Projet ISSD

Élève: Loïs GALLAUD 1A Généraliste

Date de début: 29/03/2023

```
# Importation des librairies nécessaires
import numpy as np
import csv
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
sns.set(color_codes = True)
sns.set(font_scale=1.5) # fixe la taille de la police à 1.5 * 12pt
from scipy import stats
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
from sklearn.linear_model import LinearRegression
from scipy.optimize import curve_fit
```

I - CHOIX DU SUJET

1) Analyse des données

Je décide dans un premier temps de regarder quelles sont les données que j'aurai à traiter:

- **Données météo**: `|Date|Pression au niveau mer|Variation de pression en 3 heures|Direction du vent moyen 10 mn|Vitesse du vent moyen 10 mn|Température|Point de rosée|Humidité|Visibilité horizontale|Temps présent|Temps passé 1|Temps passé 2|Nebulosité totale|Nébulosité des nuages de l' étage inférieur|Hauteur de la base des nuages de l'étage inférieur|Type des nuages de l'étage inférieur|Type des nuages de l'étage moyen|Type des nuages de l'étage supérieur|Pression station|Niveau barométrique|Variation de pression en 24 heures|Méthode de mesure Température du thermomètre mouillé|Température du thermomètre mouillé|Rafale sur les 10 dernières minutes|Rafales sur une période|Periode de mesure de la rafale|Etat du sol|Hauteur totale de la couche de neige, glace, autre au sol|Hauteur de la neige fraîche|Periode de mesure de la neige fraiche|Précipitations dans la dernière heure|Précipitations dans les 3 dernières heures|Précipitations dans les 12 dernières heures|Précipitations dans les 24 dernières heures|Phénomène spécial 1|Phénomène spécial 2|Phénomène spécial`

`3|Phénomène spécial 4|Nom|Température (°C)|Température minimale sur 12 heures (°C)|Température minimale sur 24 heures (°C)|Température maximale sur 12 heures (°C)|Température maximale sur 24 heures (°C)|Température minimale du sol sur 12 heures (en °C)|Latitude|Longitude|Altitude|department (name)|department (code)|region (name)|mois_de_l_annee`

- **Données jeu de société**: `name|yearpublished|sortindex|minplayers|maxplayers|minplaytime|maxplaytime|minage|usersrated|average|stddev|avgweight|numweights|numcomments|boardgamehonor|boardgamecategory|boardgamemechanic|boardgamefamily`

2) Remarques:

- Les données météo sont divisées en beaucoup plus de critères que les données des jeux de société.

  Cela veut donc dire que je risque de devoir faire plus de travail sur les interprétations des données des jeux de société que sur les données météo.

- Taille des fichiers de données: le fichier `csv` des données météo (`226032 Ko`) est beaucoup plus lourd que les données des jeux de société (`4580 Ko`). Les données météo sont donc beaucoup plus lourdes à traiter.

  Cela veut aussi dire que les résultats de mes analyses seront peut être plus précises sur les données météo que sur les données des jeux de société.

3) Choix

Pour pouvoir me laisser une plus grande liberté dans le choix de mes analyses, **je décide de travailler sur les données des jeux de société**.

II - PRÉTRAITEMENT DES DONNÉES

1) Chargement des données:

```
DATA = pd.read_csv('./DATA/dirty_boardgames.csv', delimiter = ';',
index_col = 'sortindex')
```

2) Données doublons/manquantes

On enlève les doublons et les valeurs manquantes:

> Je décide dans un premier temps de supprimer les lignes qui contiennent des valeurs manquantes. Sans considérer les valeurs manquantes, je peux déjà faire des analyses sur les données. Je pourrais ensuite essayer de remplacer les valeurs manquantes par des valeurs moyennes ou par des valeurs qui ont du sens si cela prend sens par la suite.

```
df = DATA.copy() # on crée l'objet df qui est une copie de DATA
length_before = len(df)

df.drop_duplicates() # supprime les lignes dupliquées
df.dropna(how = 'any') # supprime les lignes contenant des nan pour
toutes les variables du df
length_after = len(df)

print(f'Nombre de lignes supprimées : {length_before - length_after}')
# on affiche le nombre de lignes supprimées
print(f'Nombre de lignes et de colonnes au total: {df.shape}') # on
affiche le nombre de lignes et de colonnes du df
```

```
Nombre de lignes supprimées : 0
Nombre de lignes et de colonnes au total: (20000, 17)
```

On remarque que les données ne contiennent **pas de doublons ni de valeurs manquantes**.

De plus, le fichier `csv` contient exactement 20 000 lignes, **ce qui est relativement peu** comparé à l'autre jeu de données. Je n'ai donc pas intérêt à supprimer des lignes.

3) Données aberrantes

On peut dans un premier temps essayer de détecter les valeurs aberrantes à l'œil. Pour cela, on peut utiliser la fonction `describe()` qui nous donne des informations sur les données.

```
df.describe()
```

```
       yearpublished    minplayers   maxplayers   minplaytime
maxplaytime  \
count   20000.000000  20000.000000  20000.00000  20000.000000
20000.00000
mean     1981.268700      2.055250      5.59215     68.096450
94.28945
std       219.223277      0.745537     15.04921    466.502106
1005.75196
min     -3500.000000      0.000000      0.00000      0.000000
0.00000
25%      1997.000000      2.000000      4.00000     20.000000
30.00000
50%      2008.000000      2.000000      4.00000     30.000000
45.00000
75%      2015.000000      2.000000      6.00000     60.000000
90.00000
max      2021.000000     10.000000    999.00000  60000.000000
120000.00000

             minage    usersrated       average        stddev
avgweight  \
count  20000.000000  20000.000000  20000.000000  20000.000000
```

```
20000.000000
mean       9.476350    739.616350      6.275440      1.499691
1.931761
std        3.738842   3096.843206      1.065339      0.340743
0.897206
min        0.000000      0.000000      0.000000      0.000000
0.000000
25%        8.000000     47.000000      5.712778      1.309920
1.255050
50%       10.000000    105.000000      6.354170      1.471575
1.910900
75%       12.000000    333.250000      6.955793      1.663782
2.500000
max       25.000000  90730.000000      9.442860      4.500000
5.000000

         numweights   numcomments
count  20000.000000  20000.000000
mean      48.059050    191.515100
std      198.113638    606.892077
min        0.000000      0.000000
25%        4.000000     21.000000
50%        9.000000     44.000000
75%       25.000000    122.000000
max     7104.000000  17143.000000
```

Remarques:

- colonne `yearpublished`: la valeur minimale est de -3500. La valeur de l'écart s'en retrouve donc impactée.
- colonnes `min/maxplayers` & `min/maxplaytime`: les valeurs minimales et maximales ne sont pas vraiment interprétables: est ce que 0 dans `minplayers` signifie que le jeu n'a pas de minimum de joueurs ou est ce que cela signifie que le jeu est un jeu solo? De même pour les valeurs maximales comme celle de `maxplayers` de 999 (pas de maximum de joueurs?).
- colonne `usersrated`: rien qu'ne regardant la valeur d'écart type on s'attend à avoir des valeurs aberrantes. Cependant, la colonne `usersrated` est une colonne qui contient le nombre de personnes qui ont noté le jeu. Il est donc normal d'avoir des valeurs très grandes si l'ensemble de données contient des jeux de société très connus. Le même phénomène est présent su rla colone `numweights` et `numcomments`.

Conclusion de cette première analyse:

Le jeu de données doit être étudié plus en détails. Il est possible que certaines colonnes ne soient pas interprétables ou qu'elles contiennent des valeurs aberrantes OU BIEN au contraire qu'elles contiennent des valeurs qui ont du sens: pour cela j'ai besoin d'aller sur le site internet d'où le jeu de données a été téléchargé pour comprendre le sens de chaque colonne.

4) Recherches sur Board Game Geek

Pour essayer de comprendre à quoi servent chaque colonne, je décide de regarder plus en détails un exemple sur le site internet Board Game Geek. À l'heure actuelle, le jeu de société le mieux noté du site est le jeu Brass: Birmingham.

```
brass = df[df['name'].str.contains('Birmingham', na=False)]
brass.head(n=1)
```

```
                        name  yearpublished  minplayers  maxplayers  \
sortindex
5          Brass  Birmingham           2018           2           4

           minplaytime  maxplaytime  minage  usersrated  average
stddev  \
sortindex

5                   60          120      14       10070  8.62031
1.22876

           avgweight  numweights  numcomments  \
sortindex
5             3.9122         467         1702

                                              boardgamehonor  \
sortindex
5          ['2018 Golden Geek Best Board Game Artwork & P...

                                           boardgamecategory  \
sortindex
5          ['Economic', 'Industry / Manufacturing', 'Tran...

                                           boardgamemechanic  \
sortindex
5          ['Hand Management', 'Income', 'Loans', 'Market...

                                             boardgamefamily
sortindex
5          ['Beer', 'Brass', 'Cities: Birmingham (England...
```

> Veuillez noter que **le jeu de données a été téléchargé il y a quelques années**. Il est donc possible que le jeu de données ne contienne pas les dernières données du site internet. On suppose que le jeu de données a été téléchargé en 2021 car le maximum de la colonne `yearpublished` est de 2021.

On peut déjà retrouver des colonnes qui nous intéresse sur la page d'accueil du jeu et en déduire des choses:

- `avgweight`: sa note est de 3.90/5, ce qui confirme bien que la données doit aller de 0 à 5. Il s'agit du "niveau de complexité du jeu".
- `min/maxplaytime`: on comprend désormais que les notations "60 - 120 min" est notée dans la dataframe minplaytime = 60 et maxplaytime = 120.
- `min/maxplayers`: on comprend que les notations "2 - 4" est notée dans la dataframe minplayers = 2 et maxplayers = 4.
- `usersrated`: on voit sur cet exemple qu'il y a bel et bien des jeux qui peuvent avoir beaucoup de notes. Notre première analyse qui était de dire que le jeu de données contenait des jeux très connus et des jeux moins connus **était donc correcte**.

Conclusion du prétraitement des données:

- Les données ont déjà l'air propres. Il n'y a pas de valeurs manquantes ni de doublons.
- Cependant il est possible que certaines colonnes contiennent des valeurs aberrantes.
- Certaines colonnes fonctionnent ensemble et nécessite de connaitre comment la notation est faite sur le site pour pouvoir les interpréter correctement. Par exemple, les colonnes `min/maxplayers` et `min/maxplaytime` sont liées.

  À mon avis, il est cohérent que les données soient aussi propres **puisqu'il s'agit de données rentrées par des utilisateurs sur un site internet**. Les utilisateurs sont donc sensibilisés à la qualité des données qu'ils rentrent et les données sont soumises à une validation par les administrateurs du site ou par un algorithme.

RÉSUMÉ DES COLONNES: | Nom de la colonne | Description |
|:----------:|:----------:| | yearpublished | Année de sortie du jeu | | sortindex | Indice du jeu | | minplayers | Nombre minimum de joueur | | maxplayers | Nombre maximum de joueur | | minplaytime | Durée minimum du jeu | | maxplaytime | Durée maximale du jeu | | minage | Age minimum conseillé pour jouer | | usersrated | Nombre de personne ayant noté ce jeu | | average | Note moyenne attribuée par les internautes | | stddev | Ecart type des notes attribuées par les internautes | | avgweight | Difficulté moyenne attribué à ce jeu de 0 à 5| | numweights | Nombre de personne ayant noté la difficulté de ce jeu | | numcomments | Nombre de commentaires | | boardgamehonor | Prix remportés | | boardgamecategory | Mot-clé(s) caractérisant la catégorie du jeu | | boardgamemechanic | Mot-clé(s) caractérisant les mécaniques principales du jeu| | boardgamefamily | Mot-clé(s) caractérisant la famille du jeu |

III - ANALYSE DU JEU DE DONNÉES

Je ne propose pas qu'une seule problématique pour l'analyse de ce jeu de données. Je préfère rester libre dans l'exploration des données au fur et à mesure que les questions se posent.

