# Analyse des Champions de League of Legends

Groupe 11

APARICIO Maxime, AMAZIT Alan, CAPIEU Valentin, POTTIER Anne, SEIGNOUX Vivien

## Table des matières

# Introduction

League of Legends est un jeu vidéo sorti en 2009 de type arène de bataille en ligne développé et édité par Riot Games.
Le déroulement d'une partie est le suivant : deux équipes de cinq joueurs s'affrontent dans le but de détruire la base ennemie, appelée *Nexus*.

Chacun des 10 joueurs contrôle un champion parmi les 169 disponibles, chacun ayant des caractéristiques qui lui sont propres. Une fois la partie lancée, il est nécessaire pour chaque joueur d'améliorer son champion dans l'optique de combattre les champions ennemis. Cette amélioration passe par l'achat d'objets variés au fur et à mesure de la partie. Ces objets octroient des statistiques bonus, comme des dégâts, de la résistance, des points de vie, de la vitesse de déplacement ou encore du mana (et bien d'autres). Ainsi, chacun de ces objets est naturellement plus adapté à une ou plusieurs catégories de champion.

Il existe différentes classes de champions, regroupant les personnages partageant des caractéristiques similaires. C'est sur ces classes que nous allons porter notre attention dans le cadre de ce projet, puisque notre objectif final est le suivant : **déterminer quel champion de *League of Legends* est le meilleur au sein de la classe à laquelle il appartient**.

Pour cela, nous avons divisé notre travail en deux grandes parties :
— Identification des différents groupes de champions,
— Classement des champions au sein de leur propre groupe.

# 1 Identification des différents groupes de champions

Dans cette partie, il est question de récupérer des clusters de champions. Pour cela, on appliquera des méthodes de clustering différentes et nous déterminerons celui que nous garderons afin de passer à l'étape suivante.

## 1.1 Items

Les champions semblent se distinguer par les items qu'ils choisissent. Explorons cette idée en définissant des clusters de champions basés sur leurs choix d'items.

### 1.1.1 Dataset

Commençons par ce premier point, pour cela il nous faut récupérer les données dont nous avons besoin. En l'occurrence il nous faut accéder aux parties des meilleurs joueurs du globe et extraire ces données : *Match ID*, *Nom du Champion*, *item1*, *item2*, *item3*, *item4*, *item5*, *item6*. Ces variables correspondent à l'identificateur du match (qui nous sera très utile à l'avenir lorsque nous contacterons des datasets de joueurs différents), le nom du champion joué et les 6 items qu'il a choisi.

Nous obtenons 10 observations de ce type par partie récupérée (dû à la présence de 10 joueurs par partie).
N'oublions pas qu'il nous faut adapter ces données à la forme du dataset final, ce travail est fait dans le programme "etape1_recherche.py".

| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | MatchID | Champ Nam | Dark Seal | Tear | Doran Ring | Doran Blade | Cull | Doran Shield | Red Jung | Blue Jung | Green J |
| 2 | BR1_3035355 | Gangplank | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 3 | BR1_3035355 | MasterYi | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 4 | BR1_3035355 | Akali | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| 5 | BR1_3035355 | Ezreal | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | |
| 6 | BR1_3035355 | Senna | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 7 | BR1_3035355 | Gragas | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 8 | BR1_3035355 | LeeSin | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 9 | BR1_3035355 | Ekko | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 10 | BR1_3035355 | Corki | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | |
| 11 | BR1_3035355 | Soraka | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 12 | BR1_3035314 | Trundle | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| 13 | BR1_3035314 | Ekko | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 14 | BR1_3035314 | Qiyana | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 15 | BR1_3035314 | Corki | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | |
| 16 | BR1_3035314 | Sona | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 17 | BR1_3035314 | Pantheon | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

Nous aurons donc chaque colonne qui représentera un item potentiellement pris, avec la valeur 1 attribuée si tel est le cas et 0 sinon.

A la suite de cela, nous disposons d'un grand nombre de fichiers csv de joueurs différents. Par conséquent, nous concaténons tous ces fichiers en 1 final. N'oublions pas que, puisque ces joueurs ont très probablement joué des parties ensemble, il nous faut respecter la clé primaire de ce tableau : (MatchID, Champ Name). Ainsi, toute partie qui pourrait apparaître 2 fois voit ses doublons retirés. Au final, c'est un peu plus de 12 000 parties que nous avons recueilli, et donc plus de 120 000 observations ce qui fait, en moyenne, 720 observations par champion, fait dans le programme "etape1_concat.py"

Nous pouvons nous débarrasser de la colonne MatchID pour la suite.

Dans une optique d'appliquer une PCA, il nous faut un dataset avec une ligne par champion. Pour cela, nous avons décidé de ne garder que la moyenne d'apparition des items pour chaque champion. Le tableau obtenu ne comporte plus que 170 lignes, fait dans le programme "etape1_groupement.py" :

| | A | B | C | D | E | F | G | H | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Champ Nam | Dark Seal | Tear | Doran Ring | Doran Blade | Cull | Doran Shield | Red Jung | Blue J |
| 2 | Aatrox | 0.0 | 0.0 | 0.0 | 0.14181818 | 0.00242424 | 0.43272727 | 0.0 | 0.0 |
| 3 | Ahri | 0.39313244 | 0.00070077 | 0.49404344 | 0.00070077 | 0.00140154 | 0.0 | 0.0 | 0.0 |
| 4 | Akali | 0.44425817 | 0.0 | 0.19782062 | 0.00083822 | 0.00083822 | 0.26320201 | 0.0 | 0.0 |
| 5 | Akshan | 0.0 | 0.0 | 0.0 | 0.54775280 | 0.01685393 | 0.0 | 0.0 | 0.0 |
| 6 | Alistar | 0.00271002 | 0.00135501 | 0.0 | 0.0 | 0.0 | 0.00406504 | 0.0 | 0.0 |
| 7 | Ambessa | 0.0 | 0.0 | 0.0 | 0.19834282 | 0.00517866 | 0.29984464 | 0.0 | 0.0 |
| 8 | Amumu | 0.06338028 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 9 | Anivia | 0.10629921 | 0.17322834 | 0.32283464 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 10 | Annie | 0.27659574 | 0.0 | 0.44680851 | 0.0 | 0.01063829 | 0.0 | 0.0 | 0.0 |
| 11 | Aphelios | 0.0 | 0.0 | 0.0 | 0.63840830 | 0.05017301 | 0.0 | 0.0 | 0.0 |
| 12 | Ashe | 0.00040371 | 0.00322971 | 0.0 | 0.66330238 | 0.01291885 | 0.0 | 0.0 | 0.0 |
| 13 | AurelionSol | 0.24215246 | 0.10313901 | 0.46188340 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 14 | Aurora | 0.36744966 | 0.00055928 | 0.50727069 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 15 | Azir | 0.23036649 | 0.0 | 0.59162303 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 16 | Bard | 0.00956937 | 0.00119617 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 17 | Belveth | 0.0 | 0.0 | 0.0 | 0.02579365 | 0.00396825 | 0.00396825 | 0.0 | 0.0 |

