Design a recommender system on MovieLens Database

M2 Big Data and Machine Learning en Apprentissage, EFREI

Année universitaire 2022/2023

Mémoire préparé sous la direction de : FALIH Issam (Phd)

Élèves : ANGELOT Quentin / FARAMOND Emile

Introduction

Dans le cadre de notre M2 réalisé à l'EFREI Paris, nous avons réalisé un projet de Big data en utilisant pyspark. Nous avons travaillé sur les données de Movie Dataset disponible sur Kaggle.

Nous verrons dans ce rapport comment répondre aux questions suivantes :

1. Analyse de l'ensemble des données.
2. Modèles de régression pour prédire les *recettes des films* et les *moyennes des votes*.
3. Utilisez le filtrage collaboratif pour construire un système de recommandation de films avec deux fonctions :
   3.1. Suggérer les N premiers films similaires à un titre de film donné.
   3.2. Prédire l'évaluation des utilisateurs pour les films qu'ils n'ont pas évalués.

Le code source du projet est disponible au lien suivant : https://github.com/emilefrd/Design-a-recommender-system-on-MovieLens-Dataset

Remarque : Le code a été pensé pour être industrialisé (commentaire de fonction et fichiers séparés).

Table des matières

Présentation des données

Pour ce projet nous avons utilisé le jeu de données The Movies Dataset. Il est disponible au lien suivant : *https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset*.

Contexte :

Ces fichiers contiennent les métadonnées des 45 000 films répertoriés dans le jeu de données complet MovieLens. Les films présents dans le dataset sont sortis entre 1900 et juillet 2017. Les données contiennent des informations comme :

- les acteurs
- l'équipe
- les mots-clés de l'intrigue
- le budget
- les recettes
- les affiches
- les dates de sortie
- les langues
- les sociétés de production
- les pays
- le nombre de votes sur TMDB
- les moyennes des votes

Ces datasets comportent aussi des fichiers contenant 26 millions d'évaluations de 270 000 utilisateurs pour l'ensemble des 45 000 films. Les évaluations sont des notes (variant de 1 à 5) attribuées par les utilisateurs sur le site officiel de GroupLens.

Différents dataset et contenu :

Fichier	Description

movies_metadata.csv	Contient les informations sur 45 000 films présentés dans le jeu de données MovieLens complet. Les caractéristiques incluent les affiches, les arrière-plans, le budget, les revenus, les dates de sortie, les langues, les pays de production et les sociétés.
keywords.csv	Contient les mots-clés de l'intrigue des films pour nos films MovieLens. Disponible sous forme d'un objet JSON encodé en chaîne.
credits.csv	Se compose d'informations sur le casting et l'équipe pour tous nos films. Disponible sous forme d'un objet JSON encodé en chaîne.
links.csv	Le fichier qui contient les ID TMDB et IMDB de tous les films présentés dans le jeu de données MovieLens complet.
links_small.csv	Contient les ID TMDB et IMDB d'un petit sous-ensemble de 9 000 films du jeu de données complet.
ratings_small.csv	Le sous-ensemble de 100 000 notes de 700 utilisateurs sur 9 000 films.

Intérêt d'utiliser Spark pour ce projet

Spark présenté brièvement

Spark est un moteur de traitement de données distribué qui permet de traiter des volumes massifs de données de manière efficace. Il a été créé pour remplacer les systèmes de traitement de données traditionnels qui ne sont pas adaptés aux besoins de traitement de données à grande échelle. L'utilisation de Spark dans un projet de Big Data présente de nombreux avantages.

Tout d'abord, il permet de traiter des volumes de données considérables en peu de temps, car il utilise une architecture distribuée qui répartit les calculs sur plusieurs nœuds. Cela permet de réduire considérablement les temps de traitement par rapport à des solutions traditionnelles qui ne peuvent pas gérer de telles charges de travail.

De plus, Spark prend en charge de nombreux algorithmes couramment utilisés pour l'analyse de données, tels que ALS et Kmeans, ce qui permet de mener des analyses avancées sans avoir à développer des algorithmes personnalisés. Il prend également en charge de nombreuses librairies de visualisation de données, comme Koala, ce qui permet de visualiser facilement les résultats obtenus.

Une utilisation bénéfique pour notre projet

Dans notre projet, nous avons utilisé le jeu de données Movie Dataset, qui est considérablement grand. L'utilisation de Spark nous a permis de traiter efficacement ces données et de mener des analyses avancées grâce à ses algorithmes intégrés et ses librairies de visualisation de données. Cela a permis d'obtenir des résultats précis et utiles pour notre projet. En résumé, l'utilisation de Spark dans un projet de Big Data et en particulier pour notre projet permet de traiter efficacement des volumes de données considérables en peu de temps, grâce à son architecture distribuée.

1. Exploration de données

Nous avons effectué l'exploration de données dans le notebook : 1_Clean_and_Analysis

1.1. Cleaning de données

Importance : Les données du Movies Dataset sont parfois manquantes, mal renseignées, erronées donc dans notre cas de figure il est très important de faire un nettoyage des données. Cela permet de s'assurer que les données utilisées sont fiables et précises puisque les données brutes peuvent contenir des erreurs, des valeurs manquantes, des valeurs aberrantes, et des incohérences qui peuvent affecter les résultats obtenus lors des phases suivantes du projet. En nettoyant les données, on peut éliminer ces erreurs et incohérences et s'assurer que les résultats obtenus sont fiables.

Cas concret :

Dans notre projet nous avons notamment observé que certains dataset contenaient des valeurs manquantes, dupliquées, valeurs aberrantes. En particulier le dataset movie metadata. Nous allons voir comment traiter ces problèmes.

