

# SY09 Analyse de données de Pokemon Challenge

Luqin Ren, Elisa Finzy, Corentin Duhamel

14 juin 2020

## Résumé

Dans le cadre de l'UV SY09, nous réalisons un projet en trinôme dans le but d'appliquer les méthodes étudiées au cours du semestre sur des jeux de données réels.

## 1 Analyse exploratoire

Le but de cette analyse est de prendre connaissance du jeu de données, de ses composantes et de ses particularités. Cette étape permettra également d'analyser les distributions de certaines variables et de découvrir des variables plus intéressantes.

### 1.1 Introduction à l'ensemble de données

Notre ensemble de données contient trois fichiers : *pokemon.csv*, *combats.csv*, *tests.csv*. Dans *pokemon.csv*, il y a 800 lignes, et chaque ligne représente un pokémon ; 12 colonnes, chaque colonne représente un attribut de pokémon. 6 variables quantitatives : Hitpoints, Attack, Defense, Special Attack, Special Defense, Speed ; 4 qualitatives : Type 1, Type 2, Generation et Legendary. Dans *combats.csv*, il y a 50 000 lignes, chacune avec trois variables, représentant les résultats des combats de pokémon. Le gagnant est stocké dans la troisième colonne. Dans *test.csv*, il y a 10 000 lignes \* 2 colonnes, notre objectif est de prédire quel pokémon va gagner pour chaque combat dans ces 10 000 lignes.

First_pokemon	Second_pokemon	Winner
266	298	298
702	701	701
191	668	668
237	683	683
151	231	151
657	752	657
192	134	134

FIGURE 1 – combats.csv – dimension (50000,3).

### 1.2 Données manquantes

#	Name	Type 1	Type 2	HP	Attack	Defense	Sp Atk	Sp.Def	Speed	Generation	Legendary
59	Persian	Normal	nan	65	70	60	65	65	115	1	False
60	Psyduck	Water	nan	50	52	48	65	50	55	1	False
61	Golduck	Water	nan	80	82	78	95	80	85	1	False
62	Mankey	Fighting	nan	40	80	35	35	45	70	1	False
63	nan	Fighting	nan	65	105	60	60	70	95	1	False
64	Growlithe	Fire	nan	55	70	45	70	50	60	1	False
65	Arcanine	Fire	nan	90	110	80	100	80	95	1	False
66	Poliwag	Water	nan	40	50	40	40	40	90	1	False
67	Poliwhirl	Water	nan	65	65	65	50	50	90	1	False
68	Poliwrath	Water	Fighting	90	95	95	70	90	70	1	False

FIGURE 2 – pokemon.csv – dimension (800,12).

Dans *pokemon.csv*, il existe un pokémon sans nom. Nous l'avons trouvé grâce à la liste officielle de Pokémons<sup>1</sup> : *Primeape*. Il existe 386 pokémons sans *Type 2*, mais c'est un phénomène normal. Certains pokémons ont deux types et certains n'en ont qu'un. Par conséquent, nous avons directement remplacé *nan* par *No Type 2*. Il n'y a pas d'autres données manquantes dans les trois fichiers. En conclusion, nos données sont relativement propres.

### 1.3 Ingénierie des fonctionnalités

Il s'agit du processus d'utilisation de données et de leur combinaison pour créer une nouvelle variable (ou caractéristiques) à utiliser dans l'analyse. Ici, nous utilisons la méthode statistique pour calculer le taux de gain de chaque pokémon dans *combats.csv*, et l'ajoutons comme nouvelle variable *win percentage* à *pokemon.csv*. Cette étape nous aide à comparer la différence entre les attributs du gagnant et du perdant, afin que nous puissions mieux découvrir les variables qui sont les plus importantes pour gagner un combat.

### 1.4 Variables qualitatives

La figure 3 montre la distribution des types des pokémons. Dans *Type1*, *Water*, *Normal*, *Grass* sont les plus

1. [https://bulbapedia.bulbagarden.net/wiki/List\\_of\\_Pok%C3%A9mon\\_by\\_National\\_Pok%C3%A9dex\\_number](https://bulbapedia.bulbagarden.net/wiki/List_of_Pok%C3%A9mon_by_National_Pok%C3%A9dex_number)

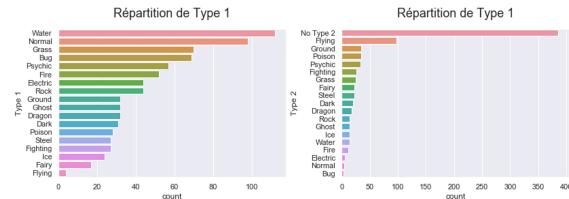


FIGURE 3 – Répartition des Type1 et Type2 des pokemons.

courantes et *Flying* est le plus rare. Dans Type2, *Flying* est le plus courant et *Bug* est le plus rare.

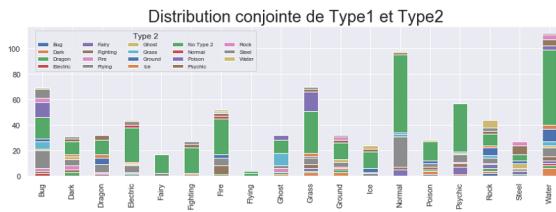


FIGURE 4 – Répartition conjointe de Type1 et Type2.

Selon la distribution conjointe de Type1 et de Type2, nous avons constaté que pour des pokemons dont le premier type est *Flying*, soit il n'y a pas de deuxième type, soit leur deuxième type est *Dragon*. Il est supposé qu'il devrait y avoir une connexion logique entre le premier type et le deuxième type.

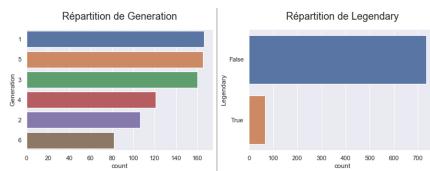


FIGURE 5 – Répartition de Generation et de Legendary.

Nous pouvons voir sur la figure 5 que la sixième génération de pokémon est la plus petite et que les pokémons légendaires ne représentent que 6% de tous les pokemons. De cela, nous devinons si les pokemons les plus rares seront plus forts.

## 1.5 Variables quantitatives

La distribution de chaque attribut de pokemon suit presque la loi Normal. Lorsque nous classons pokemon selon *Legendary*, nous constatons que pokemon legen-

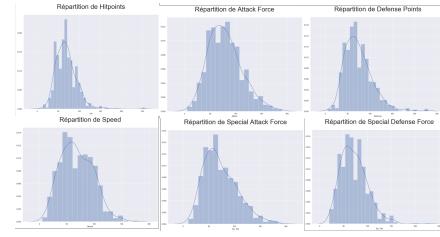


FIGURE 6 – Répartition des variables quantitatives de pokémon.