### 1.1.2 Clustering

Riot Games sépare ses champions en 6 catégories : assassin, combattant, mage, tireur, support et tank. Seulement, ses catégories sont obsolètes et ne parviennent pas à réaliser un paysage cohérent des champions habitant la faille de l'invocateur. Nous avons donc décidé d'augmenter le nombre de catégories, passant de 6 à 10. Notre objectif est désormais de parvenir à réaliser 10 clusters de champion. Nous avons essayé 3 méthodes : K-means, hierarchical labels et spectral labels. Nous avons aussi effectué un Branch and Bound afin d'avoir un d'avoir un clustering minimisant les distances intra cluster, ce qui nous permettrait d'avoir une solution globalement optimale. Malheureusement, le temps d'exécution étant trop élevé, nous n'avons pas pu avoir de résultats, voir programme "etape1_Branch_and_Bound.py".

Après analyse, 7 clusters sur 10 sont identiques avec les trois méthodes. Ils comprennent :

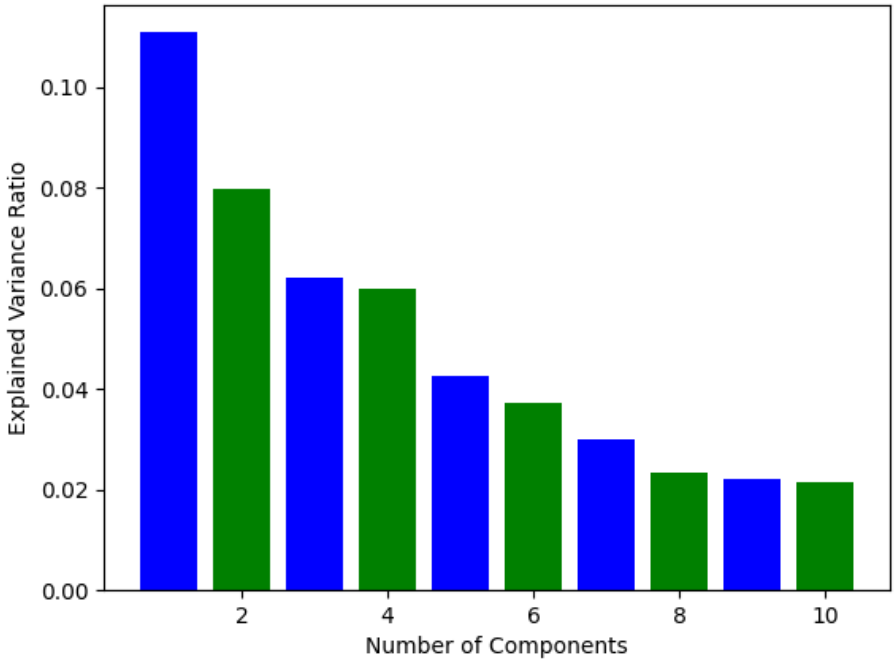
— un groupe *on-hit*,
— combattant (physique),
— tank,
— support frontline,

— support enchanteur,
— assassin (physique),
— ADC.

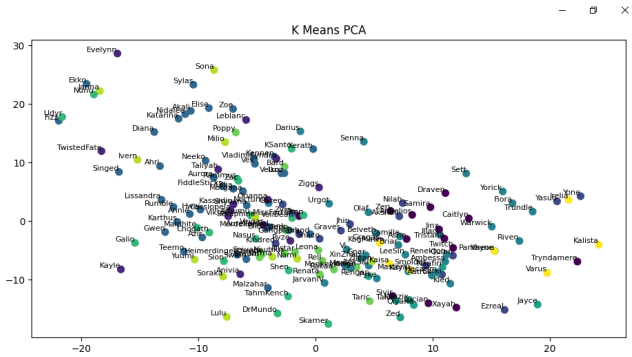
Les méthodes K-means et hierarchical labels ont ensuite créer un cluster pour les champions infligeant des dégâts physique basés sur des objets tels que Muramana, Trinité, Shojin, puis les champions mage ont été réparti au sein des 2 derniers clusters restant d'une manière un peu douteuse. Alors que la méthode spectral label à directement réparti les mages en 3 catégories cohérentes : les combattants (magique cette fois), les mages utilisant beaucoup de mana, et les mages brûlant (utilisation abusive de l'objet Tourment de Liandry, ayant pour effet de brûler les adversaires).

### 1.1.3 PCA

Pour pouvoir afficher nos résultats dans un graphe en 2D et ainsi observer les distanciations entre ces clusters, nous appliquons donc une PCA sur le jeu de données.




