Laddering Unite

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.1.3
```

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

```
## Warning: package 'ggplot2' was built under R version 4.1.3
```

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

```
library(scales)
```

Pokémon Unite is an online video game. Matches consist in 10 minutes where two teams, each with 5 players, must archive the highest score to win.

At the beginning of a match, players will be able to select which Pokémon they want to play. Pokémons have different base stats and are better in some aspects of the games than others. For example, someone have high defence, which allows them to play as a shield to Pokémons which have low health points, but high attack.

While battling, the player is also able to level up once they defeat enough Pokémon. Once a Pokémon is defeated, opponent or wild, the player collects Balls which can then be dispensed to the opposing team’s goal zones, and therefore collect points for the team. The player can heal by standing in their own team’s goal zone or scoring points. The winner is decided by the higher final score of both teams once the battle timer is over.

Datasets

Two kind of data source are used.

Pokémon Features

The first dataset carries the features of each Pokémon. Raw data are stored by Pokémon level, let’s merge them:

Data Ingestion

```
read_stats = function(){
  df = data.frame()
  for(i in 1:15){
    uri = sprintf("datasets/stats/%s.csv", i)
    dfi = read.csv(uri)
    dfi$level = i
    df = rbind(df, dfi)
  }
  return(df)
}
```

```
df = read_stats()
```

Names are added for visualization purpose:

```
df_rn = paste(
  toupper(substr(df$name, 1,4)),
  df$level,
  sep="")

row.names(df) = df_rn
df$name = NULL
```

```
df$speed = NULL
```

Each Pokémon is described by 8 features:

- Health Points (HP)
- Attack
- Defense
- Special Attack
- Special Defense
- Critical Rate
- Cooldown Reduction Percentage
- Life Steal Percentage

All of them are positive scalar, let's normalize them:

```
for(i in 1:length(df)){
  df[i] = ((df[i] - min(df[i])) /(max(df[i])-min(df[i])))
}

df = df[df$level > (13/15),]
df$level = NULL
```

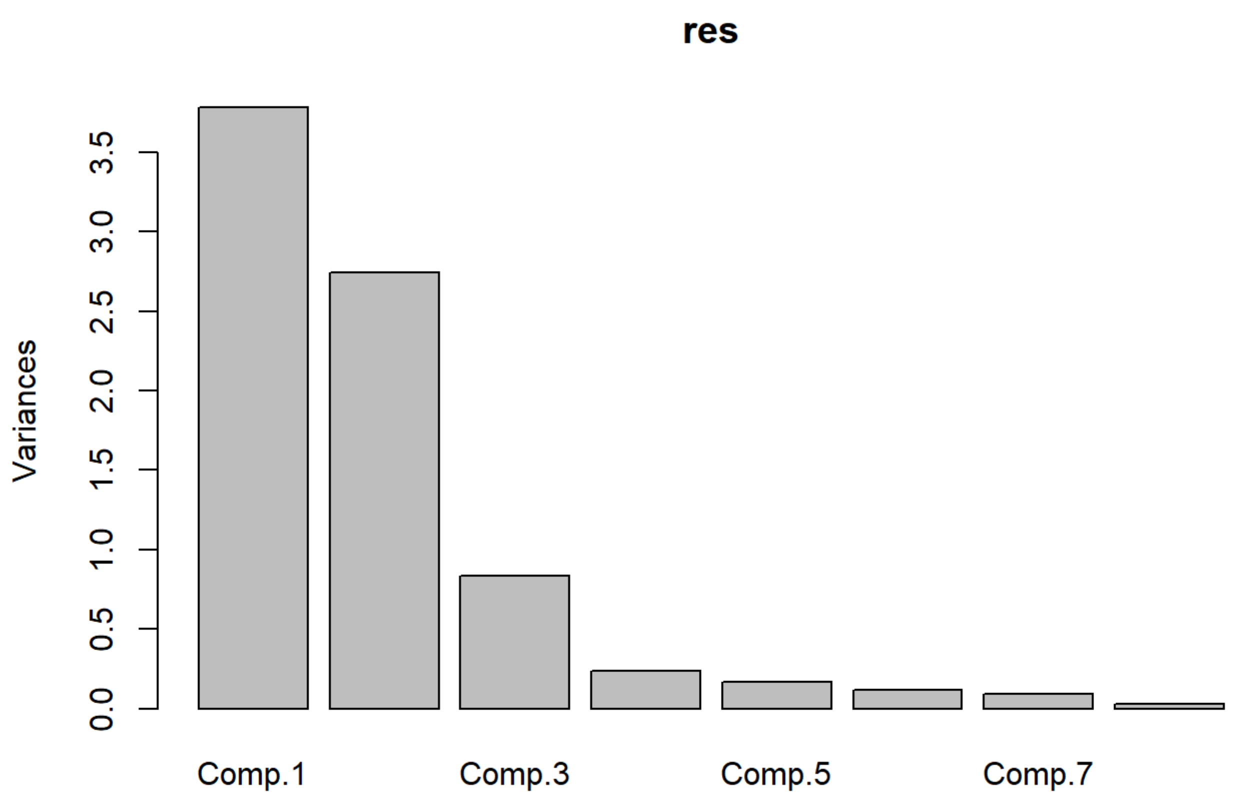
Dimensionality Reduction

The features of Pokémons seem to be a bit redundant: does the difference between `defense` and `sp_defense` really matter? Let's reduce dimensions with PCA:

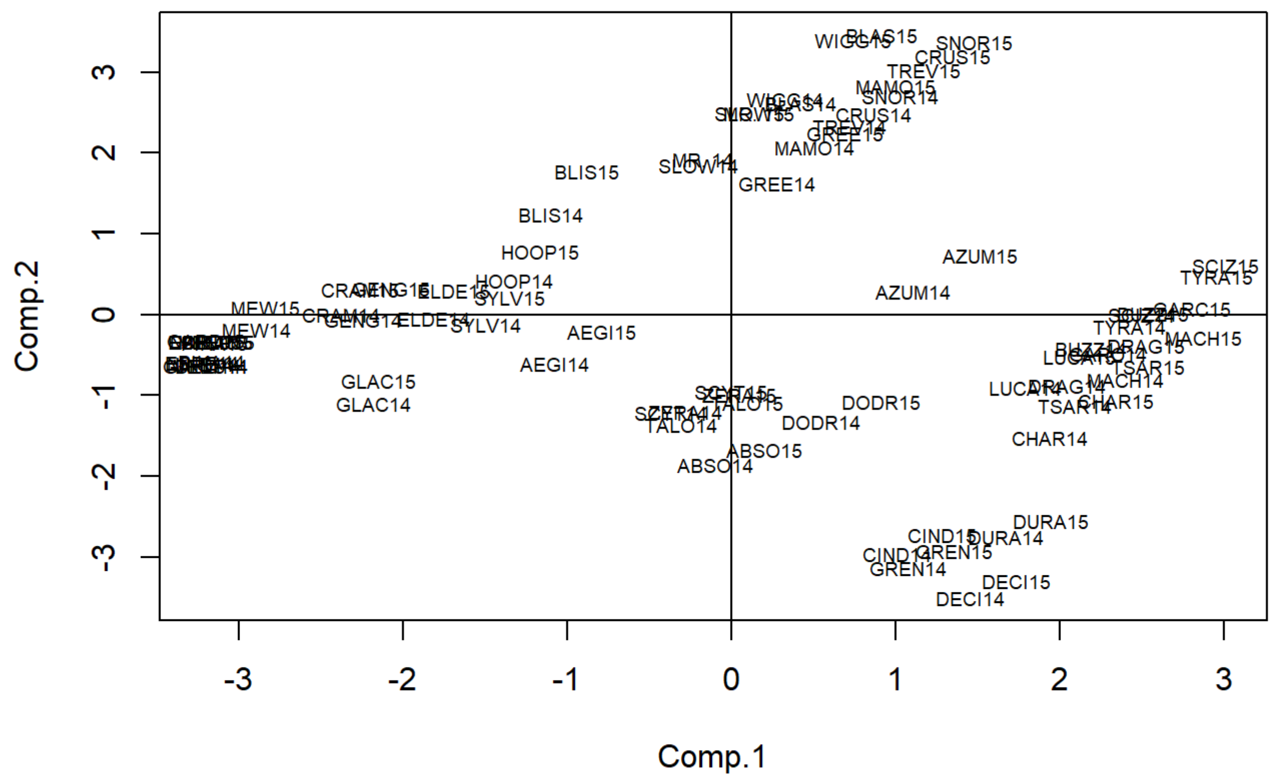
```
res = princomp(df, cor=T)
summary(res)
```

```
## Importance of components:
##                           Comp.1    Comp.2    Comp.3     Comp.4     Comp.5
## Standard deviation     1.9447788 1.6564725 0.9145924 0.48579637 0.40637021
## Proportion of Variance 0.4727706 0.3429877 0.1045599 0.02949976 0.02064209
## Cumulative Proportion  0.4727706 0.8157582 0.9203182 0.94981792 0.97046001
##                            Comp.6     Comp.7      Comp.8
## Standard deviation     0.34521247 0.29903741 0.166507853
## Proportion of Variance 0.01489646 0.01117792 0.003465608
## Cumulative Proportion  0.98535647 0.99653439 1.000000000
```