FIGURE 7 – Répartition de variables quantitatives de pokémon selon Legendary.

daire (en rouge) est plus fort que les pokémons ordinaires (en bleu) dans chaque indicateur.

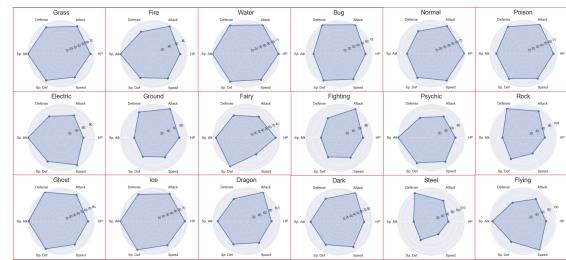


FIGURE 8 – Diagrammes radar des valeurs d'attribut pour différents types de pokémon.

Pokemon *Grass* et pokemon *Water* sont très moyens dans toutes les capacités, tandis que pokemon *Dragon* est très fort pour attaquer, pokemon *Fairy* et pokemon *Steel* sont très forts pour la défense, et pokemon *Electric* et *Flying* sont très rapides. Il semble que la capacité des pokemons soit très liée à leur genre.

En comparant 6 générations différentes, nous avons constaté que le pokemon de quatrième génération a des capacités d'attribut plus fortes que les autres générations.

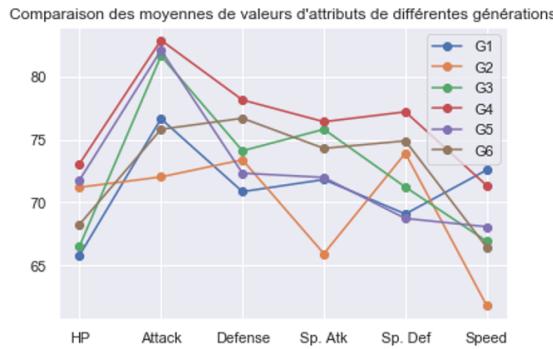


FIGURE 9 – Répartition de variables quantitatives de pokémon selon Generation.

## 1.6 Corrélation entre les variables

La matrice de covariance nous aide à comprendre la corrélation linéaire entre les variables : plus le nombre est proche de 1, plus les deux variables sont corrélées.

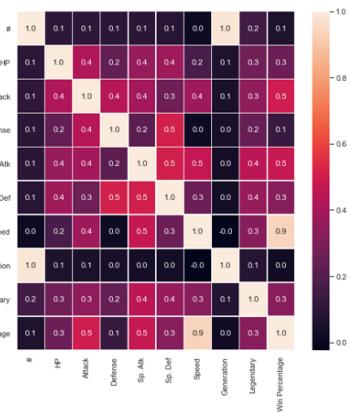


FIGURE 10 – La matrice de covariance.

Les coefficients de corrélation des couples de variables suivants sont tous égaux à 0,5, ce qui est une corrélation suffisante : *Sp Defense & Defense*, *Sp Defense & Sp Attack*, *Speed & Sp Attack*. Il convient de noter que le coefficient de corrélation entre la *Speed* et *Win percentage* est aussi élevé (0,9), ce qui indique que la vitesse joue un rôle important dans les combats de pokémon. Dans le même temps, le *Win percentage* est également très corrélé avec *Attack* et *Sp Attack*, reflétant du côté que la puissance d'attaque est plus importante que la puissance de défense.

Nous utilisons des *Scatterplot* pour obtenir un affichage plus intuitif de la corrélation entre les variables.

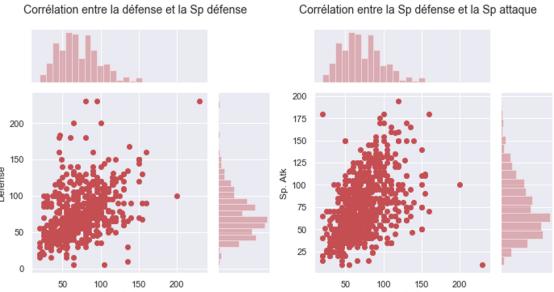


FIGURE 11 – Corrélations entre Sp Defense, Defense et Sp Attack.

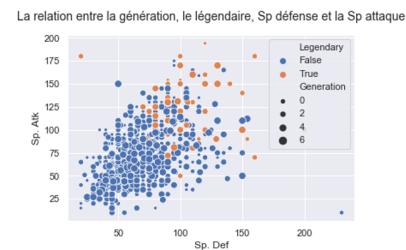


FIGURE 12 – Relation entre Legendary, Generation, Sp Defense et Sp Attack.

## 1.7 Variables importantes pour pokémon win

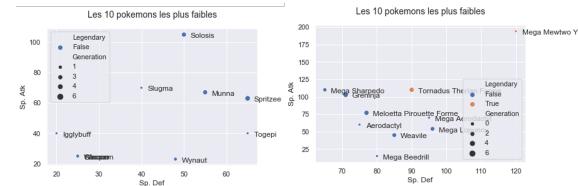


FIGURE 13 – Les individus intéressants : top 10 et end 10.

Tout d'abord, nous avons trouvé les dix plus forts et les dix plus faibles pokemons par *Win percentage* en analysant leurs caractéristiques, pour découvrir les facteurs importants pour gagner. En plus de l'attaque et de la défense plus fortes du gagnant par rapport au perdant, nous avons également constaté que le nom du gagnant contient le mot-clé "Mega", c'est-à-dire que le pokémon après Mega evolution est généralement plus fort.

Ensuite, nous comparons la différence d'attributs entre le gagnant et le perdant. La définition du gagnant est que le *Win percentage* est supérieur à 0.5, ce qui est

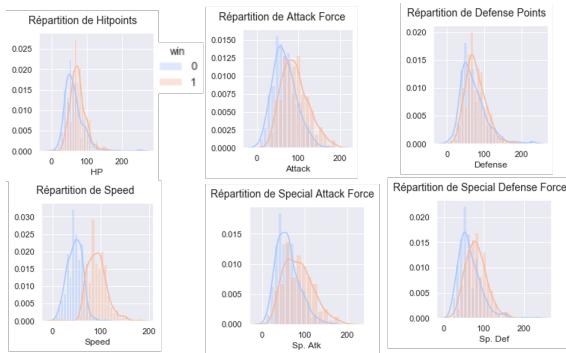


FIGURE 14 – Comparaison des gagnants et des perdants.

représenté par le rouge, et le perdant est représenté par le bleu. De toute évidence, chaque capacité du gagnant est plus forte que la plus faible, en particulier la vitesse et l'attaque.