Dû au grand nombre de colonnes, la variance totale expliquée par les deux premiers vecteurs n'est que de 19%. Traçons tout de même le résultat :



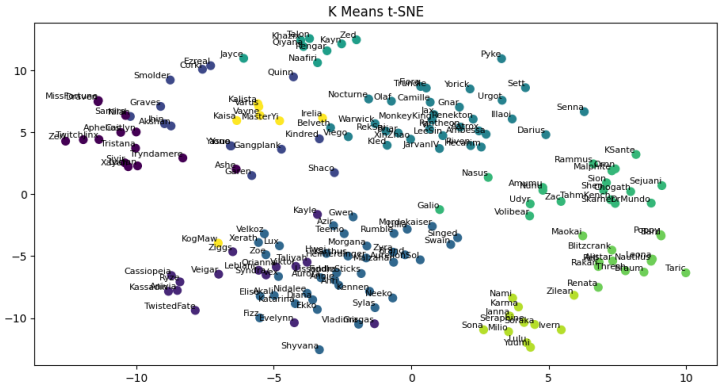

En effet, le résultat n'est pas assez bon, les clusters sont éparpillés dans tous les sens, et les positions des champions sont véritablement mauvaises : Lulu (support fragile) à côté de Mundo (tank très solide) par

exemple.

### 1.1.4 t-SNE

Nous voyons que les clusters se chevauchent ce qui suggère que des points distants dans l'espace en grande dimension sont proches dans la représentation 2D. Nous avons donc tenté une approche non-linéaire : le t-SNE. Cette approche veille à ce que les points distants dans l'espace en grande dimension reste distant dans la représentation 2D.

Voici les résultats obtenus :




On obtient ces tracés à l'aide du programme "etape1_analyse.py"

### 1.1.5 Analyse

En traçant la ligne d'équation $y = 0$, nous observons une séparation nette des types de champions :

— Au-dessus de cette ligne : les champions infligent majoritairement des dégâts physiques,

— En dessous : les champions infligent des dégâts magiques.

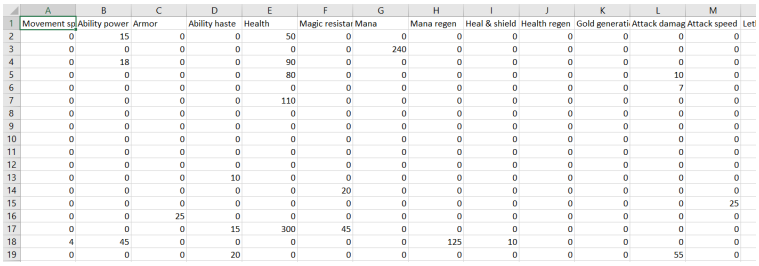
De plus, l'axe des abscisses semble corrélé à la résistance et à la tankiness. En effet plus un champion à une abscisse élevé, plus il sera tanky. Par exemple, au-delà de $x = 7$, seuls des tanks sont présents.

## 1.2 Statistiques des items

Une autre possibilité consiste à classer les champions en fonction des statistiques conférées par les items choisis. On admet une perte d'information pour simplifier l'étude.

### 1.2.1 Dataset

Nous voyons ainsi que le grand nombre de dimensions du dataset relatif (nous avons considéré 117 items et donc 117 colonnes) abaisse la qualité de la PCA. De plus, en grande dimensions, il y a de forte chance que tous les points soient équidistants. Nous avons donc réduit le nombre de dimensions en ne prenant non pas les items mais les statistiques données par les items. Il y a 23 statistiques et donc, 23 colonnes dans ce nouveau dataset. Pour obtenir ce dataset il est possible de refaire une extraction (voir le programme extraction multiplayer) cependant, une extraction peut être chronophage, d'autant plus qu'il existe une autre possibilité : reprendre le dataset des items (voir le programme "etape1_item_to_stats.py"). Nous avons donc créé une matrice permettant de passer des items aux statistiques procurées ( ITS, pour Items To Stat ). Les lignes correspondent aux items, tandis que les colonnes représentent les statistiques qu'ils confèrent :



| | A | B | C | D | E | F | G | H | I | J | K | L | M | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Movement sp | Ability power | Armor | Ability haste | Health | Magic resista | Mana | Mana regen | Heal & shield | Health regen | Gold generati | Attack damag | Attack speed | Let |
| 2 | 0 | 15 | 0 | 0 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 240 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 4 | 0 | 18 | 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 5 | 0 | 0 | 0 | 0 | 80 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | |
| 7 | 0 | 0 | 0 | 0 | 110 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 13 | 0 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 14 | 0 | 0 | 0 | 0 | 0 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 25 | |
| 16 | 0 | 0 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 17 | 0 | 0 | 0 | 15 | 300 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 18 | 4 | 45 | 0 | 0 | 0 | 0 | 0 | 125 | 10 | 0 | 0 | 0 | 0 | |
| 19 | 0 | 0 | 0 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 55 | 0 | |

On remarque, par exemple, que le 3e item donne 18 ability power et 90 Health. Ainsi, en faisant un produit matriciel entre le dataset des items et la matrice de passage ITS, on obtient le dataset des statistiques.

### 1.2.2 Clustering

Nous avons appliqué les mêmes méthodes de clustering que dans la première partie. Les clusters réalisés sont globalement identiques.

Néanmoins pour K-means nous observons l'apparition d'un cluster ne contenant que 2 champions : [Ivern et Yummi]. Je ne parviens pas à expliquer pourquoi ses deux champions ne sont pas reliés aux groupes des support enchanteurs étant données qu'ils achètent exactement les mêmes stats. Autre fait intéressant, le champion Mordekaiser, un combattant infligeant des dégâts magiques, s'est retrouvé dans le cluster des combattant infligeant des dégâts physiques. Cela fait sens étant donné que ses items lui confère (mis à part la stat offensive qui diffère) de la réduction de délais, de la résistance ainsi que des points de vie, tout comme les combattants physique en somme.

