1. Ma compréhension de Numpy

- Numpy est une bibliothèque permettant de manipuler plus aisément les tableaux (de façon plus performantes et plus économes en mémoire, particulièrement lorsqu'il y a de grandes quantités de données)

- Numpy permet des opérations sur les tableaux sans utiliser de boucles (comme for par exemple), les opérations prisent en charge sont nombreuses, comme l'addition, la multiplication, statistiques, etc...

- Numpy permet également de manipuler les tableaux de façon plus efficace et simple, on peut grâce à numpy redimensionner des tableaux, ou encore les transposer, gérer les axes, etc...

2. Valeur Ajoutée des primitives utilisées

1. `np.genfromtxt`

- **Valeur ajoutée** : Cette fonction permet de charger un fichier CSV en tant que tableau NumPy structuré tout en spécifiant le séparateur, les types de données, et d'autres options comme l'encodage.
- Elle simplifie le traitement des données en les transformant en un format facile à manipuler (tableau NumPy). On peut alors accéder aux colonnes par leur nom, comme `data["Class"]`.

2. `data[...]` et filtrage conditionnel

- **Valeur ajoutée** : NumPy permet de sélectionner directement des sous-ensembles de données via les booléens. Par exemple, `data[data["Class"] == "Guerrier"]` extrait toutes les lignes où la classe est "Guerrier".
- **Comment cela vous aide** : Cela rend le code plus lisible et élimine le besoin d'écrire des boucles explicites pour filtrer les données.

3. `np.max` et `np.argmax`

- **Valeur ajoutée** :
    - `np.max` : Retourne la valeur maximale dans un tableau.
    - `np.argmax` : Retourne l'indice de la valeur maximale dans un tableau.

- Ces fonctions simplifient le processus de recherche des éléments maximaux dans un tableau. Par exemple, elles permettent d'identifier rapidement le joueur avec le plus grand nombre de victoires.

4. `np.where`

- **Valeur ajoutée** : Renvoie les indices des éléments satisfaisant une condition donnée. Cela permet une approche plus simple pour localiser des éléments spécifiques dans les données (par exemple, les indices des guerriers).

5. `np.unique`

- **Valeur ajoutée** : Renvoie les valeurs uniques d'un tableau. Cela m'a aidé pour identifier les équipes et les classes uniques dans le jeu.

6. `np.sum`

- **Valeur ajoutée** : Calcule la somme des éléments d'un tableau. Elle m'a permis de calculer facilement le total des victoires par équipe ou autre groupe, comme dans la question 4.

7. `np.corrcoef`

- **Valeur ajoutée** : Calcule la matrice de corrélation entre deux tableaux. Elle m'a permis de mesurer la relation entre le nombre de parties jouées et le nombre de victoires.

9. `round`

- **Valeur ajoutée** : Elle arrondit les valeurs numériques à un nombre donné de décimales. Je l'ai utilisé pour arrondir au centième les moyennes de victoires par classe à la question 8, rendant les résultats plus lisibles.

10. `np.array`

- **Valeur ajoutée** : Convertit une liste Python en tableau NumPy. Cela m'a permis d'appliquer les fonctions NumPy (comme `np.sum` ou `np.argmax`) sur des listes, comme dans le calcul du total des victoires par équipe.

3. Description de mes solutions

Question 1 : Trouver le meilleur guerrier

Démarche :

J'ai commencé par filtrer les joueurs de la classe "Guerrier" en utilisant une condition sur la colonne `"Class"`. Ensuite, j'ai identifié le guerrier ayant remporté le plus de victoires en trouvant la valeur maximale dans la colonne `"Nb_Victoires"`. Enfin, j'ai extrait son pseudo pour l'afficher.

Choix faits :

- J'ai utilisé un filtrage conditionnel (`data[data["Class"] == "Guerrier"]`) pour extraire directement les guerriers sans boucle.
- Pour trouver le meilleur guerrier, j'ai préféré `np.max` et le filtrage par condition, car c'est simple et lisible.

Résultat obtenu :

Le meilleur guerrier est **Player_83**, avec un maximum de victoires parmi les guerriers.

Question 2 : Trouver le meilleur archer

Démarche :

Le raisonnement est identique à la question précédente, mais cette fois-ci appliqué aux archers. J'ai extrait les joueurs de la classe "Archer", puis j'ai cherché celui avec le plus grand nombre de victoires.

Choix faits :

J'ai utilisé la même méthodologie que pour les guerriers pour garantir cohérence et simplicité dans mon approche.

Résultat obtenu :

Le meilleur archer est **Player_75**, avec le plus grand nombre de victoires.

Question 3 : Identifier les équipes uniques

Démarche :

J'ai utilisé la fonction `np.unique` pour extraire toutes les équipes présentes dans la colonne `"Team"`. Cette méthode est rapide et efficace pour identifier des valeurs uniques.

Choix faits :

J'ai privilégié `np.unique` pour éviter les boucles ou des manipulations complexes, car cette fonction est spécifiquement conçue pour ce type de tâche.

Résultat obtenu :

Les équipes uniques sont **Blue**, **Green**, et **Red**.

Question 4 : Trouver la meilleure équipe

Démarche :

J'ai calculé le total des victoires pour chaque équipe en :

1. Identifiant les équipes uniques avec `np.unique`.
2. Utilisant une compréhension de liste pour calculer la somme des victoires des joueurs de chaque équipe.
3. Trouvant l'équipe avec le plus grand total à l'aide de `np.argmax`.

