Data Mining Project

Xuanyi He

Dec 8th, 2016

1 Introduction

There is an extremely popular game called Pokemon GO. I can feel how attractive it is by connecting the digital world to our real life. Everyone can play it anywhere while doing anything. As a consequence, brings my passion to explore deeper into this game. This project will make an analysis Pokemon GO by using classification and clustering technologies. I collected the Pokemon GO dataset on Kaggle, a platform for predictive modeling and data analysis on researchers complete data miners to find the best models together.

2 Executive Summary

To pre-process the dataset, I'm going to introduce the dataset first. There are 800 pokemons and 13 variables including their number, name, first and second type, and basic statistics such as hit points, the base modifier for normal attacks (attack), the base damage resistance against normal attacks (defense), special attack, special defense, speed, total attributes, and generations. I divide the pokemons into three groups, weak, normal, and powerful, based on total attributes and hit points. In other word, there are 121 weak Pokemons, 432 normal Pokemons, and 247 strong Pokemons based on total attribute level, and 221 weak Pokemons, 353 normal Pokemons, and 236 strong Pokemons based on HP level.

Additionally, I conduct exploratory data analysis to seek general classification and clustering of dataset. I provide ggplots, boxplots, histograms, density plots, and scatter plots to indicate some basic pattern in pokemons. Based on total attributes, I use attack, defense, special attack, special defense, and speed as the potential features of classification. Then I randomly split the dataset into two sets, 80% training set and 20% test set. I use LDA, QDA, KNN, PCA, random forest, and pruned tree to compute the test error of the classification.

For the first step, I input pokemon data and finish grouping as below:

```r
setwd("/Users/Xuanyi/Desktop/231/final project")
Pokemon <- read.csv("Pokemon.csv", header = TRUE)
colnames(Pokemon) <- c("Number", "Name", "Type1", "Type2", "Total", "HP",
                       "Attack", "Defense", "SP.ATK", "SP.DEF", "Speed",
                       "Generation", "Legendary")
Total <- Pokemon[,5]
Name <- Pokemon[,2]
HP <- Pokemon[,6]
Attack <- Pokemon[,7]
SP.ATK <- Pokemon[,9]
Defense <- Pokemon[,8]
SP.DEF <- Pokemon[,10]
Speed <- Pokemon[,11]
# Rank <= Pokemon[,21]

# Define the level of HP
for (i in 1:800){
  if(Pokemon[i,c(6)]<=50){
    Pokemon$HP_Level[i]<-as.character("weak")
  }
  else if (Pokemon[i,c(6)]>50 & Pokemon[i, c(6)]<80){
    Pokemon$HP_Level[i]<-as.character("normal")
  }
  else if (Pokemon[i,c(6)]>=80){
    k <- as.factor("Powerful")
    Pokemon$HP_Level[i]<-as.character("power")
  }
}

# Define the level of Total
for (i in 1:nrow(Pokemon)){
  if (Pokemon[i,c(5)]<=300){
    Pokemon$Total_level[i]=as.character("weak")
  }
  else if (Pokemon[i,c(5)]>300 & Pokemon[i,c(5)]<500){
    Pokemon$Total_level[i]=as.character("normal")
  }
  else if (Pokemon[i,c(5)]>=500){
    Pokemon$Total_level[i]=as.character("powerful")
  }
}
```

3 Exploratory Data Analysis

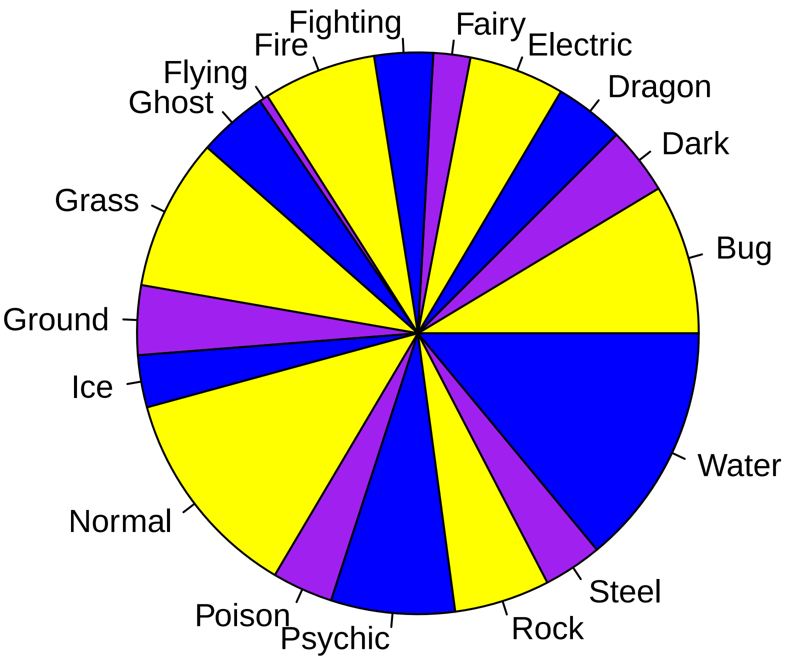
Here I show the results of exploratory data analysis based on plots and tables. Firstly I summarize the number of pokemons based on type and make a pie plot to make the scale of each type of pokemons more visually. There are 18 groups, in which grass, normal and water are three biggest ones.

Secondly, I generate the boxplots of HP and Total on each types to have a quick look at which type has the strongest health and overall battle performance. From the boxplots, I can tell the dragon pokemons are the best.

```
Type1 <- Pokemon[,3]
table(Pokemon$Type1)
```

```
## 
##      Bug     Dark   Dragon Electric    Fairy Fighting     Fire   Flying 
##       69       31       32       44       17       27       52        4 
##    Ghost    Grass   Ground      Ice   Normal   Poison  Psychic     Rock 
##       32       70       32       24       98       28       57       44 
##    Steel    Water 
##       27      112
```

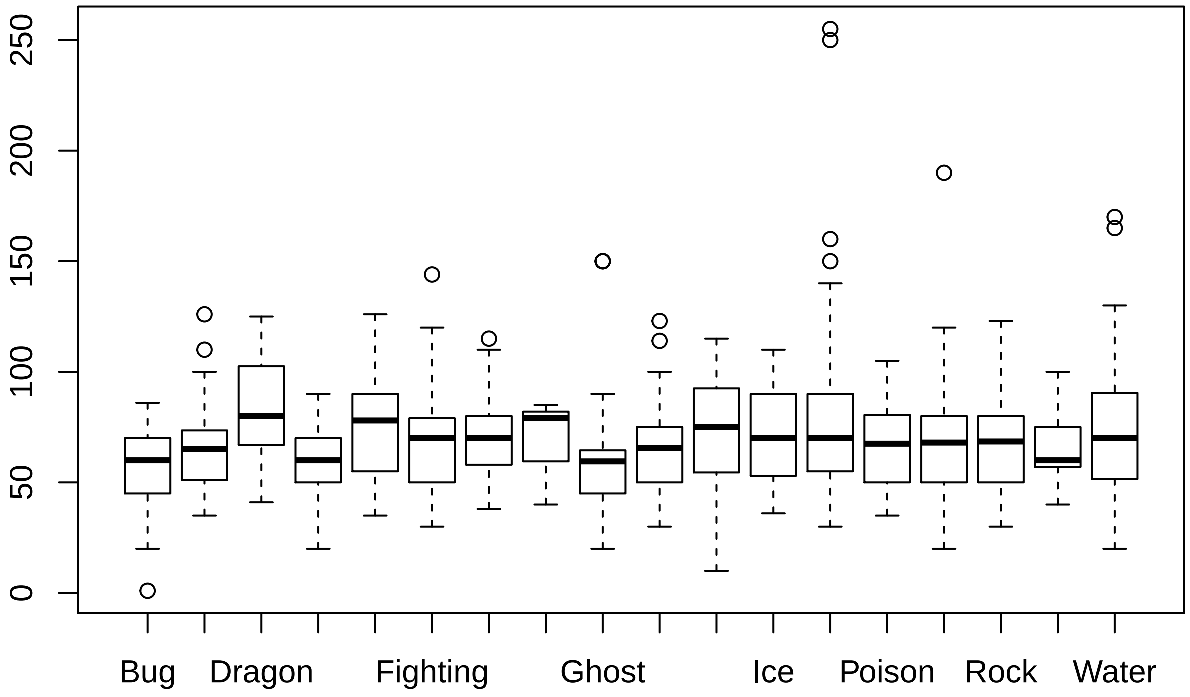
```
pie(table(Pokemon$Type1),col=c("yellow","purple","blue"), radius=1, edges=200)
```



