pitch_arm
accel_arm_z
magnet_arm_y
magnet_arm_z
gyros_arm_x
total_accel_forearm
gyros_forearm_x
yaw_belt
accel_belt_x
magnet_belt_x
roll_arm
yaw_arm
pitch_dumbbell
accel_dumbbell_x
magnet_dumbbell_x
pitch_belt
yaw_dumbbell
accel_dumbbell_z
gyros_dumbbell_x
pitch_forearm
accel_arm_x
magnet_arm_x
magnet_dumbbell_z
accel_forearm_z
gyros_belt_z
gyros_belt_x
magnet_dumbbell_y
gyros_arm_y
accel_belt_z
accel_arm_y
total_accel_arm
accel_forearm_x
magnet_forearm_x
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

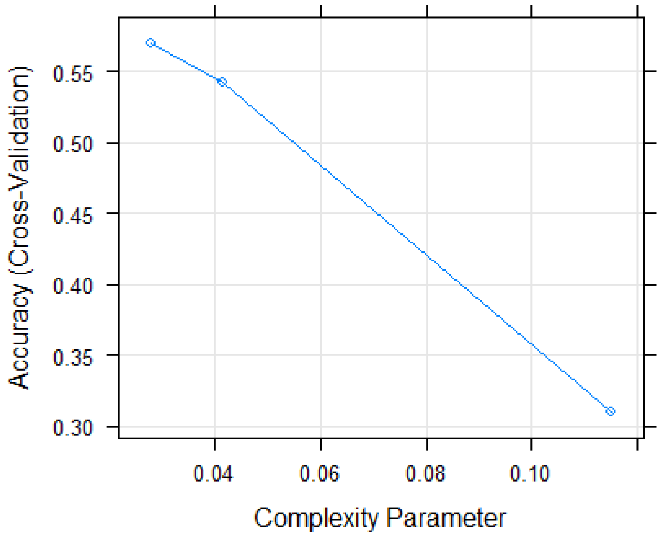
```
##split the training set into a training and testing set.
inTrain <- createDataPartition(y=pmltrain$classe, p=0.70, list=F)
training <- pmltrain[inTrain,]
testing <- pmltrain[-inTrain,]
```

```
#CREATING AND TESTING THE MODELS
control <- trainControl(method="cv", number=3,verboseIter=F)
```

```
#MODELS
##1. Recursive Partitioning and Regression Trees
fit.tree <- train(classe~., data=training, method="rpart", trControl = contro
l)
par(mar = c(1, 1, 1, 1))
options(repr.plot.width = 10, repr.plot.height = 8)
fancyRpartPlot(fit.tree$finalModel, cex = 0.6)
```




```
#Plotting the model
plot(fit.tree)
```




```
#Prediction
predict.tree <- predict(fit.tree, testing)
CMtree <- confusionMatrix(predict.tree, factor(testing$classe))
CMtree
```

```
Confusion Matrix and Statistics

          Reference
Prediction    A    B    C    D    E
         A 1035  188   28   57   13
         B  168  654   45  134  269
         C  372  255  813  527  245
         D   97   42  140  246   67
         E    2    0    0    0  488

Overall Statistics

               Accuracy : 0.5499
                 95% CI : (0.5371, 0.5626)
    No Information Rate : 0.2845
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4353

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: A Class: B Class: C Class: D Class: E
Sensitivity            0.6183   0.5742   0.7924   0.2552  0.45102
Specificity            0.9321   0.8702   0.7121   0.9297  0.99958
Pos Pred Value         0.7835   0.5150   0.3675   0.4155  0.99592
Neg Pred Value         0.8600   0.8949   0.9420   0.8643  0.88990
Prevalence             0.2845   0.1935   0.1743   0.1638  0.18386
Detection Rate         0.1759   0.1111   0.1381   0.0418  0.08292
Detection Prevalence   0.2245   0.2158   0.3759   0.1006  0.08326
Balanced Accuracy      0.7752   0.7222   0.7522   0.5924  0.72530
```

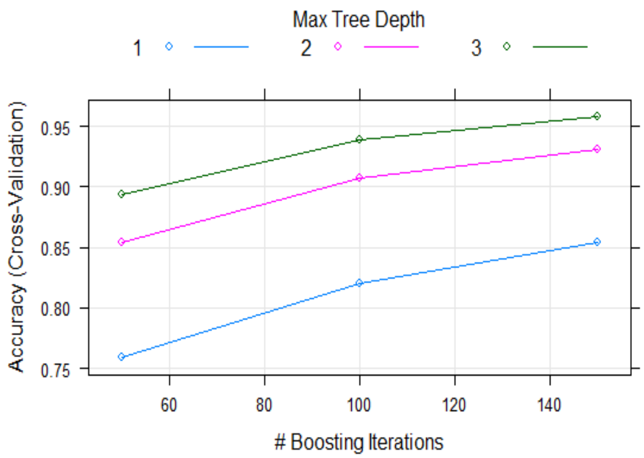
The model's accuracy score of 0.5499 is considered low and may not be suitable for adoption.

