League of Legends: Multivariate Data Analysis of Professional Games

Kenny Chen

Introduction

League of Legends is an online multiplayer game where 2 teams of 5 go at it against each other by controlling various characters called "champions". The goal of the game is to destroy the Nexus located at the heart of each team's base with three lanes leading to it. An image is shown below of the map. But in order to do so, players have to level up and get gold to become stronger. They do this by killing various monsters around the map, monsters in the lanes are called minions (also known as creeps). In this game, there are various objectives such as dragons and Barons that give the team gold and experience which can boost that team to victory.

Moreover, each of the five players has a specific role. There is Top, Middle, Bottom, Jungle, and Support. Top, Middle, Bottom correspond to where they are on the map. Jungle is a position where the player roams around the map to help the laners and Support usually partners with Bottom to help them.

```
knitr::include_graphics("lol_map.jpeg", dpi = 700)
```

This game has been tremendously popular over the past decade and the esports scene is one of the largest, constantly drawing in millions of viewers for the world championships. Because so much prestige and money is on the line for professional League of Legends teams, teams are constantly analyzing their gameplay to improve.

That is why I plan to study the professional League of Legends games in 2020 in the League of Legends Championship Series (LCS) which is the North American League (Like the NBA). There are ten franchises with each team having a starting roster of 5 with substitutes. There are 2 seasons in 2020, a Spring season and a Summer season. Teams play each other twice per season and there are playoffs at the end of each season. This gives us a lot of games and various game statistics to analyze and extract meaningful information from.

We raise two questions of interest where we can use multivariate data analysis tools to explore and help teams.

Can we classify wins and losses based off of game statistics?

We are interested in performing several classification techniques to classify wins and find the best model that gives us the lowest estimated TER. By classifying wins, we hope to identify certain things that are important to winning and it can give teams an idea of what they need to prioritize.

We also want to use different clustering algorithms to cluster players based on their game statistics like how many kills, deaths, assists they have. After choosing a final clustering, we will describe the clusters we found to see if we can find groups of players and how they compare to one another in terms of playstyles and game statistics. This could help teams compare and contrast players to make decisions on roster changes.

These questions are of interest to me and many professional players and coaches because teams are always looking for ways to constantly improve and get better and using data is an often good indicator of performance.

About the Data

This dataset (Sevenhuysen 2020) was taken from Oracle's Elixir website here: [https://oracleselixir.com/tools/downloads] which is a third party website that compiles data recorded from official professional games. All the professional games are recorded with various game statistics by their respective leagues and so this website contains all the professional League of Legends games played in 2020 throughout many regions across the world. There are over 100 variables that include measures such as kills, deaths, experience earned, gold earned, and many other game statistics. Each row contains information about one player and their game statistics or about one team and the team statistics. Each match is summed about in 12 rows, 10 rows being the players (5 on each team) and two rows for either team.

This dataset contains games played in several regions (major regions include NA, EU, S. Korea, China and then minor regions like Japan or Oceania or Brazil).) This has led to there being over 67,000 observations and so I am only going to look at the LCS league which is the NA region.

Data Pre-Processing

As there is a lot of data and many variables, we decided to cut it down to a few that are important game statistics that will be described in more detail below.

```
# reading in the data
mydata <- read.csv("ChenKData.csv")
```

```
# filtering for the lcs league only
lcs_only <- mydata %>%
  filter(league == "LCS") %>%
# turning result into a factor with yes and no
  mutate(result = as.factor(case_when(result == 0 ~ "Loss",
                              result == 1 ~ "Win"))) %>%
  select(playerid, player, team, position,
         result, kills, deaths, assists,
         dragons, barons, visionscore,
         totalgold, goldat15,xpat15, csat15) %>%
# turning any other qualitative variables into factors if not already
  mutate_if(is.character,as.factor)

# filtering for teams
teams <- lcs_only %>%
  filter(playerid > 90) %>%
  select(-player, -playerid, -position)

# filtering for players
players <- lcs_only %>%
  filter(playerid < 90) %>%
  select(-playerid,-dragons, -barons, -team, -result)

# getting player stats (average kills/deaths/...)
avg_players <- players %>%
  group_by(player,position) %>%
  summarise_all(mean) %>%
  ungroup()
```

Preliminary Analysis

The original dataset contained 69852 observations about all games played in the year 2020 from all the leagues around the world. There are over 100 variables but many of them consist of NA or are symmetric (i.e. golddiffat15 (gold difference at 15 minutes) is symmetrical because if team 1 has 1500 more gold than team 2, then team 2 has 1500 less gold than team 1.) Because of this, we chose to boil down the data set into fewer variables for our two questions of interest.

1. playerid - id of player
2. player - player name
3. team - team name
4. position - player position
5. result - win/loss
6. kills - number of kills
7. deaths - number of deaths
8. assists - number of assists
9. dragons - number of dragons slain
10. barons - number of barons slain
11. visionscore - (1 point per minute of ward lifetime provided) + (1 point per minute of ward lifetime denied) (we can think of it as having "eyes" around the map, and being able to take away your opponent's "eyes".)
12. totalgold - total gold earned by player/team
13. goldat15 - gold earned at 15 minutes (game length is around 30 minutes so using a halfway marker is a good indicator of how a team/player is doing.)
14. xpat15 - experience earned at 15 minutes
15. csat15 - number of minions or creeps killed at 15 minutes

We further wrangled the data into two dataframes for our two questions of interest.

1.`teams` consists of 528 observations with natural pairings of two rows per game (so 264 games total) . One row per team with various game statistics for that team.

2.`avg_players` consists of 73 observations with each row being a player and their average statistics, so it will include variables about their average kills, deaths, cs, gold earned for all games played.

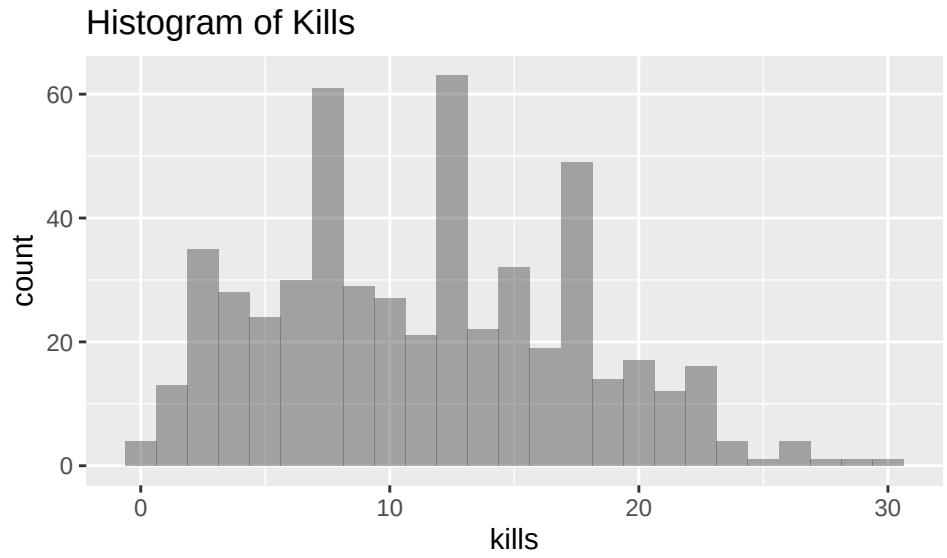
Univariate Analysis

First, let's perform some univariate analyses to identify any outliers or find any interesting observations.

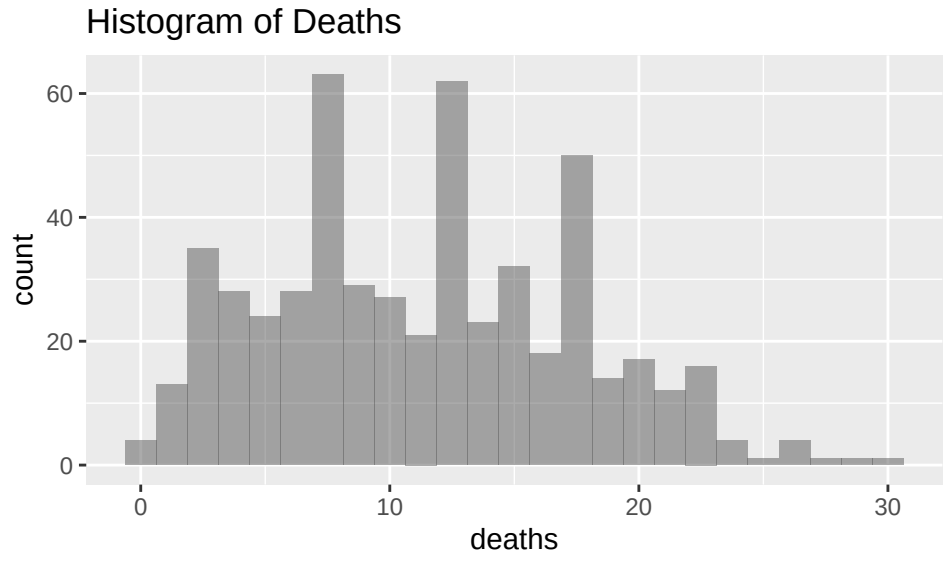
```
summary(teams)
```

```
##                team      result        kills          deaths
##  FlyQuest        : 75   Loss:264   Min.   : 0.0   Min.   : 0.00
##  Team SoloMid    : 71   Win :264   1st Qu.: 6.0   1st Qu.: 6.75
##  Evil Geniuses   : 60              Median :11.0   Median :11.00
##  Cloud9          : 57              Mean   :11.4   Mean   :11.41
##  Golden Guardians: 51              3rd Qu.:16.0   3rd Qu.:16.00
##  100 Thieves     : 49              Max.   :30.0   Max.   :30.00
##  (Other)         :165
##     assists         dragons          barons       visionscore    totalgold
##  Min.   : 0.0   Min.   :0.00   Min.   :0.000   Min.   :102   Min.   :31926
##  1st Qu.:14.0   1st Qu.:1.00   1st Qu.:0.000   1st Qu.:199   1st Qu.:50814
##  Median :26.0   Median :3.00   Median :1.000   Median :246   Median :58428
##  Mean   :27.3   Mean   :2.52   Mean   :0.688   Mean   :257   Mean   :59386
##  3rd Qu.:38.2   3rd Qu.:4.00   3rd Qu.:1.000   3rd Qu.:302   3rd Qu.:66529
##  Max.   :86.0   Max.   :7.00   Max.   :3.000   Max.   :578   Max.   :96558
##
##     goldat15         xpat15          csat15
##  Min.   :20242   Min.   :24397   Min.   :408
##  1st Qu.:22972   1st Qu.:28146   1st Qu.:489
##  Median :24122   Median :29108   Median :508
##  Mean   :24182   Mean   :29062   Mean   :509
##  3rd Qu.:25182   3rd Qu.:30009   3rd Qu.:529
##  Max.   :29792   Max.   :33530   Max.   :594
##
```

```
# sapply(teams[,3:12], hist)
gf_histogram(~kills, data = teams, title = "Histogram of Kills")
```




```
gf_histogram(~deaths, data = teams, title = "Histogram of Deaths")
```