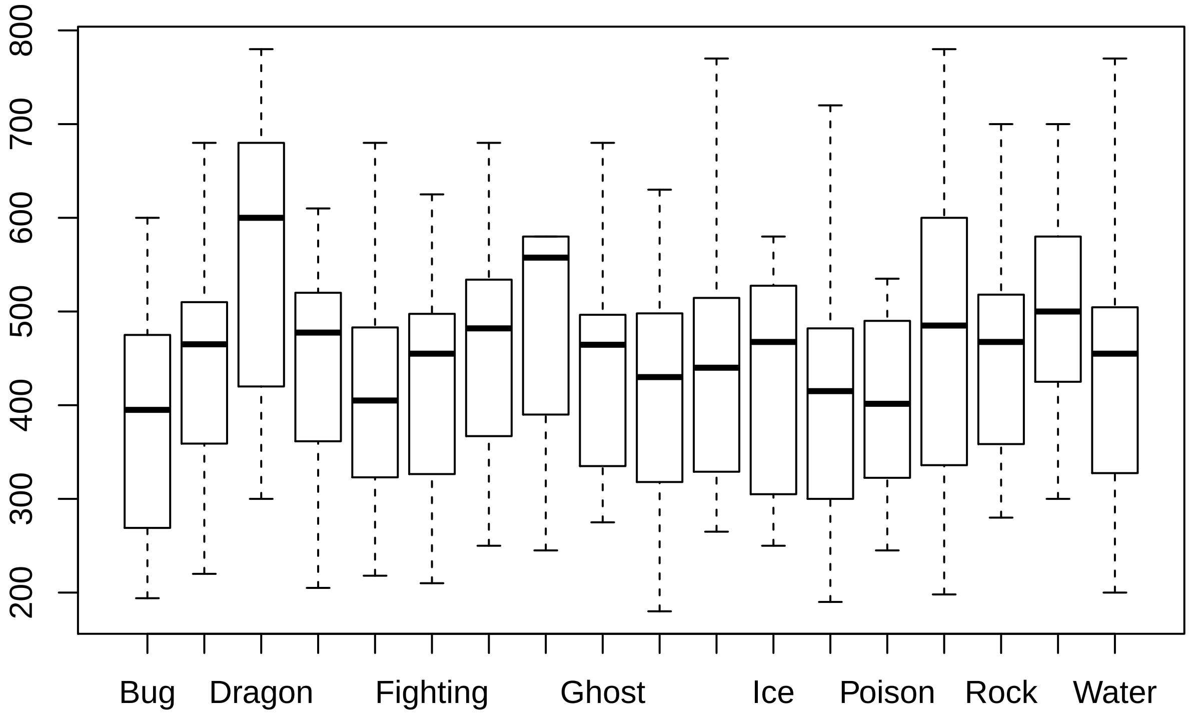
```r
aggregate(HP ~ Type1, summary, data=Pokemon)
```

```
##       Type1 HP.Min. HP.1st Qu. HP.Median HP.Mean HP.3rd Qu. HP.Max.
## 1       Bug     1.0       45.0      60.0    56.9       70.0    86.0
## 2      Dark    35.0       51.0      65.0    66.8       73.5   126.0
## 3    Dragon    41.0       67.5      80.0    83.3      101.0   125.0
## 4  Electric    20.0       50.0      60.0    59.8       70.0    90.0
## 5     Fairy    35.0       55.0      78.0    74.1       90.0   126.0
## 6  Fighting    30.0       50.0      70.0    69.9       79.0   144.0
## 7      Fire    38.0       58.0      70.0    69.9       80.0   115.0
## 8    Flying    40.0       69.2      79.0    70.8       80.5    85.0
## 9     Ghost    20.0       45.0      59.5    64.4       64.2   150.0
## 10    Grass    30.0       51.2      65.5    67.3       75.0   123.0
## 11   Ground    10.0       56.8      75.0    73.8       91.2   115.0
## 12      Ice    36.0       54.0      70.0    72.0       90.0   110.0
## 13   Normal    30.0       55.0      70.0    77.3       90.0   255.0
## 14   Poison    35.0       50.0      67.5    67.2       80.2   105.0
## 15  Psychic    20.0       50.0      68.0    70.6       80.0   190.0
## 16     Rock    30.0       50.0      68.5    65.4       80.0   123.0
## 17    Steel    40.0       57.0      60.0    65.2       75.0   100.0
## 18    Water    20.0       52.2      70.0    72.1       90.2   170.0
```

```r
boxplot(HP ~ Type1, data=Pokemon, title="Hit Points distinguished by Type 1") # Revise x-axis
```



```r
boxplot(Total ~ Type1, data=Pokemon, fill=Type1,
        title="Total Fighting Capacity distinguished by Type 1")
```

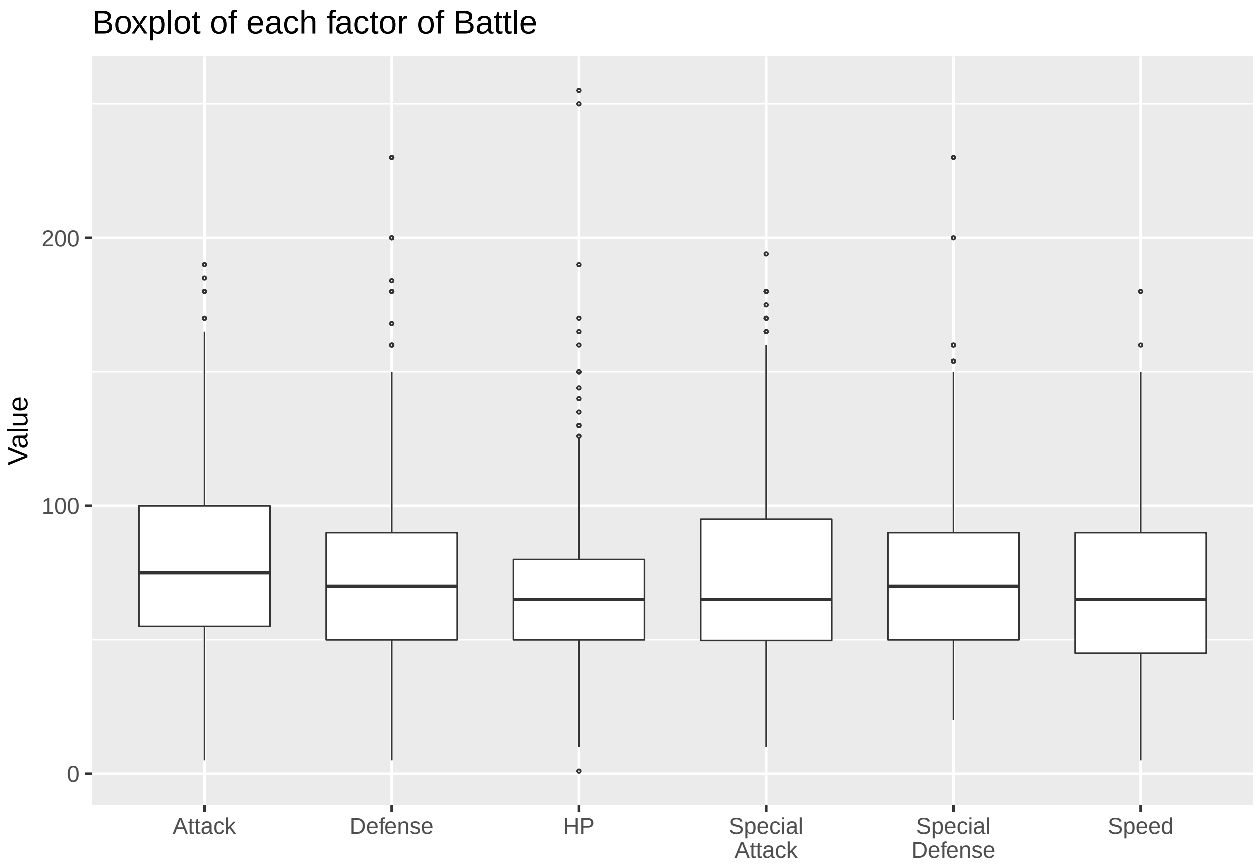


Because the battle performance includes many attributes, the following parts is going to explore the factors that can make a Pokemon to win a battle. First of all, I convert the raw data from wide format to the long format. In the long format, each row is one time point per subject. So each subject will have data in multiple rows. Based on the boxplot, it demonstrates that attack and special attack have the highest values, so I can draw a conclusion that these two factors contribute to the victory mostly.

```r
# Box-Plot
# Convert data to the long format
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(tidyr)
Generation <- Pokemon[,12]
Total <- Pokemon[,5]
Total_Level <- Pokemon[,15]
HP_Level <- Pokemon[,14]
Generation <- as.factor(Generation)
pkmon_long <- Pokemon %>%
  gather(Attribute, Value, Attack, Defense, HP, SP.ATK, SP.DEF, Speed, -Generation,
         factor_key = TRUE) %>%
  mutate(Generation) %>%
  mutate(Dual_type <- Type2 != "")

# Label the names for details in attribute
pkmon_long$Attribute <- factor(
  pkmon_long$Attribute,
  labels=c("Attack", "Defense", "HP", "Special\nAttack", "Special\nDefense", "Speed"))

options(repr.plot.width=6, repr.plot.height=6)
library(ggplot2)
ggplot(pkmon_long) +
  geom_boxplot(aes(x=Attribute, y=Value), lwd=0.3,
               width = 0.7, outlier.shape = 1, outlier.size=0.3) +
  scale_x_discrete(name="") +
  ggtitle("Boxplot of each factor of Battle ")
```