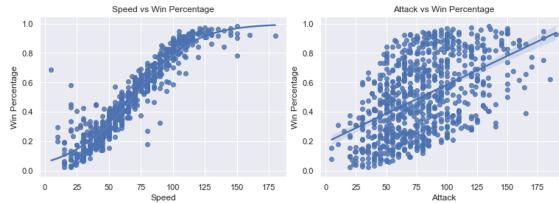


FIGURE 15 – Variables importantes pour pokemon win.

En analysant la relation entre la *Speed*, *Attack* et *Win percentage*, nous constatons que la vitesse joue un rôle très important dans la victoire. Ce résultat est cohérent avec le résultat de la matrice de covariance.

## 1.8 Analyse factorielle des données mixtes (FAMD)

Le but de FAMD est de faire une PCA pour les variables quantitatives et une MCA pour les variables qualitatives afin de transformer les variables d'origine et de les mapper dans un espace de faible dimension, en espérant ainsi réduire la redondance et le bruit. Les variables quantitatives et qualitatives sont normalisées au cours de l'analyse afin d'équilibrer l'influence de chaque ensemble de variables.

### 1.8.1 PCA

L'ensemble de données d'origine comporte 6 variables quantitatives, et nous fixons le nombre de composantes principales à 6, et observons le rapport d'inertie que

l'ensemble de données peut expliquer sur la nouvelle base. À travers la figure 16, on voit que les quatre premières composantes principales peuvent expliquer 88% d'inertie. Lorsque nous représentons l'ensemble de données sur les deux premiers axes principaux, nous constatons que le gagnant (orange) et le perdant (bleu) peuvent être distingués dans une certaine mesure.

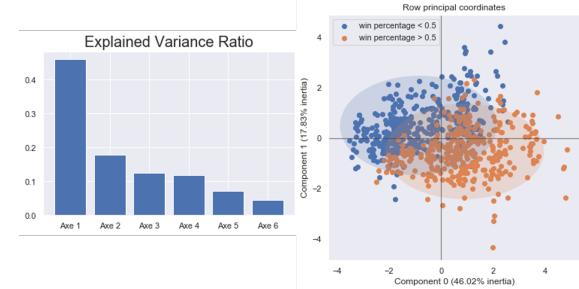


FIGURE 16 – PCA pour les 6 variables quantitatives.

### 1.8.2 MCA

L'analyse de correspondance multiple (MCA) est une extension de l'analyse de correspondance (CA). L'idée est simplement de calculer la version avec l'encodage one-hot d'un ensemble de données et d'y appliquer CA. Le nombre de variables est passé de 4 à 45.

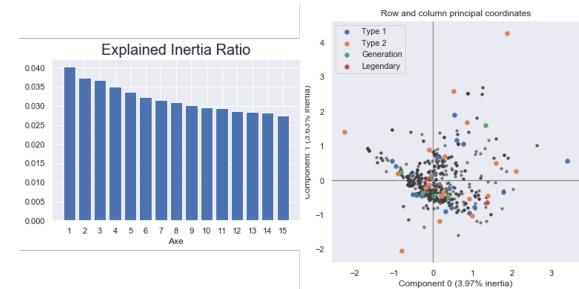


FIGURE 17 – MCA pour les 4 variables qualitatives.

Cependant, selon le rapport d'inertie expliqué par chaque axe, MCA ne peut pas réduire efficacement la dimension de l'ensemble de données. Surtout les première et deuxième composantes, elles ne peuvent expliquer que 14% de l'inertie.

### 1.8.3 FAMD

L'algorithme FAMD peut être vu comme un mélange entre PCA et MCA. En d'autres termes, il agit comme des variables quantitatives PCA et comme MCA pour

les variables qualitatives. Les deux premières composantes principales expliquent 42% d'inertie et n'ont pas d'effet évident sur la distinction entre gagnants et perdants.

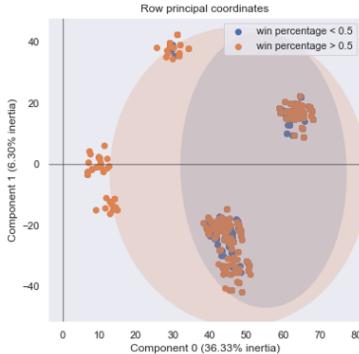


FIGURE 18 – FAMD pour les variables qualitatives et quantitatives.

En conclusion, notre ensemble de données d'origine ne comporte que 10 variables, et l'effet de réduction de la dimensionnalité en utilisant PCA et MCA n'est pas idéale. Nous préférons toujours conserver l'espace variable d'origine.

## 1.9 Valeurs aberrantes ?

Une très belle propriété des SVM est qu'ils peuvent créer une frontière de décision non linéaire en projetant les données via une fonction non linéaire dans un espace de dimension supérieure. Cela signifie que les points de données qui ne peuvent pas être séparés par une ligne droite dans leur espace d'origine I sont "levés" vers un espace caractéristique F où il peut y avoir un hyperplan "droit" qui sépare les points de données d'une classe d'une autre. Sur la base de ce principe, One-class SVM peut diviser les données en deux types, qui sont couramment utilisés pour la détection des valeurs aberrantes. Nous utilisons le noyau RBF, qui est un noyau adapté à la plupart des situations, mais également très adapté aux ensembles de données séparables non linéaires.

Après ajustement avec SVM, chaque échantillon a obtenu son score correspondant. Nous avons fixé le seuil à 0.005, ce qui signifie que les données en dehors du quantile de 0.005 seront considérées comme aberrantes. Nous avons trouvé quatre valeurs aberrantes (rouge). Après avoir examiné les données de ces quatre pokemons, nous avons constaté que leurs valeurs se situaient dans une fourchette raisonnable et n'étaient pas vraiment aberrantes. Par conséquent, nous avons décidé de ne pas les traiter. Notre ensemble de données d'origine est assez

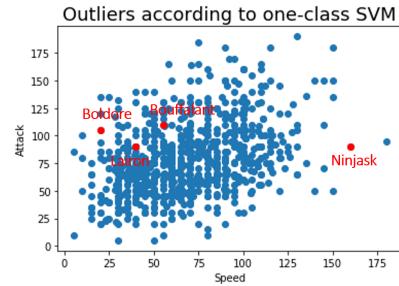


FIGURE 19 – Valeurs aberrantes selon one-class SVM.

propre.

