Codebook

1. Dépendances

Liste des bibliothèques Python et versions spécifiques nécessaires à l'exécution du code :

- requests
- re
- os
- random
- BeautifulSoup **from** bs4
- numpy
- matplotlib
- pandas

2. Étapes

a. Configuration Initiale :

- **Importation des bibliothèques nécessaires.**
    - **import** requests
    - **import** re
    - **import** os
    - **import** random
    - **from** bs4 **import** BeautifulSoup
    - **import** numpy **as** np
    - **import** matplotlib.pyplot **as** plt
    - **import** pandas **as** pd

- **Configuration des paramètres de base :**
    - Headers = {"User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0"}

- base_url = 'https://reelgood.com/'

b. Fonctions de Support :

- **fetch_url: Fonction pour récupérer les URLs avec une gestion des tentatives.**
    - J'ai mis une fonction fetch_url pour que chaque fois je veux connecter sur le lien je n'utilise le Try

```
def fetch_url(url, headers, max_attempts=10):
    for attempt in range(max_attempts):
        timeout = random.randint(1, 10)
        try:
            response = requests.get(url, headers=headers, timeout=timeout)
            response.raise_for_status()  # Raise en exception pour les HTTP erreurs
            print(f'Successfully fetched {url}')
            return response
        except requests.RequestException as e:
            print(f'Attempt {attempt + 1} failed for {url} with timeout {timeout}: {e}')
    return None
```

- **extract_categories: Fonction pour extraire les URLs des catégories à partir de la page d'accueil.**
    - A chaque fois je connecte sur les liens j'utilise la pattern_catégories pour récupérer les trois types dans reelgood

```
response = fetch_url(base_url, headers)
if response:
    contenu = response.text
    pattern_categories = '<a class="css-7s3mks ewkc9rp0" aria-expanded="false" href="(.*?)">'
    categories_urls = re.findall(pattern_categories, contenu)
```

```
“Block de code”
else:
    print("Failed to fetch the base URL")
```

c. Traitement des Catégories :

- **process_category: A chaque fois je récupère le nom de type dans site je crée un nouveau dossier qui porte son nom.**

```
for category_url in categories_urls:
        category_url_full = f"https://reelgood.com{category_url}"
        category_name = category_url.split('/')[-1]
        folder = f"Pages/{category_name}"
        os.makedirs(folder, exist_ok=True)
```

d. Traitement des Sous-catégories et Sources :

- **process_subcategories_and_sources: Pour traiter les sous-catégories et les sources spécifiques à chaque catégorie.**
    - Je utilise une nouvelle pattern qui s’appelle pattern_subcategories mais cela fonctionne pour les liens films et séries dans chaqu’un il ya trois type source, List et Genre Mais pour new non car quand j’entre dans new je trouve trois type Coming, new et leaving alors quand je connecte direct je trouve que je dans new/new pour ça j’ai créé une subcatgories ['new', 'coming', 'leaving'] pour category/name =’new’.
    - Je me connecte sur les liens dans chaque sous categorie par exemple le doctor-who-2005 dans sous-catégorie action et aventure dans le type source dans la categorie film.

      ```
      category_response = fetch_url(category_url_full, headers, max_attempts=10)
              if category_response:
                  category_content = category_response.text

                  if category_name == 'new':
```

```
                subcategories = ['new', 'coming', 'leaving']
                for subcategory in subcategories:
                    subcategory_url_full =
f"https://reelgood.com/{subcategory}"
                    subcategory_folder =
f"{folder}/{subcategory.capitalize()}"
                    os.makedirs(subcategory_folder, exist_ok=True)

                    subcategory_response =
fetch_url(subcategory_url_full, headers, max_attempts=10)
                    if subcategory_response:
                        subcategory_content =
subcategory_response.text

                        # Extraire streaming sources (e.g.,
Netflix, Amazon)
                        pattern_sources =
r'href="/{}/(.*?)"'.format(subcategory)
                        sources = re.findall(pattern_sources,
subcategory_content)

                        for source in sources:
                            source_folder =
os.path.join(subcategory_folder, source.capitalize())
                            os.makedirs(source_folder,
exist_ok=True)

                            source_url_full =
f"https://reelgood.com/{subcategory}/{source}"
                            source_response =
fetch_url(source_url_full, headers, max_attempts=10)
                            if source_response:
                                source_content =
source_response.text

                                # Extraire films dans chaque
subcategories
                                pattern_links = r'"content-
listing-link" href="(.*?)" class="css-0 ewkc9rp0"'
                                links = re.findall(pattern_links,
source_content)

                                for link in links:
                                    link_full =
f"https://reelgood.com{link}"
```

```
                                    content_response =
fetch_url(link_full, headers, max_attempts=10)
                                    if content_response:
                                        content_link =
content_response.text
                                        filename =
os.path.join(source_folder, f"{link.split('/')[-1]}.html")
                                        with open(filename, 'w',
encoding='utf8') as f:
                                            f.write(content_link)
                            renamed_folder = f"Pages/movies & tv
shows"
            else:
                # Extraire les subcategories dynamiquement
                pattern_subcategories =
f'href="/{category_name}/(.*?)"'
                subcategories = re.findall(pattern_subcategories,
category_content)

                for subcategory in subcategories:
                    subcategory_folder = os.path.join(folder,
subcategory.capitalize())
                    os.makedirs(subcategory_folder, exist_ok=True)

                    subcategory_url_full =
f"{category_url_full}/{subcategory}"
                    subcategory_response =
fetch_url(subcategory_url_full, headers, max_attempts=10)
                    if subcategory_response:

                        subcategory_content =
subcategory_response.text

                        '''
                        Extraire les liens dans chaque sous
categorie par exemple le doctor-who-2005 dans sous catégorie
action et aventure
                        dans la type source dans la categorie film
                        '''
                        pattern_links = r'"content-listing-link"
href="(.*?)" class="css-0 ewkc9rp0"'
                        links = re.findall(pattern_links,
subcategory_content)

                        for link in links:
                            '''
                            j'ai conlure que les liens des films
```

```
            ou series ont /show/ ou /movies/ avant le nom
                                et non /tv/action-and-
            aventure/nom_film pour ça j'ai mis link coller a
            https://reelgood.com
                                '''
                                link_full =
            f"https://reelgood.com{link}"
                                content_response =
            fetch_url(link_full, headers, max_attempts=10)
                                if content_response:
                                    content_link =
            content_response.text
                                    filename =
            os.path.join(subcategory_folder, f"{link.split('/')[-1]}.html")
                                    with open(filename, 'w',
            encoding='utf8') as f:
                                        f.write(content_link)
            ```
```

### e. Extraction d'Informations

- **Extraire les informations avec BeautifulSoup et regex :**