- Importer les données dans un format bien précis pour éviter les confusions de colonnes. Nous avons renseigné des paramètres comme multiline, header, escape ou encore inferschema.

```
ratings=spark.read\
.format("com.databricks.spark.csv")\
.option("multiline",True)\
.option("header",True)\
.option("escape", "\"")\
.option("inferschema",True)\
.csv("/content/drive/MyDrive/archive/ratings.csv")
```

- Écrit une fonction qui permet d'avoir toutes les informations utiles sur un dataset passé en paramètre (le schéma, le nombre de valeurs nulles, le nombre de colonnes et lignes vides):

```
from pyspark.sql.functions import isnull, count

def dataframe_info(df_list):
    """
    Cette fonction prend en entrée une liste de dataframe et renvoie les informations suivantes pour chaque dataframe:
    - Schéma
    - Nombre de valeurs nulles
    - Description
    - Nombre de colonnes et de lignes pour chaque dataframe
    """
    for i, df in enumerate(df_list):
        print(f"Dataframe {i+1}")
        # Afficher le schéma
        print("Schema:")
        df.printSchema()
        print("\n")

        # Compter le nombre de valeurs nulles
        print("Nombre de valeurs nulles:")
        df.select([count(when(isnull(c), c)).alias(c) for c in df.columns]).show()
        print("\n")

        # Afficher la description
        print("Description:")
        df.describe().show()
        print("\n")

        # Autres informations utiles
        print("Autres informations utiles:")
        print(f"Nombre de lignes: {df.count()}")
        print(f"Nombre de colonnes: {len(df.columns)}")
        print("\n")
```

  - Sur le dataset movie_metada nous avons appliqué la fonction de cleaning suivante qui permet de :
    - Supprimer les lignes avec enregistrement erroné.
    - Supprimer les lignes contenant des valeurs nulles dans les colonnes "production company", "production countries" et "genres".
    - Remplacer la valeur des colonnes avec '[]' par 'Unknown' afin d'éviter tout problème d'analyse Json.
    - Changez le type de de certaines colonnes

```python
def clean_movies_metadata(movies_metadata):
    """
    This function takes a DataFrame of movies metadata as input and performs the following cleaning steps:
    1. Drops rows with corrupted records by id.
    2. Drops rows with null values in "production_companies", "production_countries" and "genres" columns.
    3. Replace columns value with '[]'by 'Unknwon' in order to avoid any Json parsing troubles.
    4. Change the data type of columns 'budget', 'popularity' and 'revenue' to 'integer', 'float' and 'integer' respectively.
    """
    corrupted_ids = ['82663','162372','215848']
    #1_Drops rows with corrupted records by id.
    movies_metadata = movies_metadata.where(~col('id').isin(corrupted_ids))
    #2_Drops rows with null values in "production_companies", "production_countries" and "genres" columns
    movies_metadata = movies_metadata.na.drop(subset=["production_companies","production_countries","genres"])
    #3_Replace columns value with '[]'by 'Unknwon'
    movies_metadata=movies_metadata.withColumn('genres',when(col('genres')=='[]',"[{'id': 0, 'name': 'Unknown'}]").otherwise(col
              .withColumn('production_companies',when(col('production_companies')=='[]',"[{'name': 'Unknown', 'id': 0}]").other
              .withColumn('production_countries',when(col('production_countries')=='[]',"[{'iso_3166_1': 'Unknown', 'name': 'Un
    #4_Cast columns
    movies_metadata = movies_metadata.withColumn('budget', col('budget').cast('integer'))\
                                     .withColumn('popularity', col('popularity').cast('float'))\
                                     .withColumn('revenue', col('revenue').cast('integer'))
```

- Aussi, sur ce même dataset, identification et suppression des colonnes qui apparaissent plusieurs fois :

```python
def identify_and_drop_duplicates(movies_metadata):
    """
    This function takes a DataFrame as input and performs the following steps:
    1. Identifies and shows the duplicate records based on 'imdb_id', 'title', 'release_date' and 'overview' columns
    2. Shows the total number of duplicate records
    3. Drops the duplicate records based on 'imdb_id', 'title', 'release_date' and 'overview' columns
    """
    # Identify duplicate records
    df_dup = movies_metadata.groupby('imdb_id', 'title', 'release_date', 'overview').count().filter("count > 1").show()
    # Show total number of duplicate records
    movies_metadata.groupby('imdb_id','title','release_date', 'overview').count().where(f.col('count')>1).select(f.sum('count')).show()
    # Drop duplicate records
    movies_metadata = movies_metadata.drop_duplicates(['imdb_id', 'title', 'release_date', 'overview'])
```

- Vérifier le nombre de valeur vide par dataset et aussi par colonnes :

```python
def print_null_stats(data_dict):
    """
    This function takes a dictionary of DataFrames as input and prints the null statistics for each DataFrame.
    The keys of the dictionary are used as the names of the DataFrames.
    """
    for key, value in data_dict.items():
        # Create a DataFrame with the null count for each column
        df_stats = value.select([count(when(col(c).isNull() | isnan(c), 'True')).alias(c) for c, c_type in value.dtypes if c_type not in
        print("Column stats for data file: " + key + "\n")
        df_stats.show()
```

Tous ces traitements nous permettent d'obtenir des données fiables, non erronées, bien renseignées et on peut maintenant passer à la suite du projet.

1.2. Parsing de données

Importance : il est important de parser des données JSON en colonnes dans notre projet car cela permet de les organiser de manière plus structurée et facile à utiliser. Les données JSON sont souvent stockées sous forme de structures hiérarchiques de plusieurs niveaux, ce qui peut rendre difficile l'accès et l'analyse des données. En les convertissant en colonnes, les données deviennent plus faciles à utiliser et à analyser, ce qui permet d'optimiser les performances des traitements et des analyses.

Cas concret :

Dans notre projet nous avons plusieurs colonnes à parser comme belong_to_collection, production_companies , production_countries , genres.

- Pour la colonne belong_to_collection nous procédons de cette manière :

On ajoute dans une nouvelle colonne les valeurs initiales qui étaient en json. On utilise pour ça from_json pour convertir les données en type MapType puis ensuite les récupérer avec les méthodes explode et map_keys puis collect.

```
movies_metadata=movies_metadata.withColumnRenamed("id","id_ori")\
                               .withColumnRenamed("poster_path","poster_path_ori")

df=movies_metadata.withColumn("belongs_to_collection_value",from_json(movies_metadata.belongs_to_collection,MapType

key_df=df.select(explode(map_keys(col('belongs_to_collection_value')))).distinct()

keylst=list(map(lambda row:row[0],key_df.collect()))

key_cols=map(lambda f:df['belongs_to_collection_value'].getItem(f).alias(str(f)),keylst)

df=df.select(*movies_metadata.columns,*key_cols)
df.printSchema()
```