## 1.10 Conclusion de l'analyse exploratoire

C'est là que nous communiquons toutes les idées que nous avons développées de manière concise.

1. *Water, Normal, Grass* sont le type 1 le plus courant et *Flying, Ground* et *Poisson* sont le type 2 le plus courant.
2. Les types de Pokémons qui gagnent le plus sont *Flying, Dragon, Electric* et *Dark*. Les types de Pokémons qui gagnent le moins sont *Fairy, Rock, Steel, Poison*.
3. *Attack* et *Speed* sont importantes. Si nous regardons en arrière dans le top 10 des pokemons les plus gagnants, tous ont des vitesses supérieures à 100 et des attaques supérieures à 100 (à l'exception de l'attaque de Greninja).

## 2 Classification non-supervisée

Cette section aura pour but d'effectuer une classification non-supervisée sur l'ensemble des données disponibles, et donc d'obtenir une représentation simplifiée des données initiales. La non-supervision permettra une construction automatique des classifications.

L'idée sera donc de chercher à regrouper en classes les individus dont les caractéristiques seront considérées comme "proches" ou "ressemblantes".

Ainsi, il sera possible mettre en lumière la manière dont on peut regrouper les données et les caractéristiques qui ont le plus d'influence sur la classification effectuée. Cela permettra donc de comprendre l'influence des paramètres sur la répartition en classes des données.

Pour cela, nous allons réaliser des classifications sur deux types de données. Tout d'abord, nous commence-

rons par chercher à classifier les individus du fichier `pokemon.csv` (correspondant à l'ensemble des pokémons), dans le but de distinguer les caractéristiques ayant un impact fort sur leur différenciation et de les caractériser. Ensuite, nous effectuerons une classification sur les combats, en joignant le fichier `combats.csv` (correspondant aux combats des pokémons avec le gagnant associé) au fichier `pokemon.csv` afin de pouvoir discerner des combats "type", de déterminer de quelles caractéristiques dépendent le plus leur classification et, in fine, de pouvoir se faire une idée de quelle variable ou paramètre aura le plus d'importance et d'impact dans la prédiction des gagnants des combats futurs.

## 2.1 Classification sur les pokémons

Les analyses seront effectuées sur les variables quantitatives, que l'on va donc isoler du reste des variables.

Nous allons utiliser la méthode de classification par l'algorithme des k-means (ou méthode des centres mobiles) qui, à partir de centres de classes générés aléatoirement, va répartir les individus dans chaque classe en fonction de la distance qui les sépare des centres et, par la suite, converger vers des centres de classes qui permettront d'obtenir des classes définitives distinctes.

Pour pouvoir distinguer deux catégories de pokémons, à savoir ceux qui auront plus de probabilité de gagner et ceux qui auront tendance à perdre, on choisit de faire un clustering en deux classes.

Afin de visualiser la classification, on choisit de réaliser une ACP et de visualiser les deux axes ayant le plus d'inertie et qui portent donc le plus d'information. Nous obtenons le graphique suivant :

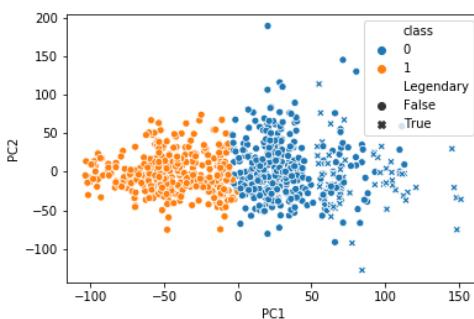


FIGURE 20 – Classification des pokémons par l'algorithme des k-means et visualisation par PCA

Nous pouvons observer deux classes bien délimitées (classe 0 et 1). Parmi celles-ci, on peut remarquer que

les pokémons paraissant avoir les valeurs de variable les plus élevées semblent se trouver dans la classe 0, tandis que les pokémons ayant des caractéristiques de valeur plus faibles, dans la classe 1. On a également affiché un indicateur permettant de localiser les pokémons dont la variable `Legendary` avait pour valeur "true". Ainsi, tous les pokémons légendaires semblent tous se situer dans la classe 0, ce qui paraît logique puisque les pokémons légendaires ont généralement des statistiques plus élevées que les autres pokémons, ce qui indique que la variable `Legendary` a une forte importance dans la manière de distinguer les pokémons ayant plus de chance de gagner un combat, des pokémons ayant moins de probabilité d'être vainqueur.

À partir de ceci, on crée un dataframde de pokémons étiquetés en fonction de leur classe d'appartenance, et on sépare ce tableau en deux tableaux respectivement pour les pokémons de classe 0 et 1. Il est ainsi possible de tracer les distributions de chacune des variables pour les classes 1 et 0, tel que montré ci-dessous.

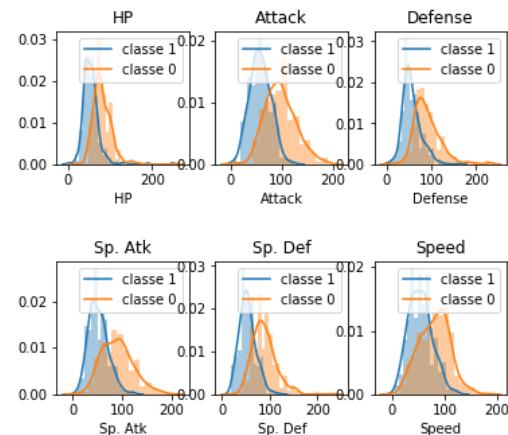


FIGURE 21 – Distributions des variables quantitatives suivant les deux classes par les k-means

On peut remarquer que la classe 0 est constituée majoritairement de pokémons dont les valeurs correspondant aux variables quantitatives sont significativement plus élevées que pour les pokémons de la classe 1. On peut également remarquer que les variables qui semblent le plus engendrer de différence entre les individus de chaque classe sont les variables `Attack`, `Sp. Atk` et `Speed`. En effet, les distributions des classes 1 et 0 pour ces variables sont plus éloignées entre elles que pour les autres variables.