  ```
def extract_info_from_html(file_path):
    with open(file_path, 'r', encoding='utf8') as file:
        content = file.read()

    soup = BeautifulSoup(content, 'html.parser')

    # Extraire les informations avec BeautifulSoup et regex
    title = soup.find('h1', class_='css-hiz1q1 e3s42hj0').text if
soup.find('h1', class_='css-hiz1q1 e3s42hj0') else None
    year = re.search(r'<div class="e1dskkhw7 css-1i5l3f2
e83cah30">((.*?))</div>', content)
    reelgood_rating = re.search(r'<div><span class="css-1ycvxny
ey9061w1">(.{1,2})', content)
    imdb_rating = re.search(r'</style><span class="css-1ycvxny
ey9061w1">(.*?)', content)
    duration = re.search(r'<span duration=".*?" class="css-1r35rcf
e1dskkhw11">(.*?)', content)
    genres = re.findall(r'href="/movies/genre/(.*?)"', content)
    équipages = re.findall(r'<p class="css-si6acb
eytn0nr3">(.{1,20})</p></div><div spacing="16px" class="css-zmol8d
eytn0nr5"></div>', content)
    origin = re.search(r'href="/origin/(.*?)"', content)
```
```

```
    availability = re.findall(r'<span title="(.*?)" class=', content)
```

- **Après j'ai mis les variables dans un dictionnaire *avec les données extraites***

```
    # Créer un dictionnaire avec les données extraites
    data = {
        'Title': title,
        'Year': year.group(1) if year else None,
        'Reelgood Rating': reelgood_rating.group(1) if reelgood_rating else None,
        'IMDB Rating': imdb_rating.group(1) if imdb_rating else None,
        'Duration': duration.group(1) if duration else None,
        'Genres': ', '.join(genres),
        'Equipages': ', '.join(équipages),
        'Origin': origin.group(1) if origin else None,
        'Availability': list(set(availability))  # Utiliser set pour supprimer les duplications
    }

    return data
```

f. Extraire les données à partir des fichiers HTML

- **J'ai utilisé la fonction précédente pour mettre toutes les données dans une data frame**

```
data_list = []
for root, dirs, files in os.walk('Pages'):
    for file in files:
        if file.endswith('.html'):
            file_path = os.path.join(root, file)
            data = extract_info_from_html(file_path)
            data_list.append(data)

df = pd.DataFrame(data_list)
```

g. Analyse et Visualisation :

- **Analyser les données manquantes**

```
# Vérifier les colonnes vides
colonnes_vides = df.isna().sum()

# Calculer le pourcentage de colonnes vides par rapport aux colonnes non vides
pourcentage_colonnes_vides = (colonnes_vides / len(df)) * 100
pourcentage_colonnes_non_vides = 100 - pourcentage_colonnes_vides

# Créer un graphique à barres
plt.figure(figsize=(10, 6))
pourcentage_colonnes_vides.plot(kind='bar', color='red', alpha=0.7, label='Colonnes vides')
pourcentage_colonnes_non_vides.plot(kind='bar', color='blue', alpha=0.7, label='Colonnes non vides')
plt.title('Pourcentage de colonnes vides par rapport aux colonnes non vides')
plt.xlabel('Colonnes')
plt.ylabel('Pourcentage')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Créer un graphique circulaire
plt.figure(figsize=(8, 8))
plt.pie([pourcentage_colonnes_vides.sum(), pourcentage_colonnes_non_vides.sum()],
        labels=['Colonnes vides', 'Colonnes non vides'],
        colors=['red', 'blue'],
        autopct='%1.1f%%',
        startangle=140)
plt.title('Pourcentage de colonnes vides par rapport aux colonnes non vides')
plt.axis('equal')
plt.tight_layout()
plt.show()

df.isnull().sum()
```