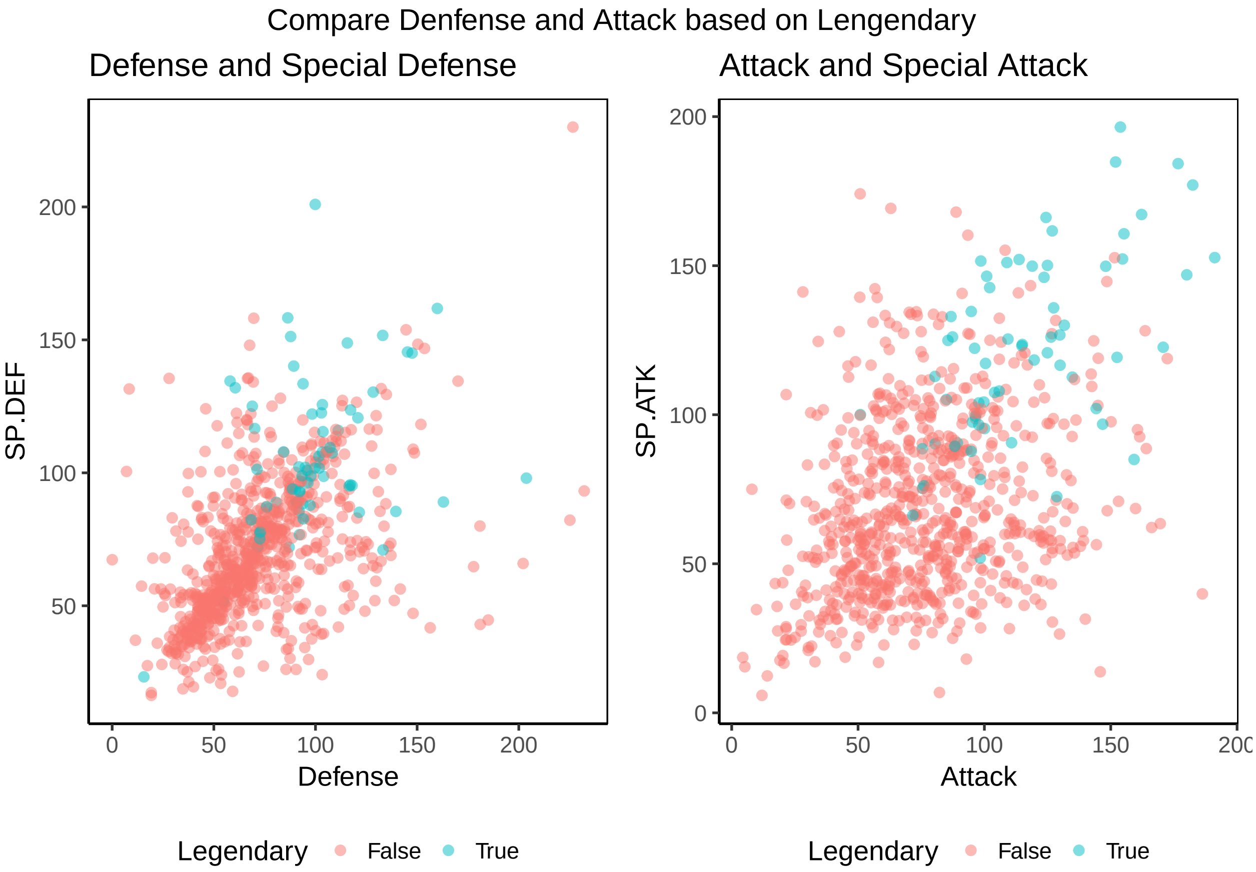


Then I generate scatter plots, based on defense and attack variables, to check the influence of legendary. The result tells the legendary is a positive factor that improve the attack, special attack, defense, and special defense at the same time.

```
# Scatter plots
library(ggplot2)
library(grid)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine
```

```
g1 <- ggplot(Pokemon) +
  geom_point(aes(x=Defense, y=SP.DEF, colour=Legendary),
             alpha=0.5, position=position_jitter(width=5, height=5)) +
  ggtitle("Defense and Special Defense") +
  theme_classic() +
  theme(legend.position="bottom",
        panel.border=element_rect(linetype=1, fill=NA))

g2 <- ggplot(Pokemon) +
  geom_point(aes(x=Attack, y=SP.ATK, colour=Legendary),
             alpha=0.5, position=position_jitter(width=5, height=5)) +
  ggtitle("Attack and Special Attack") +
  theme_classic() +
  theme(legend.position="bottom",
        panel.border=element_rect(linetype=1, fill=NA))

grid.arrange(g1,g2,nrow=1,ncol=2,
             top = "Compare Denfense and Attack based on Lengendary")
```



Following that, I am interested in exploring the strongest character for each type of Pokemons. To achieve this goal, I use density plots. I divide each character by Total to get the percentage number. From the ggplot, I can tell Ghost type has the highest speed that weighted 20% of total attribution; Poison type has the highest attack that weighted 16% of total attribution; Flying type has the highest special attack, defense and special defense; and Steel type has the highest HP.

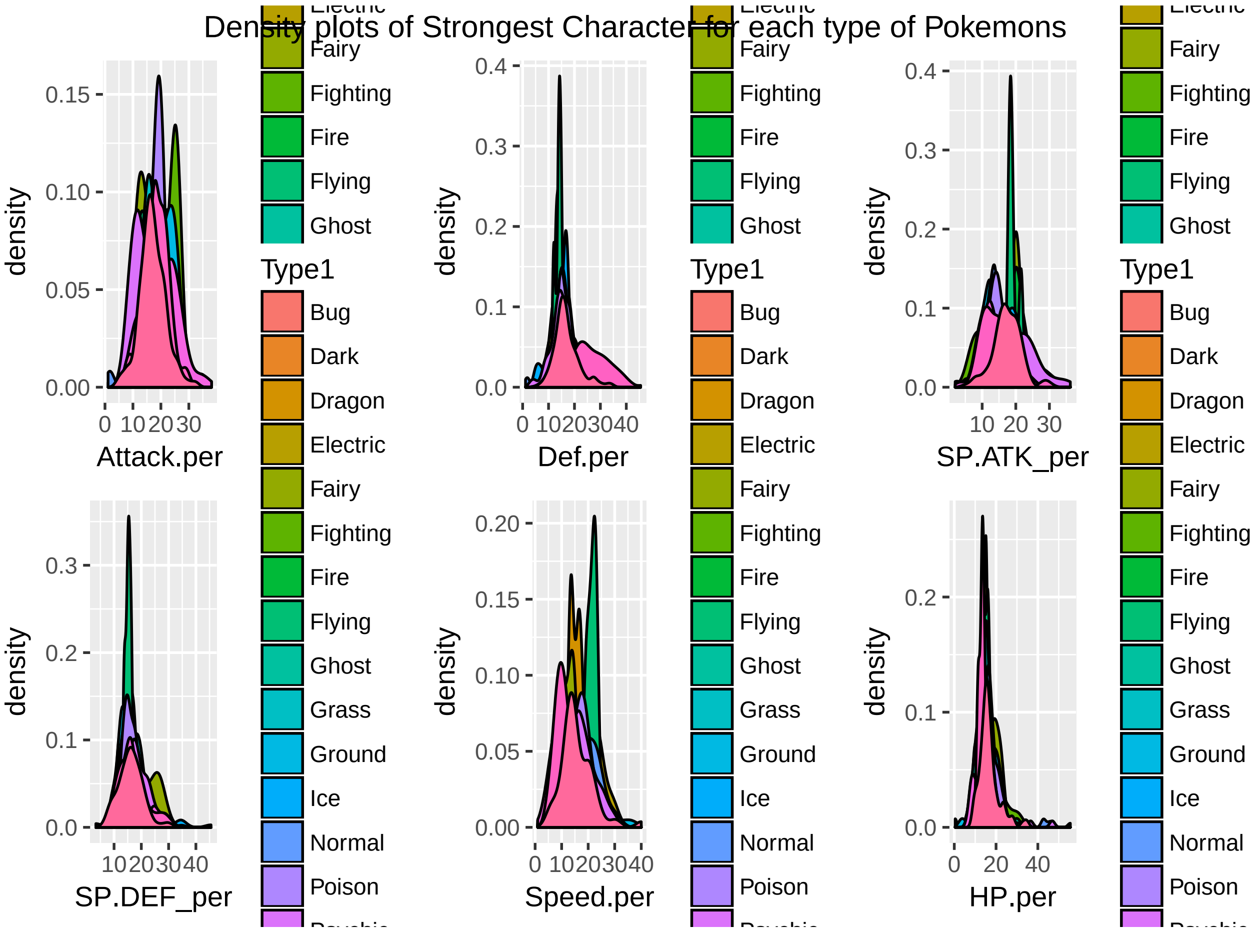
I also use histogram to show the distribution of each individual variable, in order to seek whether it contains special character. Based on the histograms, all variable follow normal distribution, which consistent with the number of pokemons in different levels.

```
### GGplot ###
library(ggplot2)
library(grid)
library(gridExtra)
Pokemon$Speed.per=(Pokemon$Speed/Pokemon$Total)*100
Pokemon$Attack.per=(Pokemon$Attack/Pokemon$Total)*100
Pokemon$SP.ATK_per=(Pokemon$SP.ATK/Pokemon$Total)*100
Pokemon$Def.per=(Pokemon$Defense/Pokemon$Total)*100
Pokemon$SP.DEF_per=(Pokemon$SP.DEF/Pokemon$Total)*100
Pokemon$HP.per=(Pokemon$HP/Pokemon$Total)*100
```

```r
names(Pokemon)
```

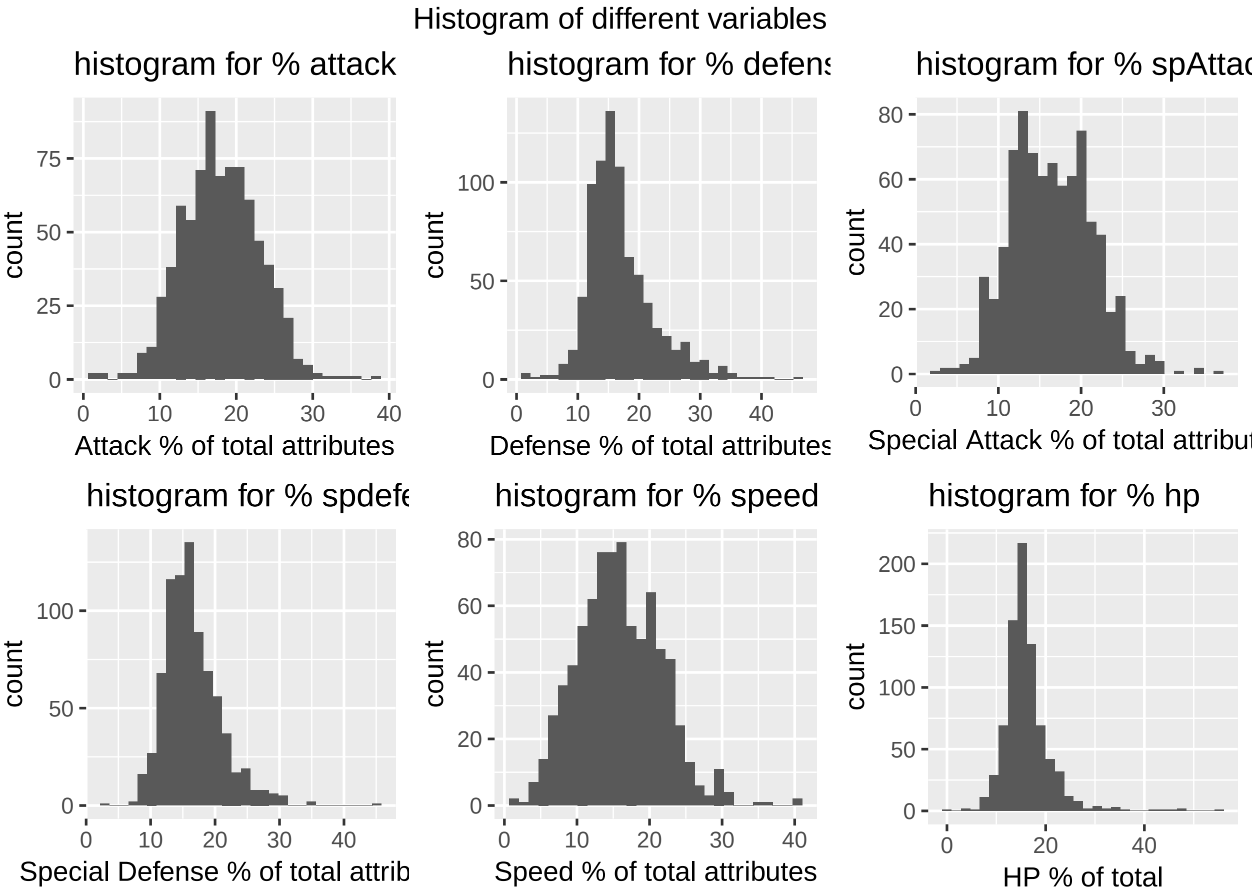
```
##  [1] "Number"      "Name"        "Type1"       "Type2"       "Total"
##  [6] "HP"          "Attack"      "Defense"     "SP.ATK"      "SP.DEF"
## [11] "Speed"       "Generation"  "Legendary"   "HP_Level"    "Total_level"
## [16] "Speed.per"   "Attack.per"  "SP.ATK_per"  "Def.per"     "SP.DEF_per"
## [21] "HP.per"
```

```r
# select Type1 and these percentage values
pok <- Pokemon[c(3,16:21)]


g1 <- ggplot(pok,aes(Attack.per,fill=Type1,colors=Type1))+geom_density() # poison
g2 <- ggplot(pok,aes(Def.per,fill=Type1,colors=Type1))+geom_density() # flying
g3 <- ggplot(pok,aes(SP.ATK_per,fill=Type1,colors=Type1))+geom_density() # flying
g4 <- ggplot(pok,aes(SP.DEF_per,fill=Type1,colors=Type1))+geom_density() # flying
g5 <- ggplot(pok,aes(Speed.per,fill=Type1,colors=Type1))+geom_density() # ghost
g6 <- ggplot(pok,aes(HP.per,fill=Type1,colors=Type1))+geom_density() # steel

grid.arrange(g1,g2,g3,g4,g5,g6,nrow=2,ncol=3,  heights=c(0.5, 0.5),
             top = "Density plots of Strongest Character for each type of Pokemons")
```



```r
q1 <- qplot(pok$Attack.per,geom = "histogram",
            main = "histogram for % attack", xlab = "Attack % of total attributes")
q2 <- qplot(pok$Def.per,geom = "histogram",
            main = "histogram for % defense", xlab = "Defense % of total attributes")
q3 <- qplot(pok$SP.ATK_per,geom = "histogram",
            main = "histogram for % spAttack", xlab = "Special Attack % of total attributes")
q4 <- qplot(pok$SP.DEF_per,geom = "histogram",
            main = "histogram for % spdefense", xlab = "Special Defense % of total attributes")
q5 <- qplot(pok$Speed.per,geom = "histogram",
            main = "histogram for % speed", xlab = "Speed % of total attributes")
q6 <- qplot(pok$HP.per,geom = "histogram",
            main = "histogram for % hp", xlab = "HP % of total")
grid.arrange(q1,q2,q3,q4,q5,q6,nrow=2,ncol=3, top = "Histogram of different variables")
```

