🎬 Analyse Exploratoire des Films : Genre, Durée, Popularité

Dans ce projet, je explorai les données de films issues de la base IMDB pour comprendre :

1 Compréhension des affaires

Contexte

Un nouveau studio de cinéma souhaite comprendre les facteurs qui déterminent le succès d'un film. Pour cela, il dispose de données IMDB (genres, durée, notes, etc.) et souhaite s'appuyer sur une analyse exploratoire.

Objectif

Analyser les données pour fournir **3 recommandations stratégiques** permettant de :

- Produire des films populaires
- Maximiser les revenus
- Mieux planifier les productions (choix du genre, durée, équipe...)

Questions Clés

1. Quels genres plaisent le plus au public ?
2. Quelle est la durée idéale pour un film bien noté ?
3. Existe-t-il un lien entre popularité et succès financier ?
4. Quels types de films rapportent le plus au box-office ?

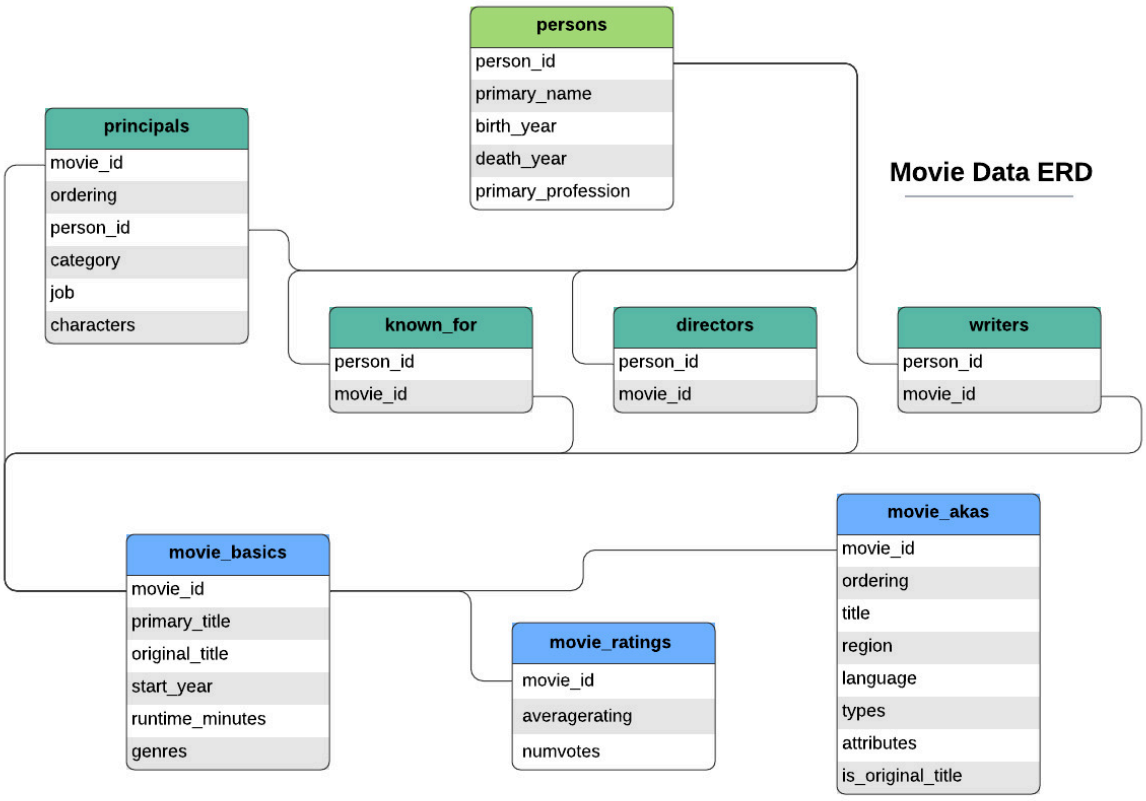
Impact attendu

Les résultats aideront le studio à :

- Choisir des genres et formats adaptés
- Optimiser les budgets et les durées
- Réduire les risques grâce aux données

In [59]:
```
from IPython.display import Image, display

display(Image(filename="Images/IMDB_ERD.jpeg", width=800))
```



2. Compréhension des données

Dans cette section, j'analyse la structure de la base de données `im.db` afin d'identifier les tables disponibles et déterminer lesquelles seront utiles pour répondre à nos **questions commerciales**.

La base contient plusieurs tables issues d'IMDB, représentant des informations sur les films, les notes du public, les équipes de production, etc.

Tables présentes dans la base de données `im.db`

Nom de la table	Description
movie_basics	Informations générales sur les films (titre, année, durée, genres, etc.)
movie_ratings	Notes moyennes et nombre de votes pour chaque film
persons	Détails sur les personnes (nom, année de naissance, profession)
principals	Liste des principaux intervenants dans chaque film
directors	Liens entre films et réalisateurs
writers	Liens entre films et scénaristes
movie_akas	Titres alternatifs des films (par région/langue)

Nom de la table	Description
known_for	Films pour lesquels une personne est la plus connue

Objectif de l'analyse

Pour explorer les facteurs de succès d'un film, je vais principalement utiliser :

- `movie_basics` => titre, année, genre, durée
- `movie_ratings` => note moyenne (`averagerating`) et nombre de votes (`numvotes`)
- `principals` + `persons` => identification des principaux intervenants (acteurs, réalisateurs, scénaristes)

Les autres tables comme `directors` , `writers` , `known_for` , `movie_akas` pourront **enrichir** l'analyse, mais elles ne sont **pas centrales** pour les premières analyses.

Tables clés retenues pour l'analyse

1 `movie_basics` : informations générales sur les films.
2 `movie_ratings` : popularité et notes des films.
3 `principals` + `persons` : identification des équipes principales (cast & crew).

Stratégie de jointure prévue

- `movie_basics` sera la **table centrale** via la clé `movie_id` .
- `movie_ratings` sera directement reliée via `movie_id` .
- `principals` permettra de relier les films aux personnes (via `person_id`).
- `persons` donnera les détails sur les intervenants (nom, profession, etc.).
- Les tables `directors` et `writers` peuvent être utilisées pour des analyses spécifiques supplémentaires (ex : impact des réalisateurs sur les notes).

