PARIS PANTHÉON-ASSAS UNIVERSITÉ

Efrei Paris

Big Data Frameworks II
Big Data Processing
Rapport

Design a recommender system on MovieLens Dataset

Élèves :
Arthur Allie
Wassim Zouitene

Enseignant :
Issam Falih

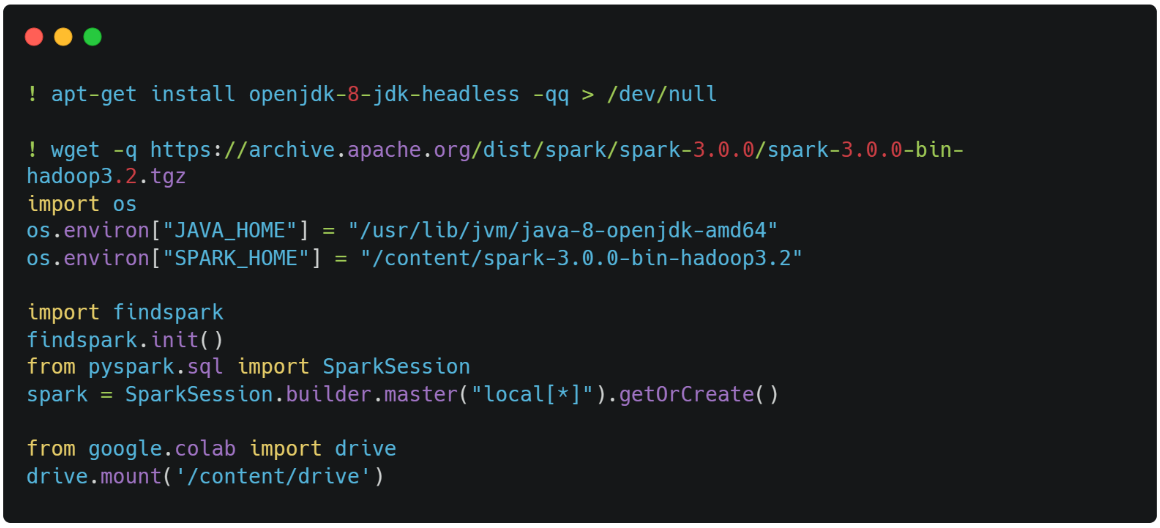
22 janvier 2023

Table des matières

1 Analysis of the dataset

1.1 Running PySpark on Colab

We start to setup the environement by installing Java and Spark and mount the content.

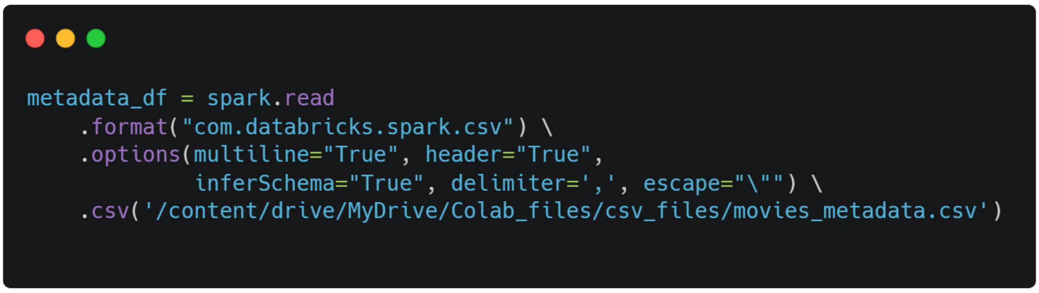


```
! apt-get install openjdk-8-jdk-headless -qq > /dev/null

! wget -q https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop3.2.tgz
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop3.2"

import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()

from google.colab import drive
drive.mount('/content/drive')
```

FIGURE 1 – Code

1.2 Reading the data

We loading the dataframes from : https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset

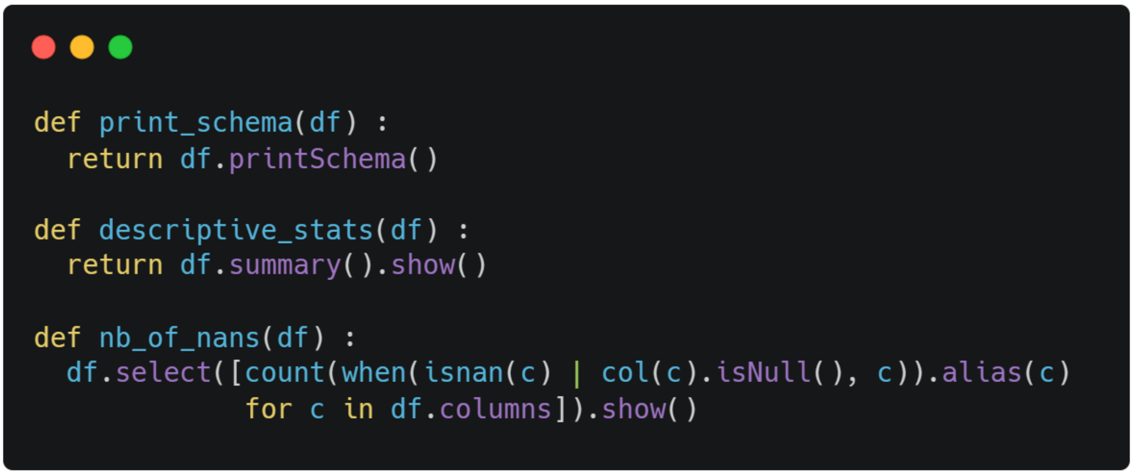


```
metadata_df = spark.read
    .format("com.databricks.spark.csv") \
    .options(multiline="True", header="True",
             inferSchema="True", delimiter=',', escape="\"") \
    .csv('/content/drive/MyDrive/Colab_files/csv_files/movies_metadata.csv')
```

FIGURE 2 – Reading from the metadata dataset

1.3 Exploratory Data Analysis

We did somes analysis such as the schema or the number of missing values for each dataframe.

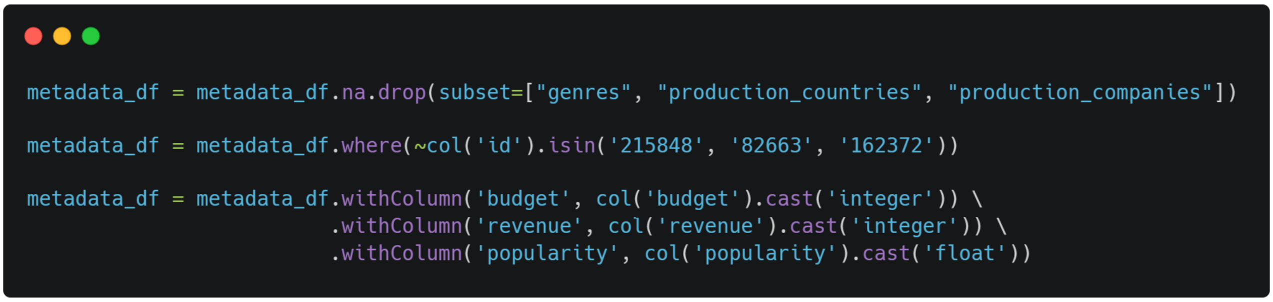


```
def print_schema(df) :
  return df.printSchema()

def descriptive_stats(df) :
  return df.summary().show()

def nb_of_nans(df) :
  df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c)
             for c in df.columns]).show()
```

FIGURE 3 – Extrait de code

1.4 Data cleaning

We did somes cleaning such as handling nans or the type of the data for each dataframe or/and columns .



```
metadata_df = metadata_df.na.drop(subset=["genres", "production_countries", "production_companies"])

metadata_df = metadata_df.where(~col('id').isin('215848', '82663', '162372'))

metadata_df = metadata_df.withColumn('budget', col('budget').cast('integer')) \
                         .withColumn('revenue', col('revenue').cast('integer')) \
                         .withColumn('popularity', col('popularity').cast('float'))
```

FIGURE 4 – Code

1.5 Some visualizations

After joining the useful dataframes we can show some visualization to have a glimpse of the data.