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



4 Classification

4.1 LDA

I begin the classification with linear discriminant analysis and quadratic discriminant analysis. I intend to test the feasibility of Total_Level and HP_Level, and I randomly sample and split each dataset into 80% training set and 20% test set. I compare three classification techniques namely LDA, QDA, and KNN by using the Pokemon dataset.

```
library(data.table)
```

```
##
## This data.table install has not detected OpenMP support. It will work but slower in single threaded r
```

```
## -------------------------------------------------------------------------
```

```
## data.table + dplyr code now lives in dtplyr.
## Please library(dtplyr)!
```

```
## -------------------------------------------------------------------------
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
library(class)
# split into two sets
pkmon <- as.data.frame(Pokemon)
X.pkmon <- pkmon[,c(6:12)]
Y.pkmon <- pkmon[15] # Total_Level
Z.pkmon <- pkmon[14] # HP_Level
train_ind <- sample(1:nrow(pkmon), size = 0.8*nrow(pkmon))

# Total_Level
pkmon.training.predictor.t <- X.pkmon[train_ind,]
pkmon.training.response.t <- Y.pkmon[train_ind,]
pkmon.train.t <- as.data.table(cbind(pkmon.training.predictor.t,
                                     pkmon.training.response.t))
pkmon.test.predictor.t <- X.pkmon[-train_ind,]
pkmon.test.response.t <- Y.pkmon[-train_ind,]

# HP_Level
pkmon.training.predictor.h <- X.pkmon[train_ind,]
pkmon.training.response.h <- Z.pkmon[train_ind,]
pkmon.train.h <- as.data.table(cbind(pkmon.training.predictor.h,
                                     pkmon.training.response.h))
pkmon.test.predictor.h <- X.pkmon[-train_ind,]
pkmon.test.response.h <- Z.pkmon[-train_ind,]
```

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
```

```
##
##     select
```

```r
LDA <- lda(Total_Level~.,
           data = pkmon[6:12],
           subset = train_ind)
predict.lda.train <- predict(object=LDA)
predict.lda.test <- predict(object=LDA, newdata=pkmon.test.predictor.t)
#training error rate
lda.er.train <- sum(predict.lda.train$class!=pkmon.training.response.t) /
                                  length(pkmon.training.response.t)
lda.er.train
```

```
## [1] 0.04688
```

```r
#test error rate
lda.er.test <- sum(predict.lda.test$class != pkmon.test.response.t) /
                                 length(pkmon.test.response.t)
lda.er.test
```

```
## [1] 0.075
```

```r
LDA <- lda(HP_Level~.,
           data = pkmon[6:12],
           subset = train_ind)
predict.lda.train <- predict(object=LDA)
predict.lda.test <- predict(object=LDA, newdata=pkmon.test.predictor.h)
#training error rate
lda.er.train <- sum(predict.lda.train$class!=pkmon.training.response.h)/
                            length(pkmon.training.response.h)
lda.er.train
```

```
## [1] 0.1453
```

```r
#test error rate
lda.er.test <- sum(predict.lda.test$class != pkmon.test.response.h)/
                             length(pkmon.test.response.h)
lda.er.test
```

```
## [1] 0.1062
```

LDA of Total level obtain a test error of 0.0625, while LDA of HP level get a test error of 0.1375. At this step, I can tell the Total level classification is better than HP level. Then I use QDA to check whether I can get a different result.

4.2 QDA

```r
library(MASS)
QDA <- qda(Total_Level~.,
           data = pkmon[6:12],
           subset = train_ind)
predict.qda.train <- predict(object=QDA)
predict.qda.test <- predict(object=QDA, newdata=pkmon.test.predictor.t)
#training error rate
qda.er.train <- sum(predict.lda.train$class!=pkmon.training.response.t)/
                      length(pkmon.training.response.t)
qda.er.train
```

```
## [1] 0.5141
```

```
#test error rate
qda.er.test <- sum(predict.lda.test$class != pkmon.test.response.t)/
                       length(pkmon.test.response.t)
qda.er.test
```

```
## [1] 0.4813
```

```
QDA <- qda(HP_Level~.,
             data = pkmon[6:12],
             subset = train_ind)
predict.qda.train <- predict(object=QDA)
predict.qda.test <- predict(object=QDA, newdata=pkmon.test.predictor.h)
#training error rate
qda.er.train <- sum(predict.lda.train$class!=pkmon.training.response.h)/
                              length(pkmon.training.response.h)
qda.er.train
```

```
## [1] 0.1453
```

```
#test error rate
qda.er.test <- sum(predict.lda.test$class != pkmon.test.response.h)/
                              length(pkmon.test.response.h)
qda.er.test
```

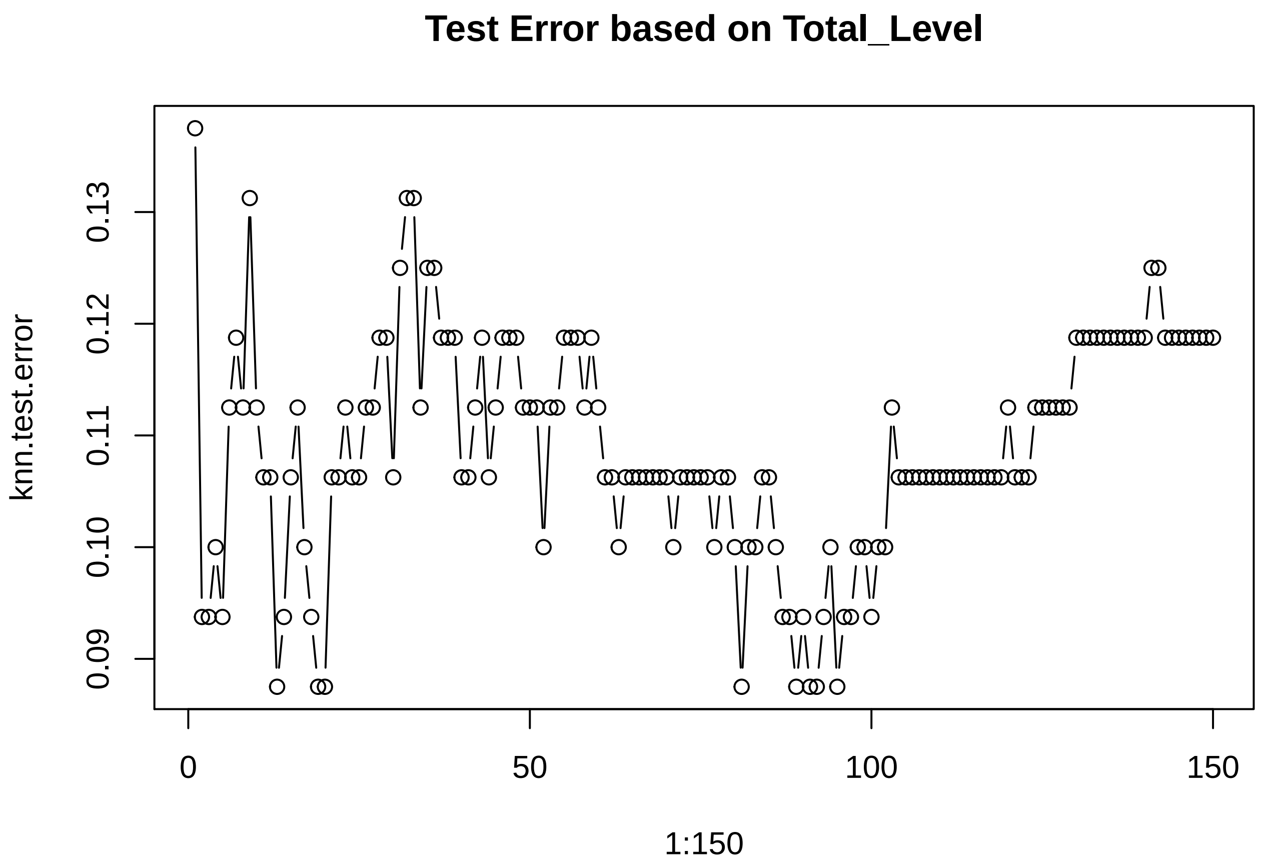
```
## [1] 0.1062
```

QDA of Total level obtain a test error rate of 0.0641, and QDA of HP level obtain a test error rate of 0.3578. The result is still the same. Due to the limit data of weak Pokemons, the discriminant analysis models fail to classify the weak Pokemons based on total level.

4.3 KNN

To train a more flexible model, I generate KNN for different classification methods. Given the predictors, I get the minimum test error rate is around 0.06, which is similar to the LDA and QDA results.