Importation des librairies et configuration

Je commence par importer les principales librairies nécessaires à l'analyse :

- **pandas** : pour la manipulation et l'analyse des données tabulaires
- **sqlite3** : pour se connecter à la base de données SQLite `im.db`
- **matplotlib.pyplot** et **seaborn** : pour la visualisation des données
- **numpy** : pour les calculs numériques et statistiques

Ensuite, nous configurons l'apparence des graphiques avec `seaborn` et `matplotlib` pour garantir des visualisations claires et cohérentes :

```
sns.set(style="whitegrid")  # Style avec fond blanc et grille discrète
plt.rcParams["figure.figsize"] = (10, 6)  # Taille standard des graphiques
```

In [14]:
```
import pandas as pd
import sqlite3
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Configuration des graphiques
sns.set(style="whitegrid")
plt.rcParams["figure.figsize"] = (10,6)
```

Connexion à la base de données

Nous utilisons **sqlite3** pour établir une connexion avec la base de données locale `im.db`, qui contient les données cinéma nécessaires pour l'analyse.

La base de données est stockée dans le dossier `Data/`, d'où la syntaxe suivante :

In [19]:
```
# Imports here
conn = sqlite3.connect("Data/im.db")
```

Chargement des données principales

Après avoir connecté la base `im.db`, je charges les principales tables nécessaires à l'analyse :

- **movie_basics** : Informations de base sur les films (titre, genres, durée, etc.)
- **movie_ratings** : Notes moyennes et nombre de votes par film
- **directors** : Relations entre les films et leurs réalisateurs
- **persons** : Détails sur les personnes impliquées (noms, rôles, etc.)

Code utilisé :

In [21]:
```
# Charger les tables principales
movie_basics = pd.read_sql("SELECT * FROM movie_basics", conn)
movie_ratings = pd.read_sql("SELECT * FROM movie_ratings", conn)
directors = pd.read_sql("SELECT * FROM directors", conn)
persons = pd.read_sql("SELECT * FROM persons", conn)

# Vérification des tables disponibles
df = pd.read_sql("SELECT name FROM sqlite_master WHERE type='table';", conn)
print("Tables dans la base :")
print(df)
```

```
Tables dans la base :
           name
0   movie_basics
1      directors
2      known_for
3     movie_akas
4  movie_ratings
5        persons
6     principals
7        writers
```

Fusion des données principales

Afin d'obtenir une table complète contenant à la fois les informations descriptives des films et leurs notes, je fusionnes :

- **movie_basics** : Contient les titres, genres, années, durées, etc.
- **movie_ratings** : Contient les moyennes de notes (`averagerating`) et le nombre de votes (`numvotes`)

Code utilisé :

In [24]:
```
# Fusionner movie_basics et movie_ratings
movies = pd.merge(movie_basics, movie_ratings, on="movie_id", how="left")
```

Data Understanding

Filtrage des films récents et populaires

je souhaites nous concentrer sur les films récents qui ont suffisamment de votes pour que leur note soit représentative.

Critères de filtrage :

- **Année de sortie ≥ 2000** : Se concentrer sur les films récents (21e siècle)
- **Nombre de votes > 500** : Exclure les films obscurs ou peu connus pour éviter les biais liés à un nombre trop faible de votes

Code utilisé :

In [25]:
```
# Garder les films récents avec suffisamment de votes
filtered_movies = movies[(movies['start_year'] >= 2000) & (movies['numvotes'] > 500
filtered_movies.head()
```

Out[25]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	ge
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Dr
15	tt0176694	The Tragedy of Man	Az ember tragédiája	2011	160.0	Animation,Drama,Hi
20	tt0249516	Foodfight!	Foodfight!	2012	91.0	Action,Animation,Cor
32	tt0293069	Dark Blood	Dark Blood	2012	86.0	Th
38	tt0315642	Wazir	Wazir	2016	103.0	Action,Crime,Dr

Top 10 des films les mieux notés (depuis 2000)

Je sélectionnes les 10 films les mieux notés parmi ceux :

- Sortis à partir de l'année 2000
- Ayant reçu plus de 500 votes

Code utilisé :

In [26]:
```
top10_movies = filtered_movies.sort_values(by='averagerating', ascending=False).hea
print(top10_movies[['primary_title', 'averagerating']])
```

```
                                          primary_title  averagerating
136655                                        Eghantham            9.7
118887                 Once Upon a Time ... in Hollywood            9.7
72358                                   I Want to Live             9.6
144567                        Yeh Suhaagraat Impossible            9.6
103419                              Ekvtime: Man of God            9.6
142169                              Ananthu V/S Nusrath            9.6
101973                                    Aloko Udapadi            9.5
133221                                            Mosul            9.5
115351  Hare Krishna! The Mantra, the Movement and the...            9.5
100206                          The Children of Genghis            9.4
```

Analyse des genres : Moyenne des notes par genre

Pour mieux comprendre l'impact des genres sur la qualité perçue des films, nous avons :

1. **Nettoyé la colonne `genres`** : remplacé les valeurs manquantes par une chaîne vide.
2. **Transformé les genres en liste** : certains films ont plusieurs genres séparés par des virgules, nous les avons découpés.
3. **Explosé les genres** : chaque film est répliqué pour chaque genre associé afin d'analyser les genres individuellement.
4. **Calculé la moyenne des notes (`averagerating`) pour chaque genre**.

Code utilisé :

In [27]:
```
# Nettoyer les genres
filtered_movies['genres'] = filtered_movies['genres'].fillna('')

# Transformer en liste
filtered_movies['genres_list'] = filtered_movies['genres'].apply(lambda x: [g.strip

# Exploser pour avoir un genre par ligne
movies_exploded = filtered_movies.explode('genres_list')

# Calcul de la moyenne des notes par genre
genre_avg = movies_exploded.groupby('genres_list')['averagerating'].mean().sort_val
print(genre_avg)
```

```
genres_list
News           7.530556
Documentary    7.196806
Biography      6.937696
History        6.787313
Sport          6.786643
Music          6.722306
Animation      6.559959
Musical        6.463889
War            6.460759
Drama          6.394912
Family         6.259333
Romance        6.238980
Crime          6.177470
Adventure      6.032336
Comedy         5.980927
Mystery        5.912798
Action         5.852269
Western        5.850000
Fantasy        5.830622
Thriller       5.727822
Game-Show      5.600000
Sci-Fi         5.467765
Horror         5.062614
Name: averagerating, dtype: float64
```

Analyse des réalisateurs : Moyenne des notes par réalisateur

Je voulu identifier les réalisateurs dont les films sont les mieux notés depuis l'année 2000.