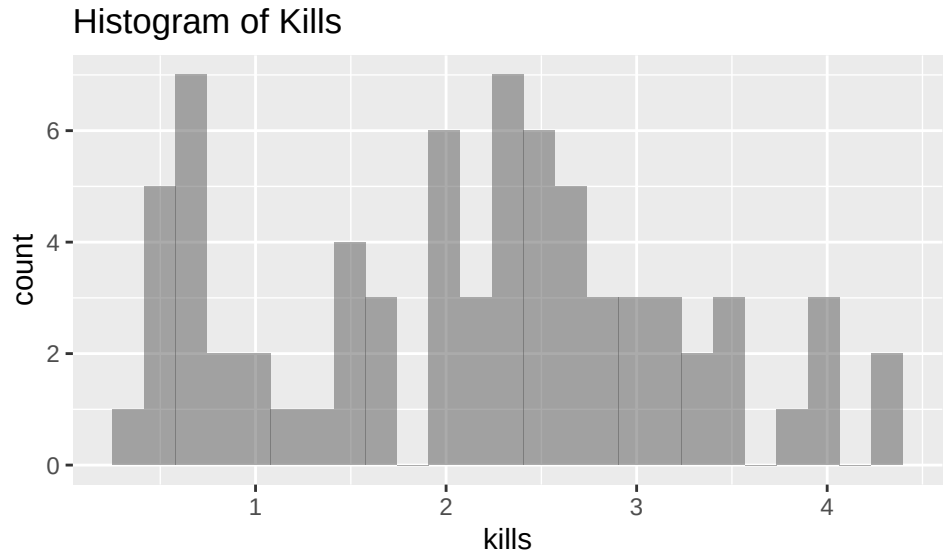
Looking at the five stat summaries of the variables, we see that they are on completely different scales. If we want to apply some analysis that uses some sort of distance, we should be careful to run it on the correlation matrix as opposed to the covariance or scale it. Looking at the various histograms (not all shown but code is available), we see most are normal, except kills and deaths which is slightly skewed to the right. This might be an issue if some of our analysis requires multivariate normality but since it's only slightly skewed, we will note it and move on.

Let's also quickly look at player statistics.

```
summary(avg_players)
```

```
##       player   position      kills           deaths        assists
##  Akaadian: 1   bot :14   Min.   :0.333   Min.   :1.21   Min.   :1.67
##  Allorim : 1   jng :16   1st Qu.:1.250   1st Qu.:1.93   1st Qu.:4.17
##  Altec   : 1   mid :15   Median :2.261   Median :2.36   Median :4.83
##  aphromoo: 1   sup :14   Mean   :2.128   Mean   :2.46   Mean   :5.03
##  Apollo  : 1   team: 0   3rd Qu.:2.750   3rd Qu.:2.82   3rd Qu.:5.76
##  Bang    : 1   top :14   Max.   :4.316   Max.   :5.33   Max.   :9.49
##  (Other) :67
##   visionscore      totalgold        goldat15        xpat15         csat15
##  Min.   : 24.0   Min.   : 6279   Min.   :3084   Min.   :3130   Min.   : 16.8
##  1st Qu.: 37.1   1st Qu.:10078   1st Qu.:4629   1st Qu.:4908   1st Qu.: 86.2
##  Median : 42.2   Median :12216   Median :5008   Median :5514   Median :119.7
##  Mean   : 50.2   Mean   :11737   Mean   :4780   Mean   :5773   Mean   :101.1
##  3rd Qu.: 52.8   3rd Qu.:13809   3rd Qu.:5323   3rd Qu.:7246   3rd Qu.:136.6
##  Max.   :105.2   Max.   :16822   Max.   :5799   Max.   :7681   Max.   :148.9
##
```

```
avg_int_only <- avg_players %>%
  select(-player, -position)
# sapply(avg_int_only, hist)
gf_histogram(~kills, data = avg_players,title = "Histogram of Kills" )
```



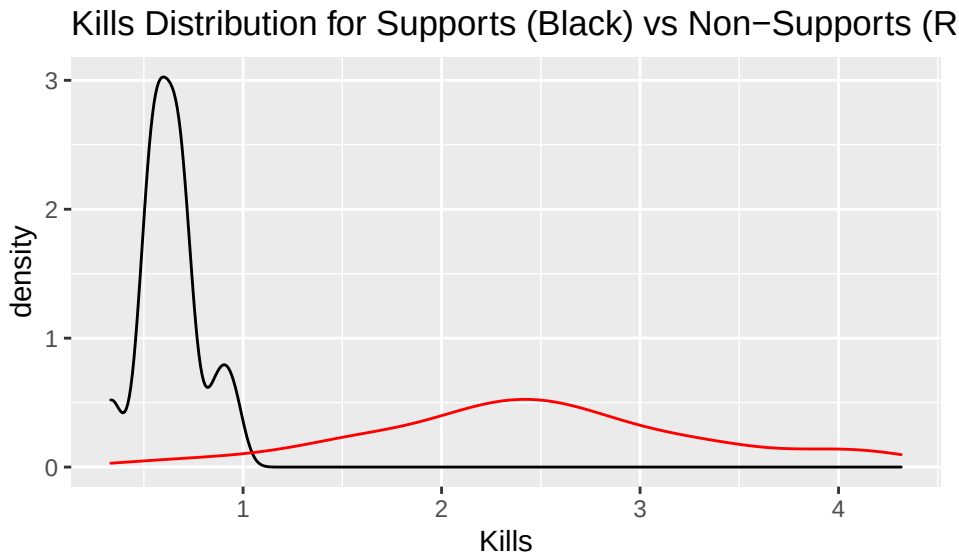
Looking at the five stat summaries of the variables for player statistics, we see that they are on completely different scales. We should be careful to scale it if scale matters (such as kNN). Looking at various histograms, we see most are normal. Looking at individual density plots (not shown), we see most distributions are normally distributed.

We looked at various histograms of player statistics and an interesting thing to note here in the histogram of kills is that it is bimodal, but that might be due to supports have less kills. If we separated this by support and non-support, we should see a better picture.

```
supp <- avg_players %>%
  filter(position == "sup")

non_supp <- avg_players %>%
  filter(position != "sup")

ggplot(supp, aes(x = kills)) +
  geom_density() +
  geom_density(data = non_supp, aes(x = kills), color = "Red") +
  labs(x = "Kills",
       title = "Kills Distribution for Supports (Black) vs Non-Supports (Red)")
```



