Analytics Edge Project

Data Analysis: League of Legends Study

Final Report with Code Examples

Team Members:
CHOUHBI Kamal
HAMMOUT Nisrine
EL MOUTAKI Ayoub
CHERRADI Anas
KHOTBI Anas
EL MARRAKI Mohamed Amine

Supervisor:
Maryam GUESSOUS

March 29, 2019

Contents

Executive Summary

During 2 months, we did a short data project on League of Legends since we are fans of the game and its use of data. The project is focused on the question of whether or not we can create a model to predict winning teams based on performances. We use logistic regression and other models to predict the match outcomes. Features are extracted from the data that Riot Games API exposes–including champions picked for the game, player role information, and mastery levels for the players' champions (pre-game knowledge) as well as in-game player statistics. We find that using only prematch knowledge (champions, masteries, roles, spells) from the very start is only a weak predictor of match outcome but using in-game statistics, the model becomes a strong predictor.

Chapter 1

Introduction

1 An Introduction to Data & Analytics in the Gaming Industry

Just as it did with traditional sports, the collection, analysis, and use of all kinds of data is starting to change the way that competitive games are played and understood. Nowadays, as the E-sports became more and more popular, data analyses on E-sports games are also become much more common. Because of the foundations of multiple commercial E-sport Leagues, data analyses is playing a significant role, just like the role they play in NBA and other sport leagues.

In this project, the online game League of Legends (LoL), developed by Riot Games in 2009, will be our case study. The main reason is everyone in our group had played and enjoyed the game and would like to bring a more passionate and informed view to this project than they would with another case study. An important aspect was also Riot Games' API, which allows the public to easily make queries and pull information from their servers. Another great selling point is the fact that League of Legends is one of the most popular modern day E-sports, meaning its balance updates affect people's lives on a professional level.

2 Related Work

As League of Legends is a popular game, there are already a couple of applications that exist in the data analysis sphere. Websites like **League of Graphs and MetaSrc** collect basic stats and information from the League of Legends API and use it to construct simple analyses like what the most popular champion and what the win rate of two champions when played together are. What these existing applications do not currently do is prediction of a currently ongoing game with all of the features of that game in mind, and that is the hole that our project is attempting to fill.

3 Background on League of Legends as a Game

3.1 Game Overview :

League of Legends is a Multiplayer Online Battleground Arena (MOBA) style game developed by Riot Games. Each game consists of two teams with 5 players each. Each team starts the game at opposite sides of the arena in a base that contains their Nexus. The goal of each game is to overcome obstacles such as minions, structures and enemy players in order to destroy the enemy Nexus. Each team generally consists of 5 standard positions divided across the map: *Top, Middle, Jungle, Attack Damage Carry(ADC)/Marksman and Support*.

Each player in a match, usually lasting between twenty minutes and an hour, controls a unique champion chosen from a pool of more than a hundred with differing characteristics and abilities. The game boasts 100 million monthly players and a flourishing competitive scene with millions in tournament prize pools as well as online viewers.

With so many variables at play within each game it is impossible to say that one variable is going to determine the success or failure of any particular player. Also, with such a large community behind League of Legends, predicting match outcomes for casual players and tournament games would be interesting and valuable for players and fans.

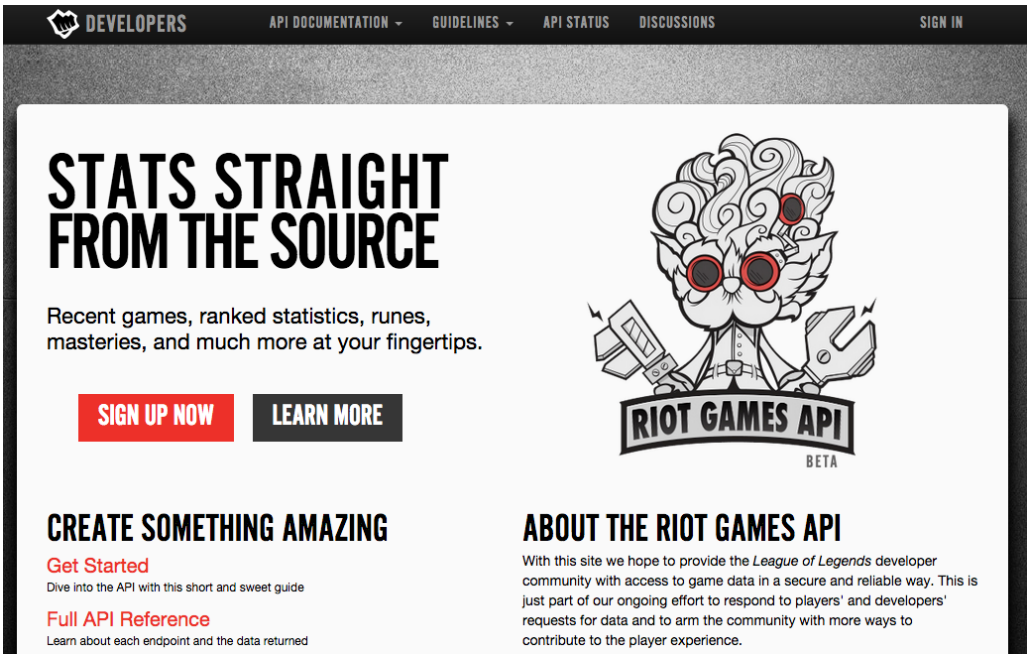
3.2 Game infos :

League of Legends is an intricate game that requires knowledge of many elements to be played proficiently. In the game, two teams of five players each battle against each other on a battlefield called Summoner's Rift. There are other battlefields with different objectives and layouts, but they will not be studied in our project. Summoner's Rift is comprised of each team's base, three lanes, and the jungle. The goal of League of Legends is to destroy the opposing team's Nexus, a structure located in the enemy base.

The "gold differential" is an example of a raw statistic that can provide a sense of the state of a match at a glance. Usually when a team has more gold than the other team, it means their champions are carrying more powerful items and are more likely to win if they get into a scrap. Combined with individual KDA (Kills, Deaths, Assists) numbers and the overall number of kills a team has accrued, it's an easy way to let the viewers know who is winning. When we turn on an LCS broadcast, gold differential is one of the first numbers we see at the top of the screen. But statistics like these don't tell the whole story.

3.3 Game API :

The collection of Data is made significantly easier when the game has an open API, or tools for scraping the game's code to find relevant data points. Riot Games provides a public API endpoint to access nearly all kinds of data that would be available to see in the official game client. The API gives access to match data which includes the champions selected and the roles of the players playing in the match and a lot of other useful information.



4 the Data

4.1 Data Collection :

Many websites present data gathered using the public Riot Games API , which allows collection of data about summoners (the LoL term for players), champions, and past matches. Some of these websites analyze the data or present it visually, while others just display the data itself. We chose one of those websites to get a file in .xlsx format containing all the data available for each game played during the 2017 season.

Remark : The downloaded dataset contained more than 75 variables, some of them were not meaningful for us. So we decided to get rid of those variables that won't be helpful in our analysis.

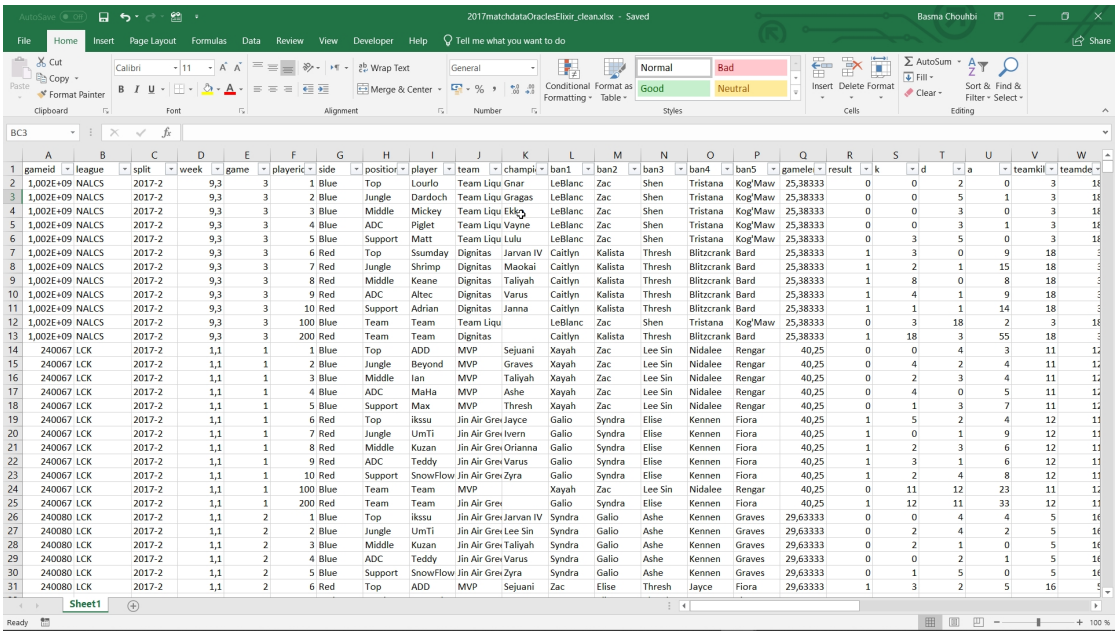


Figure 1.1: Dataset file in .xlsx format

So, we decided to delete 25 variables that won't be helpful in our prediction. Now, we have a dataset which contains 50 variables.

4.2 Data Dictionary :

In order to interpret downloaded match data file, we tried to make a sort of dictionary containing the variable names and descriptions below :

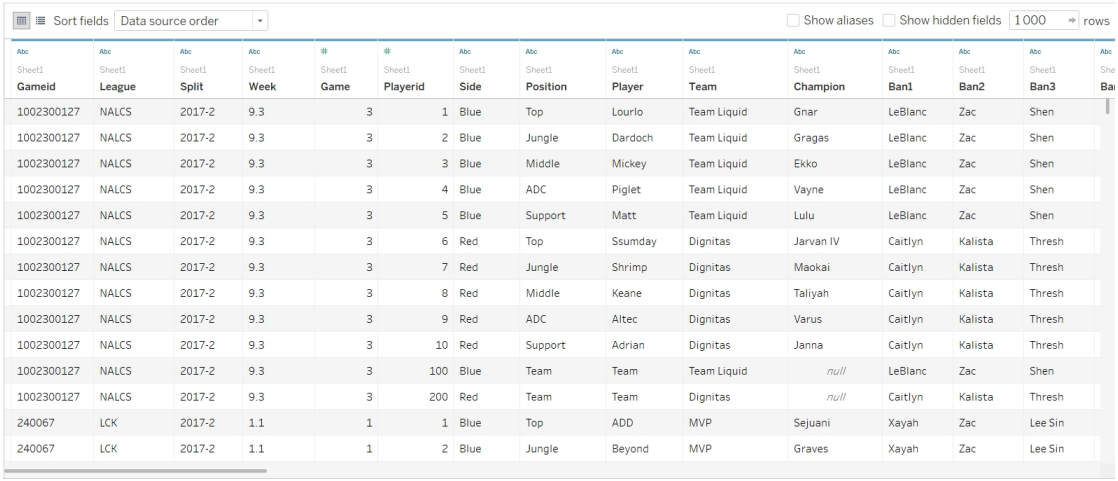
Variable	Description
gameid	Game identifier from Riot's server.
league	League
split	Time period covered, denoted by year and suffixes (1 spring, 2 summer, po playoffs, r regionals, w worlds)
week	"week within season," decimal, "day within week".
game	Game number within the series. T tiebreaker.
playerid	Player/team identifier, for differentiating map side and position.
side	Map side.
position	Player position.
player	Player name.
team	Team name.
champion	Champion name.
ban1	Team's first ban.
ban2	Team's second ban.
ban3	Team's third ban.
ban4	Team's fourth ban.
ban5	Team's fifth ban.
gamelength	Game length, in minutes. (Seconds measured in hundredths of a minute.)
result	Game result (1 win, 0 loss).
k	Total kills.
d	Total deaths.
a	Total assists.
teamkills	Total kills by team.
teamdeaths	Total deaths by team.
fb	First blood kill (1 yes, 0 no).
fbassist	First blood assist (1 yes, 0 no).
fbvictim	First blood victim (1 yes, 0 no).
fbtime	First blood time, in minutes. (Seconds measured in hundredths of a minute.)
kpm	Kills per minute (individuals and teams reported separately).
okpm	Opponent kills per minute (for players, reflects opponent in same position).
ckpm	Combined kills per minute (own kills and opponent kills combined).
fd	First dragon of game killed (1 yes, 0 no).