On cherche ensuite à savoir quelle influence peuvent avoir les caractéristiques d'un pokémon sur les combats qu'ils effectuent. Autrement dit, on cherche à savoir si

un pokémon appartenant à la classe "plus forte" (la classe 0) est plus souvent gagnant qu'un pokémon appartenant à la classe 1 et si, finalement, les caractéristiques quantitatives d'un pokémon peuvent avoir une influence forte sur l'issue d'un combat, quelque soit le pokémon adversaire.

En réalisant une jointure entre le dataframe des combats et le dataframe des pokémons étiquetés selon leur classe d'appartenance, on arrive à obtenir le pourcentage de pokémons ayant été gagnants dans un combat dans chacune des deux classes, donnant le diagramme circulaire suivant :

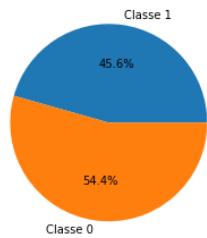


FIGURE 22 – Pourcentage de pokémons gagnants suivant les classes 0 et 1

Tel que prédit par les histogrammes, il y a plus de pokémons gagnants dans les combats dans la classe 0 que dans la classe 1. En revanche, l'écart de pourcentage n'est pas excessivement élevé, ce qui laisse penser que les caractéristiques d'un pokémon pris individuellement ne permet pas dans la totalité des cas de prévoir efficacement l'issue d'un combat le concernant, laissant ainsi supposer le rôle prépondérant que son adversaire peut jouer.

## 2.2 Classification sur les combats

L'objectif ici est de réaliser une classification des combats, afin de les représenter plus facilement et de déterminer quels groupes de variables ont le plus un rôle à jouer dans la caractérisation d'un combat, et donc dans l'issue de ce dernier.

Les analyses seront effectuées sur les variables quantitatives, que l'on va donc isoler du reste des variables. On réalise ensuite une jointure pour créer un dataframe renseignant les combats ainsi que les caractéristiques des deux pokémons impliqués dans chaque combat. On aura, dans l'ordre, les caractéristiques du pokémon "first" (le premier pokémon dans l'ordre des colonnes "First\_pokemon"), et celles du pokémon "second" ("Second\_pokemon").

On utilise l'algorithme des k-means pour réaliser une classification en deux classes, l'objectif étant de distinguer deux types de combats : les combats pour lesquels le premier pokémon (First\_pokemon) est avantage, ayant des caractéristiques plus fortes, et les combats pour lesquels le second pokémon (Second\_pokemon) l'est davantage.

Voici le graphique obtenu suite à la visualisation des individus étiquetés suivant les deux axes portant l'inertie la plus élevée suivant une ACP :

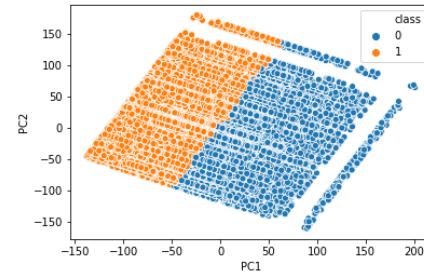


FIGURE 23 – Classification des combats par l'algorithme des k-means et visualisation par PCA

On observe deux classes très nettement délimitées.

On crée un dataframe de combats étiquetés en fonction de leur classe d'appartenance, et on sépare ce tableau en deux tableaux respectivement pour les combats de classe 0 et 1. Il est ainsi possible de tracer les distributions de chacune des variables pour les classes 1 et 0, tel que montré ci-dessous, de manière à comparer selon les classes les pokémons "First" et "Second".

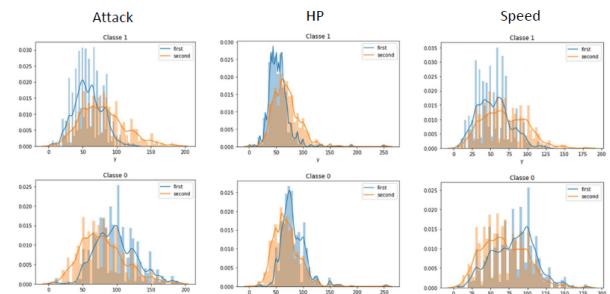


FIGURE 24 – Distributions des variables quantitatives suivant les deux classes par les k-means (Attack, HP, Speed)

On possède donc, pour chacune des 6 variables quantitatives, une distribution correspondant au pokémon First (en bleu) et au pokémon Second (en orange), pour

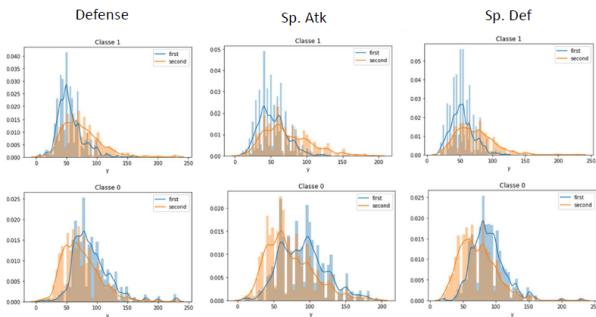


FIGURE 25 – Distributions des variables quantitatives suivant les deux classes par les k-means (Defense, Sp. Atk, Sp. Def)

les classes 1 (première ligne de graphiques pour chacune des deux figures) et 0 (deuxième ligne de graphiques pour chacune des deux figures).

On peut donc remarquer que de manière générale (voir Figure 24 et 25), dans la classe 0, le pokémon First semble avoir de meilleures caractéristiques que le pokémon Second, et on peut observer l'inverse pour la classe 1, mais de manière légèrement moins marquée. On obtient bien une différenciation entre les pokémon First et Second dans les deux classes, chaque classe contenant une prépondérance au niveau des caractéristiques d'un des deux types de pokémons, ce qui permet d'obtenir deux types de combats : les combats dans lesquels le pokémon First est avantage, et ceux pour lesquels le pokémon Second l'est davantage.

Les caractéristiques pour lesquelles il semble y avoir le plus de contraste entre les classes de combats 0 et 1 sont Attack, Speed, Sp. Atk et Sp. Def : on peut remarquer que les courbes de distributions correspondant au pokémon First et Second, dans les deux classes, sont plus éloignées entre elles par rapport aux autres variables, et pourraient donc avoir une importance plus grande pour la prédiction des combats.

On cherche donc ensuite à étudier la proportion de combats gagnés par le pokémon First dans la classe 0, où il est sensé être plus avantage par rapport au pokémon Second, et de même, la proportion de combats gagnés par le pokémon Second dans la classe 1, où il est sensé plus avantage par rapport à la classe 0. On obtient le diagramme circulaire suivant.