- Pour les autres colonnes on créer la fonction suivante qui définit un schéma avec un id et un name, créer une fonction udf pour convertir une liste en série pandas, fait le parsing pour les 3 nouvelles colonnes.

```
def extract_json_array_values(movies_metadata):
    """
    This function takes a DataFrame as input and performs the following steps:
    1. Defines the schema of the JSON array type.
    2. Defines a UDF function to convert list to column separated values.
    3. Parses the JSON array in the "production_companies", "production_countries" and "genres" columns and creates new columns with the
    """
    # Define schema of Json array type
    schema = ArrayType(StructType([
        StructField('id', IntegerType(), nullable=False),
        StructField('name', StringType(), nullable=False)]))

    # UDF function to convert list to column separated values
    convert_udf = udf(lambda s: ','.join(map(str, s)), StringType())

    # Json parsing
    df = movies_metadata.withColumn("production_companies_values", when(col('production_companies') == '[]', '').otherwise(convert_udf(
                        .withColumn("production_countries_values", convert_udf(from_json(movies_metadata.production_countries, schema).
                        .withColumn("genres_value", convert_udf(from_json(movies_metadata.genres, schema).getField("name")))
    df.select('id_ori',"genres_value","production_companies_values","production_countries_values").show(10,False)
```

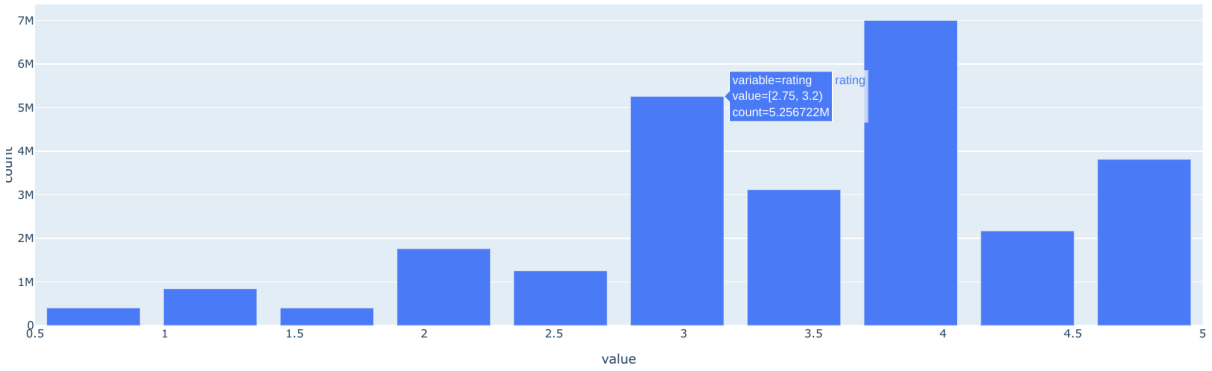
1.3. Quelques statistiques

Importance : Il est important de produire des visualisations graphiques dans notre projet car cela permet de comprendre plus facilement les données et de communiquer les résultats de manière claire et efficace. Surtout que le Data set Movie n'est pas très facile à prendre en main. De plus, les graphiques nous aideront à identifier les tendances, les modèles et les anomalies dans les données plus rapidement et pourront donc nous être utiles pour la suite du projet lors des différents modèles de régression.

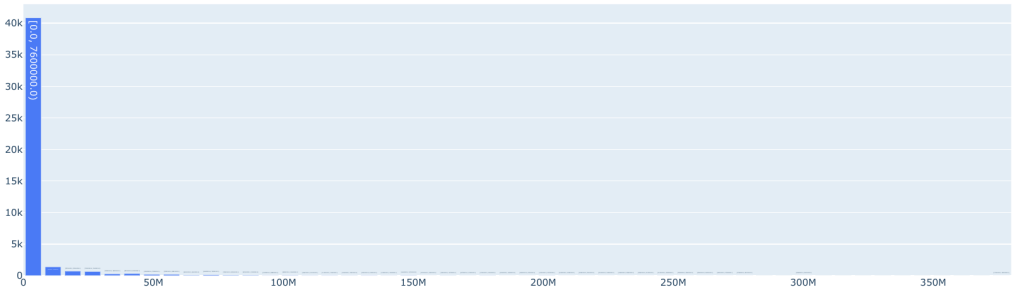
Pour effectuer nos visualisations nous avons utilisé principalement 2 librairies (koalas et matplotlib. Le principal avantage de koalas est que c'est une librairie rapide car distribuée (construite sur spark).

Visualisations Koalas :