fdtime	First dragon time, in minutes. (Seconds measured in hundredths of a minute.)
teamdragkills	Total dragons killed by team.
oppdragkills	Total dragons killed by opposing team.
ft	First tower of game killed (1 yes, 0 no).
fttime	First tower kill time, in minutes. (Seconds measured in hundredths of a minute.)
teamtowerkills	Total towers killed by team.
opptowerkills	Total towers killed by opposing team.
dmgtochamps	Total damage dealt to champions.
dmgshare	Share of team's total damage dealt to champions.
earnedgoldshare	Share of team's total gold, with starting gold and inherent gold generation removed.
totalgold	Total gold earned from all sources.
goldspent	Total gold spent.
cspm	Creep score per minute. All creep score variables include minions and monsters.
goldat10	Total gold earned at 10:00.
oppgoldat10	Opponent's total gold earned at 10:00.
gdat10	Gold difference at 10:00.
goldat15	Total gold earned at 15:00.
oppgoldat15	Opponent's total gold earned at 15:00.
gdat15	Gold difference at 15:00.

Table 1.1: DATA DICTIONARY

4.3 Data Preview :

Here is a preview of the dataset we are analyzing using the software Tableau Desktop:



Sort fields: Data source order — Show aliases — Show hidden fields — 1 000 rows

Gameid	League	Split	Week	Game	Playerid	Side	Position	Player	Team	Champion	Ban1	Ban2	Ban3
1002300127	NALCS	2017-2	9.3	3	1	Blue	Top	Lourlo	Team Liquid	Gnar	LeBlanc	Zac	Shen
1002300127	NALCS	2017-2	9.3	3	2	Blue	Jungle	Dardoch	Team Liquid	Gragas	LeBlanc	Zac	Shen
1002300127	NALCS	2017-2	9.3	3	3	Blue	Middle	Mickey	Team Liquid	Ekko	LeBlanc	Zac	Shen
1002300127	NALCS	2017-2	9.3	3	4	Blue	ADC	Piglet	Team Liquid	Vayne	LeBlanc	Zac	Shen
1002300127	NALCS	2017-2	9.3	3	5	Blue	Support	Matt	Team Liquid	Lulu	LeBlanc	Zac	Shen
1002300127	NALCS	2017-2	9.3	3	6	Red	Top	Ssumday	Dignitas	Jarvan IV	Caitlyn	Kalista	Thresh
1002300127	NALCS	2017-2	9.3	3	7	Red	Jungle	Shrimp	Dignitas	Maokai	Caitlyn	Kalista	Thresh
1002300127	NALCS	2017-2	9.3	3	8	Red	Middle	Keane	Dignitas	Taliyah	Caitlyn	Kalista	Thresh
1002300127	NALCS	2017-2	9.3	3	9	Red	ADC	Altec	Dignitas	Varus	Caitlyn	Kalista	Thresh
1002300127	NALCS	2017-2	9.3	3	10	Red	Support	Adrian	Dignitas	Janna	Caitlyn	Kalista	Thresh
1002300127	NALCS	2017-2	9.3	3	100	Blue	Team	Team	Team Liquid	*null*	LeBlanc	Zac	Shen
1002300127	NALCS	2017-2	9.3	3	200	Red	Team	Team	Dignitas	*null*	Caitlyn	Kalista	Thresh
240067	LCK	2017-2	1.1	1	1	Blue	Top	ADD	MVP	Sejuani	Xayah	Zac	Lee Sin
240067	LCK	2017-2	1.1	1	2	Blue	Jungle	Beyond	MVP	Graves	Xayah	Zac	Lee Sin

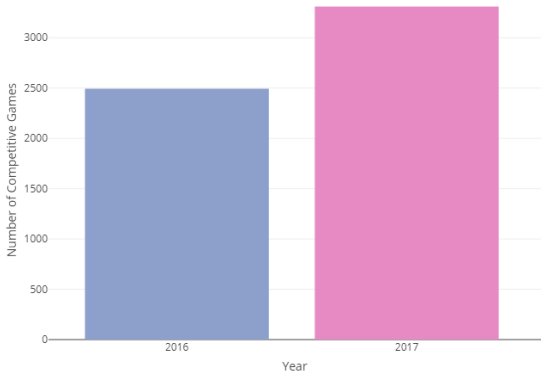
Chapter 2

Analysis

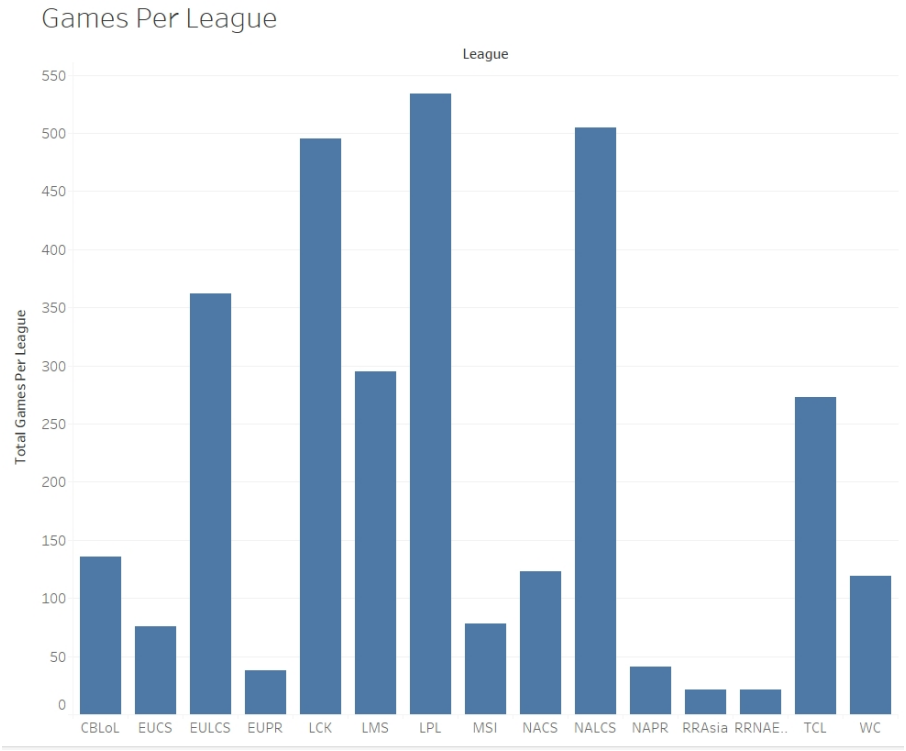
1 First Analysis

1.1 League of Legends' Evolution :

First of all, we tried to illustrate the impressive growth of the game by comparing the number of competitive games in our data with the previous year 2016.



In 2016, 2429 games take place. After 2016. League continues to grow with a substantial pace, reaching the unbelievable 3117 professional competitive games in 2017.




```
table(data_lol$league)/12
```

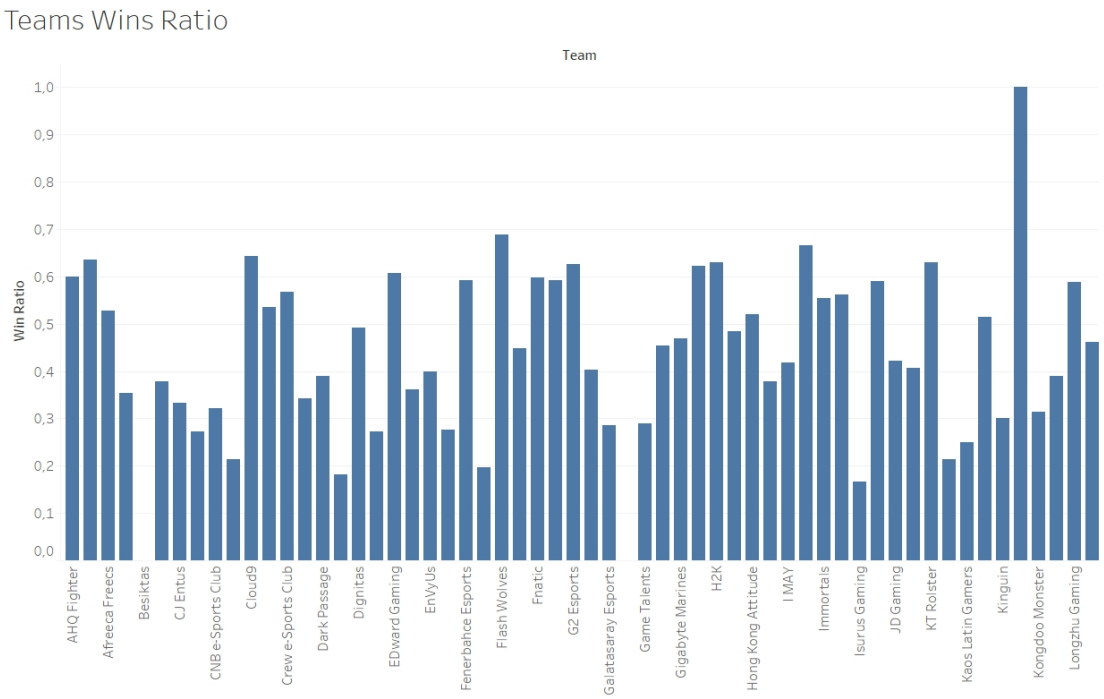
```
CBLoL   EUCS  EULCS   EUPR    LCK    LMS    LPL    MSI   NACS  NALCS   NAPR RRAsia RRNAEU    TCL
   136     76    362     38    495    295    534     78    123    505     41     21     21    273
    WC
   119
```

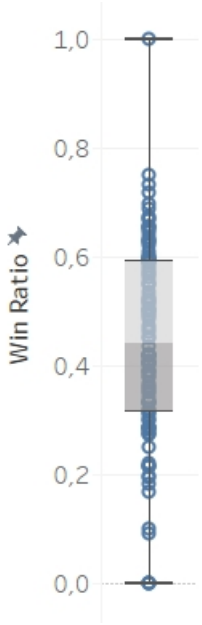
We can also deduce that LPL, NALCS, LCK and EULCS have the most games with more than 350 games each.

1.2 Teams :

Calculating the winning percentage is equivalent to estimating a proportion of wins in total number of games.