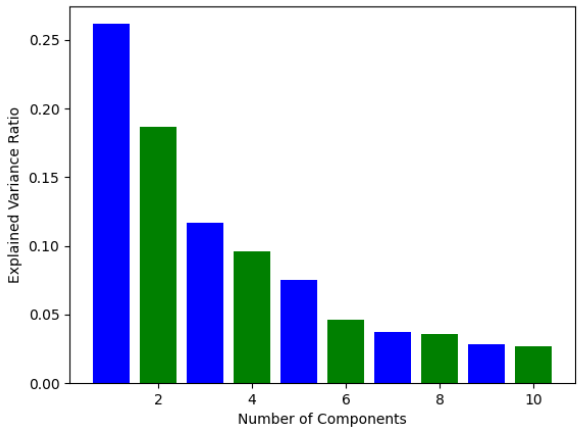
Pour la méthode hierarchical labels, nous observons que l'un des cluster s'est un peu "décentré", le cluster des champions ON-HIT qui, en faisant sur les stats, accueille aussi des champions de type combattant physique accès sur la vitesse d'attaque tels que fiora trundle ou camille. Nous avons par contre énormément de mal à expliquer la présence d'olaf au sein de ce cluster. Un autre cluster s'est un peu plus refermé sur lui-même, celui des adc/crit champions. Ce qui a forcé la création d'un cluster [ashe/garen/zeri] qui est au milieu des deux. Il y a toujours une séparation flou entre les 2 clusters mage qui est assez complexe à expliquer.

Enfin, pour la méthode spectral labels, les différentes séparations sont identiques, nous n'observons aucune différence majeure entre le clustering sur les stats ou sur les items. Nous pouvons tout de même noter quelques différences mineures, on a une contraction du cluster adc, un gonflement pour celui des tanks et des assassins, parfois des échanges de champions aux frontières (Kindred et Tryndamere par ex). Il est intéressant de noter que la séparation des mages se fait de manière plus brouillon, les stats à elle seule ne parviennent pas capturer les effets passifs fort des items de mage (celui de tourment de liandry ou sceptre de rylai qui ont une réelle importance lors du choix des items et qui sont invisible lorsqu'on passe aux stats).

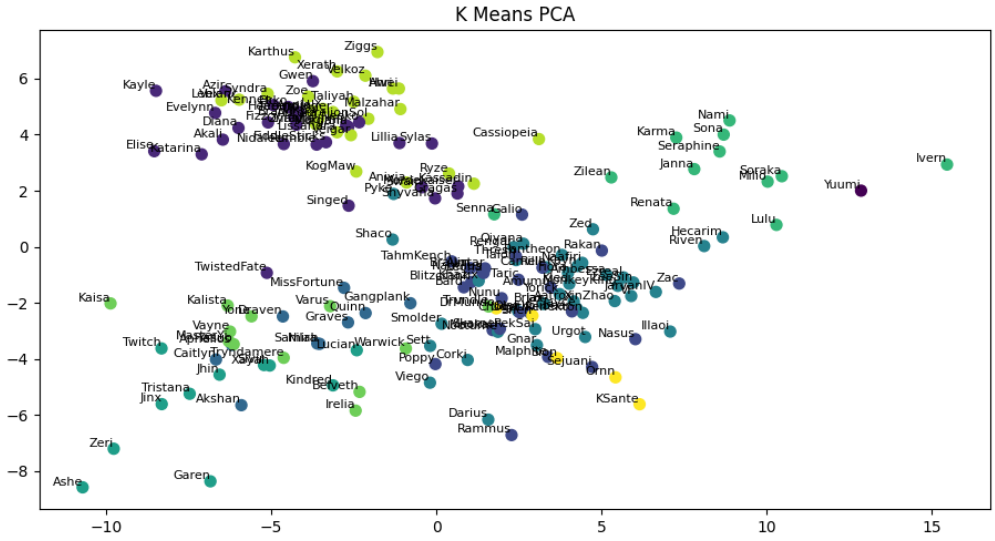
Nous pouvons désormais affirmer que la méthode de clustering permettant de représenter le mieux le paysage de la faille de l'invocateur est spectral label sur les items. C'est sur les clusters de cette méthode que nous continuons notre analyse.

### 1.2.3 PCA

La PCA appliquée aux statistiques offre des résultats plus satisfaisants. Déjà, le total de la variance expliquée contenues dans les 2 premiers vecteurs est cette fois-ci de 44%.



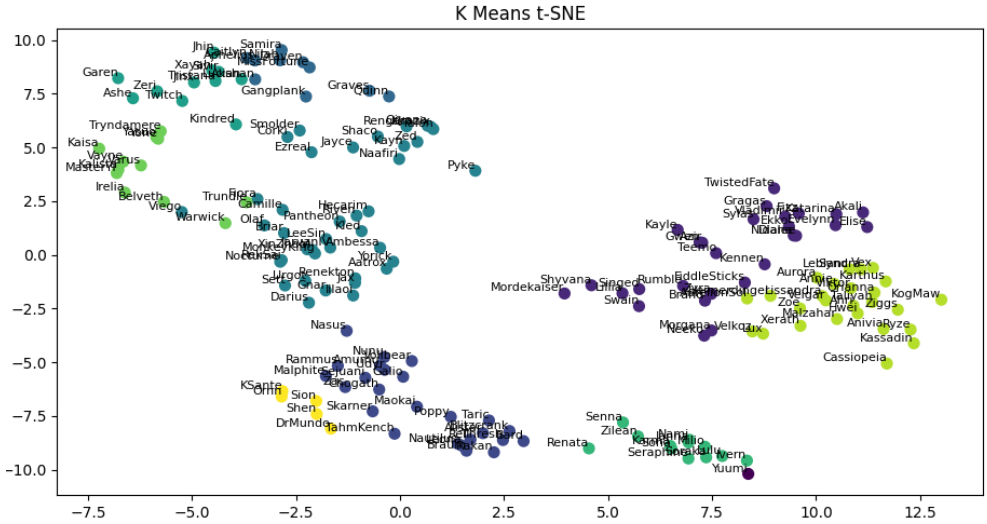

Par ailleurs, de façon générale, les résultats sont plus vraisemblables :




Nous remarquons que les tanks sont situés au centre et les attaquants, mages et supports gravitent autour. Toutefois même si l'on se rapproche du résultat désiré (Lulu n'est plus voisine de Dr Mundo), les positions des champions restent assez brouillon, nous pouvons penser notamment à Pyke (support assassin) qui se trouve juste à côté d'Anivia (mage).