```
# Total level test error
knn.test.error <- array(rep(0,150))
for (i in 1:150){
  knn.fit <- knn(train = pkmon.training.predictor.t,
                 test = pkmon.test.predictor.t,
                 cl = pkmon.training.response.t,
                  k = i);
  error.matrix <- table(knn.fit, pkmon.test.response.t);
  knn.test.error [i] <- (error.matrix[1,2] +
                           error.matrix[2,1])/sum(error.matrix)
}

plot(1:150, knn.test.error, type = "b")
title("Test Error based on Total_Level")
```



```
min(knn.test.error)
```

```
## [1] 0.0875
```

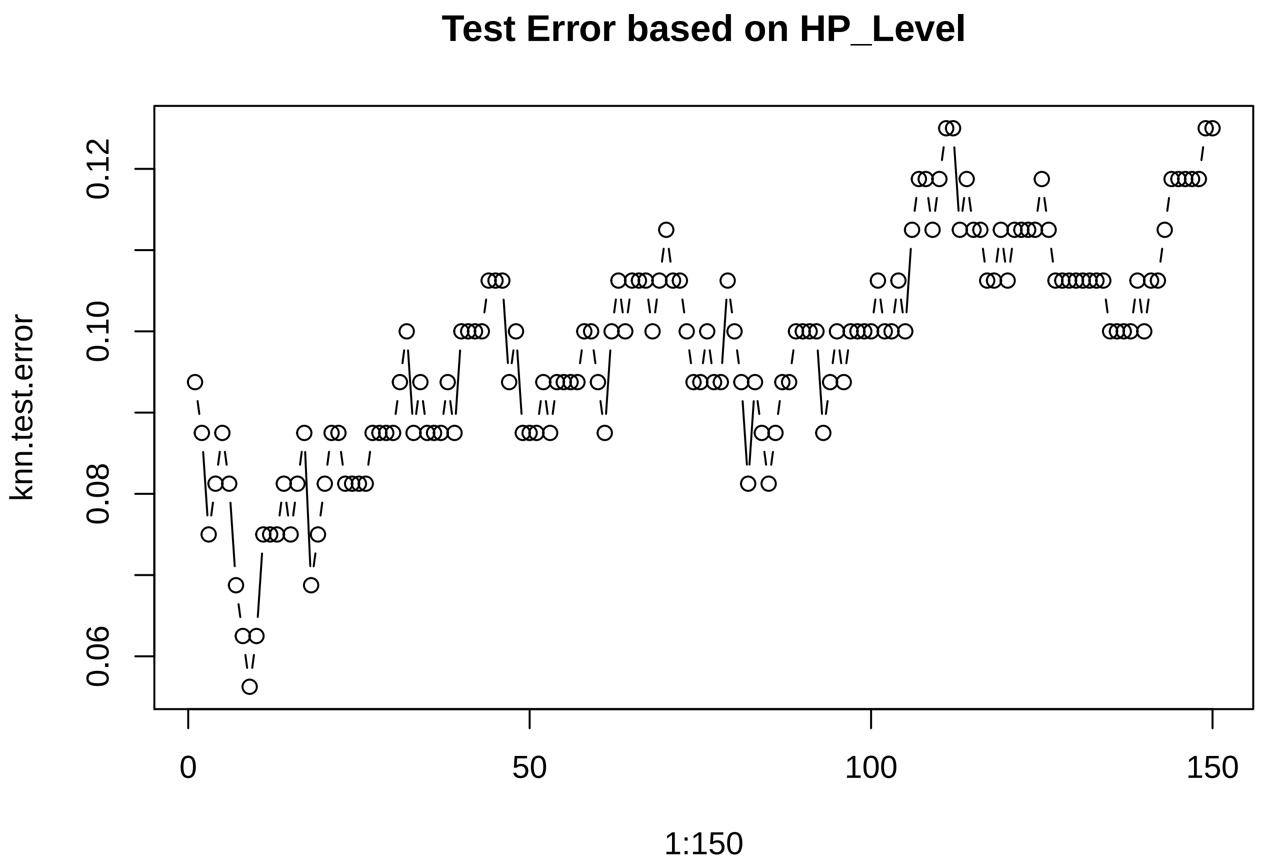
```
which.min(knn.test.error)
```

```
## [1] 13
```

```
# Using knn.cv
cv.pkm <- knn.cv(pkmon.test.predictor.t, pkmon.test.response.t, k = 4, prob = FALSE)
cv.er <- sum(cv.pkm!=pkmon.test.response.t)/length(pkmon.test.response.t)
cv.er  # k=2 ---> 0.14375  k=3 ---> 0.0875
```

```
## [1] 0.2
```

```
# HP_Level test error
knn.test.error <- array(rep(0,150))
for (i in 1:150){
  knn.fit <- knn(train = pkmon.training.predictor.h,
                test = pkmon.test.predictor.h,
                cl = pkmon.training.response.h,
                 k = i);
  error.matrix <- table(knn.fit, pkmon.test.response.h);
  knn.test.error [i] <- (error.matrix[1,2] +
                            error.matrix[2,1])/sum(error.matrix)
}

plot(1:150, knn.test.error, type = "b")
title("Test Error based on HP_Level")
```



```
min(knn.test.error)  # 0.0625
```

```
## [1] 0.05625
```

```
which.min(knn.test.error) # 6
```

```
## [1] 9
```

```
# Using knn.cv
cv.pkm <- knn.cv(pkmon.test.predictor.h, pkmon.test.response.h, k = 5, prob = FALSE)
cv.er <- sum(cv.pkm!=pkmon.test.response.h)/length(pkmon.test.response.h)
cv.er  # k=3 ---> 0.2125, k=2 ---> 0.2  k=4 ---> 0.24375  k=5 ---> 0.2125
```

```
## [1] 0.1938
```

4.4 PCA + KNN

PCA is a crucial method to reduce the data into two or three dimension. For the pokemon data, there are a bunch of variables and observations, so it is a good way to visualize data in lower dimensions. After perform PCA on the data, I recapitualte the first four principle components. That indicates 89% of the variance in data can be explained by the PC1, PC2, PC3, and PC4. Besides, the correlation of variables are not significant, which is justfied by the correlation matrix.

Then I plot biplot for PCA. The first two PC1 and PC2 explain 65% of the variance in dataset, which is not sufficient amount but still feasible.

Following that, I combine the PCA and KNN to do the clustering. I tried different numbers of clusters I am going to generate. Based on the plots, I can tell the results show good indicator that the pokemons could be clustered clearly.

```
pkmon <- as.data.frame(Pokemon)
Type1ID <- sapply(Pokemon[,3], as.numeric)
Pokemon$Type1ID <- Type1ID


centrp <- apply(pkmon.training.predictor.t,2,median)
dispp <- apply(pkmon.training.predictor.t,2,mad)

pkmon.training.predictor.p <- scale(pkmon.training.predictor.t,
                                center=centrp,scale=dispp)

pca.fit <- prcomp(pkmon.training.predictor.t, cor=F)
```

```
## Warning: In prcomp.default(pkmon.training.predictor.t, cor = F) :
##  extra argument 'cor' will be disregarded
```

```
summary(pca.fit)
```

```
## Importance of components:
##                            PC1    PC2    PC3    PC4     PC5     PC6    PC7
## Standard deviation     48.996 31.568 27.461 23.766 20.0036 15.0156 1.6383
## Proportion of Variance  0.449  0.186  0.141  0.106  0.0749  0.0422 0.0005
## Cumulative Proportion   0.449  0.636  0.777  0.882  0.9573  0.9995 1.0000
```

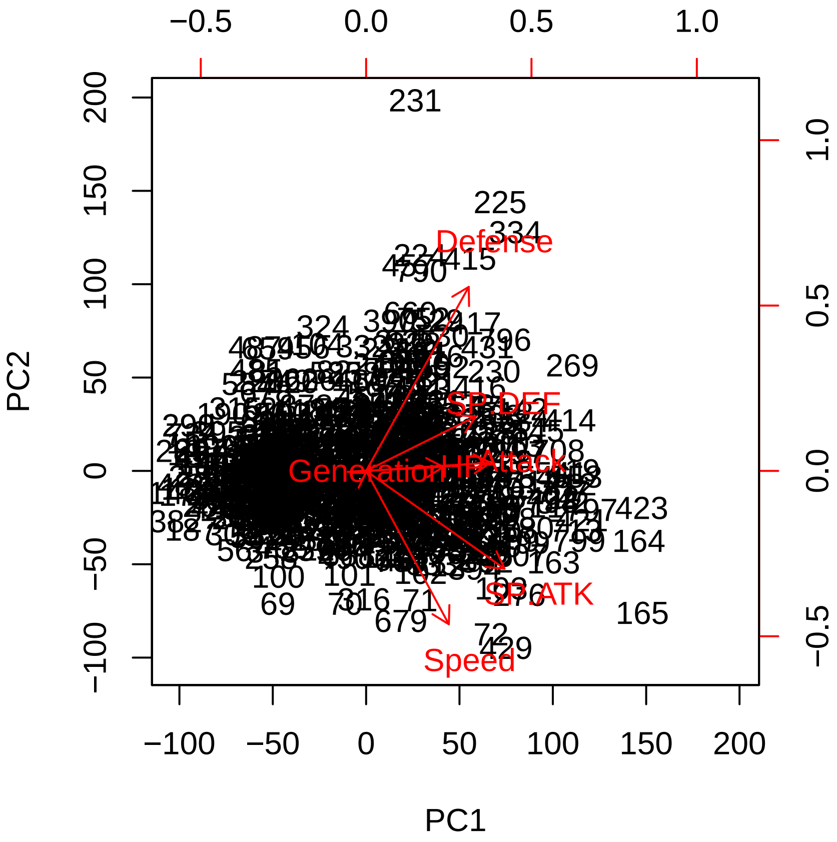
```
summary(pca.fit)$importance[,4]
```