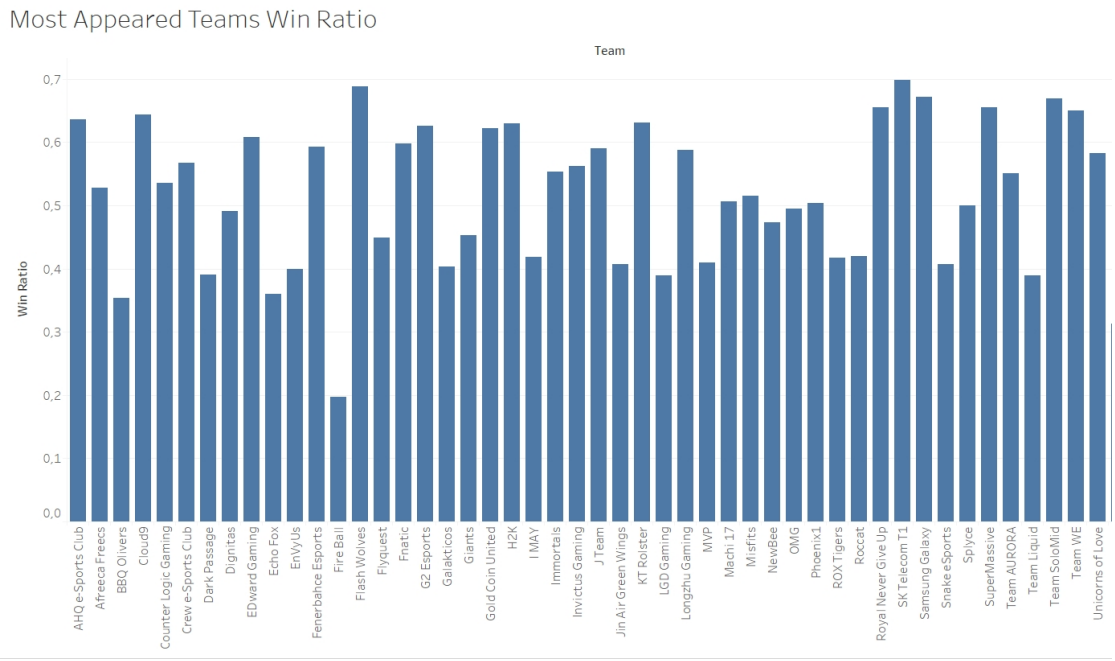
As we can see below, there are teams who have 0% and 100% as a win ratio, and this is not coherent. So we decided to plot the Boxplot and we found out that there are some points that are expanding its extremities.





The problem is that those teams have played just few games, some of them have even played just one game.

In order to overcome that problem, we need to calculate the winning percentage just for the "Most Appeared Team" who played more than 60 games in different leagues.



As we can see, SK Telecom T1, Flash Wolves and Samsung Galaxy possess the highest win rate. Now let's make sure that we have the same ratio using R, we first calculate the wins & loses for every team by typing the command :

```
table(data_lol$result,data_lol$team)/6
```

```
  ProGaming Esports QG Reapers Raise Gaming Rampage Red Bulls Red Canids RED Canids
0                 6         19           23       9         7          6          7
1                 8         23           25       1         7         21         12

  Remo Brave eSports Roccat ROX Tigers Royal Never Give Up Samsung Galaxy Schalke 04
0                 11     36         53                  39             39         11
1                  3     26         38                  74             80         30

  SK Telecom T1 Snake eSports Snake Esports Splyce Suning Gaming SuperMassive T Show E-Sports
0            42            21            27     35            25            29              14
1            97            17            16     35            18            55               0

  Team AURORA Team Gates Team Liquid Team oNe e-Sports Team oNe Esports Team SoloMid Team WE
0          31         10          66                 8                6           45      50
1          38          0          42                15                3           91      93
```

We transformed the result to a data frame in order to plot it and get the average win ratio.

```
Teams_games=as.data.frame(table(data_lol$result,data_lol$team)/6)
```

	Var1	Var2	Freq
143	0	NewBee	27
144	1	NewBee	21
145	0	Ninjas in Pyjamas	28
146	1	Ninjas in Pyjamas	11
147	0	OMG	51
148	1	OMG	50
149	0	Operation Kino	10
150	1	Operation Kino	4
151	0	Origen	40
152	1	Origen	4

We divided it into two data frames : Wins and Loses

```
wins=subset(Teams_games, Teams_games$Var1==1)
lose=subset(Teams_games, Teams_games$Var1==0)
```

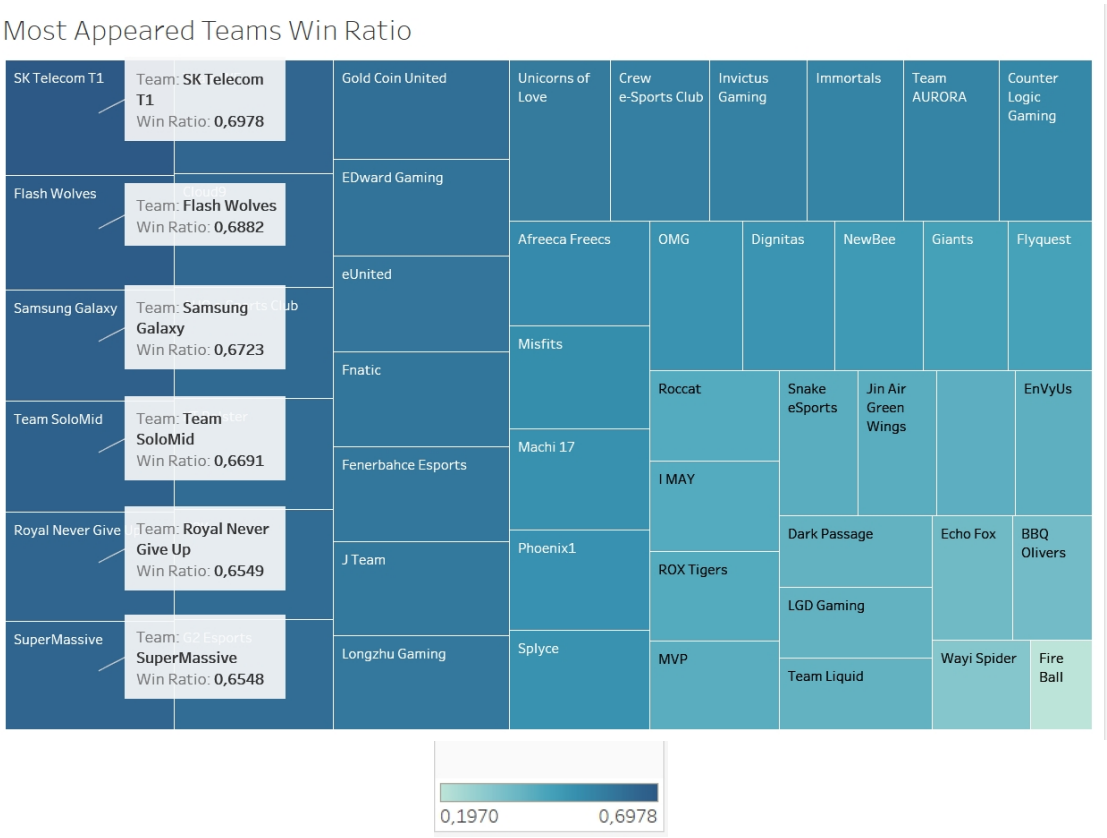
Then, we calculated the win ratio for each team individually and we found the same results.

```
winrate = wins$Freq/(wins$Freq + lose$Freq)
```

```
> winrate
    [1] 0.3333333 0.5277778 0.6363636 0.6000000 0.3535354 0.3783784 0.2142857 0.3333333 0.2727273
 [10] 0.6434109 0.3214286 0.5350877 0.5675676 0.3421053 0.3906250 0.4910714 0.2727273 0.3604651
 [19] 0.6074766 0.4000000 0.6000000 0.2765957 0.2222222 0.5930233 0.1969697 0.6881720 0.4821429
 [28] 0.4117647 0.5982143 0.5925926 0.6260870 0.4032258 0.2857143 0.2894737 0.4531250 0.2857143
 [37] 0.5200000 0.6216216 0.6296296 0.5200000 0.3793103 0.4848485 0.3421053 0.4791667 0.5533981
 [46] 0.6666667 0.5619048 0.5897436 0.4210526 0.4069767 0.2142857 0.2500000 0.5142857 0.3000000
 [55] 0.3137255 0.6306306 0.3896104 0.5882353 0.4615385 0.5066667 0.3000000 0.5157895 0.5909091
 [64] 0.4100000 0.1970000 0.5111111 0.4375000 0.2820513 0.4950495 0.2857143 0.4358974 0.5750000
 [73] 0.3600000 0.5042735 0.5714286 0.5476190 0.5208333 0.5000000 0.6315789 0.2142857 0.4193548
 [82] 0.4175824 0.6548673 0.6722689 0.6978417 0.4473684 0.3720930 0.5000000 0.4186047 0.6547619
 [91] 0.5507246 0.3888889 0.6521739 0.3333333 0.6691176 0.6503497 0.5909091 0.5824176 0.3513514
[100] 0.3333333 0.3684211 0.3134328 0.3076923 0.2857143
```

```
summary(winrate)
```

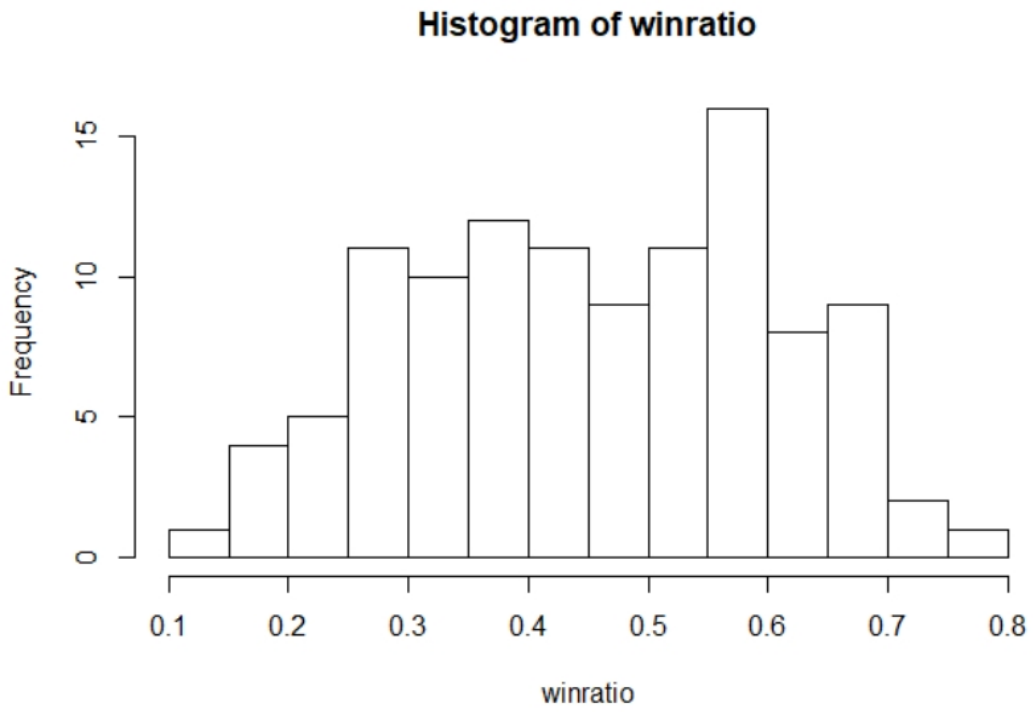
```
> summary(winrate)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1970  0.3399  0.4573  0.4562  0.5839  0.6978
```