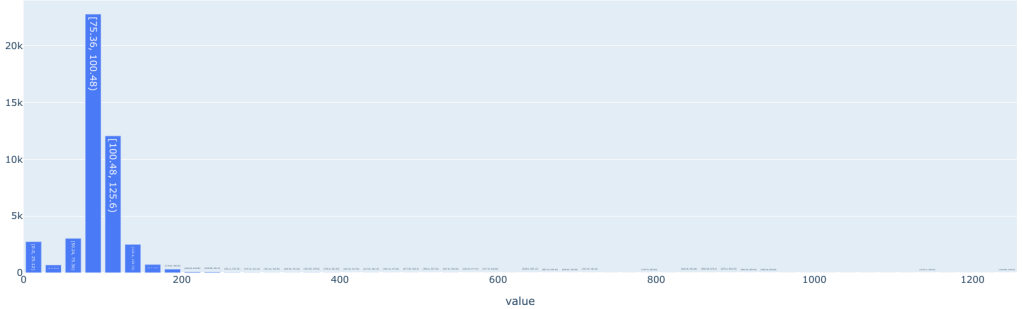
Nombre de votes par notes



Répartition du budget

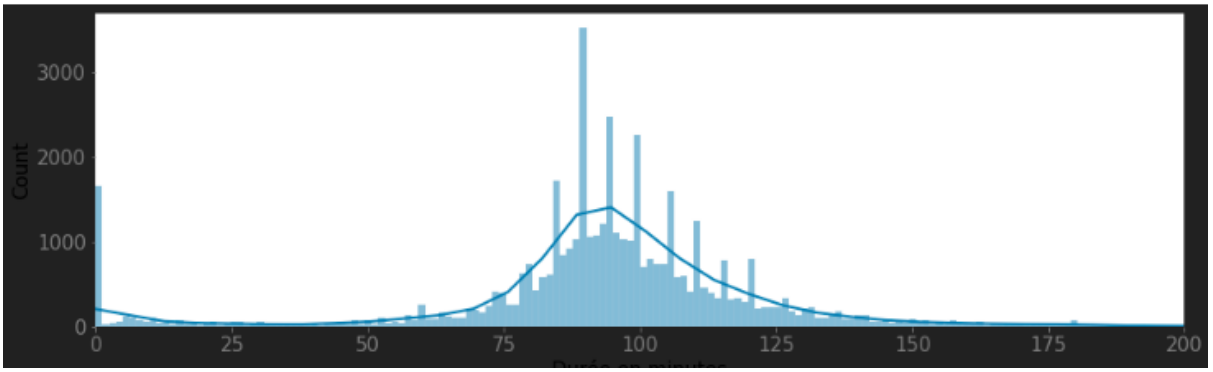


Histogramme du runtime

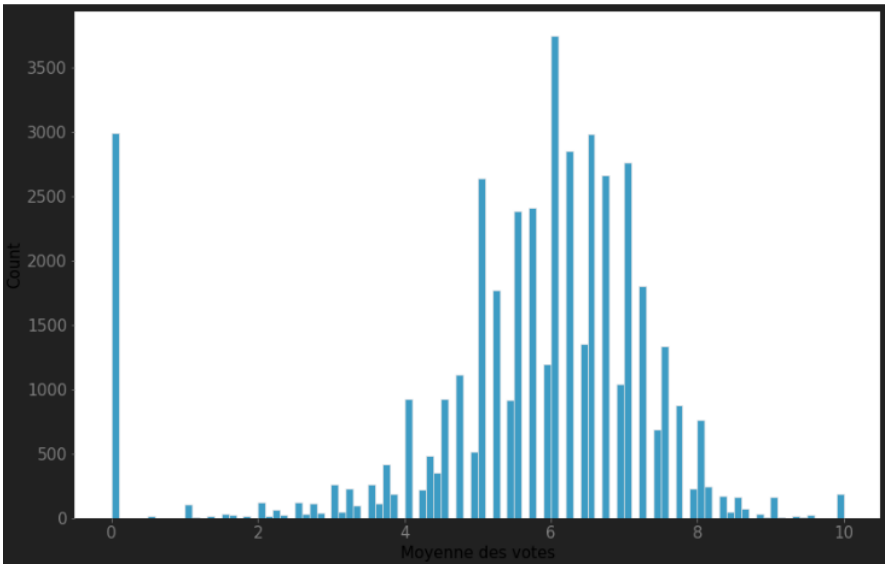


Visualisations Matplotlib :

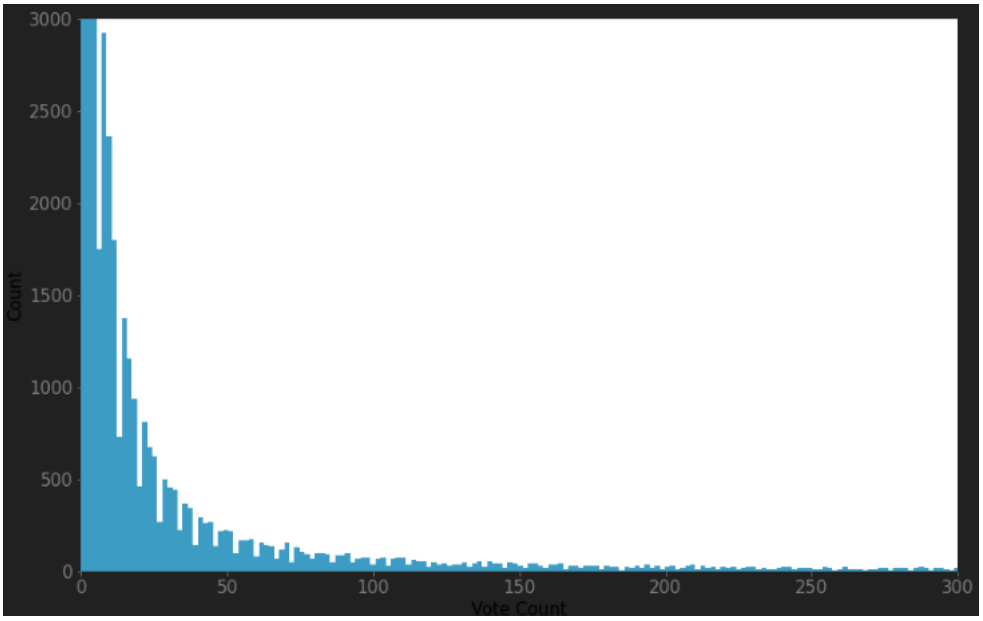
Nombre de film en fonction de la durée du film



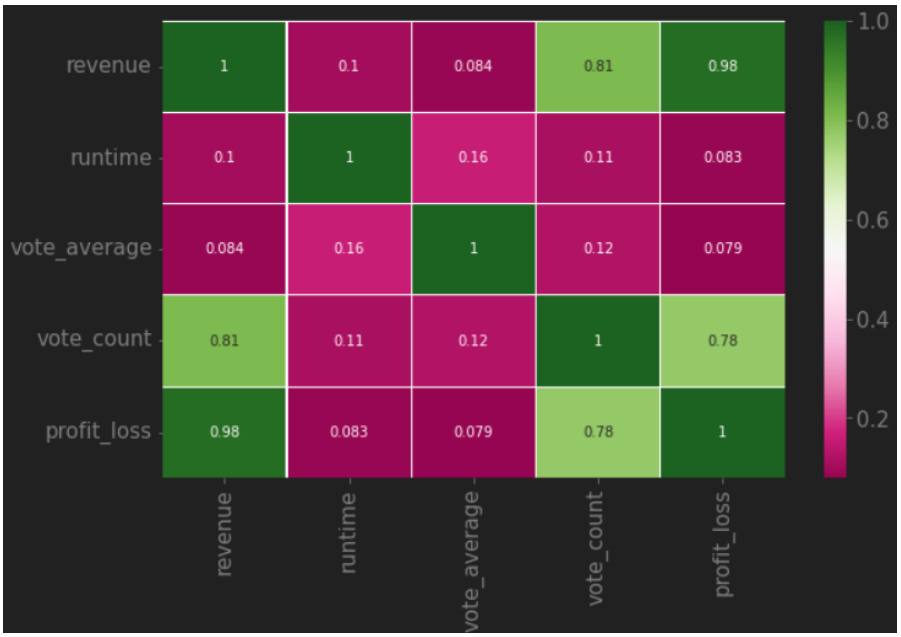
Histogramme des notes moyennes des films