0) Premiers axes de réflexion:

Quelques questions qui me viennent à l'esprit:

- Quels sont les caractéristiques des jeux de société les mieux notés? (on pourra s'intéresser à leur difficulté et à l'impact du temps)
- Est-ce que les conclusions tirées sont les mêmes pour l'entièreté des jeux de société?

---

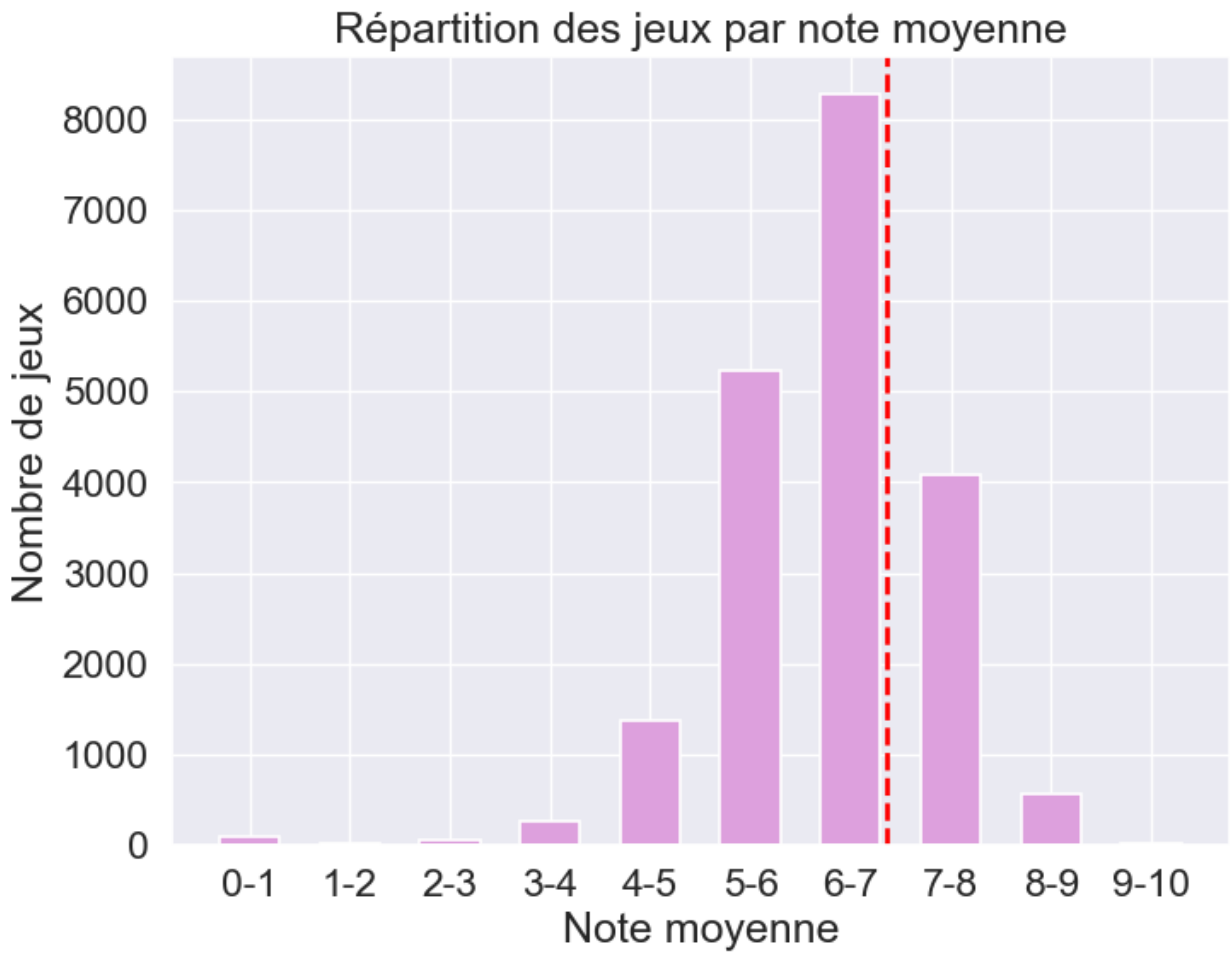
1) Les caractéristiques communes des jeux de société les mieux notés

On peut dans un premier temps se demander ce que veut dire **"les mieux notés"**. À partir de quelle note un jeu est considéré comme **"bien noté"**? Est ce qu'il suffit de prendre tous les jeux qui ont une note supérieure à 5? à 6?

Essayons de répondre à ces questions en regardant les statistiques descriptives de la colonne `average`:

```
# Intervalles
intervals = [[i, i+1] for i in range(10)]

# Calcul du nombre de jeux par intervalle
amounts = []
for interval in intervals:
    i, j = interval
    games_in_interval = df[(df['average'] >= i) & (df['average'] < j)]
    amounts.append(len(games_in_interval))

# Calcul de la moyenne, médiane et de l'écart type
mean = df['average'].mean()
median = df['average'].median()
print(median)
std = df['average'].std()
print(f"Moyenne: {mean:.2f} - Médiane: {median:.2f} - Ecart type:
{std:.2f}")

# Création de l'histogramme en barres
plt.figure(figsize=(8,6))
plt.title("Répartition des jeux par note moyenne")
plt.xlabel("Note moyenne")
plt.ylabel("Nombre de jeux")

labels = [f"{i}-{i+1}" for i in range(10)]
plt.bar(labels, amounts, linewidth=1.2, width=0.6, color="plum")
plt.axvline(median, color='red', linestyle='dashed', linewidth=2)
plt.show()
```

```
6.35417
Moyenne: 6.28 - Médiane: 6.35 - Ecart type: 1.07
```



On remarque qu'il y a peu de jeux qui ont été noté en dessous de 5. Il ne sert donc à **rien** de considérer un jeu comme "bien noté" s'il a une note supérieure à 5.

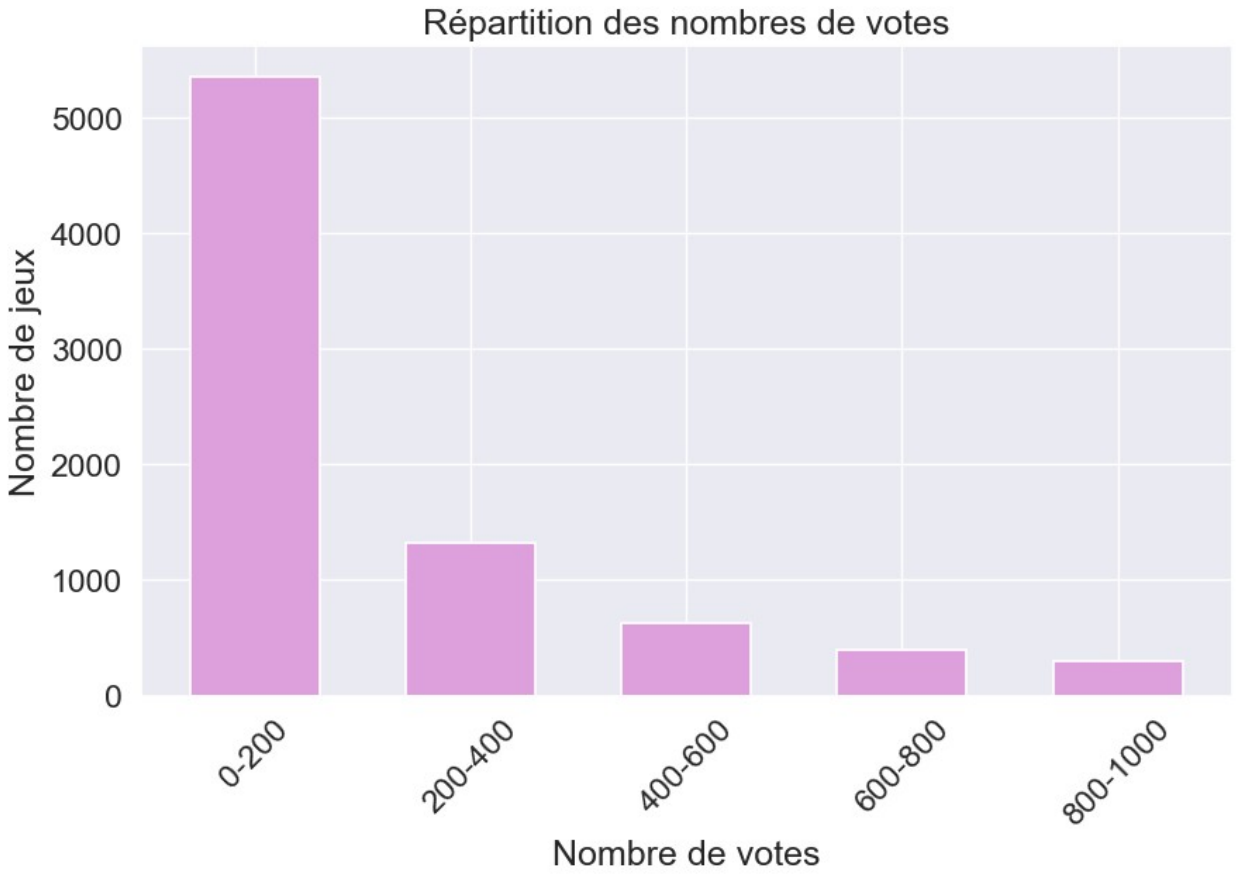
Considérons pour la suite que les jeux qui ont une note moyenne **supérieure ou égale à la médiane (6.35) sont des jeux "bien notés"**. Cela nous laisse 10 000 jeux à étudier. Appelons les jeux bien notés `good_games`.

```
good_games = df[df['average'] > median]
print(median)
print(f"Nombre de jeux ayant une note moyenne supérieure à la médiane:
{len(good_games)}")
```

```
6.35417
Nombre de jeux ayant une note moyenne supérieure à la médiane: 9997
```

Seconde chose à prendre en compte: **le nombre de vote**. Si un jeu est très bien noté mais qu'il n'y a que 5 votants, on pourrait considérer que la note n'est pas pertinente. Commençons par regarder la repartion du nombre de vote:

```
intervals = [[i, i+200] for i in range(0, 1000, 200)]

# Calcul du nombre de jeux par intervalle
amounts = []
for interval in intervals:
    i, j = interval
    games_in_interval = good_games[(good_games['usersrated'] >= i) &
(good_games['usersrated'] < j)]
    amounts.append(len(games_in_interval))

# Création de l'histogramme en barres
plt.figure(figsize=(10,6))
plt.title("Répartition des nombres de votes")
plt.xlabel("Nombre de votes")
plt.xticks(rotation=45)
plt.ylabel("Nombre de jeux")

labels = [f"{i}-{i+200}" for i in range(0, 1000, 200)]
plt.bar(labels, amounts, linewidth=1.2, width=0.6, color='plum')

plt.show()
```



On cherche à savoir **si la note moyenne est corrélée au nombre de vote**. On va utiliser l'écart type comme mesure de dispersion de notes en fonction de la note moyenne. On va donc regarder si la note moyenne est corrélée au nombre de vote grâce à une nuage de points et une légende de couleur.

On sépare le nuage de points en deux parties: les jeux notés par moins de 1000 personnes et ceux notés pas plus de 1000 personnes. Cette séparation est pertinente car on a pu voir avec le graphique ci-dessus qu'il y avait de **moins en moins** de jeux lorsqu'ils sont notés par beaucoup de personnes.

```
std_good_games1 = good_games[good_games['usersrated'] <
1000].sort_values(by='usersrated', ascending=True)
std_good_games2 = good_games[good_games['usersrated'] >
1000].sort_values(by='usersrated', ascending=True)

plt.figure(figsize=(20,5))
plt.subplot(1, 2, 1)
sns.scatterplot(x="average", y="stddev", data=std_good_games1,
hue="usersrated", hue_norm=(0, 1000))
plt.subplot(1, 2, 2)
sns.scatterplot(x="average", y="stddev", data=std_good_games2,
hue="usersrated")


plt.show()
```

On découpe en deux l'analyse en faisant un graphe comportant les moyennes des jeux les moins notés et un autre graphe comportant les moyennes des jeux les plus notés. Peu importe le nombre de votant (*montré par la couleur*) l'écart-type `stddev` **reste aux alentours de 1.5**.