Tel que prédit par les histogrammes (Figure 26), on remarque que le pokémon First est plus souvent gagnant pour les combats de la classe 0 (ses caractéristiques sont meilleures) que dans la classe 1, et que le pokémon Second est plus souvent gagnant pour les combats de la

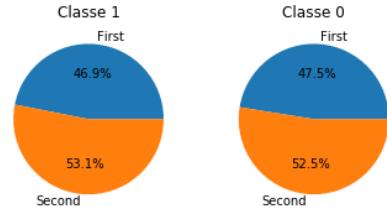


FIGURE 26 – Pourcentage de combats gagnés par les pokémons First et Second dans les deux classes 0 et 1

classe 1 que 0.

En revanche, le contraste entre les classes 1 et 0 n'est pas très élevé, on peine à obtenir deux classes dans lesquelles un pokémon a clairement l'avantage par rapport à l'autre. Il est probable que les variables quantitatives n'aient pas un rôle extrêmement décisif à jouer dans l'influence sur l'issue d'un combat, et que les variables qualitatives pourraient avoir une importance beaucoup plus grande dans la prédiction des gagnants des futurs combats.

### 3 Prédictions sur les combats

Dans cette partie, l'objectif va être d'étudier les combats à notre disposition et de construire des modèles afin de prédire l'issue de nouveaux combats. Pour cela, nous allons utiliser les fichiers `pokemon.csv` et `combats.csv`. Le premier renseigne les stats de chaque pokémon (Type, HP, défense, attaque, légendaire ...). Chaque pokémon possède un id unique.

Le fichier `combats.csv` regroupe l'issue de 50 000 combats. Chaque combat se caractérise par 2 pokemons qui s'affrontent et une colonne `winner` qui indique lequel des deux a remporté le combat. Les pokemons sont identifiés à l'aide de leur id évoqué précédemment.

Un fichier `test.csv` rassemble des données de combats. Toutefois, ce fichier n'indique pas le vainqueur des combats. Cet élément est indispensable pour vérifier l'efficacité de notre modèle afin de comparer le vainqueur prédit par le modèle et le réel vainqueur du combat. Pour cette raison, nous n'avons pas utilisé ce fichier.

### 3.1 Préparations des données

Pour commencer, nous devons rassembler les données issus des différents fichiers. On commence par charger les données des 2 fichiers. Pour chaque combat du fichier combat.csv, on ajoute les statistiques des pokemons 1 et 2. Pour cela, on utilise la fonction `np.append()` appliquée sur la ligne du fichier pokemon correspondant à l'id du pokemon souhaité.

Pour identifier le vainqueur, on utilise un booléen : 1 désigne un combat remporté par le pokemon 1, 0 une défaite du pokemon 1. On obtient alors un tableau dont chaque ligne représente un combat. Les 12 premières colonnes correspondent aux statistiques du premier pokemon. Les 12 colonnes suivantes correspondent au pokemon 2. La dernière colonne désigne le vainqueur.

On peut supprimer les noms des pokemons qui ne nous intéressent pas. Ensuite, nous allons convertir les variables catégorielles en valeurs numériques. Concernant les types, ils peuvent prendre 18 valeurs différentes (Flying, Ground, Poison, ...). Nous utilisons l'encodage one-hot afin de représenter le type des pokemons. Cette méthode consiste à créer une colonne pour chaque type (Flying, Ground, Poison) et associer un 1 à cette colonne si le pokemon est de ce type.

On aurait pu encoder les types par une variable de 1 à 18. L'inconvénient de cette méthode aurait été le calcul de distance puisque les types 1 et 18 auraient été considérés très éloignés, ce qui n'a pas de sens dans notre cas. En effet, on considère les types distincts et aucun rapprochement entre des types différents.

On sépare notre jeu de données avec d'un côté les entrées et de l'autre la sortie. À ce stade, on dispose d'une variable `x` de dimension (50000,23) et une variable `y` de dimension (50000,1).

À l'aide de `sklearn` et de sa fonction `train_test_split()`, on partage les données en train/test. On a alors 4 variables `X_train`, `X_test`, `y_train` et `y_test`.

### 3.2 Random Forest

En appliquant l'algorithme Random Forest sur ces données, on obtient le taux de bonne prédiction avec la formule suivante `sum(pred == y_test) / len(y_test)`

On obtient une accuracy de 94.8% par validation croisée, ce qui est très satisfaisant. Pour améliorer cette précision, on peut répéter l'algorithme en modifiant la valeur de l'hyperparamètre `n_estimators` de Random

Forest.

### 3.3 Régression Logistique

Pour voir l'effet d'une variable sur le résultat `y`, on peut utiliser la fonction suivante :

```
data.groupby('y').mean()
```

Cela nous permet de voir la moyenne de chaque variable lorsque `y=0` et `y=1`. Si les 2 valeurs sont proches, le paramètre a peu d'influence sur la variable à prédire.

Lorsqu'on applique un algorithme de régression logistique, la fonction `summary()` sur le modèle obtenu nous indique que certains paramètres n'ont pas ou peu d'influence sur `y`. Les p-values associées à ces paramètres sont grandes. Pour notre analyse, on décide alors de conserver uniquement les paramètres ayant des p-values inférieures à 0.05.

On entraîne le modèle sur les données d'entraînement et on obtient un taux de bonne prédiction de 90% sur le jeu de test.

### 3.4 Réseau de neurones

Pour terminer, nous souhaitions découvrir la bibliothèque Keras qui permet de concevoir des réseaux de neurones profonds (deep learning) en python.

Après avoir réalisé les imports nécessaires, on doit créer un modèle. Ce dernier s'occupera de l'organisation des couches de notre réseau de neurones. Ici, on utilise un modèle Sequential car c'est le type le plus simple. Les couches sont alors empilées les unes à la suite des autres de façon linéaire. Pour ajouter une couche au réseau, on utilise la fonction `.add()`

Les réseaux de neurones sont constitués d'une couche d'entrée qui correspond à la donnée envoyée à l'algorithme, une couche de sortie qui correspond à la prédiction renvoyée par l'algorithme et un certains nombres de couches cachées intermédiaires.