```
screeplot(res)
```

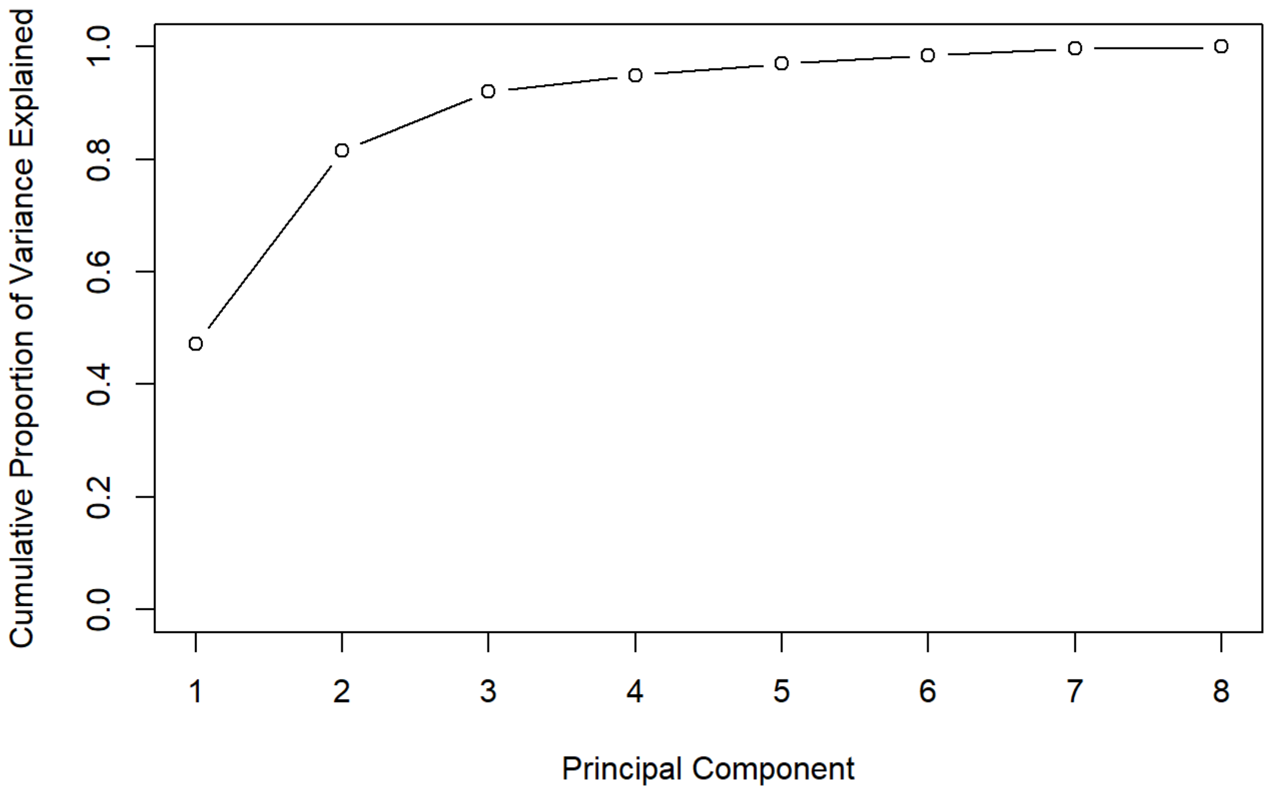


```
plot(res$scores, cex=0.0)
text(res$scores, rownames(df), cex=0.6)
abline(h=0, v=0)
```



```
pr.var=res$sdev^2
pve=pr.var/sum(pr.var)

plot(cumsum(pve), xlab="Principal Component", ylab="Cumulative Proportion of Variance Explain
ed", ylim=c(0,1),type='b')
```



The first two principal components are enough to explain the 80% of the variability among the Pokemon stats. Let’s find out their composition to give an interpretation to these new dimentions:

```
res$loading[,1:2]
```

```
##                   Comp.1      Comp.2
## hp             0.1476821  0.55601887
## attack         0.2797591 -0.25474867
## defense        0.3728057  0.38087074
## sp_attack     -0.4583060  0.04224364
## sp_defense     0.3311075  0.44716195
## crit_rate      0.3259758 -0.40775085
## cooldown_reduc -0.4495800  0.06297616
## life_steal     0.3645296 -0.33002806
```

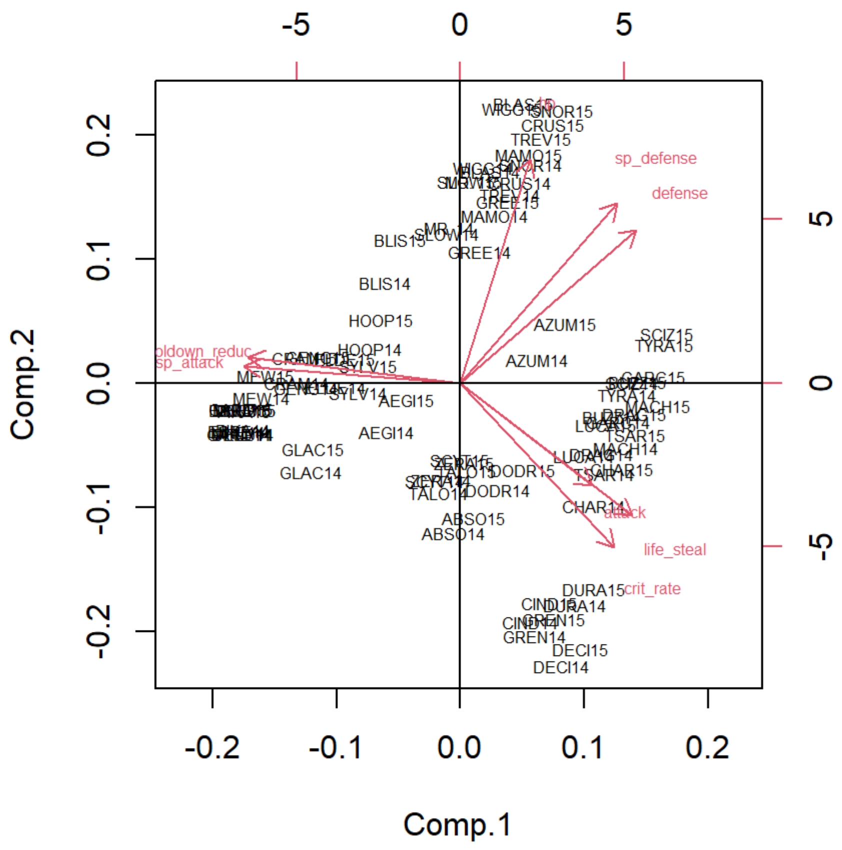
It seems that the first component measure the **adverseness** to be a core Special Attacker: very low `sp_attack` and `cooldown_reduc` .

The second component seems to measure the **defensiveness**: low `attack` , `crit_rate` and `life_stea` l, against high `defense` , `sp_defence` and `hp` .

Clustering

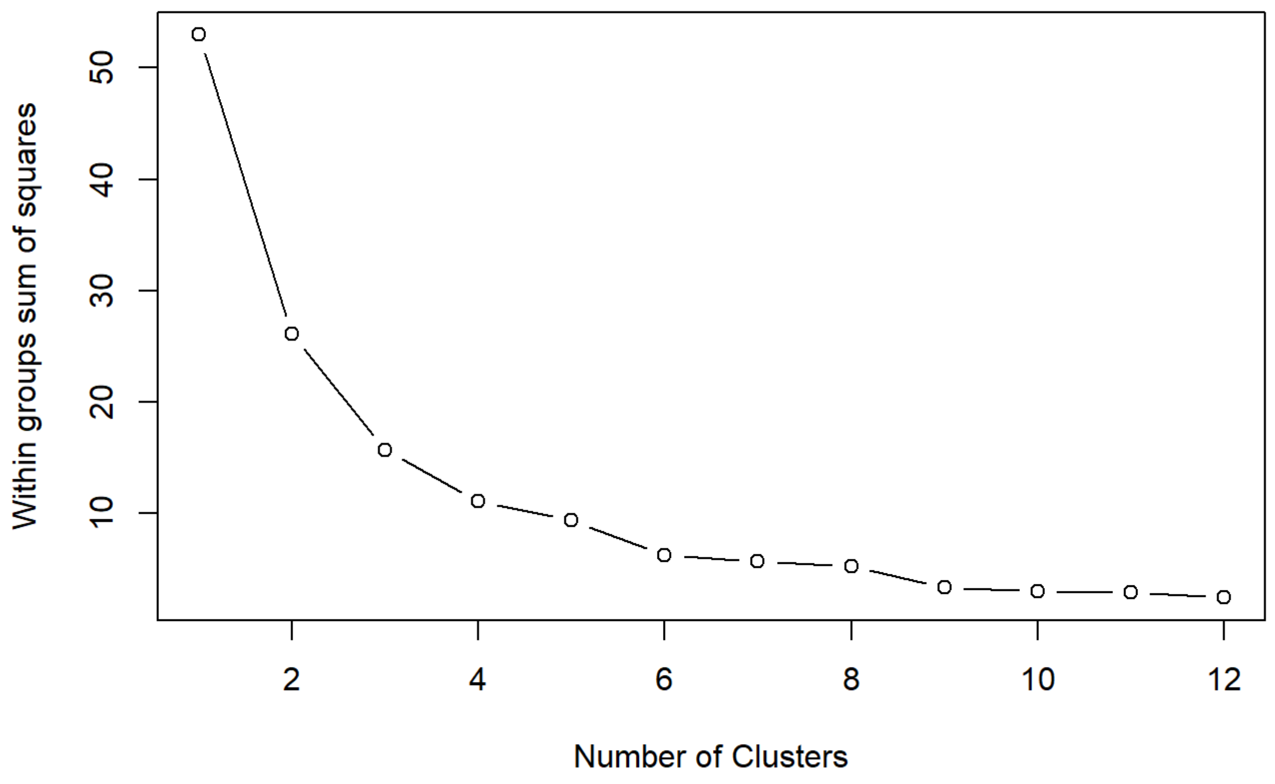
However also the number of possible playable Pokemon is quite high. Let’s apply K means to delineate groups of interchangeable Pokémon.