Conclusion de cette pré-analyse: Les deux graphes ci-dessus montrent que la note moyenne est **très peu corrélée au nombre de vote**. Cela signifie que peu importe le nombre de votant, la note moyenne est **très proche de la note moyenne de tous les jeux**.

On peu donc **uniquement** considérer que les meilleurs jeux sont les jeux qui font partie des 10 000 jeux les mieux notés.

Découvrons maintenant les caractéristiques des jeux biens notés:

- En quelle année sont sortis les meilleurs jeux de société?

- Quelle est la difficulté moyenne des meilleurs jeux de société?
- Quelle est la mécanique de jeu des meilleurs jeux de société?

1-A) En quelle **année** sont sortis les meilleurs jeux de société? 🕰

Tout d'abord, faisons une première analyse des données pour voir si les jeux bien notés sont répartis de manière homogène dans le temps. Comme explicité dans le prétraitement des données, la colonne `yearpublished` contient effectivement des valeurs aberrantes. Pour savoir à partir de quelle date il est pertinent de considérer les jeux, regardons de nouveau les statistiques descriptives de la colonne `yearpublished`:

```
# Obtenir une liste des années présentes dans la dataframe good_games
years = pd.DataFrame(good_games['yearpublished'])
years.describe()
```

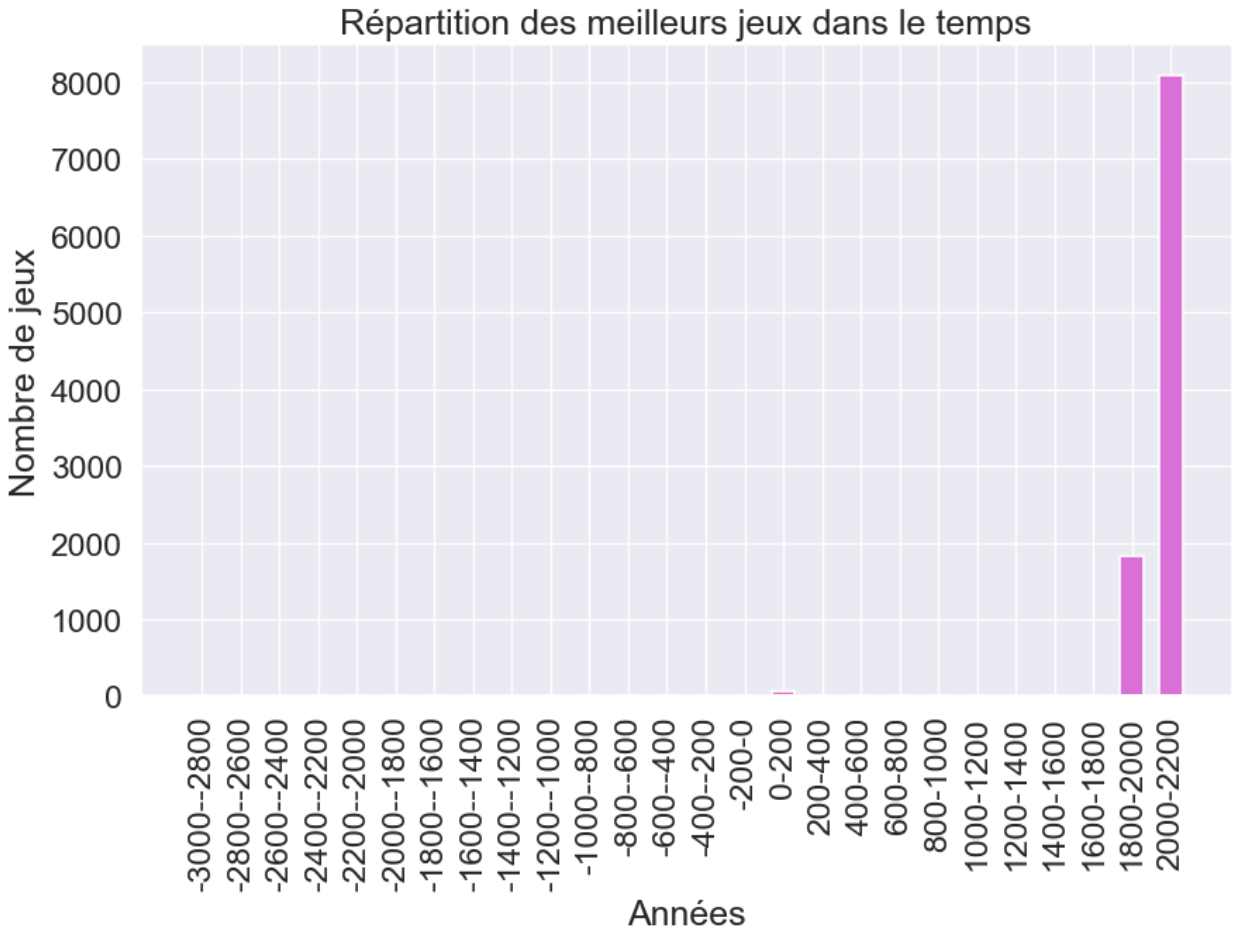
```
       yearpublished
count    9997.000000
mean     1994.642793
std       166.841691
min     -3000.000000
25%      2004.000000
50%      2013.000000
75%      2017.000000
max      2021.000000
```

On peut à présent se demander si les jeux bien notés sont répartis de manière homogène dans le temps. Pour cela, on va regarder la proportion de jeux bien notés par année:

```
intervals = [[i, i+200] for i in range(-3000, 2021, 200)]

# Calcul du nombre de jeux par intervalle
amounts = []
for interval in intervals:
    i, j = interval
    games_in_interval = years[(years['yearpublished'] >= i) &
(years['yearpublished'] < j)]
    amounts.append(len(games_in_interval))

# Calcul de la moyenne, médiane et de l'écart type
mean_years = years['yearpublished'].mean()
median_years = years['yearpublished'].median()
std_years = years['yearpublished'].std()
print(f"Moyenne: {mean_years:.2f} - Médiane: {median_years:.2f} -
Ecart type: {std_years:.2f}")

# Création de l'histogramme en barres
plt.figure(figsize=(10,6))
plt.title("Répartition des meilleurs jeux dans le temps")
plt.xlabel("Années")
plt.xticks(rotation=90)
```

```
plt.ylabel("Nombre de jeux")

labels = [f"{i}-{i+200}" for i in range(-3000, 2021, 200)]
plt.bar(labels, amounts, linewidth=1.2, width=0.6, color="orchid")
plt.show()
```

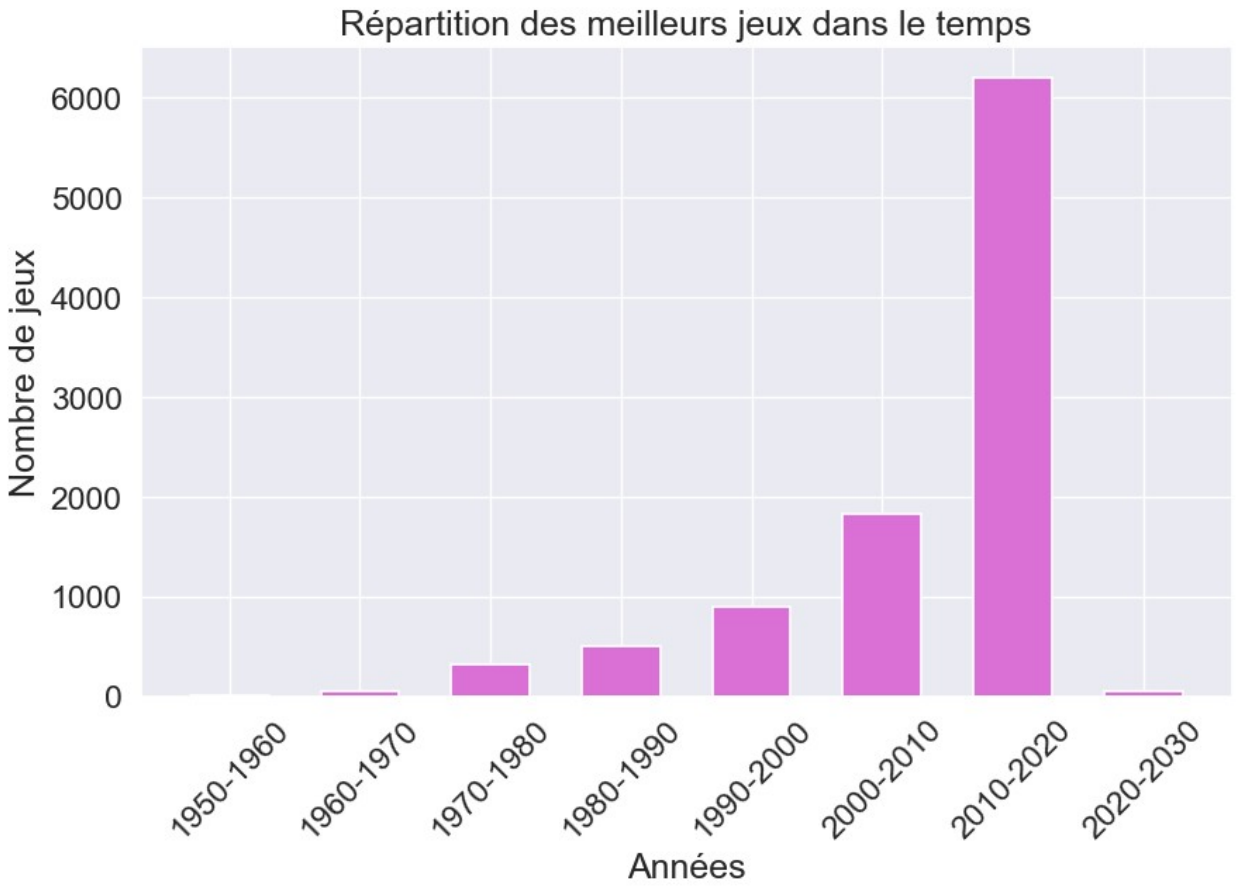
```
Moyenne: 1994.64 - Médiane: 2013.00 - Ecart type: 166.84
```



Un premier graphe montre que la proportion de jeux bien notés est essentiellement entre 1800 et 2021. En affinant les tranches d'années, on peut voir que la proportion de jeux bien notés est très faible avant 1950.

```
intervals = [[i, i+10] for i in range(1950, 2021, 10)]

# Calcul du nombre de jeux par intervalle
time_amounts = []
for interval in intervals:
    i, j = interval
    games_in_interval = years[(years['yearpublished'] >= i) &
(years['yearpublished'] < j)]
    time_amounts.append(len(games_in_interval))
```

```python
# Création de l'histogramme en barres
plt.figure(figsize=(10,6))
plt.title("Répartition des meilleurs jeux dans le temps")
plt.xlabel("Années")
plt.xticks(rotation=45)
plt.ylabel("Nombre de jeux")

time_labels = [f"{i}-{i+10}" for i in range(1950, 2021, 10)]
plt.bar(time_labels, time_amounts, linewidth=1.2, width=0.6,
color="orchid")
plt.show()
```



Visiblement **les jeux bien notés sont majoritairement sortis après 1950**. On peut conjecturer que cela est dû à la popularité grandissante du site BGG ces dernières années ou bien tout simplement à la popularité grandissante des jeux de société notamment grâce à des sites comme BGG.

> De plus, la répartition des meilleurs jeux dans le temps suit une **loi exponentielle**. On pourrait donc essayer de prédire le nombre de jeux bien notés dans les années à venir.

Conclusion: le jeu de données n'est pas fait pour étudier les jeux de société d'un point de vue temporel.

1-B) Quelle est la **mécanique de jeu / catégories** du jeu préférée?