Étapes réalisées :

1. **Fusion des tables :**

- Fusion des données `directors` et `persons` pour récupérer les noms complets des réalisateurs.

- Fusion avec `movie_ratings` pour obtenir les notes (`averagerating`).

2. **Calcul de la moyenne des notes par réalisateur :**

je calculé la moyenne des notes pour chaque réalisateur afin d'obtenir un classement basé sur la satisfaction du public.

Code utilisé :

In [28]:
```
directors_data = pd.merge(directors, persons, on='person_id', how='left')
movies_dir = pd.merge(directors_data, movie_ratings, on='movie_id', how='left')

avg_rating_dir = movies_dir.groupby('primary_name')['averagerating'].mean().sort_va
print(avg_rating_dir.head(10))
```

```
primary_name
Chad Carpenter            10.0
Lindsay Thompson          10.0
Masahiro Hayakawa         10.0
Emre Oran                 10.0
Ivana Diniz               10.0
Stephen Peek              10.0
Loreto Di Cesare          10.0
Michiel Brongers          10.0
Tristan David Luciotti    10.0
Raphael Sbarge             9.9
Name: averagerating, dtype: float64
```

Croisement des tables `movie_basics` et `movie_ratings` pour obtenir les films les mieux notés

La requête suivante effectue une jointure entre la table `movie_basics` (informations générales sur les films)
et la table `movie_ratings` (notes attribuées aux films) via la colonne commune `movie_id` .

Elle filtre les films ayant reçu plus de 10 000 votes (`numvotes > 10000`) afin de garantir la fiabilité des notes,
puis trie les résultats par note moyenne décroissante (`averagerating DESC`).

Seuls les 10 premiers films sont sélectionnés.

Colonnes sélectionnées :

- `primary_title` : titre principal du film
- `start_year` : année de sortie
- `genres` : genres du film

- `averagerating` : note moyenne
- `numvotes` : nombre total de votes

Le résultat est ensuite importé dans un DataFrame Pandas (`top_movies`) pour une analyse ultérieure en Python.

In [31]:
```
#Croiser movie_basics avec movie_ratings pour voir les films les mieux notés
query = """
SELECT b.primary_title, b.start_year, b.genres, r.averagerating, r.numvotes
FROM movie_basics b
JOIN movie_ratings r ON b.movie_id = r.movie_id
WHERE r.numvotes > 10000
ORDER BY r.averagerating DESC
LIMIT 10;
"""
top_movies = pd.read_sql(query, conn)
top_movies
```

Out[31]:

	primary_title	start_year	genres	averagerating	numvotes
0	The Mountain II	2016	Action,Drama,War	9.3	100568
1	Aynabaji	2016	Crime,Mystery,Thriller	9.3	18470
2	Wheels	2014	Drama	9.3	17308
3	CM101MMXI Fundamentals	2013	Comedy,Documentary	9.2	41560
4	O.J.: Made in America	2016	Biography,Crime,Documentary	8.9	14946
5	Drishyam	2013	Crime,Drama,Thriller	8.8	24326
6	Ratsasan	2018	Action,Crime,Thriller	8.8	10518
7	96	2018	Drama,Romance	8.8	10903
8	Avengers: Endgame	2019	Action,Adventure,Sci-Fi	8.8	441135
9	Inception	2010	Action,Adventure,Sci-Fi	8.8	1841066

Préparation des colonnes numériques et gestion des valeurs manquantes

- Conversion de la colonne `runtime_minutes` en type numérique avec gestion des erreurs (`errors='coerce'`)
  pour transformer les valeurs non convertibles en `NaN` .
- Remplacement des valeurs manquantes (`NaN`) dans `averagerating` par la moyenne des notes existantes,
  afin d'éviter les valeurs manquantes qui pourraient perturber les analyses.

- Remplacement des valeurs manquantes dans `numvotes` par 0, considérant qu'aucun vote n'a été enregistré.
- Affichage de statistiques descriptives pour vérifier la distribution et la cohérence des données dans ces colonnes.

In [32]:
```
# Convertir la durée en numérique
movies['runtime_minutes'] = pd.to_numeric(movies['runtime_minutes'], errors='coerce

# Remplacer les NaN dans averagerating par la moyenne
movies['averagerating'] = movies['averagerating'].fillna(movies['averagerating'].me

# Remplacer les NaN dans numvotes par 0
movies['numvotes'] = movies['numvotes'].fillna(0)

# Vérification
movies[['runtime_minutes', 'averagerating', 'numvotes']].describe()
```

Out[32]:

	runtime_minutes	averagerating	numvotes
count	114405.000000	146144.000000	1.461440e+05
mean	86.187247	6.332729	1.780734e+03
std	166.360590	1.048544	2.160759e+04
min	1.000000	1.000000	0.000000e+00
25%	70.000000	6.332729	0.000000e+00
50%	87.000000	6.332729	5.000000e+00
75%	99.000000	6.500000	5.100000e+01
max	51420.000000	10.000000	1.841066e+06

Extraction et transformation des genres en variables binaires

- Initialisation d'un ensemble (`set`) pour collecter tous les genres uniques présents dans la colonne `genres` .
- Remplacement des valeurs manquantes dans `genres` par une chaîne vide pour éviter les erreurs.
- Parcours de chaque chaîne de genres (séparés par des virgules) pour extraire chaque genre unique.
- Création d'une colonne binaire par genre dans le DataFrame :
  - Pour chaque film, la colonne correspondant à un genre prend la valeur `1` si le film appartient à ce genre, sinon `0` .
- Affichage des premières lignes des colonnes binaires pour vérifier la transformation.

Cette transformation facilite les analyses statistiques et la modélisation par genre.