```
biplot(res, cex=0.5)
abline(h=0, v=0)
```



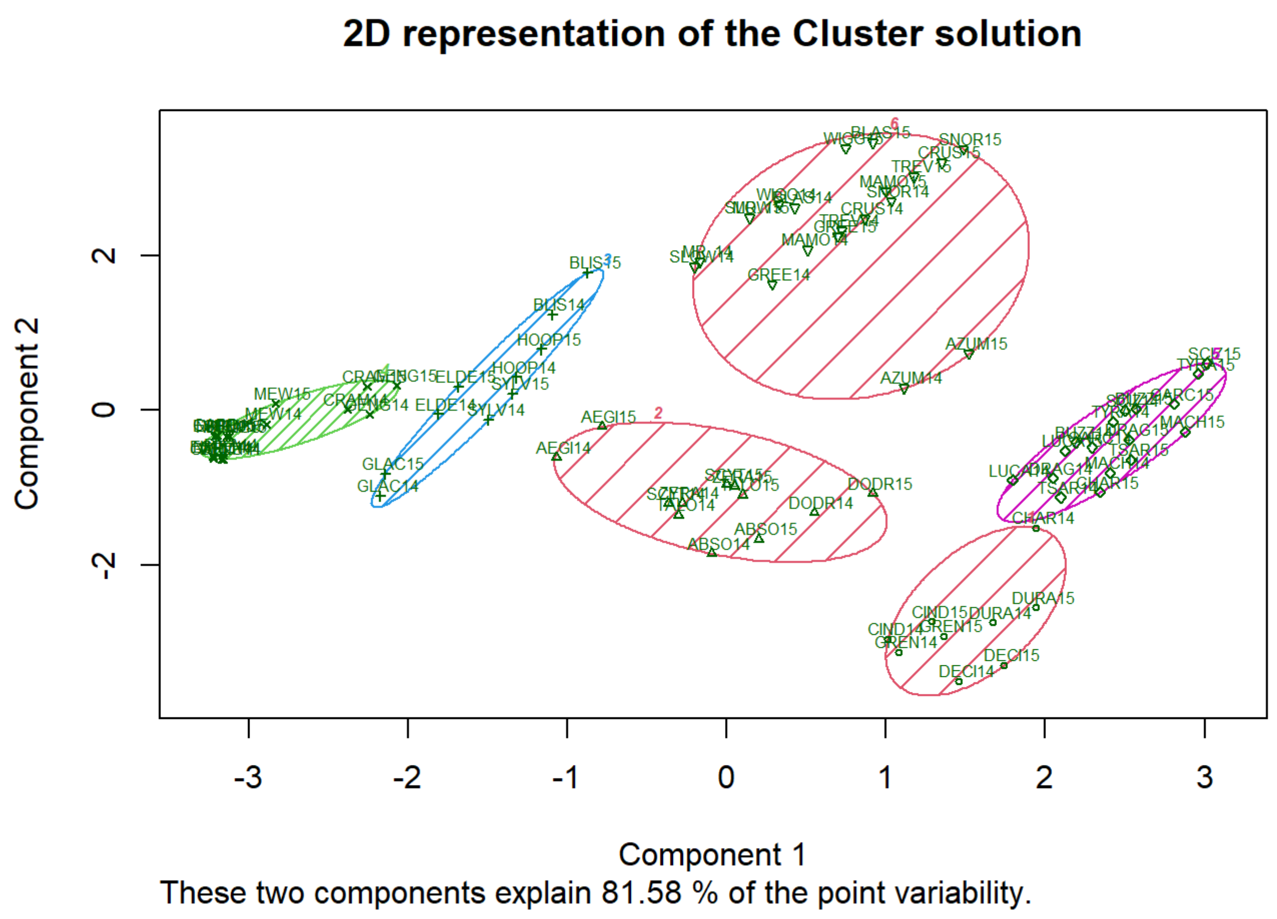
```
wssplot <- function(data, nc=15, seed=1234){
  wss = (nrow(data)-1)*sum(apply(data,2,var))
  for (i in 2:nc){
    set.seed(seed)
    wss[i] <- sum(kmeans(data, centers=i)$withinss)}
  plot(1:nc, wss, type="b", xlab="Number of Clusters",
       ylab="Within groups sum of squares")}

wssplot(df[2:length(df)], nc=12)
```



```
library(cluster)

clusplot(df, kmeans(df, centers=6)$cluster,
         main='2D representation of the Cluster solution',
         color=TRUE, shade=TRUE,
         labels=2, lines=0, cex=0.5)
```



We'll cheat a bit: a better clustering is found using **DB SCAN**

```r
# Compute DBSCAN using fpc package
library("fpc")
```

```
## Warning: package 'fpc' was built under R version 4.1.3
```

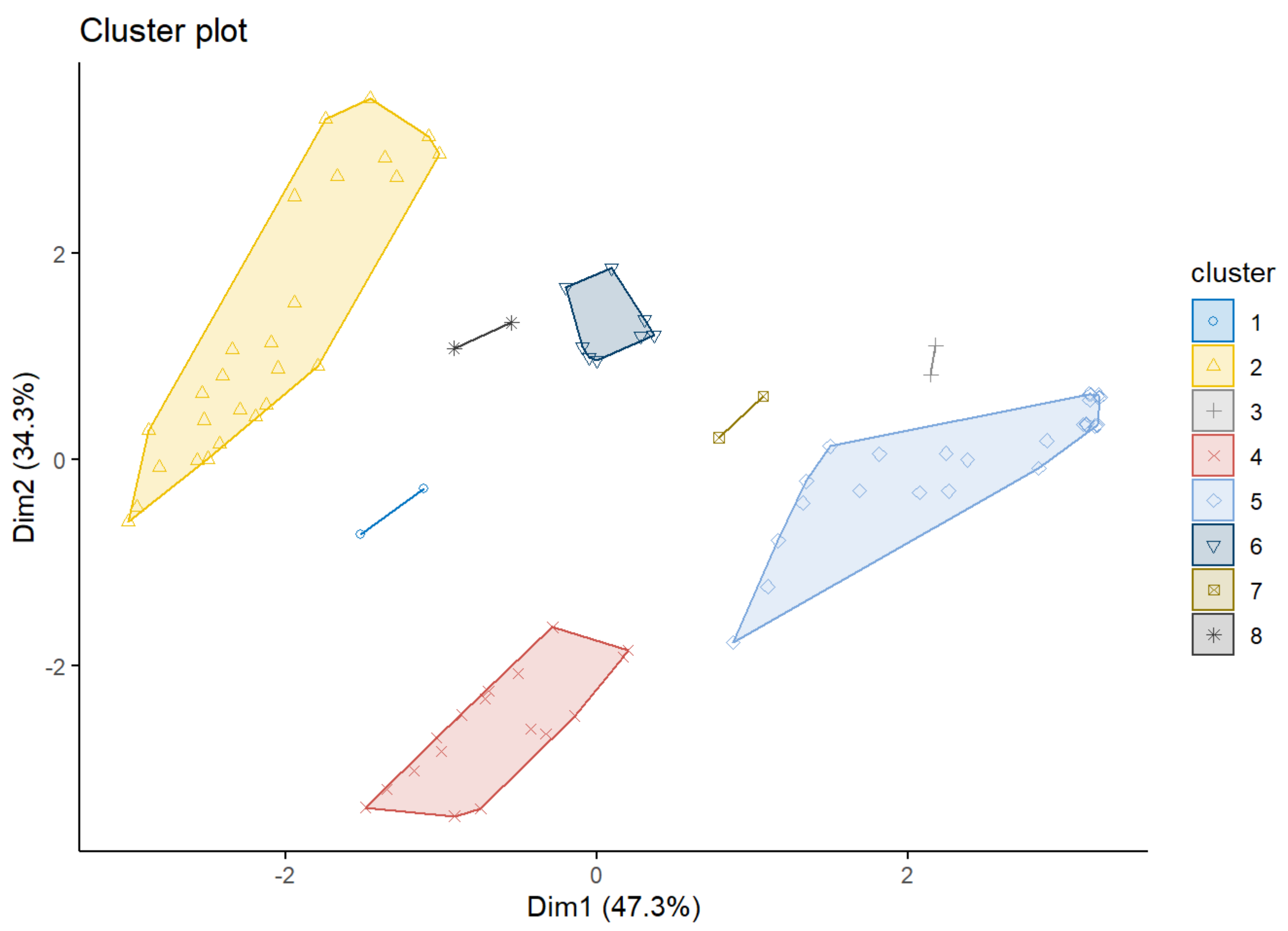
```r
db = fpc::dbscan(res$scores, eps = 1.6, MinPts = 2)

# Plot DBSCAN results
library("factoextra")
```

```
## Warning: package 'factoextra' was built under R version 4.1.3
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```r
fviz_cluster(db, data = res$scores, stand = FALSE,
ellipse = TRUE, show.clust.cent = FALSE,
geom = "point",palette = "jco", ggtheme = theme_classic())
```



```
# clookup = data.frame()
#
# for(c in 1:length(df)){
#   clookup = rbind(clookup, c(c, row.names(df)[c], db$cluster[c]))
# }
#
# colnames(clookup) = c('id','name', 'cluster')
```

```
clust_map = read.csv("datasets/clusters.csv")
```

Match Results

```
dt = read.csv('datasets/matches.csv')
```

The second dataset carries the performances for 50 matches of all team players. Relevant metrics measured are:

- Level reached by the player
- Individual points scored
- Nº of kills
- Nº of assist
- Nº of interrupt
- HP damage done
- HP damage taken
- HP self-cured

```
pkmn_count = aggregate(dt$score, by=list(pokemon=dt$pokemon), FUN=length)
pkmn_count[order(-pkmn_count$x),][1:7,]
```