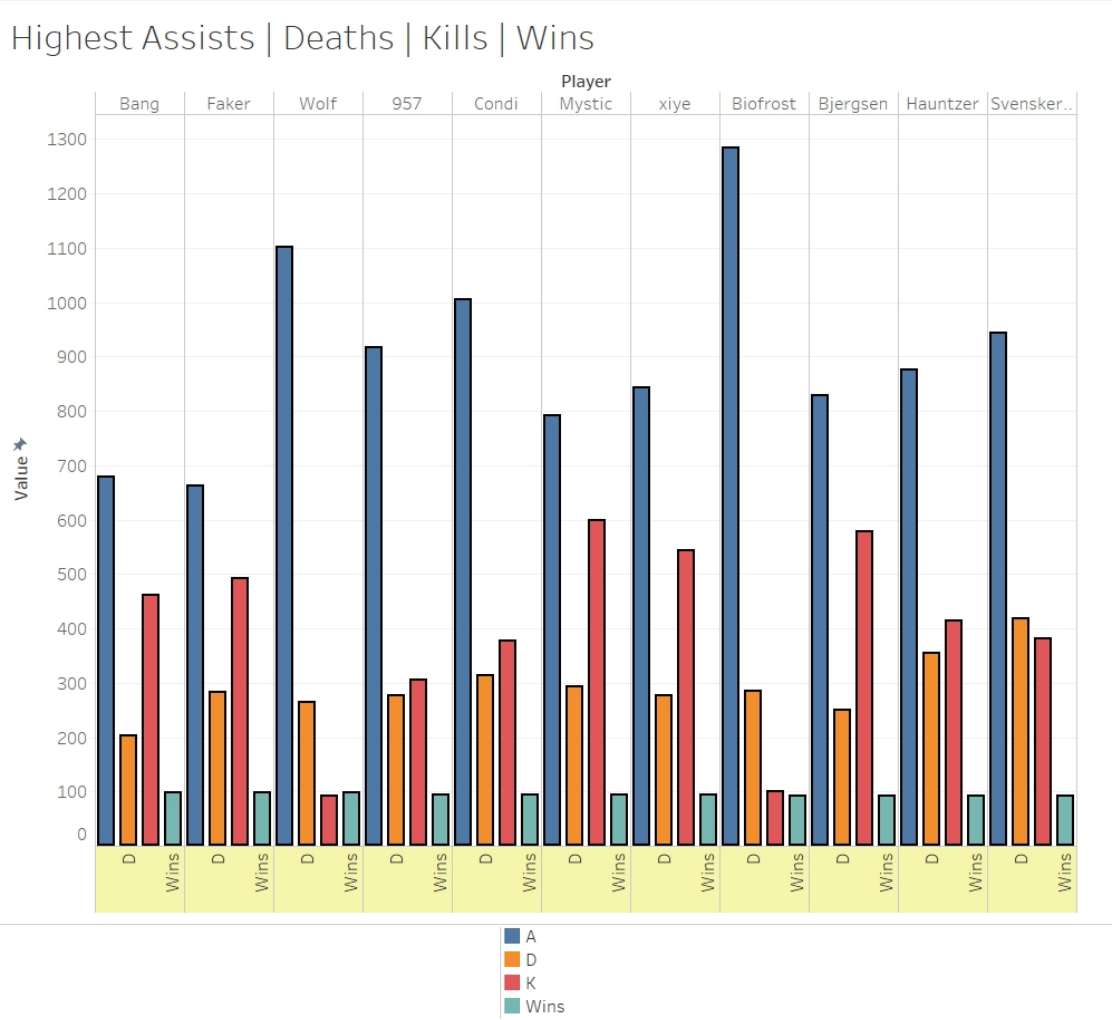
Thus, we conclude that:

- The average Win Ratio for a Pro Team is 45%.
- Team WE is the most appeared team in all the Leagues with 143 games
- SK Telecom T1 has the highest Win Ratio. Out of 139 games they managed to win 97 leading to an outstanding 69,78% Win Ratio
- Flash Wolves have 93 games winning 64 (68,82%)
- Sumsung Galaxy come third in winrate. They managed to win 80 games leading to a 67,23% win ratio

Histogram of Teams win rate :



1.3 Players :



- **Mystic(598), Bjergsen(577) Xiye(541)** are the players with the most kills in competitive League.

- **Biofrost(1282), Wolf(1099) Condi(1004)** are the players with the most assists in competitive League.

The ratio of kills, deaths and assists in the game is called **KDA** (kills, deaths, assists) ratio.

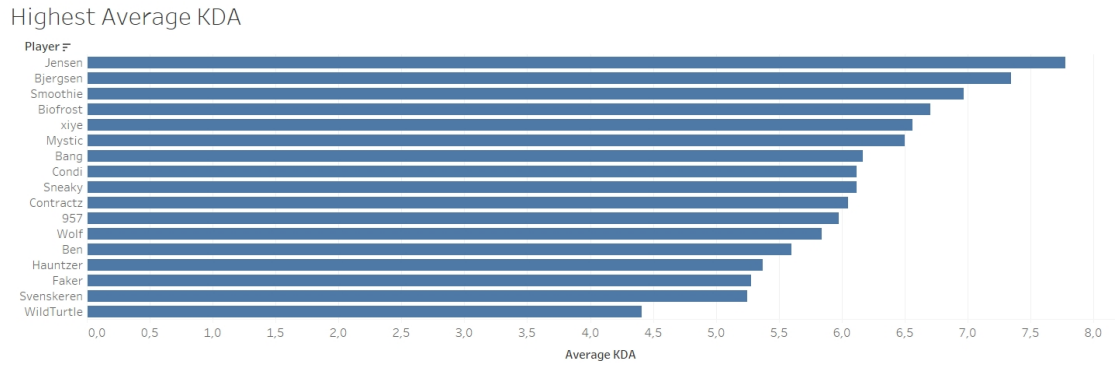
Formula:

KDA Ratio = (K+A) / Max(1,D)

Where: K = Kills | D = Deaths | A = Assists

So, we decided to add a new column in our dataset for the KDA ratio.

```
data_lol$KDA = (data_lol$k+data_lol$a)/max(1,data_lol$d)
```



- The average KDA for a pro player is 3,74
- Jensen Bjergsen are 2 of the players with the highest KDA in competitive Leagues having 7,78 7,34 respectively

1.4 Champions :

From each game we were then able look at data for each champion involved. Within a game, ten champions can be picked, ten champions can be banned, and five champions can win.

1.4.1 Champions ban rate, pick rate and win rate

With the sample of games, we calculated win rate, pick rate, and ban rate for each champion in 2017 season.

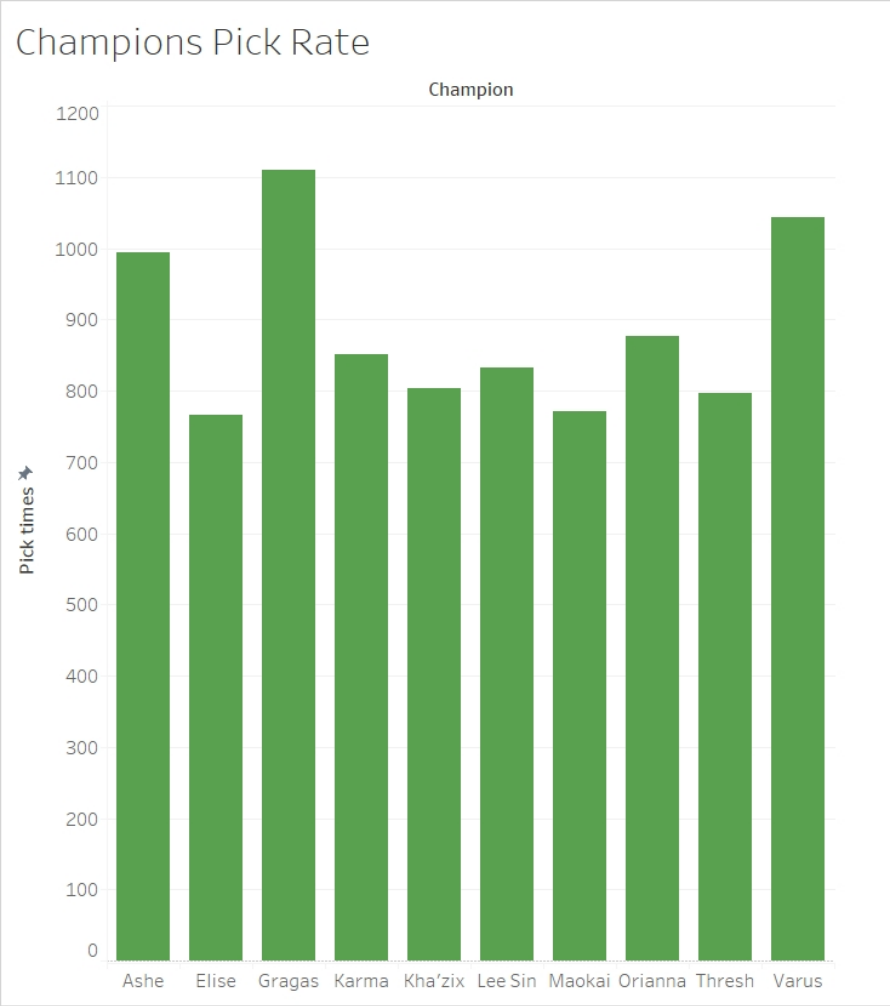
First, we used the summary to know how many times every champion appeared in the five ban columns.

```
   ban1            ban2            ban3            ban4            ban5
LeBlanc : 3372   Camille: 2232   LeBlanc: 2262   Syndra    : 1866   Syndra    : 1938
Zac     : 3282   LeBlanc: 2202   Elise  : 2232   Cassiopeia: 1578   Orianna   : 1500
Camille : 1968   Elise  : 2040   Camille: 1896   Shen      : 1446   Cassiopeia: 1278
Kalista : 1650   Rengar : 1896   Rengar : 1824   LeBlanc   : 1344   Shen      : 1242
Malzahar: 1530   Caitlyn: 1860   Caitlyn: 1572   Orianna   : 1320   LeBlanc   : 1116
Elise   : 1500   Galio  : 1800   Zac    : 1572   Fiora     : 1182   Taliyah   : 1098
(Other) :24102   (Other):25374   (Other):26046   (Other)   :28668   (Other)   :29232
```

In fact, for one single ban the name of the champion is repeated 6 times in the column of our dataset. So, in order to get the total of bans for every champion, we need to divide by 6 the number of appearances in every column and then sum all the values.

==> In 2017 Leblanc was the most banned champion with a total of 1716. Next up in times banned are : Camille (1216), Syndra (1201) Zac (1134)

Thus, we looked for the top 10 champions with the highest pick rate :



This data is a representation of champion performance. Win rates typically are associated with the actual strength and performance of a champion. Pick and ban rates typically are associated with how popular a champion is or strong it is perceived to be.
These statistics were essential for the final analysis as they represent a champion's *"success"*.