```
##     Standard deviation Proportion of Variance  Cumulative Proportion
##                23.7662                 0.1057                 0.8824
```

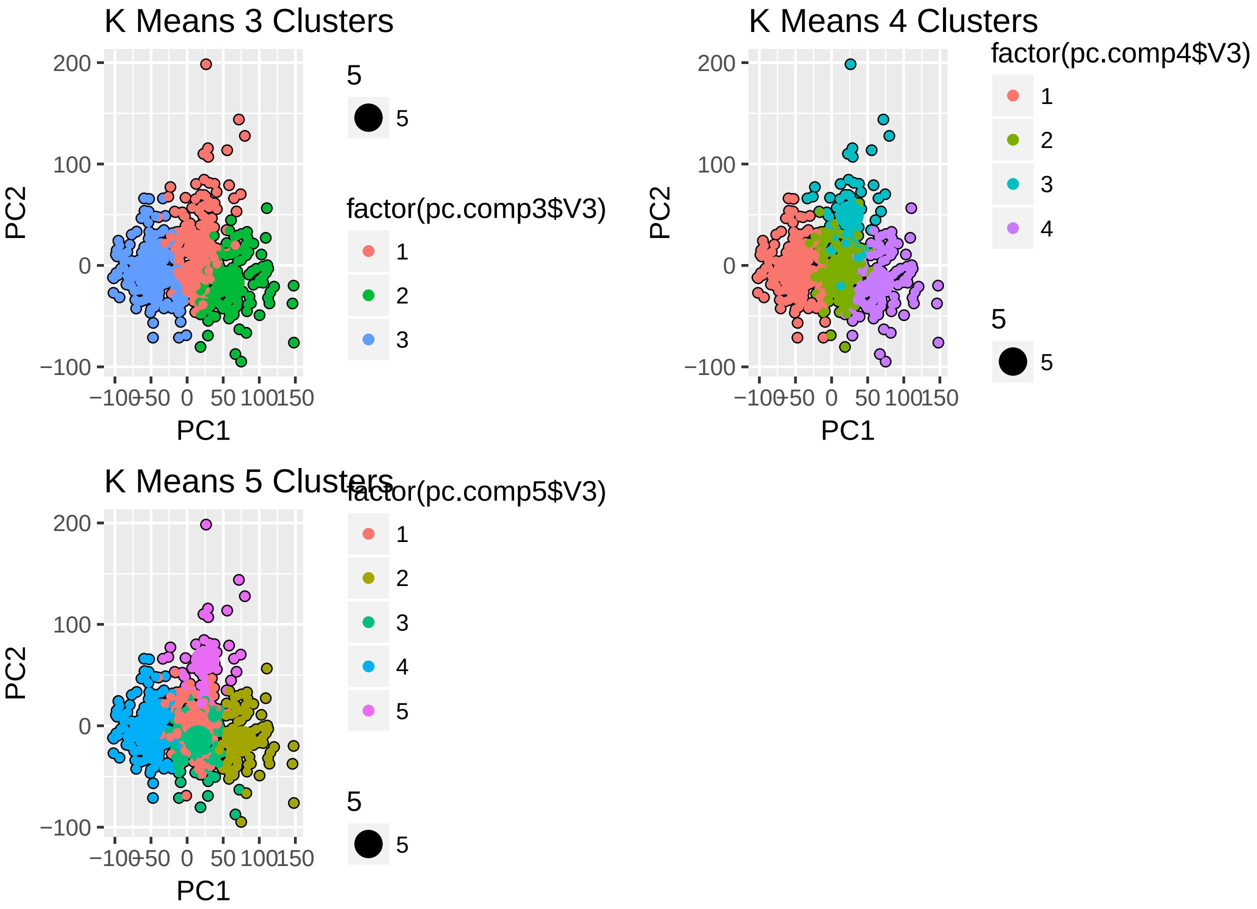
```
cor(pkmon.training.predictor.p)
```

```
##                 HP  Attack  Defense  SP.ATK  SP.DEF     Speed Generation
## HP         1.00000 0.38795 0.203971 0.33860 0.34065  0.147135    0.07959
## Attack     0.38795 1.00000 0.430685 0.37494 0.24322  0.378421    0.06929
## Defense    0.20397 0.43069 1.000000 0.22354 0.52693  0.003872    0.05751
## SP.ATK     0.33860 0.37494 0.223541 1.00000 0.51959  0.441455    0.06512
## SP.DEF     0.34065 0.24322 0.526926 0.51959 1.00000  0.236932    0.04297
## Speed      0.14714 0.37842 0.003872 0.44145 0.23693  1.000000   -0.01705
## Generation 0.07959 0.06929 0.057512 0.06512 0.04297 -0.017051    1.00000
```

```
pca.fit$rotation=-pca.fit$rotation
pca.fit$x=-pca.fit$x
biplot(pca.fit , scale =0)
```



```
# KNN procedure for clusters
# K=3
library(class)
pc.comp <- pca.fit$x[,1:2]
km.cl.3 <- kmeans(pkmon.training.predictor.p,centers=3)
pc.comp3 <- as.data.frame(cbind(pc.comp,km.cl.3$cluster))
centroids.3 <- aggregate(.~km.cl.3$cluster, pc.comp3,mean)
km.3 <- qplot(x=pc.comp3$PC1,y=pc.comp3$PC2,
main="K Means 3 Clusters", xlab="PC1", ylab= "PC2") +
geom_point(aes(color=factor(pc.comp3$V3)),size=1) +
   geom_point(data=centroids.3,aes(x=centroids.3[,2], y=centroids.3[,3],
color=factor(centroids.3[,1]), size=5))

# K=4
km.cl.4 <- kmeans(pkmon.training.predictor.p,centers=4)
pc.comp4 <- as.data.frame(cbind(pc.comp,km.cl.4$cluster))
centroids.4 <- aggregate(.~km.cl.4$cluster, pc.comp4,mean)
km.4 <- qplot(x=pc.comp4$PC1,y=pc.comp4$PC2,
main="K Means 4 Clusters", xlab="PC1", ylab= "PC2") +
geom_point(aes(color=factor(pc.comp4$V3)),size=1) + geom_point(data=centroids.4,aes(x=centroids.4[,2],y
color=factor(centroids.4[,1]), size=5))

# K=5
km.cl.5 <- kmeans(pkmon.training.predictor.p,centers=5)
pc.comp5 <- as.data.frame(cbind(pc.comp,km.cl.5$cluster))
centroids.5 <- aggregate(.~km.cl.5$cluster, pc.comp5,mean)
km.5 <- qplot(x=pc.comp5$PC1,y=pc.comp5$PC2,
main="K Means 5 Clusters",xlab="PC1",ylab="PC2") +
geom_point(aes(color=factor(pc.comp5$V3)),size=1) + geom_point(data=centroids.5,aes(x=centroids.5[,2],y
color=factor(centroids.5[,1]), size=5))

grid.arrange(km.3,km.4,km.5,ncol=2)
```



4.5 Pruned Tree

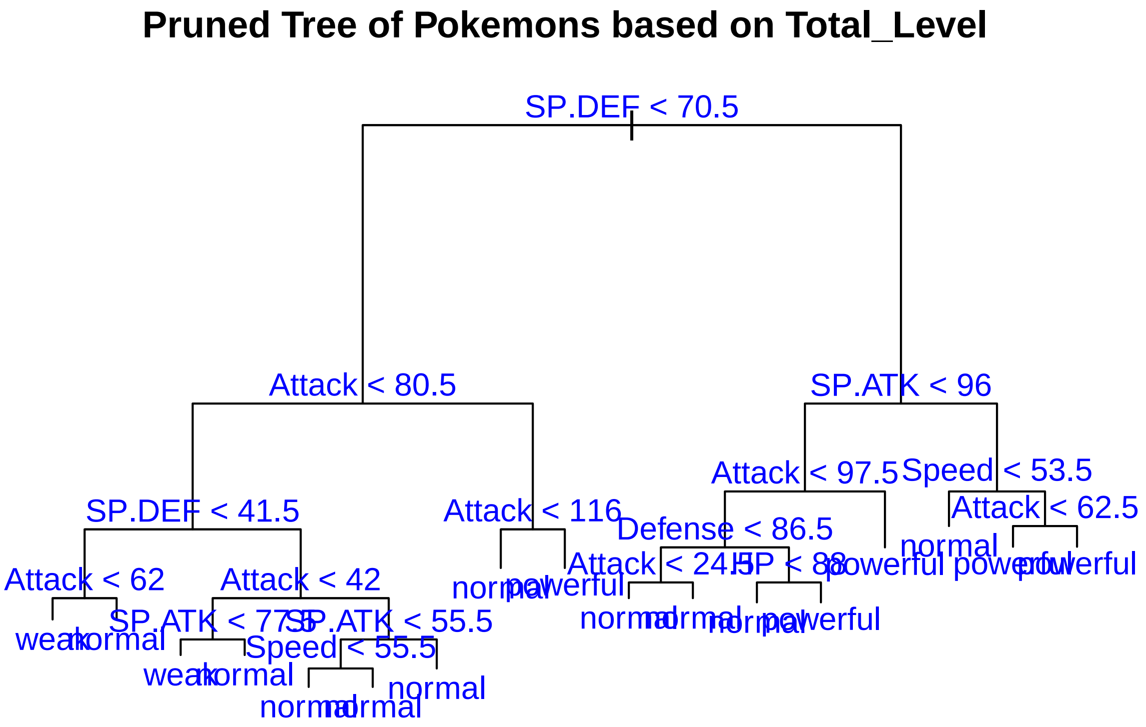
I create the decision tree for both divisions. Based on the total attributes groups, the pruned tree looks like a little bit massed up. That can be explained by six components in the calculation of total attributes, so the pruned tree would be generated by these different variables. As a result, the test error rate is larger than HP groups test error.

Compared with that, since the HP groups only contains one variable hit points, the pruned tree graph is very clear. In this condition, the test error close to zero. That makes sense due to the simpliest model I generate.