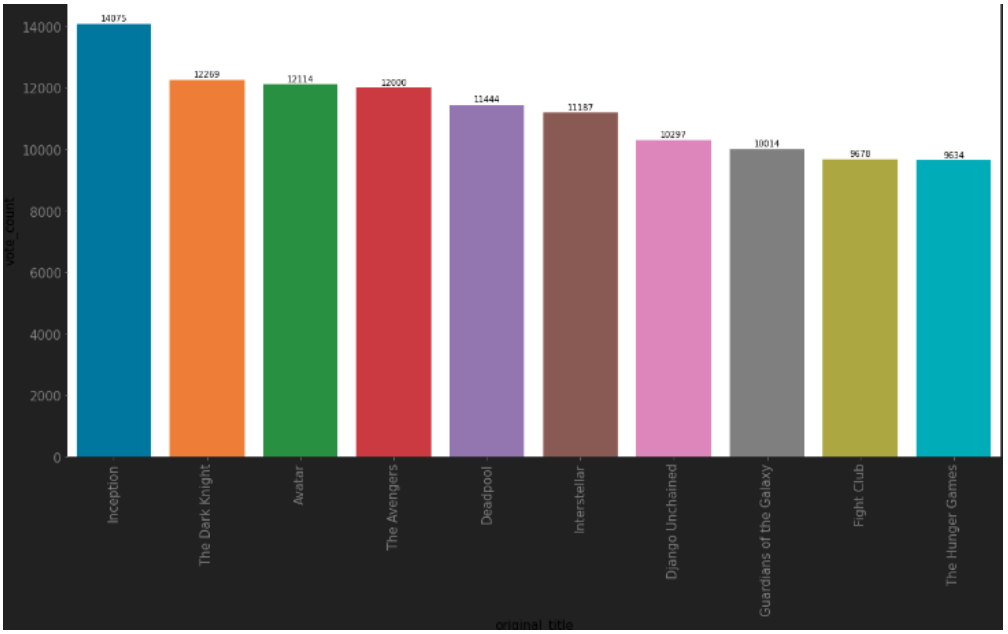
Nombre de films que les utilisateurs ont notés



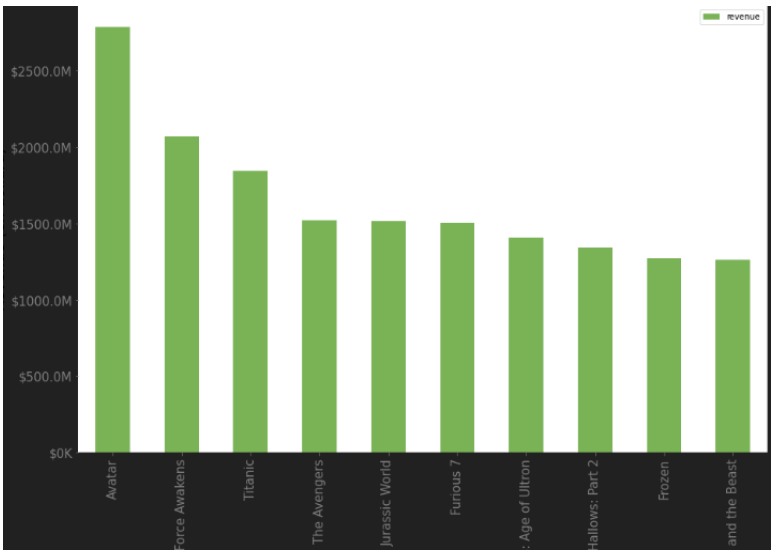
Matrice de corrélation entre différentes variables quantitatives (budget, popularity, revenue, runtime, vote_average, vote_count)



Top 10 des films les plus notés



Top 10 des films ayant eu le plus de revenue



2. Régressions

Nous avons réalisé cette partie dans le notebook : 2_ML_Regressions.ipynb

2.1. Intérêt de notre méthode

La régression en apprentissage automatique permet de prédire une variable quantitative en fonction d'autres variables (appelées variables explicatives ou caractéristiques). Elle est utilisée pour résoudre des problèmes de prédiction de valeurs numériques, tels que la prévision des ventes, la prédiction de la consommation énergétique, etc. Les modèles de régression les plus couramment utilisés incluent la régression linéaire, les forêts aléatoires ou les arbres de décisions boostés.

a) La régression linéaire

La régression linéaire est l'un des modèles les plus simples et les plus utilisés. Il suppose que la relation entre les variables est linéaire. C'est-à-dire qu'il existe une droite qui s'ajuste le mieux aux données pour prédire la variable cible. Il est utilisé pour résoudre des problèmes simples de prédiction, mais il peut ne pas être adapté pour des relations plus complexes.

b) Les forêts aléatoires

Les forêts aléatoires (ou random forests) sont un type de modèle d'apprentissage automatique basé sur l'utilisation de plusieurs arbres de décision. Il est utilisé pour la régression et la classification. Il est souvent considéré comme un outil robuste et efficace pour les problèmes de prédiction. Le fonctionnement d'une forêt aléatoire consiste à construire un grand nombre d'arbres de décision en utilisant des sous-ensembles aléatoires des données d'entraînement. Chaque arbre de décision est ensuite utilisé pour faire des prédictions individuelles, puis les prédictions de tous les arbres sont combinées pour donner une prédiction finale. La combinaison des prédictions des différents arbres permet de réduire l'erreur et d'améliorer la robustesse du modèle. Les forêts aléatoires ont plusieurs avantages pour la régression:

- Elles peuvent gérer des données avec des variables catégoriques et continues
- Elles peuvent gérer des données avec des valeurs manquantes

- Elles peuvent gérer des données avec des interactions et des non-linéarités complexes
- Elles peuvent facilement gérer des grandes quantités de données

Elles donnent une estimation de l'importance des variables qui peut être utilisée pour la sélection de caractéristiques

Pour toutes ses raisons, nous retenons le modèle random forest pour notre projet et notamment car il est capable de gérer des datasets avec des interactions et des non-linéarités complexes.