Choix faits :

- J'ai utilisé une liste de compréhension combinée à `np.sum` pour simplifier le calcul des totaux par équipe.
- `np.argmax` m'a permis d'identifier rapidement l'équipe gagnante.

Résultat obtenu :

La meilleure équipe est **Green**, avec un total de **1191 victoires**.

Question 5 : Trouver le meilleur joueur toutes classes confondues

Démarche :

J'ai recherché le joueur ayant le plus grand nombre de victoires dans l'ensemble des données, en utilisant une condition sur `data["Nb_Victoires"].max()`.

Choix faits :

J'ai préféré cette approche pour sa simplicité et sa capacité à donner directement le joueur le plus performant.

Résultat obtenu :

Le meilleur joueur est **Player_94**, toutes classes confondues.

Question 6 : Identifier les classes uniques

Démarche :

J'ai utilisé `np.unique` sur la colonne `"Class"` pour identifier les classes de personnages présentes dans le dataset.

Choix faits :

`np.unique` est une solution idéale ici, car elle est concise et adaptée à l'extraction de valeurs distinctes.

Résultat obtenu :

Les classes uniques sont **Archer**, **Guerrier**, **Healer**, et **Mage**.

Question 7 : Corrélation entre parties jouées et victoires

Démarche :

J'ai utilisé `np.corrcoef` pour calculer la corrélation entre le nombre de parties jouées (`"Nb_Parties"`) et le nombre de victoires (`"Nb_Victoires"`). Cette méthode donne directement une matrice de corrélation.

Choix faits :

`np.corrcoef` est adaptée ici, car elle fournit rapidement une mesure de la relation entre deux variables.

Résultat obtenu :

La corrélation est d'environ **0.548**, ce qui indique une relation modérée entre les parties jouées et les victoires.

Question 8 : Moyenne des victoires par classe

Démarche :

J'ai calculé la moyenne des victoires pour chaque classe en :

1. Identifiant les classes uniques avec `np.unique`.
2. Calculant la moyenne des victoires pour chaque classe avec une compréhension de dictionnaire.

Choix faits :

J'ai utilisé une compréhension de dictionnaire pour associer chaque classe à sa moyenne, et j'ai utilisé `round` pour arrondir les résultats.

Résultat obtenu :

- Archer : **32.45**
- Guerrier : **22.55**
- Healer : **28.74**
- Mage : **36.33**

Question 9 : Graphique de dispersion

Démarche :

J'ai utilisé `matplotlib` pour créer un graphique de dispersion montrant la relation entre le nombre de parties jouées et le nombre de victoires. J'ai ajouté des axes et un titre pour rendre le graphique plus lisible.

Choix faits :

J'ai utilisé `plt.scatter` car il est conçu pour représenter des relations entre deux variables quantitatives.

Résultat obtenu :

Le graphique montre une tendance positive, ce qui reflète la corrélation modérée observée précédemment.

Bonus : Explication des corrélations

1. Qu'est-ce qu'une corrélation ?

- La **corrélation linéaire** est représentée par un coefficient, souvent noté r, qui varie entre **-1** et **1** :
  - r = 1 : Relation linéaire positive parfaite (les deux variables augmentent ensemble).
  - $r = -1$: Relation linéaire négative parfaite (une variable augmente pendant que l'autre diminue).
  - $r = 0$: Absence de relation linéaire (les deux variables ne montrent aucun lien direct).

Dans la question 7 : le nombre de parties jouées et le nombre de victoires pour chaque joueur.

2. Résultat obtenu

- La matrice retournée par np.corrcoef(nb_parties, nb_victoires) est la suivante :

```
[[1.         0.54836849]
 [0.54836849 1.        ]]
```

  - La diagonale (r = 1) représente la corrélation d'une variable avec elle-même.
  - L'élément 0.548 est la corrélation entre les parties jouées et les victoires.

3. Interprétation

- Un coefficient de **0.548** indique une **relation positive (néanmoins modérée)** :
    - Plus un joueur participe à des parties, plus il a de chances de remporter des victoires.
    - Cependant, la relation n'est pas parfaite, car d'autres facteurs (comme les compétences du joueur, la classe choisie, ou l'équipe) peuvent influencer le nombre de victoires.

4. Pourquoi est-ce important ?

Analyser les corrélations peut révéler des relations significatives dans les données :

- Une forte corrélation ($r > 0.7$) aurait indiqué que le simple fait de jouer plus augmente considérablement les chances de gagner.
- Ici, une corrélation modérée ($r \approx 0.55$) montre qu'il existe une tendance positive, mais d'autres variables méritent d'être explorées pour comprendre ce qui fait vraiment un joueur performant.

5. Limites et compléments possibles

- La corrélation est une mesure linéaire. Si la relation entre les variables est non linéaire, le coefficient r peut ne pas capturer l'intégralité de cette relation.
- Une analyse plus complète pourrait inclure d'autres facteurs comme la classe du joueur, la stratégie d'équipe ou encore le ratio victoires/défaites.

En résumé, la corrélation observée confirme que **le nombre de parties jouées influence positivement le nombre de victoires**, mais elle reste **modérée**, ce qui laisse place à des analyses plus détaillées pour mieux comprendre les performances des joueurs.