```
library(tree)
# Total
tree.t <- tree(pkmon.training.response.t ~.,
                data=pkmon.train.t)
summary(tree.t)
```

```
##
## Classification tree:
## tree(formula = pkmon.training.response.t ~ ., data = pkmon.train.t)
## Variables actually used in tree construction:
## [1] "SP.DEF"  "Attack"  "SP.ATK"  "Speed"   "Defense" "HP"
## Number of terminal nodes:  17
## Residual mean deviance:  0.685 = 427 / 623
## Misclassification error rate: 0.15 = 96 / 640
```

```
plot(tree.t)
text(tree.t, pretty = 0, col = "blue")
title("Pruned Tree of Pokemons based on Total_Level")
```



```
tree.pred.t <- predict(tree.t, pkmon.test.predictor.t,
                       type = "class")
tree.matrix.t <-table(tree.pred.t, pkmon.test.response.t)
tree.matrix.t
```

```
##              pkmon.test.response.t
## tree.pred.t normal powerful weak
##    normal       63       13    8
##    powerful     13       36    0
##    weak          1        0   26
```

```
tree.error.t <- (tree.matrix.t[1,2]+
                  tree.matrix.t[2,1])/sum(tree.matrix.t)
tree.error.t
```

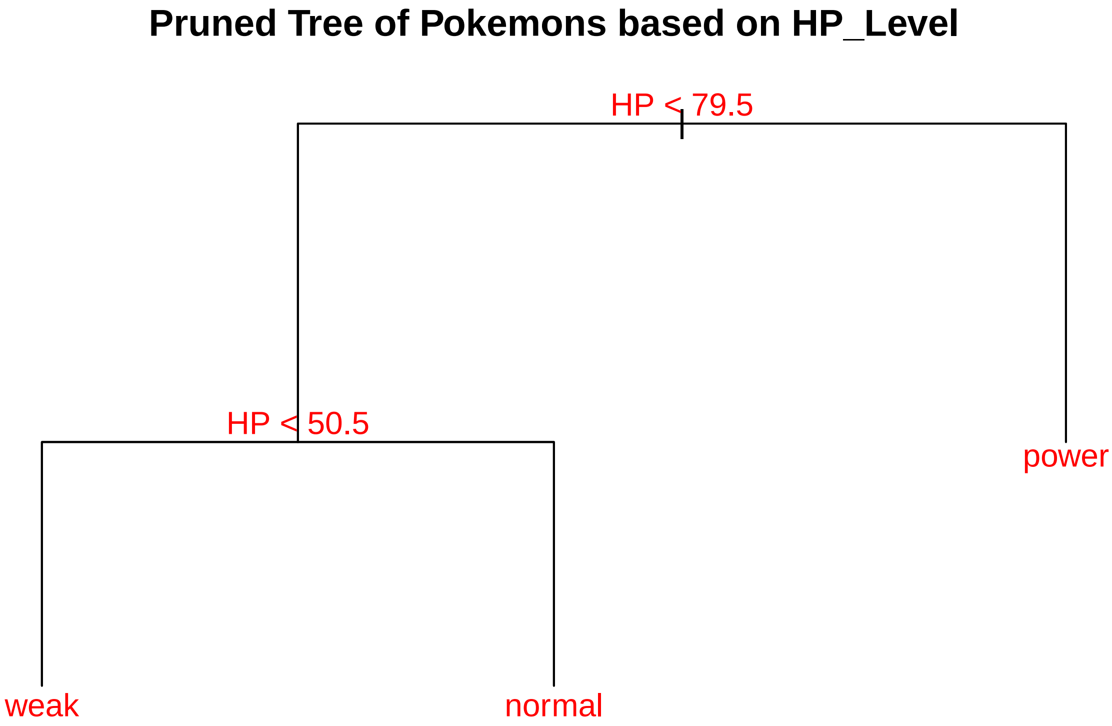
```
## [1] 0.1625
```

```
# HP
tree.h <- tree(pkmon.training.response.h ~.,
                data=pkmon.train.h)
summary(tree.h)
```

```
##
## Classification tree:
```

```
## tree(formula = pkmon.training.response.h ~ ., data = pkmon.train.h)
## Variables actually used in tree construction:
## [1] "HP"
## Number of terminal nodes:  3
## Residual mean deviance:  0 = 0 / 637
## Misclassification error rate: 0 = 0 / 640
```

```
plot(tree.h)
text(tree.h, pretty = 0, col = "red")
title("Pruned Tree of Pokemons based on HP_Level")
```



```
tree.pred.h <- predict(tree.h, pkmon.test.predictor.h,
                            type = "class")
tree.matrix.h <-table(tree.pred.h, pkmon.test.response.h)
tree.matrix.h
```

```
##              pkmon.test.response.h
## tree.pred.h normal power weak
##      normal     66     0    0
##      power       0    48    0
##      weak        0     0   46
```

```
tree.error.h <- (tree.matrix.h[1,2]+
                  tree.matrix.h[2,1])/sum(tree.matrix.h)
tree.error.h
```

```
## [1] 0
```

4.6 Random Forest

I try the random forest for both total level and hp level classifications. Based on the results below, I can tell the hp level is much better than total level. For total level, the test error is 0.1391, and the graph of the relative importance of variable shows the most important variables are special attack and attack. For hp level, the test error is 0.0016, which is pretty small. The importance variables plots indicate the most important variable is hp and attack.

```
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:gridExtra':
##
##     combine

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

## The following object is masked from 'package:dplyr':
##
##     combine
```

```
# Total
pkm.rf.t <- randomForest(pkmon.training.response.t ~., data = pkmon.train.t,
                        mtry = 3,importance = TRUE)
pkm.rf.t
```

```
##
## Call:
##  randomForest(formula = pkmon.training.response.t ~ ., data = pkmon.train.t,      mtry = 3, importanc
##                 Type of random forest: classification
##                       Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 12.19%
## Confusion matrix:
##          normal powerful weak class.error
## normal      329       15   11     0.07324
## powerful     28      170    0     0.14141
## weak         24        0   63     0.27586
```

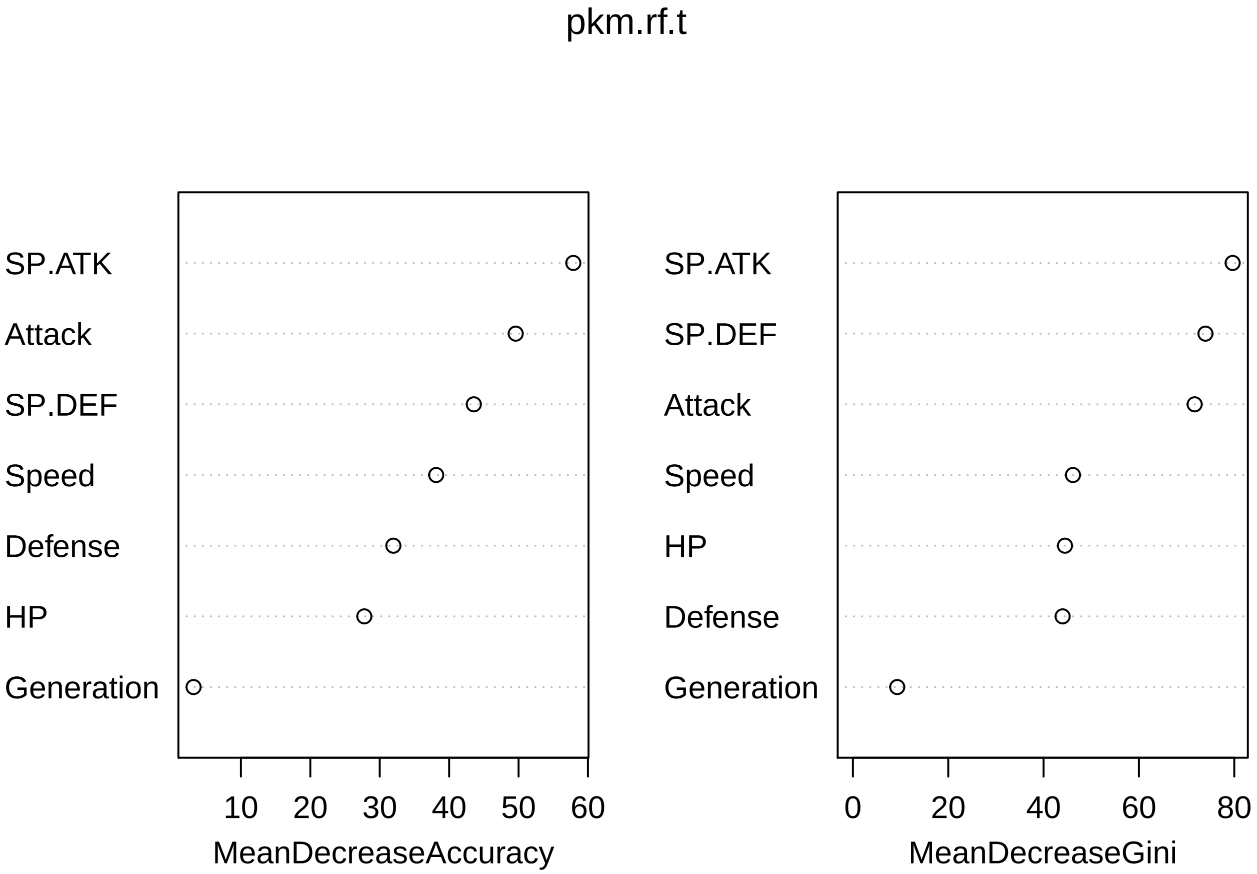
```
pkm.rf.pred.t <- predict(pkm.rf.t, pkmon.test.predictor.t,
                        type = "class")
# error rate
1 - sum(diag(pkm.rf.t$confusion))/sum(pkm.rf.t$confusion)
```

```
## [1] 0.1225
```

```
# test error
rf.confusion.t <- table(pkmon.test.response.t, pkm.rf.pred.t)
rf.test.error.t <- ((rf.confusion.t[1,2])
                  +(rf.confusion.t[2,1]))/sum(rf.confusion.t);
rf.test.error.t
```