- **Nettoyages des données**

```
df_clean = df.dropna()

df_clean
```

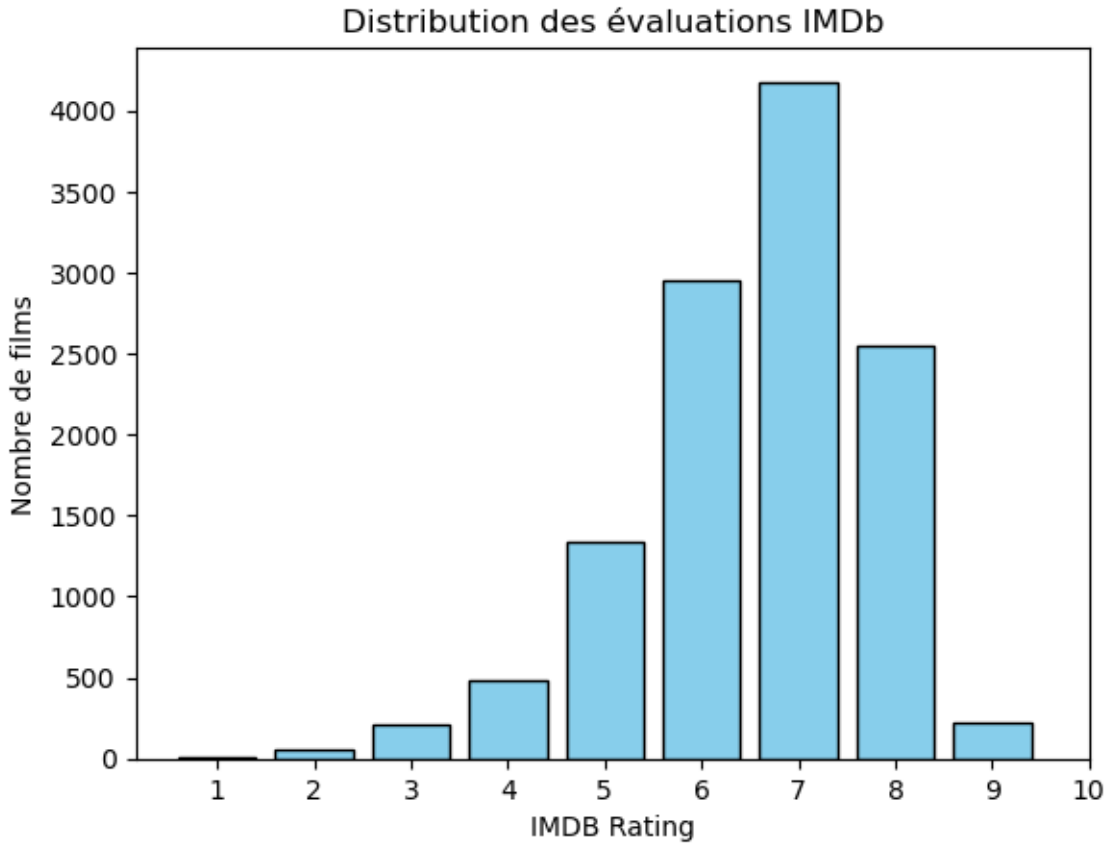
- **Analyser les données**

```

```
# Filtrer les lignes avec des valeurs non numériques dans la colonne 'IMDB Rating'
df_clean = df[pd.to_numeric(df['IMDB Rating'], errors='coerce').notna()]

Convertir les données de la colonne 'IMDB Rating' en nombres (float)
df_clean.loc[:, 'IMDB Rating'] = df_clean['IMDB Rating'].astype(float)

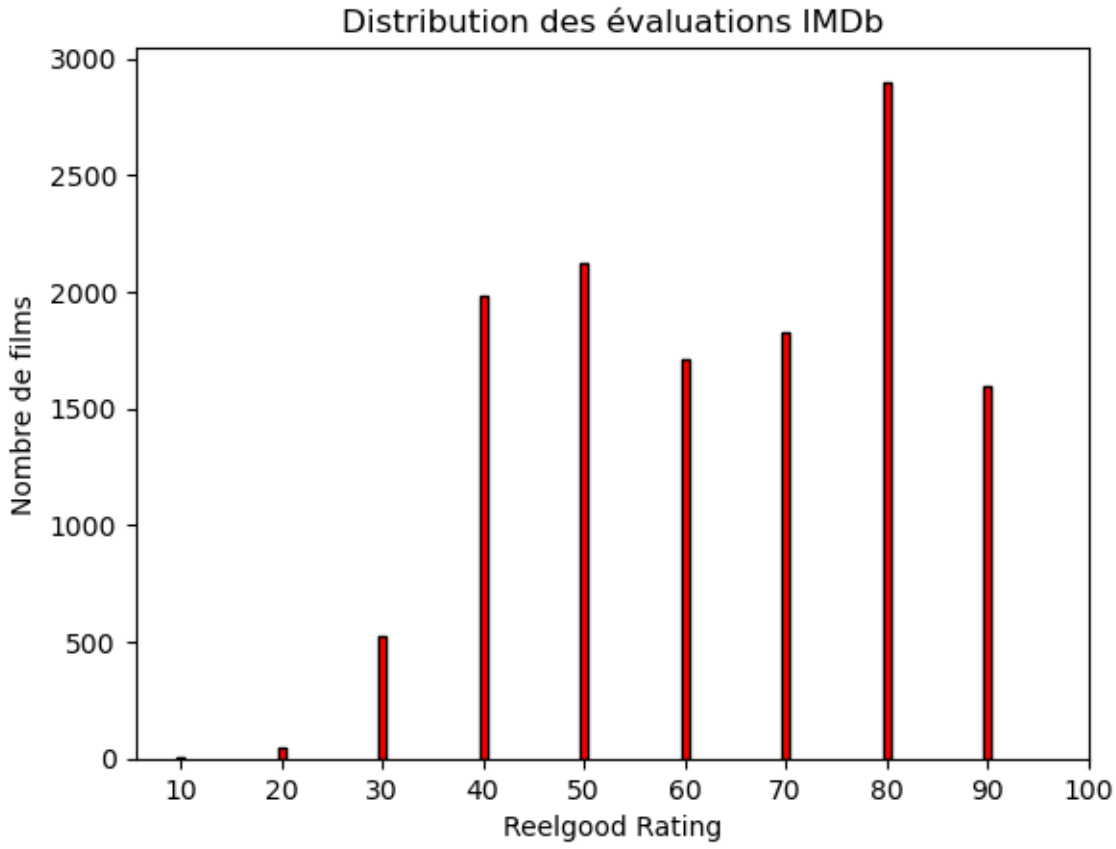
# Définir les intervalles personnalisés pour l'évaluation IMDb
intervals_x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Calculer l'histogramme avec les intervalles personnalisés
hist, bins = np.histogram(df_clean['IMDB Rating'], bins=intervals_x)

# Tracer l'histogramme
plt.bar(bins[:-1], hist, color='skyblue', edgecolor='black')
plt.xlabel('IMDB Rating')
plt.ylabel('Nombre de films')
plt.title('Distribution des évaluations IMDb')
plt.xticks(intervals_x) # Utiliser les intervalles comme marqueurs sur l'axe des x
plt.show()
```




```
# Filtrer les lignes avec des valeurs non numériques dans la colonne 'Reelgood Rating'
df_clean = df[pd.to_numeric(df['Reelgood Rating'], errors='coerce').notna()]