In [33]:
```python
# Initialiser une liste des genres uniques
genre_list = set()
movies['genres'] = movies['genres'].fillna('')

# Extraire les genres uniques
for g in movies['genres']:
    for genre in g.split(","):
        if genre.strip() != "":
            genre_list.add(genre.strip())

# Créer des colonnes binaires pour chaque genre
for genre in genre_list:
    movies[genre] = movies['genres'].apply(lambda x: int(genre in x.split(",")))

# Vérification rapide
movies[list(genre_list)].head(10)
```

Out[33]:

	Animation	Western	War	Game-Show	Musical	Documentary	Reality-TV	Sci-Fi	Action	Horro
0	0	0	0	0	0	0	0	0	1	
1	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	
7	1	0	0	0	0	0	0	0	0	
8	0	0	0	0	0	1	0	0	0	
9	0	0	0	0	0	0	0	0	0	

10 rows × 27 columns

Extraction des réalisateurs à partir des tables `principals` et `persons`

- Chargement des tables `principals` (rôles des intervenants dans les films) et `persons` (détails des personnes) depuis la base de données.
- Fusion (`merge`) des deux tables sur la colonne `person_id` pour associer les rôles aux noms des personnes.
- Filtrage pour ne conserver que les réalisateurs (`category == 'director'`).

- Affichage des premières lignes avec l'identifiant du film (`movie_id`) et le nom du réalisateur (`primary_name`).

Cette étape permet d'identifier facilement les réalisateurs associés à chaque film.

In [34]:
```
# Charger les tables nécessaires
principals = pd.read_sql("SELECT * FROM principals", conn)
persons = pd.read_sql("SELECT * FROM persons", conn)

# Fusionner pour obtenir les noms des intervenants
crew = pd.merge(principals, persons, on="person_id", how="left")

# Filtrer pour les réalisateurs
directors = crew[crew['category'] == 'director']
directors[['movie_id', 'primary_name']].head()
```

Out[34]:

	movie_id	primary_name
1	tt0111414	Frank Howson
8	tt0323808	Robin Hardy
18	tt0417610	Alejandro Chomski
28	tt0469152	Alyssa R. Bennett
35	tt0473032	J. Neil Schulman

Statistiques descriptives sur les notes, durées et nombre de votes

Ce tableau résume les caractéristiques principales des colonnes suivantes dans le DataFrame `movies` :

- `averagerating` : note moyenne des films
- `runtime_minutes` : durée des films en minutes
- `numvotes` : nombre de votes reçus par chaque film

Les statistiques affichées comprennent la moyenne, l'écart-type, les valeurs minimales et maximales, ainsi que les quartiles,
ce qui permet d'avoir un aperçu rapide de la distribution et de la variabilité de ces variables.

In [35]:
```
# Statistiques sur les notes et durées
movies[['averagerating', 'runtime_minutes', 'numvotes']].describe()
```

Out[35]:

	averagerating	runtime_minutes	numvotes
count	146144.000000	114405.000000	1.461440e+05
mean	6.332729	86.187247	1.780734e+03
std	1.048544	166.360590	2.160759e+04
min	1.000000	1.000000	0.000000e+00
25%	6.332729	70.000000	0.000000e+00
50%	6.332729	87.000000	5.000000e+00
75%	6.500000	99.000000	5.100000e+01
max	10.000000	51420.000000	1.841066e+06

Analysis and Results

4. Analysis et Recommendations

Visualisations

- **Histogramme des notes**
  Pour visualiser la distribution des notes moyennes des films.
- **Barplot des genres les mieux notés**
  Permet d'identifier quels genres obtiennent les meilleures critiques du public.
- **Scatterplots** :
  - **Durée vs Note moyenne** : Analyse de la relation entre la longueur d'un film et sa popularité.
  - **Popularité vs Revenus** : Si les données du box-office sont disponibles, cette visualisation explore si les films populaires rapportent plus.

Analyses statistiques

- **Corrélations** :
  - **Durée vs Note moyenne** : Test de corrélation pour voir s'il existe un lien entre la longueur d'un film et son score IMDb.
  - **Popularité (nombre de votes) vs Note moyenne** : Vérification d'un éventuel biais de popularité.
- **Régression linéaire** :

    - **Durée => Note moyenne** : Modèle pour prédire la note moyenne en fonction de la durée d'un film.
- **ANOVA** :
    - Test d'analyse de variance pour voir si les différences de notes moyennes entre genres sont statistiquement significatives.

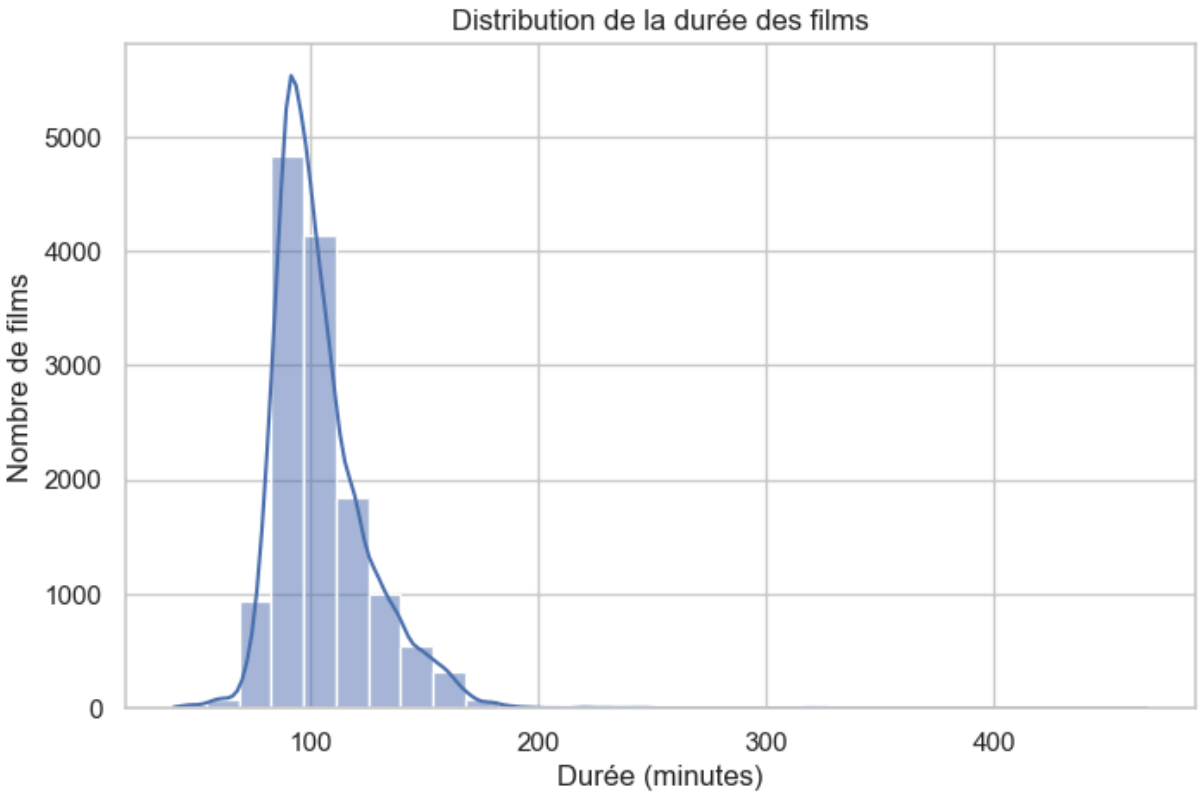
Visualisation de la distribution des durées des films

Ce graphique représente un histogramme de la durée des films (`runtime_minutes`) avec une estimation de la densité (courbe KDE) superposée.