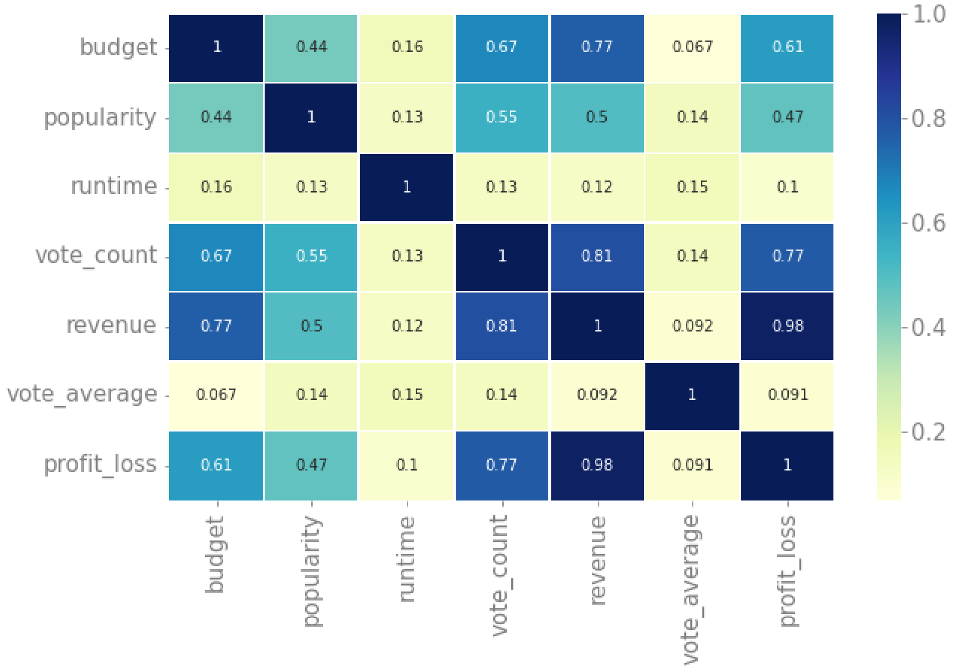


FIGURE 5 – Coorelation between attributes

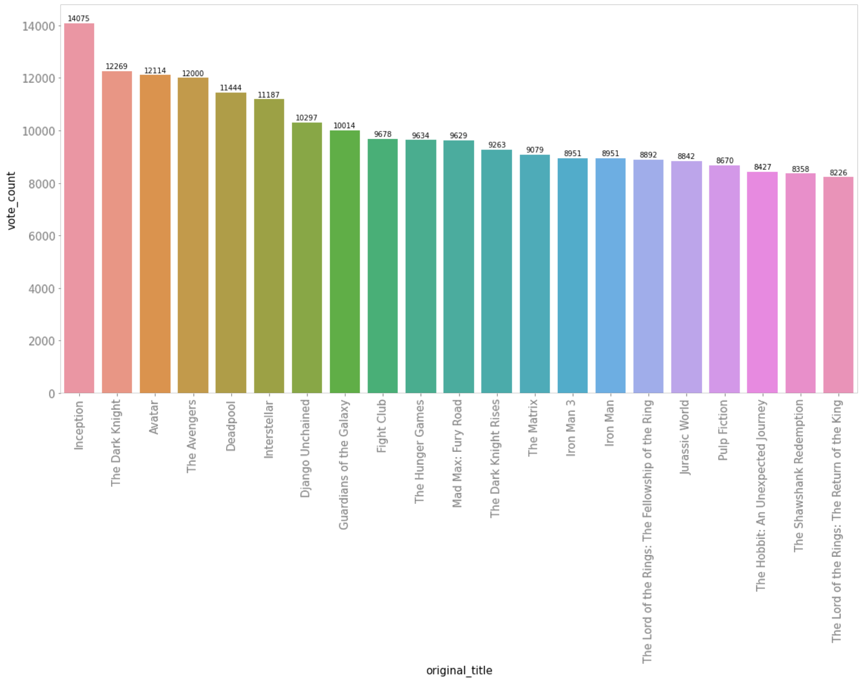


FIGURE 6 – Top 10 voted movies

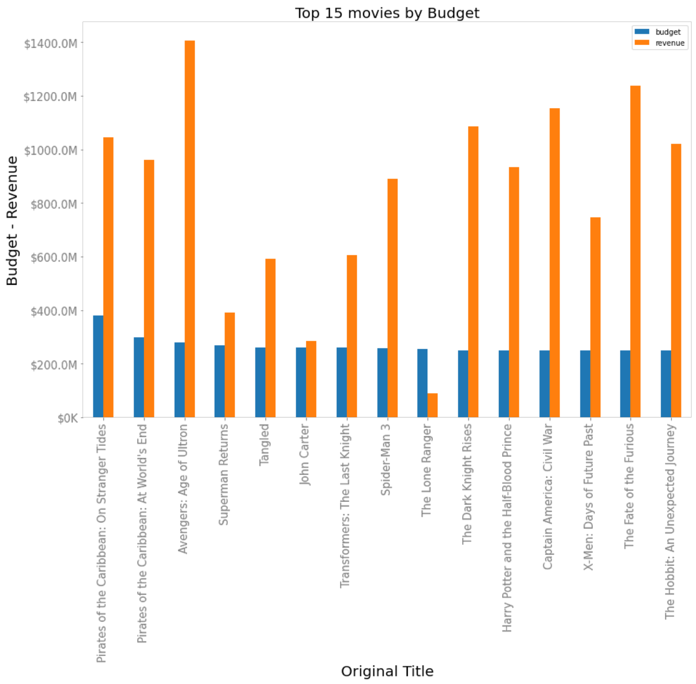
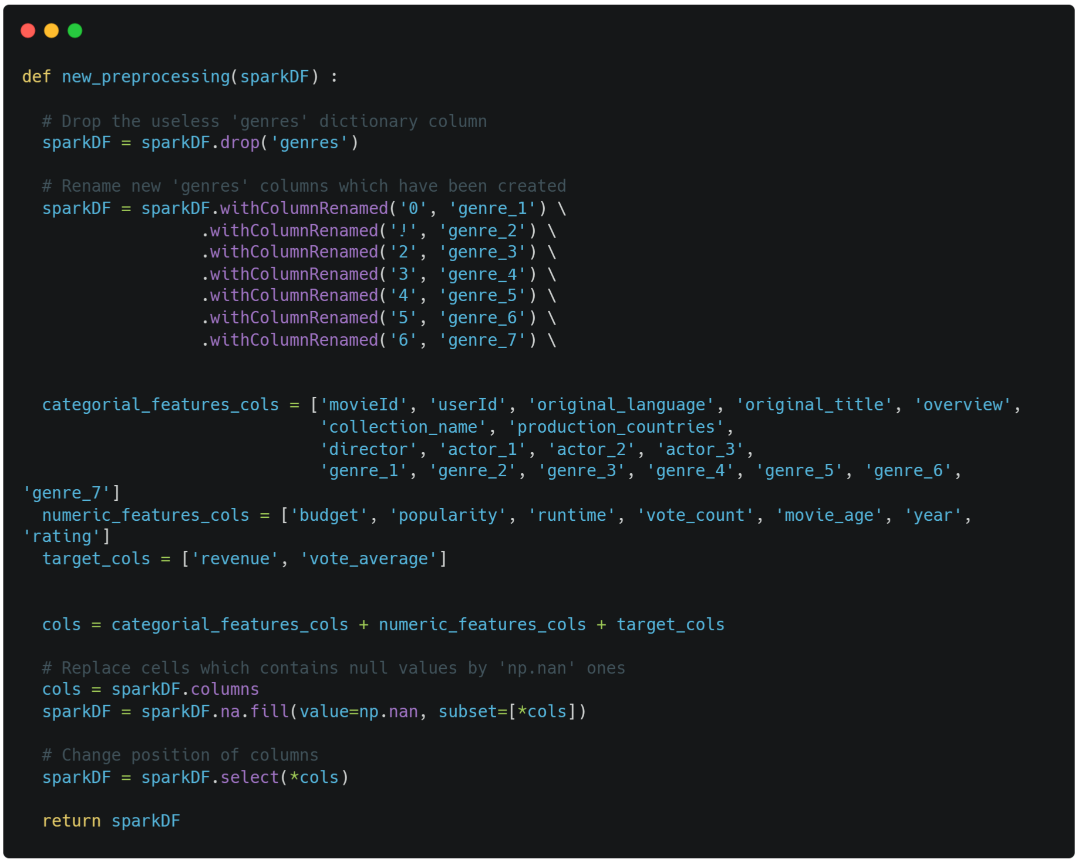


FIGURE 7 – Top 15 movies by Budget

2 Regression model

2.1 Preprocessing

We did some manipulation and dropping of data such as converting the values in column into a dictionary of values or handling corrupt and nans.