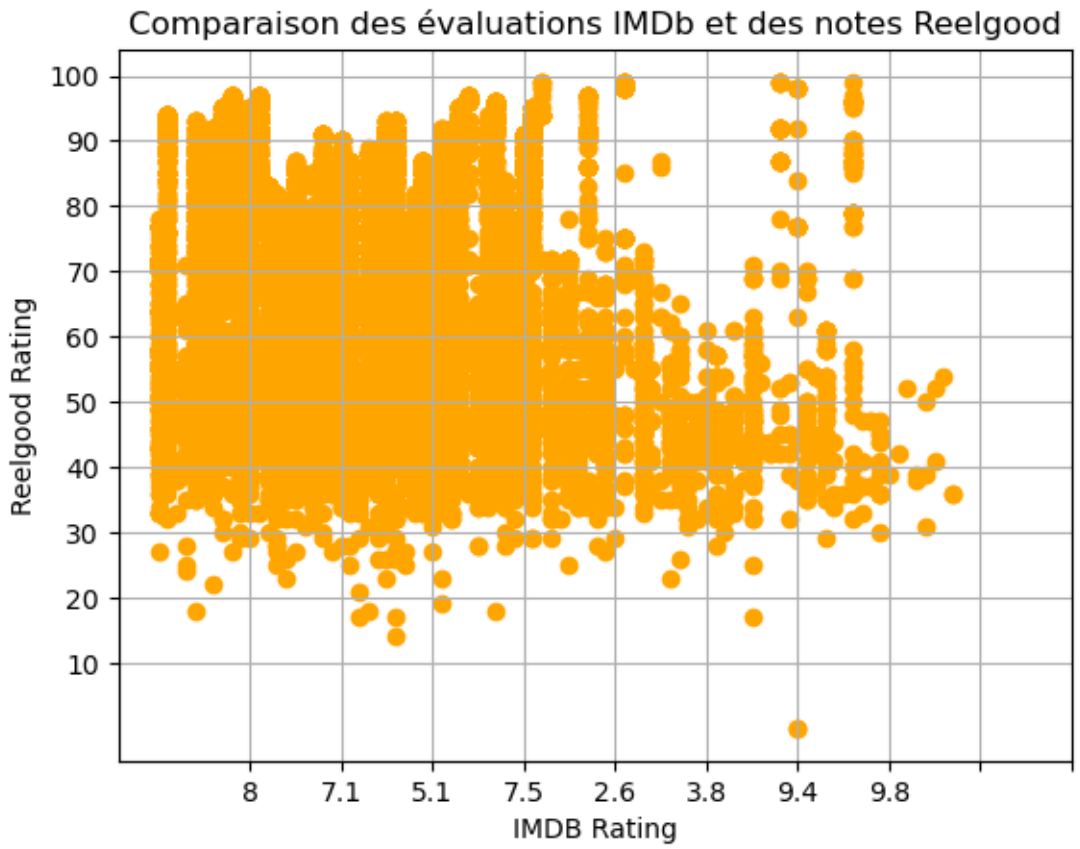
Convertir les données de la colonne 'Reelgood Rating' en nombres (float)
```
```

```
df_clean.loc[:, 'Reelgood Rating'] = df_clean['Reelgood Rating'].astype(float)

# Définir les intervalles personnalisés pour l'évaluation Reelgood
intervals_y = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# Calculer l'histogramme avec les intervalles personnalisés
hist, bins = np.histogram(df_clean['Reelgood Rating'], bins=intervals_y)

# Tracer l'histogramme
plt.bar(bins[:-1], hist, color='red', edgecolor='black')
plt.xlabel('Reelgood Rating')
plt.ylabel('Nombre de films')
plt.title('Distribution des évaluations IMDb')
plt.xticks(intervals_y)  # Utiliser les intervalles comme marqueurs sur l'axe des y
plt.show()
```

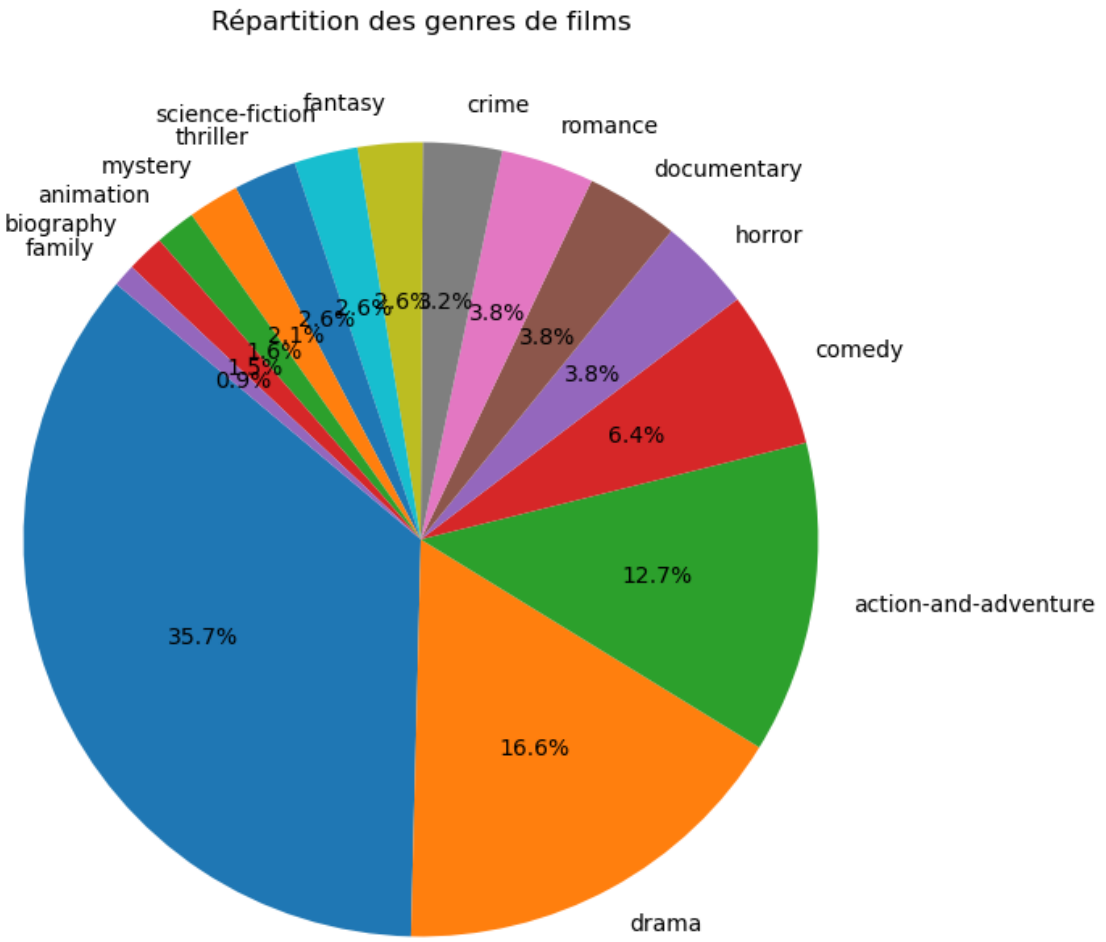



```
# Définir les intervalles personnalisés pour les axes x et y
intervals_x = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
intervals_y = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# Créer un nuage de points pour comparer les évaluations IMDb et les notes Reelgood
plt.scatter(df_clean['IMDB Rating'], df_clean['Reelgood Rating'], color='orange')
plt.xlabel('IMDB Rating')
plt.ylabel('Reelgood Rating')
plt.title('Comparaison des évaluations IMDb et des notes Reelgood')
plt.xticks(intervals_x)
plt.yticks(intervals_y)
```

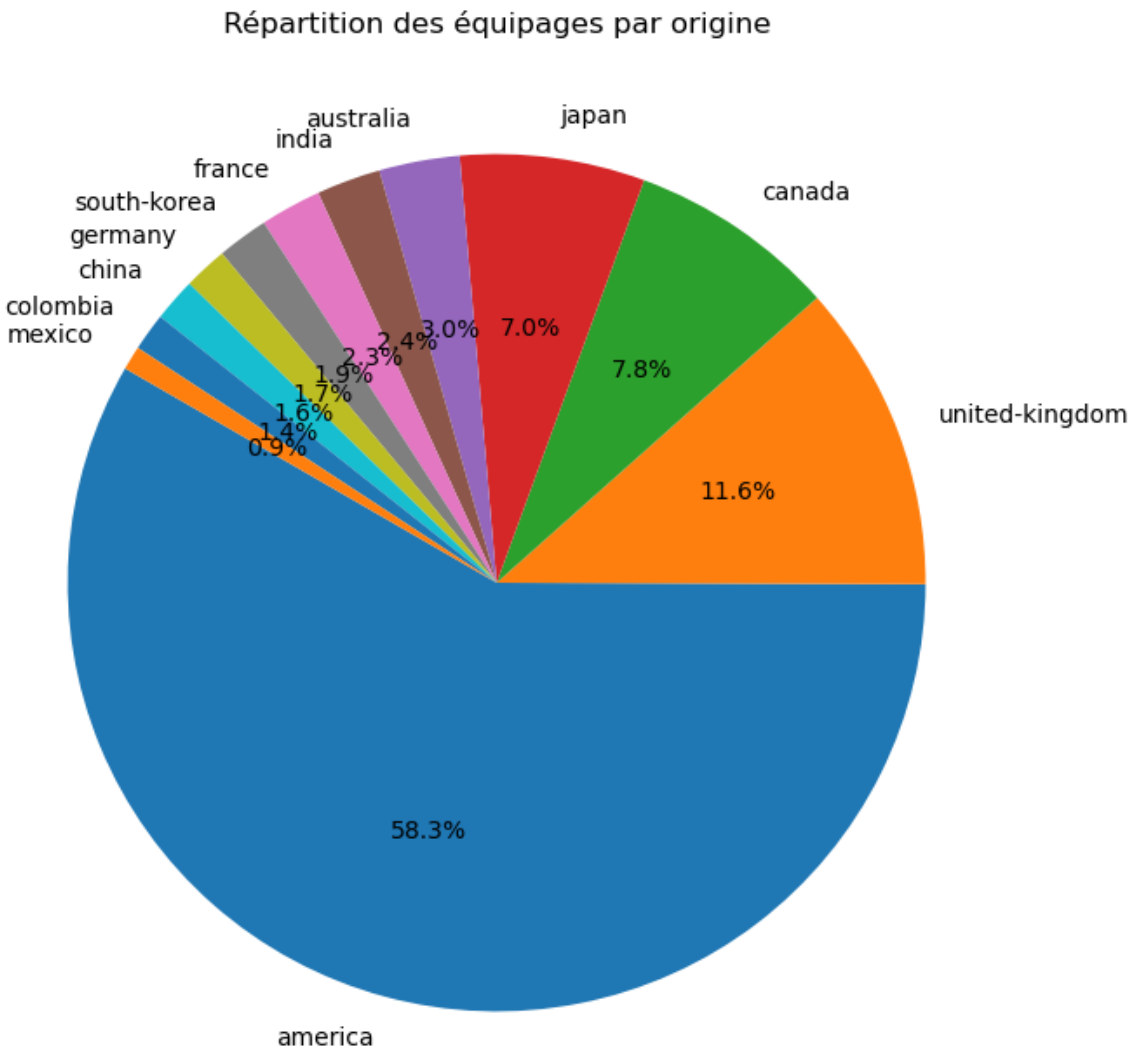
```
plt.grid(True)
plt.show()
```



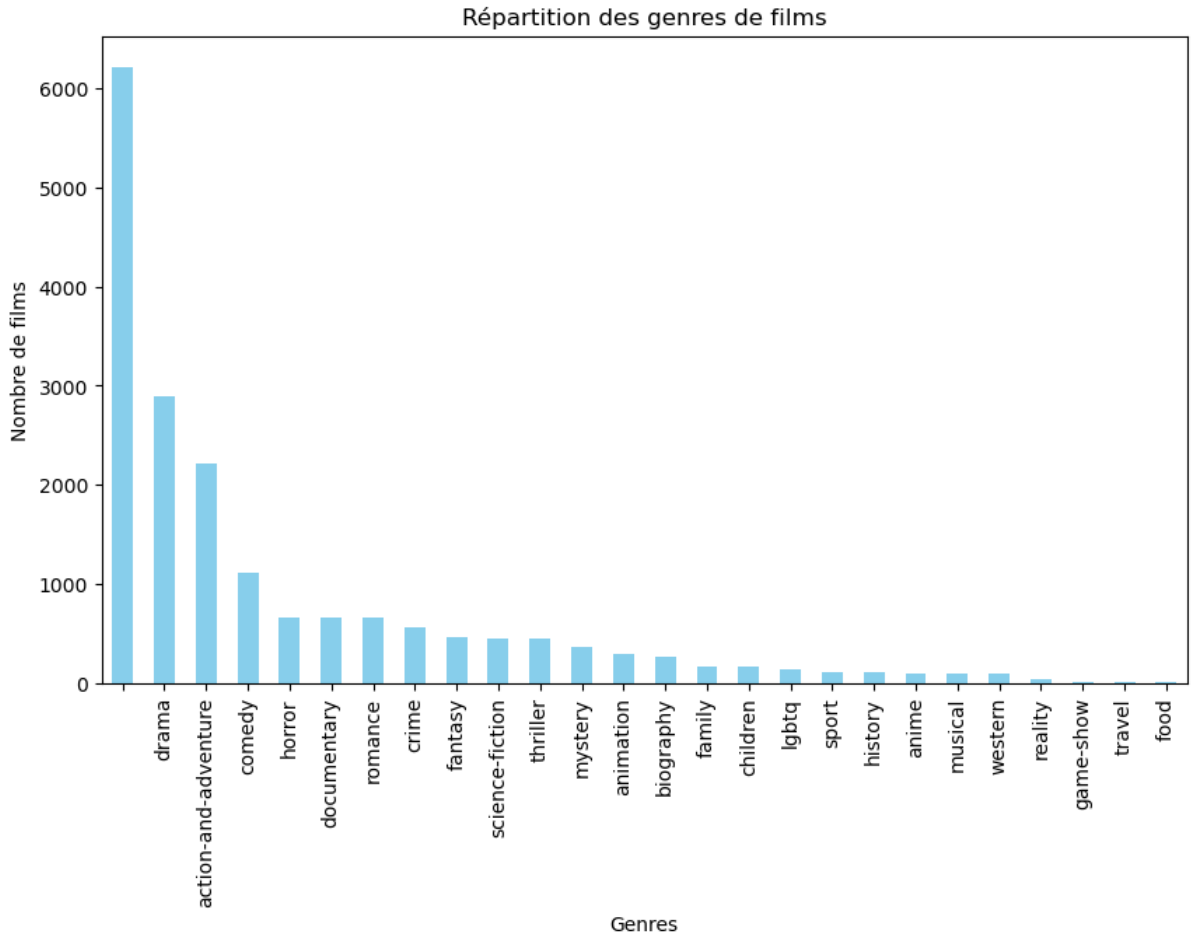

```
# Compter le nombre de films par genre
genre_counts = df_clean['Genres'].str.split(', ').explode().value_counts()
top_genres = genre_counts.head(15)

# Créer un graphique à secteurs pour visualiser la répartition des genres de films
plt.figure(figsize= (8, 8))
plt.pie(top_genres, labels=top_genres.index, autopct='%1.1f%%', startangle=140)
plt.title('Répartition des genres de films')
plt.show()
```



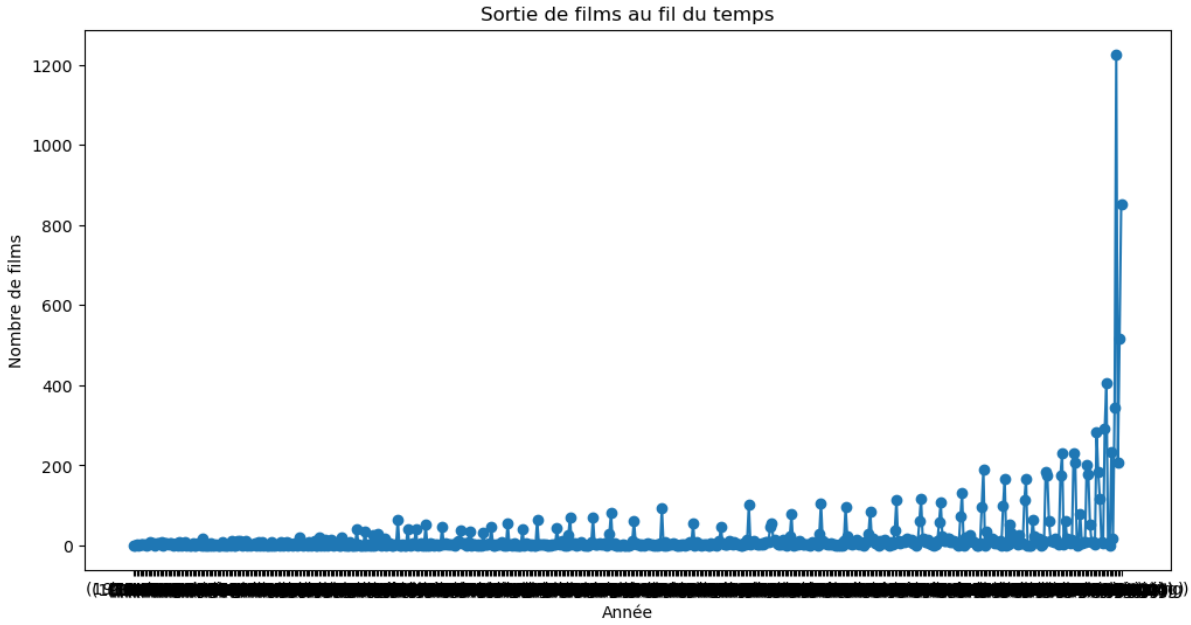

```
# Compter le nombre d'équipages par origine
origin_counts = df_clean['Origin'].value_counts()
top_origin = origin_counts.head(12)

# Créer un graphique à secteurs pour visualiser la répartition des équipages par origine
plt.figure(figsize = (8, 8))
plt.pie(top_origin, labels=top_origin.index, autopct='%1.1f%%', startangle=150)
plt.title('Répartition des équipages par origine')
plt.show()
```




```
# Compter le nombre de films par genre
genre_counts = df_clean['Genres'].str.split(', ').explode().value_counts()

# Créer un graphique à barres empilées pour visualiser la répartition des genres de films
plt.figure(figsize= (10, 6))
genre_counts.plot(kind='bar', color='skyblue')
plt.xlabel('Genres')
plt.ylabel('Nombre de films')
plt.title('Répartition des genres de films')
plt.show()
```



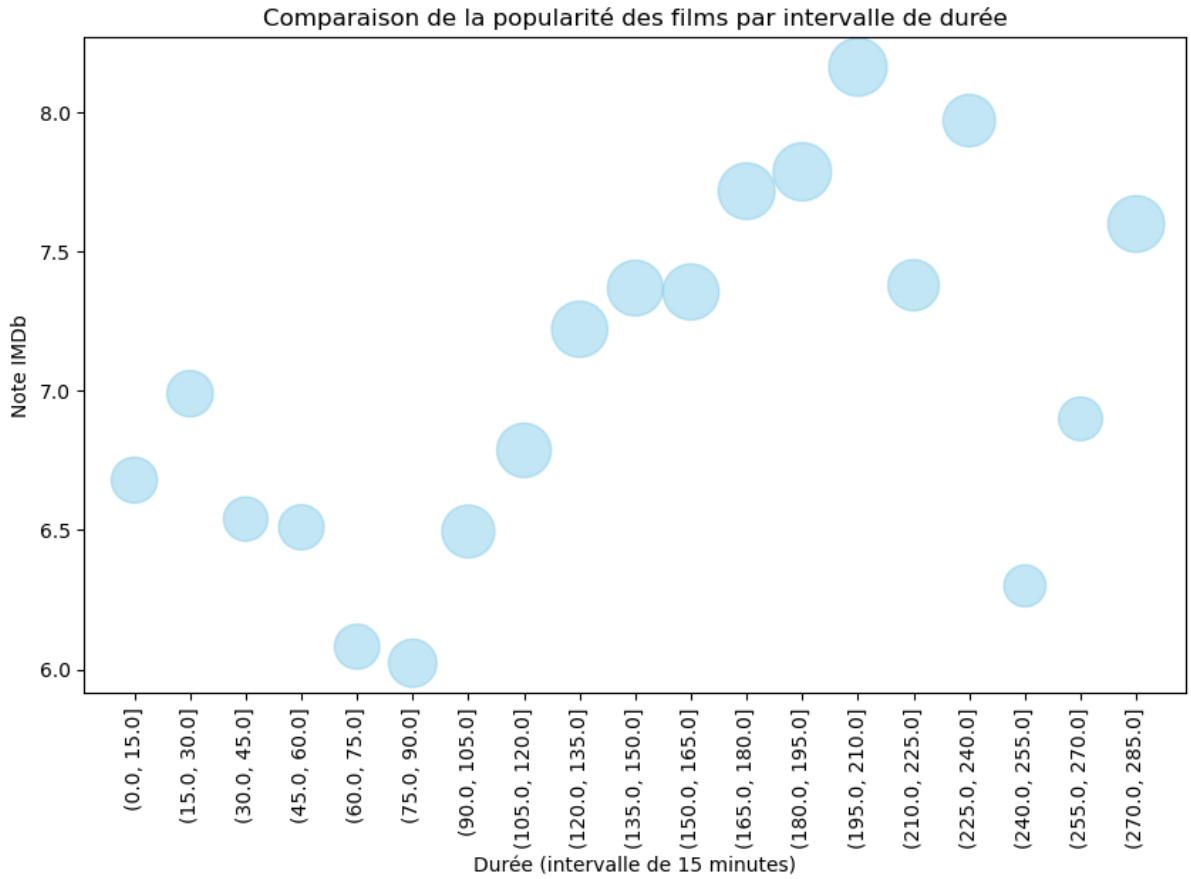

```
# Compter le nombre de films par année
year_counts = df_clean['Year'].value_counts().sort_index()

# Créer un graphique temporel
plt.figure(figsize= (12, 6))
plt.plot(year_counts.index, year_counts.values, marker='o')
plt.xlabel('Année')
plt.ylabel('Nombre de films')
plt.title('Sortie de films au fil du temps')
plt.show()
```



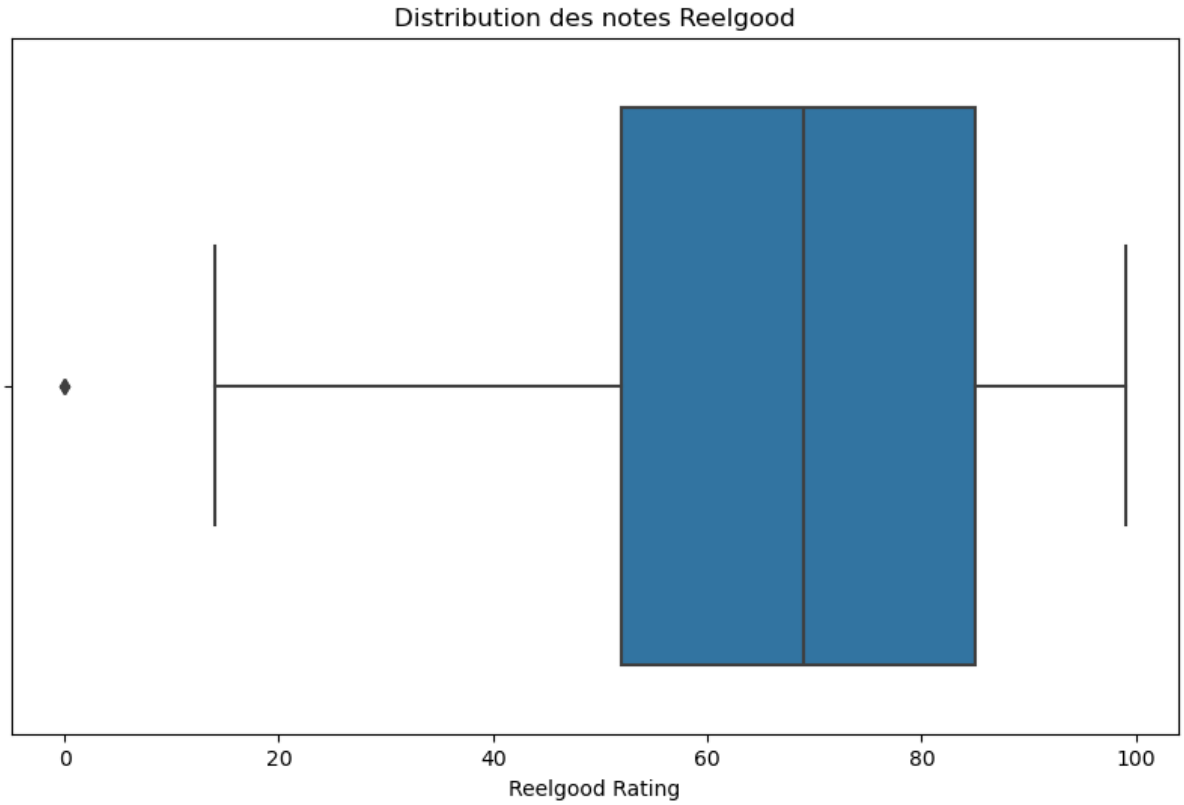

```
df_clean['IMDB Rating'] = pd.to_numeric(df_clean['IMDB Rating'], errors='coerce')
df_clean['Reelgood Rating'] = pd.to_numeric(df_clean['Reelgood Rating'],
errors='coerce')

# Calculer la moyenne des notes IMDb et Reelgood pour chaque intervalle
grouped = df_clean.groupby('Duration Interval').agg({
    'IMDB Rating': 'mean',
    'Reelgood Rating': 'mean'
}).reset_index()

grouped.head()
```