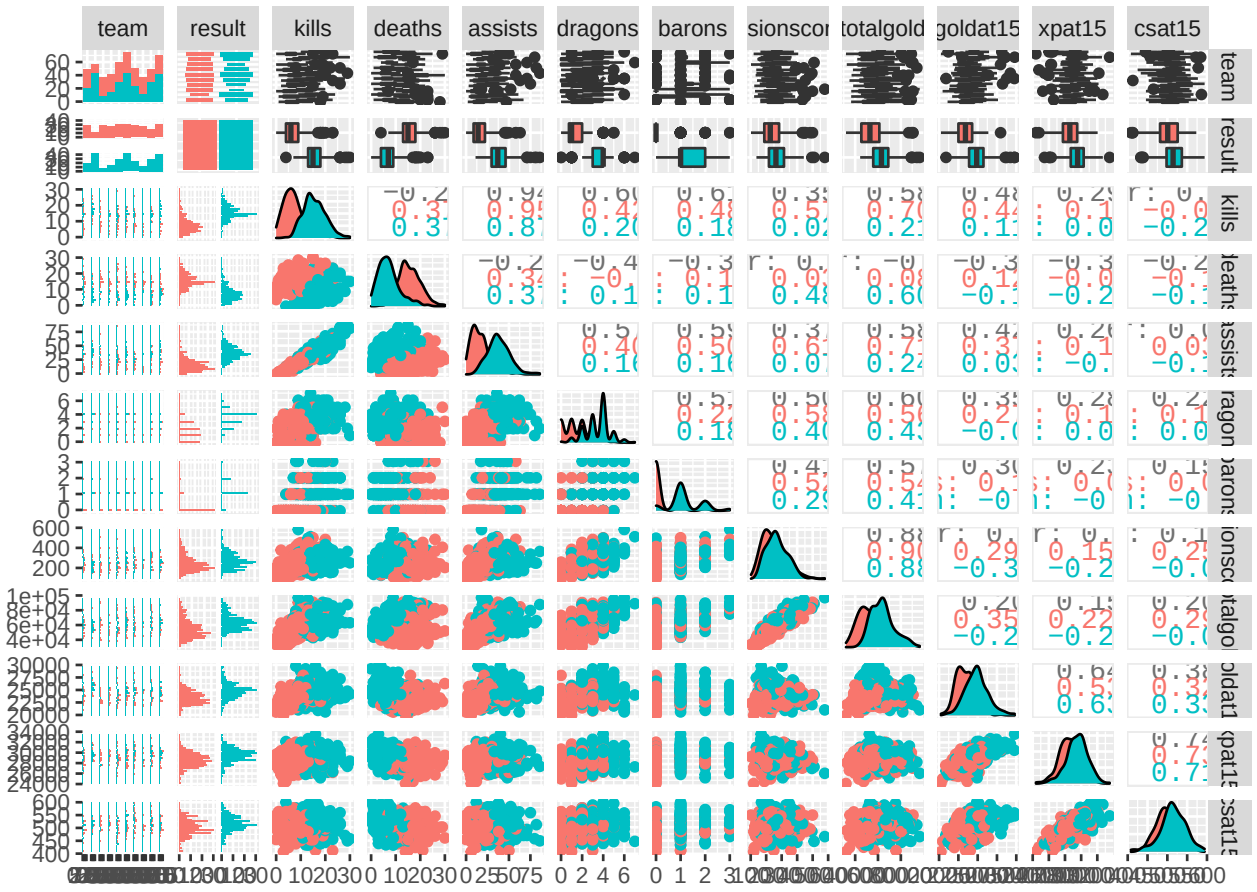
We see clear differences and this leads us to believe there to be a clear separation between support players and non-support players.

Bivariate Analysis

```
tally(~team + result, data = teams)
```

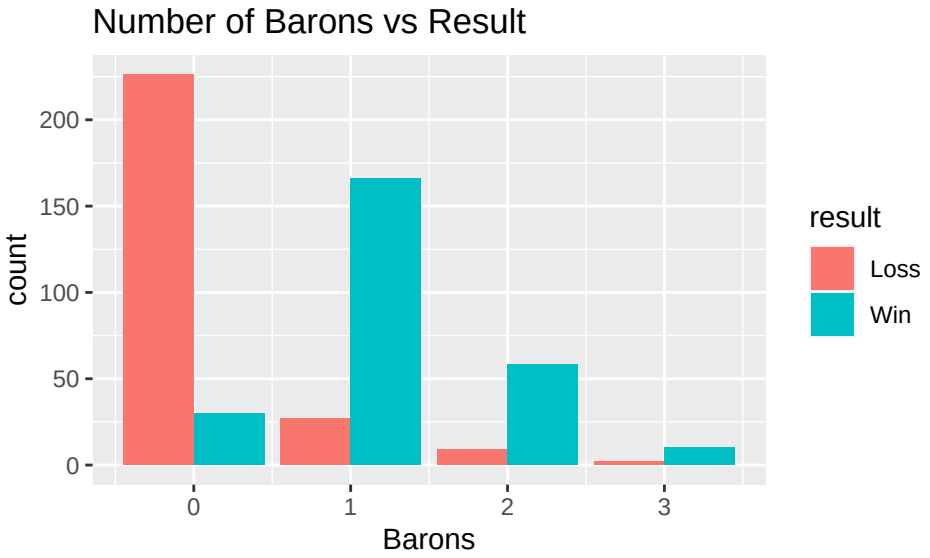
```
##                       result
## team                   Loss Win
##   100 Thieves            29  20
##   Cloud9                 13  44
##   Counter Logic Gaming   29   8
##   Dignitas               27  15
##   Evil Geniuses          31  29
##   FlyQuest               32  43
##   Golden Guardians       28  23
##   Immortals              25  12
##   Team Liquid            20  29
##   Team SoloMid           30  41
```

```
ggpairs(teams, ggplot2::aes(color = result))
```



From the plot output above colored by results, which is what we are interested in classifying, we can see there seems to be differences in the distribution of kills, deaths, assists which seems to be no surprise. Winning team usually has more kills, less deaths, more assists. The most starkly different is the barons variable which has completely different distributions. However these differences become less clear with variables such as goldat15, xpat15, and csat15 which are normally distributed and mostly overlapping. Vision score is also pretty similar and gold as well which is surprising to me. But from this, it leads me to believe kills, deaths, assists, and barons will have the most impact on classification results. There are a few variables like death when faceted by result that seem to still be right-skewed which might affect multivariate normality conditions for our future analyses such as LDA.

```
ggplot(teams, aes(x = barons, fill = result)) +
  geom_bar(position = "dodge") +
  labs(x = "Barons", title = "Number of Barons vs Result")
```



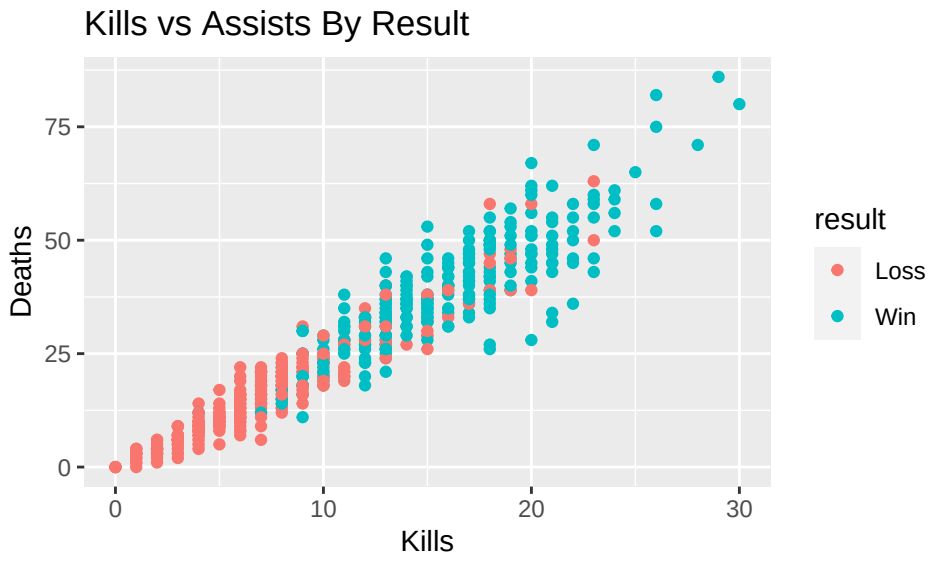
Here, we see a better picture of how barons slain vary by winning and losing team with the winning team usually having more barons.

```
cor(teams[-c(1,2)])
```

```
##                 kills     deaths    assists   dragons    barons visionscore
## kills       1.0000000 -0.2702039  0.9491964  0.607285  0.611684   0.3566789
## deaths     -0.2702039  1.0000000 -0.2653044 -0.420031 -0.342265   0.0464776
## assists     0.9491964 -0.2653044  1.0000000  0.579361  0.597827   0.3741003
## dragons     0.6072848 -0.4200305  0.5793615  1.000000  0.516420   0.5045143
## barons      0.6116841 -0.3422646  0.5978274  0.516420  1.000000   0.4189975
## visionscore 0.3566789  0.0464776  0.3741003  0.504514  0.418998   1.0000000
## totalgold   0.5836341 -0.0434038  0.5834615  0.604021  0.572373   0.8877526
## goldat15    0.4801663 -0.3362665  0.4197053  0.350676  0.300865   0.0454074
## xpat15      0.2966114 -0.3224403  0.2687798  0.282327  0.230857   0.0280857
## csat15      0.0658215 -0.2785515  0.0828852  0.229243  0.158335   0.1303525
##             totalgold   goldat15     xpat15     csat15
## kills       0.5836341  0.4801663  0.2966114  0.0658215
## deaths     -0.0434038 -0.3362665 -0.3224403 -0.2785515
## assists     0.5834615  0.4197053  0.2687798  0.0828852
## dragons     0.6040206  0.3506757  0.2823266  0.2292431
## barons      0.5723733  0.3008654  0.2308571  0.1583353
## visionscore 0.8877526  0.0454074  0.0280857  0.1303525
## totalgold   1.0000000  0.2069180  0.1501009  0.2024284
## goldat15    0.2069180  1.0000000  0.6429504  0.3845542
## xpat15      0.1501009  0.6429504  1.0000000  0.7402698
## csat15      0.2024284  0.3845542  0.7402698  1.0000000
```

Overall, we see some strong correlations between variables. One example is kills with assist at 0.95 and visionscore and totalgold at 0.88. Let's see how the relationship between kills with assist differs between wins and losses.