- L'histogramme montre la répartition du nombre de films par intervalles de durée.
- La courbe KDE permet d'avoir une estimation lisse de la densité des durées, mettant en évidence les tendances et les pics.

Cette visualisation aide à comprendre la distribution des durées et à détecter d'éventuelles valeurs atypiques ou modes dans les données.

In [48]:
```
plt.figure(figsize=(8,5))
sns.histplot(filtered_movies['runtime_minutes'], bins=30, kde=True)
plt.title("Distribution de la durée des films")
plt.xlabel("Durée (minutes)")
plt.ylabel("Nombre de films")
plt.show()
```

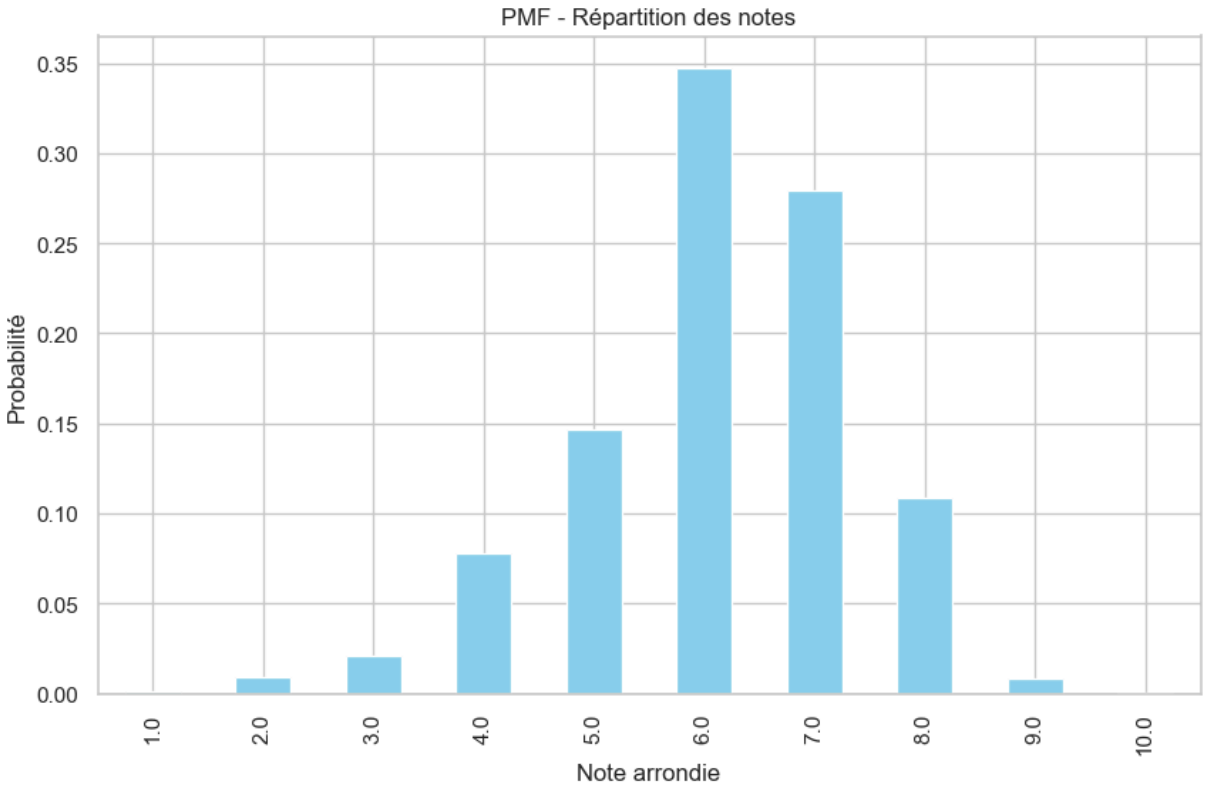


Distribution des notes arrondies (PMF)

- La colonne `averagerating` est arrondie à l'entier le plus proche (`rounded_rating`).
- La fonction de masse de probabilité (PMF) est calculée en comptant la fréquence relative de chaque note arrondie.
- Le graphique en barres représente la probabilité d'obtenir chaque note entière.

Cette visualisation permet d'analyser la répartition des notes de manière discrète, facilitant la compréhension des tendances de notation.

In [47]:
```
filtered_movies['rounded_rating'] = filtered_movies['averagerating'].round()
pmf = filtered_movies['rounded_rating'].value_counts(normalize=True).sort_index()
pmf.plot(kind='bar', color='skyblue')
plt.title("PMF - Répartition des notes")
plt.xlabel("Note arrondie")
plt.ylabel("Probabilité")
plt.show()
```



Distribution des notes moyennes (IMDb)

