Université Lumière Lyon 2 X League of Legends

LOSER QUEUE

MYTH OR OPTIMIZED MATCHMAKING ?

Written by
LEAL DE ALMEIDA Guillaume

Table of contents

Introduction

During my studies, I had the opportunity to work on a lot of projects on various topics, ranging from sports to economics, including the development of a network solution for analyzing a structured corpus. Similarly, during my apprenticeship at a large insurance group, my work mainly focused on economic issues, which is hardly surprising. Although I do not dislike economics, I wanted to tackle a more original and striking subject for this "final" project as a student.

I have always been passionate about video games. However, as my responsibilities increased and I got closer to the professional world, I had less time to dedicate to them. That’s why I chose, as a nod to my childhood passion, to dedicate my last student project to the analysis of video games.

Video games are, by nature, vast sets of data. Every movement, every action, every pixel, represents a significant amount of data. In recent years, in the competitive and professional branch of video gaming, known as e-sports, more and more people are working either out of passion for the game and analysis (which is my case) or within e-sports clubs, to improve player performance by processing and analyzing this data, much like a race engineer in F1 or an analyst in traditional sports.

For this project, I chose to focus on one of the most well-known games: League of Legends. Before diving into the heart of the matter, it is essential to present this game and explain why the question I will try to answer is crucial for players.

In March 1998, StarCraft, a real-time strategy (RTS) game, was released. It became a reference in its genre, although other similar games existed. Among them was Warcraft, particularly the third, which is known for enabling the development of mods, one of the most famous being Dota. In this mod, two teams of five players compete to destroy the opponent's base. In the United States, two Dota palyers, Brandon Beck and Marc Merrill, played the game so much that they decided to organize a university tournament for students. For them, this tournament was an opportunity to recruit associates and launch their own company dedicated to developing their version of Dota.

These two visionaries founded Riot Games, a company that would make history in video gaming by creating League of Legends (LoL). From the moment the servers opened, especially in Asia, the game experienced a meteoric rise. LoL was installed everywhere and played by everyone. This game quickly became a worldwide cultural phenomenon. Today, 15 years later, it is a global reference, with 124 million active accounts regularly playing matches, and this number continues to grow each year.

Let's move on to the basics of the game: LoL is a multiplayer online battle arena (MOBA) game where two teams of five players compete to destroy the opponent's base. Each player controls a "champion" with unique abilities and various play styles. The five players on each team occupy specific roles: Toplane, Jungle, Midlane, Bottom (or AD Carry), and Support.

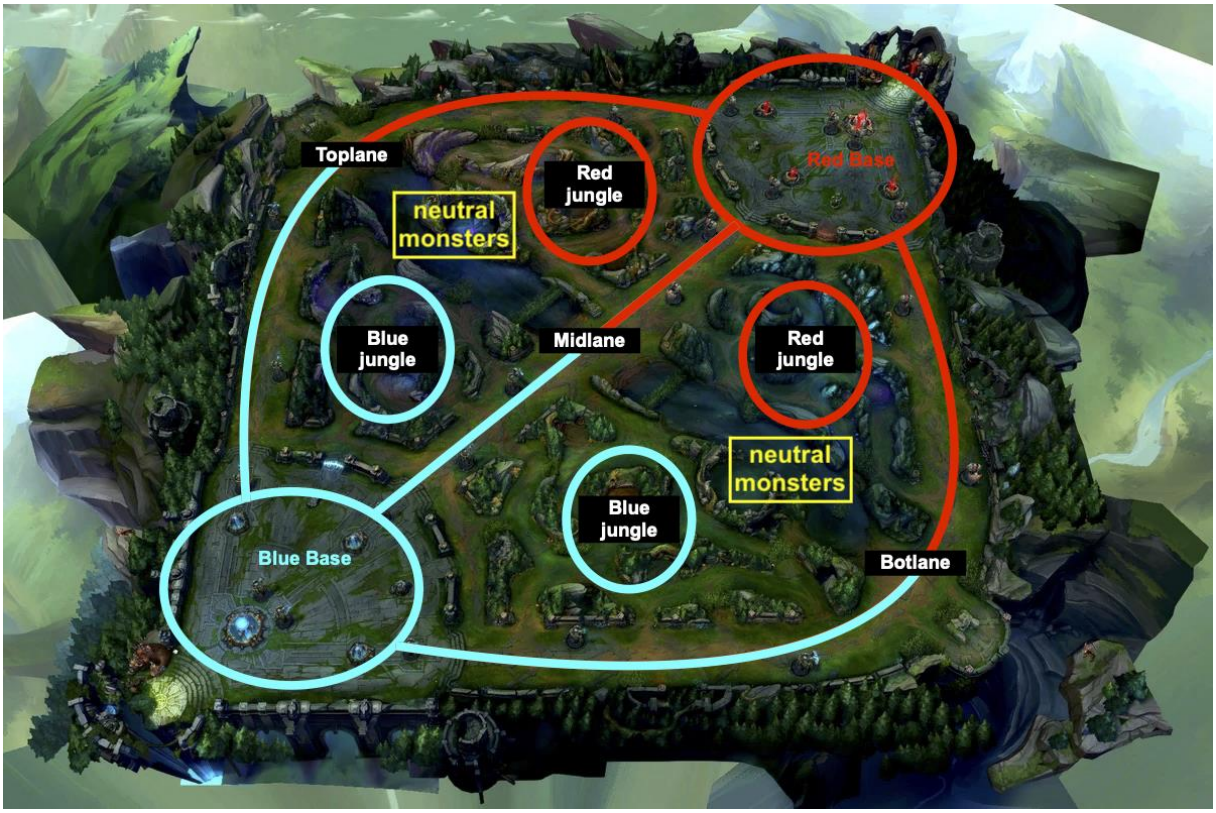


IMAGE 1 - LEAGUE OF LEGENDS MAP

As shown on this map, the toplane player operates on the upper part, the midlane player is positioned on the central lane, and the duo of the AD Carry and Support is on the bottom lane. The jungler, however, has no assigned lane and moves freely around the map, managing neutral monsters in the jungle and assisting teammates in other lanes. As you can see, League of Legends is a team game where cooperation with your teammates is essential to win. The closer the skill level between players in a match, the harder it is to win by playing solo against everyone.

Among these millions of players, there are obviously many casual players, but also a large number of enthusiasts who seek to improve their ranking from season to season. Ranked matches are not accessible to beginners: you need to have a level 30 account and have unlocked 20 champions out of the 168 offered by the game. Although this may seem frustrating, it is important to understand that League of Legends is a complex game with many nuances.

Playing ranked allows you to face players of your level, which is interesting from a competitive standpoint because it avoids overly unbalanced matches. Additionally, choosing to play

ranked adds stakes to each game, as the result directly impacts your ranking: participants are therefore more invested, making the game more interesting. The game's ranking system consists of different tiers. Your place in the ranking evolves throughout the season based on your score, before being reset at the start of the next season. If your team wins, each member earns a certain number of League Points (LP), and if it loses, each member loses LP. This number of LP is displayed on your screen at the end of each game and determines your progression through the different tiers. With stakes in play, invested players also experience frustration in defeat. The "loser queue" in League of Legends is a theory suggesting that the game's matchmaking system intentionally places players in a series of matches where they are likely to lose after winning several consecutive games. This results in poorly balanced teams, where a performing player finds themselves with less competent teammates or against much stronger opponents. This situation creates significant frustration for players, as they feel punished for their victories. They believe that their efforts and skills are not fairly rewarded, and that the results are more or less scripted by the game, regardless of individual performance. Although this theory is not officially confirmed by Riot Games, the perception of being placed in a "loser queue" fuels frustration and disillusionment among many players.

This is the question we will try to answer during this project: does a "loser queue" exist in League of Legends? Before addressing this central question, we will explore a complementary question: among the five roles in the game, which is the most decisive for victory, that is, which role has the most potential to win "alone", to solo carry? To do this, I will start by explaining the methodology used to collect the necessary data for this project. Then, I will conduct descriptive analyses, both univariate and bivariate. Next, I will perform statistical tests to identify the most influential role in achieving victory. Finally, I will look into the history of a former professional player to try to answer the main question of this project, using various tests and analysis methods.

Getting Datta

After coming up with this brilliant idea for my study project, I needed data. The simplest way would be to use predefined data, which would certainly save a lot of time. Unfortunately, LoL changes a lot in a short period. Indeed, LoL offers annual ranked seasons divided into three segments: the first starting in January, the second in May, and the third in September. These segments are generally accompanied by major game updates. These updates can range from simple adjustments to the statistics of one or more champions and/or items, to the addition of new champions and/or items, to reworking an existing champion or modifying elements on the map.

As a result, a role may be more important during a given period, and the addition of a new champion in another role, coupled with adjustments to the item stats for those champions, can make that role much less important the following season. Thus, predefined data may not be up-to-date, which is why I will use my own data, from the first part of season 14. Fortunately, Riot Games offers an API with many different endpoints that we can all use, along with documentation on data collection (https://developer.riotgames.com/).

Game data from a LoL match can be obtained via Riot Games' "Match-V5" API, specifically at "/lol/match/v5/matches/{matchId}", which allows access to information about a specific match. The match ID (matchId) can be retrieved by looking up a player's match history using "/lol/match/v5/matches/by-puuid/{puuid}/ids". This requires the player's PUUID (Player Universally Unique Identifier), which is one of the three identifiers used by Riot's API for players. The three types of identifiers are: summoner IDs, account IDs, and PUUIDs. Each type of API uses specific identifiers. Unlike summoner and account IDs, which are unique per region, PUUIDs are globally unique.

To obtain a player's PUUID, you first need to retrieve player information using their summoner ID via the "Summoner-V4" API, specifically "/lol/summoner/v4/summoners/{summonerID}". I get this summoner ID by obtaining a list of ranked players by tier via "league/v4/" and specifying the desired tier. Now that we know how to obtain the data, let's get to work!

For personal APIs, it's important to note that Riot Games imposes a rate limit of 20 requests every 1 second and 100 requests every 2 minutes. So, in 24 hours, we can make a maximum of 72,000 requests, which limits our extraction.

My initial idea was to take the same proportion of players among the top-ranked in the 7 highest tiers. Each tier, except for Master, Grandmaster, and Challenger tiers, is subdivided into four divisions, with IV being the lowest and I being the highest. The Master, Grandmaster, and Challenger tiers are not subdivided into divisions but are determined by an LP-based ranking system. Note that the highest tier is Challenger, which accounts for 0.031% of ranked players.

I selected the top 100 players from the Master, Grandmaster, and Challenger tiers, and the top 20 from the other tiers. For each of these players, I took a sample of 100 matches played. This would have given me a total of 54,000 matches and thus requests, by selecting the top

540 players with at least 101 matches played. Unfortunately, my computer crashed before the 24-hour period ended, but I still obtained 51,533 data points. In the graph (FIGURE 1), we can see the distribution of players I was able to obtain in my extraction.

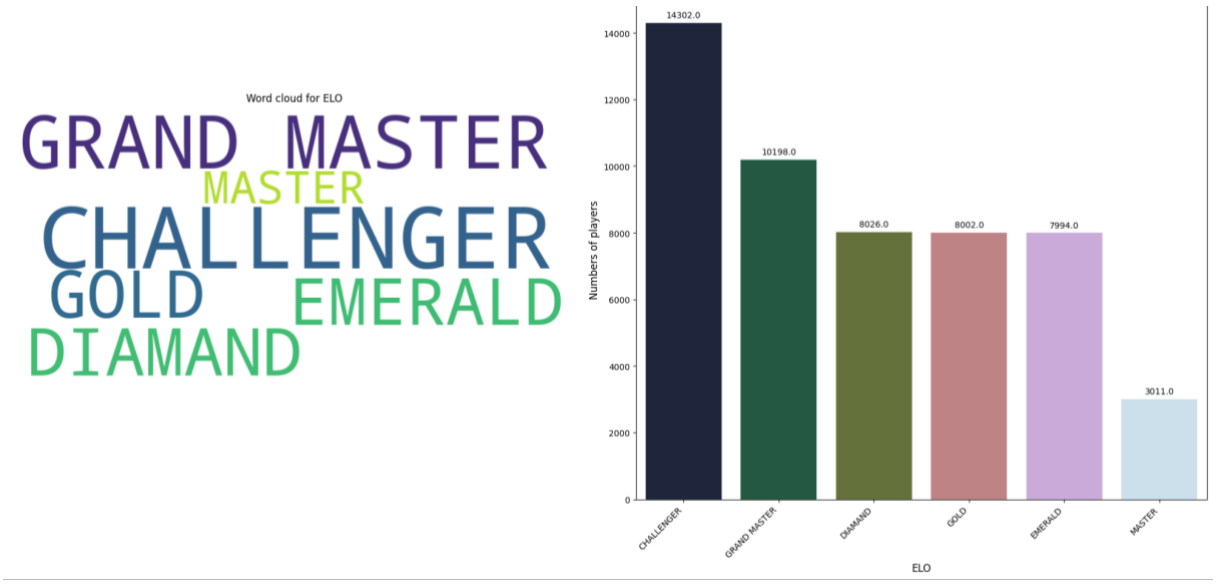


FIGURE 1 - DISTRIBUTION OF PLAYER BY THEIR RANKING ELO

Descriptive analysis

Univariate

In case I didn't explain it well before, my data corresponds to match data. So, a row represents a player's data in a game. It is possible that the same match appears multiple times in my data, but from different players' perspectives. The goal here is to get familiar with the dataset and try to produce interesting graphs to visualize the dataset.

The dataset contains 51,533 rows and initially 76 columns. As the study progresses, I will add a few additional variables. The details and meanings of all variables are available in the appendix. For now, I will analyze some of them, starting with the players' Elo. To retrieve the players' Elo, I had to use their PUUID to get their summonerID, which allows me to get the Elo during the match.