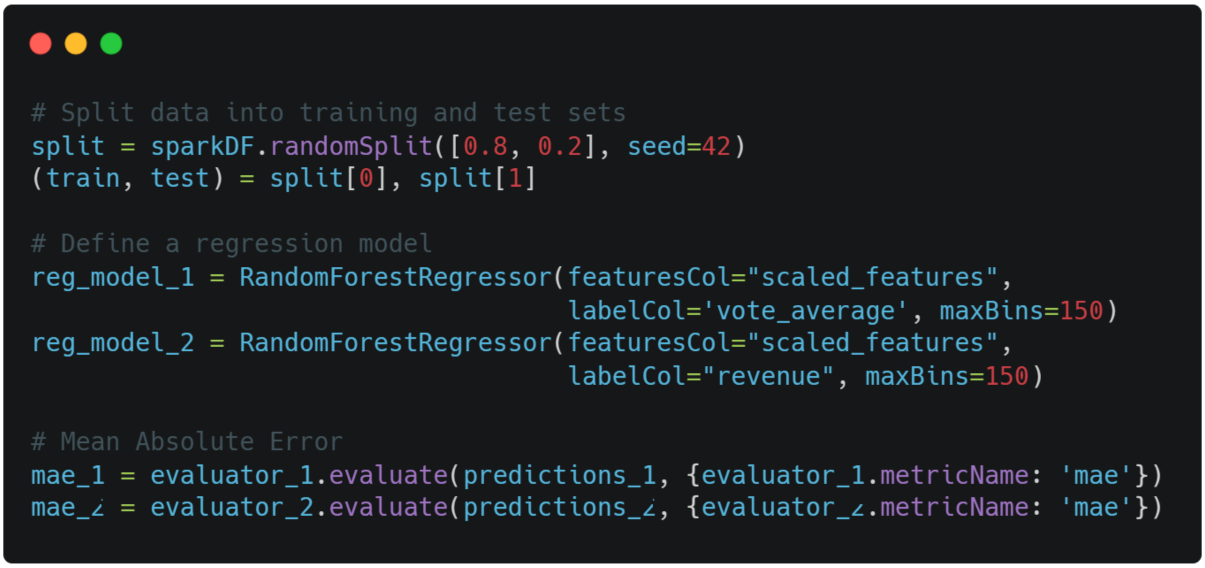


```
def new_preprocessing(sparkDF) :

  # Drop the useless 'genres' dictionary column
  sparkDF = sparkDF.drop('genres')

  # Rename new 'genres' columns which have been created
  sparkDF = sparkDF.withColumnRenamed('0', 'genre_1') \
                  .withColumnRenamed('1', 'genre_2') \
                  .withColumnRenamed('2', 'genre_3') \
                  .withColumnRenamed('3', 'genre_4') \
                  .withColumnRenamed('4', 'genre_5') \
                  .withColumnRenamed('5', 'genre_6') \
                  .withColumnRenamed('6', 'genre_7') \


  categorial_features_cols = ['movieId', 'userId', 'original_language', 'original_title', 'overview',
                              'collection_name', 'production_countries',
                              'director', 'actor_1', 'actor_2', 'actor_3',
                              'genre_1', 'genre_2', 'genre_3', 'genre_4', 'genre_5', 'genre_6',
'genre_7']
  numeric_features_cols = ['budget', 'popularity', 'runtime', 'vote_count', 'movie_age', 'year',
'rating']
  target_cols = ['revenue', 'vote_average']


  cols = categorial_features_cols + numeric_features_cols + target_cols

  # Replace cells which contains null values by 'np.nan' ones
  cols = sparkDF.columns
  sparkDF = sparkDF.na.fill(value=np.nan, subset=[*cols])

  # Change position of columns
  sparkDF = sparkDF.select(*cols)

  return sparkDF
```