2.2. Régression pour movie revenue

Il est à noter que pour les deux régressions nous aurons besoin de joindre les datasets metadata et crédits afin d'inclure davantage d'informations pour nos modèles.

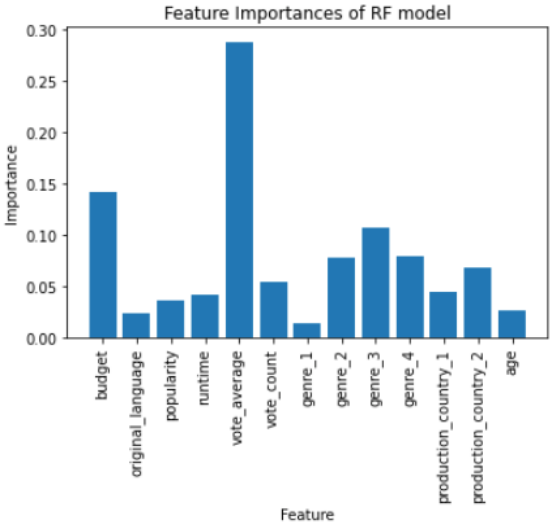
```
df = metadata_ml.join(credits_ml, metadata_ml.id==credits_ml.creditsId, how='inner')
```

Pour prédire le revenu, il nous faut le séparer des autres features et le définir comme la cible de notre modèle. Ensuite, nous utilisons une régression basée sur le modèle des forêts aléatoires, ainsi, il n'est pas nécessaire de normaliser les données car c'est un modèle basé sur les arbres de décisions.

Cependant, il est important de séparer le traitement des variables catégorielles et numériques comme dans l'implémentation suivante :

```
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.regression import GBTRegressor
from pyspark.ml import Pipeline


# Create a StringIndexer for each categorical column
indexers = [StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c)).setHandleInvalid("keep") for c in cat_cols]

# Create a VectorAssembler to combine the numerical and indexed categorical columns
indexed_categorical_cols = [col + "_indexed" for col in cat_cols]
assembler = VectorAssembler(inputCols=num_cols+indexed_categorical_cols, outputCol="features")

# Create a RandomForestRegressor
gbt = GBTRegressor(labelCol="revenue", featuresCol="features", maxBins=150)

# Create a pipeline
pipeline = Pipeline(stages=indexers+[assembler, gbt])

# Split the data into training and test sets (20% held out for testing)
(trainingData, testData) = df_filtered.randomSplit([0.8, 0.2])

# Fit the pipeline on the data
model = pipeline.fit(trainingData)
```

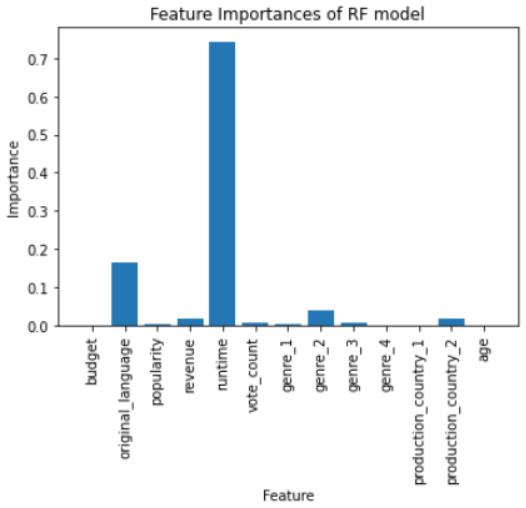
Le modèle random forest permet d'obtenir des informations sur les features les plus informatives pour le modèle et ainsi d'améliorer l'interopérabilité de celui-ci.



2.3. Régression pour vote average

Pour prédire le vote average, il nous faut le séparer des autres features et le définir comme la cible de notre modèle. Ensuite, nous utilisons une régression basée sur le modèle des forêts aléatoires, ainsi, il n'est pas nécessaire de normaliser les données car c'est un modèle basé sur les arbres de décisions.

Le modèle random forest permet d'obtenir des informations sur les features les plus informatives pour le modèle et ainsi d'améliorer l'interopérabilité de celui-ci.



3. Collaborative filtering

Nous avons réalisé cette partie dans les notebooks 3_1 Collaborative Filterings.ipynb et 3_2 Predict Users Ratings.ipynb

3.1. Intérêt du collaborative filtering pour le projet

Le filtrage collaboratif utilise les interactions passées entre les utilisateurs et les articles pour prédire les interactions futures. Il existe deux principaux types de filtrage collaboratif: le filtrage basé sur les utilisateurs et le filtrage basé sur les articles. Filtrage basé sur les utilisateurs: Il utilise les interactions passées des utilisateurs similaires pour prédire les préférences de l'utilisateur cible. Vous pouvez utiliser la similarité des utilisateurs pour trouver les utilisateurs similaires à l'utilisateur cible et utiliser leurs évaluations pour prédire l'évaluation de l'utilisateur cible pour les films qu'ils n'ont pas encore évalués. Filtrage basé sur les articles: Il utilise les interactions passées des articles similaires pour prédire les préférences de l'utilisateur cible. Vous pouvez utiliser la similarité des articles pour trouver les articles similaires à ceux évalués par l'utilisateur cible et utiliser les évaluations de ces articles pour prédire l'évaluation de l'utilisateur cible pour les films qu'ils n'ont pas encore évalués. Il existe également des techniques de filtrage hybrides qui combinent les avantages des deux types de filtrage.

3.2. Construire un algorithme de collaborative filtering from scratch

Le filtrage collaboratif est une technique qui permet de filtrer les éléments qu'un utilisateur pourrait aimer sur la base des réactions d'utilisateurs similaires.

Il fonctionne en recherchant un grand groupe de personnes et en trouvant un plus petit ensemble d'utilisateurs ayant des goûts similaires à ceux d'un utilisateur particulier. Il examine les articles qu'ils aiment et les combine pour créer une liste classée de suggestions.

Pour expérimenter les algorithmes de recommandation, nous avons besoin de données qui contiennent un ensemble d'articles et un ensemble d'utilisateurs qui ont réagi à certains de ces articles.

La matrice utilisateur-article est une matrice dont les lignes représentent les utilisateurs et les colonnes des articles.