Commençons par recenser les différentes mécanique de jeu présentes dans les meilleurs jeux de société:

```
mech_cat_games = pd.DataFrame({'name': good_games['name'],
                               'boardgamemechanic':
good_games['boardgamemechanic'],
                               'boardgamecategory':
good_games['boardgamecategory'],
                               'average': good_games['average'],})

mech_cat_games.head()
```

```
                                                    name  \
sortindex
1                                             Gloomhaven
2                                Pandemic Legacy  Season 1
3                                       Terraforming Mars
4          Through the Ages  A New Story of Civilization
5                                       Brass  Birmingham

                                             boardgamemechanic  \
sortindex
1           ['Campaign / Battle Card Driven', 'Cooperative...
2           ['Action Points', 'Cooperative Game', 'Hand Ma...
3           ['Card Drafting', 'End Game Bonuses', 'Hand Ma...
4           ['Action Points', 'Auction/Bidding', 'Auction:...
5           ['Hand Management', 'Income', 'Loans', 'Market...

                                             boardgamecategory  average

sortindex

1           ['Adventure', 'Exploration', 'Fantasy', 'Fight...  8.85292

2                                ['Environmental', 'Medical']  8.62499

3           ['Economic', 'Environmental', 'Industry / Manu...  8.42299

4                   ['Card Game', 'Civilization', 'Economic']  8.49419

5           ['Economic', 'Industry / Manufacturing', 'Tran...  8.62031
```

En utilisant la méthode `explode` de pandas, on peut facilement compter le nombre de fois où une mécanique de jeu est présente dans les meilleurs jeux de société:

```
mechanics = mech_cat_games.explode('boardgamemechanic')
['boardgamemechanic']

# Compter les occurrences de chaque mécaniques
```

```
mechanics_counts = mechanics.value_counts()
mechanics_counts.head(10)
```

```
['Hexagon Grid']                                    500
['Dice Rolling', 'Hexagon Grid', 'Simulation']      193
['Dice Rolling', 'Hexagon Grid']                    151
['Hand Management']                                 120
['Dice Rolling']                                    120
['Hexagon Grid', 'Simulation']                      104
['Tile Placement']                                   90
['Hand Management', 'Set Collection']                83
['Grid Movement']                                    66
['Cooperative Game']                                 66
Name: boardgamemechanic, dtype: int64
```

Les principales mécaniques de jeu les plus aimées sont les suivantes:

- **Hexagon Grid** (jeu sur un plateau hexagonal)
- **Dice Rolling** (jeu avec des dés)
- **Hand Management** (jeu avec des cartes)
- **Simulation** (jeu de simulation)
- **Tile Placement** (jeu avec des tuiles)

  On peut considérer que c'est représentatif des mécaniques les plus présentes même sans extraire chaque mécanique individuellement.

On peut faire la même chose pour les catégories de jeu:

```
categories = mech_cat_games.explode('boardgamecategory')
['boardgamecategory']

# Compter les occurrences de chaque mécaniques
categories_counts = categories.value_counts()
categories_counts.head(10)
```

```
['Wargame', 'World War II']          549
['Card Game']                        334
['Abstract Strategy']                289
['Napoleonic', 'Wargame']            167
['Card Game', 'Fantasy']             146
['American Civil War', 'Wargame']    108
['Dice']                              91
['Modern Warfare', 'Wargame']         87
['Fantasy']                           83
['Wargame', 'World War I']            79
Name: boardgamecategory, dtype: int64
```

Les principales catégories de jeu les plus aimées sont les suivantes:

- **Wargame** (jeu de guerre)
- **Card Game** (jeu de cartes)
- **Abstract Strategy** (jeu abstrait)
- **Fantasy** (jeu de fantasy)
- **Napoleonic** (jeu sur la période napoléonienne)

  Comme précédemment, on peut considérer que c'est représentatif des catégories les plus présentes même sans extraire chaque catégorie individuellement.

Les jeux sur un plateau hexagonal sont les plus aimés, ce qui coïncide avec le fait que les jeux de guerre soient les plus aimés puisque la majorités des jeux de plateau de guerre se jouent sur un plateau hexagonal. En effet, pour des raisons de game design, les jeux de guerres sont souvent représentés sur de tel plateau car ils permettent aux joueurs de se déplacer de manière plus naturelle et donne souvent plus de liberté au joueur.

Sur meeple mountain, on peut trouver des articles sur les jeux de guerre sur plateau hexagonal datant de 2020 qui montrent pourquoi les plateaux hexagonaux permettent des stratégies riches et variées ingame.

On peut donc en conclure qu'il peut exister des **corrélations entre les catégories de jeu et les mécaniques des meilleurs jeux**.

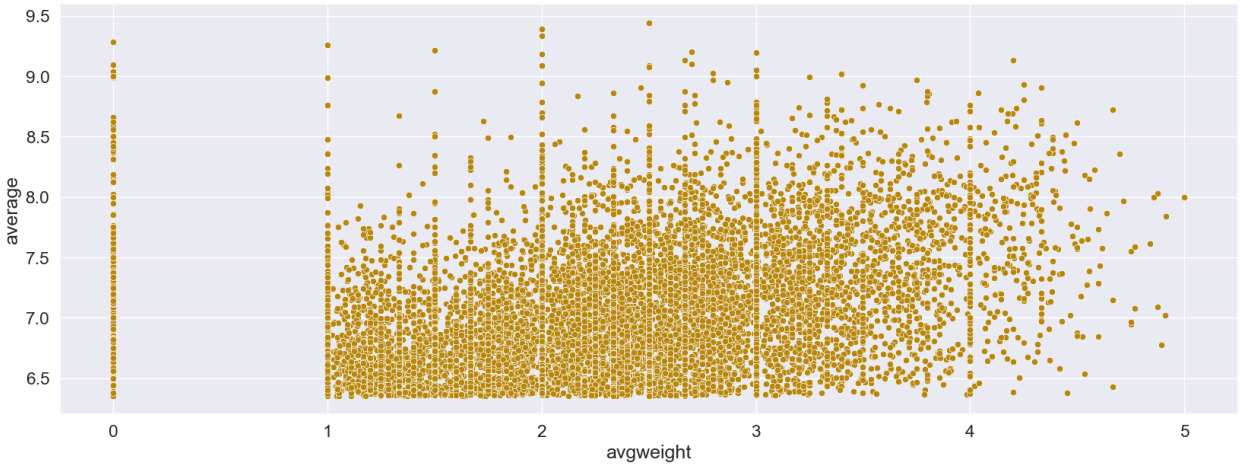
1-C) Quelle est la **difficulté** des meilleurs jeux de société?

On peut à présent se demander quelle est la difficulté moyenne des meilleurs jeux de société.

Pour cela, on va regarder la distribution de la colonne `avgweight`. On peut tracer un nuage de point des note moyenne en fonction de la difficulté:

```
plt.figure(figsize=(20,7))
sns.scatterplot(data=good_games, x='avgweight', y='average',
color="darkgoldenrod")
```

```
<AxesSubplot: xlabel='avgweight', ylabel='average'>
```



Remarques:

- Pour les valeurs entières de difficulté, on remarques des sortes de colonnes de points. Cela est dû au fait que de manière naturelle lorsqu'un internaute note un jeu il a tendance à mettre une note ronde plutôt qu'une note décimale. Par exemple, si un jeu est noté 6.5, il est plus probable qu'il soit noté 6 que 7.
- On remarque que globalement les points sont de plus en plus rare à mesure que la difficulté augmente. Cela est peut-être dû au fait que les internautes ont tendance à plus aimer les jeux simples.
- On peut voir un trou entre 0 et 1 de difficulté. Après des recharches sur le site BGG, on peut trouver que la note `weight` est une note compris entre 1 et 5.

Dans ce cas, ***pourquoi il y a des jeux à 0 de difficulté***? Créons une nouvelle dataframe et étudions les caractéristique communes de ces jeux:

```
difficulty0 = good_games[good_games['avgweight'] == 0]
difficulty0.head()
```

```
                                                  name  yearpublished
\
sortindex

4323                              Monikers  More Monikers           2018

5154              The Red Dragon Inn 7  The Tavern Crew           2018

5695                                          KLASK 4           2019

6322       Codex  Card-Time Strategy \u2013 Starter Set           2016

6514                          Monikers  Serious Nonsense           2019


           minplayers  maxplayers  minplaytime  maxplaytime  minage  \
```

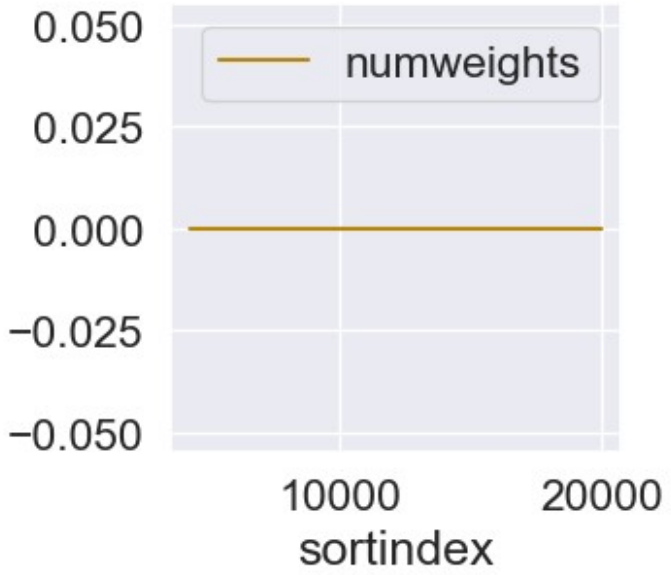
```
sortindex
4323                4          20           60           60      18
5154                2           4           30           60      13
5695                4           4           10           10       8
6322                2           2           45           45      13
6514                4          20           60           60      18

           usersrated  average   stddev  avgweight  numweights
numcomments  \
sortindex

4323              108  8.31523  1.10406        0.0           0
38
5154               99  8.16434  1.47135        0.0           0
24
5695               94  7.52106  1.41630        0.0           0
18
6322              102  6.99461  1.87742        0.0           0
30
6514               51  8.50980  1.03607        0.0           0
12

          boardgamehonor
boardgamecategory  \
sortindex

4323            ['None']  ['Card Game', 'Humor', 'Mature / Adult',
'Part...
5154            ['None']    ['Card Game', 'Fantasy', 'Humor', 'Party
Game']
5695            ['None']                                ['Action /
Dexterity']
6322            ['None']  ['Card Game', 'Fantasy', 'Fighting', 'Video
Ga...
6514            ['None']          ['Humor', 'Mature / Adult', 'Party
Game']

                                              boardgamemechanic  \
sortindex
4323              ['Acting', 'Card Drafting', 'Role Playing']
5154       ['Betting and Bluffing', 'Hand Management', 'P...
5695                                    ['Action / Dexterity']
6322       ['Deck', 'Hand Management', 'Secret Unit Deplo...
6514              ['Acting', 'Card Drafting', 'Role Playing']

                                               boardgamefamily
sortindex
4323                           ['Crowdfunding: Kickstarter']
5154       ['Crowdfunding: Kickstarter', 'The Red Dragon ...
5695                                                 ['KLASK']
```

```
6322                                          ['Fantasy Strike']
6514       ['Admin: Unreleased Games', 'Crowdfunding: Kic...
```

On Remarque que cela est dû au fait que les jeux à 0 de difficulté n'ont tout simplement **pas été noté.** On peut voir que dès que la colonne `avgweight` est à 0, la colonne `numweights` l'est aussi. On peut vérifier cette conjecture en regardant la distribution de la colonne `numweights` lorsque l'on prend que les jeux qui ont une difficulté de 0:

```
difficulty0.plot(y='numweights', figsize=(3,3), color="darkgoldenrod")
```

```
<AxesSubplot: xlabel='sortindex'>
```