```
##      pokemon  x
## 10   Crustle 44
## 29    MrMime 35
## 7  Charizard 28
## 31   Pikachu 24
## 28       Mew 23
## 17    Espeon 19
## 20    Gengar 19
```

Players do not choose Pokémon uniformly, somehow cluster delineated above will help to analyze the whole picture of a Pokémon role-play, instead of focusing on each single Pokémon.

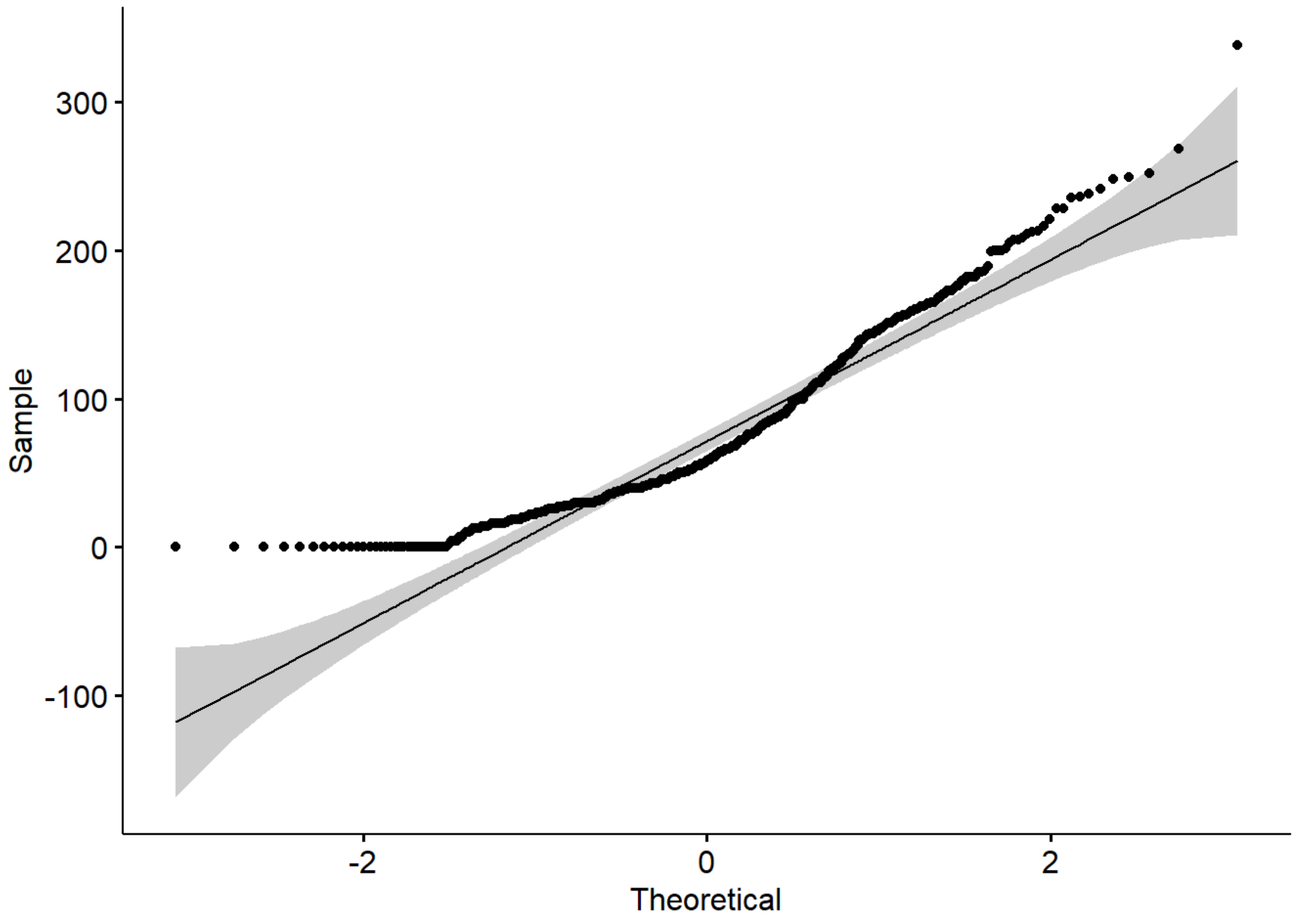
```
shapiro.test(dt$score)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  dt$score
## W = 0.91439, p-value = 3.301e-16
```

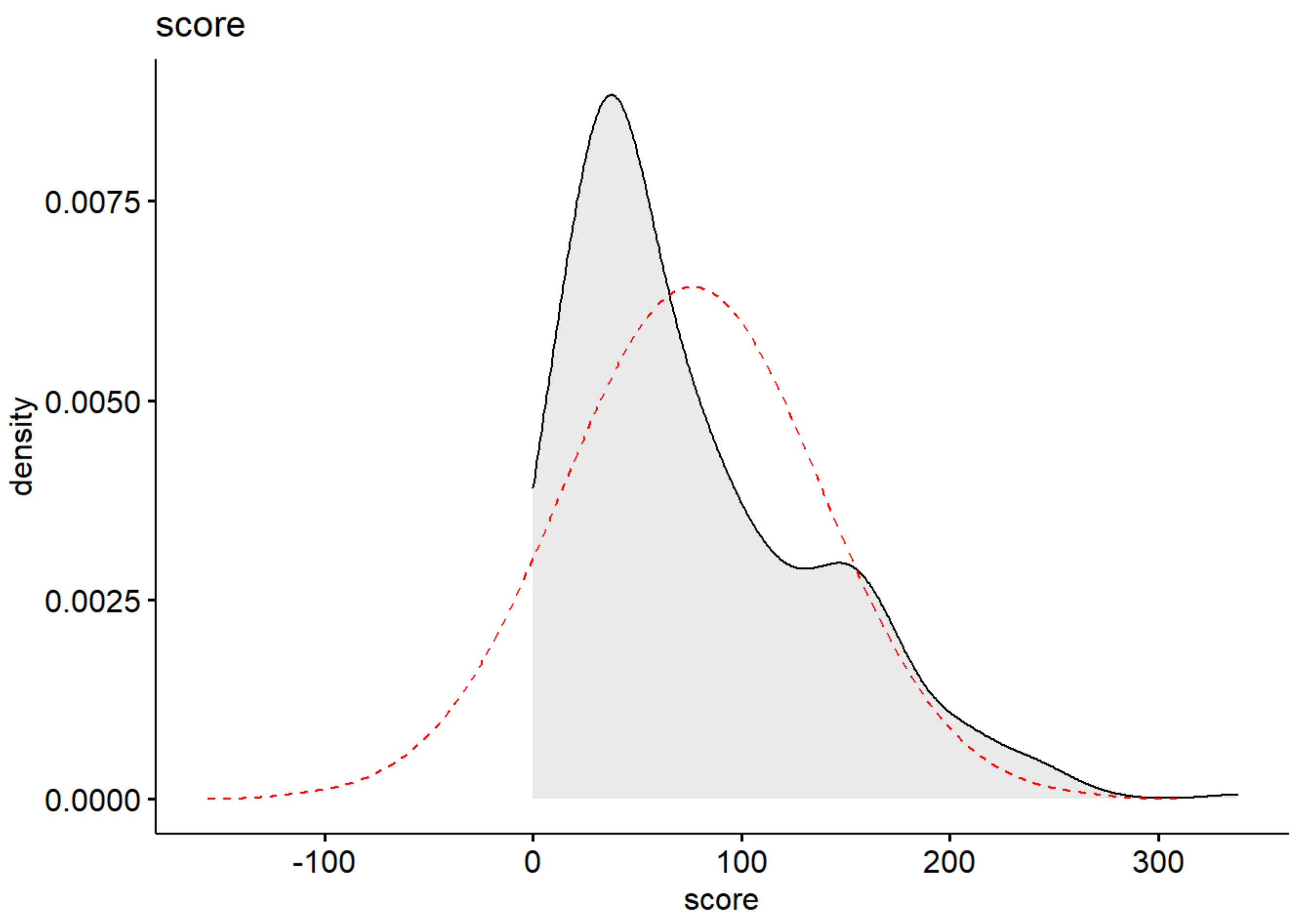
```
library(ggpubr)
```

```
## Warning: package 'ggpubr' was built under R version 4.1.3
```

```
ggqqplot(dt$score)
```



```
ggdensity(dt, x = "score", fill = "lightgray", title = "score") +
  stat_overlay_normal_density(color = "red", linetype = "dashed")
```



The `score` archived by each single player may not be a reliable dependent variable, it’s weirdly bi-modal due to game mechanics, and somewhat poissionian. Instead, the pretty binary win-lose outcome will be considered.

Optimize Gameplay - Forward Selection and Logistic Regression

Let’s see regression for two different Pokémons. Forward Selection is implemented to point which match stats are more relevant for each Pokémon.