FIGURE 8 – Exemple of pre-processing

2.2 PySpark & MLlib *RandomForestRegressor* model

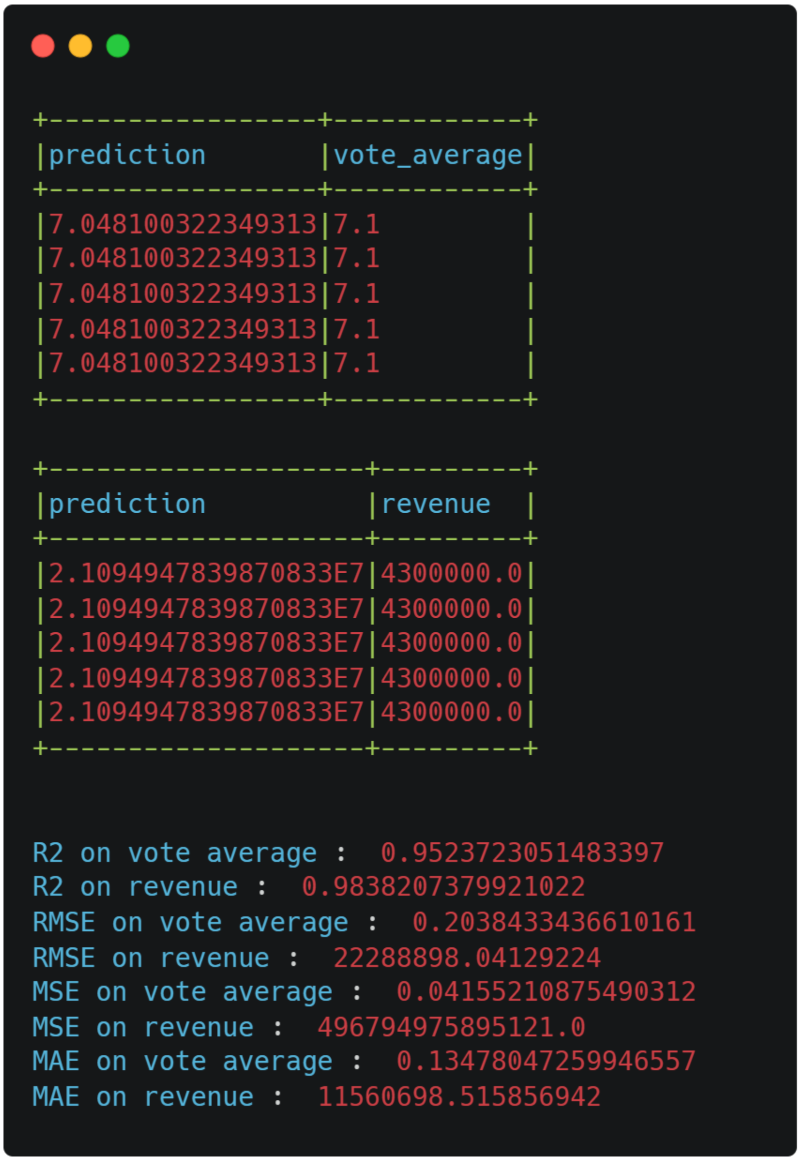
Let's try with PySpark & MLlib *RandomForestRegressor* model



```
# Split data into training and test sets
split = sparkDF.randomSplit([0.8, 0.2], seed=42)
(train, test) = split[0], split[1]

# Define a regression model
reg_model_1 = RandomForestRegressor(featuresCol="scaled_features",
                                    labelCol='vote_average', maxBins=150)
reg_model_2 = RandomForestRegressor(featuresCol="scaled_features",
                                    labelCol="revenue", maxBins=150)

# Mean Absolute Error
mae_1 = evaluator_1.evaluate(predictions_1, {evaluator_1.metricName: 'mae'})
mae_2 = evaluator_2.evaluate(predictions_2, {evaluator_2.metricName: 'mae'})
```

Figure 9 – Code

Some results :