Les fonctions d'activation sont des fonctions mathématiques qui vont déterminer l'état des sorties d'une couche. Elle détermine pour chaque neurone s'il doit être activé ou non. Cette fonction d'activation peut être binaire, c'est à dire qu'on décide d'activer un neurone si la valeur obtenue par le modèle dépasse un certain seuil. Elle peut être linéaire. Dans ce cas, la fonction d'activation est de la forme  $y = ax$ . Par ailleurs, elle peut être non linéaire. Plusieurs fonctions sont régulièrement utilisées car cette famille de fonction partage des caractéristiques intéressantes comme la rétropropagation du gradient qui n'est pas possible pour les fonctions li-

néaires. Pour notre réseau de neurones, nous allons utiliser les fonctions Sigmoid et ReLU. La première est une couche en S défini par :

$$f(x) = \frac{1}{1 + e^{-\lambda x}} \quad (1)$$

Comme on peut le voir sur la figure 27, cette fonc-

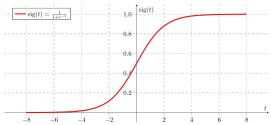


FIGURE 27 – Fonction Sigmoid. Lorsque lambda vaut 1, cela correspond à la fonction logistique.

tion renvoie des valeurs condensées entre 0 et 1, c'est pourquoi elle nous sera utile pour la couche de sortie.

La seconde fonction d'activation que l'on utilise est ReLU. Elle est définie par :

$$f(x) = \max(0, x) \quad (2)$$

Si la valeur entrée est négative, elle retourne 0 sinon elle retourne x.

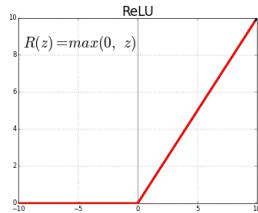


FIGURE 28 – Fonction d'activation ReLU

Nous avons réalisé un réseau de neurones à 3 couches cachées.

La fonction d'activation de la dernière couche est sigmoid afin d'obtenir une valeur entre 0 et 1 qui indique la fiabilité de la prédiction. Une valeur proche de 0.9 indiquera une prédiction de 1 avec quasi certitude alors qu'une valeur de 0.6 indiquera aussi une prédiction de 1 mais avec une certitude moindre. Si on veut uniquement prédire la classe sans indication sur la confiance du modèle, on peut attribuer 0 si la valeur est inférieure à 0.5 et 1 si la valeur est supérieure à 0.5.

Ce réseau de neurones nous permet d'obtenir une accuracy de 95.16% sur le jeu de test. On peut afficher la matrice de confusion avec le code suivant :

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

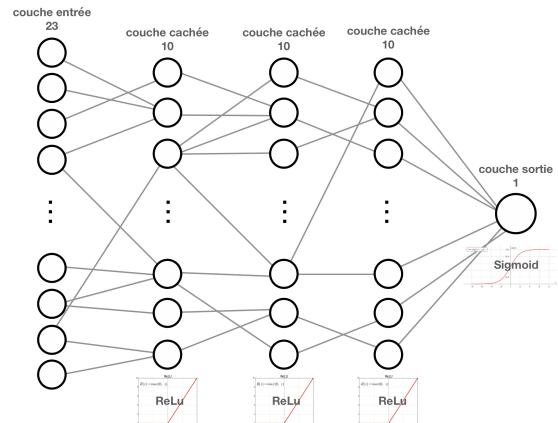


FIGURE 29 – Schéma de notre réseau de neurones

```
print(cm)
```

On obtient les résultats suivants :

```
[[7605 352]
 [374 6669]]
```

On constate alors une très légère augmentation des performances aux dépens d'une perte d'interprétabilité : on ne peut pas comprendre et justifier les résultats obtenus par le réseau de neurones.

## 4 Conclusion

Dans ce projet, nous avons d'abord effectué une analyse exploratoire sur l'ensemble de données, acquis une compréhension globale des caractéristiques des données et effectué des travaux de prétraitement tels que le remplissage des valeurs manquantes, la recherche de valeurs aberrantes et l'ingénierie des fonctionnalités. Ensuite, nous avons essayé d'utiliser des méthodes d'apprentissage non supervisées telles que KMeans pour trouver des associations de regroupement entre les données. Enfin, nous avons essayé des méthodes d'apprentissage telles que la forêt aléatoire, la régression logistique et les réseaux de neurones pour prédire les résultats du combat. Le résultat de prédiction du réseaux de neurones est le plus précis, mais l'entraînement du réseaux de neurones nécessite beaucoup de données et de temps, et il n'est pas aussi interprétable que le modèle basé sur l'arbre de décision tel que la forêt aléatoire. Un autre avantage de la forêt aléatoire est qu'elle est simple à appliquer et ne nécessite pas d'ingénierie d'entités complexes ni de transformation d'entités sur l'ensemble de données. De plus, afin d'améliorer les performances du modèle, nous pouvons également essayer d'autres modèles d'appren-

tissage intégrés basés sur des arbres tels que Xgboost et GBDT.

## 5 Code

Le code suivant permet de charger les données `pokemon` et de représenter les plots dans la partie de l'analyse exploratoire :

```

1 """load documents"""
2 combats =
    → pd.read_csv("pokemon-challenge/combats.csv")
3 pokemon =
    → pd.read_csv("pokemon-challenge/pokemon.csv")
4 """fill missing values"""
5 data = pokemon.copy()
6 data["Name"] =
    → pokemon['Name'].fillna('Primeape')
7 data["Type 2"] = pokemon['Type
    → 2'].fillna('No Type 2')
8 """feature engineering"""
9 # calculate the win % of each pokemon
10 total_Wins =
    → combats.Winner.value_counts()
11 # get the number of wins for each pokemon
12 numberOfWorks =
    → combats.groupby('Winner').count()
13 countByFirst =
    → combats.groupby('Second_pokemon').count()
14 countBySecond =
    → combats.groupby('First_pokemon').count()
15 numberOfWorks = numberOfWorks.sort_index()
16 numberOfWorks['Total Fights'] =
    → countByFirst.Winner +
    → countBySecond.Winner
17 numberOfWorks['Win Percentage']=
    → numberOfWorks.First_pokemon/numberOfWorks[
        →  Fights']
18 # merge the winning dataset and the
    → original pokemon dataset
19 results2 = pd.merge(data, numberOfWorks,
    → right_index = True, left_on='#')
20 results3 = pd.merge(data, numberOfWorks,
    → left_on='#', right_index = True,
    → how='left')
21 results4 =
    → results3.drop(columns=['First_pokemon',
    → 'Second_pokemon', 'Total Fights'])
22 """PCA"""
23 results2["win"] = 'win percentage < 0.5'
24 results2.loc[results3["Win
    → Percentage"]>0.5, "win"] = 'win
    → percentage > 0.5'

25 X = results2.iloc[:,4:10]
26 pca = prince.PCA(n_components=6,
    → n_iter=3, rescale_with_mean=True,
    → rescale_with_std=True, copy=True,
    → check_input=True, engine='auto',
    → random_state=42)
27 pca = pca.fit(X)
28 ax = pca.plot_row_coordinates(X, ax=None,
    → figsize=(6, 6), x_component=0,
    → y_component=1, labels=None,
    → color_labels=results2.iloc[:, -1],
    → ellipse_outline=False,
    → ellipse_fill=True, show_points=True)
29 """One-class SVM to detect outliers"""
30 svm = OneClassSVM(kernel='rbf',
    → gamma=0.001, nu=0.02)
31 pred = svm.fit_predict(X)
32 scores = svm.score_samples(X)
33 thresh = quantile(scores, 0.005)
34 index = where(scores<=thresh)
35 index = index[0]
36 values = X.iloc[index]
37 plt.scatter(X['Speed'], X['Attack'])
38 plt.scatter(values['Speed'],
    → values['Attack'], color='r')