Workflow :

1. Créer une matrice utilisateur-élément

```
from pyspark.sql import functions as F

# Create a pivot table with user_id as index, item_id as columns and rating as values
matrix = subset.groupBy("userId").pivot("movieId").agg(F.first("rating"))

# Fill the null values with 0
matrix = matrix.na.fill(0)
```

2. Regrouper les données des utilisateurs et des articles avec deux Kmeans

```
# cluster the users in the user-item matrix into k clusters

# set the number of clusters
k = 10

# fit the model
kmeans = KMeans().setK(k).setSeed(1)
model = kmeans.fit(matrix)
```

```
# cluster the items in the user-item matrix into k clusters

# set the number of clusters
k = 15

# fit the model
kmeans_items = KMeans().setK(k).setSeed(1)
model_items = kmeans_items.fit(matrix_items)
```

3. Système de prédiction : trouver le taux donné par l'utilisateur 2 pour l'article 7.

Une fois que nous avons regroupé les utilisateurs et les articles, nous pouvons donner des recommandations de films à un utilisateur spécifique en recommandant des films qui sont populaires parmi les autres utilisateurs du même groupe que l'utilisateur cible.

- Nous devons trouver le cluster auquel l'utilisateur cible appartient
- Puis trouver les éléments qui sont dans le même cluster que l'utilisateur cible
- Puis filtrer les éléments avec lesquels l'utilisateur cible a déjà interagi.
- Joindre le cadre de données des évaluations avec le cadre de données des films
- Regroupez ensuite les éléments par ID et regroupez les évaluations par moyenne (ou autre statistique).
- Enfin, triez les éléments par note moyenne dans l'ordre décroissant.

```python
def suggest_top_n_movies(target_user_id: int, nb_movies: int):

  # Find the cluster that the target user belongs to
  target_user_cluster = clusters.filter(clusters.userId == target_user_id).select("prediction").first()[0]

  # Find the items that are popular among users in the same cluster as the target user
  cluster_it = clusters_items.filter(clusters_items.prediction == target_user_cluster)

  # Extract the item_id from the cluster_items
  cluster_movies = cluster_it.select('movieId')

  # Filter the items that the target user has not interacted with yet
  # and join the ratings dataframe with the items dataframe
  target_user_items = subset.filter(subset.userId != target_user_id).select('movieId')
  cluster_movies_ = cluster_movies.subtract(target_user_items)

  # join cluster_items (popular items among user's cluster) and ratings_df on movieId column
  cluster_movies = cluster_movies.join(subset, 'movieId')

  # group the items by item_id and aggregate the ratings by average
  cluster_movies = cluster_movies.groupBy("movieId").agg({"rating": "avg"})

  # show the average rating for each item
  cluster_items = cluster_movies.sort(col("avg(rating)").desc())
  cluster_items.show(nb_movies)
```

```python
suggest_top_n_movies(5, 10)
```

```
+-------+-----------+
|movieId|avg(rating)|
+-------+-----------+
|  81156|        5.0|
|  77846|        5.0|
|  85438|        5.0|
|  79677|        5.0|
|  86000|        5.0|
|  86781|       4.75|
|  78574|        4.5|
|  84187|        4.5|
|  89492|        4.5|
|  90428|        4.5|
+-------+-----------+
only showing top 10 rows
```

3.3. Suggestion des N premiers films similaires à un titre de film donné

Pour suggérer les N premiers films similaires à un titre de film donné, on peut utiliser une technique appelée filtrage collaboratif basé sur les éléments. Cette technique utilise les interactions passées entre les éléments pour prédire les interactions futures.

On créer une matrice user-item :

```python
# create the user-item matrix
matrix_movies = subset.groupBy(['userId','movieId']).agg({'rating': 'mean'}).selectExpr("userId", "movieId", "`avg(rating)` as rating")
```

On utilise un modèle ALS (Alternating Least Squares) sur cette matrice pour obtenir un feature vector pour chaque film :

```
# fit the Alternating Least Squares (ALS) model on the user-item matrix
als = ALS(rank=10, maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating", coldStartStrategy="drop")
model = als.fit(matrix_movies)
```

Ensuite, on crée une fonction qui pour un titre donné, un nombre de films à recommander, un dataframe de films et les feature vectors issus d'ALS nous retourne les N films les plus similaires.

1. Trouver feature vector pour le titre de film donné en joignant les feature vectors au dataframe des films
2. Convertir les features en Vector Pyspark
3. Calculez la cosine similarity entre le film donné et tous les autres films
4. Trier les films par similarité avec le film donné et sélectionner les N meilleurs films

La fonction suivante implémente ces fonctionnalités.

```python
def suggest_top_N_movies(given_movie_title, N_movies, movies_df, movie_vectors):

  # Find the feature vector for the given movie title by joining the movie_vectors
  # dataframe with the movie titles dataframe on the item_id column
  given_movie_vector = movie_vectors.join(movies_df, movie_vectors.id == movies_df.id) \
      .filter(movies_df.title == given_movie_title) \
      .select(movie_vectors.features)
  given_movie_vector.show(truncate=False)

  # Define the UDF that converts an array of floats to a vector
  array_to_vector = udf(lambda x: Vectors.dense(x), returnType=VectorUDT())

  # Apply the UDF to the "features" column
  movie_vectors = movie_vectors.withColumn("features", array_to_vector(col("features")))

  # Create a VectorAssembler to combine the feature vectors
  assembler = VectorAssembler(inputCols=['features'], outputCol='vec')
  movie_vectors = assembler.transform(movie_vectors)

  # Same steps for given_movie_vector
  given_movie_vector = given_movie_vector.withColumn("features", array_to_vector(col("features")))
  assembler = VectorAssembler(inputCols=["features"], outputCol="vec")
  given_movie_vector = assembler.transform(given_movie_vector)

  # Define a UDF to compute the cosine similarity
  cosine_similarity = udf(lambda x, y: float(x.dot(y)), DoubleType())

  # Compute the cosine similarity between the given movie and all other movies
  similarities = movie_vectors.alias('mv').crossJoin(given_movie_vector.alias('gv')) \
      .selectExpr('mv.id as item_id', 'gv.vec as vec_given', 'mv.vec as vec') \
      .withColumn('similarity', cosine_similarity(col('vec_given'), col('vec')))

  # sort the movies by similarity to the given movie and select the top N movies
  top_n_movies = similarities.sort(col('similarity').desc()).limit(N_movies)

  # join the top_n_movies with the movie titles dataframe
  top_n_movies = top_n_movies.join(movies_df, top_n_movies.item_id == movies_df.id)
  top_n_movies.show()
```

```python
suggest_top_N_movies('Bad Education', 1, movies_df, movie_vectors)
```

```
+--------------------------------------------------------------------------------------------------------------------------------+
|features                                                                                                                        |
+--------------------------------------------------------------------------------------------------------------------------------+
|[0.96099156, 0.41215488, -1.9007094, -0.344806, -0.87736833, 0.4032965, 0.07572884, -3.1969612, -0.05779577, 0.45853907]|
+--------------------------------------------------------------------------------------------------------------------------------+

+-------+------------------+------------------+-----------------+---+-------+
|item_id|          vec_given|               vec|       similarity| id|  title|
+-------+------------------+------------------+-----------------+---+-------+
|     25|[0.96099156141281...|[3.25526905059814...|24.31481715947114| 25|Jarhead|
+-------+------------------+------------------+-----------------+---+-------+
```