### 1.2.4 t-SNE

Pareillement à la première partie, les clusters se chevauchent, nous avons aussi appliqué un t-SNE pour la visualisation.




### 1.2.5 Analyse

Nous pouvons observer comme dans la section I.1.e une séparation nette entre les champions infligeant des dégâts magiques et ceux infligeant des dégâts physiques. Si x>2.5, le champion est un mage. Les champions tanks se retrouvent avec une ordonnée inférieure à 3. Le t-SNE permet de conserver des distances raisonnables entre les champions similaires, ce qui facilite la lecture et l'analyse. Pour obtenir ces graphes, nous réutilisons le programme "etape1_analyse.py".

## 1.3 Autre approche possible : la FCA

Toujours dans l'objectif de visualiser les relations entre les champions grâce aux items utilisés, nous utilisons une FCA (Analyse des Composantes Factorielles) sur un dataset qui nous sert de table de contingence avec les items en colonnes et les champions en lignes et un compte du nombre de fois où l'item a été choisi par le champion.

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Champ Nam | Dark Seal | Tear | Doran Ring | Doran Blade | Cull | Doran Shield | Red Jung | Blue Jung | Green Ju |
| 2 | Aatrox | 0 | 0 | 0 | 117 | 2 | 357 | 0 | 0 | |
| 3 | Ahri | 561 | 1 | 705 | 1 | 2 | 0 | 0 | 0 | |
| 4 | Akali | 530 | 0 | 236 | 1 | 1 | 314 | 0 | 0 | |
| 5 | Akshan | 0 | 0 | 0 | 195 | 6 | 0 | 0 | 0 | |
| 6 | Alistar | 2 | 1 | 0 | 0 | 0 | 3 | 0 | 0 | |
| 7 | Ambessa | 0 | 0 | 0 | 383 | 10 | 579 | 0 | 0 | |
| 8 | Amumu | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 9 | Anivia | 27 | 44 | 82 | 0 | 0 | 0 | 0 | 0 | |
| 10 | Annie | 26 | 0 | 42 | 0 | 1 | 0 | 0 | 0 | |
| 11 | Aphelios | 0 | 0 | 0 | 369 | 29 | 0 | 0 | 0 | |
| 12 | Ashe | 1 | 8 | 0 | 1643 | 32 | 0 | 0 | 0 | |
| 13 | AurelionSol | 54 | 23 | 103 | 0 | 0 | 0 | 0 | 0 | |
| 14 | Aurora | 657 | 1 | 907 | 0 | 0 | 0 | 0 | 0 | |
| 15 | Azir | 88 | 0 | 226 | 0 | 0 | 0 | 0 | 0 | |

Ainsi, notre code ("fca.py") s'assure de :

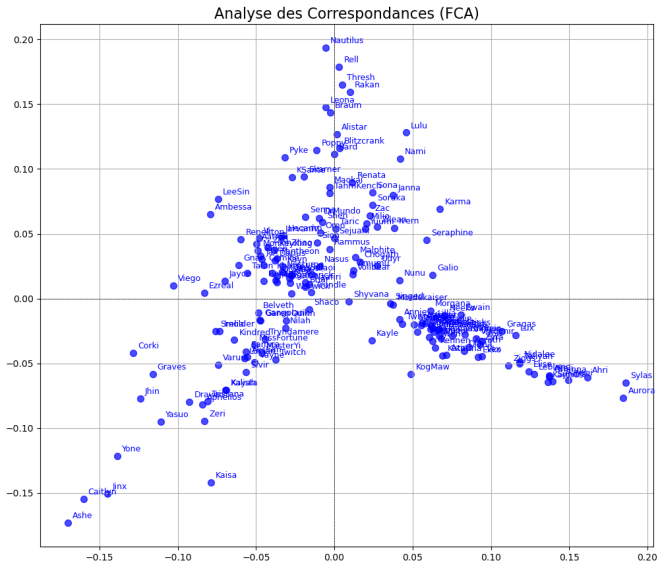
— normaliser les données pour transformer toutes les données en fréquences relatives, c'est-à-dire que chaque cellule correspond maintenant à la fréquence du temps où le champion utilise l'item

— calculer pour chaque champion quelle proportion il représente de l'utilisation totale des items et pour chaque item, quelle proportion ce dernier représente de l'utilisation totale des items

— centrer les données grâce aux calculs précédents

— pondérer les données pour s'assurer que les données ne soient pas biaisées par l'utilisation plus populaire d'un item ou d'un champion

— décomposer la matrice pondérée pour la simplifier selon des grands axes et grâce à cette décomposition, calculer les coordonnées dites factorielles des champions.




Comme nous pouvons le constater, nous observons une forme d'étoile à 3 branches. La branche en bas à gauche est composée des champions infligeant des dégâts physiques, La branche du haut représente les tanks et les supports, et la branche de droite les mages. Pour faire simple, tout champion ayant une abscisse négative inflige des dégâts physiques, tout champion ayant une abscisse positive inflige des dégâts magiques. Une abscisse proche de 0 signifie que le champion inflige à peu près autant de dégâts magiques que de dégâts physiques.

Certains champions sont entre deux branches, ou tout simplement proches du centre. Cela signifie que ces champions ont différentes qualités, par exemple Lee Sin est un champion infligeant des dégâts physiques mais aussi très résistant tout comme les tanks, il est donc placé à gauche et assez haut. Il en va de même pour Shyvana, placée au milieu de la PCA, ce champion est très particulier vu qu'il achète à la fois des objets pour les combattants physique et à la fois des objets pour les combattants magique ; il en résulte un champion résistant, infligeant à la fois des dégâts magique et physiques.

Nous pouvons également observer des agglomérations de points qui rendent l'interprétabilité difficile.