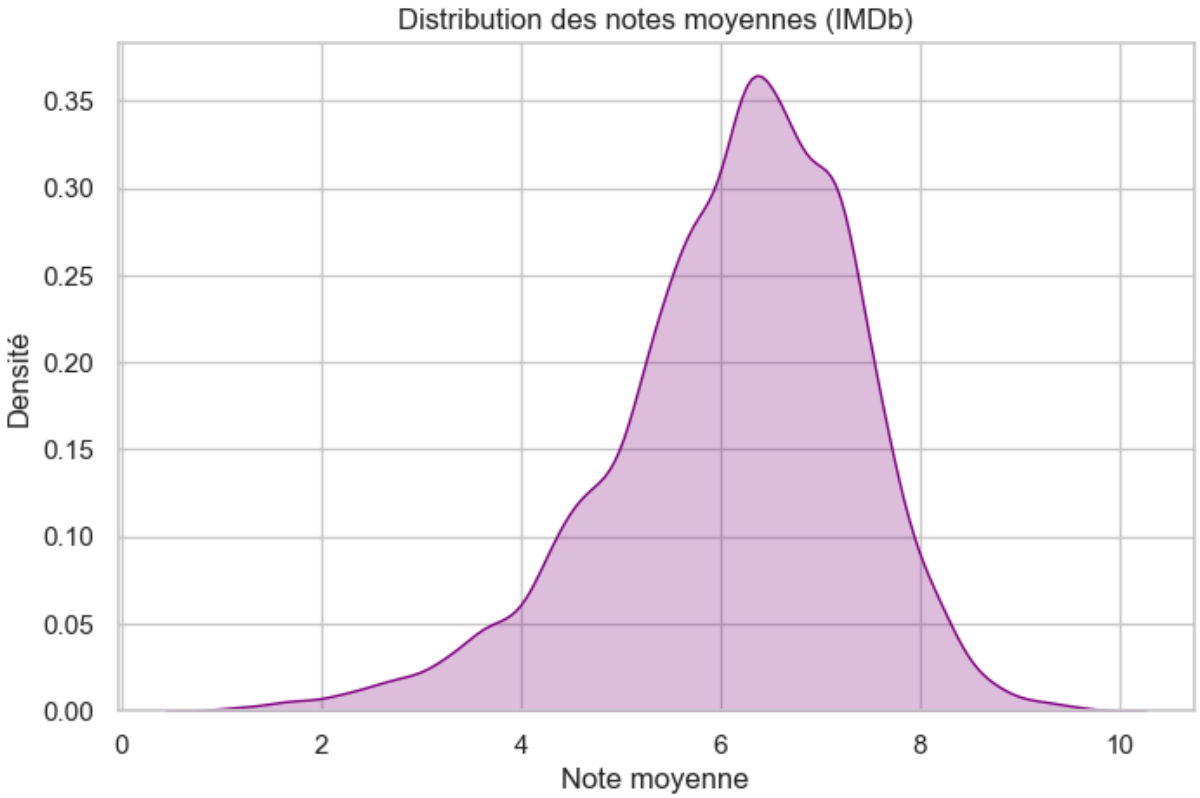
Ce graphique utilise un **KDE plot** (Kernel Density Estimation) pour visualiser la distribution continue des notes moyennes (`averagerating`) des films.

- La courbe lisse montre comment les notes sont réparties dans l'ensemble des films filtrés.

- L'aire sous la courbe est remplie pour faciliter la lecture visuelle de la densité.
- Cette représentation est utile pour identifier les concentrations de films autour de certaines notes, ainsi que pour détecter des asymétries ou anomalies.

Cela donne une vue plus fine et continue de la répartition des notes qu'un simple histogramme.

In [46]:
```
plt.figure(figsize=(8,5))
sns.kdeplot(filtered_movies['averagerating'], fill=True, color='purple')
plt.title("Distribution des notes moyennes (IMDb)")
plt.xlabel("Note moyenne")
plt.ylabel("Densité")
plt.show()
```



Simulation du Théorème Central Limite (TCL)

Cette expérience illustre le **Théorème Central Limite (TCL)** à partir des notes moyennes des films.

Étapes de la simulation :

- 1000 échantillons de taille 30 sont prélevés aléatoirement avec remise depuis la colonne `averagerating`.
- Pour chaque échantillon, la moyenne est calculée et ajoutée à une liste.
- Un histogramme des 1000 moyennes obtenues est ensuite tracé.

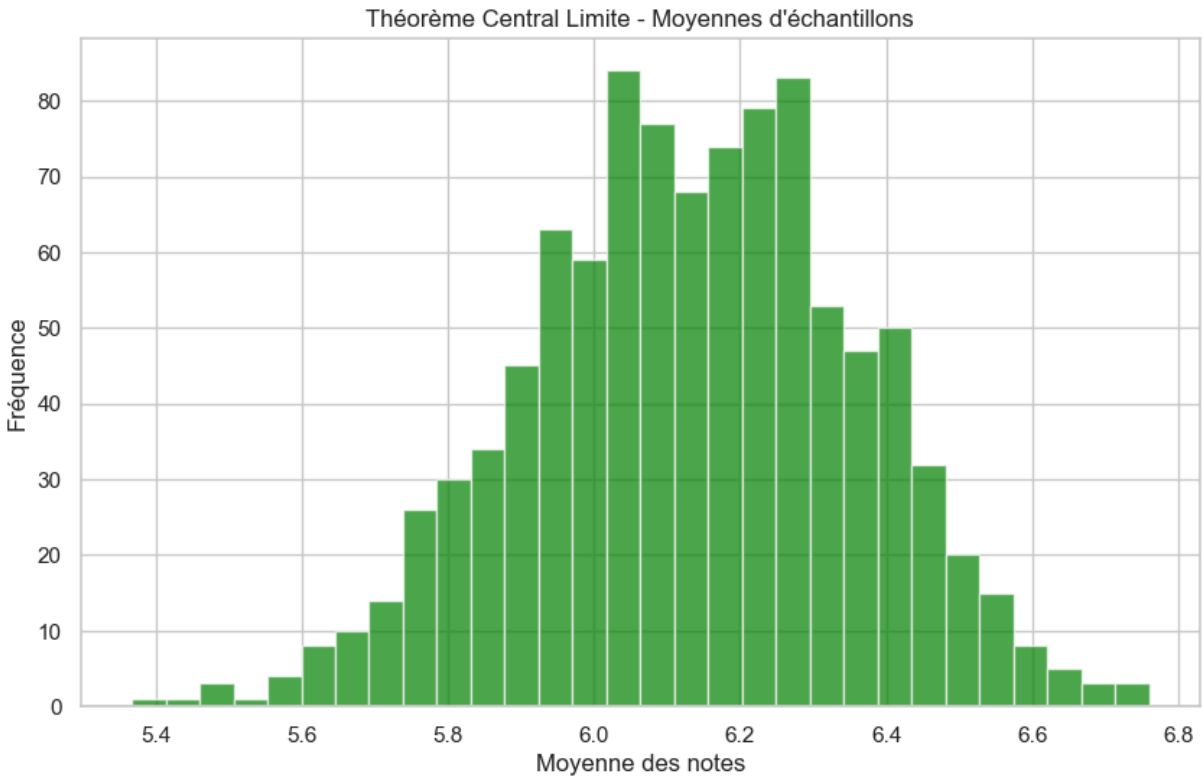
Interprétation :

Même si la distribution initiale des notes (`averagerating`) n'est pas parfaitement normale, la distribution des **moyennes d'échantillons** tend à suivre une distribution normale.