Let's see what we find in the graph :
We have 14,302 players out of 51,533, which means 28% of the matches involve a Challenger-ranked player, the top players in the game. Why this choice? Because League of Legends (LoL) is a strategy game where individual skills, i.e., player dexterity, are crucial. To illustrate this choice, imagine we want to develop the best Formula 1 car. We wouldn't have someone with no talent drive it, nor would we collect their data to improve the car. We would put it in the hands of the best drivers to push it to its limits. Similarly, to obtain reliable data for this project, I must collect data from players who master the game and understand its mechanics. That's why I chose a panel of good players, with a majority of Challenger-ranked players, in my sample.

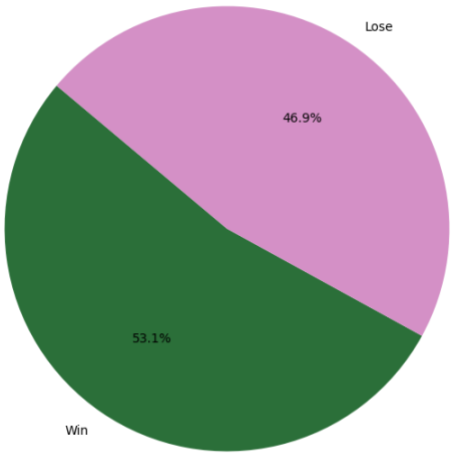


FIGURE 2 - DISTRIBUTION OF RESULTS

In my sample, the 539 different players achieved a win rate of 53%, which could suggest that the "loser queue" phenomenon is real. But let's set that aside for now. On the other hand, the distribution of positions is quite balanced.

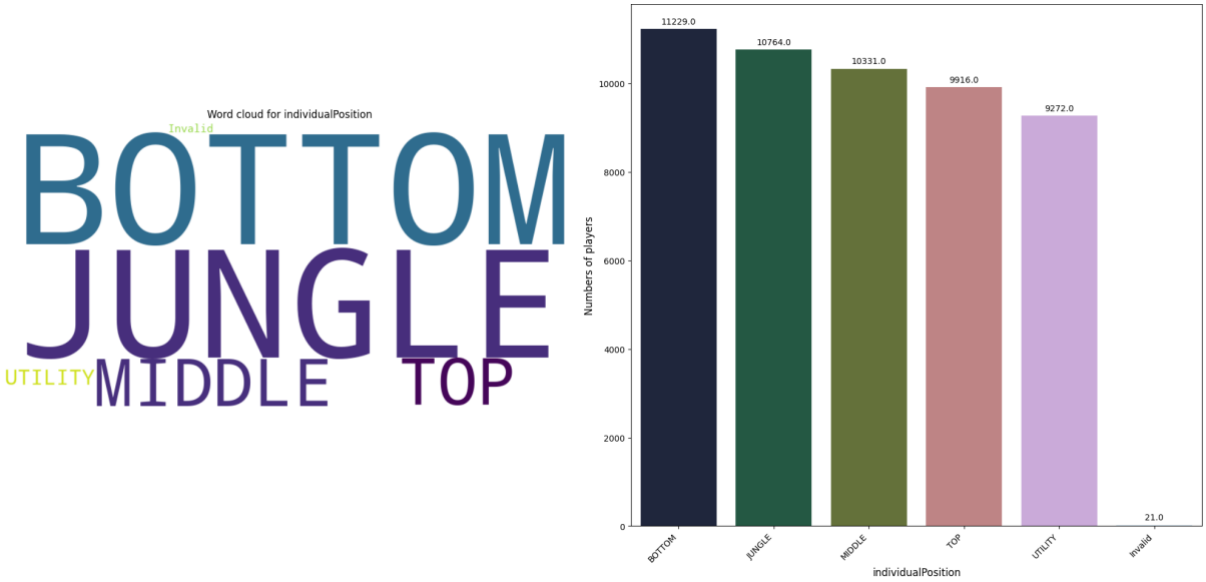


FIGURE 3 - DISTRIBUTION OF PLAYER BY THEIR POSITION DURING THE GAME

It's important to note that a player who wants to play a ranked match must select two roles: their primary role and their secondary role. It's possible that the player will be "autofilled," meaning they don't get their preferred role during the match and are assigned another role.

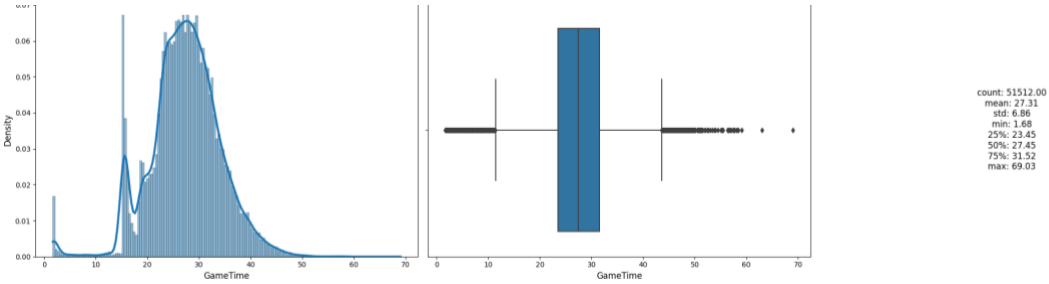


FIGURE 4 - GAMETIME DISTRIBUTION

A League of Legends match lasts an average of 27 minutes.

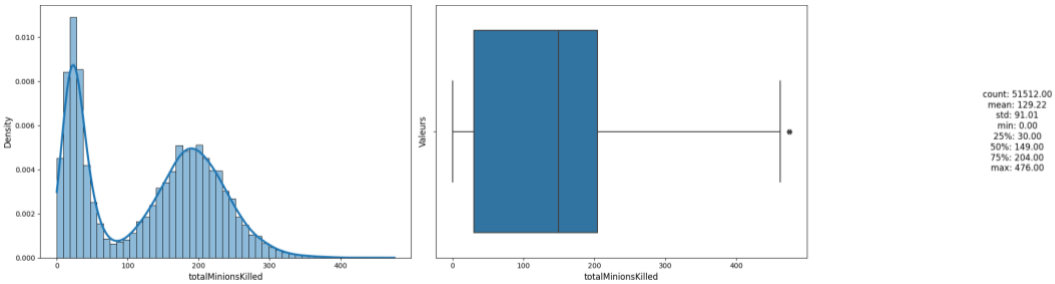


FIGURE 5 - TOTAL MINIONS KILLED DISTRIBUTION

On average, a player kills 129 minions per game. But what is a minion? Minions are units generated by the Nexus (base). They spawn periodically and move along a lane towards the enemy Nexus, automatically attacking any enemy unit or structure they encounter. They are controlled by artificial intelligence and use only basic attacks. If a player kills minions, they receive a certain amount of Gold, allowing them to buy items to strengthen their champion during the game. Generally, a good player kills about 10 minions per minute. In my sample, the average is less than 5 minions per minute, which is low. However, it's important to

remember that the sample includes players playing the support role, who kill almost no minions to leave them for the AD Carry, which logically pulls the average down.

Now let's analyze pings. Toxicity in League of Legends often stems from frustration and the intense competitiveness characteristic of online games. Players often become toxic when they blame their teammates for losses or poor performances, exacerbating their own frustration. This situation is amplified by the anonymity and physical distance of online interactions, which reduce social inhibitions and encourage aggressive or disparaging behavior.

To address this problem, Riot Games has implemented reporting and sanction systems, temporarily or permanently banning toxic players from using in-game chat. Pings were introduced as an alternative for quick and effective communication between teammates, allowing them to indicate objectives, dangers, or instructions without using text messages. Unfortunately, some players abuse pings, using them excessively to distract or annoy their teammates, or even to express their discontent in a passive-aggressive manner, such as repeatedly pinging a fallen ally with a danger ping to subtly express their frustration.

In our dataset, we have the list of all pings used by players. On average, a player uses 44 pings during a match, with the third quartile at 100 pings.

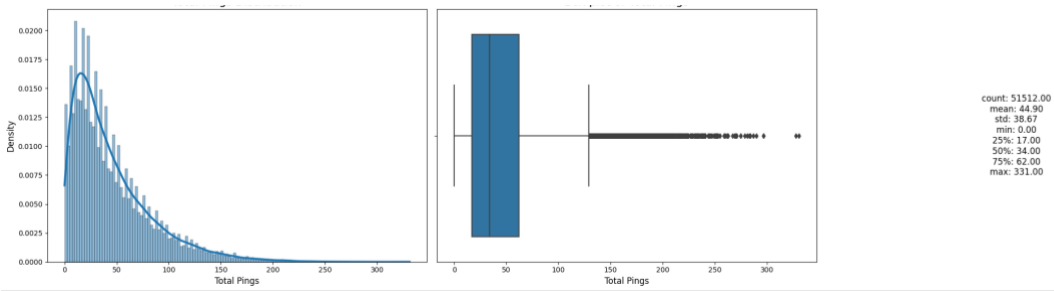
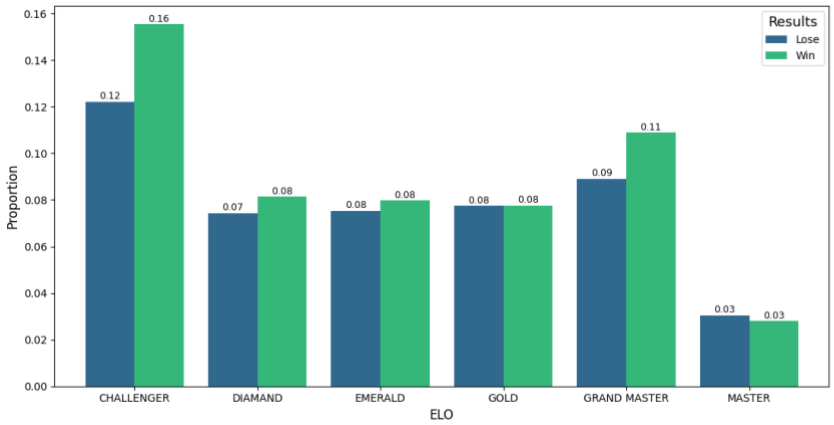


FIGURE 6 - TOTAL PINGS DISTRIBUTION

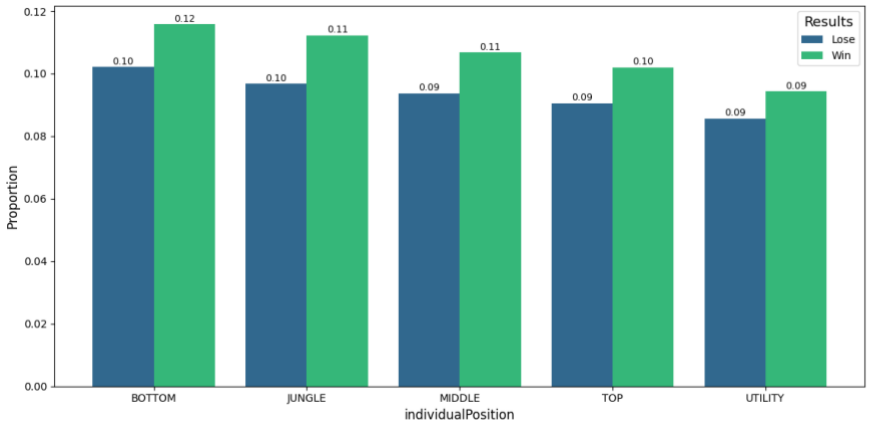
Bivariate



	Lose	Win
CHALLENGER	44.0%	56.0%
DIAMAND	47.7%	52.3%
EMERALD	48.55%	51.45%
GOLD	49.93%	50.07%
GRAND MASTER	45.0%	55.0%
MASTER	52.14%	47.86%

FIGURE 7 - DISTRIBUTION OF RESULTS BY ELO

In this graph, we see the distribution of results by Elo. The win rates are around 50% for most categories, but Challenger and Grandmaster players stand out with higher win rates, reaching 56% and 55% respectively. Conversely, Master players have a win rate below 50%, at 47.86%. Players in other ranks, such as Diamond and Emerald, show slightly above 50% win rates, indicating slightly positive performance. The proportion of wins and losses varies by Elo, highlighting the differences in skills and performance among the various ranking levels in the game.



	Lose	Win
BOTTOM	46.86%	53.14%
JUNGLE	46.28%	53.72%
MIDDLE	46.74%	53.26%
TOP	47.04%	52.96%
UTILITY	47.56%	52.44%

FIGURE 8 - DISTRIBUTION OF RESULTS BY POSITION

This graph shows the distribution of results by individual player positions. Win rates are fairly balanced across different positions, with slightly higher proportions for the jungle (53.72%) and midlane (53.26%) roles. The botlane and toplane roles follow closely with win rates of 53.14% and 52.96% respectively. The support position has the lowest win rate at 52.44%, although it remains close to other positions. These results suggest that while every position is important, certain positions like jungle might have a slightly more significant influence on the match outcome.

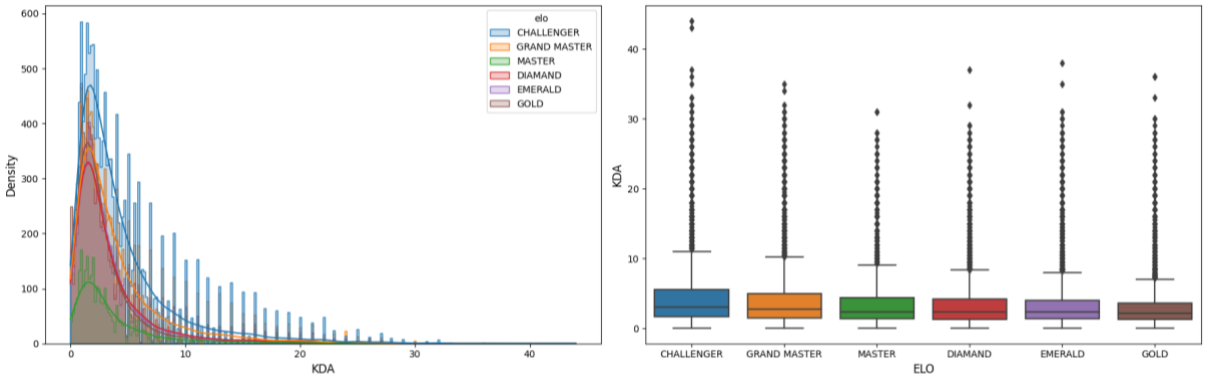


FIGURE 9 - DISTRIBUTION OF KDA BY ELO

In these graphs, we analyze the distribution of KDA by Elo. The density graph shows that Challenger players have a higher KDA density in the upper values, indicating better performance in terms of kills and assists relative to deaths. The box plots confirm this observation: the median KDAs are higher for Challenger and Grandmaster players, around 4 to 5, compared to other ranks where the median is closer to 3.