```

Ce code permet de préparer les données dans le but d'effectuer une classification automatique sur les pokémons, correspondant à la partie de la classification non-supervisée.

```

1 pokemon = pd.read_csv("pokemon.csv")
2 combats = pd.read_csv("combats.csv")
3 pok = pokemon.drop(columns=["Name", "Type
    → 1", "Type 2", "Generation",
    → "Legendary", "#"])
4 # KMEANS
5 Totalcls = KMeans(n_clusters=2, init="random")
6 cls.fit(pok)
7 # Visualisation - AVEC PCA
8 uti.scatterplot_pca(hue=cls.labels_,
    → data=pok, style = pokemon.Legendary)
9
10 pok_etiqu = pd.DataFrame({ "HP":(
    → pok["HP"], "Attack": pok["Attack"],
    → "Defense": pok["Defense"], "Sp. Atk":(
        → pok["Sp. Atk"], "Sp. Def": pok["Sp.
        → Def"], "Speed": pok["Speed"],
        → "etiquette": cls.labels_)})
11 # Récupération des pokemons classe 1
12 pok_c1 = pok_etiqu[pok_etiqu["etiquette"]
    → == 1]
13 pok_c0 = pok_etiqu[pok_etiqu["etiquette"]
    → == 0]

```

```

14 # Distribution HP
15 plt.subplot(2,3,1)
16 hp = sns.distplot(pok_c1["HP"],
17     kde_kws={"label": "classe 1"})
17 sns.distplot(pok_c0["HP"],
18     kde_kws={"label": "classe 0"})
18 pok_combats = pd.merge(combats,
19     pok_etiqu, left_on = ['Winner'],
20     right_on = pokemon.index)
21 # Découpage du dataframe selon les
22 # classes 0 et 1
20 pok_combats_c1 =
21     pok_combats[pok_combats["etiquette"]
22     == 1]
21 pok_combats_c0 =
22     pok_combats[pok_combats["etiquette"]
23     == 0]
22 # Pourcentage de gagnants classe 1 et 0
23 pour_c1 =
24     len(pok_combats_c1)/len(pok_combats)
24 pour_c0 =
25     len(pok_combats_c0)/len(pok_combats)
25 prop1 = [pour_c1, pour_c0]
26 plt.pie(prop1, labels = ['Classe 1',
27     'Classe 0'], autopct='%.1f%')

```

Ce code permet de préparer les données dans le but d'effectuer une classification automatique sur les combats, correspondant à la partie de la classification non-supervisée.

```

1 # Récupération de pokemon avec que les
2     variables quantitatives
2 pokemon_sans_qual =
3     pokemon.drop(columns=["Name", "Type
4     1", "Type 2", "Generation",
5     "Legendary", "#"])
3 new_pok = pd.merge(combats, pokemon,
4     left_on = ['First_pokemon'], right_on
5     = ['#'])
4 new_pokem = pd.merge(new_pok, pokemon,
5     left_on = ['Second_pokemon'],
6     right_on = ['#'])
5 pokemon_combats =
6     new_pokem.drop(columns=["Name_x",
7     "Type 1_x", "Type 2_x",
8     "Generation_x", "Legendary_x", "#_x",
9     "Name_y", "Type 1_y", "Type 2_y",
10    "Generation_y", "Legendary_y", "#_y",
11    "First_pokemon", "Second_pokemon",
12    "Winner"])
7
8 cls = KMeans(n_clusters=2, init="random")
9 cls.fit(pokemon_combats)

```

```

10 uti.scatterplot_pca(hue=cls.labels_,
11     data=pokemon_combats)
11 attack_temp = pd.DataFrame({"x":
12     pokemon_combats["Attack_x"], "y":
13     pokemon_combats["Attack_y"],
14     "etiquette": cls.labels_})
12 attack_c1 =
13     attack_temp[attack_temp["etiquette"]
14     == 1]
13 # Distribution Classe 1
14 sns.distplot(attack_c1["x"],
15     kde_kws={"label": "first"})
15 at_1 = sns.distplot(attack_c1["y"],
16     kde_kws={"label": "second"})
16
17 att_com_temp = pd.merge(combats,
18     attack_temp, on = combats.index)
18 attaque_combats =
19     att_com_temp.drop(columns =
20     ["key_0"])
19 attaque_combats.columns =
20     ["First_pokemon", "Second_pokemon",
21     "Winner", "Attack_first",
22     "Attack_second", "etiquette"]
20 attaque_combats_c1 =
21     attaque_combats[attaque_combats["etiquette"]
22     == 1]
21 attaque_combats_c0 =
22     attaque_combats[attaque_combats["etiquette"]
23     == 0]
22 # Pourcentage de pokemon First et second
23     gagnant dans CLASSE 1
23 first_winner_1 =
24     attaque_combats_c1[attaque_combats_c1["First_poke
25     == attaque_combats_c1["Winner"]]
24 pourcentage_first_1 =
25     len(first_winner_1)/len(attaque_combats_c1)*100
25 second_winner_1 =
26     attaque_combats_c1[attaque_combats_c1["Second_pok
27     == attaque_combats_c1["Winner"]]
26 pourcentage_second_1 =
27     len(second_winner_1)/len(attaque_combats_c1)*100
27 # VISUALISATION POURCENTAGES Classe 1
28 prop1 = [pourcentage_first_1,
29     pourcentage_second_1]
29 plt.subplot(1,2,1)
30 plt.pie(prop1, labels = ['First',
31     'Second'], autopct='%.1f%')

```