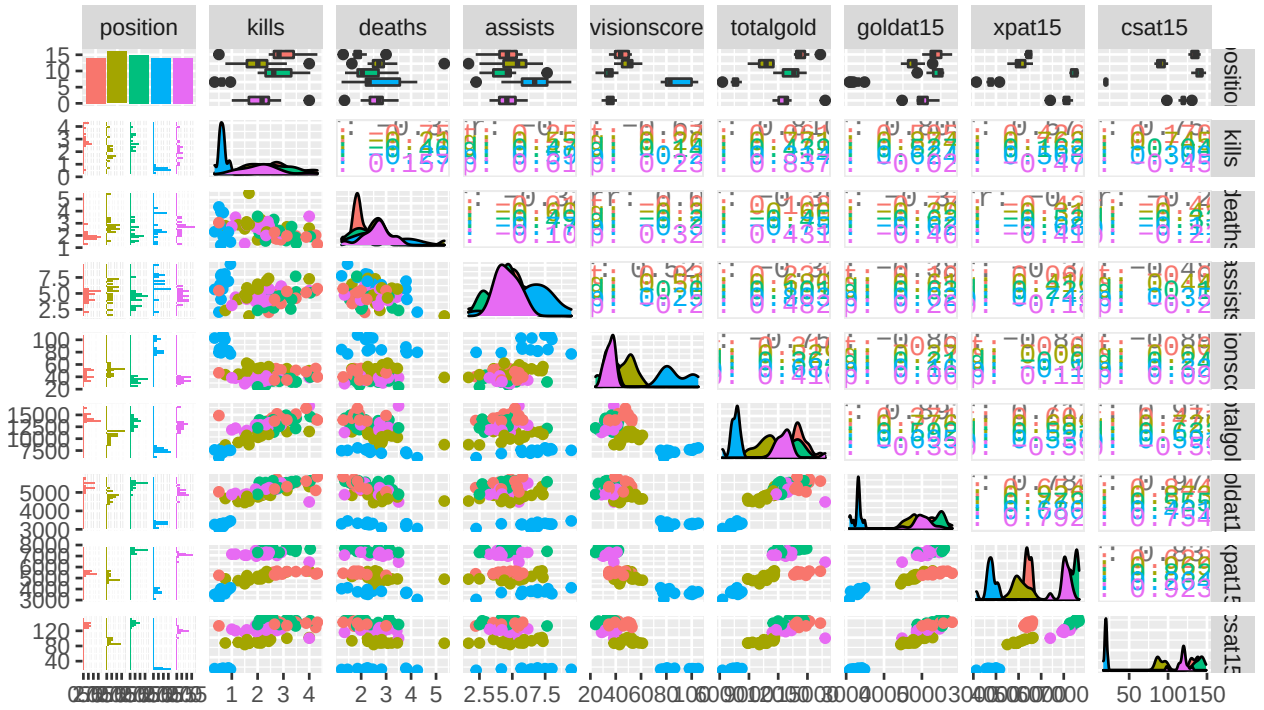
```
ggplot(teams, aes(x = kills, y = assists, color = result)) +
  geom_point() +
  labs(x = "Kills", y = "Deaths", title = "Kills vs Assists By Result")
```



Here we see a difference in kills/assists for winning and losing teams. Winning teams tend to have more kills/assists which makes sense.

Next, let's take a look at player statistics colored by position, since we observed above that support players had very different distributions compared to non-support players. How do each of the different positions compare then?

```
ggpairs(avg_players[,-1], ggplot2::aes(color = position))
```



From the plot output above colored by position, we can see there seems to be obvious differences in the "____at15" variables where blue are the support players who's job is not to get gold and experience but to support by providing vision around the map. If we were to cluster players, we will probably see a natural modality separating support players versus non-support players.

Methods

Classification Methods

For classification, we will construct several different models using different techniques listed below and choose the best one for classifying results. We want to use classification due to the dichotomous nature of what we are interested in (wins vs losses).

Trees: Classification trees are made by recursive partitioning of the variable space into hypercubes where each hyper cube gets a class label. This is visualized in a tree format where splits in the tree are where hypercubes are split and the branches are the hypercubes. At the end of each branch is a "node" which is the class label. When creating the tree, the user can determine how to create hypercubes and how many hypercubes there should be. This is also called splitting and pruning. In splitting, the user can tell the tree to not split if there are x or fewer number of observations at that node. We can also tell the tree the minimum number of observations each node can have after split. We can also choose to prune the tree which is choosing how many hypercubes there will be. We can determine this by some complexity parameter cutoff, number of splits, or number of nodes. We can also choose how our input variables as well. In terms of CV, we can choose how many k folds we want for cross validation techniques.

Random Forests: Random forests are a tree-based method as well but instead of one tree, many trees are created and majority voting is used to predict classes. To do so, n bootstrapped samples are created where a tree is created from each. And in those trees, k variables are randomly considered when performing splits. Then, majority voting is conducted over all trees to identify the class. When performing Random Forests, the user can determine the number of bootstrap samples (trees) and number of variables to consider at each split.

Linear Discriminant Analysis: Linear Discriminant Analysis is a classification technique that searches for linear combinations of variables that separate the groups. In terms of user input, LDA has to use all variables so if we want subsets of variables, we would have to perform LDA again but on different combinations of variables. In order to perform LDA, there has to be multivariate normality and equal covariances. Because of some multivariate normality concerns mentioned in the preliminary analyses section, we must proceed with caution using LDA.

k-Nearest Neighbor: k-Nearest Neighbor is a nonparametric method that classifies objects based on their nearest neighbors. This is determined by some distance, usually the Euclidean distance. And the class is based on what the majority class is for those k-Nearest Neighbors. The user determines how many nearest-neighbors are used to classify the object.

Clustering

For clustering, we will use two different clustering algorithms to cluster players, which we briefly explain below. We will choose the best clustering algorithm and describe it more in depth.

Agglomerative hierarchical clustering: This is a type of clustering that starts with all individuals as their own cluster and combines the two most similar clusters together and repeats this process until it becomes one cluster that contains all the individuals. This is often represented as dendrogram (picture an upside down tree). In order to choose how many clusters, we will look at the dendrogram and see where joining becomes really slow (large gaps). We will use Ward's method for our linkage to avoid issues of chaining from single linkage.

Kmeans clustering: This is a type of clustering that clusters n individuals into k groups based off trying to minimize some measure which is usually the Within Group Sum of Squares or the variation within a group. Before applying kmeans clustering, we will choose the k groups based off an elbow plot of the WGSS and choosing the number of clusters based off of where the "elbow" is.

Before applying these two methods, we will first scale our data in both clustering methods in order to maintain consistency across the board and because from our preliminary analyses, we found the scales to be

very different across the variables. We also decided on using the Euclidean as a measure of distance because all our variables are quantitative.

Finally, we will then use the **clValid** package to look at cluster strength and validity using the internal validation criteria to see if there are better cluster solutions than the ones we chose. clValid will help us pick the best model from hierarchical or kmeans clustering.

After finding the best clustering solution, we will then visualize the clusters in 2D space using Principal component analysis (PCA), a method of dimension reduction since we have several variables. We will then describe the clusters found and provide any insights extracted on player groupings.

Results

Classification

Trees (3 models)

We constructed 3 different tree models, each using minsplit of 10 and minbucket of 3 but differ in terms of variables used and number of folds in cross-validation.

```
# useful for later
teams_noname <- select(teams, - team)
```

```
#Tree Model 1 using jackknife cv
set.seed(1337)
g.control <- rpart.control(minsplit = 10, minbucket = 3, xval = 528)
g.treeorig <-rpart(result ~., data = teams_noname, method = "class", control = g.control)
printcp(g.treeorig)
plot(g.treeorig)
text(g.treeorig, cex = 0.7)
#aer: 0.5*0.1136 = 0.0568
#est ter 0.5*0.1780 = 0.089
```

This first tree performs LOOCV using minsplit of 10 and minbucket of 3. We get an AER of 5.7% and estimated TER of 8.9%.

```
#Tree Model 2
set.seed(1337)
g.control2 <- rpart.control(minsplit = 10, minbucket = 3, xval = 10)
g.treeorig2 <-rpart(result ~., data = teams_noname, method = "class", control = g.control2)
printcp(g.treeorig2)
plot(g.treeorig2)
text(g.treeorig2, cex = 0.7)
#aer 0.5*0.1136 = 0.0568
#est ter 0.5*0.1818 = 0.0909
```