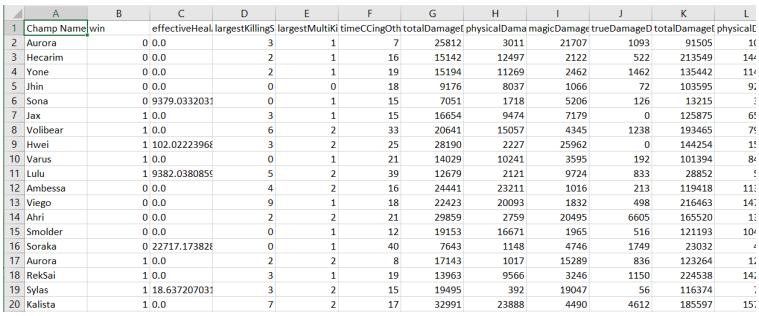
# 2 Classements des champions au sein de leur propre groupe

Dans cette section, nous analyserons quels champions se distinguent par leur capacité à regrouper les statistiques les plus déterminantes pour la victoire, ainsi que ceux qui, à l'inverse, se révèlent moins performants par rapport à leurs homologues au sein du même groupe.

## 2.1 Dataset

Pour la confection de ce dataset, on collecte les données de fin de partie des joueurs parmi un large éventail de statistiques, en se concentrant sur celles jugées déterminantes pour la victoire.

Nous sommes donc réduit à une vingtaine de statistiques auxquelles on ajoute évidemment le **Match ID**, le **Nom du Champion** et pour finir une colonne **Victoire** (1 pour une victoire, 0 pour une défaite) à l'aide du programme "endgame_extractor.py". Le processus d'extraction des données est semblable à ceux effectués précédemment.



| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Champ Name | win | effectiveHeal | largestKilling | largestMultiKi | timeCCingOth | totalDamageI | physicalDama | magicDamage | trueDamageD | totalDamageI | physicalI |
| 2 | Aurora | 0 | 0.0 | 3 | 1 | 7 | 25812 | 3011 | 21707 | 1093 | 91505 | 1 |
| 3 | Hecarim | 0 | 0.0 | 2 | 1 | 16 | 15142 | 12497 | 2122 | 522 | 213549 | 14 |
| 4 | Yone | 0 | 0.0 | 2 | 1 | 19 | 15194 | 11269 | 2462 | 1462 | 135442 | 11 |
| 5 | Jhin | 0 | 0.0 | 0 | 0 | 18 | 9176 | 8037 | 1066 | 72 | 103595 | 9 |
| 6 | Sona | 0 | 9379.033203 | 0 | 1 | 15 | 7051 | 1718 | 5206 | 126 | 13215 | |
| 7 | Jax | 1 | 0.0 | 3 | 1 | 15 | 16654 | 9474 | 7179 | 0 | 125875 | 6 |
| 8 | Volibear | 1 | 0.0 | 6 | 2 | 33 | 20641 | 15057 | 4345 | 1238 | 193465 | 7 |
| 9 | Hwei | 1 | 102.022396 | 3 | 2 | 25 | 28190 | 2227 | 25962 | 0 | 144254 | 1 |
| 10 | Varus | 1 | 0.0 | 0 | 1 | 21 | 14029 | 10241 | 3595 | 192 | 101394 | 8 |
| 11 | Lulu | 1 | 9382.038085 | 5 | 2 | 39 | 12679 | 2121 | 9724 | 833 | 28852 | |
| 12 | Ambessa | 0 | 0.0 | 4 | 2 | 16 | 24441 | 23211 | 1016 | 213 | 119418 | 11 |
| 13 | Viego | 0 | 0.0 | 9 | 1 | 18 | 22423 | 20093 | 1832 | 498 | 216463 | 14 |
| 14 | Ahri | 0 | 0.0 | 2 | 2 | 21 | 29859 | 2759 | 20495 | 6605 | 165520 | 1 |
| 15 | Smolder | 0 | 0.0 | 0 | 1 | 12 | 19153 | 16671 | 1965 | 516 | 121193 | 10 |
| 16 | Soraka | 0 | 22717.17382 | 0 | 1 | 40 | 7643 | 1148 | 4746 | 1749 | 23032 | |
| 17 | Aurora | 1 | 0.0 | 2 | 2 | 8 | 17143 | 1017 | 15289 | 836 | 123264 | 1 |
| 18 | RekSai | 1 | 0.0 | 3 | 1 | 19 | 13963 | 9566 | 3246 | 1150 | 224538 | 14 |
| 19 | Sylas | 1 | 18.637207 | 3 | 2 | 15 | 19495 | 392 | 19047 | 56 | 116374 | |
| 20 | Kalista | 1 | 0.0 | 7 | 2 | 17 | 32991 | 23888 | 4490 | 4612 | 185597 | 15 |

A l'aide des clusters choisis dans la partie précédente, nous définissons 10 nouveaux datasets (1 par groupe) auxquels nous allons appliquer une suite de transformations à l'aide du programme "etape2_creation_cluster.py"..

Pour chaque dataset :

— On ne sélectionne que les champions d'un même groupe.
— On crée 1 matrice X qui contient les statistiques, et un vecteur y qui contient la colonne win.
— On centre et réduit les données de X.
— On applique un lasso afin de déterminer les variables les plus importantes ainsi que le signe des coefficients qui nous serviront plus tard.
— On ne garde dans un nouveau dataset que les données des statistiques pour lesquelles les coefficients du modèle du lasso sont non nuls et le nom des champions.
— On regroupe les données pour n'avoir qu'une ligne par champion en appliquant la moyenne des statistiques.
— On ajoute 5 champions fictifs témoins de performances, de 1 Star pour le moins bon à 5 Star pour le meilleur. Le moins bon verra ses statistiques moins bonnes que la moyenne et le meilleur aura de meilleures statistiques que la moyenne.