1.4.2 Average Damage Dealt by Champion

```
tapply(data_lol$dmgtochamps, data_lol$champion, mean)
```

```
     Aatrox        Ahri        Akali      Alistar        Amumu       Anivia
NA 16505.667    20006.487    14538.417     4355.214     7891.000    23348.308
```

Annie	Ashe	Aurelion Sol	Azir	Bard	Blitzcrank	Brand
8362.000	16343.566	20947.674	20256.175	8098.687	5373.529	16676.593
Braum	Caitlyn	Camille	Cassiopeia	Cho'gath	Corki	Darius
5076.226	19564.395	14350.546	17495.448	13349.190	24687.020	14802.600
Dr. Mundo	Draven	Ekko	Elise	Evelynn	Ezreal	Fiddlesticks
15462.000	17059.667	16232.538	11912.544	17734.000	22486.568	8088.400
Fiora	Fizz	Galio	Gangplank	Gnar	Gragas	Graves
16330.289	14742.571	12386.418	22629.426	16914.846	9980.142	13652.594
Hecarim	Heimerdinger	Illaoi	Irelia	Ivern	Janna	Jarvan IV
7764.200	13564.000	18794.733	13531.636	5894.161	2433.325	11175.449
Jax	Jayce	Jhin	Jinx	Kalista	Karma	Karthus
13653.054	21212.869	18587.658	23099.706	15965.014	7634.178	29267.250
Kassadin	Katarina	Kayle	Kayn	Kennen	Kha'zix	Kindred
18304.915	16487.000	10546.000	11351.316	15629.631	12340.819	8460.400
Kled	Kog'Maw	LeBlanc	Lee Sin	Leona	Lissandra	Lucian
14204.989	24034.231	19054.063	8344.196	4048.091	18560.200	18366.132
Lulu	Lux	Malphite	Malzahar	Maokai	Miss Fortune	Mordekaiser
4624.349	8546.286	10428.333	11476.878	13580.775	15767.958	15491.750
Morgana	Nami	Nasus	Nautilus	Nidalee	Nocturne	Nunu
7765.840	5223.055	17768.250	12298.204	13586.615	10918.500	5063.200
Olaf	Orianna	Pantheon	Poppy	Quinn	Rakan	Rek'Sai
10937.162	18540.353	11970.333	13311.922	16903.500	5051.891	8383.144
Renekton	Rengar	Riven	Rumble	Ryze	Sejuani	Shaco
13083.664	10528.291	18641.000	20894.149	14876.582	8683.357	9303.500
Shen	Shyvana	Singed	Sion	Sivir	Skarner	Sona
11772.182	9673.000	12687.944	14198.955	21642.813	6787.000	5524.583
Soraka	Swain	Syndra	Tahm Kench	Taliyah	Talon	Taric
3462.583	20617.893	20227.258	4412.194	21738.010	17511.091	2647.119
Teemo	Thresh	Tristana	Trundle	Twisted Fate	Twitch	Urgot
12232.000	4391.714	20844.991	10123.540	18359.167	21778.046	11780.250
Varus	Vayne	Veigar	Vel'koz	Vi	Viktor	Vladimir
19439.461	19575.588	4033.000	25968.714	8763.400	22766.717	18263.693
Warwick	Wukong	Xayah	Xerath	Xin Zhao	Yasuo	Yorick
7219.706	8649.000	20781.939	35734.833	4112.000	14175.800	25754.500
Zac	Zed	Ziggs	Zilean	Zyra		
7109.671	18814.540	24513.574	14373.018	13949.654		

==> Using a plot from Tableau here

1.5 Red or blue side :

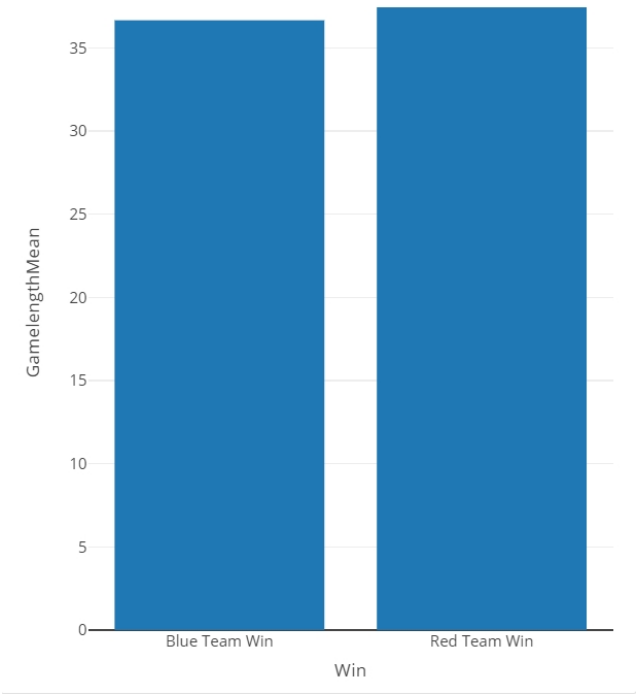
Does Blue team has an advantage in winning against the Red Team?

This has been debated for numerous times through the years. We can see that the Blue - Red Team win percentage is steady around 55%-45% and some years a bit more, in favor of the Blue Team.

```
tapply(data_lol$result, data_lol$side, sum)/6
```

```
> tapply(data_lol$result, data_lol$side, sum)/6
Blue  Red
1783 1334
```

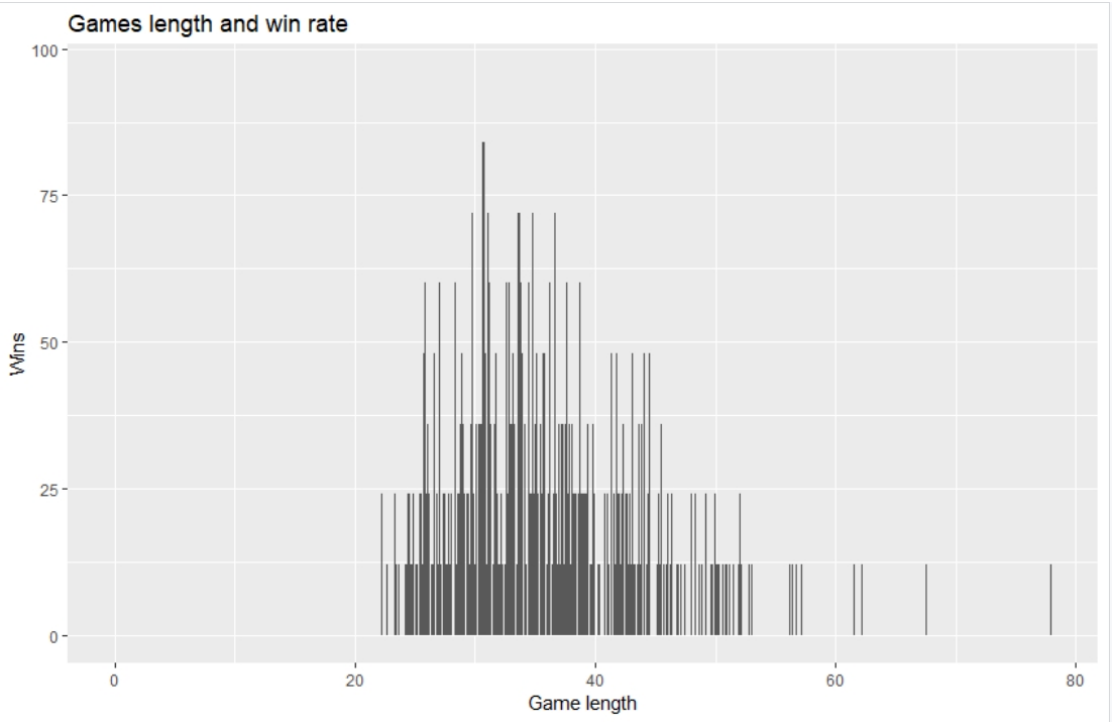
The average game length on a Blue Team Win is around 36,4 mins but when Red team wins the average game length is around 37,3 mins.



In conclusion, we can say that when Red team wins, games tend to have bigger duration than when Blue Team wins.

2 Game length and win rate

Additionally, we want to know if the game length is an important factor to win the game. As shown in the chart below, the win rate is apparently lower after 40 minutes, and when it was before 23 minutes, the win rate is zero. This gives us an idea that most of the games won have between 25 and 40 minutes.



3 Gold and Damage

Meanwhile, we also tried to use the linear regression method to see the correlation between Gold earned and Damage made. As shown below, there is a strong linear correlation between Gold and Damage, which means that players make good use of the gold earned in the games, and convert the gold into damage to the enemies.

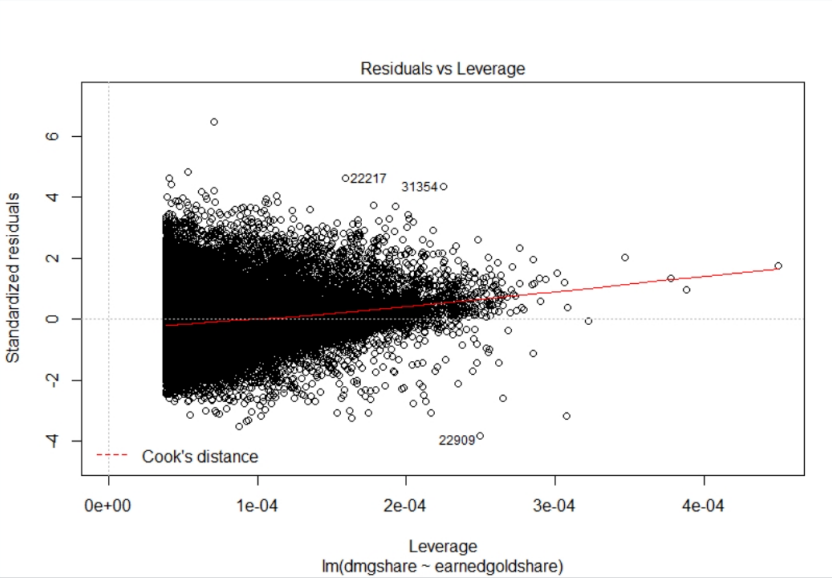
```
cor(data_lol$dmgshare, data_lol$earnedgoldshare)
```

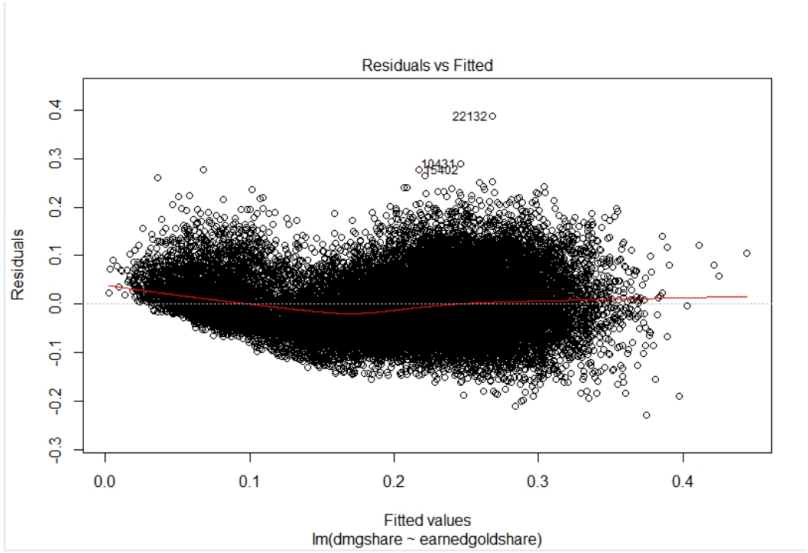
```
[1] 0.7812287
```

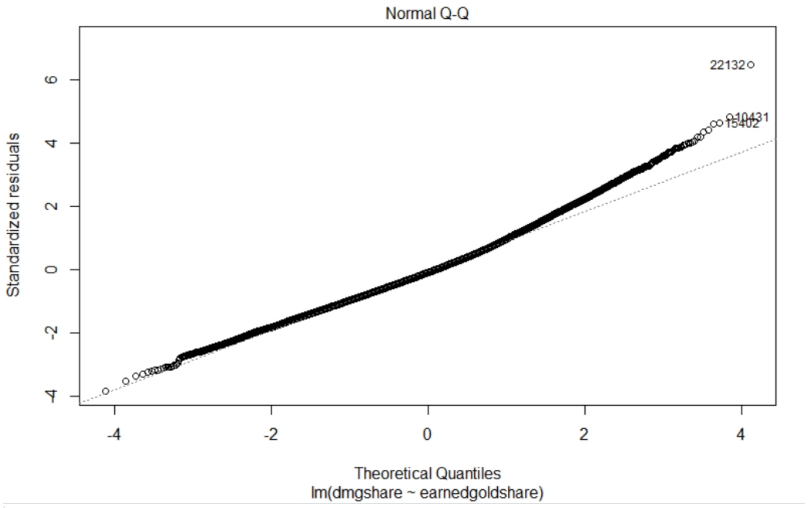
```
model_1 <- lm(dmgshare ~ earnedgoldshare, data = data_lol)
summary(model_1)
```