```
## [1] 0.0875
```

```
# plot
varImpPlot(pkm.rf.t)
```



```
# HP
pkm.rf.h <- randomForest(pkmon.training.response.h ~., data = pkmon.train.h,
                        mtry = 3,importance = TRUE)
pkm.rf.h
```

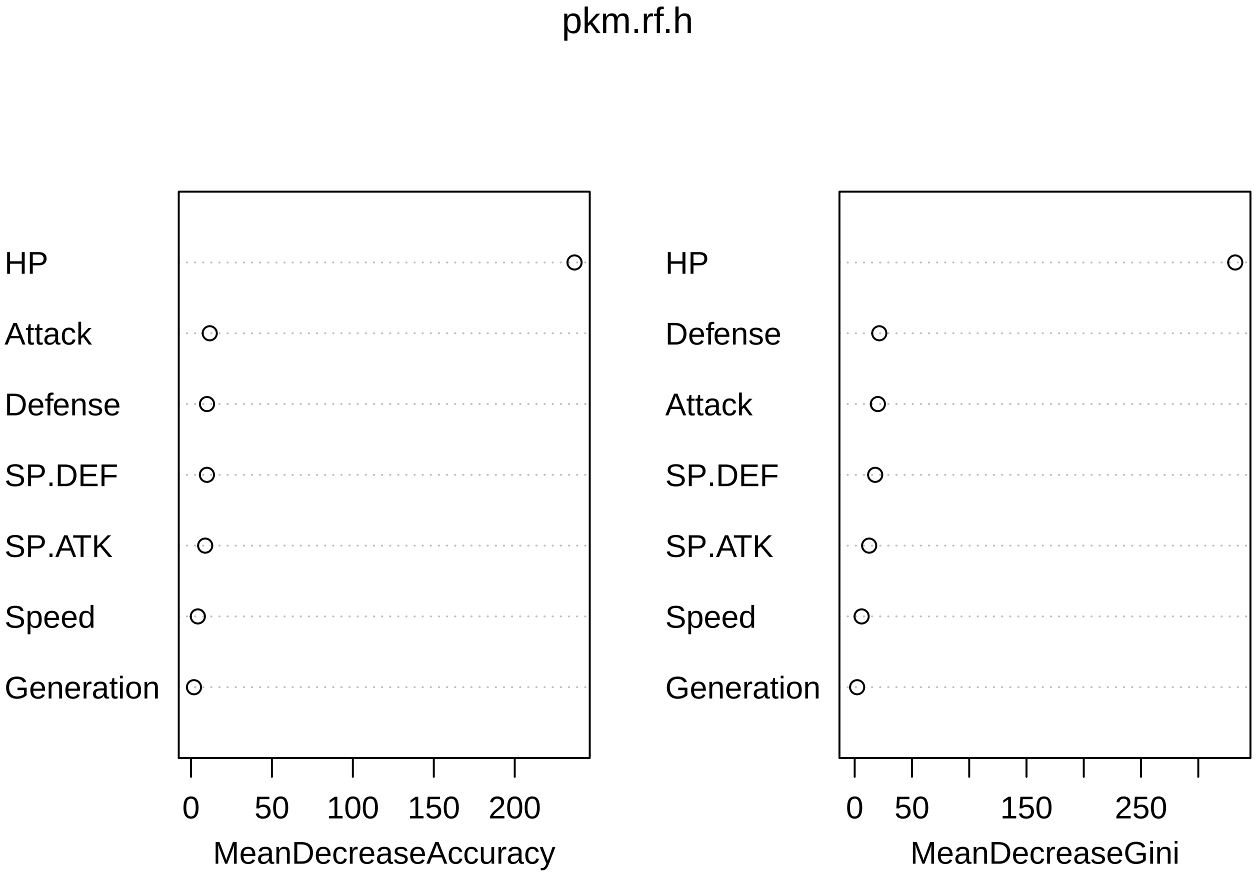
```
##
## Call:
##  randomForest(formula = pkmon.training.response.h ~ ., data = pkmon.train.h,      mtry = 3, importanc
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 0.16%
## Confusion matrix:
##        normal power weak class.error
## normal    286     0    1    0.003484
## power       0   188    0    0.000000
## weak        0     0  165    0.000000
```

```
pkm.rf.pred.h <- predict(pkm.rf.h, pkmon.test.predictor.h,
                        type = "class")
rf.confusion.h <- table(pkmon.test.response.h, pkm.rf.pred.h)
rf.test.error.h <- ((rf.confusion.h[1,2])
                  +(rf.confusion.h[2,1]))/sum(rf.confusion.h)
```

```r
rf.test.error.h
```

```
## [1] 0
```

```r
# plot
varImpPlot(pkm.rf.h)
```



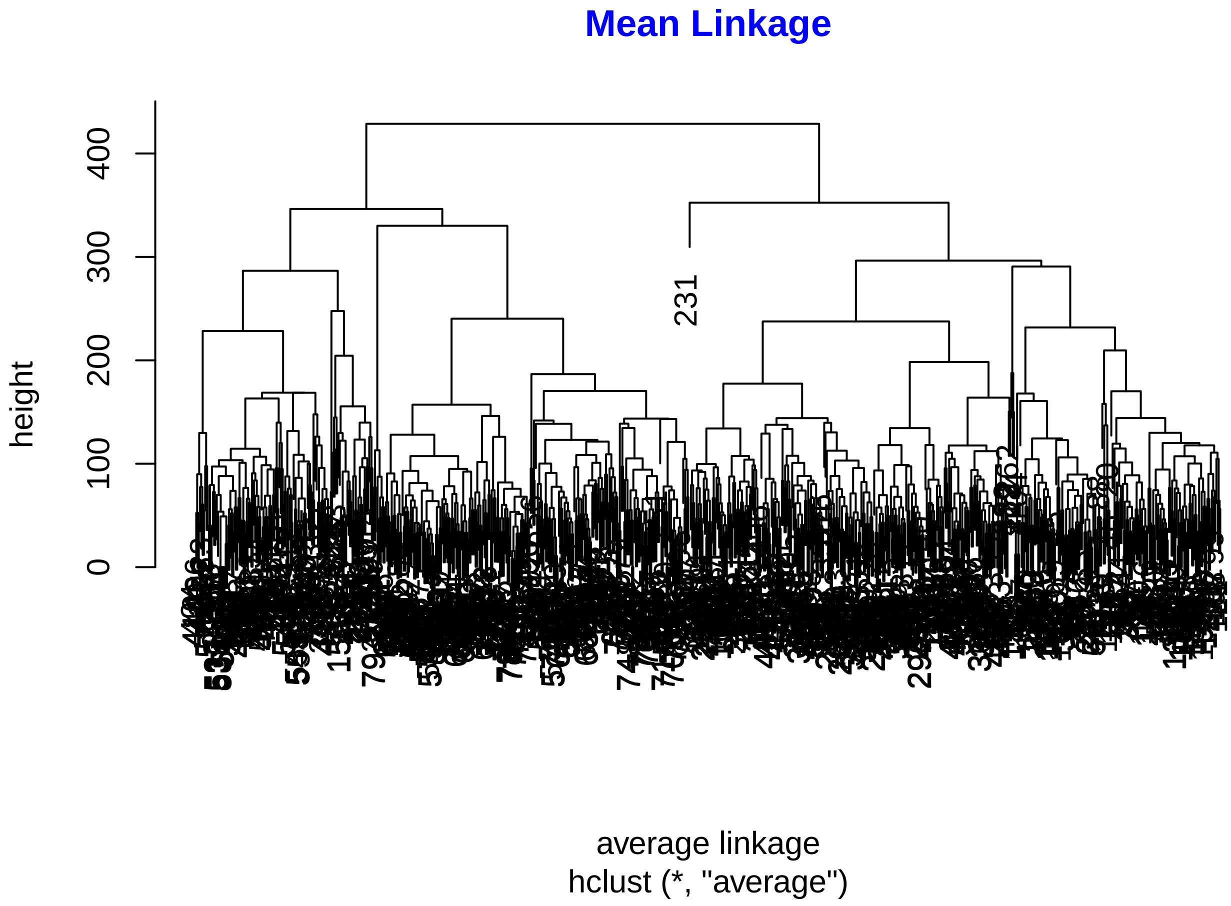
5 Clustering

Hierarchical clustering could build a hierarchy from the bottom-up, which does not require the specific number of clusters in advance. I use the mean linkage method here to generate clustering. Based on the graph, I can tell that the best choices for total number of clusters are either 3 or 4. Then I cut off the tree to verify there are 3 clusters that includes 414, 36, 350 pokemons in each clusters. More specifically, I create the table of number of observations in each cluster and type1 to see the clusters differentiate by different types of pokemons. The algorithm does not classify the pokemons into right clusters, since all types of pokemons spread out among three clusters. After I adjust the cut off line to 170 that gives me 18 clusters, the distribution still not satisfy with their types. That makes sense since pokemons cannot be classified by a single variable, there are lots of factors could affect the clustering process.

```r
# Hierarchical clustering
d <- dist(Pokemon[,-c(2,3,4,13)],method="euclidean")
```

```
## Warning in dist(Pokemon[, -c(2, 3, 4, 13)], method = "euclidean"): NAs
## introduced by coercion
```

```
fit <- hclust(d,method = "average")
plot(fit,main="Mean Linkage",col.main="Blue",xlab="average linkage",ylab="height")
```



6 Conclusion

In the project, I compared two kinds of grouping method based on different response variables, total attributes and hit points. After I did classification and clustering, I can tell both of these two have advantages and disadvantages. For the total attributes groups, they consider more factors that could affect the performance of pokemons, which can be regarded as a relatively comprehensive method. Although the random forest and decision tree provide very low test error rate for the hit points groups, they still cannot be considered as a good grouping method.

All in all, I can draw a conclusion that, based on test error rate, considering the various conditions in a battle, the total attributes groups may provide a more feasible suggestion when the players are choosing a correct pokemon. That would be the most crucial task for this project.

7 Reference

[1] Alberto Barradas. Pokemon with stats [Data file]. Retrieved from https://www.kaggle.com/abcsds/pokemon.

[2] Perry, Cheng. “Visualising Pokemon stats with ggplot”, https://www.kaggle.com/kitman0804/d/abcsds/pokemon/visualising-pokemon-stats-with-ggplot/comments. Accessed 27th Nov 2016.

[3] ElpidaM. “Decision Tree and Clustering”, https://www.kaggle.com/emadata88/d/abcsds/pokemon/decision-tree-and-clustering/code. Accessed 27th Nov 2016

[4] Nishant Bhadauria. “Pokemon Speed Attack HP Defense analysis by type” https://www.kaggle.com/nishantbhadauria/d/abcsds/pokemon/pokemon-speed-attack-hp-defense-analysis-by-type. Accessed 27th Nov 2016

[5] Cortez, Paulo, et al. “Using data mining for wine quality assessment.”Discovery Science. Springer Berlin Heidelberg, 2009.