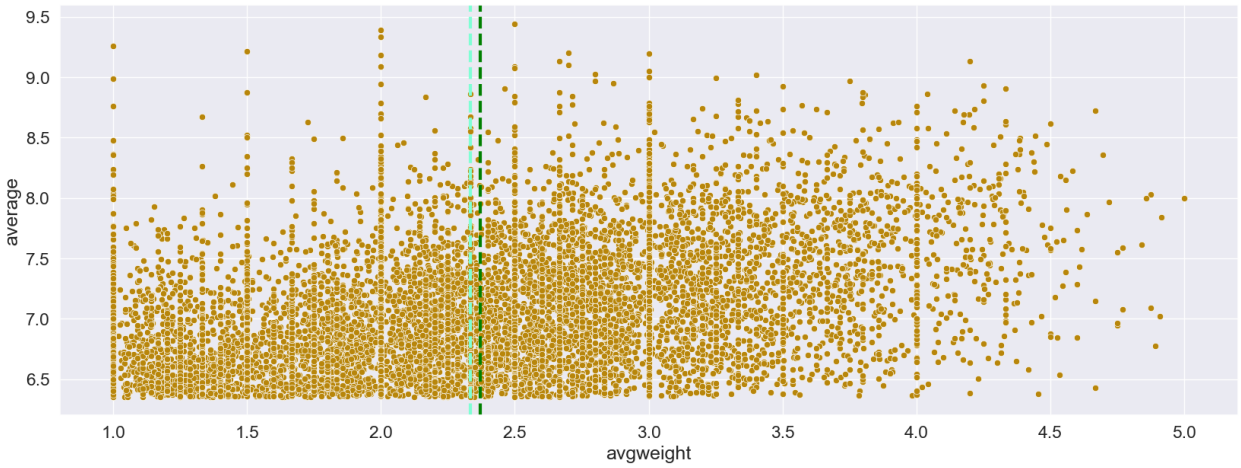


On peut donc supprimer ces jeux de notre dataframe et voici notre nouvelle distribution de la difficulté:

```
difficulty = good_games[good_games['avgweight'] > 0]
mean_difficulty = difficulty['avgweight'].mean()
median_difficulty = difficulty['avgweight'].median()

print(f"Médiane de la difficulté des meilleurs jeux:
{median_difficulty:.2f}")
print(f"Moyenne de la difficulté des meilleurs jeux:
{mean_difficulty:.2f}")

plt.figure(figsize=(20,7))
sns.scatterplot(data=difficulty, x='avgweight', y='average',
color="darkgoldenrod")
plt.axvline(mean_difficulty, color='green', linestyle='dashed',
linewidth=3)
plt.axvline(median_difficulty, color='aquamarine', linestyle='dashed',
```

```
linewidth=3)
plt.show()
```

```
Médiane de la difficulté des meilleurs jeux: 2.33
Moyenne de la difficulté des meilleurs jeux: 2.37
```

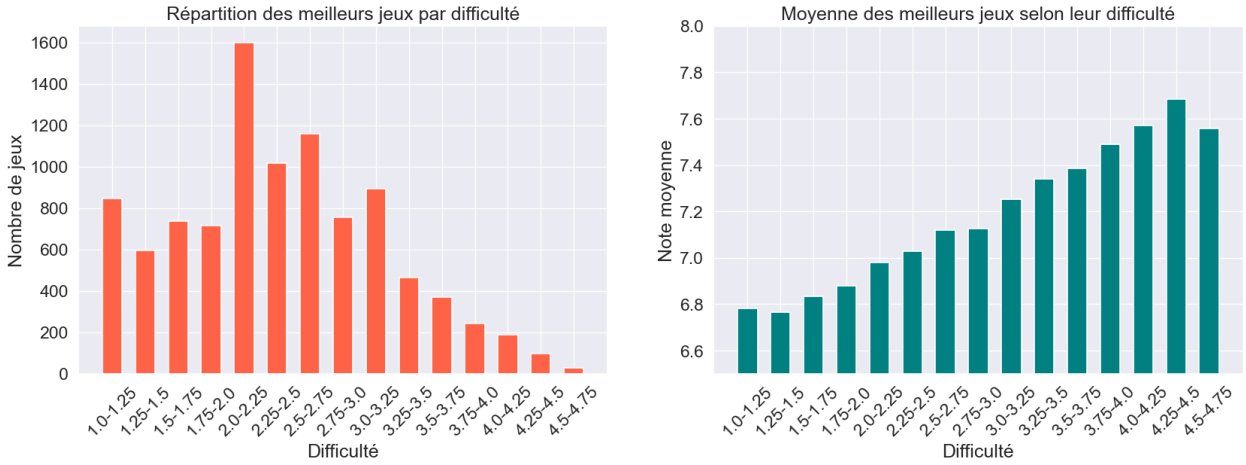


On peut se demander quelle est la tendance des gens lorsqu'ils notent bien les jeux. Ce que je vais essayer de chercher à voir c'est une éventuelle corrélation entre l'appréciation d'un jeu et sa difficulté chez les internautes du site BGG.

Regardons comment évolue la difficulté avec la note moyenne.

```
intervals = [[i, i+0.25] for i in np.arange(1, 4.75, 0.25)]

# Calcul du nombre de jeux par intervalle
amounts = []
average_ratings = []
for interval in intervals:
    i, j = interval
    games_in_interval = difficulty[(difficulty['avgweight'] >= i) &
(difficulty['avgweight'] < j)]
    amounts.append(len(games_in_interval))
    average_ratings.append(games_in_interval['average'].mean())

# Création de l'histogramme en barres
plt.figure(figsize=(20,6))
plt.subplot(1, 2, 1)
plt.title("Répartition des meilleurs jeux par difficulté")
plt.xlabel("Difficulté")
plt.xticks(rotation=45)
plt.ylabel("Nombre de jeux")

labels = [f"{i}-{i+0.25}" for i in np.arange(1, 4.75, 0.25)]
plt.bar(labels, amounts, linewidth=1.2, width=0.6, color='tomato')
```

```
plt.subplot(1, 2, 2)
plt.bar(labels, average_ratings, linewidth=1.2, width=0.6,
color='teal')
plt.title("Moyenne des meilleurs jeux selon leur difficulté")
plt.xlabel("Difficulté")
plt.xticks(rotation=45)
plt.ylabel("Note moyenne")
plt.ylim(6.5, 8)

plt.show()
```



Comme nous montre le graphique de droite, il y a une tendance à noter plus haut les jeux plus dur, ce qui est **contre-intuitif**. On peut se demander si cette tendance est due au fait que les jeux plus dur sont plus rares et donc plus appréciés. Cependant, la répartition des jeux selon la difficulté montre qu'il y moins de jeux dur notés que de jeux simples notés. Cela veut donc dire qu'il est plus facile pour une note moyenne d'être plus élevée pour un jeu dur que pour un jeu simple **s'il y a moins d'internautes qui ont noté ce jeu**.

Ce qui est intéressant dans ces graphiques, c'est qu'ils montrent que **l'on ne peut pas vraiment établir la note moyenne d'un jeu en fonction de sa difficulté tant qu'assez d'internautes n'ont pas noté ce jeu.** Ce phénomène est visible dans les zones à partir de 3 de difficulté.

Cela veut donc dire que **la difficulté d'un jeu n'est pas un bon indicateur de la note moyenne d'un jeu si nous considérons uniquement les meilleurs jeux.** Qu'en est-il des autres jeux?

---

2) Comparaison avec les jeux de société les moins bien notés

Le but de cette partie est de comparer les jeux de société les moins bien notés avec les meilleurs jeux de société. On va donc mettre en évidences les caractéristiques communes et différences de ces deux sous-populations.

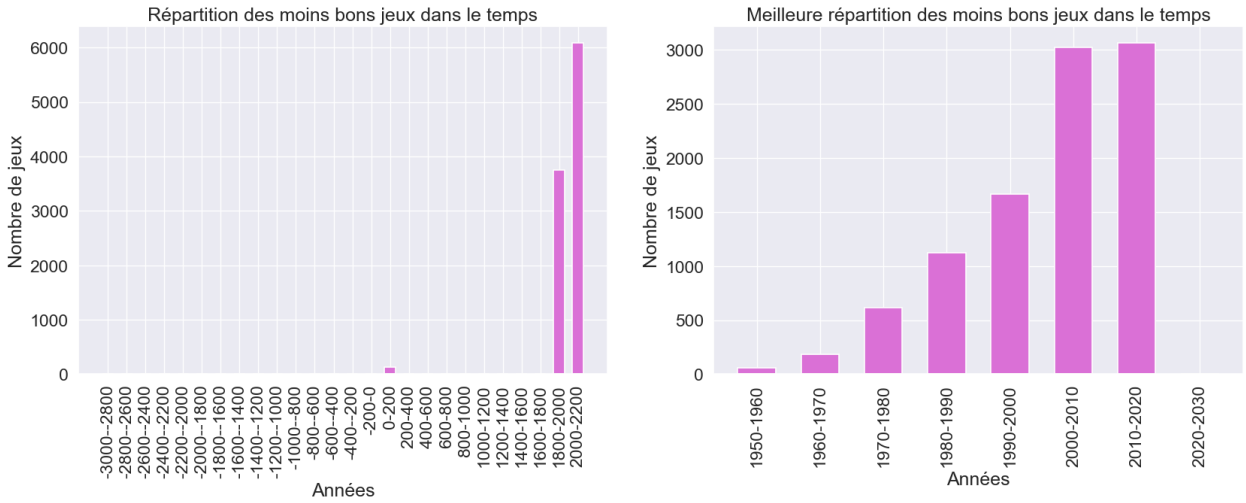
Commençons tout d'abord par créer une nouvelle dataframe contenant les jeux de société les moins bien notés, ainsi que les homologues des dataframe crées précédemment afin de pouvoir les comparer deux à deux:

```
bad_games = df[df['average'] <= median]
bad_years = pd.DataFrame(bad_games['yearpublished'])
bad_mech_cat_games = pd.DataFrame({'name': bad_games['name'],
                                   'boardgamemechanic':
bad_games['boardgamemechanic'],
                                   'boardgamecategory':
bad_games['boardgamecategory'],
                                   'average': bad_games['average'],})
bad_difficulty = bad_games[bad_games['avgweight'] > 0]
```

2-A) Analyse **temporelle**

```
# Intervalles de 10 ans
intervals = [[i, i+200] for i in range(-3000, 2021, 200)]
better_intervals = [[i, i+10] for i in range(1950, 2021, 10)]

# Calcul du nombre de jeux par intervalle
amounts = []
for interval in intervals:
    i, j = interval
    games_in_interval = bad_years[(bad_years['yearpublished'] >= i) &
(bad_years['yearpublished'] < j)]
    amounts.append(len(games_in_interval))

better_amounts = []
for interval in better_intervals:
    i, j = interval
    games_in_interval = bad_years[(bad_years['yearpublished'] >= i) &
(bad_years['yearpublished'] < j)]
    better_amounts.append(len(games_in_interval))

# Création de l'histogramme en barres
plt.figure(figsize=(20,6))
plt.subplot(1, 2, 1)
plt.title("Répartition des moins bons jeux dans le temps")
plt.xlabel("Années")
plt.xticks(rotation=90)
plt.ylabel("Nombre de jeux")
labels = [f"{i}-{i+200}" for i in range(-3000, 2021, 200)]
plt.bar(labels, amounts, linewidth=1.2, width=0.6, color="orchid")

plt.subplot(1, 2, 2)
plt.title("Meilleure répartition des moins bons jeux dans le temps")
plt.xlabel("Années")
plt.xticks(rotation=90)
plt.ylabel("Nombre de jeux")
```

```
labels = [f"{i}-{i+10}" for i in range(1950, 2021, 10)]
plt.bar(labels, better_amounts, linewidth=1.2, width=0.6,
color="orchid")


plt.show()
```



Passons rapidement l'analyse temporelle des jeux les moins biens notés: la répartition est la même que celle des meilleurs jeux de société à la différence près qu'il y a plus de moins bons jeux dans la période 2000-2010. La courbe exponentielle est **moins marquée**.

On remarque également sur le premier graphe qu'il y a quelques jeux dans la période 0-200. Il s'agit des jeux anciens ainsi que **les erreurs de rentrées de données**. Certains jeux ont certainement dû être rentrés sans dates de sortie: **le site a donc rentré la date 0 par défaut**.

2-B) Analyse des **mécaniques / catégories**

```
bad_mechanics = bad_mech_cat_games.explode('boardgamemechanic')
['boardgamemechanic']

# Compter les occurrences de chaque mécaniques
bad_mechanics_counts = bad_mechanics.value_counts()
bad_mechanics_counts.head(10)
```

```
['Roll / Spin and Move']    489
['Hexagon Grid']            371
['Hand Management']         323
['Dice Rolling']            276
['Tile Placement']          225
['Set Collection']          201
['Memory']                  126
['Card Game']               125
['Pattern Recognition']     112
['Pattern Building']        102
Name: boardgamemechanic, dtype: int64
```

```
bad_categories = bad_mech_cat_games.explode('boardgamecategory')
['boardgamecategory']

# Compter les occurrences de chaque mécaniques
bad_categories_counts = bad_categories.value_counts()
bad_categories_counts.head(10)
```

```
['Card Game']                                604
['Abstract Strategy']                        384
['Wargame', 'World War II']                  198
['Dice']                                     140
['Party Game']                               117
['Economic']                                 113
['Childrens Game']                            80
['Party Game', 'Trivia']                      77
['Action / Dexterity', 'Childrens Game']      77
['Science Fiction', 'Wargame']                71
Name: boardgamecategory, dtype: int64
```

De manière générale, les jeux les moins bien notés sont **moins variés** que les meilleurs jeux de société. On peut voir que les jeux les moins bien notés sont **plus concentrés** dans les mêmes catégories que les meilleurs jeux de société. Cependant, on remarque que les meilleurs jeux et moins bons jeux ont des catégories communes comme "Card Game" ou "Hexagon Grid".

On peut noter aussi que les mécaniques de jeu *Memory*, *Pattern Building* et *Pattern Recognition* sont les **plus présentes** dans les jeux les moins bien notés: **ces mécaniques sont peu complexes à mettre en place lors du gamedesign**. Même si la richesse des mécaniques d'un jeu ne définit pas entièrement sa qualité, il faut avouer que ces mécaniques peuvent facilement être utilisées pour **créer facilement des jeux**. Au contraire, les mécaniques *Abstract Strategy* ou *Area Control / Area Influence* (présentes dans les meilleurs jeux de société) **sont plus complexes inventer et implémenter** sur un jeu de plateau car elles requiert plus d'originalité dans le processus créatif.

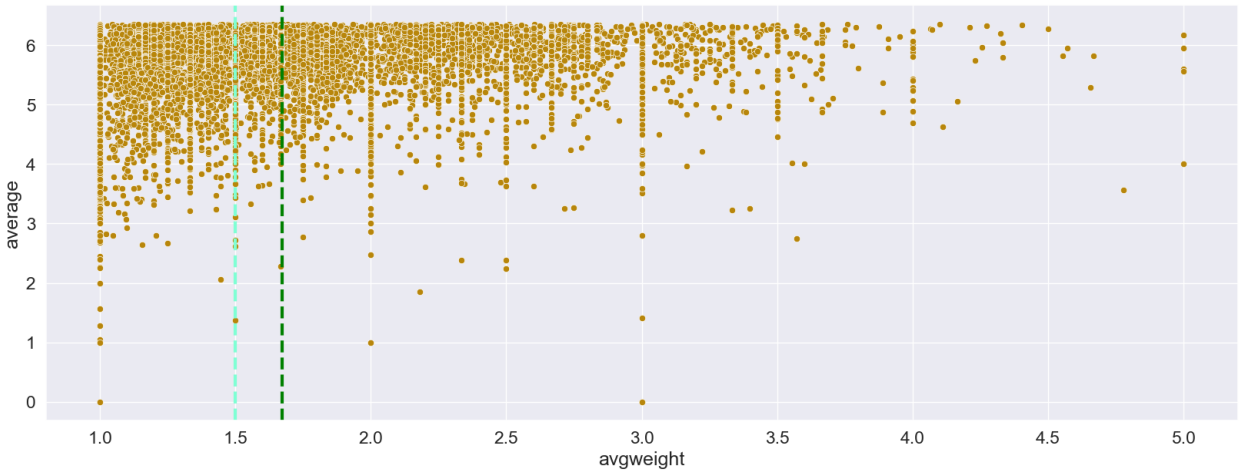
Cela peut expliquer pourquoi ces mécaniques sont présentes dans les jeux les moins bien notés.

2-C) Analyse de la **difficulté**

```
bad_mean_difficulty = bad_difficulty['avgweight'].mean()
bad_median_difficulty = bad_difficulty['avgweight'].median()

print(f"Médiane de la difficulté des meilleurs jeux:
{bad_median_difficulty:.2f}")
print(f"Moyenne de la difficulté des meilleurs jeux:
{bad_mean_difficulty:.2f}")

plt.figure(figsize=(20,7))
sns.scatterplot(data=bad_difficulty, x='avgweight', y='average',
color="darkgoldenrod")
plt.axvline(bad_mean_difficulty, color='green', linestyle='dashed',
linewidth=3)
plt.axvline(bad_median_difficulty, color='aquamarine',
```

```
linestyle='dashed', linewidth=3)
plt.show()
```

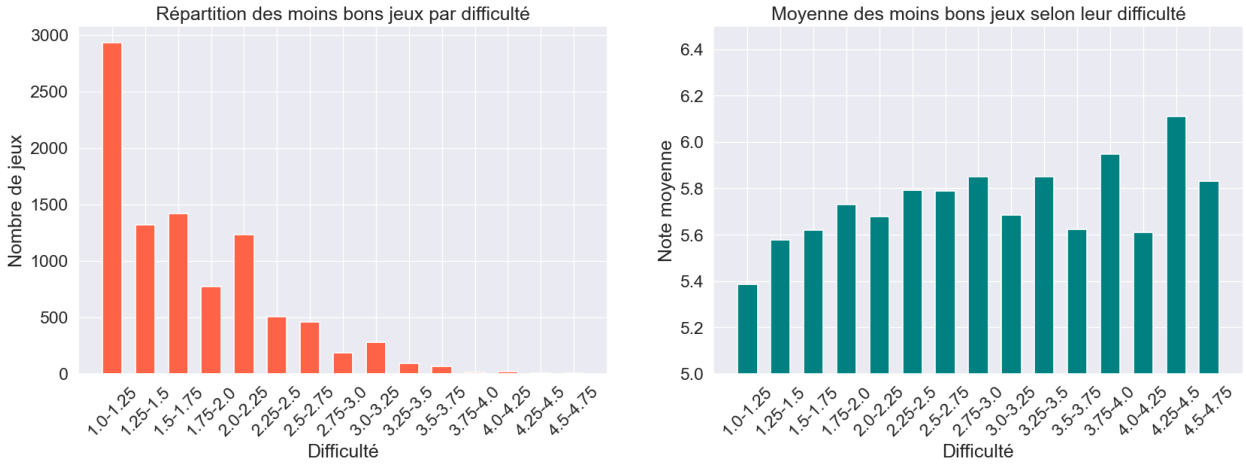
```
Médiane de la difficulté des meilleurs jeux: 1.50
Moyenne de la difficulté des meilleurs jeux: 1.67
```



Cette fois ci, on remarque que chez les jeux les moins biens notés les jeux sont **plus concentrés** dans la difficulté 1 et 2. Cela peut être dû au fait que les jeux les moins bien notés sont **plus simples** que les meilleurs jeux de société. On peut voir que les jeux les moins bien notés sont **plus concentrés** dans les mêmes catégories que les meilleurs jeux de société. Quid de la relation entre la difficulté et la note moyenne?

```
intervals = [[i, i+0.25] for i in np.arange(1, 4.75, 0.25)]

# Calcul du nombre de jeux par intervalle
bad_amounts = []
bad_average_ratings = []
for interval in intervals:
    i, j = interval
    games_in_interval = bad_difficulty[(bad_difficulty['avgweight'] >=
i) & (bad_difficulty['avgweight'] < j)]
    bad_amounts.append(len(games_in_interval))
    bad_average_ratings.append(games_in_interval['average'].mean())

labels = [f"{i}-{i+0.25}" for i in np.arange(1, 4.75, 0.25)]

plt.figure(figsize=(20,6))
plt.subplot(1, 2, 1)
plt.title("Répartition des moins bons jeux par difficulté")
plt.xlabel("Difficulté")
plt.xticks(rotation=45)
plt.ylabel("Nombre de jeux")

plt.bar(labels, bad_amounts, linewidth=1.2, width=0.6, color='tomato')
```

```
plt.subplot(1, 2, 2)
plt.bar(labels, bad_average_ratings, linewidth=1.2, width=0.6,
color='teal')
plt.title("Moyenne des moins bons jeux selon leur difficulté")
plt.xlabel("Difficulté")
plt.xticks(rotation=45)
plt.ylabel("Note moyenne")
plt.ylim(5, 6.5)

plt.show()
```



Contrairement aux meilleurs jeux de société, la répartition des jeux dans les niveaux de difficulté est **plus concentrée dans les difficultés basses**. Cela veut donc dire, que comme précédemment, on ne pourra vraiment établir la note moyenne d'un jeu en fonction de sa difficulté dans les difficultés basses.

Cette fois ci, pour les difficultés en dessous de 2.0-2.25 les notes moyennes sont **plus basses** que pour les meilleurs jeux mais ce n'est pas non plus très démarqué. On comprend bien que les internautes de BGG préfèrent les jeux un peu plus difficiles.

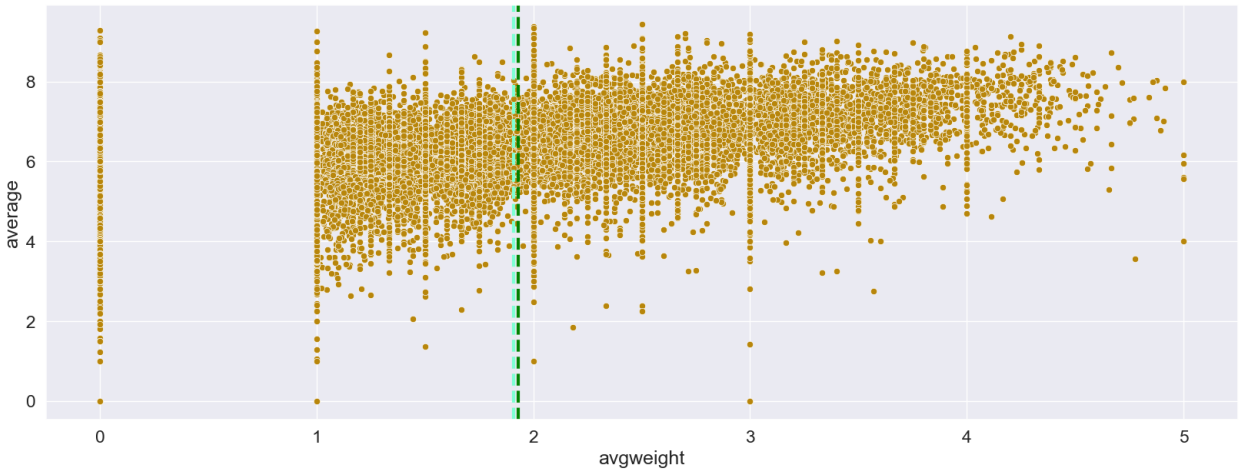
Cette difficulté que nous avons à pouvoir prétendre que les notes moyennes que nous pouvons lire sur le graphe de droite sont pertinentes est lié au fait que plus les jeux deviennent durs, moins de gens les ont noté. Seulement, nous avons coupé en deux nous jeu de données au départ et il est normal que les moins bons jeux ait moins de jeu avec une difficulté de 3 assez notés.

2-C bis) Analyse de la difficulté sur tous les jeux

```
total_mean_difficulty = df['avgweight'].mean()
total_median_difficulty = df['avgweight'].median()

print(f"Médiane de la difficulté des meilleurs jeux:
{bad_median_difficulty:.2f}")
print(f"Moyenne de la difficulté des meilleurs jeux:
{bad_mean_difficulty:.2f}")
```

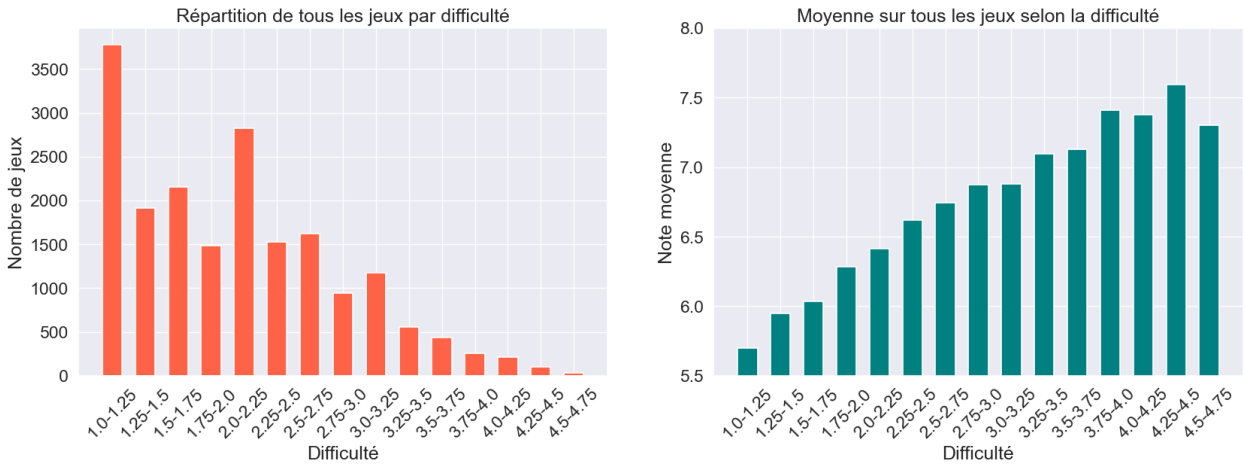
```
plt.figure(figsize=(20,7))
sns.scatterplot(data=df, x='avgweight', y='average',
color="darkgoldenrod")
plt.axvline(total_mean_difficulty, color='green', linestyle='dashed',
linewidth=3)
plt.axvline(total_median_difficulty, color='aquamarine',
linestyle='dashed', linewidth=3)
plt.show()
```

```
Médiane de la difficulté des meilleurs jeux: 1.50
Moyenne de la difficulté des meilleurs jeux: 1.67
```




```
intervals = [[i, i+0.25] for i in np.arange(1, 4.75, 0.25)]

# Calcul du nombre de jeux par intervalle
total_amounts = []
total_average_ratings = []
for interval in intervals:
    i, j = interval
    games_in_interval = df[(df['avgweight'] >= i) & (df['avgweight'] <
j)]
    total_amounts.append(len(games_in_interval))
    total_average_ratings.append(games_in_interval['average'].mean())

labels = [f"{i}-{i+0.25}" for i in np.arange(1, 4.75, 0.25)]

plt.figure(figsize=(20,6))
plt.subplot(1, 2, 1)
plt.title("Répartition de tous les jeux par difficulté")
plt.xlabel("Difficulté")
plt.xticks(rotation=45)
plt.ylabel("Nombre de jeux")

plt.bar(labels, total_amounts, linewidth=1.2, width=0.6,
color='tomato')
```

```
plt.subplot(1, 2, 2)
plt.bar(labels, total_average_ratings, linewidth=1.2, width=0.6,
color='teal')
plt.title("Moyenne sur tous les jeux selon la difficulté")
plt.xlabel("Difficulté")
plt.xticks(rotation=45)
plt.ylabel("Note moyenne")
plt.ylim(5.5, 8)

plt.show()
```



Il s'agit donc ici d'une combinaison des deux analyses précédentes. Nous avons toujours le problème de manque de données pour les difficultés hautes mais nous pouvons maintenant comparer les jeux les moins bien notés avec les meilleurs jeux de société pour des difficulté entre 2 et 3.

Pourquoi avons nous besoin de passer sur le jeu de données entier pour faire analyse? Comme nous l'avons vu, lorsque nous avons fait l'analyse de la difficulté sur les meilleurs jeux de société, nous avons remarqué que la difficulté n'était pas un bon indicateur de la note moyenne d'un jeu en dehors de la plage de difficulté 2-3. Cependant, nous avons remarqué que la difficulté était un bon indicateur de la note moyenne d'un moins bon jeu pour les difficultés basses mais dès que l'on approchait de 3, ce n'était plus un bon indicateur.

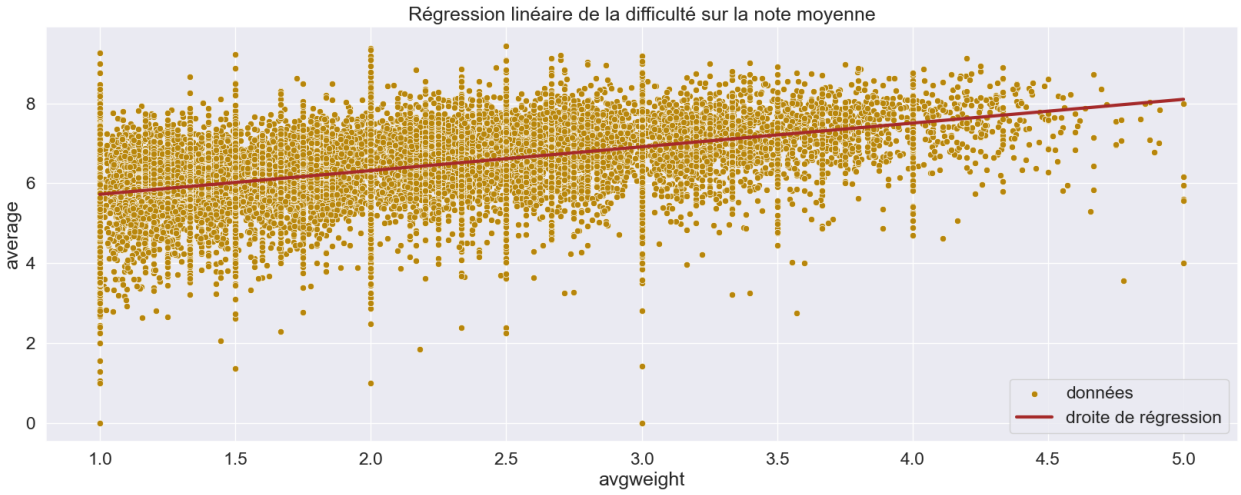
Cela est dû au fait que nous coupons en deux nos jeux de données au départ. Cette fracture sectionne la tendance générale des points à augmenter.

2-D) Allure générale de la difficulté selon la note moyenne

Faisons une regression linéaire sur le nuage de point représentant la note moyenne en fonction de la difficulté.

```
linear_regressor = LinearRegression(copy_X=True, fit_intercept=True)

df0 = df[df['avgweight'] != 0] # on supprime les jeux avec une
difficulté de 0 du jeu de données intial
```

```
df = pd.DataFrame(df0, columns=['avgweight', 'average'])
df2 = (df0 - df0.mean(axis = 0))/df.std(axis = 0)
X = df['avgweight'].to_numpy()
X = X.reshape(-1, 1) # étape nécessaire pour la mise en forme des
données attendue par LinearRegression
Y = df['average'].to_numpy()
Y = Y.reshape(-1, 1)

result = linear_regressor.fit(X, Y)
print("Estimation : a = %.1f" %result.coef_[0] + " et b = %.1f"
%result.intercept_)
print(f"Corrélation entre la difficulté et la note moyenne :
{df['avgweight'].corr(df['average']):.2f}")

plt.figure(figsize=(20,7))
plt.title("Régression linéaire de la difficulté sur la note moyenne")
sns.scatterplot(x="avgweight", y='average',legend="full", data=df,
color="darkgoldenrod")
plt.plot([np.min(X), np.max(X)], [result.intercept_ +
np.min(X)*result.coef_[0], result.intercept_ +
np.max(X)*result.coef_[0]], 'brown', linewidth=3)
plt.legend([ 'données' ,'droite de régression'])
plt.show()

Estimation : a = 0.6 et b = 5.1
Corrélation entre la difficulté et la note moyenne : 0.51
```



Nous avons à présent une représentation graphique de la tendance qu'il existe entre la difficulté et la note moyenne des jeux de société. On calcule également le coefficient de corrélation, qui est de 0.51, ce qui n'est pas très précis. De plus, le coefficient directeur de la droite de régression est de 0.6. On peut remarquer une tendance générale mais elle n'est pas très marquée.

> On pourrait justement se demander à partir de quelle valeur de a la droite de régression la tendance à moins bien noter les jeux faciles devient significative.

Lorsque nous coupions notre jeu de données en deux, nous perdions une partie de la tendance générale: **la courbe rouge était coupée en deux**. C'est pourquoi il était dur pour nous de prétendre que la difficulté était un bon indicateur de la note moyenne d'un jeu.

La note moyenne des jeux n'est pas la seule caractéristique qui peut suivre une tendance. En effet, ous avons pu voir lors de l'analyse temporelle de la sous-population des meilleurs jeux (**1-A)**) que la répartition des meilleurs jeux dans le temps était exponentielle. Peut-être pourrions nous utiliser un modèle de prédiction pour prédire combien de jeux de société sortiront dans les années qui suivent.

3) Modèle de **prédiction**

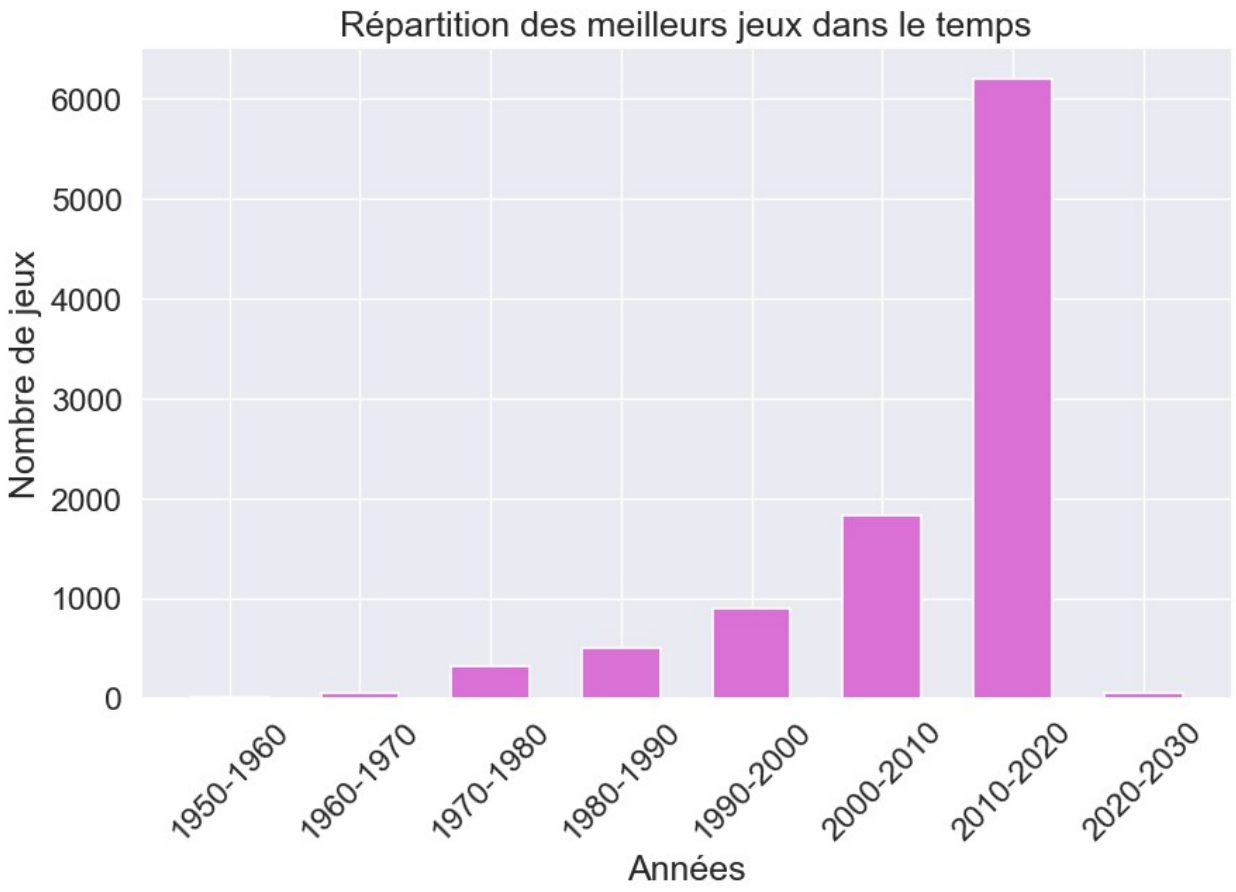
Reprenons tout d'abord les données de la partie **1-A)**.

On va chercher à **prédire le nombre de jeux de société qui sortiront dans les années à venir**. Pour cela, nous allons utiliser un modèle de **régression exponentielle**. Nous allons utiliser la fonction **`curve_fit`** de `scipy.optimize` pour trouver les coefficients de la courbe de régression.

Nous devons également penser à **ne pas prendre en compte la valeur de la décennie 2020-2030**. En effet, nous n'avons pas encore assez de données pour cette décennie.

Nous allons ensuite calculer les valeurs du prochains point théorique selon le modèle de régression.

```
plt.figure(figsize=(10,6))
plt.title("Répartition des meilleurs jeux dans le temps")
plt.xlabel("Années")
plt.xticks(rotation=45)
plt.ylabel("Nombre de jeux")

plt.bar(time_labels, time_amounts, linewidth=1.2, width=0.6,
color="orchid")
plt.show()
```



On peut se demander si le modèle de régression exponentielle est le meilleur modèle pour prédire le nombre de jeux de société qui sortiront dans les années à venir.

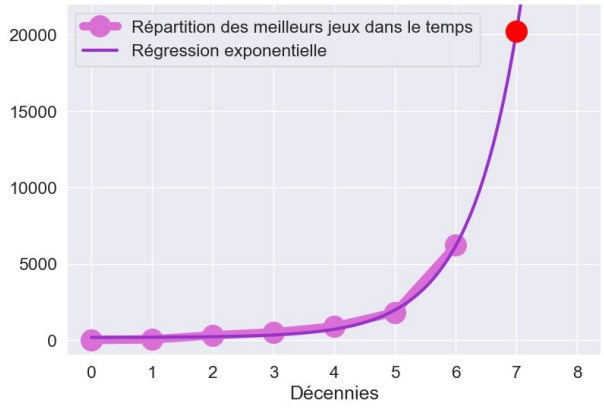
Pour cela, deux approches:

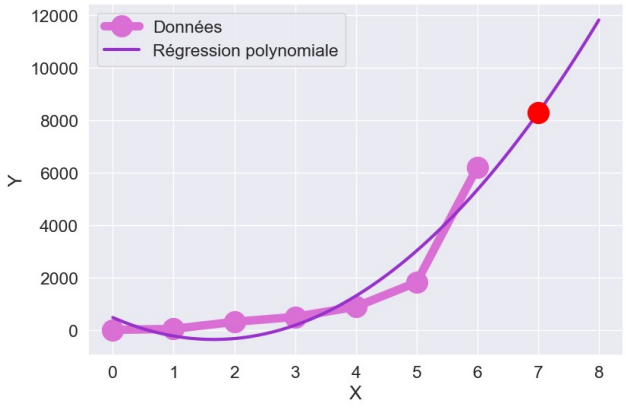
- Calculer **l'erreur quadratique moyenne** entre les différents modèles de regressions
- Calculer **la différence de résultat** entre les valeurs prédites et les valeurs réelles sur la décennie 2010-2020 lorsque l'on ne prend pas en compte cette dernière dans les différents modèles de régression.

```
x_data = [i for i in range(7)]
y_data = time_amounts[:7]

def exponential_func(x, a, b, c):
    return a * np.exp(b * x) + c

exp_popt, exp_pcov = curve_fit(exponential_func, x_data, y_data)

# Tracé de la courbe exponentielle ajustée sur vos données
exp_x_fit = np.linspace(0, 8, 100)
exp_y_fit = exponential_func(exp_x_fit, *exp_popt)

plt.figure(figsize=(20,6))
plt.subplot(1, 2, 1)
```

```
plt.plot(x_data, y_data, label='Répartition des meilleurs jeux dans le
temps', color='orchid', marker='o',markersize=20, linewidth=8)
plt.plot(exp_x_fit, exp_y_fit, label='Régression exponentielle',
color='darkorchid', linewidth=3)
plt.legend()
plt.ylim(-1000, 22000)
plt.xlabel("Décennies")

# Prédiction du nombre de jeux sortis dans la prochaine décennie
n_years = 7
x_pred = np.linspace(0, n_years, n_years+1)
exp_y_pred = exponential_func(x_pred, *exp_popt)
print("Prédiction pour la prochaine décennie :", exp_y_pred[-1])
plt.plot(7, exp_y_pred[-1], marker='o', markersize=20, color='red',
linewidth=2)

plt.subplot(1, 2, 2)
coeffs = np.polyfit(x_data, y_data, deg=2)
poly_x_fit = np.linspace(0, 8, 100)
poly_y_fit = np.polyval(coeffs, poly_x_fit)


poly_func = np.poly1d(coeffs)
poly_y_pred = poly_func(x_pred)

# Affichage de la prédiction pour la régression polynomiale
print("Prédiction pour la prochaine décennie (régression
polynomiale) :", poly_y_pred[-1])

plt.plot(x_data, y_data, label='Données', color='orchid',
marker='o',markersize=20, linewidth=8)
plt.plot(poly_x_fit, poly_y_fit, label='Régression polynomiale',
color='darkorchid', linewidth=3)
plt.legend()
plt.xlabel('X')
plt.ylabel('Y')
plt.plot(7, poly_y_pred[-1], marker='o', markersize=20, color='red',
linewidth=2)
plt.show()
```

```
Prédiction pour la prochaine décennie : 20239.66657320063
Prédiction pour la prochaine décennie (régression polynomiale) :
8280.57142857144
```






```
poly_mse = 1/len(y_data) * np.sum((y_data[:6] - poly_y_pred[:6])**2)
exp_mse = 1/len(y_data) * np.sum((y_data[:6] - exp_y_pred[:6])**2)

print(f"MSE de la régression polynomiale: {poly_mse:.2f}")
print(f"MSE de la régression exponentielle: {exp_mse:.2f}")

print(f"Amélioration de la régression exponentielle par rapport à la
régression polynomiale: {100*(poly_mse - exp_mse)/poly_mse:.2f}%")
```

```
MSE de la régression polynomiale: 344685.31
MSE de la régression exponentielle: 20147.83
Amélioration de la régression exponentielle par rapport à la
régression polynomiale: 94.15%
```

Après avoir calculé l'erreur quadratique moyenne, on remarque que le modèle de régression exponentielle est le meilleur modèle pour prédire le nombre de jeux de société qui sortiront dans les années à venir. On a un modèle qui est **94% plus précis** que le modèle de régression linéaire.

Il n'y a donc aucun doute et même pas besoin de calculer les erreurs de prédiction pour la décennie 2010-2020, on peut même se convaincre graphiquement que le modèle polynomial est moins précis.

On obtient donc grâce au modèle de prédiction une valeur de 20 239 jeux qui sortiront sur la décennie 2020-2030.

```
augmentation = (exp_y_pred[-1] - exp_y_pred[-2])/exp_y_pred[-2]
valeur_augm = exp_y_pred[-1] - exp_y_pred[-2]
print(f"Augmentation du nombre de jeux sortis dans la prochaine
décennie: {100*augmentation:.2f}%")
print(f"Valeur de l'augmentation du nombre de jeux sortis dans la
prochaine décennie: {valeur_augm:.2f}")
```

```
Augmentation du nombre de jeux sortis dans la prochaine décennie:
227.30%
Valeur de l'augmentation du nombre de jeux sortis dans la prochaine
décennie: 14055.93
```

Cela représente une augmentation de 14 000 jeux par rapport à la décennie 2010-2020.

> *De quoi s'amuser pendant un moment!*

On peut **se demander si cette augmenation est réalisable**. Le problème est que nous n'avons pas assez de données sur d'assez longues période pour pouvoir faire une analyse plus poussée du modèle de prédiction à choisir. Au final un modèle de regression qui fait un compromis entre la régression exponentielle et polynomiale serait le modèle le plus optimal et plus représentatif du comportement humain.

Quoi qu'il en soit, seul l'avenir nous le dira! 🚀

IV Conclusions du projet ISSD

Dans le cadre de ce projet nous avons pu étudier les caractéristiques communes des jeux de société en fonction de leur note moyenne auprès des internautes de Board Game Geek. Les observations sont les suivantes:

- Les jeux de société notés sur le site BGG sont **essentiellement des jeux récents**. Peu importe leur note moyenne, la plupart des jeux de société sont sortis après 1950 et suivent **une tendance exponentielle**. C'est à dire que plus les années passent et plus il y a de jeux de société notés sur le site BGG. On a par la même occasion pu faire une prédiction sur le nombre de bons jeux de société qui allait sortir l'année prochaine.

- Les mécaniques et catégories de jeux préférées sont en générale des mécanique **complexes** et des catégories de jeux plus **originales** que ceux des jeux les moins populaires. On a pu trouver des corrélations entre mécaniques et catégories, notamment avec le fait que les jeux de guerre sont plus intéressants d'un point de vue stratégique lorsqu'ils sont joués sur des plateaux hexagonaux, ou bien le fait que les jeux de cartes sont en général plus limités en terme de possibilités et donc moins palpitants.

- Il existe une tendance générale à **noter les jeux de société plus difficiles plus haut**. Cependant, cette tendance n'est pas significative. En effet, lorsque nous avons coupé notre jeu de données en deux, nous avons perdu une partie de la tendance générale. C'est pourquoi il était dur pour nous de prétendre que la difficulté était un bon indicateur de la note moyenne d'un jeu. Cependant, même en faisant l'analyse sur la totalité du jeu de données on se rend compte qu'il n'y a pas vraiment de corrélation directe entre la difficulté et la note moyenne.

Pour aller plus loin

On s'est rendu compte que la difficulté n'était pas un bon indicateur de la note moyenne d'un jeu. Cependant, **d'autres caractéristiques inexploitées dans ce projet pourrait servir** à définir une échelle de difficulté: le temps maximum et minimum de jeu, le nombre de joueurs minimum et maximum...

Une manière de définir la complexité grâce aux autres caractéristiques des jeux de société serait de **construire un modèle de machine learning** pour prédire la difficulté d'un jeu de société donné en fonction de ses caractéristiques grâce à la bibliothèque scikit-learn. Cette bibliothèque permet de construire des modèles de machine learning de manière très simple et rapide. Lors de son apprentissage, **le modèle de ML classifiera les jeux de société en groupe de difficulté**; groupes définis par la relation **difficulté <-> `min/maxplayers&min/maxplaytime`**; et pourra ainsi prédire la difficulté d'un jeu de société en fonction de ses caractéristiques.

Remarques personnelles:

- J'ai remarqué que **j'aurai dû consacré plus de temps au travail** de prétraitement car je me suis rendu compte à plusieurs de moments qu'il y a avait des problèmes à cause de valeurs incohérentes dans certaines colonnes. Je me suis rendu compte tard que cela provenait du fait que sil ny avait pas de valeurs manquantes cetait parce que par défaut si le site n'a aucune valeur de rentrée il prend la valeur 0. Mais j'imagine que cela fait aussi parti du processus de se rendre compte en plein travail qu'il y a des soucis.
- J'ai appris qu'il fallait faire attention à **comment on modifiait son jeu de données** au fur et a mesure que le notebook jupyter avance. Parfois on se retrouve avec des résultats incohérents et qui nous font parfois penser que nous nous sommes trompés alors qu'en réalité le jeu de données a été modifié quelques lignes plus haut à cause d'un effet de bord.
- A plein d'instant j'ai dû restreindre ma manière d'aborder les problèmes parce que **je n'avais pas assez d'expérience avec les bibliothèques *pandas* en *seaborn***. Même avec l'aide de la documentation officielle des librairies et de *Chat GPT* je ne savais pas bien quelle représentation utiliser et comment faire en sorte que ces représentations répondent de manière efficace à mes questions.