```
library(leaps)
```

```
## Warning: package 'leaps' was built under R version 4.1.3
```

```
cols = c('level', 'score', 'kill', 'assist', 'interrupt', 'damage_done', 'damage_taken', 'dam
age_healed')
```

> *Why Y and Log X? Interpretation purpose:*
>
> ***LINEAR**: A 1% increase in X would lead to a β% increase/decrease in Y*
>
> ***LOGIT**: A k-factor increase in X would lead to a k**β increase in odds.*
>
> *https://stats.stackexchange.com/questions/8318/interpretation-of-log-transformed-predictors-in-logistic-regression* (https://stats.stackexchange.com/questions/8318/interpretation-of-log-transformed-predictors-in-logistic-regression)

Crustle

```
who = "Crustle"

dtw = dt[dt$pokemon == who, ]
dtw[,cols] = log(dtw[,cols]+1)

regfit.fwd = regsubsets(win ~ level + score + kill + assist + interrupt + damage_done + damag
e_taken + damage_healed, data=dtw, method="forward")

summary(regfit.fwd)
```

```
## Subset selection object
## Call: regsubsets.formula(win ~ level + score + kill + assist + interrupt +
##     damage_done + damage_taken + damage_healed, data = dtw, method = "forward")
## 8 Variables  (and intercept)
##               Forced in Forced out
## level             FALSE      FALSE
## score             FALSE      FALSE
## kill              FALSE      FALSE
## assist            FALSE      FALSE
## interrupt         FALSE      FALSE
## damage_done       FALSE      FALSE
## damage_taken      FALSE      FALSE
## damage_healed     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: forward
##          level score kill assist interrupt damage_done damage_taken
## 1  ( 1 ) " "   " "   " "  "*"    " "       " "         " "
## 2  ( 1 ) "*"   " "   " "  "*"    " "       " "         " "
## 3  ( 1 ) "*"   " "   " "  "*"    " "       " "         " "
## 4  ( 1 ) "*"   " "   " "  "*"    " "       " "         "*"
## 5  ( 1 ) "*"   " "   "*"  "*"    " "       " "         "*"
## 6  ( 1 ) "*"   " "   "*"  "*"    " "       "*"         "*"
## 7  ( 1 ) "*"   "*"   "*"  "*"    " "       "*"         "*"
## 8  ( 1 ) "*"   "*"   "*"  "*"    "*"       "*"         "*"
##          damage_healed
## 1  ( 1 ) " "
## 2  ( 1 ) " "
## 3  ( 1 ) "*"
## 4  ( 1 ) "*"
## 5  ( 1 ) "*"
## 6  ( 1 ) "*"
## 7  ( 1 ) "*"
## 8  ( 1 ) "*"
```

```
glm.fit <- glm(win ~ assist + level , data = dtw, family=binomial(link='logit'))
summary(glm.fit)
```

```
## 
## Call:
## glm(formula = win ~ assist + level, family = binomial(link = "logit"), 
##     data = dtw)
## 
## Deviance Residuals: 
##     Min       1Q   Median       3Q      Max  
## -1.7461  -0.8556   0.2972   0.8821   1.7616  
## 
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)  
## (Intercept)  -34.787     13.786  -2.523   0.0116 *
## assist         2.939      1.176   2.498   0.0125 *
## level         11.206      4.974   2.253   0.0243 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
##     Null deviance: 59.534  on 43  degrees of freedom
## Residual deviance: 45.151  on 41  degrees of freedom
## AIC: 51.151
## 
## Number of Fisher Scoring iterations: 5
```

Crustle, in order to raise its win odds, have, first of all, to keep its level high and secondly to assist its teammates during fights.

Pikachu

```
who = "Pikachu"

dtw = dt[dt$pokemon == who, ]
dtw[,cols] = log(dtw[,cols]+1)

regfit.fwd = regsubsets(win ~ level + score + kill + assist + interrupt + damage_done + damag
e_taken + damage_healed, data=dtw, method="forward")

summary(regfit.fwd)
```

```
## Subset selection object
## Call: regsubsets.formula(win ~ level + score + kill + assist + interrupt +
##     damage_done + damage_taken + damage_healed, data = dtw, method = "forward")
## 8 Variables  (and intercept)
##               Forced in Forced out
## level             FALSE      FALSE
## score             FALSE      FALSE
## kill              FALSE      FALSE
## assist            FALSE      FALSE
## interrupt         FALSE      FALSE
## damage_done       FALSE      FALSE
## damage_taken      FALSE      FALSE
## damage_healed     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: forward
##          level score kill assist interrupt damage_done damage_taken
## 1  ( 1 ) " "   " "   " "  " "    " "       " "         "*"
## 2  ( 1 ) " "   "*"   " "  " "    " "       " "         "*"
## 3  ( 1 ) " "   "*"   " "  "*"    " "       " "         "*"
## 4  ( 1 ) " "   "*"   " "  "*"    "*"       " "         "*"
## 5  ( 1 ) "*"   "*"   " "  "*"    "*"       " "         "*"
## 6  ( 1 ) "*"   "*"   " "  "*"    "*"       " "         "*"
## 7  ( 1 ) "*"   "*"   "*"  "*"    "*"       " "         "*"
## 8  ( 1 ) "*"   "*"   "*"  "*"    "*"       "*"         "*"
##          damage_healed
## 1  ( 1 ) " "
## 2  ( 1 ) " "
## 3  ( 1 ) " "
## 4  ( 1 ) " "
## 5  ( 1 ) " "
## 6  ( 1 ) "*"
## 7  ( 1 ) "*"
## 8  ( 1 ) "*"
```

```
glm.fit <- glm(win ~ damage_taken + score , data = dtw, family=binomial(link='logit'))
summary(glm.fit)
```

```
##
## Call:
## glm(formula = win ~ damage_taken + score, family = binomial(link = "logit"),
##     data = dtw)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.95316  -0.65115  -0.09791   0.87001   1.51511
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   89.4655    48.5409   1.843   0.0653 .
## damage_taken  -8.7857     4.6469  -1.891   0.0587 .
## score          1.0993     0.5533   1.987   0.0469 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 33.104  on 23  degrees of freedom
## Residual deviance: 20.938  on 21  degrees of freedom
## AIC: 26.938
##
## Number of Fisher Scoring iterations: 6
```

On the other hand, Pikachu have to avoid damage from opponents, and prioritize scoring.

Team synergy

Winning it’s not just a matter of single players behavior, but also of compatibility between Pokemons. Most of the time a team with balanced roles and stats is the main key for the victory.

```
dp = read.csv("datasets/pivot.group.matches.csv")
```

Match results have been pivoted, the metrics that have been kept for each team are:

- The average performance achieved
- The count of Pokemon that were played for each role-cluster

Let’s move a bit out of linearity:

```
# for( c in names(dp)[55:60]){
#   dp[paste0(c,'_2')] = as.integer(dp[c] > 1)
#   dp[c] = as.integer(dp[c] == 1)
# }

for( c in names(dp)[55:60]){
  dp[paste0(c,'_2')] = (dp[c]^2)[1]
}
```

A Balanced Team - Logistic Regression

To model how team roles should be balanced a logistic regression is carried out over team compositions, using the role-groups delineated in the previous clustering. For visualization purpose, label names have been assigned manually:

```
names(dp)[55:60]
```

```
## [1] "support"    "versatile"  "atk_ranged" "sp_atk"     "speedster"
## [6] "defence"
```

```
glm.fit <- glm(win ~ support + versatile + atk_ranged + sp_atk + speedster + defence
               + support_2 + versatile_2 + atk_ranged_2 + sp_atk_2 + speedster_2 + defence_2
               , data = dp, family=binomial(link='logit'))
summary(glm.fit)
```

```
##
## Call:
## glm(formula = win ~ support + versatile + atk_ranged + sp_atk +
##     speedster + defence + support_2 + versatile_2 + atk_ranged_2 +
##     sp_atk_2 + speedster_2 + defence_2, family = binomial(link = "logit"),
##     data = dp)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7978  -1.0045   0.2275   1.1031   1.6541
##
## Coefficients: (1 not defined because of singularities)
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   2.70226    5.29804   0.510   0.6100
## support       8.29985  528.47501   0.016   0.9875
## versatile    -1.87832    1.51146  -1.243   0.2140
## atk_ranged   -2.69874    1.88824  -1.429   0.1529
## sp_atk       -0.77876    1.55093  -0.502   0.6156
## speedster    -1.41344    1.47315  -0.959   0.3373
## defence            NA         NA      NA       NA
## support_2    -8.46513  528.47347  -0.016   0.9872
## versatile_2   0.70974    0.38237   1.856   0.0634 .
## atk_ranged_2  1.29400    0.82956   1.560   0.1188
## sp_atk_2      0.07928    0.29942   0.265   0.7912
## speedster_2   0.46978    0.65556   0.717   0.4736
## defence_2     0.03727    0.39650   0.094   0.9251
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 138.63  on 99  degrees of freedom
## Residual deviance: 119.37  on 88  degrees of freedom
## AIC: 143.37
##
## Number of Fisher Scoring iterations: 15
```

Sadly results are nor meaningful neither interpretable, a different approach should be used.

Some spourious regression - just for fun

May be fun to regress the winning odds of each single Pokemon, despite they won't be statistically significant.

Over all Pokémon

```
glm.fit <- glm(win ~ Absol + Aegislash + Azumarill + Blastoise + Blissey + Buzzwole + Chariza
rd + Cinderace + Cramorant + Crustle + Decidueye + Delphox + Dodrio + Dragonite + Duraludon +
Eldegoss + Espeon + Garchomp + Gardevoir + Gengar + Glaceon + Greedent + Greninja + Hoopa + L
ucario + Machamp + Mamoswine + Mew + MrMime + Ninetales + Pikachu + Scizor + Slowbro + Snorla
x + Sylveon + Talonflame + Trevenant + Tsareena + Tyranitar + Venusaur + Wigglytuff
               , data = dp, family=binomial(link='logit'))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(glm.fit)
```