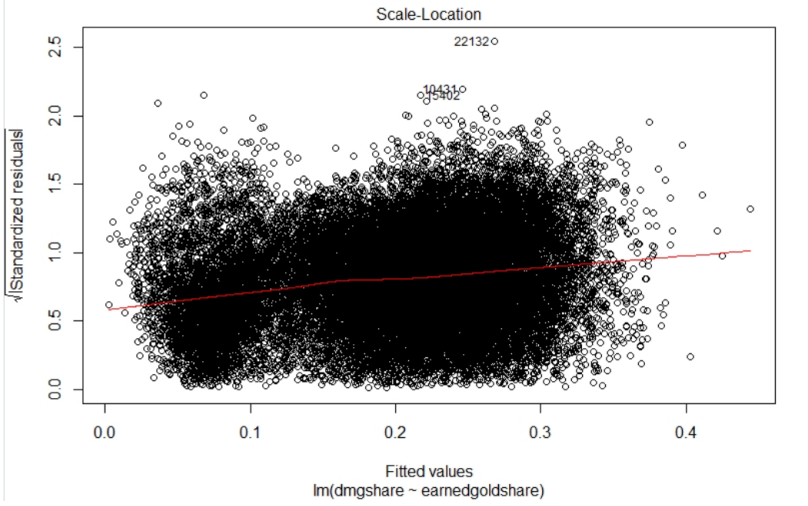
```
Call: lm(formula = dmgshare ~ earnedgoldshare, data = data_lol)
Residuals:
     Min       1Q   Median       3Q      Max
-0.22815 -0.04071 -0.00477  0.03508  0.38724
Coefficients:
                             Estimate Std. Error t value Pr(>|t|)
(Intercept)                -0.090846   0.001493  -60.84   <2e-16 ***
data_rowcsv$earnedgoldshare  1.454231   0.007230  201.13   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 0.0598 on 25828 degrees of freedom
  (11574 observations deleted due to missingness)
Multiple R-squared:  0.6103,Adjusted R-squared:  0.6103
F-statistic: 4.045e+04 on 1 and 25828 DF,  p-value: < 2.2e-16
```

```
plot(model_1)
```




Residuals vs Fitted
22132
10431
15402
Residuals
-0.3
-0.2
-0.1
0.0
0.1
0.2
0.3
0.4
0.0
0.1
0.2
0.3
0.4
Fitted values
lm(dmgshare ~ earnedgoldshare)


Normal Q-Q
22132
10431
15402
Standardized residuals
-4
-2
0
2
4
6
-4
-2
0
2
4
Theoretical Quantiles
lm(dmgshare ~ earnedgoldshare)


Scale-Location
22132
10431
15402
√|Standardized residuals|
0.0
0.5
1.0
1.5
2.0
2.5
0.0
0.1
0.2
0.3
0.4
Fitted values
lm(dmgshare ~ earnedgoldshare)

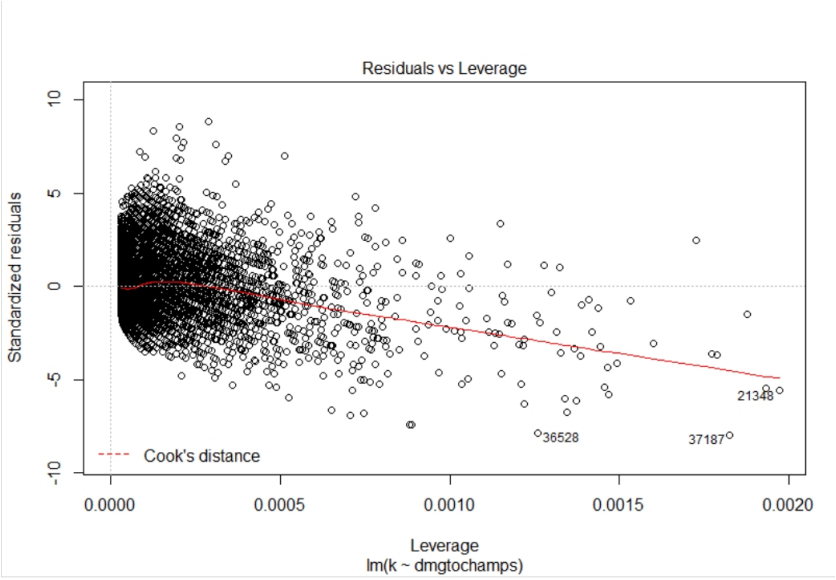
4 Damage to champions and Kills

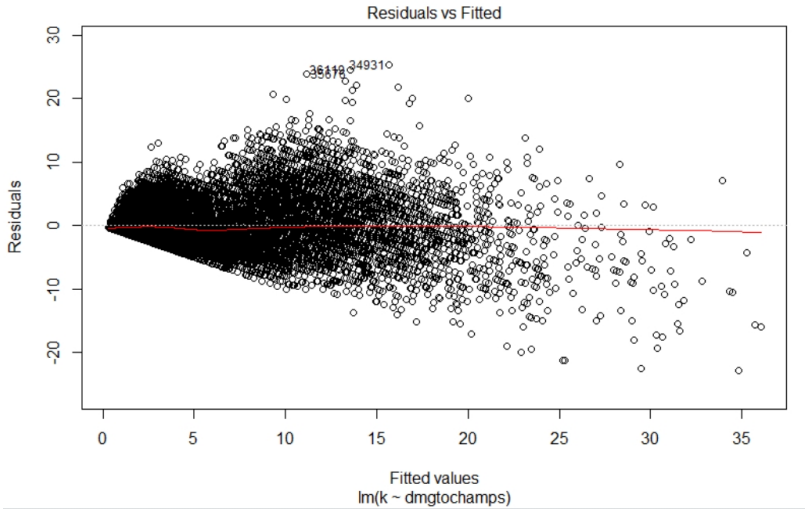
Moreover, we wanted to see if the *Damage To Champions* had a strong correlation with the Kills. Thus, we used the linear regression method to see the relationship between Kills and DTC. As we expected, DTC did have a strong correlation with kills. As more damage made would result in more kills, the game seemed fair and encouraging for players.

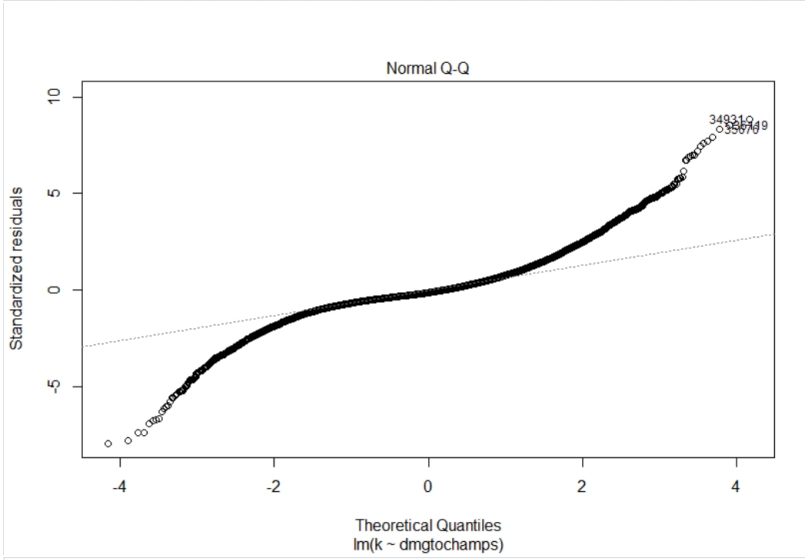
```
model_2 <- lm(k ~ dmgtochamps, data = data_lol)
summary(model_2)
```

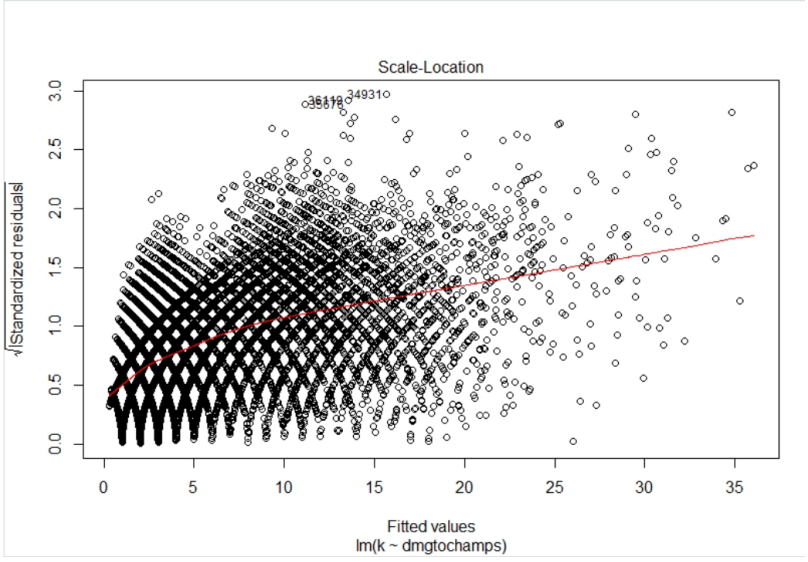
```
Call: lm(formula = k ~ dmgtochamps, data = data_lol)
Residuals:
     Min       1Q   Median       3Q      Max
-22.8057  -1.3960  -0.4151   1.1231  25.3133
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.520e-01  2.215e-02   11.38   <2e-16 ***
dmgtochamps 1.623e-04  6.415e-07  253.05   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 2.87 on 30994 degrees of freedom
  (6408 observations deleted due to missingness)
Multiple R-squared:  0.6738,Adjusted R-squared:  0.6738
F-statistic: 6.403e+04 on 1 and 30994 DF,  p-value: < 2.2e-16
```

```
plot(model_2)
```




Residuals vs Fitted
Residuals
Fitted values
lm(k ~ dmgtochamps)


Normal Q-Q
Standardized residuals
Theoretical Quantiles
lm(k ~ dmgtochamps)


Scale-Location
√|Standardized residuals|
Fitted values
lm(k ~ dmgtochamps)

5 Winner Prediction

Can we predict a game's result?

Actually, using only prematch knowledge (champions, masteries, roles, spells) from the very start is only a weak predictor of match outcome but using in-game statistics, the model becomes a strong predictor. That's why we will try to build our model based on in-game statistics.

In fact, we're not sure which of our variables are useful in predicting a result outcome. In fact, it's often helpful to build bivariate models, which are models that predict the outcome using a single independent variable. But the problem is which of the 28 variables is a significant predictor of the Result variable in a bivariate logistic regression model?

In order to avoid building 28 different models, we will use the correlation function to get an idea about the variables that are highly correlated with the variable Result.

```
#Removing Rows with all NAs (missing values) in the dataframe
data_clean=data_lol[complete.cases(data_lol), ]

#Correlation with result
i1 <- sapply(data_clean, is.numeric)
y1 <- "result"
x1 <- setdiff(names(data_clean)[i1], y1)
cor(data_clean[x1], data_clean[[y1]])
```

```
                      [,1]
playerid       -1.163223e-01
k               3.410497e-01
d              -4.586080e-01
a               5.210516e-01
teamkills       6.216402e-01
teamdeaths     -6.207185e-01
fb              6.216006e-02
fbassist        6.582484e-02
fbvictim       -6.216006e-02
fd              1.010101e-01
teamdragkills   5.403406e-01
oppdragkills   -5.408755e-01
ft              3.426573e-01
teamtowerkills  8.800471e-01
opptowerkills  -8.800471e-01
dmgtochamps     1.341979e-01
dmgshare        9.272443e-08
earnedgoldshare 1.592324e-08
totalgold       2.973579e-01
goldspent       2.642095e-01
cspm            3.663903e-02
```

```
goldat10          1.670677e-01
oppgoldat10      -1.670677e-01
gdat10            2.212700e-01
goldat15          2.398132e-01
oppgoldat15      -2.398132e-01
gdat15            3.403898e-01
```

Now that we have prepared the dataset, we need to split it into a training and testing set. We set the random seed to 1 and use the sample.split function to select the 50% of observations for the training set (the dependent variable for sample.split is Result), and we name the data frames train and test.

```
# - SPLITTING INTO A TRAINING AND TESTING SET

set.seed(10)
library(caTools)
split = sample.split(data_clean$result, SplitRatio = 0.5)

train = subset(data_clean, split == TRUE)
test = subset(data_clean, split == FALSE)
```

Now, we use logistic regression trained on the training set in order to predict the dependent variable Result using all the independent variables.