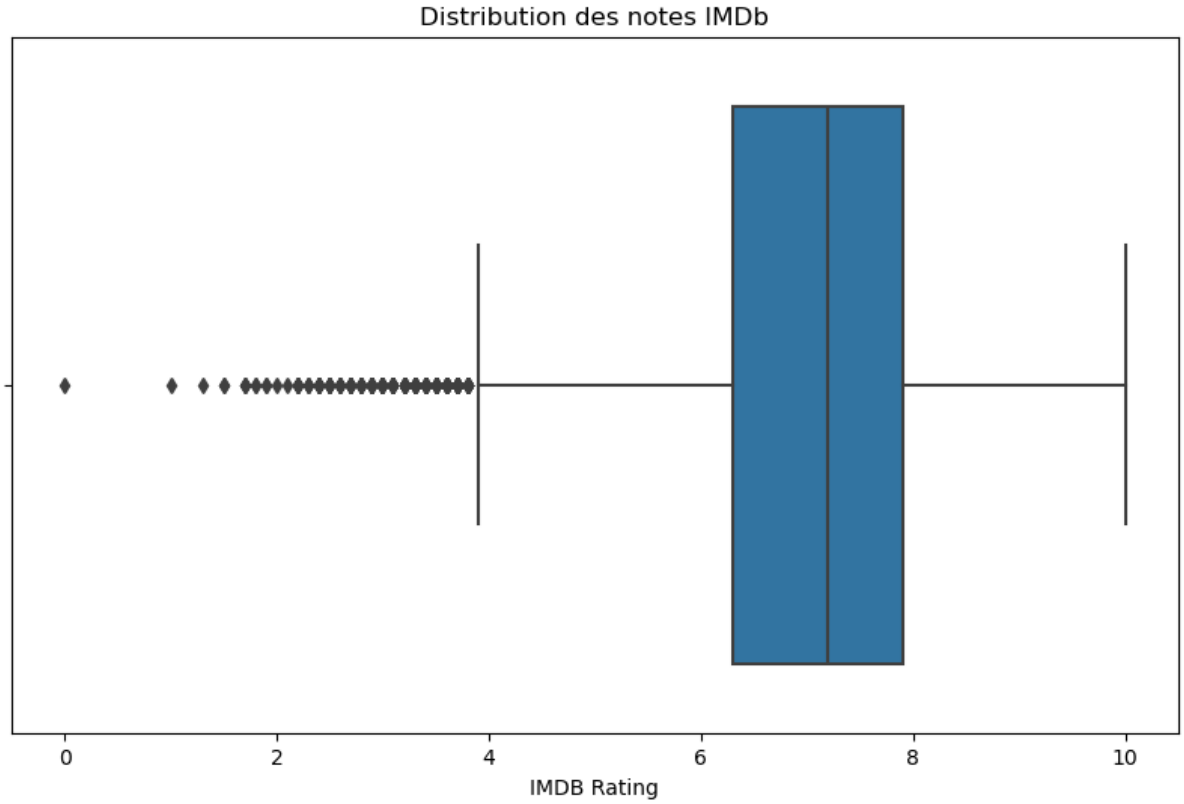
	Duration Interval	IMDB Rating	Reelgood Rating
0	(0.0, 15.0]	6.679762	54.102804
1	(15.0, 30.0]	6.990541	54.876289
2	(30.0, 45.0]	6.540000	50.113208
3	(45.0, 60.0]	6.510370	52.497076
4	(60.0, 75.0]	6.081522	52.070093

```
# Définir la taille des bulles
bubble_size = grouped['Reelgood Rating'] * 10

# Créer un diagramme à bulles
plt.figure(figsize= (10, 6))
plt.scatter(grouped['Duration Interval'].astype(str), grouped['IMDB Rating'],
s=bubble_size, alpha=0.5, color='skyblue')
plt.xlabel('Durée (intervalle de 15 minutes)')
plt.ylabel('Note IMDb')
plt.title('Comparaison de la popularité des films par intervalle de durée')
plt.xticks(rotation=90)
plt.show()
```




```
import seaborn as sns
# Créer un diagramme en boîte à moustaches pour les notes IMDb
plt.figure(figsize= (10, 6))
sns.boxplot(x=df_clean['IMDB Rating'])
plt.title('Distribution des notes IMDb')
plt.show()

# Créer un diagramme en boîte à moustaches pour les notes Reelgood
plt.figure(figsize= (10, 6))
sns.boxplot(x=df_clean['Reelgood Rating'])
plt.title('Distribution des notes Reelgood')
plt.show()
```


Distribution des notes Reelgood
0
20
40
60
80
100
Reelgood Rating


Distribution des notes IMDb
0
2
4
6
8
10
IMDB Rating

```
```

3. Configuration de l'Environnement

- **Structure des Dossiers**

Pour garantir un fonctionnement correct, l'environnement de travail doit être organisé comme suit :

```
├── Codebook
├── Pages/
│   ├── tv/
│   │   ├── Source/
│   │   │   ├── apple-tv
│   │   │   ├── netflix
│   │   │   └── ...
│   │   ├── Genre/
│   │   │   ├── action-and-adventure
│   │   │   ├── animation
│   │   │   ├── anime
│   │   │   ├── biography
│   │   │   ├── children
│   │   │   └── ...
│   │   ├── List/
│   │   │   ├── baseball
│   │   │   ├── college
│   │   │   ├── fashing
│   │   │   ├── games
│   │   │   ├── feel-good
│   │   │   └── ...
│   ├── movies/
│   │   ├── Source/
│   │   │   ├── apple-tv
│   │   │   ├── netflix
│   │   │   └── ...
│   │   ├── Genre/
│   │   │   ├── action-and-adventure
│   │   │   ├── animation
│   │   │   ├── anime
```

```
|   |   |   ├── biography
|   |   |   ├── children
|   |   |   └── ...
|   |   ├── List/
|   |   |   ├── africa
|   |   |   ├── animal
|   |   |   ├── car
|   |   |   ├── boxing
|   |   |   └── ...
|   ├── new/
|   |   ├── New/
|   |   |   ├── amazon
|   |   |   ├── Disney_plus
|   |   |   ├── Peacock
|   |   |   └── ...
|   |   ├── Coming/
|   |   |   ├── Netflix
|   |   |   ├── Hbo
|   |   |   ├── Hulu
|   |   |   └── ...
|   |   ├── Leaving/
|   |   |   ├── Amazon
|   |   |   ├── Apple_tv_plus
|   |   |   ├── Hbo
|   |   |   └── ...
├── projet_web_scraping.ipynb
├── scraping.log
├── projet web scarping.pdf
└── Resumé
```