```
##
## Call:
## glm(formula = win ~ Absol + Aegislash + Azumarill + Blastoise +
##     Blissey + Buzzwole + Charizard + Cinderace + Cramorant +
##     Crustle + Decidueye + Delphox + Dodrio + Dragonite + Duraludon +
##     Eldegoss + Espeon + Garchomp + Gardevoir + Gengar + Glaceon +
##     Greedent + Greninja + Hoopa + Lucario + Machamp + Mamoswine +
##     Mew + MrMime + Ninetales + Pikachu + Scizor + Slowbro + Snorlax +
##     Sylveon + Talonflame + Trevenant + Tsareena + Tyranitar +
##     Venusaur + Wigglytuff, family = binomial(link = "logit"),
##     data = dp)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.15765  -0.73826  -0.00007   0.65258   2.23185
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -3.7186     6.6783  -0.557   0.5776
## Absol         -0.4267     1.4904  -0.286   0.7747
## Aegislash     -0.3059     1.7889  -0.171   0.8642
## Azumarill      0.6711     2.0982   0.320   0.7491
## Blastoise      0.4757     2.3384   0.203   0.8388
## Blissey       -2.0624     2.3340  -0.884   0.3769
## Buzzwole      -0.2171     1.5525  -0.140   0.8888
## Charizard      1.9789     1.5265   1.296   0.1949
## Cinderace      0.7130     1.8487   0.386   0.6997
## Cramorant      3.9090     2.0634   1.894   0.0582 .
## Crustle        1.6400     1.8859   0.870   0.3845
## Decidueye     -4.2462     2.0528  -2.068   0.0386 *
## Delphox       -0.2954     1.8229  -0.162   0.8713
## Dodrio        -2.0708     1.9430  -1.066   0.2865
## Dragonite     20.6186  2443.1429   0.008   0.9933
## Duraludon      2.3831     2.6566   0.897   0.3697
## Eldegoss     -13.3765  6522.6405  -0.002   0.9984
## Espeon         1.2126     1.6183   0.749   0.4537
## Garchomp      -4.0366     2.2587  -1.787   0.0739 .
## Gardevoir     -0.2491     1.4826  -0.168   0.8666
## Gengar         0.5466     1.6378   0.334   0.7386
## Glaceon        1.2529     1.9886   0.630   0.5287
## Greedent      25.0845  2243.8556   0.011   0.9911
## Greninja      -0.8560     1.5889  -0.539   0.5901
## Hoopa        -35.3293  6965.1836  -0.005   0.9960
## Lucario        3.8192     2.4756   1.543   0.1229
## Machamp        3.0341     2.1586   1.406   0.1598
## Mamoswine      4.0264     1.9002   2.119   0.0341 *
## Mew            1.5980     1.6265   0.982   0.3259
## MrMime         2.6521     1.7042   1.556   0.1197
## Ninetales      5.5103     3.0421   1.811   0.0701 .
## Pikachu       -0.3317     1.6932  -0.196   0.8447
## Scizor         2.0729     2.1384   0.969   0.3324
## Slowbro       -0.2147     2.3393  -0.092   0.9269
## Snorlax        2.9405     1.7438   1.686   0.0918 .
## Sylveon        0.0113     1.9458   0.006   0.9954
## Talonflame     1.7263     3.7487   0.461   0.6452
## Trevenant     -0.9012     1.7182  -0.525   0.5999
## Tsareena      -1.1144     1.7644  -0.632   0.5277
## Tyranitar      2.2295     1.5617   1.428   0.1534
## Venusaur      -2.8436     1.9805  -1.436   0.1511
## Wigglytuff     1.7087     2.1694   0.788   0.4309
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 138.629  on 99  degrees of freedom
## Residual deviance:  78.108  on 58  degrees of freedom
## AIC: 162.11
##
## Number of Fisher Scoring iterations: 17
```

Over a single Pokémon

```
glm.fit <- glm(win ~ Decidueye , data = dp, family=binomial(link='logit'))
summary(glm.fit)
```

```
##
## Call:
## glm(formula = win ~ Decidueye, family = binomial(link = "logit"),
##     data = dp)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.2450  -1.2450   0.2389   1.1113   1.8465
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.1576     0.2127   0.741   0.4586
## Decidueye    -1.6617     0.8100  -2.051   0.0402 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 138.63  on 99  degrees of freedom
## Residual deviance: 133.26  on 98  degrees of freedom
## AIC: 137.26
##
## Number of Fisher Scoring iterations: 3
```

A Balanced Team - Decision Tree

Trees are more powerful, they can take in account interaction between variables and archive a sort of binary win-lose classification.

```
dp_group = dp[55:60]
dp_group$win = dp$win
```

The accuracy of tree predictions is evaluated by computing metrics like accuracy and MSE. A side dataset obtained by surveying some players is used as test set.

```
new_data =read.csv("datasets/survey.test.set.csv")
```

Tree #0 - tree rbase

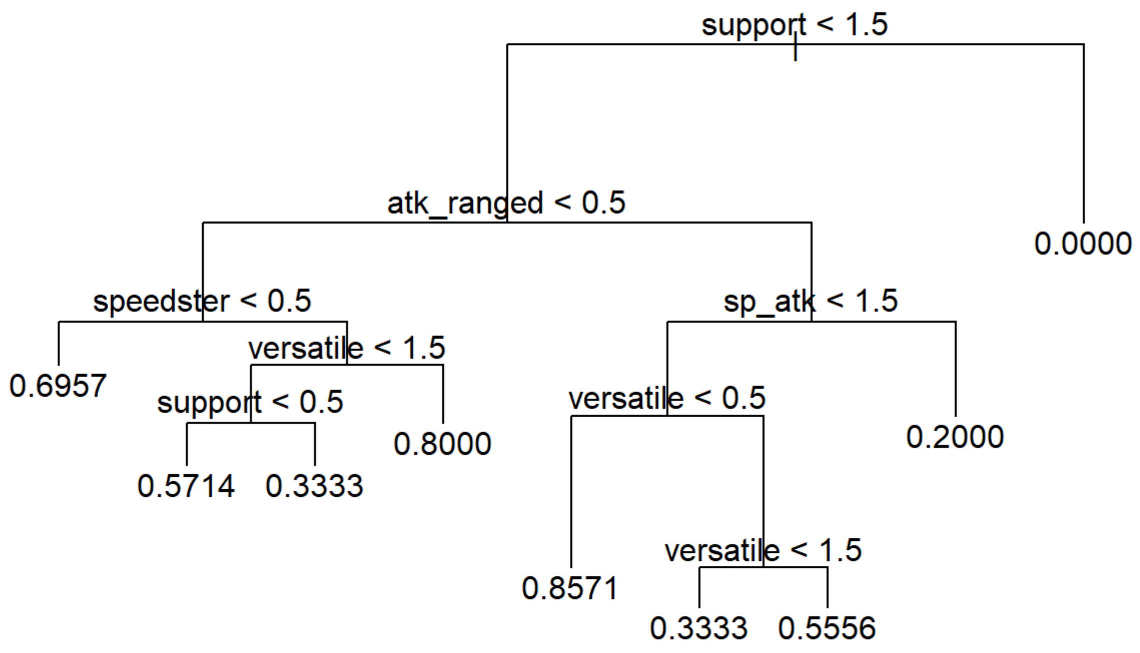
```
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.1.3
```

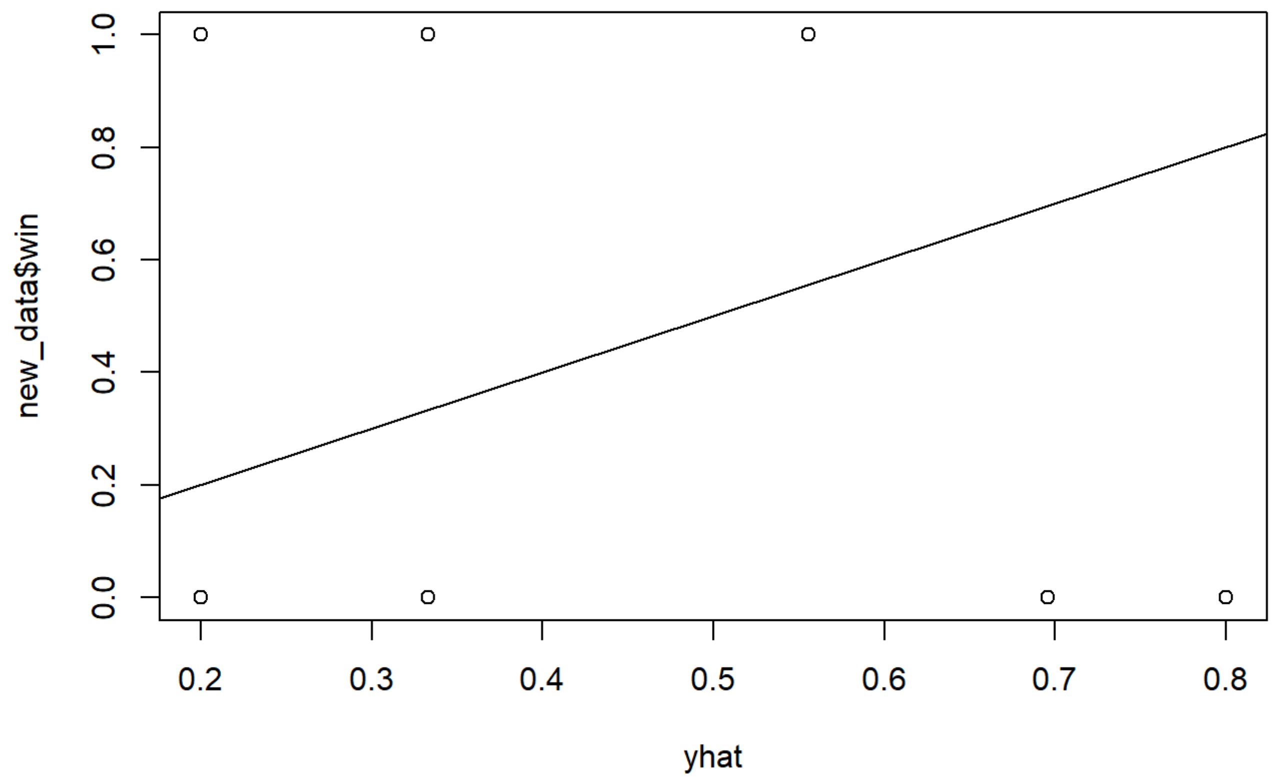
```
tree <- tree(win ~ ., data = dp_group)
summary(tree)
```