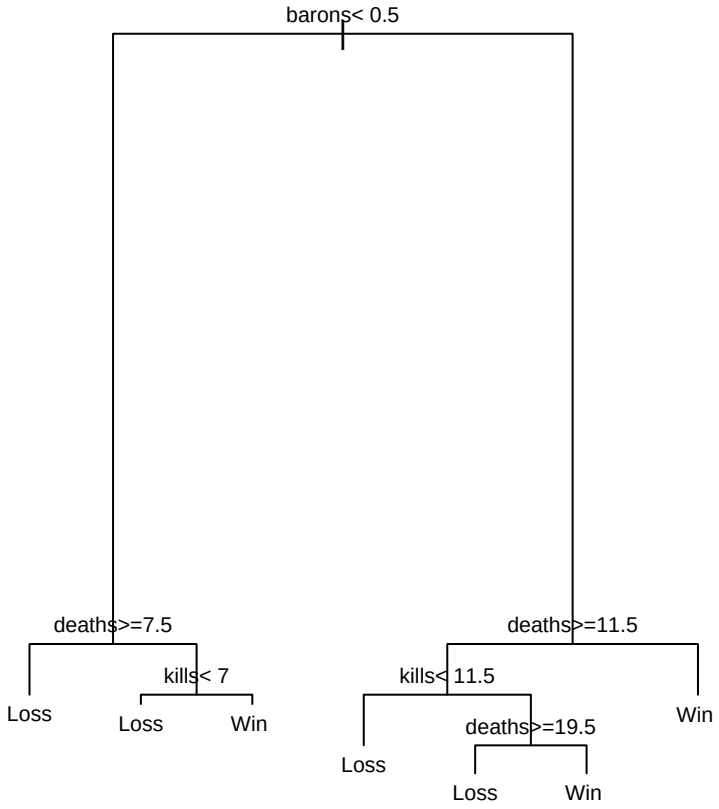
This second tree performs 10 fold CV using minsplit of 10 and minbucket of 3. We get an AER of 5.7% and estimated TER of 9%.

```
#Tree Model 3
set.seed(1337)
g.control3 <- rpart.control(minsplit = 10, minbucket = 3, xval = 528)
g.treeorig3 <-rpart(result ~ kills + deaths + assists + barons, data = teams_noname, method = "class",
printcp(g.treeorig3)
```

```
##
## Classification tree:
## rpart(formula = result ~ kills + deaths + assists + barons, data = teams_noname,
##     method = "class", control = g.control3)
##
## Variables actually used in tree construction:
## [1] barons deaths kills
##
## Root node error: 264/528 = 0.5
```

```
##
## n= 528
##
##         CP nsplit rel error xerror    xstd
## 1 0.74242      0    1.0000 2.0000 0.00000
## 2 0.04167      1    0.2576 0.2576 0.02915
## 3 0.02273      2    0.2159 0.2765 0.03004
## 4 0.01136      5    0.1364 0.2121 0.02680
## 5 0.01000      6    0.1250 0.1705 0.02430
```

```
plot(g.treeorig3)
text(g.treeorig3, cex = 0.7)
```




```
#aer 0.5*0.1250 = 0.0625
#est ter 0.5*0.1705 = 0.08525
```

The third tree use LOOCV but differs in the variables inputted. It has the top 4 variables indicated from the Random Forest models below. However, in model creation, it only uses barons, deaths, and kills. We get an AER of 6.3% and estimated TER of 8.5%.

Our third tree model has the lowest estimated TER at 8.5% (the others are near 8-9% as well so we are choosing by slim margins here) and the tree itself is pretty interpretable and not too complex. Any tree would be fine here but going off purely lowest estimated TER, we choose model 3.

Looking at model 3, we see the first split is based on how many barons a team has then the other splits depends on deaths and kills. It tends to be that the more deaths and the less kills are classified as losses and vice versa, more kills and less deaths are wins. Suppose we have a team that had 0 barons, 10 deaths, and 6 kills, we see that it goes down the left side then because the team has 10 deaths, it classifies it as a loss.

What is interesting is that in the preliminary analyses, we thought assists might also help in classification, but in this tree, assists is not used at all.

RFs (3 Models)

Next, we construct 3 different Random Forest models where the number of variables tried at each split and how many trees constructed are changed for each model.

```
#RF Model 1
set.seed(1337)
g.rf <- randomForest(result ~ ., data = teams_noname, mtry = 3, ntree = 500,
                     importance = T, proximity = T)
g.rf
# 0% aer
#6.63%% est ter
table(teams$result, predict(g.rf, teams_noname))%>%
  get_error()
gf_histogram(~ treesize(g.rf))
varImpPlot(g.rf)
```

In our first RF model, we set our mtry = 3 and ntree = 500. We get an AER of 0% and estimated TER of 6.63%%. This is already lower than trees itself so it's a good sign but let's see how other models stack up.

```
g.rf2 <- randomForest(result ~ ., data = teams_noname, mtry = 10, ntree = 1000,
                        importance = T, proximity = T)
g.rf2
#aer 0%
#est. ter 7.01%
table(teams$result, predict(g.rf2, teams)) %>%
  get_error()
gf_histogram(~ treesize(g.rf2))
varImpPlot(g.rf2)
```

In our second RF model, we set our mtry = 10 and ntree = 1000. We get an AER of 0% and estimated TER of 7.01%.

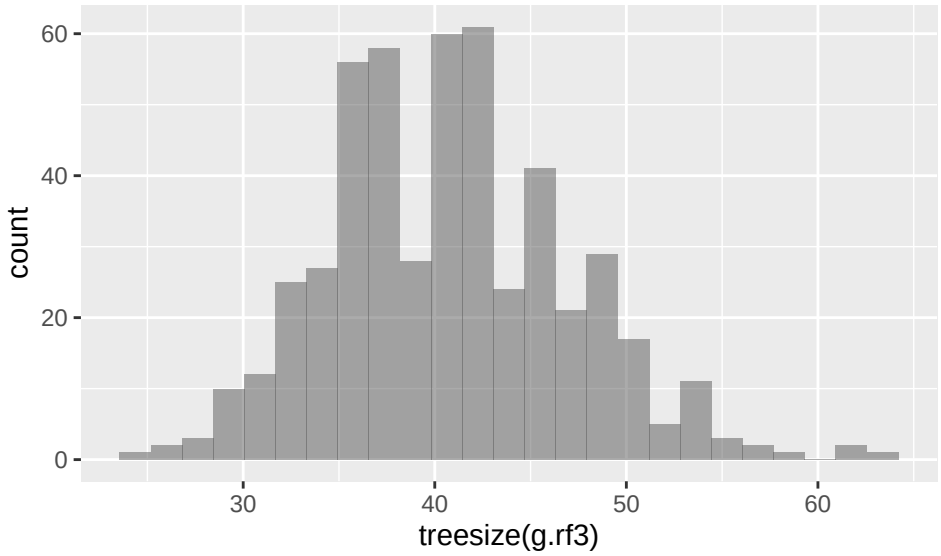
```
g.rf3 <- randomForest(result ~ ., data = teams_noname, mtry = 2, ntree = 500,
                        importance = T, proximity = T)
g.rf3
```

```
##
## Call:
##  randomForest(formula = result ~ ., data = teams_noname, mtry = 2,      ntree = 500, importance = T,
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 6.25%
## Confusion matrix:
##      Loss Win class.error
## Loss  245  19   0.0719697
## Win    14 250   0.0530303
```

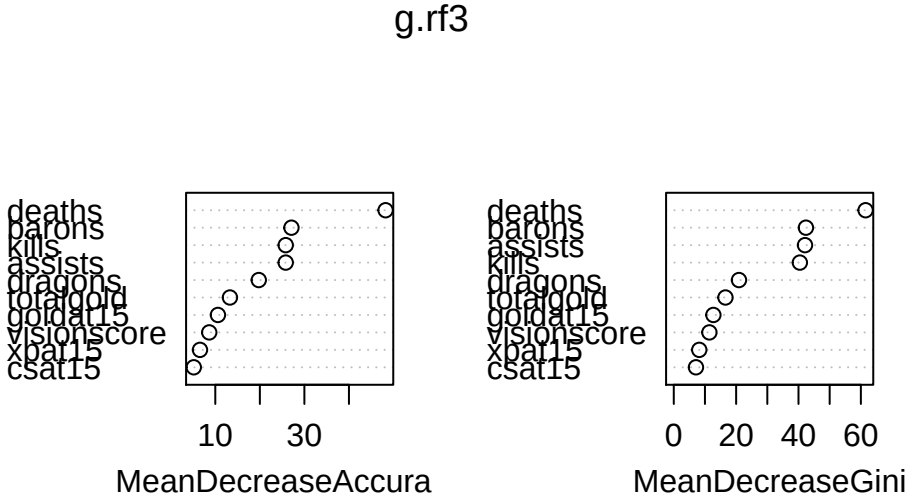
```
table(teams$result, predict(g.rf3, teams)) %>%
  get_error()
```

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

```
gf_histogram(~ treesize(g.rf3))
```




```
varImpPlot(g.rf3)
```



In our third RF model, we set our mtry = 2 (how many variables considered at each split) and ntree = 500. We get an AER of 0% and estimated TER of 6.44%%.

Based off lowest estimated TER, we choose our third RF model which has an estimated TER of 6.63%. The histogram shows us the size of the 500 trees so it looks like around 40 is the average size. Looking at the RF model and it's Gini variable importance plot, we see that death / barons / kills / assists seem to matter the most in classifying wins. There is a clear difference between the the top four (deaths, barons, kills, assists) and the 4th (dragons). Because the variables seem to be the most important, let's use only these to construct a tree model, a LDA model, and a kNN model as opposed to all predictors.

Seeing the importance of variables is interesting as it seems variables like gold, cs, xp do not seem to matter as much in classifying results but getting barons and how much kills/deaths/assists you have matter. Deaths seem to matter the most in classifying wins, followed by barons. Moreover, from our preliminary analysis, distribution of dragons differed by result so we thought dragons would also be important in classification but we see here that dragons are less important than barons. In the game, some teams prioritize barons over dragons depending on the situation so this model possibly suggests barons might help a team win more than getting dragons.

