Spark Preparation

We check if we are in Google Colab. If this is the case, install all necessary packages.

To run spark in Colab, we need to first install all the dependencies in Colab environment i.e. Apache Spark 3.3.2 with hadoop 3.3, Java 8 and Findspark to locate the spark in the system. The tools installation can be carried out inside the Jupyter Notebook of the Colab. Learn more from A Must-Read Guide on How to Work with PySpark on Google Colab for Data Scientists!

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/dri
```

```
try:
  import google.colab
  IN_COLAB = True
except:
  IN_COLAB = False
```

```
if IN_COLAB:
    !apt-get install openjdk-8-jdk-headless -qq > /dev/null
    !wget -q https://archive.apache.org/dist/spark/spark-3.3.2/spark-3.3.2-bin-hadoop3.tgz
    !tar xf spark-3.3.2-bin-hadoop3.tgz
    !mv spark-3.3.2-bin-hadoop3 spark
    !pip install -q findspark
    import os
    os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
    os.environ["SPARK_HOME"] = "/content/spark"
```

```
mv: cannot move 'spark-3.3.2-bin-hadoop3' to 'spark/spark-3.3.2-bin-hadoop3': Directory not empty
```

Start a Local Cluster

```
import findspark
findspark.init()

from pyspark.sql import SparkSession

spark = SparkSession.builder\
        .master("local")\
        .appName("Colab")\
        .config('spark.ui.port', '4050')\
        .getOrCreate()

spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
    v3.3.2
Master
    local[*]
AppName
    LocalCluster

```
# Verify Spark Configuration
sc = spark.sparkContext
print(f"Spark version: {spark.version}")
print(f"Master: {sc.master}")
print(f"Number of executors: {sc.defaultParallelism}")
```

```
Spark version: 3.3.2
Master: local[*]
Number of executors: 2
```

Spark Assignment

Based on the movie review dataset in 'netflix-rotten-tomatoes-metacritic-imdb.csv', answer the below questions.

Note: do not clean or remove missing data

```
df = spark.read.csv("netflix-rotten-tomatoes-metacritic-imdb.csv", header=True, inferSchema=True)
df.show()
```

```
+--------------------+--------------------+--------------------+--------------------+---------------+---
|               Title|               Genre|                Tags|           Languages|Series or Movie|Hi
+--------------------+--------------------+--------------------+--------------------+---------------+---
|    Lets Fight Ghost|Crime, Drama, Fan...|Comedy Programmes...|    Swedish, Spanish|         Series|
| HOW TO BUILD A GIRL|              Comedy|Dramas,Comedies,F...|             English|          Movie|
|          Centigrade|     Drama, Thriller|           Thrillers|             English|          Movie|
|               ANNE+|               Drama|TV Dramas,Romanti...|             Turkish|         Series|
|               Moxie|Animation, Short,...|Social Issue Dram...|             English|          Movie|
|    The Con-Heartist|     Comedy, Romance|Romantic Comedies...|                Thai|          Movie|
|        Gleboka woda|               Drama|TV Dramas,Polish ...|              Polish|         Series|
|            Instynkt|               Crime|TV Dramas,Crime T...|              Polish|         Series|
|       Only a Mother|               Drama|Social Issue Dram...|             Swedish|          Movie|
|          Snowroller|              Comedy|Sports Movies,Spo...|Swedish, English,...|          Movie|
|        Sunes Summer|Comedy, Family, F...|Children & Family...|             Swedish|          Movie|
|       The Invisible|Crime, Drama, Fan...|Thriller Movies,M...|             English|          Movie|
|The Simple Minded...|               Drama|Social Issue Dram...|    Scanian, Swedish|          Movie|
|The Stig-Helmer S...|       Comedy, Drama|Romantic Dramas,R...|    Swedish, English|          Movie|
|     To Kill a Child|        Short, Drama|Dramas,Swedish Mo...|             Spanish|          Movie|
|               Joker|Crime, Drama, Thr...|Dark Comedies,Cri...|             English|          Movie|
|                   I|Action, Adventure...|Dramas,Swedish Mo...|   English, Sanskrit|          Movie|
|    Harrys Daughters|Adventure, Drama,...|Dramas,Swedish Mo...|             English|          Movie|
|        Gyllene Tider|               Music|Music & Musicals,...|             Swedish|          Movie|
|      False As Water|     Drama, Thriller|Psychological Thr...|             Swedish|          Movie|
+--------------------+--------------------+--------------------+--------------------+---------------+---
only showing top 20 rows
```

What is the maximum and average of the overall hidden gem score?

```
from pyspark.sql.functions import max, avg

max_hidden_gem_score = df.select(max("Hidden Gem Score")).first()[0]
avg_hidden_gem_score = df.select(avg("Hidden Gem Score")).first()[0]

print(f"Maximum Hidden Gem Score: {max_hidden_gem_score}")
print(f"Average Hidden Gem Score: {avg_hidden_gem_score}")
```

```
Maximum Hidden Gem Score: 9.8
Average Hidden Gem Score: 5.937551386501234
```

How many movies that are available in Korea?

```
num_movies_korea = df.filter(df["Languages"].contains("Korea")).count()

print(f"Number of movies available in Korea: {num_movies_korea}")
```

```
Number of movies available in Korea: 735
```

Which director has the highest average hidden gem score?

```
from pyspark.sql.functions import max, desc

directors_df = df.groupby('Director').agg(avg('Hidden Gem Score'))
highest_avg_hidden_gem_director = directors_df.orderBy(desc("avg(Hidden Gem Score)")).first()

director_name = highest_avg_hidden_gem_director["Director"]
highest_avg_hidden_gem_score = highest_avg_hidden_gem_director["avg(Hidden Gem Score)"]

print(f"Director with highest average Hidden Gem Score: {director_name}")
print(f"Highest Average Hidden Gem Score: {highest_avg_hidden_gem_score}")
```

```
Director with highest average Hidden Gem Score: Dorin Marcu
Highest Average Hidden Gem Score: 9.8
```

How many genres are there in the dataset?

```
from pyspark.sql.functions import explode, split, countDistinct

df_genres = df.withColumn("Genre", split("Genre", ",\s*"))
df_exploded = df_genres.select(explode("Genre").alias("Genre"))

unique_count = df_exploded.select(countDistinct("Genre").alias("Unique Genre Count")).first()[0]

print(f"Unique Genre Count: {unique_count}")
```

```
Unique Genre Count: 28
```