```
## 
## Regression tree:
## tree(formula = win ~ ., data = dp_group)
## Variables actually used in tree construction:
## [1] "support"    "atk_ranged" "speedster"  "versatile"  "sp_atk"
## Number of terminal nodes:  9 
## Residual mean deviance:  0.2173 = 19.78 / 91 
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
## -0.85710 -0.33330  0.07143  0.00000  0.30430  0.80000
```

```
plot(tree)
text(tree, pretty = 0)
```



```
yhat <- predict(tree, newdata = new_data)
plot(yhat, new_data$win)
abline(0, 1)
```



```
y_mse = mean((yhat - new_data$win)^2)
y_acc = sum(round(yhat, digits=0)==new_data$win)/length(yhat)
print(c(y_acc, y_mse))
```

```
## [1] 0.4000000 0.3752574
```

```
confusionMatrix(data=factor(round(yhat, digits=0)), reference = factor(new_data$win))
```

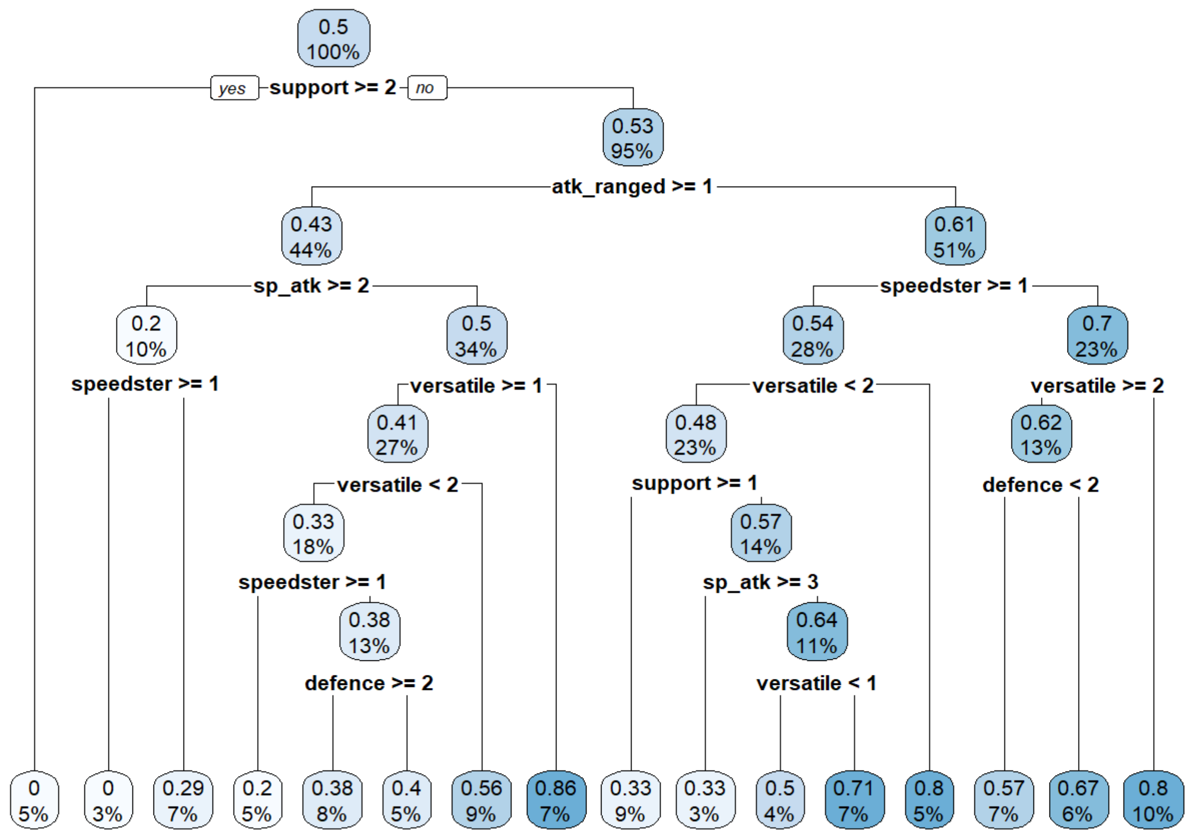
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 0 1
##          0 3 4
##          1 2 1
##
##                Accuracy : 0.4
##                  95% CI : (0.1216, 0.7376)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.8281
##
##                   Kappa : -0.2
##
##  Mcnemar's Test P-Value : 0.6831
##
##             Sensitivity : 0.6000
##             Specificity : 0.2000
##          Pos Pred Value : 0.4286
##          Neg Pred Value : 0.3333
##              Prevalence : 0.5000
##          Detection Rate : 0.3000
##    Detection Prevalence : 0.7000
##       Balanced Accuracy : 0.4000
##
##        'Positive' Class : 0
##
```

Tree #1 - rpart

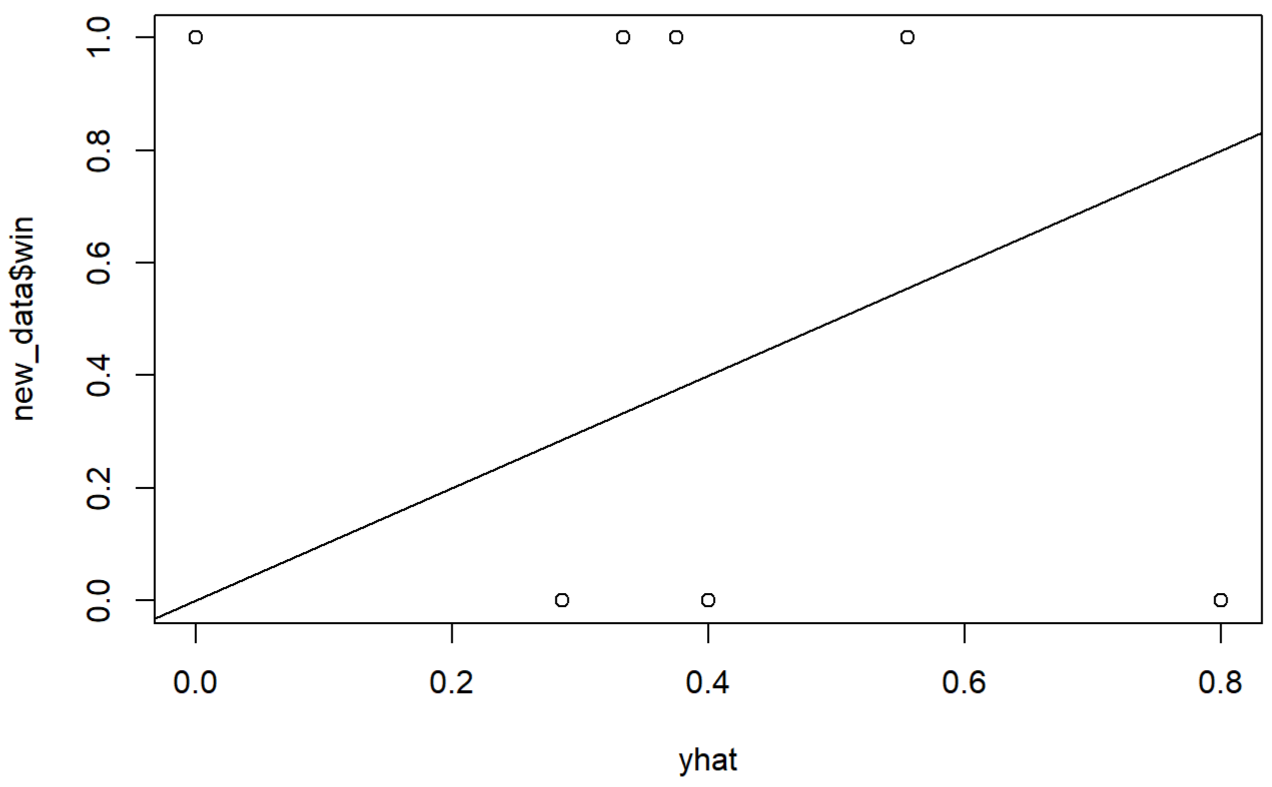
```
library("rpart")
library("rpart.plot")
tree1 <- rpart(win ~ ., data = dp_group, control = rpart.control(cp = 0, minsplit = 10, maxsu
rrogate = 10))
printcp(tree1)
```

```
##
## Regression tree:
## rpart(formula = win ~ ., data = dp_group, control = rpart.control(cp = 0,
##     minsplit = 10, maxsurrogate = 10))
##
## Variables actually used in tree construction:
## [1] atk_ranged defence    sp_atk     speedster  support    versatile
##
## Root node error: 25/100 = 0.25
##
## n= 100
##
##            CP nsplit rel error xerror      xstd
## 1  5.2632e-02      0   1.00000 1.0263 0.0070528
## 2  3.4022e-02      1   0.94737 1.0288 0.0315830
## 3  1.4963e-02      4   0.84530 1.1582 0.0801283
## 4  1.2422e-02      6   0.81537 1.3196 0.1072443
## 5  1.1852e-02      7   0.80295 1.3520 0.1097617
## 6  8.6580e-03      8   0.79110 1.3641 0.1104505
## 7  7.7057e-03      9   0.78244 1.3553 0.1133937
## 8  6.8571e-03     10   0.77474 1.3550 0.1145089
## 9  4.9231e-03     11   0.76788 1.3667 0.1152427
## 10 4.6753e-03     12   0.76296 1.3681 0.1153486
## 11 1.1722e-03     13   0.75828 1.3795 0.1153362
## 12 7.6923e-05     14   0.75711 1.3846 0.1160244
## 13 0.0000e+00     15   0.75703 1.3846 0.1160244
```

```
rpart.plot(tree1)
```



```
yhat <- predict(tree1, newdata = new_data)
plot(yhat, new_data$win)
abline(0, 1)
```



```
y_mse = mean((yhat - new_data$win)^2)
y_acc = sum(round(yhat, digits=0)==new_data$win)/length(yhat)
print(c(y_acc, y_mse))
```