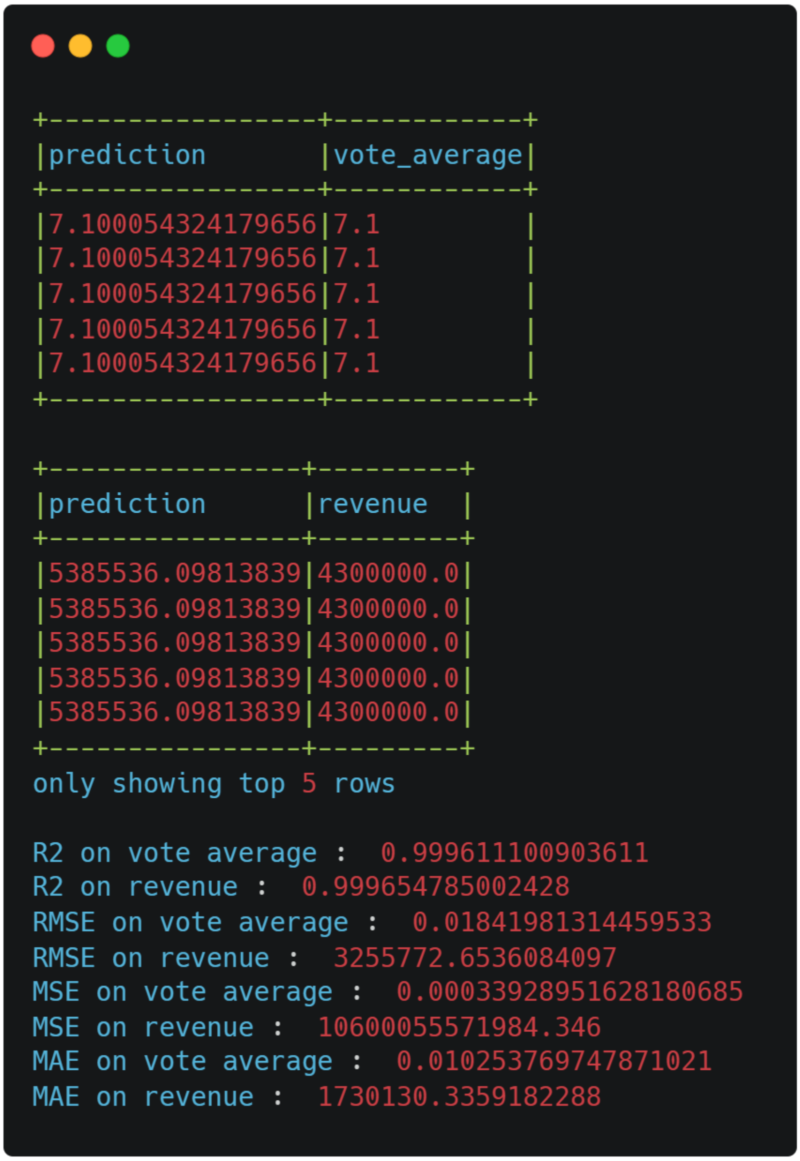


```
+-----------------+------------+
|prediction       |vote_average|
+-----------------+------------+
|7.048100322349313|7.1         |
|7.048100322349313|7.1         |
|7.048100322349313|7.1         |
|7.048100322349313|7.1         |
|7.048100322349313|7.1         |
+-----------------+------------+

+--------------------+---------+
|prediction          |revenue  |
+--------------------+---------+
|2.1094947839870833E7|4300000.0|
|2.1094947839870833E7|4300000.0|
|2.1094947839870833E7|4300000.0|
|2.1094947839870833E7|4300000.0|
|2.1094947839870833E7|4300000.0|
+--------------------+---------+


R2 on vote average :  0.9523723051483397
R2 on revenue :  0.9838207379921022
RMSE on vote average :  0.2038433436610161
RMSE on revenue :  22288898.04129224
MSE on vote average :  0.04155210875490312
MSE on revenue :  496794975895121.0
MAE on vote average :  0.13478047259946557
MAE on revenue :  11560698.515856942
```

FIGURE 10 – Results

2.3 PySpark & MLlib *GBTRegressor* model

Let's try with PySpark & MLlib *GBTRegressor* model and see some results :



```
+-----------------+------------+
|prediction       |vote_average|
+-----------------+------------+
|7.100054324179656|7.1         |
|7.100054324179656|7.1         |
|7.100054324179656|7.1         |
|7.100054324179656|7.1         |
|7.100054324179656|7.1         |
+-----------------+------------+

+----------------+---------+
|prediction      |revenue  |
+----------------+---------+
|5385536.09813839|4300000.0|
|5385536.09813839|4300000.0|
|5385536.09813839|4300000.0|
|5385536.09813839|4300000.0|
|5385536.09813839|4300000.0|
+----------------+---------+
only showing top 5 rows

R2 on vote average :  0.999611100903611
R2 on revenue :  0.999654785002428
RMSE on vote average :  0.01841981314459533
RMSE on revenue :  3255772.6536084097
MSE on vote average :  0.00033928951628180685
MSE on revenue :  10600055571984.346
MAE on vote average :  0.010253769747871021
MAE on revenue :  1730130.3359182288
```

Figure 11 – Results

3 Collaborative filtering

3.1 Suggest top N movies similar to a given movie title

3.1.1 K-Means algorithm

All explanations are provided step-by-step in the code to facilitate reading and comprehension.

3.1.2 ALS algorithm

We tried. Although our evaluation could be better, we have reduced the study dataframe to the maximum (1000 rows) to be able to train the model without it running too long. For more details, see the code.

3.2 Predict user rating for the movies they have not rated for

As same as top N movies, all explanations are provided step-by-step in the code to facilitate reading and comprehension.