To determine significance, we look at the stars in the summary output of the model. We define an independent variable as significant if there is at least one star at the end of the coefficients row for that variable (this is equivalent to the probability column having a value smaller than 0.05).

5.1 Logistic Regression Model

In fact, we need to build different models so as to find the best one with the highest accuracy. First, we will exclude the variables referring to stats at the end of the game, because it would be too late to predict a win in this case. Also, we will omit the variables related to individual stats since it is a team game and one player can't win by himself.

5.1.1 Variables inserted in the Model: Gold (goldat10, goldat15, oppgoldat10, oppgoldat15, gdat10, gdat15)

Since **gdat10 & gdat15** are correlated with **goldat10, goldat15, oppgoldat10 and oppgoldat15**, we have 2 options :

1. Using **goldat10, goldat15, oppgoldat10 and oppgoldat15** :

```
Model1 <- glm(result ~ goldat10+oppgoldat10+goldat15+oppgoldat15, data=train,
     family = binomial)
summary(Model1)
```

```
Call: glm(formula = result ~ goldat10 + oppgoldat10 + goldat15 + oppgoldat15,
    family = binomial, data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.3392  -1.0816  -0.3237   1.0801   2.3574

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  8.433e-03  1.099e-01   0.077    0.939
goldat10    -5.875e-04  9.867e-05  -5.954 2.61e-09 ***
oppgoldat10  6.278e-04  9.875e-05   6.358 2.04e-10 ***
goldat15     1.342e-03  6.296e-05  21.315  < 2e-16 ***
oppgoldat15 -1.371e-03  6.299e-05 -21.760  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 17842  on 12869  degrees of freedom
Residual deviance: 16199  on 12865  degrees of freedom
AIC: 16209
Number of Fisher Scoring iterations: 4
```

2. Using **gdat10 and gdat15** :

```
Model2 <- glm(result ~ gdat10+gdat15, data=train, family = binomial)
summary(Model2)
```

```
Call: glm(formula = result ~ gdat10 + gdat15, family = binomial, data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.3462  -1.0815  -0.3253   1.0810   2.3505

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.0053672  0.0187876  -0.286    0.775
```

```
gdat10      -0.0006076  0.0000715  -8.498   <2e-16 ***
gdat15       0.0013564  0.0000460  29.487   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 17842  on 12869  degrees of freedom
Residual deviance: 16199  on 12867  degrees of freedom
AIC: 16205
Number of Fisher Scoring iterations: 4
```

So, it is more convenient to work with the second model in order to make predictions. We also compute the confusion matrix using a threshold of 0,5.

```
testPredict = predict(Model2, type="response", newdata=test)
table(test$result, testPredict>0.5)
```

```
   FALSE TRUE
0  4196 2228
1  2291 4155
```

Accuracy : 0.64887

We can deduce that the model has poor accuracy at the threshold 0,5.

5.1.2 Variables inserted in the Model: Kills (fb, teamkills, teamdeaths)

We use logistic regression trained on the training set to predict the dependent variable **result** using **fb, teamkills, teamdeaths** as independent variables.

```
Model3 <- glm(result ~ fb+teamkills+teamdeaths, data=train, family = binomial)
summary(Model3)
```

```
Call: glm(formula = result ~ fb + teamkills + teamdeaths, family = binomial,
    data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-4.2402  -0.1779  -0.0029   0.1696   4.1743

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.02418    0.10960  -0.221  0.82540
fb           0.34567    0.13339   2.591  0.00956 **
teamkills    0.52396    0.01092  47.997  < 2e-16 ***
teamdeaths  -0.51732    0.01085 -47.680  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As we can see, the variable **teamkills** has a positive coefficient (meaning that higher values of the variable lead to an increased chance of winning) and **teamdeaths** has a negative coefficient (meaning that higher values of the variable lead to a decreased chance of winning).

```
testPredict = predict(Model3, type="response", newdata=test)
table(test$result, testPredict>0.5)
```

```
   FALSE TRUE
  0  6016  408
  1   300 6146
```

Accuracy : 0.94499

We try with threshold of 0,85 :

```
table(test$result, testPredict>0.85)
```

```
   FALSE TRUE
  0  6307  117
  1  1092 5354
```

Accuracy : 0.90606

We can deduce that the model keep its strong accuracy even at the threshold 0,85, and by increasing the threshold we make more false positives mistakes.

5.1.3 Variables inserted in the Model: Towers (ft, teamtowerkills, opptowerkills)

We use logistic regression trained on the training set to predict the dependent variable **result** using **ft, teamtowerkills, opptowerkills** as independent variables.

```
Model4 <- glm(result ~ ft+teamtowerkills+opptowerkills, data=train, family =
    binomial)
summary(Model4)
```

```
Call:
glm(formula = result ~ ft + teamtowerkills + opptowerkills, family = binomial,
    data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.3041  -0.0481  -0.0063   0.0338   3.3032

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
```

```
(Intercept)     0.60981    0.40526   1.505    0.132
ft              1.29441    0.14349   9.021   <2e-16 ***
teamtowerkills  1.01909    0.03825  26.641   <2e-16 ***
opptowerkills  -1.01422    0.03895 -26.042   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
testPredict = predict(Model4, type="response", newdata=test)
table(test$result, testPredict>0.5)
```

```
    FALSE TRUE
  0  6273  151
  1   146 6300
```

Accuracy : 0.97693

So, it’s logical that building a predictive model based on variables related to towers statistics is more reasonable, since achieving a victory condition, typically destroying the core building (the Nexus), teams need to bypass all the towers in a line.

5.1.4 Variables inserted in the Model: Dragons (fd, teamdragkills, oppdragkills)

We use logistic regression trained on the training set to predict the dependent variable **result** using **fd, teamdargkills, oppdragkills** as independent variables.

```
Model5 <- glm(result ~ fd+teamdragkills+oppdragkills, data=train, family =
    binomial)
summary(Model5)
```

```
Call:
glm(formula = result ~ fd + teamdragkills + oppdragkills, family = binomial,
    data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.9147  -0.6794  -0.1701   0.7627   2.9130

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)    0.09130    0.06627   1.378    0.168
fd             0.38604    0.07253   5.322 1.02e-07 ***
teamdragkills  0.69035    0.02214  31.177  < 2e-16 ***
oppdragkills  -0.71993    0.02233 -32.246  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
testPredict = predict(Model5, type="response", newdata=test)
table(test$result, testPredict>0.5)
```

```
    FALSE TRUE
  0  4756 1668
  1  1030 5416
```

Accuracy : 0.79037

5.1.5 Final Model with all significant variables

```
Model_final <- glm(result ~ gdat10+gdat15+fb+teamkills+teamdeaths+ft+
    teamtowerkills+opptowerkills+fd+teamdragkills+oppdragkills, data=train, family
     = binomial)
summary(Model_final)
```

```
Call:
glm(formula = result ~ gdat10 + gdat15 + fb + teamkills + teamdeaths +
    ft + teamtowerkills + opptowerkills + fd + teamdragkills +
    oppdragkills, family = binomial, data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.3770  -0.0189  -0.0010   0.0181   3.3640

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)     0.9241448  0.5267717   1.754  0.07937 .
gdat10          0.0003145  0.0003138   1.002  0.31637
gdat15         -0.0005429  0.0002070  -2.623  0.00871 **
fb             -0.5025296  0.2823763  -1.780  0.07513 .
teamkills       0.2768595  0.0182261  15.190  < 2e-16 ***
teamdeaths     -0.2684347  0.0180833 -14.844  < 2e-16 ***
ft             -1.7174641  0.1903946  -9.021  < 2e-16 ***
teamtowerkills  0.9032026  0.0497984  18.137  < 2e-16 ***
opptowerkills  -0.9241948  0.0509522 -18.138  < 2e-16 ***
fd              0.3033756  0.2680315   1.132  0.25769
teamdragkills   0.1530605  0.0812589   1.884  0.05962 .
oppdragkills   -0.1426903  0.0808219  -1.765  0.07748 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
testPredict = predict(Model_final, type="response", newdata=test)
table(test$result, testPredict>0.5)
```

```
    FALSE TRUE
  0  6318  106
  1    95 6351
```

Accuracy : 0.98438

Confusion Matrix Explained

- Correct : Predicted Red Team to win and actually Won : 6318 times
- Correct : Predicted Blue Team to win and actually Won : 6351 times
- Wrong: Predicted Red Team to win but Red team Lost : 95 times
- Wrong: Predicted Blue Team to win but Blue team Lost : 106 times

Note: We also tried to omit the insignificant variables and the model has shown to have similar performance without these variables (gdat10, fb, fd, teamdragkills, oppdragkills) with an accuracy of *0.98412*.

5.2 CART Model

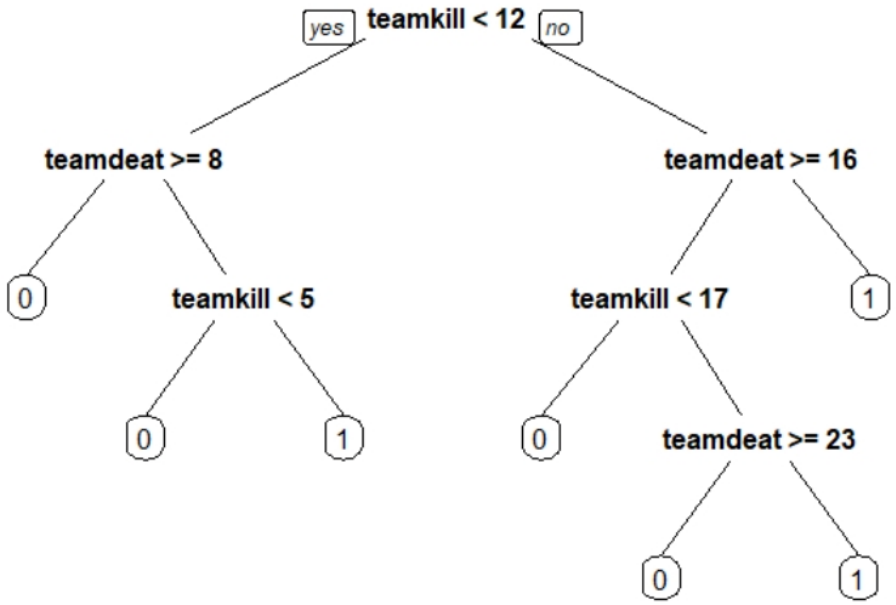
In each subset of a CART tree, we have a bucket of observations, which may contain both possible outcomes. In our case of study, we will be classifying observations as either win or lose, again a binary outcome.

5.2.1 Variables inserted in the Model: Kills (fb, teamkills, teamdeaths)

```
library(rpart)
library(rpart.plot)
LOLTree = rpart(result ~ fb+teamkills+teamdeaths, data = train, method="class",
    minbucket=50)
```

- method="class" arg tells us to make classification tree and not regression tree.
- We need to give type = "class" because we want the majority class predictions.This is like using a threshold of 0,5.
- minbucket = 50 limits the tree so that it does not overfit to our training set.