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | Champ Name | totalHeal | magicDamage | goldSpent | | |
| 2 | AurelionSol | 1919.709677 | 8458.667232 | 10321.103565365025 | | |
| 3 | Brand | 3693.994366 | 8480.478873 | 9028.985915492958 | | |
| 4 | FiddleSticks | 15464.59139 | 9434.279569 | 8996.31421744325 | | |
| 5 | Heimerdinger | 1395.687919 | 6134.057046 | 9192.634228187919 | | |
| 6 | Karthus | 8464.843563 | 9045.781640 | 11085.475285171102 | | |
| 7 | Lillia | 23375.33449 | 11755.10300 | 10005.653935185184 | | |
| 8 | Malzahar | 1160.771971 | 7404.947743 | 10061.662707838479 | | |
| 9 | Mordekaiser | 8027.104683 | 10119.98071 | 9572.58953168044 | | |
| 10 | Morgana | 9328.595541 | 8331.425159 | 8459.140127388535 | | |
| 11 | Rumble | 2856.493582 | 7938.638273 | 9476.878646441073 | | |
| 12 | Singed | 3933.880269 | 8370.544688 | 9333.575042158516 | | |
| 13 | Swain | 8279.091310 | 11009.70741 | 9178.036327933236 | | |
| 14 | Teemo | 4730.304932 | 7793.431988 | 9944.40956651719 | | |
| 15 | Zyra | 5564.690221 | 7057.367594 | 9386.391862955032 | | |
| 16 | 1 Star | 3506.967640 | 13000.11545 | 4787.244677134212 | | |
| 17 | 2 Star | 5260.451460 | 10833.42954 | 7180.867015701318 | | |
| 18 | 3 Star | 7013.935280 | 8666.743639 | 9574.489354268424 | | |
| 19 | 4 Star | 8767.419100 | 6500.057729 | 11968.11169283553 | | |
| 20 | 5 Star | 10520.90292 | 4333.371819 | 14361.734031402637 | | |
| 21 | | | | | | |
| 22 | | | | | | |

Cela nous permettra de mettre en évidence la séparation entre les bons champions et les moins bons lors du tracé de la PCA.

Nous savons alors pour chaque groupe de champions quelles sont les statistiques les plus importantes, il ne nous reste plus qu'à afficher ces résultats.

## 2.2 PCA

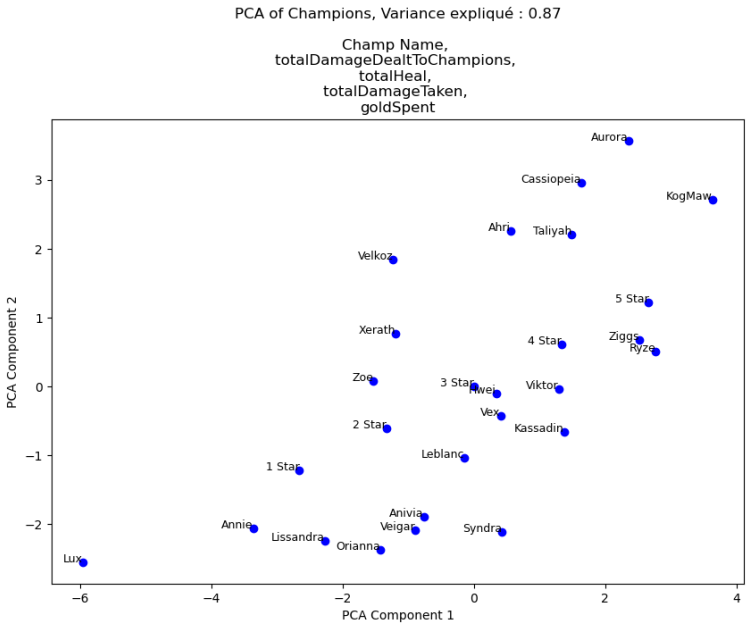
Suite à toutes ces transformations, nos 10 datasets sont prêts pour la PCA. Puisqu'en moyenne nous avons 5 statistiques choisies, le processus est assez fiable : 80% de variance expliquée en moyenne.

Pour pouvoir visualiser chaque clusters et ainsi connaître quels sont les meilleurs champions, nous avons bien utilisé une PCA. Nous avons fait ce choix plutôt que t-SNE car nous voulions une transformation linéaire, qui respecte l'emplacement des champions plutôt que leurs distances relative.

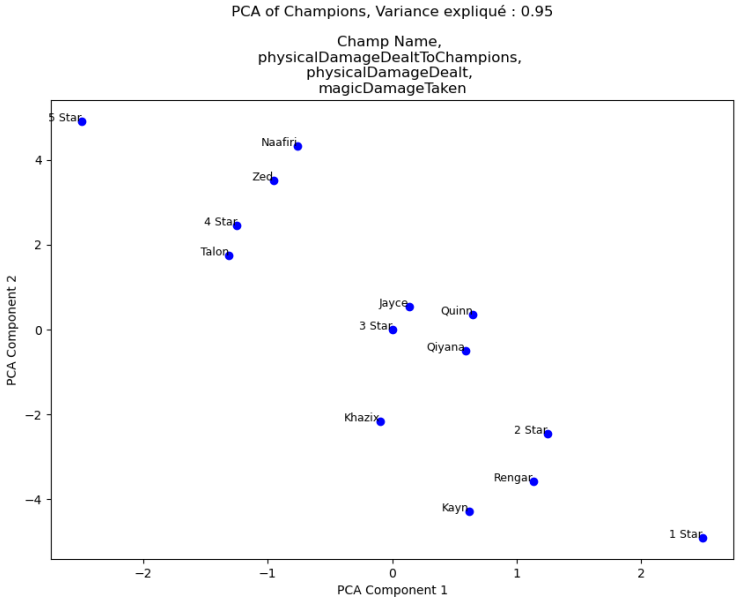
Voyons quelques exemples de résultats.