##2. Stochastic gradient boosting trees

```
fit.gbm <- train(classe~., data=training, method="gbm", trControl = control)
```

#Plotting the model

```
plot(fit.gbm)
```



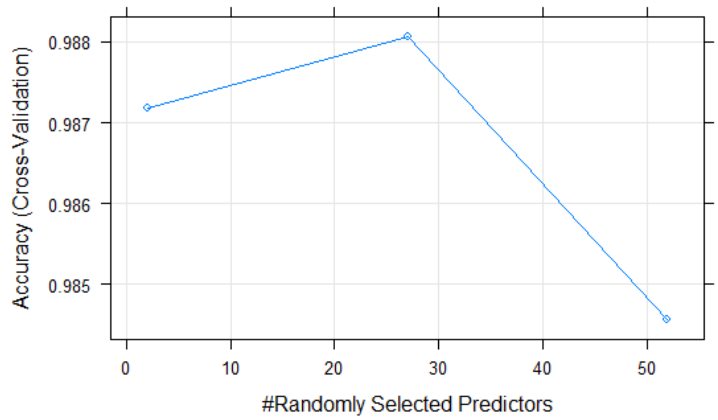
#Prediction

```
predict.gbm <- predict(fit.gbm, testing)
CMgbm <- confusionMatrix(predict.gbm, factor(testing$classe))
CMgbm
```

```
Confusion Matrix and Statistics

          Reference
Prediction    A    B    C    D    E
         A 1632   27    0    0    3
         B   23 1089   40    3   16
         C   10   21  980   39   10
         D    8    2    5  913   17
         E    1    0    1    9 1036

Overall Statistics

               Accuracy : 0.9601
                 95% CI : (0.9547, 0.9649)
    No Information Rate : 0.2845
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9495

 Mcnemar's Test P-Value : 1.524e-12

Statistics by Class:

                     Class: A Class: B Class: C Class: D Class: E
Sensitivity            0.9749   0.9561   0.9552   0.9471   0.9575
Specificity            0.9929   0.9827   0.9835   0.9935   0.9977
Pos Pred Value         0.9819   0.9300   0.9245   0.9661   0.9895
Neg Pred Value         0.9901   0.9894   0.9905   0.9897   0.9905
Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
Detection Rate         0.2773   0.1850   0.1665   0.1551   0.1760
Detection Prevalence   0.2824   0.1990   0.1801   0.1606   0.1779
Balanced Accuracy      0.9839   0.9694   0.9694   0.9703   0.9776
```

We have observed that the stochastic gradient boosting trees model has outperformed the tree model with an accuracy of 0.9601. However, we will further evaluate another model before drawing any conclusions.

```
## 3. Random Forest
fit.rf <- train(classe~., data=training, method="rf", trControl = control)
#Plotting the model
plot(fit.rf)
```




```
#Prediction
predict.rf <- predict(fit.rf, testing)
CMrf <- confusionMatrix(predict.rf, factor(testing$classe))
CMrf
```

```
Confusion Matrix and Statistics

          Reference
Prediction    A    B    C    D    E
         A 1669   14    0    0    0
         B    3 1124    7    0    0
         C    2    1 1017   14    3
         D    0    0    2  950    1
         E    0    0    0    0 1078

Overall Statistics

               Accuracy : 0.992
                 95% CI : (0.9894, 0.9941)
    No Information Rate : 0.2845
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9899

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: A Class: B Class: C Class: D Class: E
Sensitivity            0.9970   0.9868   0.9912   0.9855   0.9963
Specificity            0.9967   0.9979   0.9959   0.9994   1.0000
Pos Pred Value         0.9917   0.9912   0.9807   0.9969   1.0000
Neg Pred Value         0.9988   0.9968   0.9981   0.9972   0.9992
Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
Detection Rate         0.2836   0.1910   0.1728   0.1614   0.1832
Detection Prevalence   0.2860   0.1927   0.1762   0.1619   0.1832
```

```
Balanced Accuracy      0.9968   0.9924   0.9936   0.9924   0.9982
```

The Random Forest model displays an impressive accuracy of 0.992, surpassing the performance of the previous two models. This level of efficiency is sufficient for conducting further analysis.

##RESULTS

```
# Summarize the results for each model
modelnames <- c("Decision Tree", "Gradient Boosting", "Random Forest")
confusion_matrices <- list(CMtree, CMgbm, CMrf)

# Display confusion matrices and accuracy for each model
for (i in seq_along(model_names)) {
  cat("Model:", model_names[i], "\n")
   cat("Accuracy:", confusion_matrices[[i]]$overall["Accuracy"], "\n")
  cat("Out of Sample error:", 1-confusion_matrices[[i]]$overall["Accuracy"],
"\n")
  cat("\n")
}
```

```
Model: Decision Tree
Accuracy: 0.5498726
Out of Sample error: 0.4501274

Model: Gradient Boosting
Accuracy: 0.960068
Out of Sample error: 0.03993203

Model: Random Forest
Accuracy: 0.9920136
Out of Sample error: 0.007986406
```

We have determined that the Random Forest model is the most effective, with an accuracy of 0.992 and a mere 0.007 out of sample error rate. As a result, we will be implementing this model for our test sets.

#PREDICTIONS ON TEST SET

```
prediction <- predict(fit.rf, pmltest)
print(prediction)
```

```
[1] 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
```