```
#The model can be represented as a decision tree:
prp(LOLTree)
```



Now lets assess the accuracy of the model through confusion matrix :

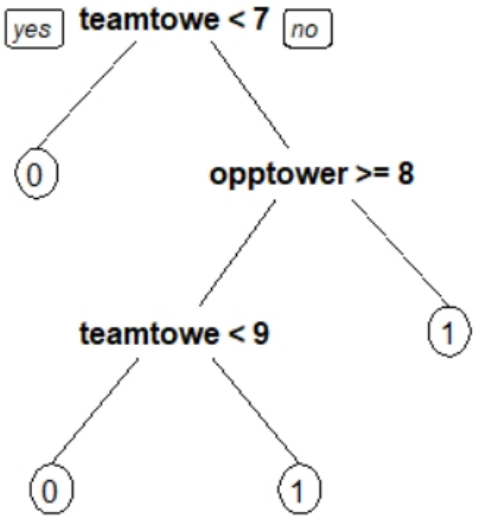
```
PredictCART1 = predict(LOLTree, newdata = test, type = "class")
cmat_CART<-table(test$result, PredictCART1)
cmat_CART
```

```
   PredictCART
    PredictCART
       0    1
  0 5930  494
  1  397 6049
```

Accuracy : 0.93076

5.2.2 Variables inserted in the Model: Towers (ft, teamtowerkills, opptowerkills)

```
library(rpart)
library(rpart.plot)
LOLTree = rpart(result ~ ft+teamtowerkills+opptowerkills, data = train, method="
    class", minbucket=150)
#Plotting the decision tree:
prp(LOLTree)
```



Now lets assess the accuracy of the model through confusion matrix :

```
PredictCART2 = predict(LOLTree, newdata = test, type = "class")
cmat_CART<-table(test$result, PredictCART2)
cmat_CART
```

```
   PredictCART
       0    1
  0 6218  206
  1  176 6270
```

Accuracy : 0.97031

5.3 ROC Curve

Lets evaluate the last CART model using the ROC curve :

```
library(ggplot2)
library(ROCR)

PredictROC = predict(LOLTree, newdata = test)
head(PredictROC)
```

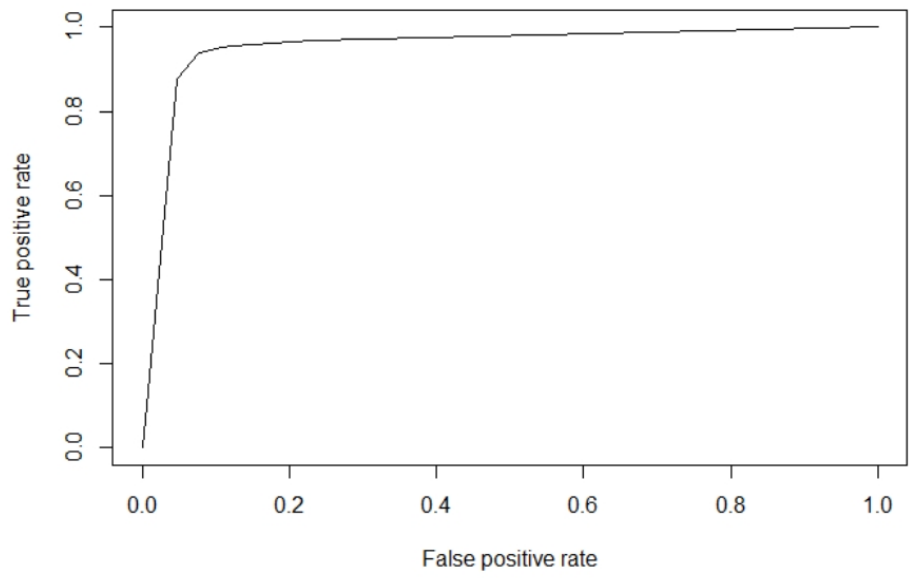
```
          0          1
1 0.9559329 0.04406715
3 0.9559329 0.04406715
4 0.9559329 0.04406715
5 0.9559329 0.04406715
6 0.0508982 0.94910180
9 0.0508982 0.94910180
```

For each observation in the test set, it gives two numbers which can be thought of as the probability of outcome 0 and the probability of outcome 1.
More concretely, each test set observation is classified into a subset, or bucket, of our CART tree. These numbers give the percentage of training set data in that subset with outcome 0 and the percentage of data in the training set in that subset with outcome 1. We will use the second column as our probabilities to generate an ROC curve.

```
pred = prediction(PredictROC[,2], test$result)
perf = performance(pred, "tpr", "fpr")
```

First we use the prediction() function with first argument the second column of PredictROC, and second argument the true outcome values, test$result. We pass the output of prediction() to performance() to which we give also two arguments for what we want on the X and Y axes of our ROC curve, true positive rate and false positive rate.

```
plot(perf)
```



Then, we compute the AUC of our CART model, using the following command in your R console:

```
as.numeric(performance(pred, "auc")@y.values)
```

```
as.numeric(performance(pred, "auc")@y.values)
[1] 0.9518861
```

5.4 Random Forest

```
library(randomForest)
LOLForest = randomForest(result ~ gdat15+teamkills+teamdeaths, data = train1,
    ntree=100, nodesize=25 )
```

```
Warning in randomForest.default(m, y, ...): The response has five or fewer unique values.
Are you sure you want to do regression?
```

So R shows a warning which requires that we convert the outcome variables as factors before building the Random Forest model.

```
# Convert outcome to factor
train1$result = as.factor(train$result)
test1$result = as.factor(test$result)

LOLForest = randomForest(result ~ ft+teamtowerkills+opptowerkills , data = train1,
     ntree=100, nodesize=25 )

# Make predictions
PredictForest = predict(LOLForest, newdata = test1)
table(test1$result, PredictForest)
```

```
   PredictForest
       0    1
  0 6245  179
  1  121 6325
```

Accuracy : 0.97669

6 Discussion:

In our study, we used data related to one patch of the game because Riot Games attempts to keep champion win rates as close to 50%. Any win rate higher than 50% implies that the champion is giving the player an advantage while any win rate lower than 50% implies the champion is giving the player a disadvantage. In order to maintain balance, Riot decides to respectively nerf and buff these champions. That means there is a strong correlation between the number of changes in the patch and the win rate.

Chapter 3

R Shiny Application

Shiny is an R package that makes it easy to build interactive web apps straight from R. This webinar will cover the basics of building and deploying a Shiny app, including essentials of reactive programming for building apps that are more efficient, robust, and correct.

Therefore, we decide to build our web application containing two part:

- First part using our best prediction model which has 98% as accuracy; you can fill a form and get the percentage of your win chance.
- Second one using riot games Api to compare League of Legends champion statistics. Two champions can be compared at a time, and patch version is also selectable.

There is a preview of our application:

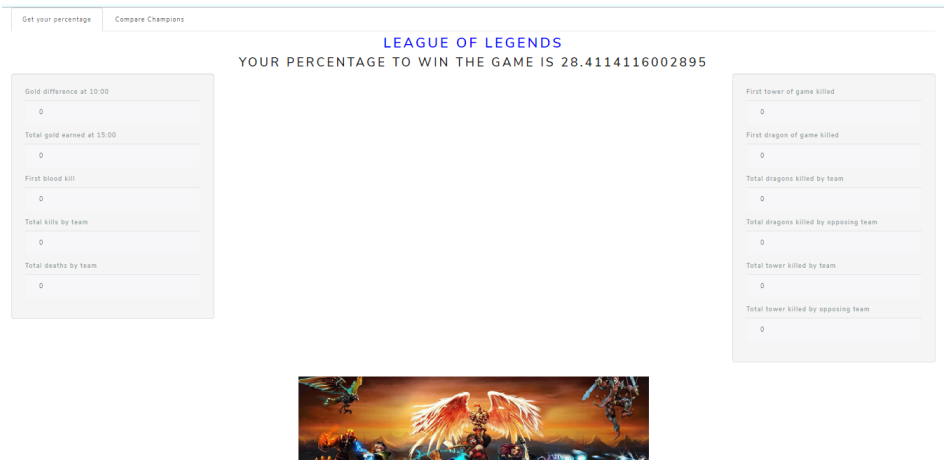


Figure 3.1: First tab: Predict wining

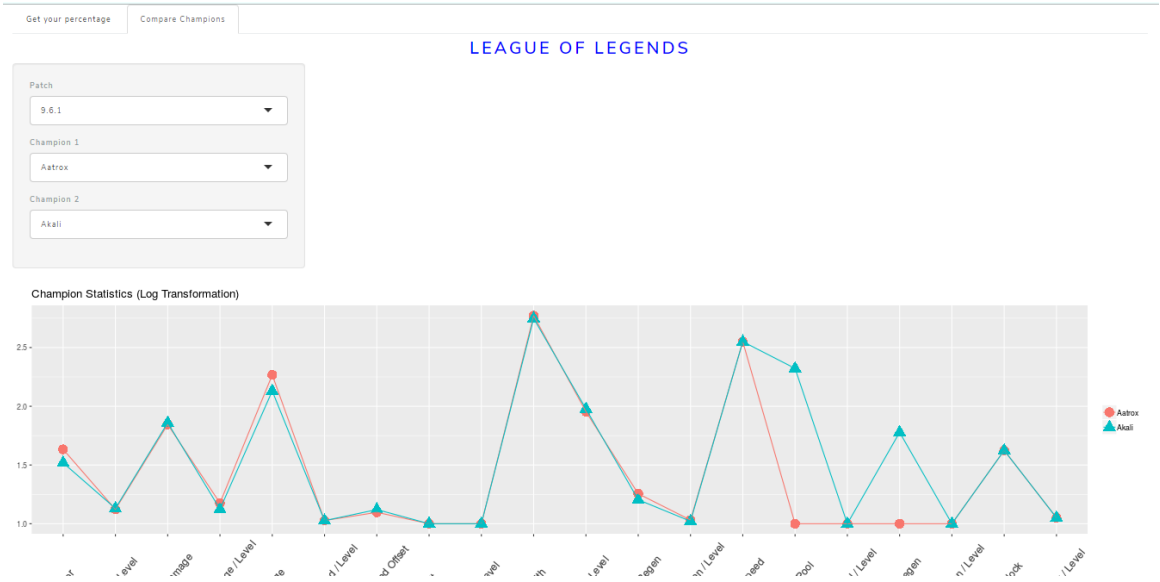


Figure 3.2: Second tab: Compare between two champions

You can visit our R shiny application from this website: https://nisrinehammout.shinyapps.io/Lol-Emines/

Chapter 4

Conclusion

We hope that we'll be able to look at more current data in the future, but this is a great demonstration of how data can power anyone from normal players, to Riot Games professional broadcasts, and even teams looking for ways to best analyze the strengths and weaknesses of enemy teams to make informed decisions. We can even use data analysis to improve matchmaking algorithms and game balance to the benefit of everyone that plays League of Legends around the world.

Despite there being many existing sites that mine data through the Riot API and release interesting, personalized statistics, none can effectively predict the winner team during a game. Our project takes on this task using different variables that are related to both teams stats in order to predict the winner.

In the future, it would be interesting to develop a finer sense of how accurate the prediction system becomes given the amount of data at a certain point in a match. In other words, how much more accurate would the prediction becomes if data extracted from the first X minutes of each match is used as part of the feature vector as compared to the first Y minutes as well as how the accuracy changes with respect to X.

We also have the data to be able to implement many more features if we desired.The Riot API also contains much more information which we ignore but could use for many purposes.

Bibliography

[1] League of Legends : Game API
https://developer.riotgames.com/

[2] Match Data Downloads
https://oracleselixir.com/match-data/

[3] Doublelift Analysis 2018
https://rpubs.com/kail/dlift

[4] Kaggle : League of Legends Ranked Games
https://www.kaggle.com/jaytegge/league-of-legends-data-analysis/data

[5] League of Analytics
http://league-analytics.com/

[6] Analysis of Data Gathered from League of Legends and the Riot Games API
http://web.cs.wpi.edu/ claypool/mqp/lol-crawler/final-report.pdf