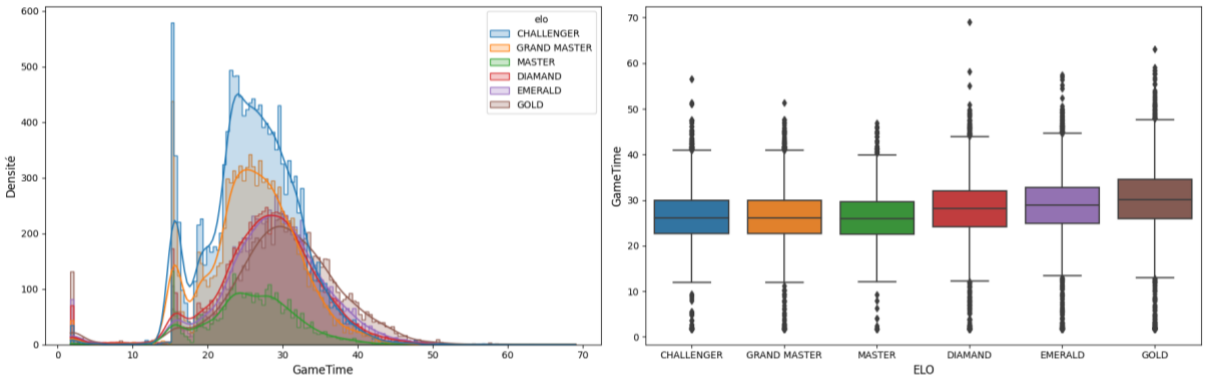


FIGURE 10 - DISTRIBUTION OF GAME TIME BY ELO

The graphs analyze the distribution of game duration (GameTime) by player Elo. The density graph shows that Challenger and Grandmaster players have slightly shorter games, with peak densities around 25 to 30 minutes. The box plots show that the median game durations for higher-ranked players (Challenger and Grandmaster) are slightly lower than those of other ranks, around 26 minutes for Challenger and Grandmaster, while lower-ranked players, like Gold, have medians close to 30 minutes.

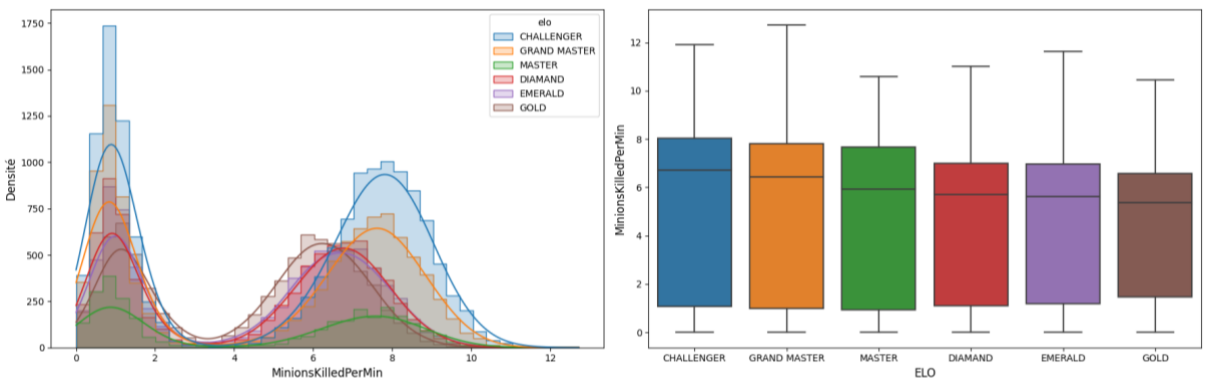


FIGURE 11 - DISTRIBUTION OF MINIONS KILLED PER MINUTE BY ELO

These graphs show the distribution of minions killed per minute (MinionsKilledPerMin) by player Elo. The density graph indicates that Challenger players have a higher concentration of values around 7 to 8 minions per minute, with a marked initial peak, demonstrating their efficiency in minion collection. The box plots show that the medians for minions killed per minute for Challenger and Grandmaster players are higher, around 7, while other ranks have slightly lower medians, around 6.

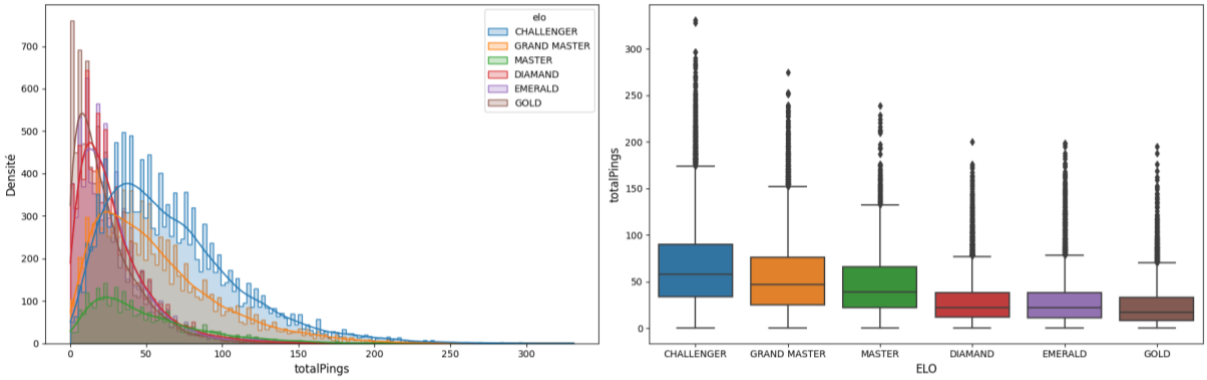


FIGURE 12 - DISTRIBUTION OF TOTAL PINGS BY ELO

These graphs show the distribution of total pings used (totalPings) by player Elo. The density graph reveals that Challenger players use more pings, with a higher density in the upper values, indicating more intensive communication. The box plots confirm this observation: the medians for pings used by Challenger players are higher, around 60 pings per game, while players in other ranks show lower medians, ranging around 30 pings. This could mean two things. Firstly, it is possible that the best players communicate more frequently and more effectively with each other, leading them to victory more quickly and efficiently. Secondly, it could indicate that top players are more prone to toxic behavior due to the increased pressure and higher stakes in high-level matches.

Analysis of the most influential role for victory

Chi-Square test

To determine the most influential role in a game, I began by analyzing the dataset, which now contains 51,512 rows and 80 variables. The objective is to identify which role can independently have the greatest impact on victory. For this, I used the chi-square test. This statistical test helps determine if there is a significant association between two categorical variables by comparing the observed frequencies with the expected frequencies under the assumption of independence.

$$x^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Equation 1 - Chi-Square mathematical formula

Where O_i represents the observed frequencies and E_i the expected frequencies.

To interpret the results of the chi-square test, we consider the chi-square statistic, the p-value, and the degrees of freedom. A p-value less than 0.05 indicates that the variables are significantly associated.

Analyzing the results for different roles, we see that for Toplaner, the chi-square is 19.3772 with a p-value of 0.0016 and 5 degrees of freedom. This p-value, being much less than 0.05, indicates a significant association between the variables for this role. Similarly, for Jungle, the chi-square statistic is 36.1324 with a p-value of 8.936e-07, also suggesting a significant association.

For Midlaner, the chi-square statistic reaches 20.9835 with a p-value of 0.0008. Here, the p-value below 0.05 reveals a significant association between the variables, suggesting that some positions might have a greater impact on the game's outcome at this level. For AD Carry, the chi-square statistic is 31.5016 with a p-value of 7.456e-06, and for Support, the statistic is 46.4424 with a p-value of 7.381e-09. In both cases, the p-values indicate a significant association.

The results show that for all roles analyzed, there is a significant link between the roles played and victories. However, the Jungle and Support roles show particularly strong associations, suggesting that these roles may have a greater influence on the outcomes of the games.

Kernel PCA

Using the chi-square test, this initial analysis has shown that support and jungle may be the most influential role in determining game outcomes on their own. Let's try another method to see if it confirms or not this theory. My idea is to train predictive models, specifically a Random Forest and a Gradient Boosting model.

Random Forest is an ensemble model that constructs a multitude of decision trees during training and outputs the class that is the mode of the classes (classification) or the mean prediction of the individual trees (regression). It works by creating several random subsamples of the training dataset and building a decision tree for each subsample. The predictions from all the trees are then combined. This process, known as "bagging" (Bootstrap Aggregating), helps reduce variance and improve model accuracy. Mathematically, if we have N trees $T, T_1, T_2, \dots, T_N$, the prediction for an observation x is:

$$\hat{y} = \frac{1}{N} \sum_{i=1}^{N} T_i(x)$$

EQUATION 2 - RANDOM FOREST EQUATION

Gradient Boosting is an ensemble technique that builds models sequentially, with each model attempting to correct the errors of its predecessor. Unlike Random Forest, which builds trees independently, Gradient Boosting adjusts each new tree to reduce the residual errors of the previous trees. Models are weighted based on their performance, and a loss function is minimized to improve overall predictions. Mathematically, if we have a series of models $f_1, f_2, \dots, f_m$, the prediction for an observation x is:

$$\hat{y} = \sum_{m=1}^{M} \gamma_m f_m(x)$$

EQUATION 3 - GRADIENT BOOSTING EQUATION

where γ is the weight assigned to the model based on its performance.

To handle my large dataset with many variables on a less powerful machine, I use Kernel PCA to reduce the dimensionality of my dataset.

Kernel PCA is an extension of Principal Component Analysis (PCA) that can handle nonlinear relationships between variables. While classic PCA uses a linear transformation to reduce data dimensionality, Kernel PCA uses kernel functions to project data into a higher-dimensional space where nonlinear relationships can be captured. This method better captures the structure of complex data. Classic PCA transforms data via a covariance matrix, whereas Kernel PCA transforms data via a kernel matrix.

Classic PCA formula: $X_{pca} = XW$, where X is the original dataset and W is the matrix of eigenvectors of the principal components.

Kernel PCA formula: $K_{pca} = KV$, where K is the kernel matrix and V is the matrix of eigenvectors of the principal components in the kernel matrix space.

I used the RBF (Radial Basis Function) kernel because it is very effective at capturing complex nonlinear relationships without requiring explicit specification of the transformation. The Gamma parameter controls the range of the RBF function's effect. A high Gamma means that each data point strongly influences its close neighbors, while a low Gamma means that each data point has a broader but weaker influence.

To avoid negatively impacting model performance, I normalized Gamma based on the dataset's dimensionality. This ensures that the Gamma parameter is appropriate for the data scale, thereby optimizing model performance. To determine the optimal number of components, I conduct several tests. Since my target variable is categorical, I use logistic regression, which allows me to obtain a probability of victory.

For each test of the number of components, I calculate the accuracy score, which measures the proportion of correct predictions made by the model compared to the total number of predictions. I then display a graph showing the accuracy as a function of the number of Kernel PCA components, adding an annotation at the point of maximum accuracy. This helps me identify the optimal number of components for Kernel PCA that maximizes the accuracy of my logistic regression model.

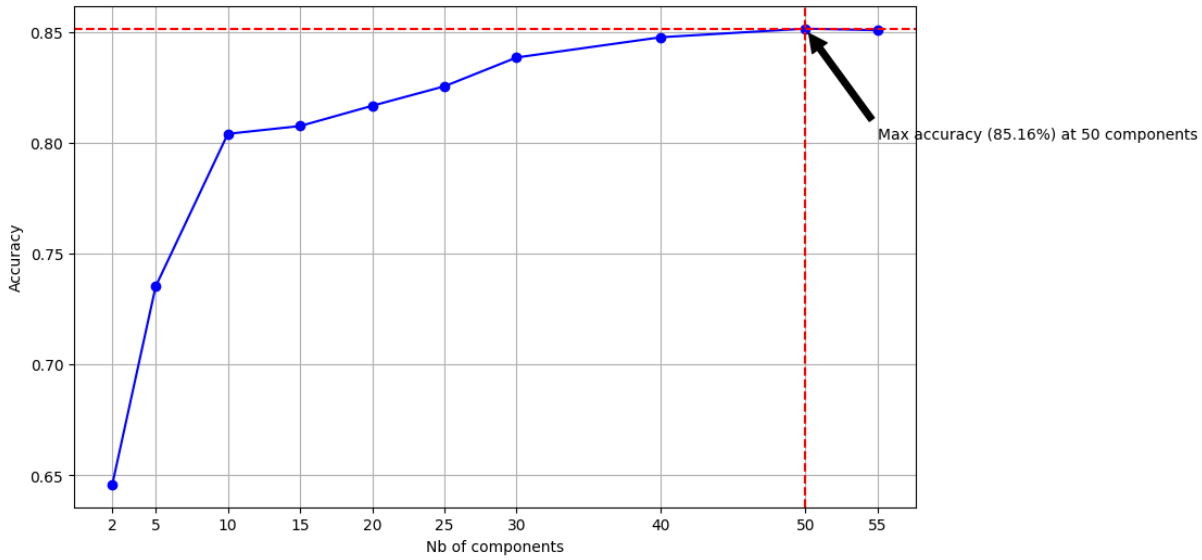


FIGURE 13 - ACCURACY DEPENDING ON THE NUMBER OF KERNEL PCA COMPONENTS

As shown in the graph, the optimal number of components is 50. However, with only a 0.05 loss in accuracy, we can reduce the number of components by 40. This saves machine resources without sacrificing performance.

I apply Kernel PCA with an RBF kernel and an adjusted Gamma parameter to the training and test data to reduce their dimensionality. Then, I encode the individual player positions numerically and combine these encodings with the Kernel PCA-transformed data.

Next, a Random Forest classifier is trained on this combined data, and the model's performance is evaluated using accuracy and cross-validation measures. Similarly, I train a Gradient Boosting classifier on the same data and evaluate its performance as well.