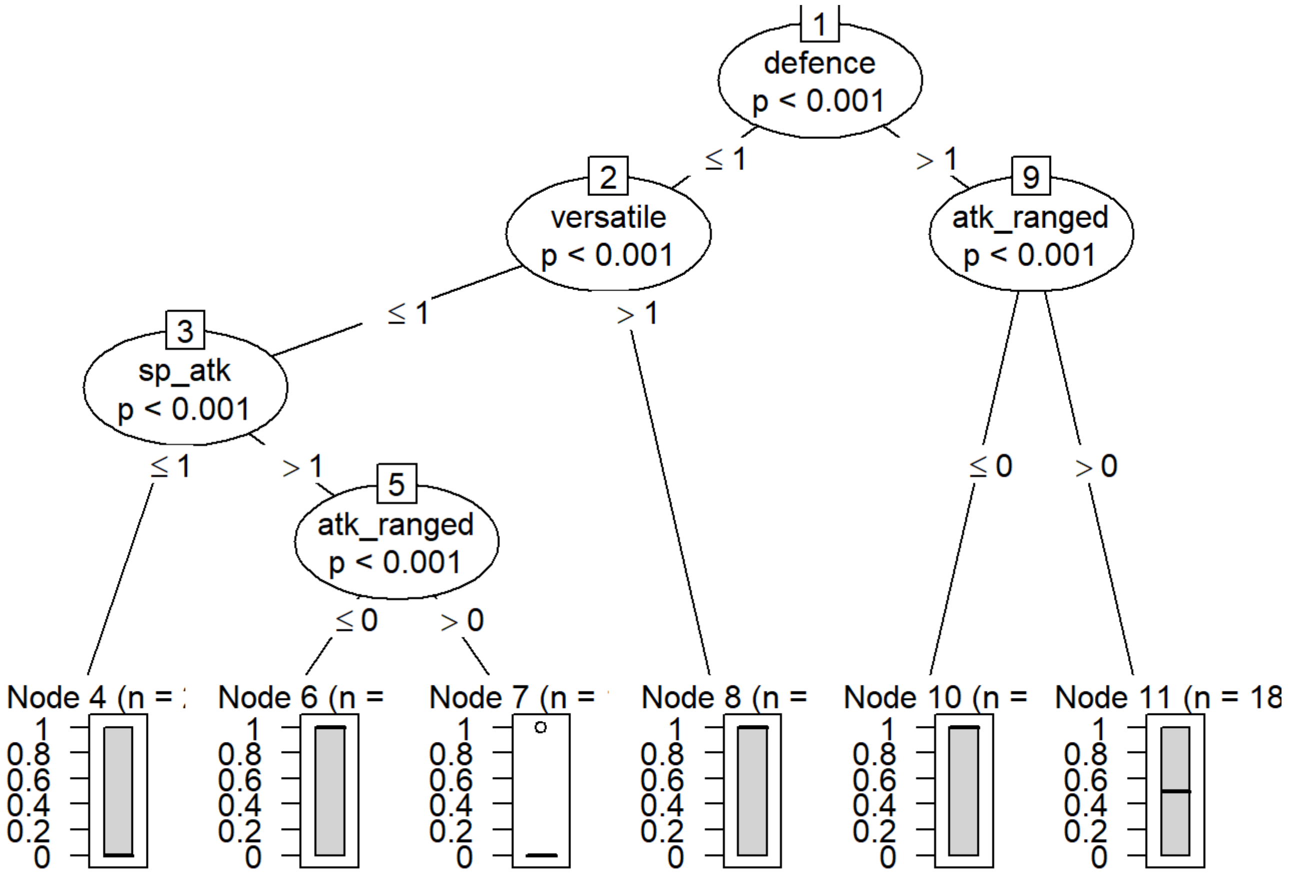
```
## [1] 0.4000000 0.4714233
```

```
confusionMatrix(data=factor(round(yhat, digits=0)), reference = factor(new_data$win))
```

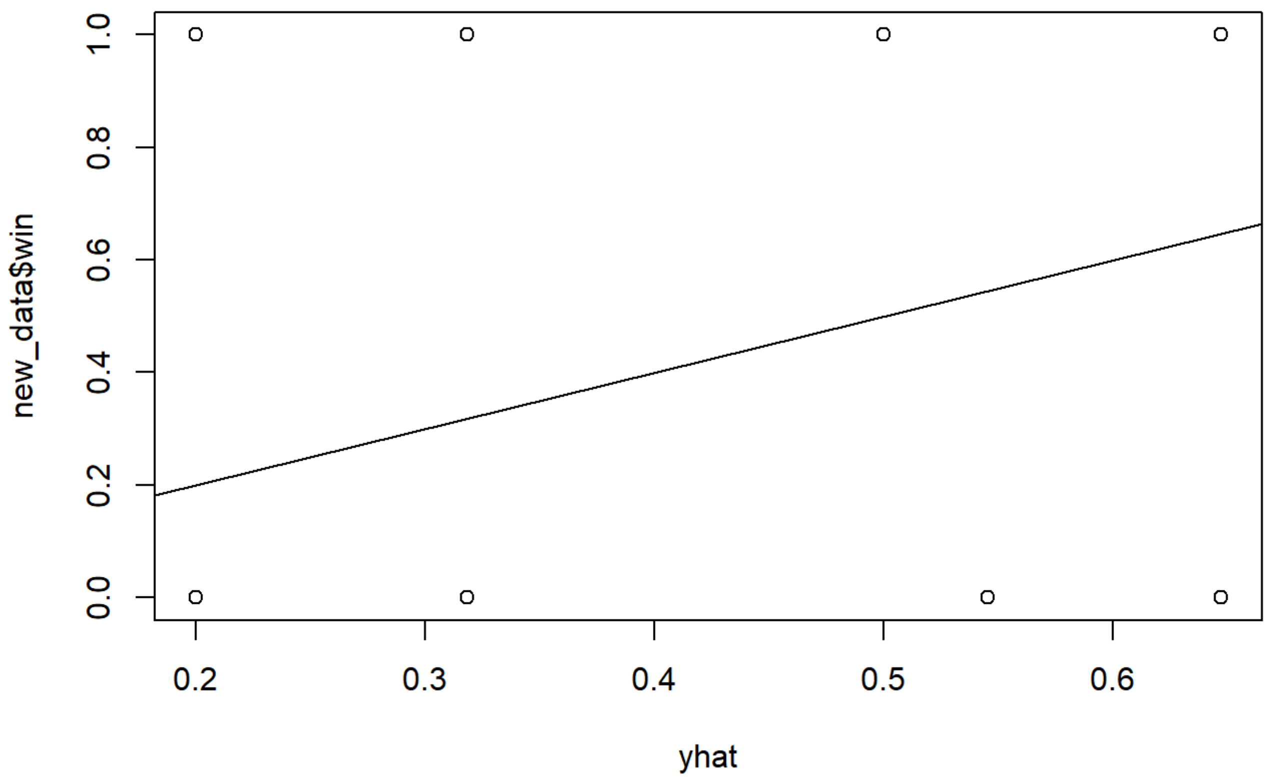
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 0 1
##          0 3 4
##          1 2 1
##
##                Accuracy : 0.4
##                  95% CI : (0.1216, 0.7376)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.8281
##
##                   Kappa : -0.2
##
##  Mcnemar's Test P-Value : 0.6831
##
##             Sensitivity : 0.6000
##             Specificity : 0.2000
##          Pos Pred Value : 0.4286
##          Neg Pred Value : 0.3333
##              Prevalence : 0.5000
##          Detection Rate : 0.3000
##    Detection Prevalence : 0.7000
##       Balanced Accuracy : 0.4000
##
##        'Positive' Class : 0
##
```

Tree #2 - partykit

```
library("partykit")
library("party")
tree2 <- ctree(win ~ ., data = dp_group, controls = ctree_control(testtype = "Teststatistic",
maxsurrogate = 1, mincriterion = 0.5, minsplit = 1))
plot(tree2, cgp = gpar(fontsize = 2))
```



```
yhat <- predict(tree2, newdata = new_data)
plot(yhat, new_data$win)
abline(0, 1)
```



```
y_mse = mean((yhat - new_data$win)^2)
y_acc = sum(round(yhat, digits=0)==new_data$win)/length(yhat)
print(c(y_acc, y_mse))
```

```
## [1] 0.4000000 0.3078129
```

```
confusionMatrix(data=factor(round(yhat, digits=0)), reference = factor(new_data$win))
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction 0 1
##          0 3 4
##          1 2 1
## 
##                Accuracy : 0.4
##                  95% CI : (0.1216, 0.7376)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.8281
## 
##                   Kappa : -0.2
## 
##  Mcnemar's Test P-Value : 0.6831
## 
##             Sensitivity : 0.6000
##             Specificity : 0.2000
##          Pos Pred Value : 0.4286
##          Neg Pred Value : 0.3333
##              Prevalence : 0.5000
##          Detection Rate : 0.3000
##    Detection Prevalence : 0.7000
##       Balanced Accuracy : 0.4000
## 
##        'Positive' Class : 0
## 
```

Consideration

The best tree (#2) archive a MSE of 0.30, and not by chances it’s the smallest tree. Smaller trees are less prone to overfit data, they return better predictions and they are way easier to interpret.

However performance are globally poor, highly due to lack of enough data that would help to build a more consistent model.