## 2.3 Affiche et Analyse

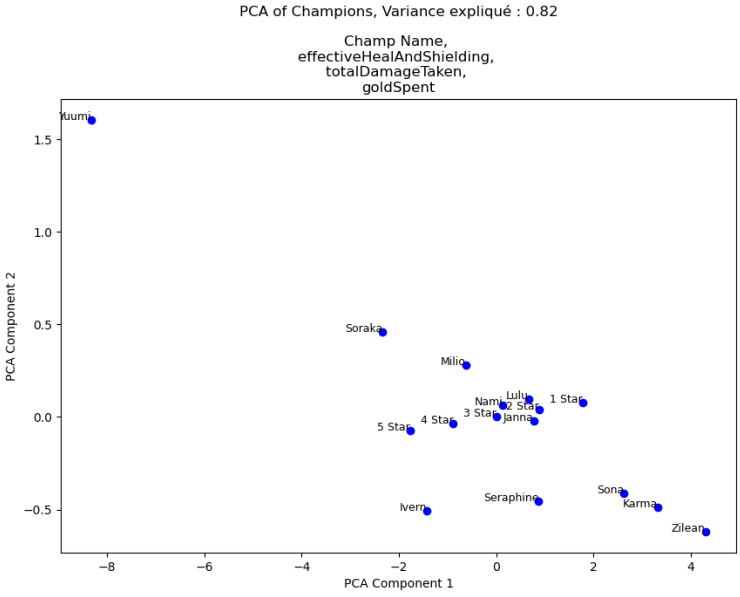
Les analyses suivantes sont faites à l'aide du programme "etape2_visualisation.py"




Dans cette PCA, nous observons que Lux est la championne la moins bonne de sa catégorie. Ce qui s'explique facilement, pour près d'une partie sur 4, Lux n'est pas joué comme un mage avec un build axé sur sa puissance mais plus comme un support (avec des objets comme shurelya, augmentant la vitesse de déplacement de ses compagnons). Ce build se concentre sur l'aide qu'elle peut apporter à ses alliés via son bouclier et sur ses contrôles de foule, bref, son côté utilitaire. C'est donc naturel qu'elle soit catégorisée comme la moins forte de son cluster.




Cette PCA ne présente aucune anomalie, Les champions 1 Star, 2 Star etc. permettent de situer quels champions sont les plus pertinents au sein de ce cluster. Nous pouvons donc affirmer que Zed, Naafiri, Talon sont les meilleurs assassins.




Les axes de cette PCA sont une combinaison linéaire de l'effectiveHealAndShielding, totalDamageTaken et goldSpent. Yummi ayant la particularité de se lier à un champion allié et de ne pas prendre de dégâts, elle se retrouve donc excentrée.

# 3 Perspectives d'amélioration et d'explorations futures

La première possibilité serait de refaire l'analyse ci-dessus mais avec seulement des parties d'ARAM, un autre mode de jeu. L'avantage de l'ARAM est que les champions sont choisis aléatoirement, et donc, tous les

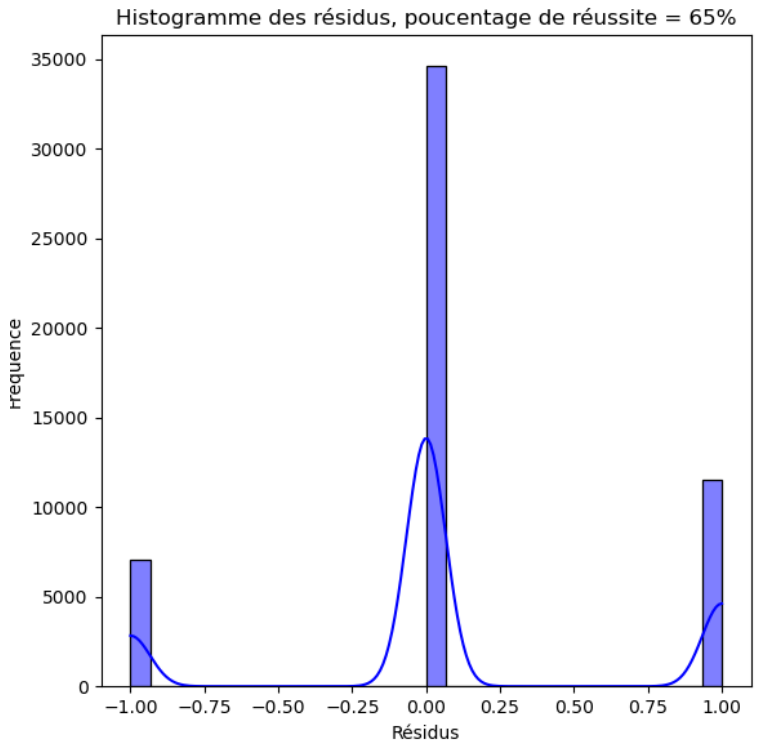
champions seront équitablement représentés.

Pour une étude complémentaire, nous pourrions ne pas regarder les champions seuls mais une combinaison de deux rôles jouant ensemble (adc+support) ou (mid+jungle). Nous pourrions ainsi refaire l'analyse ci-dessus, permettant de trouver le meilleur duo, ou, prendre toutes les parties contenant un champion donné et refaire l'analyse sur ce sous dataset. Ceci permettrait de savoir quel est le meilleur champion sachant que notre coéquipier a déjà choisi un champion.

Nous remarquons que le meilleur clustering est celui fait avec les items et non celui fait avec leur statistique nous en déduisons donc que l'information donnée par les effets passifs des items est cruciale. Nous pourrions quantifier cet effet en comparant le pourcentage de pick d'un item avec sa rentabilité (basé sur son coût et les statistiques qu'il procure).

En ce qui concerne les pistes d'amélioration, lors de la détermination des statistiques à garder, nous avions utilisé une méthode lasso. Or, il serait préférable de ne pas se limiter à des régressions linéaires dû à la nature non linéaire de nos données. En effet, pour se convaincre, nous avons récupéré les valeurs des $R^2$ adjusted, et nous avons observé que ces dernières varient entre 0.01 et 0.1.

Voici la distribution des résidus :




Le pourcentage de précision est alors de 65%, c'est 15% meilleur qu'un système aléatoire, ce qui n'est pas vraiment bon.

Un modèle non linéaire nous aurait permis ici de capter des relations plus complexes au sein du dataset et ainsi de déterminer avec plus de précision et de certitude les statistiques déterminantes à chaque groupe.