Both predictive models, Random Forest and Gradient Boosting, demonstrated strong performance, with respective accuracies of 81.86% and 81.19%. The Random Forest showed higher precision, recall, and f1-scores for the "Win" and "Lose" classes compared to Gradient Boosting. The RFC classification report indicates a better balance of scores, with precision and recall of 0.84 for the "Win" class and 0.79 for the "Lose" class. In contrast, the GBM shows a slight drop in performance for the "Lose" class with a recall of 0.76. Cross-validation confirms these results, with an average accuracy of 82.15% for RFC and 81.84% for GBM.

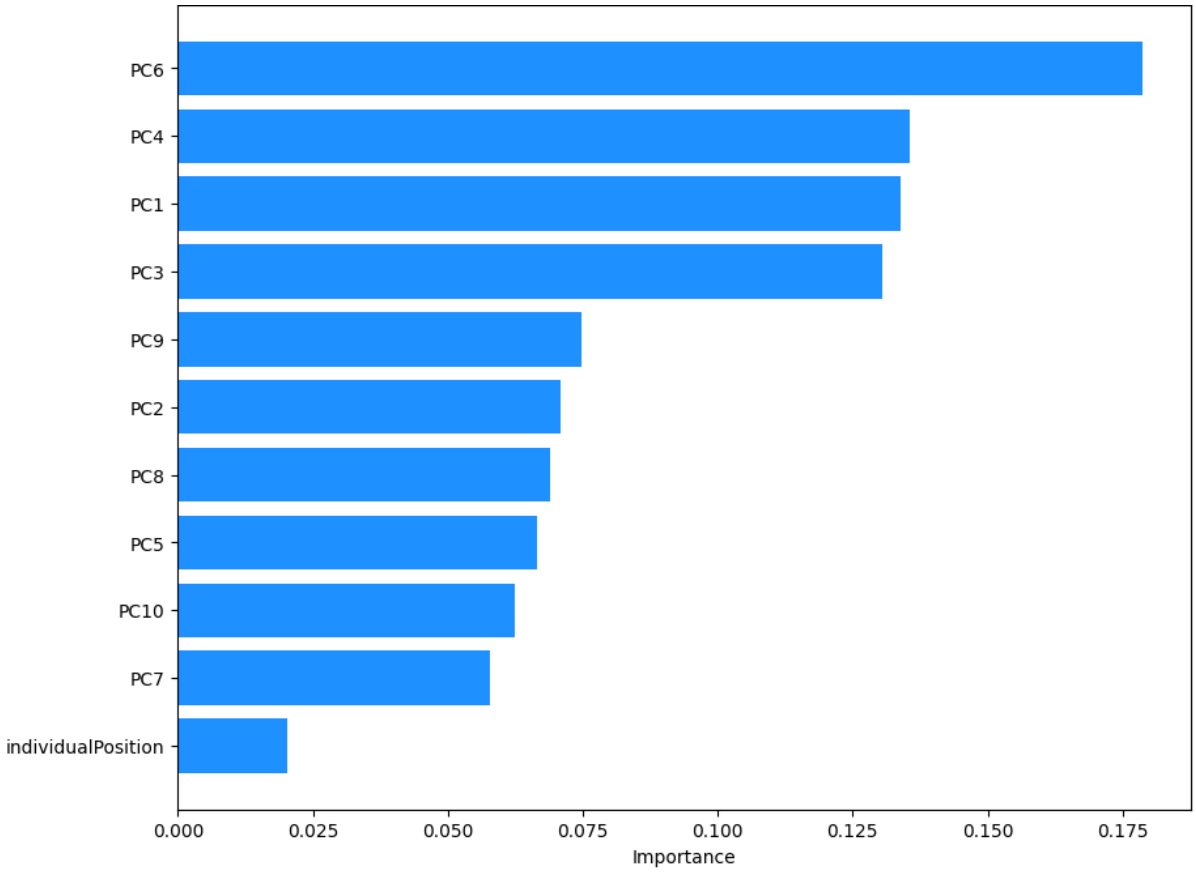


FIGURE 14 - IMPORTANCE OF FEATURES ACCORDING TO RANDOMFOREST

As might have been expected—or not—the "individual position" variable is not the most critical factor in determining victory or defeat. So, which role is the most important?

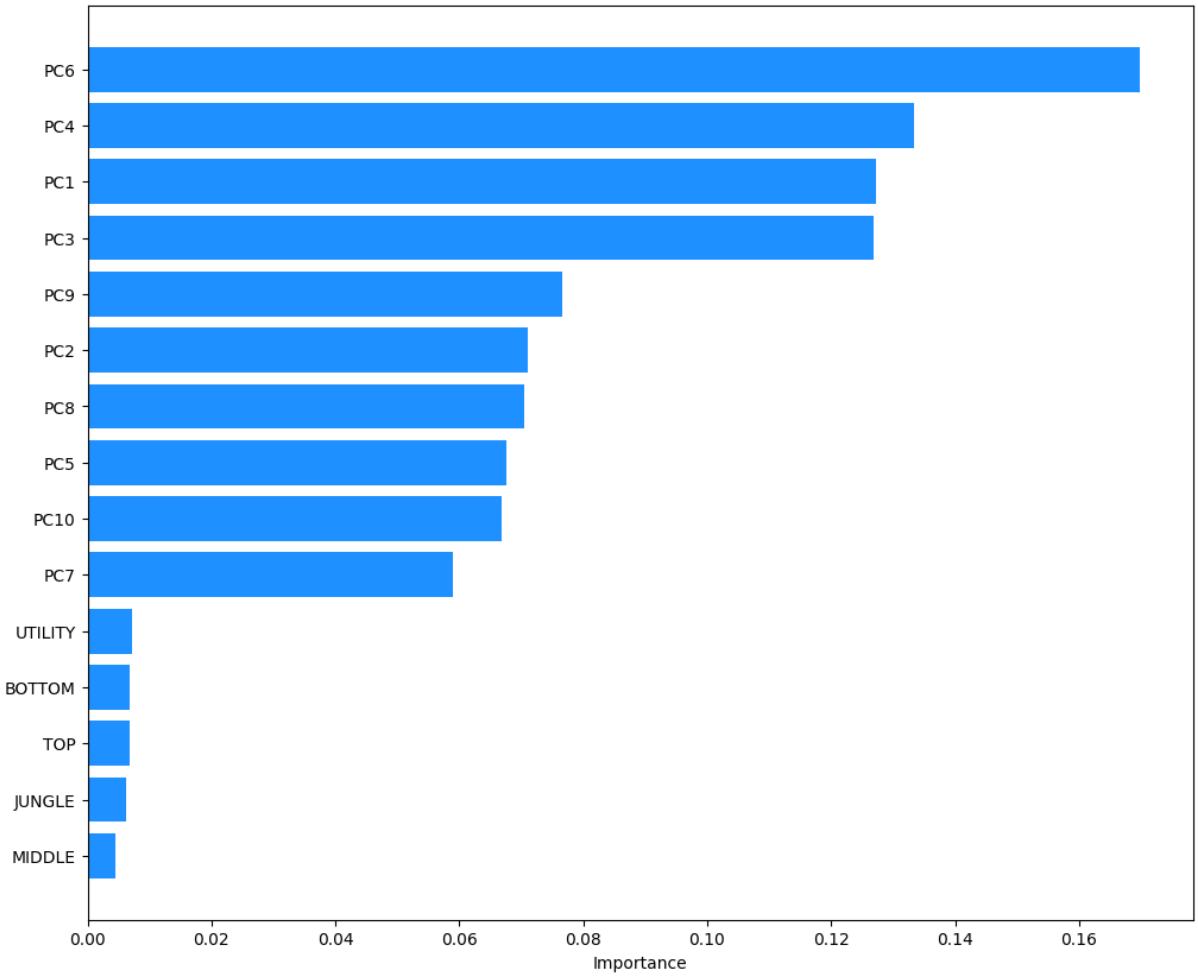


FIGURE 15 - IMPORTANCE OF FEATURES ACCORDING TO RANDOMFOREST (DETAILED)

This graph confirms our earlier findings: support and jungle roles are the most crucial for victory, despite their relative importance compared to other variables. So let's continue the analysis by focusing on Support.

Focus on the support role, does the loser queue exist?

Getting the new Dataset

Now that I have identified the two most important roles, namely Support and Jungle, I need to determine who will be my test subjects. For this, I chose to focus on the country that has the best League of Legends players: South Korea. Personally, I would have preferred if the most impactful role was that of Midlaner, so I could work on the best player in the game, the GOAT (Greatest of All Time), Faker. Unfortunately, that is not the case.

To give some context, Lee Sang-hyeok, known by his pseudonym Faker, is the most famous League of Legends player in the world and continues to play to this day. At the time of writing this report, his team and he are the reigning world champions (I hope he wins his fifth title this winter in London). All this to say that I wanted to study him or a player from his team. However, for the past few months, they have been victims of cyber-attacks, forcing them to play on anonymous accounts unknown to the public.

Therefore, I opted for the history of Cho "Mata" Se-hyeong, an exceptional former professional player in the support role, who is now one of the coaches of the Korean champion team (which is not Faker's team 😢). I also chose a player in the jungler role, Koo "Guwon" Kwan-mo, who is currently the jungler for T1's academy team (Faker's team). However, these additional analyses regarding Guwon will be presented in the appendix.

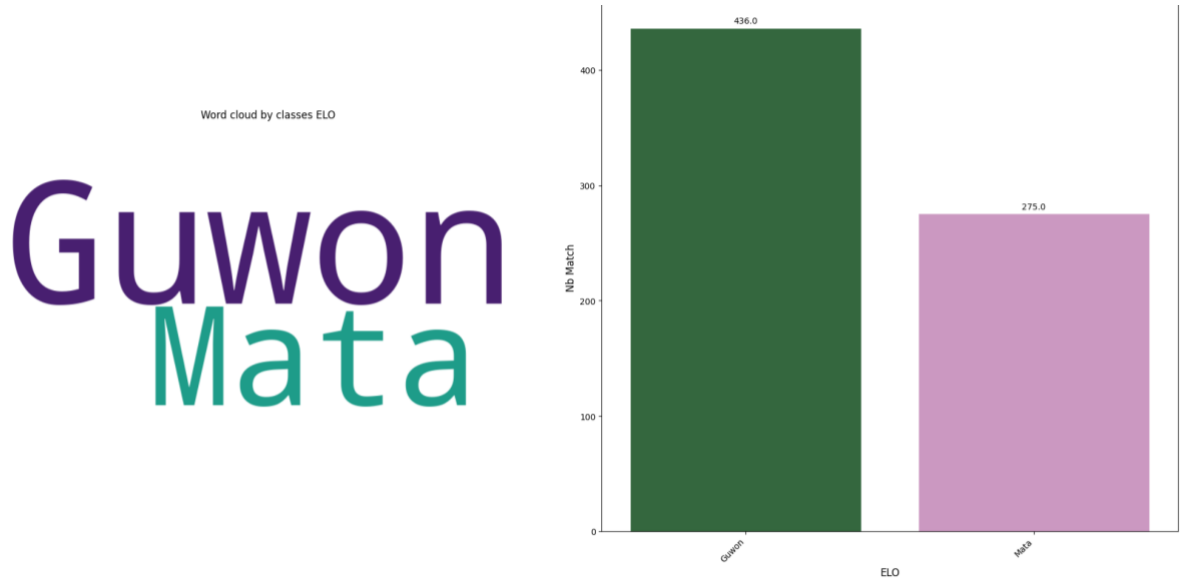


FIGURE 16 - DISTRIBUTION OF GAMES BY NAME

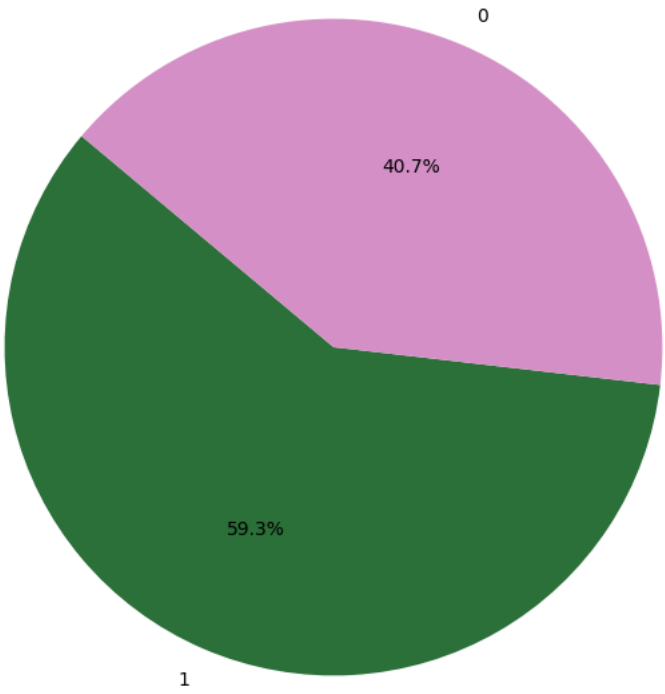


FIGURE 17 - MATA'S SOLOQUEUE WIN RATE

Shap Values

My objective here is to understand how the support player alone can most influence the result of a game. To do this, I will use SHAP values. SHAP values, or SHapley Additive exPlanations, are values derived from cooperative game theory to explain the output of a model. They assign an importance to each feature for a given prediction, allowing us to understand how each feature contributes to the model's decision. SHAP values are calculated by evaluating all possible combinations of features and measuring the variation in predictions. The SHAP value of a feature is the average of its marginal contributions across all possible combinations.

To apply SHAP values, a performant predictive model is necessary. I built on my previous work with kernel PCA, adding an XGBoost (XGB) model, a gradient boosting algorithm.

XGBoost is an ensemble technique that builds models sequentially, with each model attempting to correct the errors of its predecessor. Unlike random forests, which build trees independently, gradient boosting adjusts each new tree to reduce the residual errors of previous trees. Models are weighted based on their performance, and a loss function is minimized to improve overall predictions.

Mathematically, if we have a series of models $f_1, f_2, \ldots, f_m$, the prediction for an observation x is:

$$\hat{y} = \sum_{m=1}^{M} \gamma_m f_m(x)$$

Equation 4 - XGBoost Equation

After training the three models (Random Forest Classifier - RFC, Gradient Boosting Classifier - GBC, and XGBoost - XGB), let's look at the results obtained:

After training the 3 models, the results show varied performances. The RFC model has an accuracy of 0.8727. The classification report indicates a precision of 0.81 for the defeat class and 0.96 for the victory class, with a recall of 0.96 for the defeat class and 0.79 for the victory class. The F1 scores are 0.88 for the defeat class and 0.87 for the victory class, with a weighted average of 0.87. The GBC model has an accuracy of 0.80. The precision for the defeat class is 0.74 and for the victory class is 0.88, with a recall of 0.88 for the defeat class and 0.72 for the victory class. The F1 scores are 0.81 for the defeat class and 0.79 for the victory class, with a weighted average of 0.80. Finally, the XGB model has an accuracy of 0.8182. The precision is 0.77 for the defeat class and 0.86 for the victory class, with a recall of 0.88 for the defeat class and 0.72 for the victory class. The F1 scores are 0.82 for the defeat class and 0.79 for the victory class, with a weighted average of 0.82. The selected best model is the RFC due to its overall better performance.

Here is what the SHAP values applied to the RFC model show:

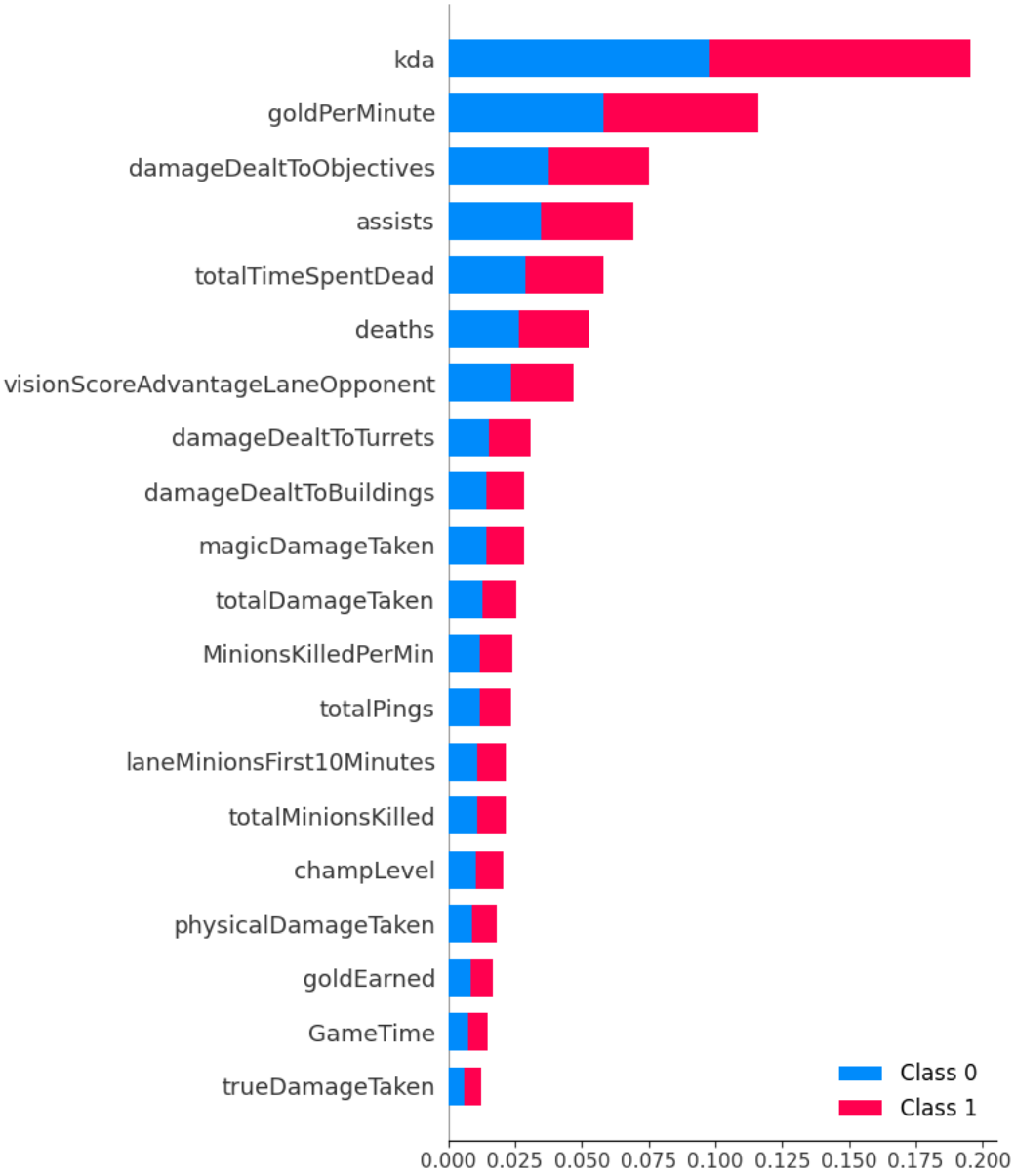


FIGURE 18 - SHAP SUMMARY PLOT

The summary plot presents the average importance of SHAP values for different features, showing their average impact on the model output, ranked in descending order of influence. The KDA ratio (Kill/Death/Assist) is the most determinant feature, with a significant influence on predicting victory (class 1). The features "gold per minute" and "damage dealt to objectives" follow in importance, also showing notable contributions, especially for the victory class. Assists also have considerable importance but affect both classes equally. Other variables like "total time spent dead," "deaths," and "vision score advantage lane opponent" show a more moderate but noticeable influence, especially for the victory class. Features with a lesser impact, such as "total damage taken," "magic damage taken," and "total pings," have reduced influence on the model. Overall, features related to direct combat performance (KDA, gold per minute, damage dealt to objectives) are the most crucial for determining the outcome of the game, particularly in the support role.

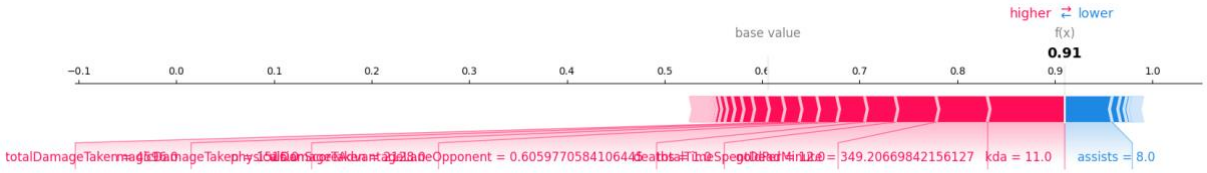


FIGURE 19 - SHAP FORCE PLOT

The SHAP force plot shows the impact of different features on a specific prediction. Each bar represents a feature, and the length of the bar indicates that feature's contribution to the final prediction. Positive values (in red) push the prediction towards a higher value, while negative values (in blue) push it towards a lower value. In the analyzed case, the final prediction is 0.91. The most influential features are the KDA ratio (with a value of 11.0) and assists (with a value of 8.0), indicating that a high KDA and a high number of assists significantly increase the probability of victory.

Other features that contribute positively are gold per minute and total time spent dead. In contrast, features like total damage taken, magic damage taken, and physical damage taken have minor influence and are close to the base value, indicating they don't affect this particular prediction much.

The features selected as most important for victory in the support role are:

- KDA
- Gold Per Minute
- Damage Dealt to Objectives
- Damage Dealt to Turrets
- Assists
- Vision Score Advantage Lane Opponent
- Vision Score Per Minute
- Total Damage Taken
- Total Damage Dealt
- Total Pings
- Total Heals On Teammates

Monte-Carlo simulation

To analyze and simulate the impact of these ten features on the probability of victory, I decided to use Monte Carlo simulations. Monte Carlo simulations are a technique based on random sampling to estimate the properties of a system. They are particularly useful for assessing uncertainty and variability in predictive models, providing a detailed perspective on the influence of each feature on the outcome of a game.

I generated 500 scenarios by producing random feature values following a normal distribution based on their means and standard deviations. For this, I used three classification models: RandomForestClassifier, GradientBoostingClassifier, and XGBClassifier, combined in a VotingClassifier that uses a soft voting method to aggregate the predictions of each base model. I added the predicted win probabilities for the entire dataset into a new column in the original dataset, allowing for the analysis of the predicted win probability for each instance.

The win probabilities for each simulated scenario were displayed as histograms for certain specific simulations.

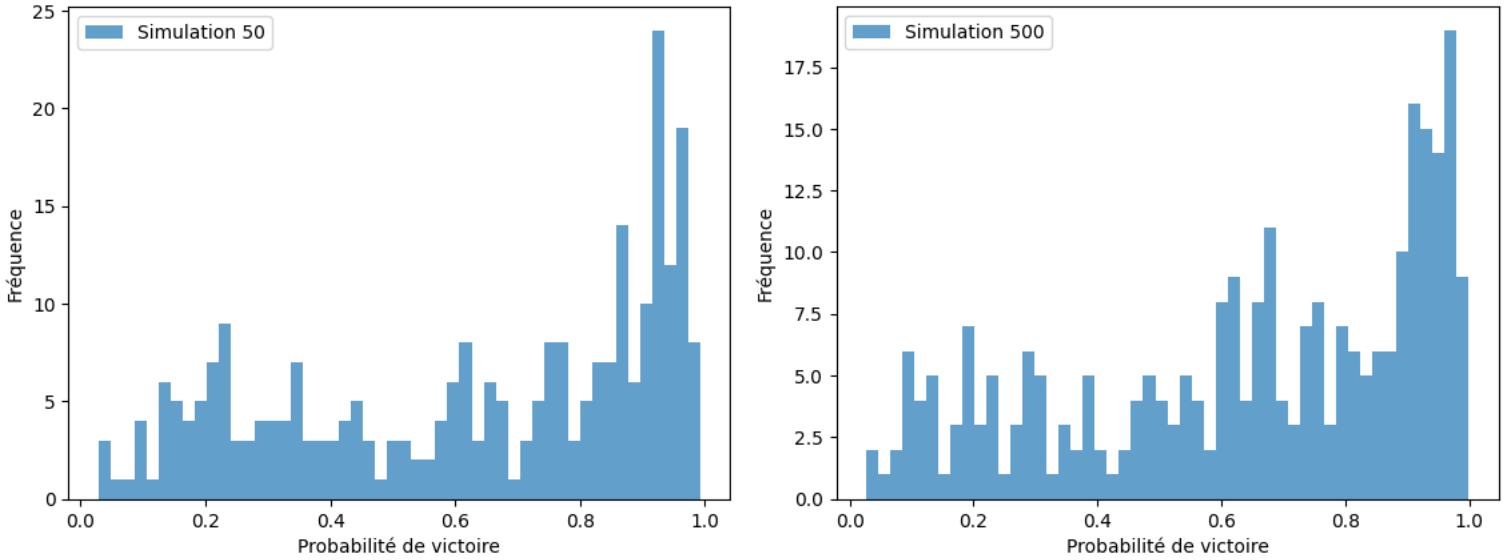


FIGURE 20 - DISTRIBUTION OF SIMULATED VICTORY PROBABILITIES

To analyze the impact of the features, I calculated the influence of each feature by analyzing the variation in the means of the feature values across the scenarios, then sorted the features in descending order of impact. The results showed that the most important features were totalDamageDealt, totalDamageTaken, totalHealsOnTeammates, damageDealtToObjectives, and damageDealtToTurrets, indicating that performance in terms of damage and healing is crucial.

I also evaluated the effect of increasing or decreasing certain features by 10% on the probability of victory. The results showed that increasing features like goldPerMinute, totalPings, kda, and totalDamageTaken by 10% had a significant impact on the probability of victory, while decreasing these same features could reduce the probability of victory. For

example, increasing goldPerMinute by 10% raises the probability of victory by 0.68%, while decreasing it by 10% lowers it by 0.58%.

Next, I assessed the overall influence of feature modifications. The results revealed that features like kda, goldPerMinute, damageDealtToObjectives, damageDealtToTurrets, and assists have significant impacts on the probability of victory. For instance, increasing goldPerMinute by 10% increases the probability of victory by 4.07%, while decreasing it by 10% reduces it by 6.34%. Conversely, features like totalDamageTaken and totalDamageDealt showed a negative influence when they were increased or decreased, suggesting that excessive damage taken or dealt without control can harm the chances of victory.

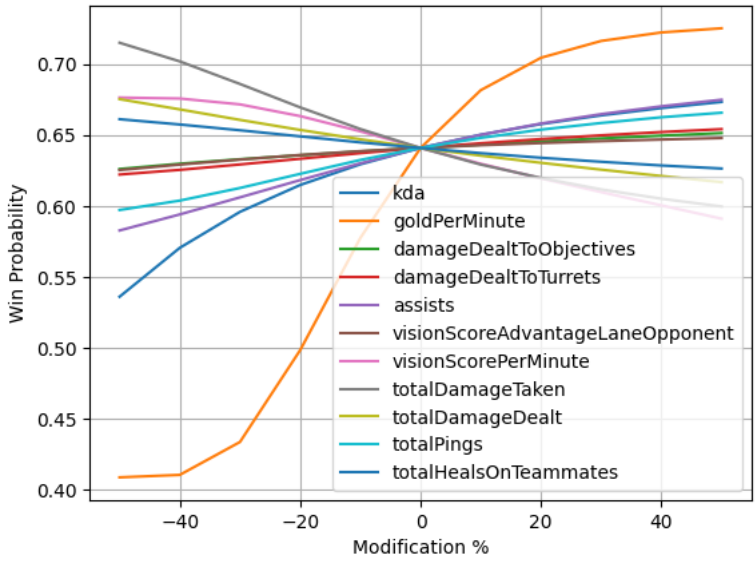


FIGURE 21 - IMPACT OF FEATURES CHANGES ON WIN PROBABILITY

The graph shows that certain features, such as gold per minute and KDA, have a much more significant impact on the probability of victory compared to others, indicating that these features are essential levers for maximizing the chances of success in a game.

Hidden Markov Model

Markov models are mathematical models used to describe a system that evolves from one state to another in a discrete time chain. Each state depends solely on the previous state, described by the Markov property. The transition probability between states is fixed and independent of prior transitions, allowing for the modeling of simple stochastic processes.

However, hidden Markov models (HMM) go further by introducing hidden states that are not directly observable, with observations probabilistically linked to these hidden states. This allows for modeling systems where underlying states influence observations but are not directly visible. This is particularly relevant for complex problems where states are not directly accessible.

In this project, using HMM is particularly relevant for verifying the existence of a "loser queue" because the states of "winning" or "losing" are not always explicitly observable. Match results can be influenced by various hidden factors such as player morale and individual performance, as illustrated throughout this report. HMMs capture this complexity by modeling hidden states (such as a series of consecutive unseen losses) and their impacts on observed outcomes (wins or losses).

HMMs function by defining a set of hidden states and a set of observations. They use transition matrices to model the probability of moving from one hidden state to another and emission matrices to model the probability that a hidden state generates a particular observation. The mathematical formula for HMMs is based on these conditional probabilities. For an HMM with hidden states (S) and observations (O), the probability of the sequence of observations given a sequence of hidden states is expressed by:

$$P(O|S) = \prod_{t=1}^{T} P(O_t|S_t) * P(S_t|S_{t-1})$$

EQUATION 5 - HIDDEN MARKOV MODEL EQUATION

where $P(O_t|\ S_t)$ is the probability of observing O_t at state S_t, and $P(O_t|\ S_{t-1})$ is the probability of transitioning from state S_{t-1} to state S_t.

I chose a Gaussian hidden Markov model to analyze these transitions. A Gaussian HMM is particularly suitable here because my data is continuous, with various player performance metrics (e.g., KDA, gold per minute, etc.). I also defined the covariance matrix of the Gaussian distributions as diagonal. This simplifies the model by assuming that the different dimensions of the data are independent and that each variable has its own variance, but without covariance between different dimensions. This choice reduces computational complexity because there are fewer parameters to estimate compared to a full covariance matrix. In practice, this allows for faster and more stable model training, especially with a large number of features.

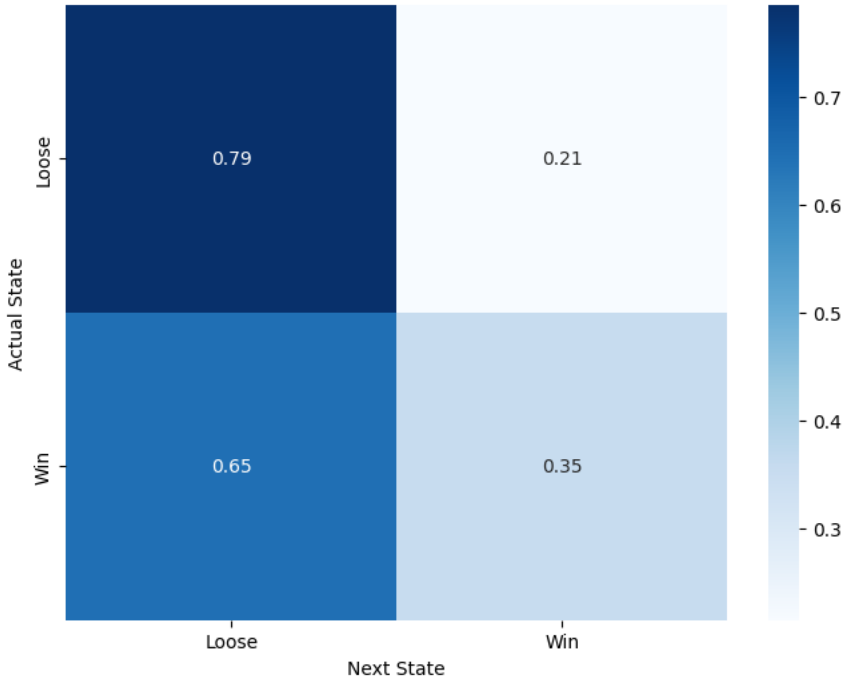


FIGURE 22 - TRANSITION MATRIX OF THE WHOLE DATASET

The results from the obtained transition matrix show that the probability of remaining in the losing state is 78.54%, while the probability of transitioning from losing to winning is 21.46%. For the winning state, the probability of remaining in that state is 35.47%, and the probability of transitioning to losing is 64.53%. These results suggest a strong tendency to remain in the current state, with a significant probability of transitioning from winning to losing, which could support the hypothesis of a "loser queue."

Next, I divided the data into two subsets based on match results (win or loss) and calculated the means of different characteristics for each subset. I then performed independent t-tests on each characteristic to identify those with significant differences between wins and losses. A t-test is a statistical test that compares the means of two samples to determine if they come from different populations. It is useful for identifying characteristics that significantly influence match outcomes.

The results show notable differences between the average statistics of wins and losses. For example, the average KDA for wins is much higher (7.15) than for losses (2.23), which makes sense as better individual performance often leads to victory. In wins, players also have more assists on average (15.02) compared to losses (9.45). In terms of damage, players on winning teams inflict less total damage (24,952) than those on losing teams (26,558) but also take less total damage (13,564 vs. 16,440).

The results also highlight significant differences in aspects such as gold per minute (335.21 for wins vs. 290.82 for losses) and total pings (165.19 for wins vs. 136.21 for losses), indicating that resource management and communication are key factors for victory.

Subsequently, I filtered the features showing significant differences between wins and losses ($p < 0.05$) based on the previously conducted t-tests. I used these characteristics to train a Gaussian hidden Markov model to analyze the transitions between winning and losing states.

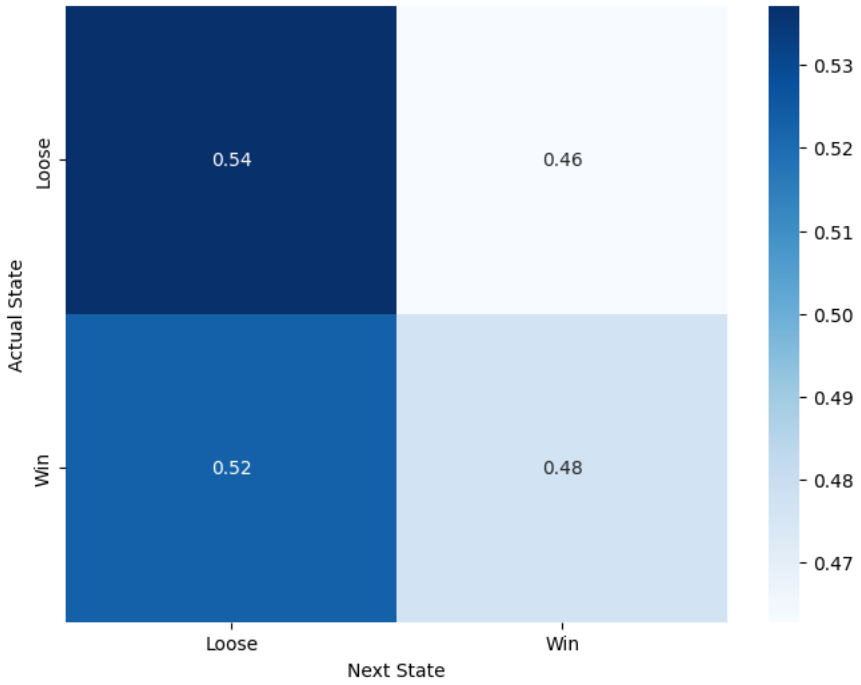


FIGURE 23 - TRANSITION MATRIX ON FILTERED FEATURES

The results show a transition matrix with nearly equal transition probabilities between winning and losing states. Specifically, the probability of transitioning from losing to winning is 46.29% and of remaining in the losing state is 53.71%, while the probability of transitioning from winning to losing is 52.30% and of remaining in the winning state is 47.70%. The fact that transition probabilities are relatively balanced between winning and losing states suggests a significant chance of changing states after each match, whether from winning to losing or vice versa. This could indicate that, contrary to the "loser queue" hypothesis where players remain stuck in a series of losses, there is a real possibility of recovering after a loss, counterbalancing the results of the first transition matrix trained on the entire dataset.

Time Series

Thus, although the initial results suggested the existence of a "loser queue," analyses of targeted features and balanced transitions between states indicate that players have a fair chance to bounce back after a defeat, countering the initial hypothesis. To try to settle the question, I decided to analyze the history as a time series. Before applying a statistical model to a time series, it is essential to ensure it is stationary. But what do we mean by stationary series? Essentially, this means that several conditions must be met:

1. The mean of the series should not be a function of time, implying that the mean should not increase over time.
2. The variance of the series should not be a function of time, indicating that the dispersion of the data should not change over time.
3. The covariance between term i and term $(i + m)$ should not vary as a function of time.

When a time series is stationary and exhibits specific behavior over a given period, it can reasonably be assumed to exhibit the same behavior at a later time. This is why most statistical modeling methods assume or require that the time series is stationary. It is therefore crucial to check the stationarity of our time series before proceeding with model building. To do this, I will use two main approaches:

Analyzing moving averages and moving standard deviations involves examining the moving averages and moving standard deviations of the time series. A series is considered stationary if these measures remain stable over time. Visually, this translates to straight lines parallel to the x-axis.
The Augmented Dickey-Fuller (ADF) and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests evaluate stationarity by analyzing the p-value associated with the null hypothesis. A low p-value suggests that the series is stationary. Additionally, ADF statistics should be close to critical values at the 1%, 5%, and 10% confidence intervals.

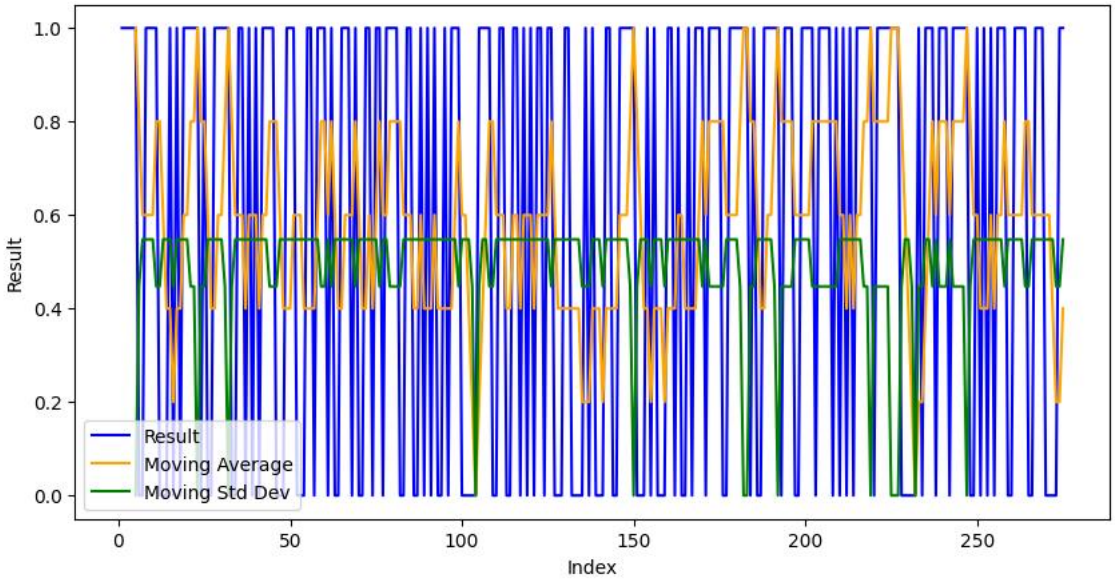


FIGURE 24 - RESULT WITH MOVING AVERAGE AND MOVING STD DEV OVER INDEX

Examining the series graph, we can see that the moving averages and moving standard deviations exhibit temporal variations. This observation suggests that the series may not be stationary. However, to confirm this hypothesis, I will use the ADF and KPSS tests.

The ADF test is used to test the null hypothesis that a time series has a unit root, meaning it is non-stationary. It does this by estimating an autoregressive regression model and testing whether the first-order lag of the time series is significantly different from zero. Unlike the ADF test, the KPSS test tests the null hypothesis that the time series is stationary around a trend or level. The test is based on estimating the variance of the errors in a regression that models the time series as a trend plus a stationary error term. If the variance of the errors is significantly different from zero, this suggests that the time series is not stationary.

```
ADF Statistique test : -11.588549310962136
p-value : 2.828409361333189e-21
Nb lags : 2
Nb observations : 272
Critical values : {'1%': -3.4546223782586534, '5%': -2.8722253212300277, '10%': -2.5724638500216264}
KPSS Statistique test : 0.2014555424668318
p-value : 0.1
Nb lags : 11
Critical values : {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}
The test statistic is outside of the range of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.
```

IMAGE 2 - RESULTS OF ADF AND KPSS TEST

The results of the ADF and KPSS tests on the series show that the series is stationary. The ADF test shows strong evidence against the presence of a unit root, and the KPSS test confirms the stationarity of the series and I can use time series forecasting models that assume the stationarity of the series.

Time series models play a crucial role in time series analysis, allowing for understanding underlying dynamics and predicting future values. Among them, Auto-Regressive (AR), Moving Average (MA), Auto-Regressive Moving Average (ARMA), and Auto-Regressive Integrated Moving Average (ARIMA) models are fundamental.

The Auto-Regressive (AR) model: An AR(p) model expresses a current value of the series as a linear combination of its p previous values plus an error term:

$$X_t = \alpha + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} + \varepsilon_t$$

EQUATION 6 - AUTO-REGRESSIVE MODEL

The Moving Average (MA) model: An MA(q) model models a current value of the time series as a linear combination of the q past forecast errors:

$$X_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$$

EQUATION 7 - MOVING AVERAGE MODEL

The Auto-Regressive Moving Average (ARMA) model: The ARMA(p, q) model combines the first two, incorporating both autoregressive and moving average terms:

$$X_t = \alpha + \phi_1 X_{t-1} + \cdots + \phi_p X_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}$$

EQUATION 8 - AUTO-REGRESSIVE MOVING AVERAGE MODEL

The Auto-Regressive Integrated Moving Average (ARIMA) model: The ARIMA(p, d, q) extends the ARMA model by adding a differencing step d to make the series stationary:

$$(1 - B)^d X_t = \alpha + \phi_1 X_{t-1} + \cdots + \phi_p X_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}$$

EQUATION 9 - AUTO-REGRESSIVE INTEGRATED AVERAGE MODEL

, where B is the backshift operator.

Determining the parameters (p, d, q) is essential to specify an ARIMA model. Their determination is based on analyzing the Auto-Correlation Function (ACF) and Partial Auto-Correlation Function (PACF):

- The ACF measures the correlation between observations of a time series separated by different numbers of periods, helping to identify the order q of the MA terms.
- The PACF measures the correlation between observations with the effect of other lags removed, helping to identify the order p of the AR terms.

The order of differencing d is determined by the number of times a series must be differenced to achieve stationarity. In our case, since no differencing was applied as the data already appeared stationary, d would be 0.

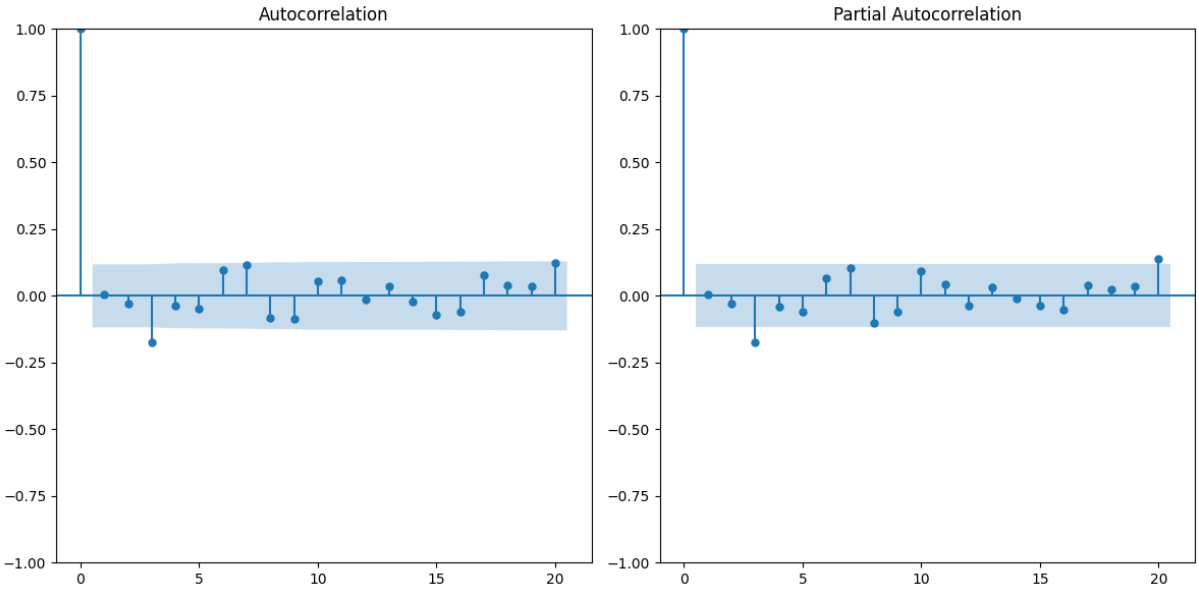


FIGURE 25 - ACF AND PACF GRAPH

The ACF graph shows the autocorrelation of lags up to 20. The autocorrelations for the first lags are significant but quickly diminish to zero, which is typical of a stationary series. The significant autocorrelation at the first lags indicates a dependence between successive observations, hat does not refute the loser queue thory. The fact that the bars fall outside the blue zone (confidence interval) at the first lags but quickly approach zero supports the idea that the series is stationary but with short-term temporal dependence. Here, p could be 1. The PACF graph reveals the partial autocorrelation for lags up to 20, controlling for the effects of other lags. As with the ACF, a few significant partial autocorrelations are observed at the first lags, which then fall within the confidence interval, indicating no strong linear correlation between observations separated by more than a few periods. Here, q could be 2.

Given the graphs, it seems there is no strong seasonality; in this case, we could take m = 1 for the SARIMA model. However, for the SARIMA model, the seasonal periodicity m must be greater than 1, so I must determine a seasonality to test this model. As I am analyzing a player's history, the number of games depends on the days. Indeed, Mata can play several games on the same day or may not play for several consecutive days, so I set a weekly seasonality m = 7. To determine the best model between the two I have adjusted, I use the AIC (Akaike Information Criterion) and BIC (Bayesian Information Criterion) criteria. These criteria allow for comparing different models, considering both the quality of the fit and the complexity of the model. The lower the AIC value, the better the model, and a lower BIC value indicates a better model.

Model	AIC	BIC
ARIMA(1, 0, 2)	395.943031	414.026887
SARIMA(1, 0, 2, 1, 0, 1, 7)	415.278908	436.979534

TABLE 1 - AIC AND BIC RESULTS

Based on the AIC and BIC criteria, the ARIMA(1, 0, 2) model is the best model among the two. It is simpler and has lower AIC and BIC values, indicating better overall performance compared to the SARIMA model.

When discussing a time series model, such as an ARIMA model, the goal is to capture the trends, seasons, and cycles present in the historical data. Residuals, which represent the differences between observed values and the values predicted by the model, are essential for evaluating what the model has failed to explain. Analyzing the residuals helps identify the elements the model has "missed." A good model should produce residuals that resemble white noise, meaning residuals that are random, independent, and follow a normal distribution with a mean of zero and constant variance.

To assess whether the residuals meet these criteria, I performed the Shapiro-Wilk normality test and the Ljung-Box test. The Shapiro-Wilk test gives a p-value less than 0.05, leading me to reject the null hypothesis that the residuals follow a normal distribution. This means the residuals are not normally distributed, suggesting that the ARIMA model has not captured all the structures of the data correctly or that the residuals contain non-normal elements. In contrast, the Ljung-Box test, with a p-value greater than 0.05, indicates that I cannot reject the null hypothesis that the residuals are independent. This means there is no significant residual correlation up to lag 20, and the residuals can therefore be considered independent.

To complement these tests, a visual analysis of the residuals is necessary, as human intuition remains irreplaceable.

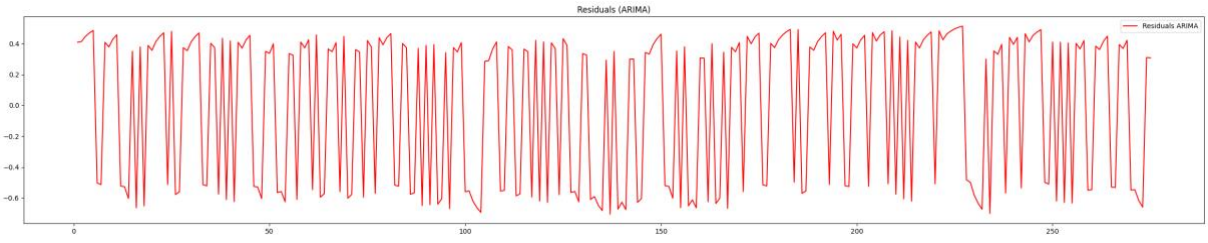


FIGURE 26 - ARIMA RESIDUALS GRAPH

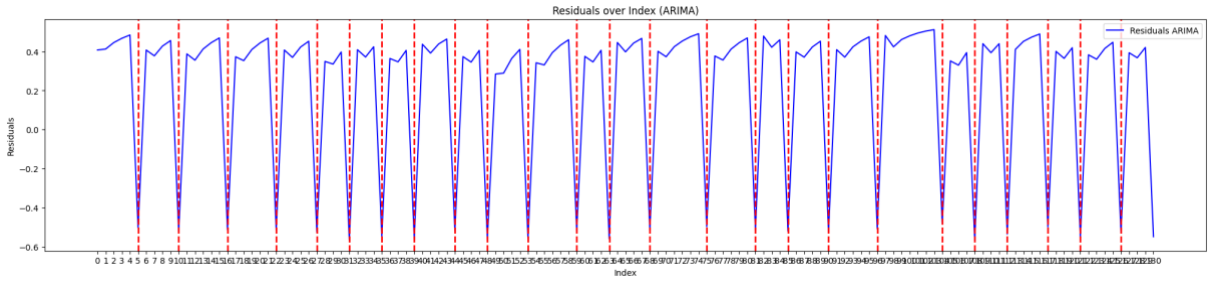


FIGURE 27 - SPLITTED ARIMA RESIDUALS GRAPH

The residuals graph shows regular and periodic variations, confirmed by the graph's segmentation, suggesting several possible interpretations. On the one hand, it may be that the ARIMA model has not completely captured the structures present in the data, or the data itself presents recurring patterns that cannot be captured by a standard ARIMA model. This could indicate that the ARIMA model is not suitable for this data, necessitating the trial of more complex models like SARIMA, or the inclusion of additional explanatory variables. However, the SARIMA model we have already trained is less performant than the ARIMA. On the other hand, if the data is not random and presents systematic patterns, this could support the "loser queue" hypothesis, where losing players are placed in situations where they continue to lose.

Conclusion

In conclusion, my study on the game League of Legends has highlighted several crucial aspects of this domain. Through various analyses and the use of advanced models, I determined that the support and jungle roles are more influential for a team's victory than other roles.
Using SHAP values to understand the impact of each variable, it became apparent that direct combat performance, such as KDA ratio, gold per minute, and damage dealt to objectives, are essential for determining the outcome of a match, especially in the support role. Moreover, Monte Carlo simulations confirmed that some of these characteristics significantly affect the probability of victory.

The use of hidden Markov models to explore the existence of a "loser queue" showed mixed results. While the initial transition matrix suggested a high probability of remaining in a losing state, further analysis revealed balanced chances of transitioning between victory and defeat, challenging the initial hypothesis.

Finally, time series analysis allowed for the examination of residuals and the testing of series stationarity, demonstrating the complexity of game dynamics. Although ARIMA and SARIMA models were performant, they revealed recurring patterns in the residuals, suggesting that the data is not random.

Integrating external factors such as player behavior, more precise in-game communication data, and even psychological data could offer a more holistic and nuanced view of the reasons behind victories and defeats. This could not only improve model performance but also contribute to a better understanding of human behavior in virtual environments.

It is time to conclude. So, which side am I on? Am I on the side of those who believe the "loser queue" exists, or the other side? If I have to decide, I think that, based on the results of this report, believing in the existence of a "loser queue" is not unreasonable. And, it is easier to accept defeat if we blame the "loser queue."

This project has been very enjoyable for me. I thoroughly enjoyed developing the scripts, testing and discovering new analysis methods, and especially observing the results.

Appendix

The details and meanings of all variables

Variable	Definition
Match_ID	Unique identifier for the match
GameVersion	Version of the game played
puuid	Unique identifier for the player
Team	Team identifier (e.g., Blue or Red)
GameTime	Total duration of the game
result	Match result (win or loss)
individualPosition	Player's position in the game (e.g., Top, Jungle)
championName	Name of the champion played by the player
kda	Kill/Death/Assist ratio
kills	Number of kills made by the player
deaths	Number of times the player died
assists	Number of assists made by the player
SoloKills	Number of solo kills achieved
quickSoloKills	Number of quick solo kills achieved
killsUnderOwnTurret	Kills made under own turret
killsNearEnemyTurret	Kills made near enemy turret
firstBloodAssist	Whether the player assisted in getting the first blood
firstBloodKill	Whether the player got the first blood
doubleKills	Number of double kills achieved
tripleKills	Number of triple kills achieved
quadraKills	Number of quadra kills achieved
pentaKills	Number of penta kills achieved
killingSprees	Number of killing sprees achieved
turretKills	Number of turrets killed
quickFirstTurret	Whether the player got the first turret quickly
firstTowerAssist	Whether the player assisted in getting the first tower
firstTowerKill	Whether the player got the first tower kill
turretPlatesTaken	Number of turret plates taken
kTurretsDestroyedBeforePlatesFall	Number of turrets destroyed before plates fall
champLevel	Player's champion level at the end of the game
visionScore	Score based on vision wards placed
visionScoreAdvantageLaneOpponent	Advantage in vision score over lane opponent
visionScorePerMinute	Vision score per minute
totalMinionsKilled	Total number of minions killed
laneMinionsFirst10Minutes	Number of lane minions killed in the first 10 minutes
neutralMinionsKilled	Number of neutral minions killed
skillshotsDodged	Number of skillshots dodged

skillshotsHit	Number of skillshots hit
dodgeSkillShotsSmallWindow	Number of dodged skillshots in a small window
goldPerMinute	Gold earned per minute
goldEarned	Total gold earned by the player
goldSpent	Total gold spent by the player
laningPhaseGoldExpAdvantage	Advantage in gold and experience during laning phase
damagePerMinute	Damage dealt per minute
magicDamageDealt	Total magic damage dealt
magicDamageDealtToChampions	Magic damage dealt to enemy champions
magicDamageTaken	Total magic damage taken
physicalDamageDealt	Total physical damage dealt
physicalDamageDealtToChampions	Physical damage dealt to enemy champions
physicalDamageTaken	Total physical damage taken
trueDamageDealt	Total true damage dealt
trueDamageDealtToChampions	True damage dealt to enemy champions
trueDamageTaken	Total true damage taken
totalDamageDealt	Total damage dealt
totalDamageDealtToChampions	Total damage dealt to enemy champions
totalDamageTaken	Total damage taken
damageDealtToBuildings	Damage dealt to buildings
damageDealtToObjectives	Damage dealt to objectives
damageDealtToTurrets	Damage dealt to turrets
totalTimeSpentDead	Total time spent dead
totalHeal	Total health restored
totalHealsOnTeammates	Total health restored on teammates
saveAllyFromDeath	Whether the player saved an ally from death
allInPings	Number of "All In" pings
assistMePings	Number of "Assist Me" pings
basicPings	Number of basic pings
commandPings	Number of command pings
dangerPings	Number of danger pings
enemyMissingPings	Number of enemy missing pings
enemyVisionPings	Number of enemy vision pings
getBackPings	Number of "Get Back" pings
holdPings	Number of "Hold" pings
needVisionPings	Number of "Need Vision" pings
onMyWayPings	Number of "On My Way" pings
pushPings	Number of "Push" pings
visionClearedPings	Number of vision cleared pings
SummonerID	Unique identifier for the summoner
elo	Player's ranking in the game
totalPings	Total number of pings