3.4. Prédire l'évaluation des utilisateurs pour les films qu'ils n'ont pas évalués

On prédit ici en fonction d'un utilisateur, les films qu'il est susceptible d'apprécier ainsi que les notes qu'il aurait attribuées à ces films s'il les avait effectivement visionnés.

Cela peut venir du fait que toute l'information contenue dans les autres dataset est aussi contenue dans le dataset ratings, c'est en quelque sorte un résumé.

Cas d'usage :

Pour cet algorithme, nous utilisons uniquement le jeu de données "ratings", qui contient des informations sur les notes données à des films par différents utilisateurs. Ces données comprennent les identifiants des utilisateurs (userId), les identifiants des films (movieId) et les notes données (rating). Ce jeu de données est crucial pour entraîner et évaluer le modèle de régression ALS (Alternating Least Squares) utilisé dans ce code. Il permet de prédire les notes que les utilisateurs donneraient à des films qu'ils n'ont pas encore notés.

Sur ce dataset :

- On sépare le jeu de données en deux parties : une pour l'entraînement (80%) et une pour les tests (20%).
- On créer un modèle de régression appelé ALS (Alternating Least Squares) avec différents paramètres comme : maxIter, regParam, userCol, itemCol et ratingCol.
- On entraîne ensuite les données d'entraînement puis on les teste sur les données de test avec la fonction transform().
- Enfin on évalue la régression : avec la RMSE (Root Mean Squared Error) entre les prédictions et les vraies notes sur les données de test.

```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row

print("Un aperçu du dataframe ratings : ")
ratings.select('userId','movieId','rating','timestamp').show(5)

#Creation du train et du test avec respectivement 80% et 20% du df original ratings
(training, test) = ratings.randomSplit([0.8, 0.2])


als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating",
          coldStartStrategy="drop")

model = als.fit(training)

# Evaluatation du modele en utilsiant la RMSE sur les données test
predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))
```

Remarque sur les paramètres de l'als utilisé :

Nom du paramètre	Valeur dans le code	Description
maxIter	5	Nombre maximum d'itérations pour trouver les facteurs d'utilisateur et d'article.
regParam	0.01	Coefficient de régularisation pour éviter l'overfitting.
userCol	"userId"	Nom de la colonne contenant les identifiants d'utilisateur dans le DataFrame.
itemCol	"movieId"	Nom de la colonne contenant les identifiants d'articles dans le DataFrame.
ratingCol	"rating"	Nom de la colonne contenant les évaluations dans le DataFrame.
coldStartStrategy	"drop"	Stratégie à utiliser lorsqu'un utilisateur ou un article n'a pas d'évaluation. "drop" permet de supprimer ces lignes.

- On stocke tous les résultats sous forme d'une grande matrice pyspark

```
# Top 10 des films recommandés pour chaque utilisateur
userRecs = model.recommendForAllUsers(10)
```

- On créer une fonction qui interroge la matrice créée et on obtient pour un utilisateur les films qu'il est susceptible d'apprécier et une note associée à ce film (estimée)

```
def get_recommended_film_for_given_user(user_id,n):
    line = userRecs_df[userRecs_df['userId'] == user_id]['recommendations']
    df = pd.DataFrame(line.tolist(), columns=["1", "2","3","4","5","6","7","8","9","10"])
    df = df.T
    df = df.rename(columns={0: "Predictions"})
    print(f"Pour l'utilisateur {user_id}, le top {n} des films à voir par ordre de priorité sont les suivants (chaque film a    une n
    return df.head(n)
```

- Par exemple, en appelant la fonction, on peut voir que l'utilisateur 5678 pourrait attribuer la note de 17,4 au film 150667.

```
get_recommended_film_for_given_user(5678,10)
```

```
Pour l'utilisateur 5678, le top 10 des films à voir par ordre de priorité sont les suivants (chaque film a    une note associée :
```

	Predictions
1	(150667, 17.45391273498535)
2	(170731, 15.960968017578125)
3	(152711, 15.3615083694458)
4	(170477, 12.068924903869629)
5	(127126, 11.878308296203613)
6	(96255, 11.138431549072266)

Conclusion

En conclusion, ce projet informatique que nous avons réalisé en tant qu'étudiants à EFREI Paris a été l'occasion pour nous de découvrir l'outil PySpark, un outil de plus en plus utilisé dans l'analyse de données à grande échelle.

Nous avons pu mettre en pratique nos connaissances théoriques en utilisant cet outil pour réaliser une analyse complète d'un jeu de données, incluant le nettoyage des données, la visualisation des données, le parsing et l'implémentation de modèles d'apprentissage automatique tels que ALS et Random Forest Regressor.

Les résultats obtenus ont été probants et nous ont permis de démontrer l'efficacité de PySpark dans l'analyse de données à grande échelle et l'implémentation de modèles d'apprentissage automatique.

Cette expérience a été très enrichissante pour nous car elle nous a permis de mettre en pratique nos connaissances acquises lors de notre formation, et de découvrir les possibilités offertes par PySpark. Nous sommes convaincus de l'importance de cet outil pour les professionnels de l'analyse de données et de l'apprentissage automatique et nous espérons pouvoir continuer à explorer ses capacités dans l'avenir.