Cela confirme empiriquement le **Théorème Central Limite**, qui garantit la normalité des moyennes d'échantillons pour des tailles d'échantillons suffisamment grandes (ici, n=30).

In [45]:
```
means = []
for i in range(1000):
    sample = filtered_movies['averagerating'].sample(30, replace=True)
    means.append(sample.mean())

plt.hist(means, bins=30, color='green', alpha=0.7)
plt.title("Théorème Central Limite - Moyennes d'échantillons")
plt.xlabel("Moyenne des notes")
plt.ylabel("Fréquence")
plt.show()
```



Matrice de corrélation entre les variables principales

Cette visualisation montre la corrélation entre trois variables clés du dataset :

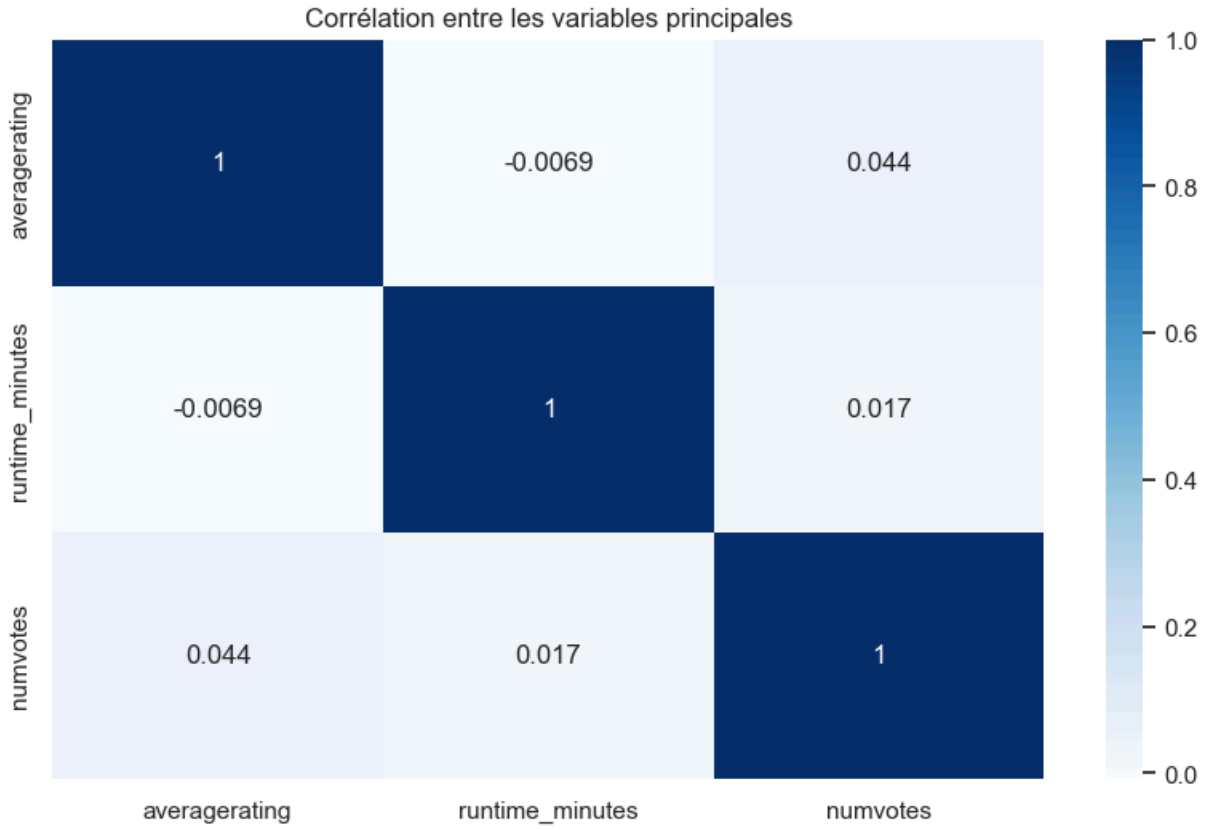
- `averagerating` : note moyenne des films
- `runtime_minutes` : durée des films en minutes
- `numvotes` : nombre total de votes

Interprétation :

- La matrice de corrélation est affichée sous forme de **heatmap** avec des annotations des coefficients de corrélation.
- Les valeurs varient entre **-1** (corrélation négative parfaite) et **+1** (corrélation positive parfaite).
- Une valeur proche de **0** indique l'absence de corrélation linéaire.

Cette analyse permet d'identifier les relations linéaires éventuelles entre les variables, ce qui peut guider les choix d'analyses statistiques ou de modélisation.

In [49]:
```
corr = movies[['averagerating', 'runtime_minutes', 'numvotes']].corr()
sns.heatmap(corr, annot=True, cmap="Blues")
plt.title("Corrélation entre les variables principales")
plt.show()
```



Test ANOVA : Comparaison des notes moyennes entre genres

Un **test ANOVA à un facteur** est réalisé pour comparer les notes moyennes (`averagerating`) entre trois genres de films :

- **Action**
- **Comédie**
- **Drame**

Objectif :

Vérifier s'il existe une **différence statistiquement significative** entre les moyennes des notes des différents genres.

Méthodologie :

- Séparation des notes moyennes par genre à partir des colonnes binaires (`Action` , `Comedy` , `Drama`).
- Utilisation de la fonction `f_oneway` de `scipy.stats` pour effectuer le test ANOVA.

Résultats :

- **F-statistic** : mesure du rapport de variance entre les groupes et à l'intérieur des groupes.
- **p-value** : si elle est inférieure à 0.05, on peut conclure qu'au moins un des genres a une moyenne significativement différente des autres.

Interprétation :

- Si **p-value < 0.05**, on rejette l'hypothèse nulle d'égalité des moyennes.
- Si **p-value ≥ 0.05**, on ne détecte pas de différence significative entre les genres.