MinionsKilledPerMin	Minions killed per minute

Guwon analysis

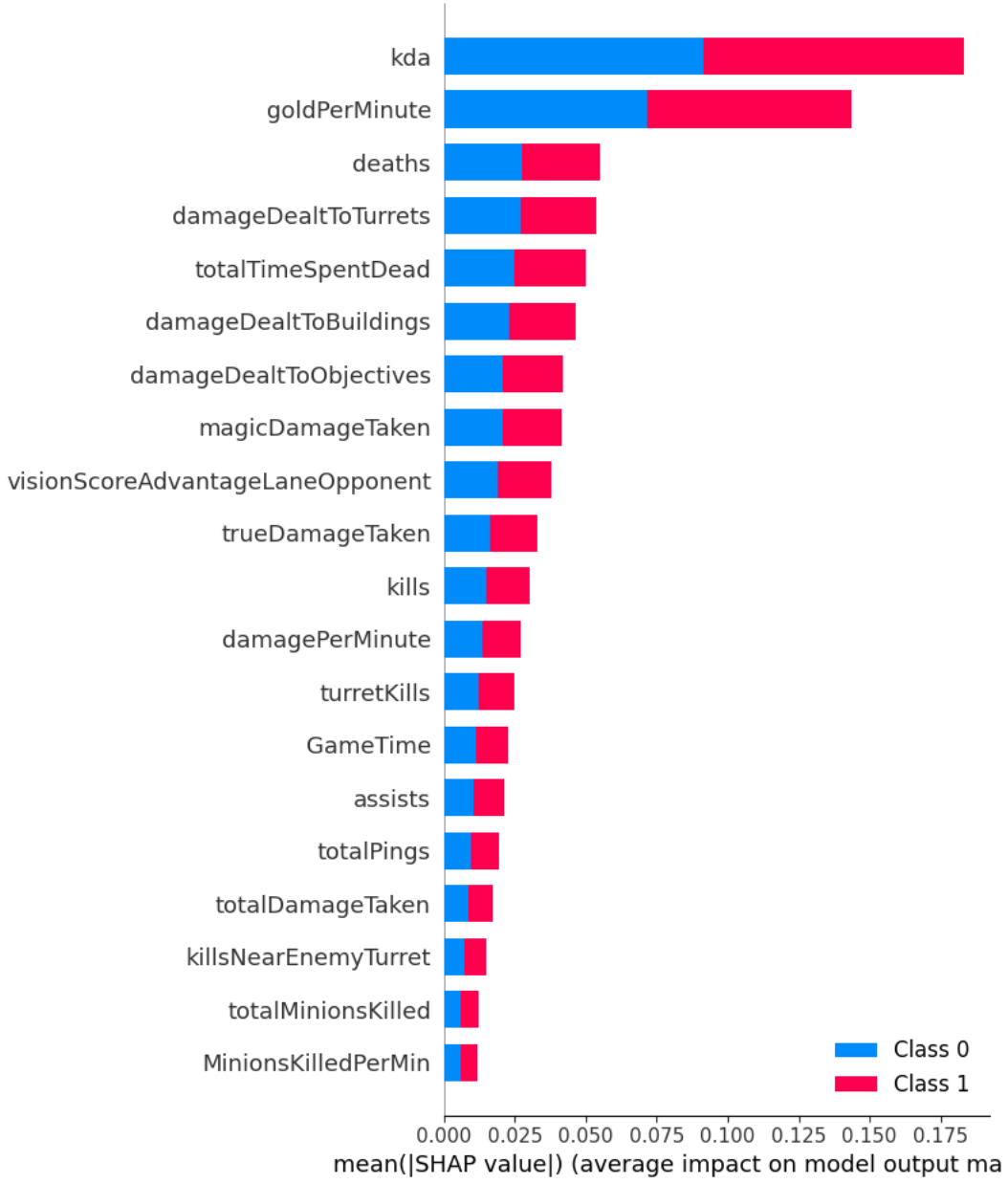


FIGURE 28 - GUWON SHAP SUMMARY PLOT

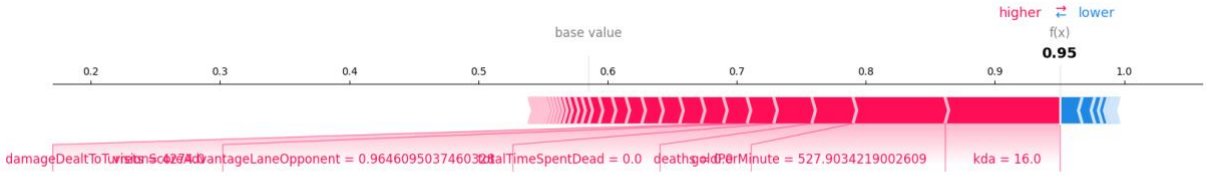


FIGURE 29 - GUWON SHAP FORCE PLOT

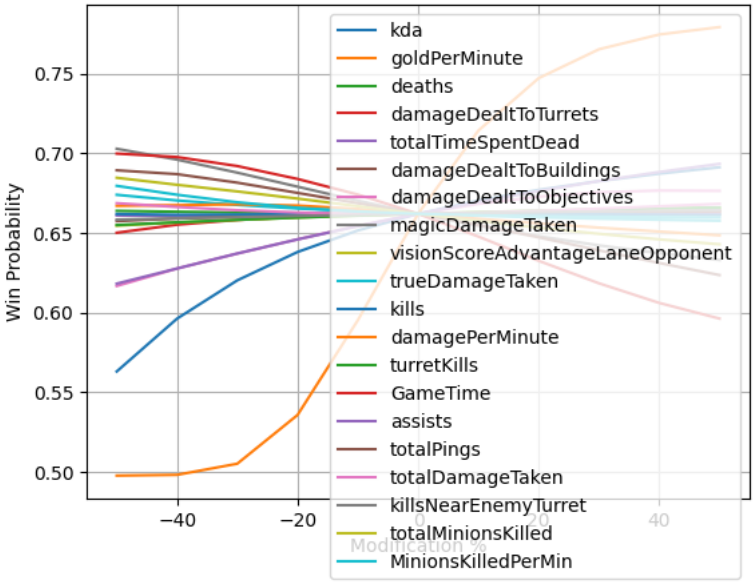


FIGURE 30 - GUWON IMPACT OF FEATURES CHANGES ON WIN PROBABILITY

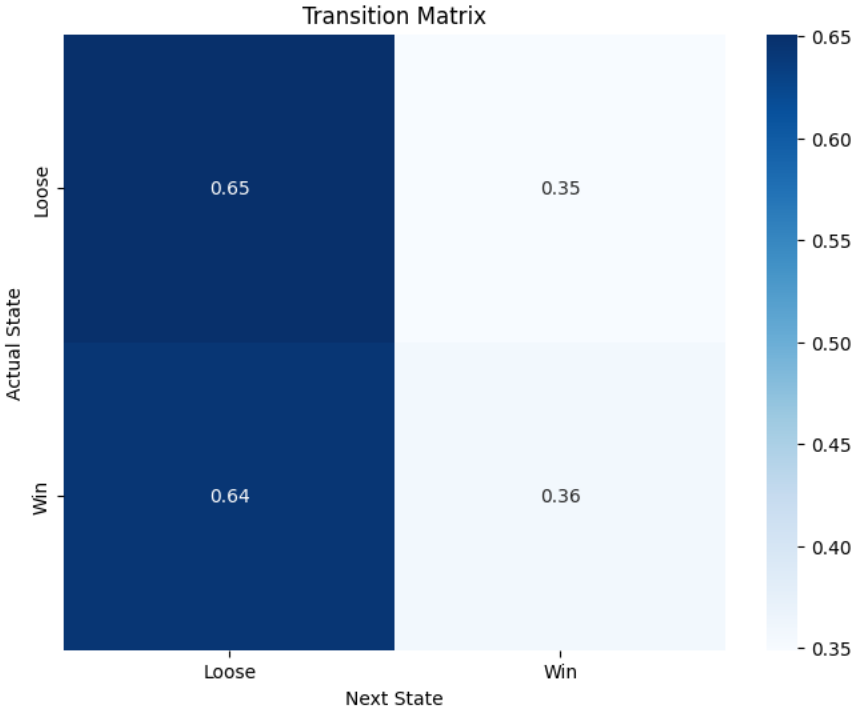


FIGURE 31 - GUWON TRANSITION MATRIX OF THE WHOLE DATASET

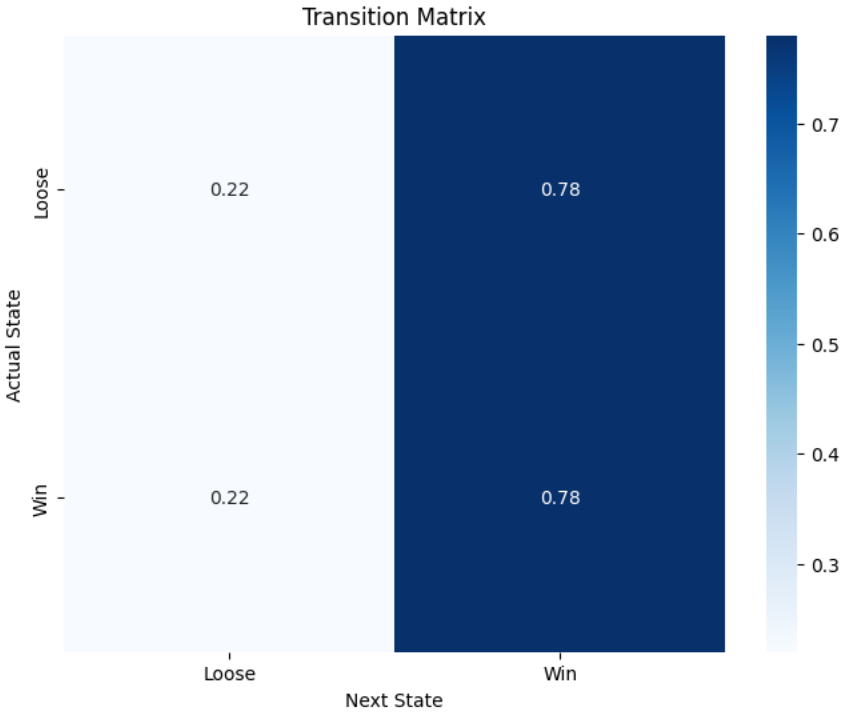


FIGURE 32 - GUWON TRANSITION MATRIX ON FILTERED FEATURES

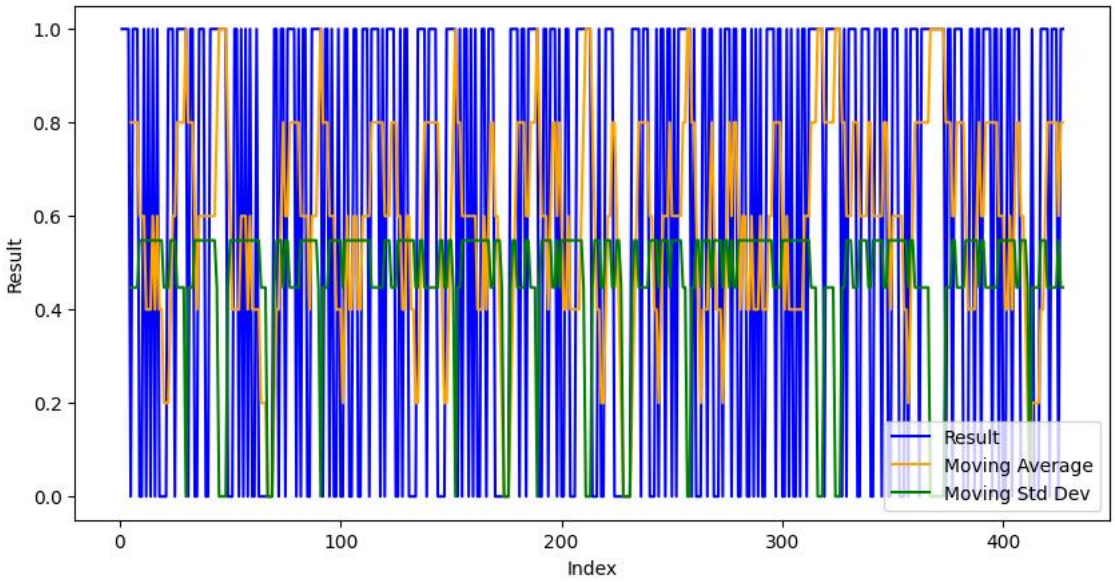


FIGURE 33 - GUWON RESULT WITH MOVING AVERAGE AND MOVING STD DEV OVER INDEX

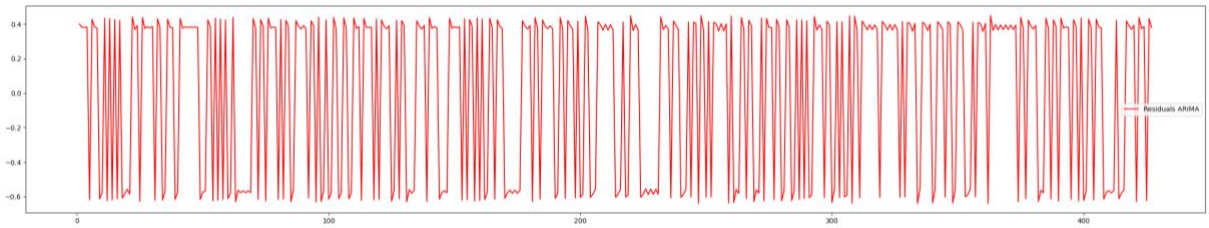


FIGURE 34 - GUWON ARIMA RESIDUALS GRAPH

References

Equation

Figure

Image

Table