LDA (2 Models)

We also consider 2 LDA models but because multivariate normality might not be met, we should be cautious about our results.

```
#LDA Model 1
g.lda <- MASS::lda(result ~ ., data = teams)
g.ldapred <- predict(g.lda, teams)
table(teams$result, g.ldapred$class)
table(teams$result, g.ldapred$class) %>%
  get_error()
#aer = 0.0492424
#cross-validation
g.lda_cv <- MASS::lda(result ~ ., data = teams, CV = T)
table(teams$result, g.lda_cv$class) %>%
  get_error()
# est. TER = 0.0568182
```

In our first LDA model, we used every input variable and we get an AER of 4.9% and an estimated TER of 5.6%.

```
#LDA Model 2
g.lda2 <- MASS::lda(result ~ kills + deaths + assists + barons, data = teams)
g.lda2
```

```
## Call:
## lda(result ~ kills + deaths + assists + barons, data = teams)
##
## Prior probabilities of groups:
## Loss  Win
##  0.5  0.5
##
## Group means:
##         kills   deaths assists   barons
## Loss  7.18939 15.61742 16.4280 0.193182
## Win  15.60985  7.20076 38.2159 1.181818
##
## Coefficients of linear discriminants:
##               LD1
## kills    0.128060
## deaths  -0.192404
## assists  0.014418
## barons   0.608596
```

```
g.lda2pred <- predict(g.lda2, teams)
table(teams$result, g.lda2pred$class)
```

```
##
##        Loss Win
##   Loss  248  16
##   Win    12 252
```

```
table(teams$result, g.lda2pred$class) %>%
  get_error()
```

```
## [1] 0.0530303
```

```
#aer = 0.0530303
#cross-validation
g.lda2_cv <- MASS::lda(result ~ kills + deaths + assists + barons, data = teams, CV = T)
table(teams$result, g.lda2_cv$class) %>%
  get_error()
```

```
## [1] 0.0568182
```

```
# est. TER = 0.0568182
```

In our second LDA model, we used the four variables identified as most important from the RF section and we get an AER of 5.3% and an estimated TER of 5.6%.

Both models have similar estimated TER's so either could work here. Model 1 does have a lower AER than model 2 but model 2 has less variables in it's input. Both models are very similar in terms of error rates so we choose the second LDA model over the first because because it has less input variables so it's less complicated.

From the output, we can see the group means for those that win and those that lose which supports our preliminary analyses that winners have more kills, less deaths, more assists, and more barons.

kNN (3 Models)

We constructed 3 different kNN models. The first two used all variables while the last one used the top four identified in the Random Forest section. In all models, the variables are scaled since our variables are on vastly different scales.

```
set.seed(1337)
#kNN Model 1
g.knn <- knn(scale(select(teams_noname, -result)),
             scale(select(teams_noname, -result)),
             teams_noname$result, k = 4, prob = T)
table(teams_noname$result, g.knn) %>%
  get_error()
```

```
## [1] 0.0435606
```

```
# AER =  0.0435606
g.knncv <- knn.cv(scale(select(teams_noname, -result)),
                  teams_noname$result, k = 4, prob = T)
table(teams_noname$result, g.knncv) %>%
  get_error()
```

```
## [1] 0.0625
```

```
# est. TER = 0.0625
```

For our first kNN model, we used k = 4 and got an AER of 4.4% and an estimated TER of 6.25%.

```
set.seed(1337)
#kNN Model 2
g.knn <- knn(scale(select(teams_noname, -result)),
             scale(select(teams_noname, -result)),
             teams_noname$result, k = 5, prob = T)
table(teams_noname$result, g.knn)%>%
  get_error()
# AER =  0.0435606
g.knncv <- knn.cv(scale(select(teams_noname, -result)),
                  teams_noname$result, k = 5, prob = T)
table(teams_noname$result, g.knncv) %>%
  get_error()
# est. TER = 0.0643939
```

For our second kNN model, we used k = 5 and got an AER of 4.4% and an estimated TER of 6.5%.

```
#Useful for later models
teams_noname2 <- select(teams_noname, result, kills, deaths, assists, barons)
```

```
set.seed(1337)
#kNN Model 3
g.knn3 <- knn(scale(select(teams_noname2, -result)),
              scale(select(teams_noname2, -result)),
              teams_noname2$result, k = 4, prob = T)

table(teams_noname2$result, g.knn3)%>%
  get_error()
# AER =  0.0549242
g.knncv3 <- knn.cv(scale(select(teams_noname2, -result)),
                   teams_noname2$result, k = 4, prob = T)
table(teams_noname2$result, g.knncv3) %>%
  get_error()
# est. TER = 0.0814394
```

For our third kNN model, we used k = 4 and only used the 4 variables the RF model deemed to be the most important and got an AER of 5.5% and an estimated TER of 8.1%.

Our best kNN model happens to be the first model with an estimated TER of 6.25%.

Final Model / Comments

The best models from each technique were:

Tree Model 3 with an AER of 6.3% and estimated TER of 8.5%.

RF Model 3 with an AER of 0% and estimated TER of 6.44%%.

LDA Model 2 with an AER of 5.3% and an estimated TER of 5.6%.

kNN Model 1 with an an AER of 4.4% and an estimated TER of 6.25%.

The best two models we constructed were the first kNN model with an estimated TER of 6.25% and the second LDA model with an estimated TER of 5.6%. While LDA model 2 has a slightly lower estimated TER than kNN model 1 (<1%), conditions of multivariate normality might not be met which is a concern that the kNN model does not need. So we will choose the kNN model with k = 4 which uses all variables as inputs to help us classify wins.

Clustering

Clustering Players

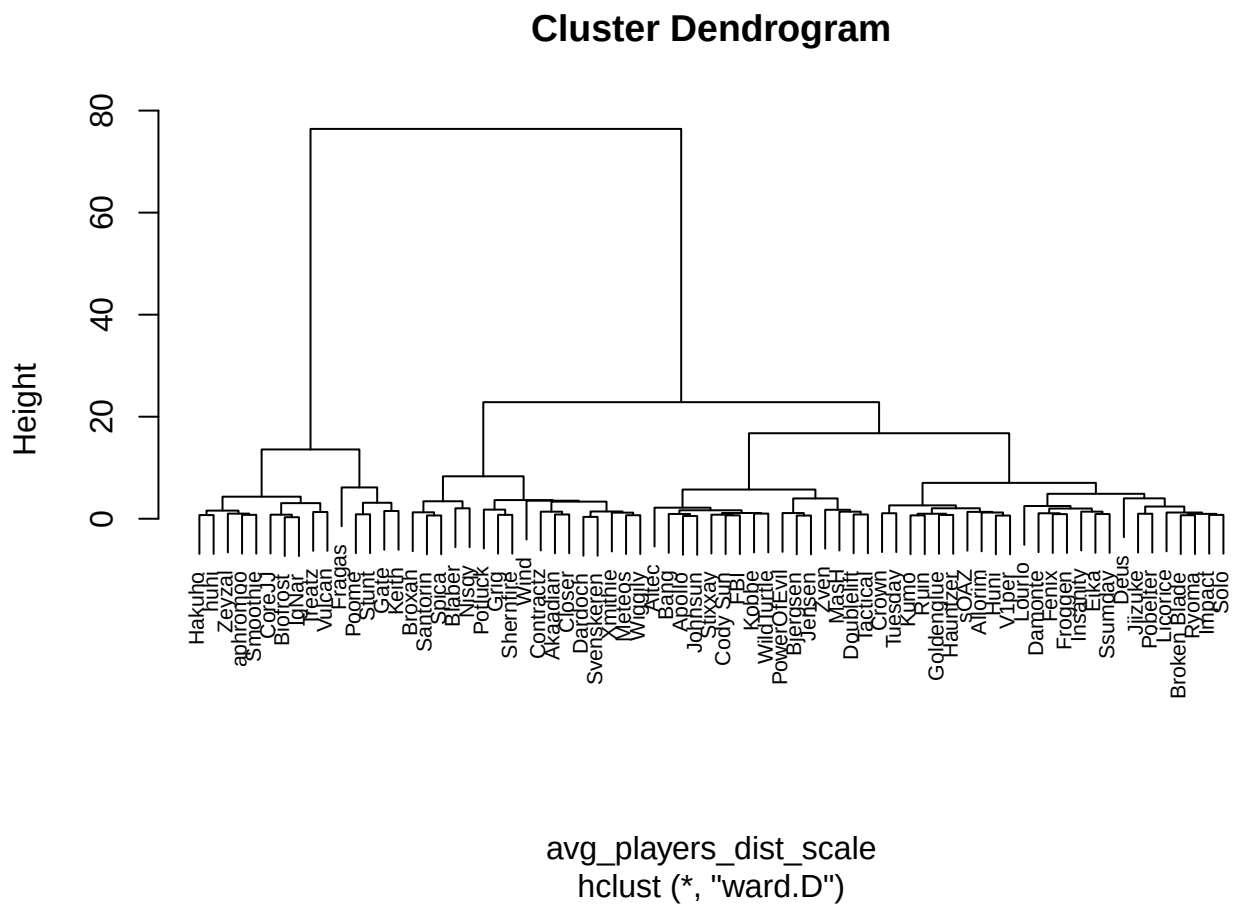
Let's do some quick data wrangling to get just the quantitative data, scale it, and a scaled distance matrix.

```
# getting only ints
avg_players_int <- select(avg_players,-player,-position)
# scaling
avg_players_int_scale <- scale(avg_players_int)
# scaled distance
avg_players_dist_scale <- dist(avg_players_int_scale)
```

Hierarchical Clustering

Let's first do agglomerative hierarchical clustering using Ward's method and plot it's dendrogram.

```
hcward <- hclust(avg_players_dist_scale, method = "ward.D")
plot(hcward, cex = 0.7, labels = avg_players$player)
```

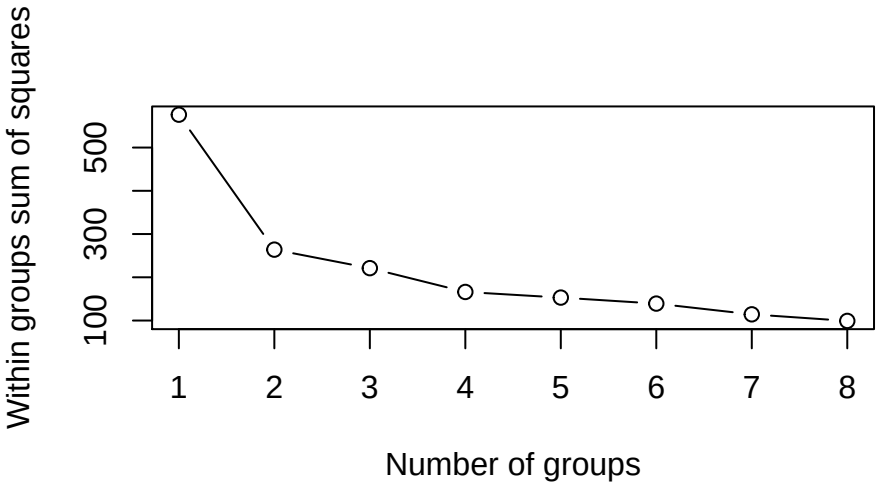


Looking at the dendrogram, there appear to be 2 clusters of players if we cut at a height of 30. From a quick glance, it appears they clustered the supports (left cluster) together and then the non-supports (right cluster) in the other cluster which is what we originally suspected would happen. Now let's try using kmeans and see what we get.

Kmeans Elbow Plot

Before running kmeans, we need to choose our cluster solution. In order to pick the number of clusters, let's look at the WGSS elbow plot to help us choose.

```
set.seed(1337)
n <- nrow(avg_players_int)
wss <- rep(0, 8)
for(i in 1:8){wss[i] <- sum(kmeans(avg_players_int_scale, centers = i)$withinss)}
plot(1:8, wss, type = "b", xlab = "Number of groups", ylab = "Within groups sum of squares")
```

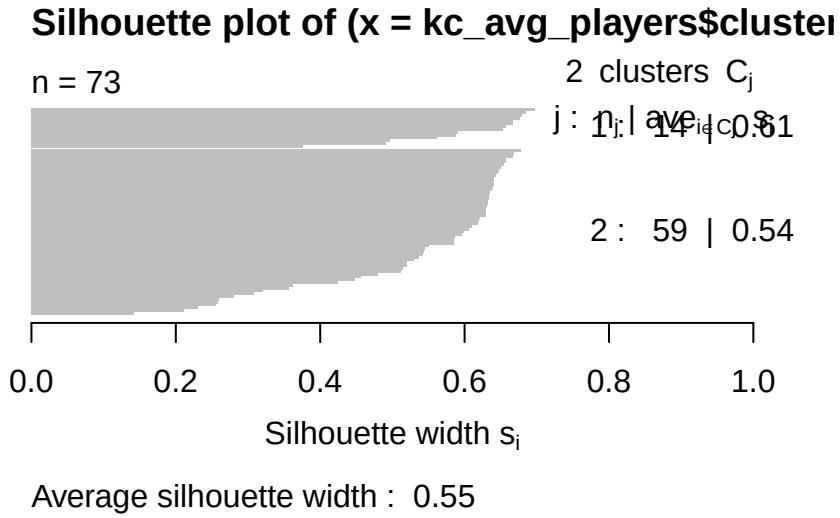


From the WGSS plot, we can see there to be an elbow at 2 clusters so let's continue with 2 clusters. From the hierarchical clustering that separated support players and nonsupport players, we have a suspicion kmeans will do the same.

Kmeans Silhouette Plot

Now, let's cluster using a 2 cluster solution and look at it's silhouette plot which can give us some indication about cluster strength.

```
kc_avg_players <- kmeans(avg_players_int_scale, centers = 2)
kmeansSil <- silhouette(kc_avg_players$cluster, avg_players_dist_scale)
plot(kmeansSil)
```

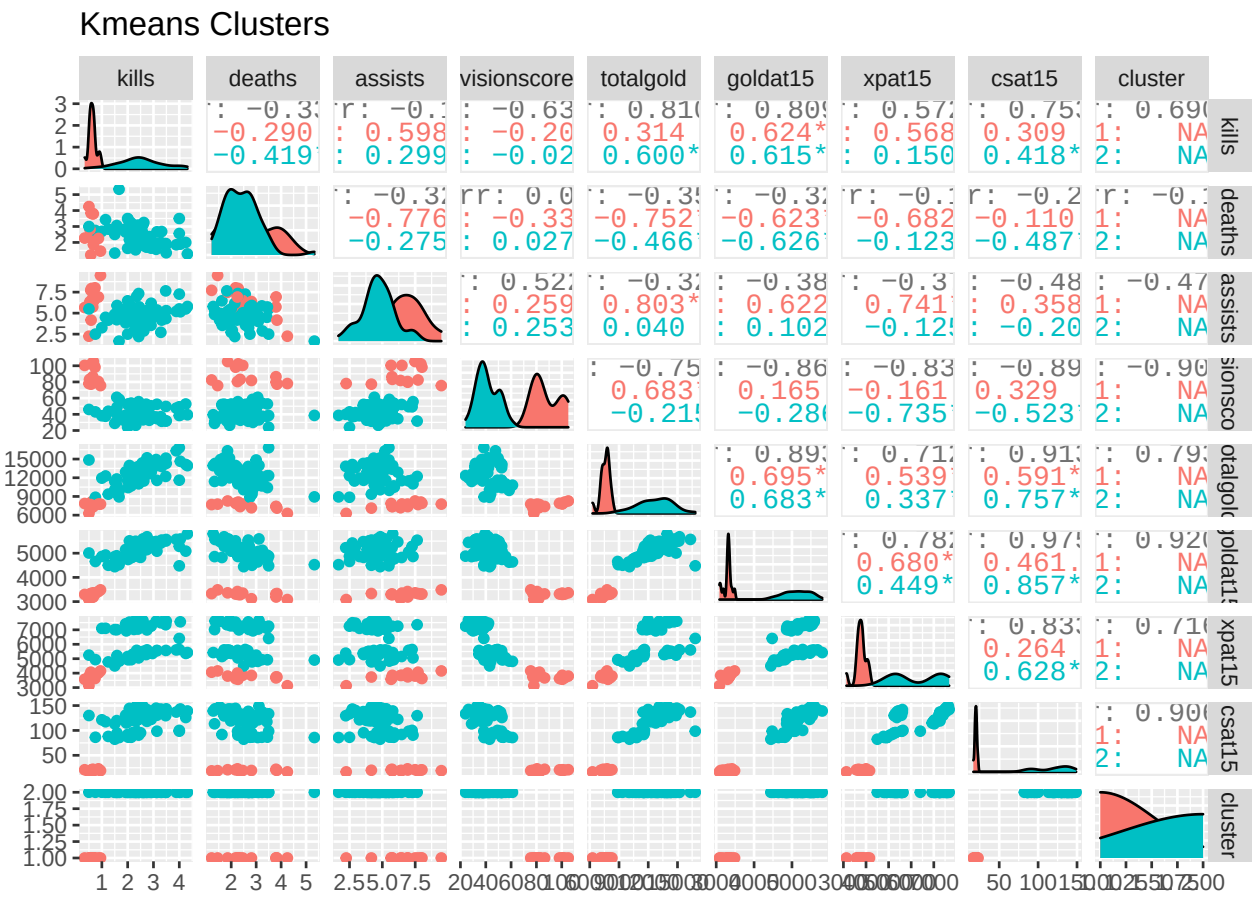


From the silhouette plot, we get an average silhouette width of 0.55 which is pretty decent. Rule of thumb says anything over 0.5 so there appears to be moderately strong clustering. Cluster 1 has 14 observations while cluster 2 has 59 so they are not quite even. Once again, probably support versus non-support players.

Let's plot these clusters using a scatterplot matrix to see differences.

```
avg_players_cluster <- mutate(avg_players, cluster = kc_avg_players$cluster)

ggpairs(data = select(avg_players_cluster, -player, -position),
        mapping=ggplot2::aes(colour = factor(cluster))) %>%
gf_labs(title = "Kmeans Clusters")
```



Looking at this scatterplot matrix, we see cluster 1 tends to have less kills, more assists, and really high visionscore, which provides very strong evidence these are support players being clustered.

Cluster Validation

Let's also see if there is a better solution than our hierarchical or kmeans clustering solution. And to help us, we will use the *clValid* package to see how different clustering solutions compare.

```
library(clValid)
set.seed(1337)
compare <- clValid(avg_players_int_scale, 2:8,
                   clMethods = c("hierarchical", "kmeans"), metric = "euclidean",
                   method = "ward", validation = c("internal"))
optimalScores(compare)
```

```
##                  Score Method Clusters
```

```
## Connectivity 0.772222 kmeans        2
## Dunn         0.415962 kmeans        2
## Silhouette   0.549057 kmeans        2
```

Using internal validation measures, a kmeans solution with 2 clusters is the best across all measures (Connectivity, Dunn, Silhouette). So we can stick with our kmeans 2 cluster solution and visualize it in PC space.

PCA Visualization

Let's construct PCs for our data using the correlation matrix since our variables are on drastically different scales.

```
avg_playersPCA <- princomp(avg_players_int, cor = TRUE)
summary(avg_playersPCA)
```

```
## Importance of components:
##                           Comp.1   Comp.2   Comp.3    Comp.4   Comp.5    Comp.6
## Standard deviation     2.30273 1.198778 0.677785 0.6491166 0.394380 0.3836512
## Proportion of Variance 0.66282 0.179633 0.057424 0.0526691 0.019442 0.0183985
## Cumulative Proportion  0.66282 0.842453 0.899877 0.9525465 0.971988 0.9903870
##                           Comp.7     Comp.8
## Standard deviation     0.2607451 0.09442337
## Proportion of Variance 0.0084985 0.00111447
## Cumulative Proportion  0.9988855 1.00000000
```

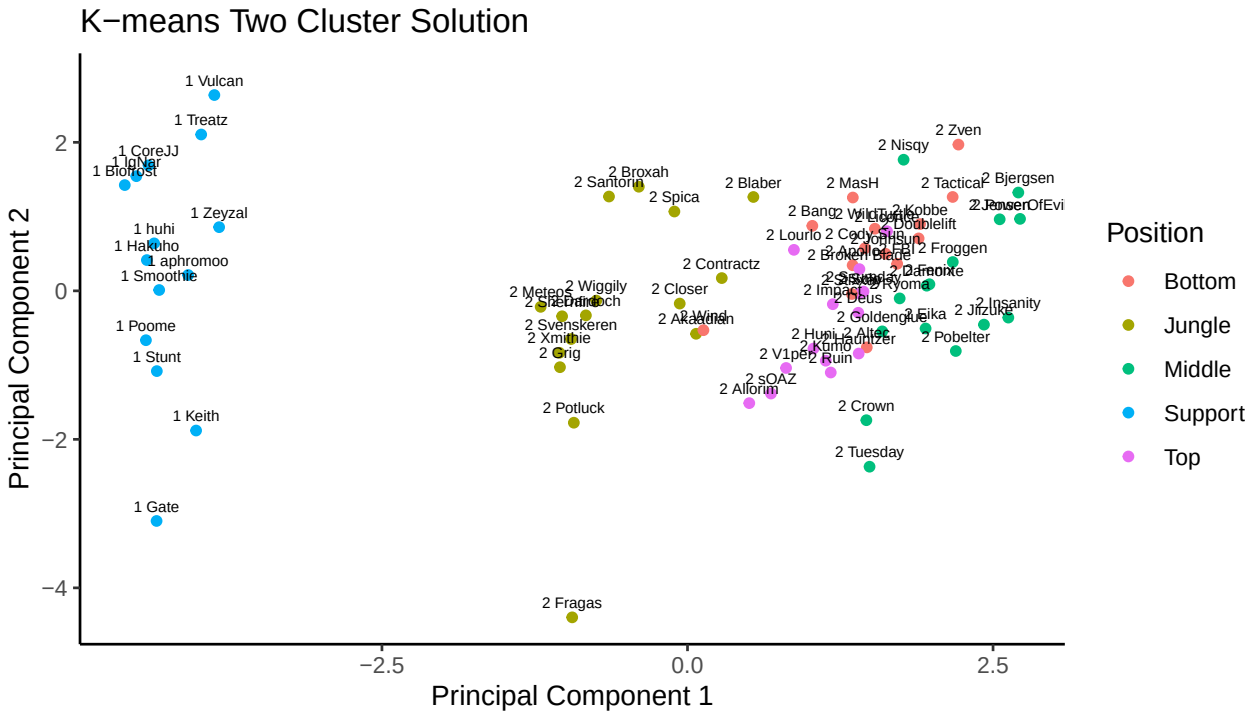
```
avg_playersPCA$loadings
```

```
##
## Loadings:
##             Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## kills        0.355  0.229  0.548  0.391  0.594
## deaths      -0.127 -0.701  0.654 -0.107 -0.161 -0.115 -0.119
## assists     -0.200  0.627  0.493 -0.485 -0.275
## visionscore -0.396  0.194         0.295  0.107 -0.614 -0.573
## totalgold    0.401  0.124  0.103  0.228 -0.486 -0.575  0.395 -0.196
## goldat15     0.422                      -0.226  0.282 -0.624 -0.543
## xpat15       0.373        -0.107 -0.677  0.425 -0.434
## csat15       0.428                      -0.260        -0.313  0.802
##
##                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## SS loadings     1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000
## Proportion Var  0.125  0.125  0.125  0.125  0.125  0.125  0.125  0.125
## Cumulative Var  0.125  0.250  0.375  0.500  0.625  0.750  0.875  1.000
```

Two PCs explains 84% of variation so using PCA to explore in 2D space seems appropriate.

```
ggplot(data = as.data.frame(avg_playersPCA$scores), aes(x =Comp.1, y = Comp.2)) +
  geom_point(aes(color = avg_players$position)) +
  geom_text(aes(label=paste(kc_avg_players$cluster, avg_players$player)),
            size = 2, nudge_y = 0.2) +
  labs(x = "Principal Component 1", y = "Principal Component 2") +
```

```
  ggtitle("K-means Two Cluster Solution") +
  theme_classic() +
  labs(color = "Position") +
  scale_color_discrete(name = "Position",
                       labels = c("Bottom", "Jungle", "Middle", "Support", "Top"))
```



Comments:

Using a 2 cluster kmeans solution, and plotting it in 2d PC space, we see it recovered a clear separation between support players and non-support players as we suspected. As a fan of the game, this makes a lot of sense why this a 2 cluster solution is the best since Supports contrast in pretty much all statistics versus the other roles. Supports don't get gold or experience and are much weaker but provide more utility to the team.

Although the 2 groups are distinct, when we examine our clusters in 2D space, we see separation between the different roles in the non-Support cluster. For instance, Jungle is near the center and each role is sort of clustered together. It makes sense for these natural modalities to appear since each position has different roles in the game. Supports don't get kills, farm for gold, but are meant to increase vision on the map. Bot/Mid are the "Carries" and they have to farm up and are meant to carry the game by killing everyone. Jungle also plays a more support style role so it makes sense they are more in the middle. Top is sort of between jungle and the carries since sometimes they can carry or sometimes they are supposed to be the tank and soak up damage.

Taking a look at the PCA loadings, we can get a better sense of how players stack up in this graph. Looking at PC1, we see it is a contrast of deaths/assists/visionscore versus everything else. That is why we see supports way on the left since their job is to assist and have "eyes" around the map. PC 2 is a contrast between deaths and everything else and the first thing that comes to my mind is that the player Fragas probably had a lot of deaths. Let's see who were the top three that died the most.

```
avg_players %>%
  arrange(desc(deaths)) %>%
  select(player, deaths) %>%
  head(3)
```

```
## # A tibble: 3 x 2
##   player deaths
##   <fct>   <dbl>
## 1 Fragas   5.33
## 2 Gate     4.25
## 3 Keith    3.83
```

No surprise that these three players with the highest average deaths per game are found on the graph as having very low PC 2 scores.

From the loadings, we can think of Support Players with high PC 2 scores to be "better" supports. Perhaps future clustering can go into looking at support players only to see if we can cluster players into tiers.

```
avg_players %>%
  filter(player %in% c("Gate", "Vulcan"))
```

```
## # A tibble: 2 x 10
##   player position kills deaths assists visionscore totalgold goldat15 xpat15
##   <fct>  <fct>    <dbl>  <dbl>   <dbl>       <dbl>     <dbl>    <dbl>  <dbl>
## 1 Gate   sup      0.5     4.25    2.25        78       6279     3084.  3130.
## 2 Vulcan sup      0.947   1.46    9.49        75.5     7770.    3485.  4154.
## # ... with 1 more variable: csat15 <dbl>
```

Here, we compare two support players, Gate who has the lowest PC2 score out of supports and Vulcan who has the highest and we see that Vulcan has more kills, assists, less deaths which are generally good things overall. When a support dies frequently, they can't roam around and provide vision for the team. Having a lot of assists also means you're doing your job as a support which is to assist your team.

Although a 2 cluster kmeans solution was the best based off internal validation measures, we can see natural clusters based off the different positions. Perhaps if we did three, we would then cluster for supports, junglers, and everyone else.

While a 2 cluster solution has the best silhouette value and showed extremely clear separation, this might be expected as supports is a very different role compared to everyone else. (Like a healer versus a Warrior) Perhaps more interesting clustering would be done on the positions themselves, like can we find clusters within support players? Our cluster solution does suggest the possibility of those clusters and it does reveal where players compare to other players within their role.

Conclusion

In our exploration of professional League of Legends games, we utilized two exploratory tools to answer two questions: "Can we classify wins and losses by various game statistics?" and "Can we find clusters of players based off their playstyles or game statistics?".

To answer the first, we explored several classification techniques but ultimately chose a kNN model that used k=4 with all variables as input that gave us an estimated TER of 6.25%. From our variable importance plot when creating RF models, we see that barons are more important than dragons for classifying wins. This might be lead us to suggest teams to prioritize differently when playing around these two objectives. Teams might want to give up dragons so they can get barons.

To answer the second, we looked at two clustering solutions, agglomerative hierarchical clustering and kmeans clustering, and ultimately decided on a kmeans 2 cluster solution that clustered the players into those that were Support players and those that weren't. We suspected these clusters to be recovered by the clustering algorithm since support players have very distinct roles compared to everyone else. This clustering solution has a fairly strong clustering structure from the silhouette value of 0.55. From the PC visualization, we can sort of see where the players stack up to one another. For supports specifically, the higher PC 2 generally means you are a better support. Further analysis should go into stratifying by specific role to see if we can cluster those players together. Analysis can also be done to compare players of the same role across different regions to see how playstyles compare and differ.

Citations

Sevenhuysen, Tom. Oracle's Elixir - LoL Esports Stats: 2020 Match Data, 2020, oracleselixir.com/tools/downloads.