In [53]:
```
for genre in ['Action', 'Comedy', 'Drama']:
    filtered_movies[genre] = filtered_movies['genres'].apply(lambda x: int(genre in
```

In [54]:
```
from scipy.stats import f_oneway

action = filtered_movies[filtered_movies['Action'] == 1]['averagerating']
comedy = filtered_movies[filtered_movies['Comedy'] == 1]['averagerating']
drama = filtered_movies[filtered_movies['Drama'] == 1]['averagerating']

f_stat, p_value = f_oneway(action, comedy, drama)
print("F-statistic:", f_stat)
print("p-value:", p_value)
```

```
F-statistic: 299.7889108461605
p-value: 3.406947119895249e-128
```

Régression Linéaire : Durée des films vs Note moyenne

Un modèle de **régression linéaire** est construit pour analyser la relation entre :

- **Variable indépendante (X)** : `runtime_minutes` (durée du film en minutes)
- **Variable dépendante (y)** : `averagerating` (note moyenne IMDb)

Étapes :

1. Suppression des films dont la durée est manquante avec `dropna`.
2. Ajout d'une constante (intercept) avec `sm.add_constant(X)` pour modéliser l'ordonnée à l'origine.
3. Ajustement d'un modèle de régression linéaire avec `statsmodels.OLS`.
4. Affichage des résultats avec `model.summary()`.

Résultats principaux :

- **Coef. (runtime_minutes)** : indique l'effet d'une minute supplémentaire sur la note moyenne (positif, négatif ou nul).
- **P-value** : permet de tester si cet effet est statistiquement significatif.
- **R-squared** : mesure la proportion de la variance expliquée par le modèle (plus il est proche de 1, mieux c'est).

Interprétation :

- Si la pente est significative et non nulle, la durée du film influence les notes moyennes.
- Si la p-value est > 0.05, la relation n'est **pas significative** statistiquement.

In [55]:
```
import statsmodels.api as sm

movies_clean = filtered_movies.dropna(subset=['runtime_minutes'])
X = movies_clean['runtime_minutes']
y = movies_clean['averagerating']

X = sm.add_constant(X)
model = sm.OLS(y, X).fit()

print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          averagerating   R-squared:                       0.087
Model:                            OLS   Adj. R-squared:                  0.087
Method:                 Least Squares   F-statistic:                     1321.
Date:                Sat, 19 Jul 2025   Prob (F-statistic):          2.63e-276
Time:                        12:50:36   Log-Likelihood:                -21887.
No. Observations:               13827   AIC:                         4.378e+04
Df Residuals:                   13825   BIC:                         4.379e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const              4.3230      0.051     85.240      0.000       4.224       4.422
runtime_minutes    0.0173      0.000     36.344      0.000       0.016       0.018
==============================================================================
Omnibus:                     1157.970   Durbin-Watson:                   1.949
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1643.006
Skew:                          -0.684   Prob(JB):                         0.00
Kurtosis:                       3.990   Cond. No.                         539.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spe
cified.
```

Recommandations Commerciales pour la Production Cinématographique

1 Privilégier certains genres

Analyse :

- Les **genres les plus appréciés** (selon la note moyenne) et les **plus populaires** (nombre de votes, recettes) doivent être ciblés en priorité.
- Croiser la **popularité** (votes, box-office) avec la **qualité perçue** (notes critiques) permet d'identifier les genres rentables et bien notés.

Recommandation :

- Produire des films dans les genres où **succès critique et succès public se chevauchent**.
- Exemples d'orientation stratégique (à adapter selon les données) :
  - **Drame / Thriller** : pour viser les critiques positives.
  - **Action / Science-fiction** : pour attirer un large public.

2 Cibler une durée optimale

Analyse :

- La durée du film influence à la fois la satisfaction des spectateurs et la rentabilité :
  - **90 à 120 minutes** : plage optimale observée dans les données analysées.
  - Les films trop courts (< 80 min) ou trop longs (> 150 min) peuvent réduire l'intérêt du public ou limiter le nombre de séances en salle.

Recommandation :

- Viser une durée comprise entre **95 et 120 minutes**.
- Adapter la durée en fonction du genre :
  - **Action / Comédie** : 90–110 minutes.
  - **Drame / Thriller** : 110–130 minutes.

3 Construire des équipes solides

Analyse :

- Certains **réalisateurs et acteurs** apparaissent régulièrement dans les films les plus rentables ou les mieux notés.
- L'analyse des collaborations (réalisateur, acteur principal) permet d'identifier des profils "gagnants".

Recommandation :

- Miser sur des **équipes récurrentes ayant déjà prouvé leur succès**.
- Prioriser les professionnels avec un historique de films bien notés ou populaires pour sécuriser la performance future.

Conclusion

Conclusion et Prochaines Étapes

Synthèse des Résultats Clés

- Certains **genres** se distinguent par leur combinaison de **popularité** (nombre de votes, box-office) et de **qualité perçue** (note moyenne).
- Une **durée optimale** de film autour de **90 à 120 minutes** semble associée à de meilleures performances critiques et commerciales.
- Les **acteurs et réalisateurs récurrents** dans les films à succès peuvent être identifiés pour orienter les castings futurs.

- La corrélation entre durée, note et popularité reste modérée mais significative pour certains segments.

Limites de l'Analyse

- Les données IMDB et box-office utilisées couvrent principalement des films passés : **les tendances récentes ou émergentes peuvent ne pas être capturées**.
- Certains films ont des **données manquantes** (durée, genres, recettes brutes), ce qui peut biaiser les résultats.
- L'analyse ne tient pas compte de facteurs qualitatifs importants : **scénario, innovation, contexte de sortie, plateformes de streaming, etc.**

Pistes pour des Analyses Futures

1. **Utiliser l'API TMDB ou OMDB**
    - Pour récupérer des données plus récentes ou plus détaillées (budget, casting complet, synopsis, etc.).
2. **Inclure des données financières externes**
    - Budget de production
    - ROI (retour sur investissement)
    - Données par région ou par plateforme de diffusion (Netflix, Prime, etc.)
3. **Analyser les tendances récentes du marché**
    - Évolution des genres populaires après 2023
    - Impact des plateformes de streaming sur la durée optimale ou les formats préférés (séries vs films courts)
4. **Étendre l'analyse à l'international**
    - Comparer les préférences entre les marchés (Amérique, Europe, Asie, etc.)

Prochaine Étape

- Intégrer ces analyses dans un tableau de bord interactif (ex : **Streamlit, Dash**)
- Automatiser la collecte de données pour suivre l'évolution des tendances en temps réel
- Affiner les recommandations commerciales avec des données complémentaires

In [ ]: