

Project_Analysis

December 10, 2024

1 Analysis of IMDB Data

We will analyze a subset of IMDB's actors, genres, movie actors, and movie ratings data. This dataset comes to us from Kaggle (<https://www.kaggle.com/datasets/ashirwadsangwan/imdb-dataset>) although we have taken steps to pull this data into a public S3 bucket:

- s3://cis9760-lecture9-movieanalysis/name.basics.new.tsv —> Name Basics
- s3://cis9760-lecture9-movieanalysis/title.basic.new.tsv —> Title Basics
- s3://cis9760-lecture9-movieanalysis/title.principles.new.tsv —> Title Principles
- s3://cis9760-lecture9-movieanalysis/title.ratings.new.tsv —> Title Ratings

2 Content

name.basics.tsv.gz – Contains the following information for names: nconst (string) - alphanumeric unique identifier of the name/person. primaryName (string)– name by which the person is most often credited. birthYear – in YYYY format. deathYear – in YYYY format if applicable, else . primaryProfession (array of strings)– the top-3 professions of the person. knownForTitles (array of tconsts) – titles the person is known for. **title.basics.tsv.gz - Contains the following information for titles:** tconst (string) - alphanumeric unique identifier of the title. titleType (string) – the type/format of the title (e.g. movie, short, tvseries, tvepisode, video, etc). primaryTitle (string) – the more popular title / the title used by the filmmakers on promotional materials at the point of release. originalTitle (string) - original title, in the original language. isAdult (boolean) - 0: non-adult title; 1: adult title. startYear (YYYY) – represents the release year of a title. In the case of TV Series, it is the series start year. endYear (YYYY) – TV Series end year. for all other title types. runtimeMinutes – primary runtime of the title, in minutes. genres (string array) – includes up to three genres associated with the title. **title.principals.tsv – Contains the principal cast/crew for titles:** tconst (string) - alphanumeric unique identifier of the title. ordering (integer) – a number to uniquely identify rows for a given titleId. nconst (string) - alphanumeric unique identifier of the name/person. category (string) - the category of job that person was in. job (string) - the specific job title if applicable, else. characters (string) - the name of the character played if applicable, else. **title.ratings.tsv.gz – Contains the IMDb rating and votes information for titles:** tconst (string) - alphanumeric unique identifier of the title. averageRating – weighted average of all the individual user ratings. numVotes - number of votes the title has received.

3 PART 1 - Installation and Initial Setup

Begin by installing the necessary libraries that you may need to conduct your analysis. At the very least, you must install pandas and matplotlib

[2] : %%info

```
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
```

[1] : sc.list_packages()

```
VBox()

Starting Spark application

<IPython.core.display.HTML object>

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...

SparkSession available as 'spark'.

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...
```

Package	Version
aws-cfn-bootstrap	2.0
beautifulsoup4	4.9.3
boto	2.49.0
click	8.1.7
docutils	0.14
jmespath	1.0.1
joblib	1.3.2
lockfile	0.11.0
lxml	4.9.3
mysqlclient	1.4.2
nltk	3.8.1
nose	1.3.4
numpy	1.20.0
pip	20.2.2
py-dateutil	2.2
pystache	0.5.4
python-daemon	2.2.3
python37-sagemaker-pyspark	1.4.2
pytz	2023.3
PyYAML	5.4.1
regex	2021.11.10
setuptools	28.8.0

```
simplejson      3.2.0
six             1.13.0
tqdm            4.66.1
wheel            0.29.0
windmill         1.6
```

WARNING: The directory '/home/.cache/pip' or its parent directory is not owned or is not writable by the current user. The cache has been disabled. Check the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.

```
[122]: sc.install_pypi_package("pandas==1.0.5")
sc.install_pypi_package("matplotlib==3.2.1")
```

```
VBox()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...
```

An error was encountered:

Package already installed for current Spark context!

Traceback (most recent call last):

```
  File "/mnt1/yarn/usercache/livy/appcache/application_1733797350059_0001/container_1733797350059_0001_01_000001/pyspark.zip/pyspark/context.py", line 2403, in
install_pypi_package
    raise ValueError("Package already installed for current Spark context!")
ValueError: Package already installed for current Spark context!
```

```
[27]:
```

```
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
```

Let's install the necessary packages here

```
[3]:
```

```
VBox()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...
```

Collecting pandas==1.0.5

 Downloading pandas-1.0.5-cp37-cp37m-manylinux1_x86_64.whl (10.1 MB)

Collecting python-dateutil>=2.6.1

 Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)

Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/site-packages (from pandas==1.0.5) (2023.3)

Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib64/python3.7/site-

```

packages (from pandas==1.0.5) (1.20.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/site-
packages (from python-dateutil>=2.6.1->pandas==1.0.5) (1.13.0)
Installing collected packages: python-dateutil, pandas
Successfully installed pandas-1.0.5 python-dateutil-2.9.0.post0

Collecting matplotlib==3.2.1
  Downloading matplotlib-3.2.1-cp37-cp37m-manylinux1_x86_64.whl (12.4 MB)
Collecting cycler>=0.10
  Downloading cycler-0.11.0-py3-none-any.whl (6.4 kB)
Collecting pyparsing!=2.0.4,!>=2.1.2,!>=2.1.6,>=2.0.1
  Downloading pyparsing-3.1.4-py3-none-any.whl (104 kB)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib64/python3.7/site-
packages (from matplotlib==3.2.1) (1.20.0)
Requirement already satisfied: python-dateutil>=2.1 in ./tmp/spark-
bc6b2510-a0a5-4d1b-83c3-d6764d7c491a/lib/python3.7/site-packages (from
matplotlib==3.2.1) (2.9.0.post0)
Collecting kiwisolver>=1.0.1
  Downloading
kiwisolver-1.4.5-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.whl (1.1 MB)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/site-
packages (from python-dateutil>=2.1->matplotlib==3.2.1) (1.13.0)
Collecting typing-extensions; python_version < "3.8"
  Downloading typing_extensions-4.7.1-py3-none-any.whl (33 kB)
Installing collected packages: cycler, pyparsing, typing-extensions, kiwisolver,
matplotlib
Successfully installed cycler-0.11.0 kiwisolver-1.4.5 matplotlib-3.2.1
pyparsing-3.1.4 typing-extensions-4.7.1

```

WARNING: The directory '/home/.cache/pip' or its parent directory is not owned or is not writable by the current user. The cache has been disabled. Check the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.

WARNING: The directory '/home/.cache/pip' or its parent directory is not owned or is not writable by the current user. The cache has been disabled. Check the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.

Now, import the installed packages from the previous block below.

```
[123]: import pandas as pd
        import matplotlib.pyplot as plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', ▾
    layout=Layout(height='25px', width='50%'), ...
```

4 Loading Data

Load all data from S3 into a Spark dataframe object

```
[124]: name = spark.read.csv('s3://cis9760-lecture9-movieanalysis/name.basics.new.  
    ↪tsv', sep=r'\t', header=True)  
titles = spark.read.csv('s3://cis9760-lecture9-movieanalysis/title.basic.new.  
    ↪tsv', sep=r'\t', header=True)  
principles = spark.read.csv('s3://cis9760-lecture9-movieanalysis/title.  
    ↪principles.new.tsv', sep=r'\t', header=True)  
ratings = spark.read.csv('s3://cis9760-lecture9-movieanalysis/title.ratings.new.  
    ↪tsv', sep=r'\t', header=True)
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    ↪layout=Layout(height='25px', width='50%'),...
```

[49] :

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    ↪layout=Layout(height='25px', width='50%'),...
```

4.1 Name Basics

Display the schema below:

```
[125]: name.printSchema()
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    ↪layout=Layout(height='25px', width='50%'),...
```

```
root  
|-- nconst: string (nullable = true)  
|-- primaryName: string (nullable = true)  
|-- birthYear: string (nullable = true)  
|-- deathYear: string (nullable = true)  
|-- primaryProfession: string (nullable = true)  
|-- knownForTitles: string (nullable = true)
```

[50] :

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    ↪layout=Layout(height='25px', width='50%'),...
```

```

root
|-- nconst: string (nullable = true)
|-- primaryName: string (nullable = true)
|-- birthYear: string (nullable = true)
|-- deathYear: string (nullable = true)
|-- primaryProfession: string (nullable = true)
|-- knownForTitles: string (nullable = true)

```

Display the first 15 rows with the following columns:

- nconst
- primaryName
- primaryProfession
- birthYear

```
[126]: name.select("nconst", "primaryName", "primaryProfession", "birthYear").
    ↪show(15,truncate=False)
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:',
 ↪layout=Layout(height='25px', width='50%'),...

nconst	primaryName	primaryProfession	birthYear
nm0000001	Fred Astaire	soundtrack,actor,miscellaneous	1899
nm0000002	Lauren Bacall	actress,soundtrack	1924
nm0000003	Brigitte Bardot	actress,soundtrack,music_department	1934
nm0000004	John Belushi	actor,soundtrack,writer	1949
nm0000005	Ingmar Bergman	writer,director,actor	1918
nm0000006	Ingrid Bergman	actress,soundtrack,producer	1915
nm0000007	Humphrey Bogart	actor,soundtrack,producer	1899
nm0000008	Marlon Brando	actor,soundtrack,director	1924
nm0000009	Richard Burton	actor,soundtrack,producer	1925
nm0000010	James Cagney	actor,soundtrack,director	1899
nm0000011	Gary Cooper	actor,soundtrack,stunts	1901
nm0000012	Bette Davis	actress,soundtrack,make_up_department	1908
nm0000013	Doris Day	soundtrack,actress,producer	1922
nm0000014	Olivia de Havilland	actress,soundtrack	1916
nm0000015	James Dean	actor,miscellaneous	1931

only showing top 15 rows

```
[51]:
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:',
 ↪layout=Layout(height='25px', width='50%'),...

nconst	primaryName	primaryProfession	birthYear
nm0000001	Fred Astaire	soundtrack,actor,miscellaneous	1899
nm0000002	Lauren Bacall	actress,soundtrack	1924
nm0000003	Brigitte Bardot	actress,soundtrack,music_department	1934
nm0000004	John Belushi	actor,soundtrack,writer	1949
nm0000005	Ingmar Bergman	writer,director,actor	1918
nm0000006	Ingrid Bergman	actress,soundtrack,producer	1915
nm0000007	Humphrey Bogart	actor,soundtrack,producer	1899
nm0000008	Marlon Brando	actor,soundtrack,director	1924
nm0000009	Richard Burton	actor,soundtrack,producer	1925
nm0000010	James Cagney	actor,soundtrack,director	1899
nm0000011	Gary Cooper	actor,soundtrack,stunts	1901
nm0000012	Bette Davis	actress,soundtrack,make_up_department	1908
nm0000013	Doris Day	soundtrack,actress,producer	1922
nm0000014	Olivia de Havilland	actress,soundtrack	1916
nm0000015	James Dean	actor,miscellaneous	1931

only showing top 15 rows

4.2 Title Basics

```
[127]: titles.printSchema()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

root
|-- tconst: string (nullable = true)
|-- titleType: string (nullable = true)
|-- primaryTitle: string (nullable = true)
|-- originalTitle: string (nullable = true)
|-- isAdult: string (nullable = true)
|-- startYear: string (nullable = true)
|-- endYear: string (nullable = true)
|-- runtimeMinutes: string (nullable = true)
|-- genres: string (nullable = true)
```

```
[11]:
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...
```

```

root
|-- tconst: string (nullable = true)
|-- titleType: string (nullable = true)
|-- primaryTitle: string (nullable = true)
|-- originalTitle: string (nullable = true)
|-- isAdult: string (nullable = true)
|-- startYear: string (nullable = true)
|-- endYear: string (nullable = true)
|-- runtimeMinutes: string (nullable = true)
|-- genres: string (nullable = true)

```

Display the first 5 rows with the following columns:

- tconst
- titleType
- primaryTitle
- genres

```
[128]: titles.select("tconst", "titleType", "primaryTitle", "genres").show(5, truncate=False)
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...
```

tconst	titleType	primaryTitle	genres
tt0000001	short	Carmencita	Documentary,Short
tt0000002	short	Le clown et ses chiens	Animation,Short
tt0000003	short	Pauvre Pierrot	Animation,Comedy,Romance
tt0000004	short	Un bon bock	Animation,Short
tt0000005	short	Blacksmith Scene	Comedy,Short

only showing top 5 rows

[52]:

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...
```

tconst	titleType	primaryTitle	genres
tt0000001	short	Carmencita	Documentary,Short
tt0000002	short	Le clown et ses chiens	Animation,Short
tt0000003	short	Pauvre Pierrot	Animation,Comedy,Romance

```
|tt0000004|short    |Un bon bock           |Animation,Short      |
|tt0000005|short    |Blacksmith Scene       |Comedy,Short        |
+-----+-----+-----+
only showing top 5 rows
```

Display the unique title types below:

```
[129]: titleType = titles.select("titleType").distinct()

titleType.show(truncate=False)
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...

+-----+
|titleType   |
+-----+
|tvSeries    |
|tvMiniSeries|
|movie       |
|videoGame   |
|tvSpecial   |
|video       |
|tvMovie     |
|tvEpisode   |
|tvShort     |
|short       |
|tvPilot     |
+-----+
```

```
[53]: 
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...

+-----+
|titleType   |
+-----+
|tvSeries    |
|tvMiniSeries|
|movie       |
|videoGame   |
|tvSpecial   |
|video       |
|tvMovie     |
|tvEpisode   |
+-----+
```

```
|tvShort      |
|short        |
|tvPilot      |
+-----+
```

Display the schema below:

```
[130]: titles.printSchema()
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', □
    layout=Layout(height='25px', width='50%'), ...

root
|-- tconst: string (nullable = true)
|-- titleType: string (nullable = true)
|-- primaryTitle: string (nullable = true)
|-- originalTitle: string (nullable = true)
|-- isAdult: string (nullable = true)
|-- startYear: string (nullable = true)
|-- endYear: string (nullable = true)
|-- runtimeMinutes: string (nullable = true)
|-- genres: string (nullable = true)
```

```
[54] :
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', □
    layout=Layout(height='25px', width='50%'), ...

root
|-- tconst: string (nullable = true)
|-- titleType: string (nullable = true)
|-- primaryTitle: string (nullable = true)
|-- originalTitle: string (nullable = true)
|-- isAdult: string (nullable = true)
|-- startYear: string (nullable = true)
|-- endYear: string (nullable = true)
|-- runtimeMinutes: string (nullable = true)
|-- genres: string (nullable = true)
```

Remove the 'originalTitle' from the dataframe and display the schema to verify it.

```
[131]: titles = titles.drop("originalTitle")
titles.printSchema()
```

```
VBox()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', □
    layout=Layout(height='25px', width='50%'), ...
```

```
root
|-- tconst: string (nullable = true)
|-- titleType: string (nullable = true)
|-- primaryTitle: string (nullable = true)
|-- isAdult: string (nullable = true)
|-- startYear: string (nullable = true)
|-- endYear: string (nullable = true)
|-- runtimeMinutes: string (nullable = true)
|-- genres: string (nullable = true)
```

[55]:

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

root
|-- tconst: string (nullable = true)
|-- titleType: string (nullable = true)
|-- primaryTitle: string (nullable = true)
|-- isAdult: string (nullable = true)
|-- startYear: string (nullable = true)
|-- endYear: string (nullable = true)
|-- runtimeMinutes: string (nullable = true)
|-- genres: string (nullable = true)
```

4.3 Title Principles

Display the schema below:

[132]: principles.printSchema()

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

root
|-- tconst: string (nullable = true)
|-- ordering: string (nullable = true)
|-- nconst: string (nullable = true)
|-- category: string (nullable = true)
|-- job: string (nullable = true)
|-- characters: string (nullable = true)
```

[56]:

```
VBox()
```

```

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

root
|-- tconst: string (nullable = true)
|-- ordering: string (nullable = true)
|-- nconst: string (nullable = true)
|-- category: string (nullable = true)
|-- job: string (nullable = true)
|-- characters: string (nullable = true)

```

Display the first 15 rows where the “category” column is “producer”

[133]: `principles.filter(principles.category == "producer").show(15,truncate=False)`

```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

+-----+-----+-----+-----+-----+
|tconst |ordering|nconst |category|job      |characters|
+-----+-----+-----+-----+-----+
|tt0000003|2     |nm1770680|producer|producer|\N      |
|tt0000005|4     |nm0249379|producer|producer|\N      |
|tt0000007|5     |nm0249379|producer|producer|\N      |
|tt0000020|2     |nm0666972|producer|producer|\N      |
|tt0000024|4     |nm0666972|producer|producer|\N      |
|tt0000025|2     |nm0666972|producer|producer|\N      |
|tt0000039|1     |nm0666972|producer|producer|\N      |
|tt0000041|2     |nm0525908|producer|producer|\N      |
|tt0000061|3     |nm0666972|producer|producer|\N      |
|tt0000089|3     |nm0525910|producer|producer|\N      |
|tt0000104|1     |nm0525910|producer|producer|\N      |
|tt0000121|5     |nm0666972|producer|producer|\N      |
|tt0000125|1     |nm0666972|producer|producer|\N      |
|tt0000147|6     |nm0103755|producer|producer|\N      |
|tt0000160|2     |nm0666972|producer|producer|\N      |
+-----+-----+-----+-----+-----+
only showing top 15 rows

```

[57]:

```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

+-----+-----+-----+-----+-----+
|tconst |ordering|nconst |category|job      |characters|
+-----+-----+-----+-----+-----+

```

```

|tt0000003|2      |nm1770680|producer|producer|\N   |
|tt0000005|4      |nm0249379|producer|producer|\N   |
|tt0000007|5      |nm0249379|producer|producer|\N   |
|tt0000020|2      |nm0666972|producer|producer|\N   |
|tt0000024|4      |nm0666972|producer|producer|\N   |
|tt0000025|2      |nm0666972|producer|producer|\N   |
|tt0000039|1      |nm0666972|producer|producer|\N   |
|tt0000041|2      |nm0525908|producer|producer|\N   |
|tt0000061|3      |nm0666972|producer|producer|\N   |
|tt0000089|3      |nm0525910|producer|producer|\N   |
|tt0000104|1      |nm0525910|producer|producer|\N   |
|tt0000121|5      |nm0666972|producer|producer|\N   |
|tt0000125|1      |nm0666972|producer|producer|\N   |
|tt0000147|6      |nm0103755|producer|producer|\N   |
|tt0000160|2      |nm0666972|producer|producer|\N   |
+-----+-----+-----+-----+
only showing top 15 rows

```

4.4 Title Ratings

Display the schema below:

```
[134]: ratings.printSchema()
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

root
 |-- tconst: string (nullable = true)
 |-- averageRating: string (nullable = true)
 |-- numVotes: string (nullable = true)
```

Display the first 10 rows in a descending order by the number of votes

```
[135]: from pyspark.sql.functions import col, round
ratings = ratings.withColumn("averageRating", round(col("averageRating")...
    .cast("float"), 2))
ratings = ratings.withColumn("numVotes", col("numVotes").cast("int"))
ratings.sort(col("numVotes").desc()).show(10)
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

+-----+-----+-----+
|  tconst|averageRating|numVotes|
+-----+-----+-----+
```

```

|tt0111161|      9.3| 2868594|
|tt0468569|      9.0| 2850372|
|tt1375666|      8.8| 2531543|
|tt0137523|      8.8| 2303989|
|tt0944947|      9.2| 2265760|
|tt0109830|      8.8| 2239746|
|tt0110912|      8.9| 2203191|
|tt0903747|      9.5| 2114358|
|tt0816692|      8.7| 2073181|
|tt0133093|      8.7| 2038364|
+-----+
only showing top 10 rows

```

[59]:

```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...

+-----+-----+-----+
| tconst|averageRating|numVotes|
+-----+-----+-----+
|tt0111161|      9.3| 2868594|
|tt0468569|      9.0| 2850372|
|tt1375666|      8.8| 2531543|
|tt0137523|      8.8| 2303989|
|tt0944947|      9.2| 2265760|
|tt0109830|      8.8| 2239746|
|tt0110912|      8.9| 2203191|
|tt0903747|      9.5| 2114358|
|tt0816692|      8.7| 2073181|
|tt0133093|      8.7| 2038364|
+-----+
only showing top 10 rows

```

5 Overview of Data

Display the number of rows and columns in each DataFrame object.

```

[136]: print("Number of columns in Name Basics table:", len(name.columns))
print("Number of rows in Name Basics table:", name.count())
print("")

print("Number of columns in Title Basics table:", len(titles.columns))
print("Number of rows in Title Basics table:", titles.count())
print("")

```

```
print("Number of columns in Title Principles table:", len(principles.columns))
print("Number of rows in Title Principles table:", principles.count())
print("")  
  
print("Number of columns in Title Ratings table:", len(ratings.columns))
print("Number of rows in Title Ratings table:", ratings.count())
```

```
VBox()  
  
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    layout=Layout(height='25px', width='50%'),...  
  
Number of columns in Name Basics table: 6  
Number of rows in Name Basics table: 13329316  
  
Number of columns in Title Basics table: 8  
Number of rows in Title Basics table: 10613322  
  
Number of columns in Title Principles table: 6  
Number of rows in Title Principles table: 60833800  
  
Number of columns in Title Ratings table: 3  
Number of rows in Title Ratings table: 1412275
```

[60]:

```
VBox()  
  
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    layout=Layout(height='25px', width='50%'),...  
  
Number of columns in Name Basics table: 6  
Number of rows in Name Basics table: 13329316  
  
Number of columns in Title Basics table: 8  
Number of rows in Title Basics table: 10613322  
  
Number of columns in Title Principles table: 6  
Number of rows in Title Principles table: 60833800  
  
Number of columns in Title Ratings table: 3  
Number of rows in Title Ratings table: 1412275
```

6 PART 2 - Analyzing Movie Genres

Let's now answer this question: how many unique movie genres are represented in this dataset?

Essentially, we have the genres per movie as a list - this is useful to quickly see what each movie might be represented as but it is difficult to easily answer questions such as:

- How many movies are categorized as Comedy, for instance?
- What are the top 20 most popular genres available?

6.1 Association Table

We need to “break out” these genres from the tconst? One common approach to take is to build an association table mapping a single tconst multiple times to each distinct genre.

For instance, given the following:

tconst	titleType	genres
abcd123	XXX	a,b,c

We would like to derive something like:

tconst	titleType	genre
abcd123	XXX	a
abcd123	XXX	b
abcd123	XXX	c

What this does is allow us to then perform a myriad of rollups and other analysis on this association table which can aid us in answering the questions asked above.

Implement the code necessary to derive the table described from the data set

```
[33]: titles.printSchema()
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', 
    layout=Layout(height='25px', width='50%'), ...

root
|-- tconst: string (nullable = true)
|-- titleType: string (nullable = true)
|-- primaryTitle: string (nullable = true)
|-- isAdult: string (nullable = true)
|-- startYear: string (nullable = true)
|-- endYear: string (nullable = true)
|-- runtimeMinutes: string (nullable = true)
|-- genres: string (nullable = true)
```

```
[137]: titles_assoc = titles.select("tconst", "titleType", "genres")
titles_assoc.show(15, truncate=False)
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

+-----+-----+
|tconst |titleType|genres           |
+-----+-----+
|tt0000001|short   |Documentary,Short      |
|tt0000002|short   |Animation,Short        |
|tt0000003|short   |Animation,Comedy,Romance|
|tt0000004|short   |Animation,Short        |
|tt0000005|short   |Comedy,Short          |
|tt0000006|short   |Short                 |
|tt0000007|short   |Short,Sport            |
|tt0000008|short   |Documentary,Short      |
|tt0000009|movie    |Romance               |
|tt0000010|short   |Documentary,Short      |
|tt0000011|short   |Documentary,Short      |
|tt0000012|short   |Documentary,Short      |
|tt0000013|short   |Documentary,Short      |
|tt0000014|short   |Comedy,Short          |
|tt0000015|short   |Animation,Short        |
+-----+-----+
only showing top 15 rows
```

[61]:

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

+-----+-----+
|tconst |titleType|genres           |
+-----+-----+
|tt0000001|short   |Documentary,Short      |
|tt0000002|short   |Animation,Short        |
|tt0000003|short   |Animation,Comedy,Romance|
|tt0000004|short   |Animation,Short        |
|tt0000005|short   |Comedy,Short          |
|tt0000006|short   |Short                 |
|tt0000007|short   |Short,Sport            |
|tt0000008|short   |Documentary,Short      |
|tt0000009|movie    |Romance               |
|tt0000010|short   |Documentary,Short      |
|tt0000011|short   |Documentary,Short      |
|tt0000012|short   |Documentary,Short      |
|tt0000013|short   |Documentary,Short      |
|tt0000014|short   |Comedy,Short          |
+-----+-----+
```

```
|tt0000015|short |Animation,Short      |
+-----+-----+-----+
only showing top 15 rows
```

Display the first 25 rows of your association table below

```
[138]: from pyspark.sql.functions import split, col, explode
titles_assoc = titles_assoc.withColumnRenamed("genres", "Genre")
titles_assoc = titles_assoc.withColumn('Genre', explode(split('Genre', ",")))

titles_assoc.show(25, truncate=False)
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', ▾
    layout=Layout(height='25px', width='50%'), ...
```

```
+-----+-----+-----+
|tconst |titleType|Genre      |
+-----+-----+-----+
|tt0000001|short |Documentary|
|tt0000001|short |Short      |
|tt0000002|short |Animation   |
|tt0000002|short |Short      |
|tt0000003|short |Animation   |
|tt0000003|short |Comedy     |
|tt0000003|short |Romance    |
|tt0000004|short |Animation   |
|tt0000004|short |Short      |
|tt0000005|short |Comedy     |
|tt0000005|short |Short      |
|tt0000006|short |Short      |
|tt0000007|short |Short      |
|tt0000007|short |Sport      |
|tt0000008|short |Documentary|
|tt0000008|short |Short      |
|tt0000009|movie |Romance    |
|tt0000010|short |Documentary|
|tt0000010|short |Short      |
|tt0000011|short |Documentary|
|tt0000011|short |Short      |
|tt0000012|short |Documentary|
|tt0000012|short |Short      |
|tt0000013|short |Documentary|
|tt0000013|short |Short      |
+-----+-----+-----+
only showing top 25 rows
```

[]:

[62] :

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    layout=Layout(height='25px', width='50%'),...

+-----+-----+-----+
|tconst |titleType|Genre      |
+-----+-----+-----+
|tt0000001|short   |Documentary|
|tt0000001|short   |Short       |
|tt0000002|short   |Animation   |
|tt0000002|short   |Short       |
|tt0000003|short   |Animation   |
|tt0000003|short   |Comedy      |
|tt0000003|short   |Romance     |
|tt0000004|short   |Animation   |
|tt0000004|short   |Short       |
|tt0000005|short   |Comedy      |
|tt0000005|short   |Short       |
|tt0000006|short   |Short       |
|tt0000007|short   |Short       |
|tt0000007|short   |Sport       |
|tt0000008|short   |Documentary|
|tt0000008|short   |Short       |
|tt0000009|movie    |Romance     |
|tt0000010|short   |Documentary|
|tt0000010|short   |Short       |
|tt0000011|short   |Documentary|
|tt0000011|short   |Short       |
|tt0000012|short   |Documentary|
|tt0000012|short   |Short       |
|tt0000013|short   |Documentary|
|tt0000013|short   |Short       |
+-----+-----+-----+
only showing top 25 rows
```

6.2 Total Unique Movie Genres

What is the total number of unique movie genres?

```
[139]: from pyspark.sql.functions import countDistinct
print(titles_assoc.filter(titles_assoc.titleType == "movie").
    select(countDistinct('Genre')).collect()[0][0])
```

```
VBox()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    layout=Layout(height='25px', width='50%'),...
```

29

[63]:

```
VBox()  
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    layout=Layout(height='25px', width='50%'),...
```

29

What are the unique movie genres?

```
[140]: titles_assoc.filter(titles_assoc.titleType == "movie").select('Genre').  
    distinct().show(30,truncate=False)
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    layout=Layout(height='25px', width='50%'),...
```

Genre
Mystery
Musical
Action
Sport
Talk-Show
Romance
Thriller
\N
Reality-TV
Family
Fantasy
History
Animation
Short
Sci-Fi
News
Drama
Documentary
Western
Comedy
Crime
War
Game-Show
Adult

```
|Music      |
|Biography  |
|Adventure  |
|Horror     |
|Film-Noir  |
+-----+
```

[64]:

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    layout=Layout(height='25px', width='50%'), ...

+-----+
|genre   |
+-----+
|Mystery  |
|Musical   |
|Sport     |
|Action    |
|Talk-Show |
|Romance   |
|Thriller  |
|\N
|Reality-TV|
|Family    |
|Fantasy   |
|History   |
|Animation |
|Film-Noir |
|Short     |
|Sci-Fi    |
|News      |
|Drama     |
|Documentary|
|Western   |
|Comedy    |
|Crime     |
|War       |
|Game-Show |
|Adult     |
|Music     |
|Biography |
|Adventure |
|Horror    |
+-----+
```

Oops! Something is off!

```
[141]: nll = '\\\\N'
titles_assoc = titles_assoc.filter(col("Genre") != nll)
titles_assoc.filter(titles_assoc.titleType == "movie").select('Genre') .
    distinct().show(30,truncate=False)
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...

+-----+
|Genre      |
+-----+
|Mystery   |
|Musical   |
|Sport     |
>Action    |
|Talk-Show |
|Romance   |
|Thriller  |
|Reality-TV|
|Family    |
|Fantasy   |
|History   |
|Animation |
|Film-Noir |
|Short     |
|Sci-Fi    |
|News      |
|Drama     |
|Documentary|
|Western   |
|Comedy   |
|Crime     |
|War       |
|Game-Show |
|Adult     |
|Music     |
|Biography |
|Adventure |
|Horror    |
+-----+
```

```
[65]:
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...
```

```

+-----+
|genre      |
+-----+
|Mystery    |
|Musical    |
|Sport      |
|Action     |
|Talk-Show   |
|Romance    |
|Thriller   |
|Reality-TV |
|Family     |
|Fantasy    |
|History    |
|Animation  |
|Film-Noir  |
|Short      |
|Sci-Fi     |
|News       |
|Drama      |
|Documentary|
|Western    |
|Comedy     |
|Crime      |
|War        |
|Game-Show  |
|Adult      |
|Music      |
|Biography  |
|Adventure  |
|Horror     |
+-----+

```

6.3 Top Genres by Movies

Now let's find the highest rated genres in this dataset by rolling up genres.

6.3.1 Average Rating / Genre

So now, let's unroll our distinct count a bit and display the per average rating value of per genre.

The expected output should be:

genre	averageRating
a	8.5
b	6.3
c	7.2

```
genre    averageRating
```

Or something to that effect.

First, let's join our two dataframes (title ratings and title basics) by tconst. Use inner join.

```
[142]: from pyspark.sql.functions import split, col, explode
titles_df = titles.filter(titles.titleType == "movie")
titles_df = titles_df.withColumnRenamed("genres", "Genre")
titles_df = titles_df.withColumn('Genre', explode(split('Genre', ',')))
titles_df = titles_df.filter(col("Genre") != null)
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', ▾
    layout=Layout(height='25px', width='50%'), ...
```

```
[144]: joined_df = titles_df.join(ratings, "tconst", "inner")

joined_df.select("Genre", "averageRating").show(10)
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', ▾
    layout=Layout(height='25px', width='50%'), ...
```

```
+-----+-----+
|   Genre|averageRating|
+-----+-----+
|   Drama|        4.2|
|   Drama|        4.5|
|Biography|        3.6|
|   Drama|        3.6|
| History|        3.6|
|   Drama|        6.0|
|   Drama|        5.0|
| History|        5.0|
|Biography|        6.2|
|   Drama|        6.2|
+-----+
only showing top 10 rows
```

```
[66]:
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', ▾
    layout=Layout(height='25px', width='50%'), ...
```

```

+-----+-----+
|   Genre|averageRating|
+-----+-----+
|   Drama|      4.2|
|   Drama|      4.5|
|Biography|      3.6|
|   Drama|      3.6|
| History|      3.6|
|   Drama|      6.0|
|   Drama|      5.0|
| History|      5.0|
|Biography|      6.2|
|   Drama|      6.2|
+-----+
only showing top 10 rows

```

Now, let's aggregate along the averageRating column to get a resultant dataframe that displays average rating per genre.

```
[145]: from pyspark.sql.functions import col, avg, round
average_rating_per_genre = (
    joined_df.groupBy("Genre")
    .agg(round(avg(col("averageRating")), 3).alias("Rating"))
)

average_rating_per_genre.show(truncate=False)
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', ...)
layout=Layout(height='25px', width='50%'), ...
```

```

+-----+-----+
|Genre      |Rating|
+-----+-----+
|Mystery    |5.847 |
|Musical    |6.187 |
>Action     |5.732 |
|Sport      |6.623 |
|Talk-Show  |6.858 |
|Romance    |6.102 |
|Thriller   |5.613 |
|Reality-TV |6.701 |
|Family     |6.205 |
|Fantasy    |5.898 |
|History    |6.798 |
|Animation  |6.367 |
|Film-Noir  |6.463 |
|Sci-Fi     |5.353 |

```

```
|News      |7.203 |
|Drama    |6.248 |
|Documentary|7.216 |
|Western   |5.84  |
|Comedy    |5.906 |
|Crime     |5.985 |
+-----+
only showing top 20 rows
```

[67] :

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...

+-----+-----+
|Genre      |Rating|
+-----+-----+
|Mystery   |5.847 |
|Musical   |6.187 |
>Action     |5.732 |
|Sport      |6.623 |
|Talk-Show  |6.858 |
|Romance    |6.102 |
|Thriller   |5.613 |
|Reality-TV |6.701 |
|Family     |6.205 |
|Fantasy    |5.898 |
|History    |6.798 |
|Animation  |6.367 |
|Film-Noir  |6.463 |
|Sci-Fi     |5.353 |
|News       |7.203 |
|Drama      |6.248 |
|Documentary|7.216 |
|Western    |5.84  |
|Comedy    |5.906 |
|Crime     |5.985 |
+-----+
only showing top 20 rows
```

6.3.2 Horizontal Bar Chart of Top Genres

With this data available, let us now build a barchart of all genres

HINT: don't forget about the matplotlib magic!

```
%matplotlib plt
```

```
[146]: plot_df = average_rating_per_genre.sort(average_rating_per_genre.Rating.desc())
plot_df.show(30,truncate=False)
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

+-----+-----+
|Genre      |Rating|
+-----+-----+
|Documentary|7.216 |
|News       |7.203 |
|Biography   |6.951 |
|Game-Show   |6.88  |
|Talk-Show   |6.858 |
|History     |6.798 |
|Music       |6.755 |
|Reality-TV  |6.701 |
|Sport       |6.623 |
|Film-Noir   |6.463 |
|War         |6.403 |
|Animation   |6.367 |
|Drama       |6.248 |
|Family      |6.205 |
|Musical     |6.187 |
|Romance     |6.102 |
|Crime       |5.985 |
|Comedy      |5.906 |
|Fantasy     |5.898 |
|Adventure   |5.866 |
|Mystery     |5.847 |
|Western     |5.84  |
>Action      |5.732 |
|Thriller    |5.613 |
|Adult       |5.554 |
|Sci-Fi      |5.353 |
|Horror      |5.002 |
|Short       |5.0   |
+-----+-----+
```

```
[68]:
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

+-----+-----+
```

Genre	Rating
Documentary	7.216
News	7.203
Biography	6.951
Game-Show	6.88
Talk-Show	6.858
History	6.798
Music	6.755
Reality-TV	6.701
Sport	6.623
Film-Noir	6.463
War	6.403
Animation	6.367
Drama	6.248
Family	6.205
Musical	6.187
Romance	6.102
Crime	5.985
Comedy	5.906
Fantasy	5.898
Adventure	5.866
Mystery	5.847
Western	5.84
Action	5.732
Thriller	5.613
Adult	5.554
Sci-Fi	5.353
Horror	5.002
Short	5.0

You do not have to match the color and the figure size but all other aspects of the graph should be matched.

```
[147]: import matplotlib.pyplot as plt

plot = plot_df.toPandas()

plt.figure(figsize=(12, 8)) #I tried the size as best as possible
plt.barh(plot["Genre"], plot["Rating"], color="purple", height=0.5, ↴
         label="Rating")

plt.xlabel("Rating")
plt.ylabel("Genre")
plt.title("Top Genres in the Movie Category")

plt.xlim(4.5, 7.5)
```

```

plt.gca().invert_yaxis()

plt.legend()
plt.tight_layout()
plt.show()

%matplotlib plt

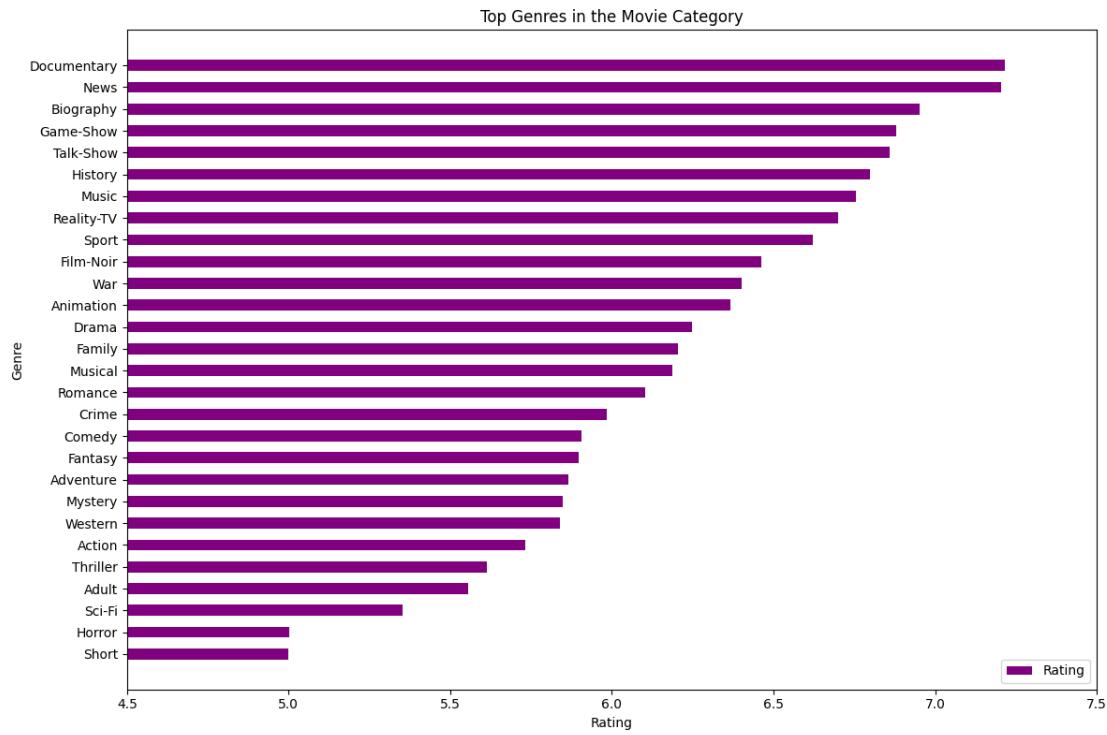
```

VBox()

```

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...  
layout=Layout(height='25px', width='50%'), ...

```



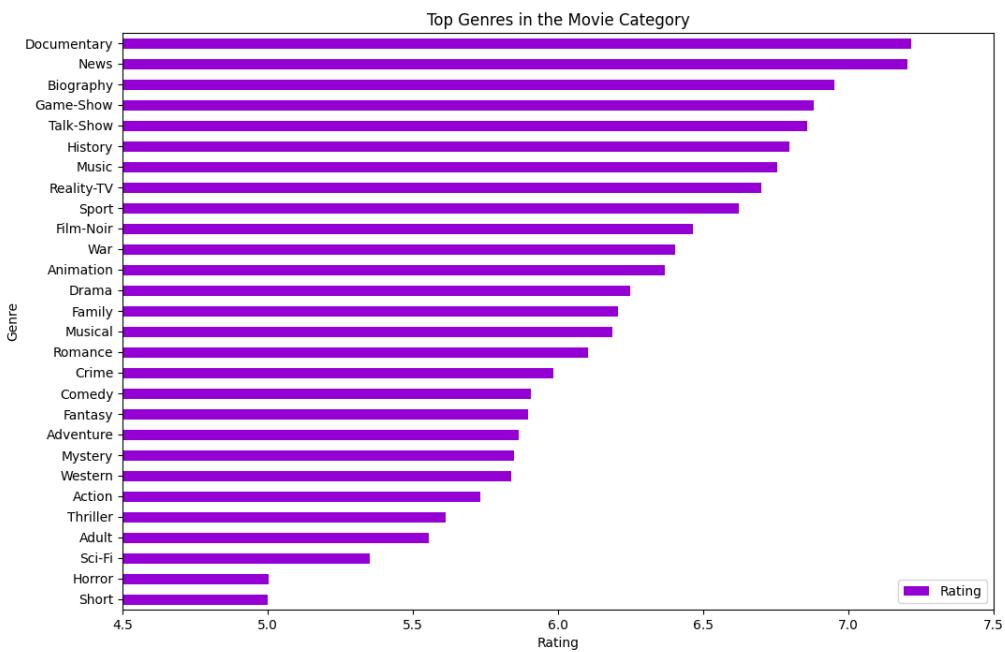
[70] :

VBox()

```

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...  
layout=Layout(height='25px', width='50%'), ...

```



6.4 PART 3 - Analyzing Job Categories

6.5 Total Unique Job Categories

What is the total number of unique job categories?

```
[95]: principles.printSchema()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...
root
|-- tconst: string (nullable = true)
|-- ordering: string (nullable = true)
|-- nconst: string (nullable = true)
|-- category: string (nullable = true)
|-- job: string (nullable = true)
|-- characters: string (nullable = true)
```

```
[148]:
```

```

unique_categories_df = principles.select("tconst", "category") .
    sort(col("tconst").asc())

unique_categories_df.show(30, truncate=False)

```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:',
 layout=Layout(height='25px', width='50%'),...

tconst	category
tt0000001	self
tt0000001	director
tt0000001	cinematographer
tt0000002	director
tt0000002	composer
tt0000003	director
tt0000003	producer
tt0000003	composer
tt0000003	editor
tt0000004	director
tt0000004	composer
tt0000005	actor
tt0000005	actor
tt0000005	director
tt0000005	producer
tt0000006	director
tt0000007	actor
tt0000007	actor
tt0000007	director
tt0000007	director
tt0000007	producer
tt0000008	actor
tt0000008	director
tt0000008	cinematographer
tt0000009	actress
tt0000009	actor
tt0000009	actor
tt0000009	director
tt0000010	director
tt0000011	actor

only showing top 30 rows

[71]:

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...

+-----+-----+
|tconst |category |
+-----+-----+
|tt0000001|self   |
|tt0000001|director|
|tt0000001|cinematographer|
|tt0000002|director|
|tt0000002|composer|
|tt0000003|director|
|tt0000003|producer|
|tt0000003|composer|
|tt0000003|editor  |
|tt0000004|director|
|tt0000004|composer|
|tt0000005|actor   |
|tt0000005|actor   |
|tt0000005|director|
|tt0000005|producer|
|tt0000006|director|
|tt0000007|actor   |
|tt0000007|actor   |
|tt0000007|director|
|tt0000007|director|
|tt0000007|producer|
|tt0000008|actor   |
|tt0000008|director|
|tt0000008|cinematographer|
|tt0000009|actress |
|tt0000009|actor   |
|tt0000009|actor   |
|tt0000009|director|
|tt0000010|director|
|tt0000011|actor   |
+-----+-----+
only showing top 30 rows
```

```
[149]: from pyspark.sql.functions import countDistinct
print(unique_categories_df.select(countDistinct('category')).collect()[0][0])
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...
```

[72] :

```
VBox()  
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    layout=Layout(height='25px', width='50%'),...
```

12

What are the unique job categories available?

[150] : `unique_categories_df.select("category").distinct().show(truncate=False)`

```
VBox()  
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    layout=Layout(height='25px', width='50%'),...  
  
+-----+  
| category |  
+-----+  
| actress |  
| producer |  
| production_designer |  
| writer |  
| actor |  
| cinematographer |  
| archive_sound |  
| archive_footage |  
| self |  
| editor |  
| composer |  
| director |  
+-----+
```

[73] :

```
VBox()  
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    layout=Layout(height='25px', width='50%'),...  
  
+-----+  
| category |  
+-----+  
| actress |  
| producer |  
| production_designer |  
| writer |  
| actor |  
| cinematographer |
```

```
|archive_sound      |
|archive_footage   |
|self              |
|editor            |
|composer          |
|director          |
+-----+
```

6.6 Top Job Categories

Now let's find the top job categories in this dataset by rolling up categories.

6.6.1 Counts of Titles / Job Category

The expected output should be:

category	count
a	15
b	2
c	45

Or something to that effect.

```
[151]: count_categories = unique_categories_df.groupby("category").count()
count_categories.show(truncate=False)
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', 
layout=Layout(height='25px', width='50%'), ...

+-----+-----+
|category|count|
+-----+-----+
|actress|10492210|
|producer|3944711 |
|production_designer|383761 |
|writer|8495903 |
|actor|13443688|
|cinematographer|2068164 |
|archive_sound|4794    |
|archive_footage|404581 |
|self|10562296|
|editor|2012800 |
|composer|2014049 |
|director|7006843 |
```

```
+-----+-----+
[74]: 
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...

+-----+-----+
|category | count |
+-----+-----+
|actress | 10492210 |
|producer | 3944711 |
|production_designer | 383761 |
|writer | 8495903 |
|actor | 13443688 |
|cinematographer | 2068164 |
|archive_sound | 4794 |
|archive_footage | 404581 |
|self | 10562296 |
|editor | 2012800 |
|composer | 2014049 |
|director | 7006843 |
+-----+-----+
```

6.6.2 Bar Chart of Top Job Categories

With this data available, let us now build a barchart of the top 5 categories.

HINT: don't forget about the matplotlib magic!

```
%matplotlib plt
```

```
[152]: plot2_df = count_categories.orderBy(col("count").desc())
plot2_df.show(truncate=False)
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...

+-----+-----+
|category | count |
+-----+-----+
|actor | 13443688 |
|self | 10562296 |
|actress | 10492210 |
|writer | 8495903 |
|director | 7006843 |
```

```
|producer          |3944711 |
|cinematographer |2068164 |
|composer         |2014049 |
|editor           |2012800 |
|archive_footage  |404581  |
|production_designer|383761 |
|archive_sound    |4794   |
+-----+-----+
```

[75]:

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', ▾
    layout=Layout(height='25px', width='50%'), ...

+-----+-----+
|category      |count   |
+-----+-----+
|actor         |13443688|
|self          |10562296|
|actress        |10492210|
|writer         |8495903 |
|director       |7006843 |
|producer       |3944711 |
|cinematographer|2068164 |
|composer        |2014049 |
|editor          |2012800 |
|archive_footage |404581  |
|production_designer|383761 |
|archive_sound   |4794   |
+-----+-----+
```

You do not have to match the color and the figure size but all other aspects of the graph should be matched.

[153]:

```
import matplotlib.pyplot as plt

top_categories_df = plot2_df.limit(5).toPandas()

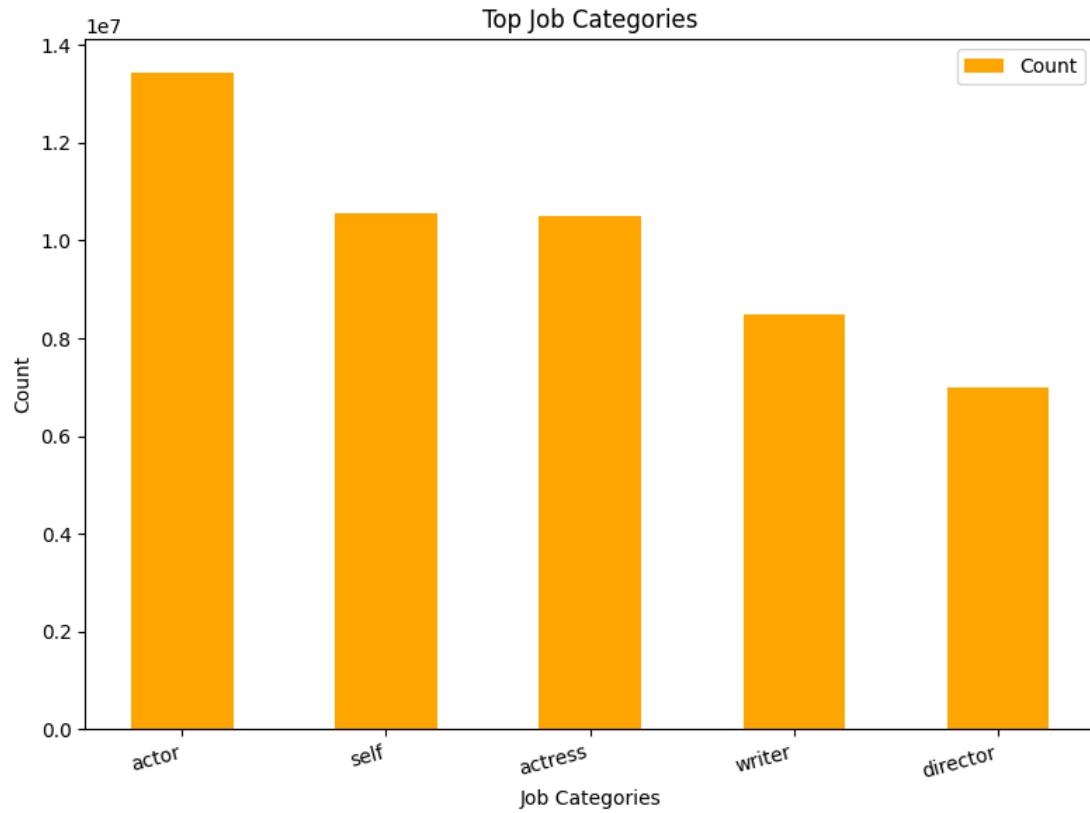
plt.figure(figsize=(8,6))#I tried the size as best as possible
plt.bar(top_categories_df["category"], top_categories_df["count"], ▾
    color="orange", width = 0.5, label="Count")

plt.xlabel("Job Categories")
plt.ylabel("Count")
plt.title("Top Job Categories")
```

```
plt.xticks(rotation=15, ha="right")  
  
plt.legend()  
plt.tight_layout()  
plt.show()  
  
%matplotlib plt
```

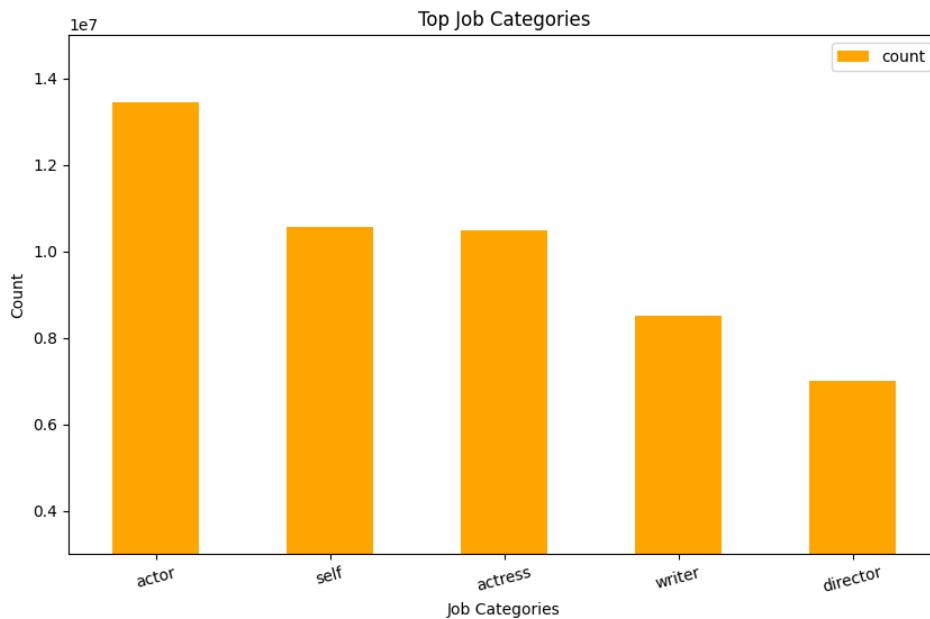
VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
             layout=Layout(height='25px', width='50%'), ...
```



[76]:

```
VBox()  
  
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
             layout=Layout(height='25px', width='50%'), ...
```



7 PART 4 - Answer to the following questions:

- 1) You will need to join tables to answer the following questions. Not every question will require four tables.
- 2) Your code should meet all the requirements asked in the questions.
- 3) Your code should be generalizable enough for any given arguments.

7.1 1) Which movies, released in 2003, have received more than 50,000 votes and have an average rating of 8 or higher?

```
[154]: titles.createOrReplaceTempView("titles")
ratings.createOrReplaceTempView("ratings")
principles.createOrReplaceTempView("principles")
name.createOrReplaceTempView("name")

df1 = titles.join(ratings, "tconst", "inner")
df2 = df1.join(principles, "tconst", "inner")
ultimate_df = df2.join(name, "nconst", "inner")
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...
```

```
[155]: ultimate_df2 = ultimate_df.withColumn("startYear", (col("startYear").
    ~cast("int")))
ultimate_df2 = ultimate_df2.withColumn("endYear", (col("endYear").cast("int")))
ultimate_df2 = ultimate_df2.withColumn("birthYear", (col("birthYear").
    ~cast("int")))
ultimate_df2 = ultimate_df2.withColumn("deathYear", (col("deathYear").
    ~cast("int")))

ultimate_df2.printSchema()
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', ~
    ~layout=Layout(height='25px', width='50%'), ...
```

```
root
|-- nconst: string (nullable = true)
|-- tconst: string (nullable = true)
|-- titleType: string (nullable = true)
|-- primaryTitle: string (nullable = true)
|-- isAdult: string (nullable = true)
|-- startYear: integer (nullable = true)
|-- endYear: integer (nullable = true)
|-- runtimeMinutes: string (nullable = true)
|-- genres: string (nullable = true)
|-- averageRating: float (nullable = true)
|-- numVotes: integer (nullable = true)
|-- ordering: string (nullable = true)
|-- category: string (nullable = true)
|-- job: string (nullable = true)
|-- characters: string (nullable = true)
|-- primaryName: string (nullable = true)
|-- birthYear: integer (nullable = true)
|-- deathYear: integer (nullable = true)
|-- primaryProfession: string (nullable = true)
|-- knownForTitles: string (nullable = true)
```

```
[156]: titles_df2 = titles.filter(titles.titleType == "movie")
```

```
joined_df2 = titles_df2.join(ratings, "tconst", "inner")
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', ~
    ~layout=Layout(height='25px', width='50%'), ...
```

```
[157]:
```

```

filtered_movies = (joined_df2.filter((joined_df2.startYear == "2003") &
    (joined_df2.numVotes > 50000) & (joined_df2.averageRating >= 8)).
    select(col("primaryTitle").alias("Movie"), col("averageRating").
    alias("Ratings"), col("numVotes").alias("Number of Votes")))
filtered_movies = filtered_movies.orderBy(col("Ratings").desc())

filtered_movies.show(truncate=False)

```

VBox()

```

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

+-----+-----+
|Movie |Ratings|Number of Votes|
+-----+-----+
|The Lord of the Rings: The Return of the King |9.0 |1965196 |
|Oldboy |8.3 |630695 |
|Finding Nemo |8.2 |1106772 |
|Kill Bill: Vol. 1 |8.2 |1184605 |
|Memories of Murder |8.1 |213610 |
|Pirates of the Caribbean: The Curse of the Black Pearl|8.1 |1202458 |
|Munna Bhai M.B.B.S. |8.1 |87972 |
|Spring, Summer, Fall, Winter... and Spring |8.0 |86510 |
|Dogville |8.0 |157921 |
|Big Fish |8.0 |457515 |
+-----+-----+

```

[77]:

VBox()

```

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

+-----+-----+
|Movie |Ratings|Number of Votes|
+-----+-----+
|The Lord of the Rings: The Return of the King |9.0 |1965196 |
|Oldboy |8.3 |630695 |
|Finding Nemo |8.2 |1106772 |
|Kill Bill: Vol. 1 |8.2 |1184605 |
|Memories of Murder |8.1 |213610 |
|Pirates of the Caribbean: The Curse of the Black Pearl|8.1 |1202458 |
|Munna Bhai M.B.B.S. |8.1 |87972 |
|Spring, Summer, Fall, Winter... and Spring |8.0 |86510 |
|Dogville |8.0 |157921 |
|Big Fish |8.0 |457515 |
+-----+-----+

```

7.2 2) List the films featuring Cillian Murphy as an actor since 2007, including their ratings. What is his highest-rated movie?

```
[158]: from pyspark.sql.functions import col, max, round
cillian_movies = ultimate_df2.filter((ultimate_df2.titleType == "movie") &
    (ultimate_df2.primaryName == "Cillian Murphy") & (ultimate_df2.category == "actor") & (ultimate_df2.startYear >= 2007))
cillian_movies = cillian_movies.sort(cillian_movies.startYear.desc(), cillian_movies.averageRating.desc())
cillian_movies = cillian_movies.select(col("primaryTitle").
    alias("Movies"), col("startYear").alias("Year"), round(col("averageRating") .
    cast("double"), 2).alias("Avg Rating"))
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...
```

```
[159]: highestRatedMovie = cillian_movies.orderBy(col("Avg Rating").desc()).first()

movie_name = highestRatedMovie["Movies"]
movie_rating = highestRatedMovie["Avg Rating"]
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...
```

```
[160]: cillian_movies.show(truncate=False)

print("Highest rated movie:", movie_name, "with a rating of", movie_rating)
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...
```

Movies	Year	Avg Rating
Small Things Like These	2024	7.2
Oppenheimer	2023	8.4
Kensuke's Kingdom	2023	7.1
A Quiet Place Part II	2020	7.2
Anna	2019	6.6
Anthropoid	2016	7.2
Free Fire	2016	6.3
In the Heart of the Sea	2015	6.9
Transcendence	2014	6.2

Aloft	2014 5.3	
Red Lights	2012 6.2	
In Time	2011 6.7	
Retreat	2011 5.8	
Peacock	2010 6.2	
Perrier's Bounty	2009 6.3	
Waveriders	2008 6.8	
Sunshine	2007 7.2	
Watching the Detectives	2007 6.2	

Highest rated movie: Oppenheimer with a rating of 8.4

[79]:

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...

+-----+-----+
|Movies|Year|Avg Rating|
+-----+-----+
|Small Things Like These|2024|7.2| |
|Oppenheimer|2023|8.4| |
|Kensuke's Kingdom|2023|7.1| |
|A Quiet Place Part II|2020|7.2| |
|Anna|2019|6.6| |
|Anthropoid|2016|7.2| |
|Free Fire|2016|6.3| |
|In the Heart of the Sea|2015|6.9| |
|Transcendence|2014|6.2| |
|Aloft|2014|5.3| |
|Red Lights|2012|6.2| |
|Retreat|2011|5.8| |
|In Time|2011|6.7| |
|Peacock|2010|6.2| |
|Perrier's Bounty|2009|6.3| |
|Waveriders|2008|6.8| |
|Sunshine|2007|7.2| |
|Watching the Detectives|2007|6.2| |
+-----+-----+
```

Highest rated movie: Oppenheimer with a rating of 8.4

7.3 3) How many movies has Zendaya featured as an actress in each year?

```
[161]: dfz = principles.join(name, "nconst", "inner")
dfz2 = dfz.join(titles, "tconst", "inner")

zendaya_movies = dfz2.filter((dfz2.titleType == "movie") & (dfz2.primaryName == "Zendaya") & (dfz2.category == "actress"))
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', ...  
layout=Layout(height='25px', width='50%'), ...
```

```
[162]: from pyspark.sql.functions import count
zendaya_movies = zendaya_movies.filter(col("startYear") != null)

movies_per_year = zendaya_movies.groupBy("startYear").agg(count("titleType") .  
alias("Total"))

movies_per_year = movies_per_year.sort(movies_per_year.startYear.desc())

movies_per_year = movies_per_year.select(col("startYear").alias("Year"), ...  
col("Total"))

movies_per_year.show(truncate=False)
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', ...  
layout=Layout(height='25px', width='50%'), ...
```

```
+----+-----+
|Year|Total|
+----+-----+
|2024|2    |
|2021|3    |
|2018|2    |
|2017|1    |
+----+-----+
```

[80]:

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', ...  
layout=Layout(height='25px', width='50%'), ...
```

```
+----+-----+
|Year|Total|
+----+-----+
```

2024 2	
2021 3	
2018 2	
2017 1	
+----+-----+	

7.4 4) At what age did Audrey Hepburn, known for her role in the movie ‘Breakfast at Tiffany’s,’ pass away?

```
[163]: breakfast_tconst = titles.filter(titles.primaryTitle == "Breakfast at Tiffany's").select("tconst").collect()[0]["tconst"]
audrey_nconst = principles.filter(principles.tconst == breakfast_tconst).
    filter(principles.category == "actress").select("nconst").
    collect()[0]["nconst"]

audrey_data = name.filter(name.nconst == audrey_nconst)
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:',
 layout=Layout(height='25px', width='50%'),...

```
[164]: audrey_age = audrey_data.select((col("deathYear").cast("int") -  

    col("birthYear").cast("int")).alias("Age"))
audrey_age = audrey_age.select((col("Age").cast("string"))).collect()[0]["Age"]  

    #need to convert back to string to add fullstop

print("Audrey Hepburn passed away at the age of", audrey_age + ".")
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:',
 layout=Layout(height='25px', width='50%'),...

Audrey Hepburn passed away at the age of 64.

[87]:

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:',
 layout=Layout(height='25px', width='50%'),...

Audrey Hepburn passed away at the age of 64.

7.5 5) What is the movie(s) with the highest average rating among those featuring Chris Evans, known for his role in ‘Captain America: The First Avenger’?

Write your code in a way that it finds and displays all movies with the highest rating, even if there's more than one.

```
[166]: from pyspark.sql.functions import col, max

cap_tconst = titles.filter(titles.primaryTitle == "Captain America: The First Avenger").select("tconst").collect()[0]["tconst"]
chris_nconst = principles.filter(principles.tconst == cap_tconst).
    filter(principles.category == "actor").select("nconst").collect()[0]["nconst"]

chris_evans_movies = ultimate_df2.filter((ultimate_df2.nconst == chris_nconst) & (ultimate_df2.titleType == "movie"))

highest_rating = chris_evans_movies.agg(max(col("averageRating"))).
    alias("highest_rating")).collect()[0]["highest_rating"]

highestRatedMovies = chris_evans_movies.filter(chris_evans_movies.
    averageRating == highest_rating)

highestRatedMovies = highestRatedMovies.sort(highestRatedMovies.startYear.
    asc())

highestRatedMovies = highestRatedMovies.select(col("primaryTitle").
    alias("Movies"), col("averageRating").alias("Highest Avg Rating"))

highestRatedMovies.show(truncate=False)
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...  
+-----+-----+  
| Movies | Highest Avg Rating |  
+-----+-----+  
| Avengers: Infinity War | 8.4 |  
| Avengers: Endgame | 8.4 |  
+-----+-----+
```

[88]:

```
VBox()  
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'), ...
```

Movies	Highest Avg Rating
Avengers: Infinity War	8.4
Avengers: Endgame	8.4

- 7.6 6) Among the movies in which Clint Eastwood, known for ‘The Good, the Bad and the Ugly’, and Harrison Ford, known for ‘Raiders of the Lost Ark’, have acted, who has the higher average rating?

Hint: You will need to calculate the average rating across all movies for each actor.

```
[167]: clint_tconst = titles.filter(col("primaryTitle") == "The Good, the Bad and the
        ↪Ugly").select("tconst").collect()[0]["tconst"]
harrison_tconst = titles.filter(col("primaryTitle") == "Raiders of the Lost
        ↪Ark").select("tconst").collect()[0]["tconst"]

clint_nconst = ultimate_df2.filter((col("tconst") == clint_tconst) &
        ↪(col("primaryName") == "Clint Eastwood")).select("nconst").
        ↪collect()[0]["nconst"]
harrison_nconst = ultimate_df2.filter((col("tconst") == harrison_tconst) &
        ↪(col("primaryName") == "Harrison Ford")).select("nconst").
        ↪collect()[0]["nconst"]
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', ↪
 layout=Layout(height='25px', width='50%'), ...

```
[168]: clint_movies = ultimate_df2.filter((ultimate_df2.titleType == "movie") &
        ↪(ultimate_df2.nconst == clint_nconst) & (ultimate_df2.category == "actor"))
harrison_movies = ultimate_df2.filter((ultimate_df2.titleType == "movie") &
        ↪(ultimate_df2.nconst == harrison_nconst) & (ultimate_df2.category == "actor"))
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', ↪
 layout=Layout(height='25px', width='50%'), ...

```
[169]: from pyspark.sql.functions import col, avg
clint_avg_rating = clint_movies.agg(round(avg(col("averageRating")),2).
        ↪alias("average_rating"))

clint = clint_avg_rating.collect()[0]["average_rating"]
```

```
harrison_avg_rating = harrison_movies.agg(round(avg(col("averageRating")),2),  
                                         alias("average_rating"))

harrison = harrison_avg_rating.collect()[0]["average_rating"]
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
              layout=Layout(height='25px', width='50%'), ...
```

```
[170]: print("The average rating of Harrison Ford is", harrison)
       print("The average rating of Clint Eastwood is", clint)
       if clint > harrison:
           print("Clint Eastwood has a higher average rating")

       elif harrison > clint:
           print("Harrison Ford has a higher average rating")

       else:
           print("Harrison Ford and Clint Eastwood have same average rating")
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
              layout=Layout(height='25px', width='50%'), ...
```

The average rating of Harrison Ford is 6.83
The average rating of Clint Eastwood is 6.86
Clint Eastwood has a higher average rating

[]:

[89]:

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
              layout=Layout(height='25px', width='50%'), ...
```

The average rating of Harrison Ford is 6.83
The average rating of Clint Eastwood is 6.86
Clint Eastwood has a higher average rating

7.7 7) What are the movies in which both Johnny Depp and Helena Bonham Carter have acted together?

```
[ ]: #create 2 df, use inner join to join using tconst or use or to filter using both ↵names then unique names
```

```
[171]: johnny_movies = (ultimate_df2.filter((col("primaryName") == "Johnny Depp") & ↵(col("titleType") == "movie")).select("tconst").distinct())
helena_movies = (ultimate_df2.filter((col("primaryName") == "Helena Bonham ↵Carter") & (col("titleType") == "movie")).select("tconst").distinct())

common_movies = johnny_movies.join(helena_movies, "tconst", "inner")

common_movies = (common_movies.join(titles, "tconst").
    ↵select(col("primaryTitle").alias("Common Movies")))

common_movies.show(truncate=False)
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', ↵layout=Layout(height='25px', width='50%'), ...
```

```
+-----+
|Common Movies|
+-----+
|Sweeney Todd: The Demon Barber of Fleet Street|
|Charlie and the Chocolate Factory|
|Corpse Bride|
|Dark Shadows|
|Alice in Wonderland|
|Alice Through the Looking Glass|
+-----+
```

```
[90]:
```

VBox()

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', ↵layout=Layout(height='25px', width='50%'), ...
```

```
+-----+
|Common Movies|
+-----+
|Dark Shadows|
|Sweeney Todd: The Demon Barber of Fleet Street|
|Corpse Bride|
|Charlie and the Chocolate Factory|
|Alice in Wonderland|
+-----+
```

```
|Alice Through the Looking Glass |  
+-----+
```

7.8 8) Among the TV series featuring David Tennant, who is known for his role in Doctor Who, which rank in the top 5 for viewer engagement? Does Doctor Who make it into the highest-ranked series?

```
[ ]: #filter david tennant + tv shows, if new df needed then inner join principles  
    ↵with title, then with rating, then with name
```

```
[172]: david_tennant_series = principles.join(titles, "tconst", "inner")  
david_tennant_series = david_tennant_series.join(ratings, "tconst", "inner")  
david_tennant_series = david_tennant_series.join(name, "nconst", "inner")  
  
david_tennant_series = david_tennant_series.filter((david_tennant_series.  
    ↵primaryName == "David Tennant") & (david_tennant_series.titleType ==  
    ↵"tvSeries") & (david_tennant_series.category == "actor"))  
david_tennant_series = david_tennant_series.select(col("primaryTitle").  
    ↵alias("Tv Series"), col("numVotes").cast("int").alias("Number of Votes"))  
david_tennant_series = david_tennant_series.orderBy(col("Number of Votes").  
    ↵desc()).limit(5)  
  
doctor_who_show = david_tennant_series.filter(col("Tv Series") == "Doctor Who").  
    ↵collect()  
votes = doctor_who_show[0]["Number of Votes"]
```

```
VBox()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    ↵layout=Layout(height='25px', width='50%'), ...
```

```
[173]: david_tennant_series.show(truncate=False)  
  
if doctor_who_show:  
    print(f"Doctor Who is in the top 5 TV series for viewer engagement with  
    ↵{votes} votes.")  
else:  
    print("Doctor Who is not in the top 5 TV series for viewer engagement.")
```

```
VBox()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',  
    ↵layout=Layout(height='25px', width='50%'), ...
```

```
+-----+-----+  
|Tv Series      |Number of Votes|  
+-----+-----+  
|Doctor Who     |245190          |
```

```
+-----+-----+
| Jessica Jones | 225869 |
| Broadchurch   | 126132 |
| Good Omens    | 111450 |
| Ahsoka        | 107410 |
+-----+-----+
```

Doctor Who is in the top 5 TV series for viewer engagement with 245190 votes.

[91]:

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

+-----+-----+
| Tv Series | Number of Votes |
+-----+-----+
| Doctor Who | 245190 |
| Jessica Jones | 225869 |
| Broadchurch | 126132 |
| Good Omens | 111450 |
| Ahsoka | 107410 |
+-----+-----+
```

Doctor Who is in the top 5 TV series for viewer engagement with 245190 votes.

7.9 9) What are the highest and lowest-rated movies in the Harry Potter franchise featuring Daniel Radcliffe, and what are their ratings?

First, get the ratings for each movie in the franchise, and then find the highest and lowest-rated movies.

[]: #search by "Harry Potter and" in the name then sort both ways and use
 ↵collect then print

```
[179]: harry_potter_titles = ultimate_df2.filter((col("primaryTitle").
    ↵startswith("Harry Potter and the ")) & (col("titleType") == "movie") &_
    ↵(col("primaryName") == "Daniel Radcliffe"))

harry_potter_titles = harry_potter_titles.
    ↵select(col("primaryTitle"), round(col("averageRating").cast("double"), 2)).
    ↵alias("averageRating"))

harry_potter_titles.orderBy("averageRating", ascending=False).
    ↵show(truncate=False)

sorted_hp = harry_potter_titles.orderBy("averageRating", ascending=False)
sorted_hp_highest = sorted_hp.collect()[0]
```

```

highest_title = sorted_hp_highest["primaryTitle"]
highest_rating = sorted_hp_highest["averageRating"]

sorted_hp2 = harry_potter_titles.orderBy("averageRating", ascending=True)
sorted_hp_lowest = sorted_hp2.collect()[0]

lowest_title = sorted_hp_lowest["primaryTitle"]
lowest_rating = sorted_hp_lowest["averageRating"]

print("Highest Rating in the Harry Potter Franchise:", highest_title, "with a\u202a\u202a rating of", highest_rating)
print("Lowest Rating in the Harry Potter Franchise:", lowest_title, "with a\u202a\u202a rating of", lowest_rating)

```

VBox()

```

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

```

primaryTitle	averageRating
Harry Potter and the Deathly Hallows: Part 2	8.1
Harry Potter and the Prisoner of Azkaban	7.9
Harry Potter and the Deathly Hallows: Part 1	7.7
Harry Potter and the Goblet of Fire	7.7
Harry Potter and the Sorcerer's Stone	7.6
Harry Potter and the Half-Blood Prince	7.6
Harry Potter and the Order of the Phoenix	7.5
Harry Potter and the Chamber of Secrets	7.4

Highest Rating in the Harry Potter Franchise: Harry Potter and the Deathly Hallows: Part 2 with a rating of 8.1

Lowest Rating in the Harry Potter Franchise: Harry Potter and the Chamber of Secrets with a rating of 7.4

[93]:

VBox()

```

FloatProgress(value=0.0, bar_style='info', description='Progress:', ...
    layout=Layout(height='25px', width='50%'), ...

```

primaryTitle	averageRating
Harry Potter and the Half-Blood Prince	7.6

Harry Potter and the Prisoner of Azkaban	7.9	
Harry Potter and the Deathly Hallows: Part 2	8.1	
Harry Potter and the Deathly Hallows: Part 1	7.7	
Harry Potter and the Chamber of Secrets	7.4	
Harry Potter and the Goblet of Fire	7.7	
Harry Potter and the Sorcerer's Stone	7.6	
Harry Potter and the Order of the Phoenix	7.5	
-----+-----+		

Highest Rating in the Harry Potter Franchise: Harry Potter and the Deathly Hallows: Part 2 with a rating of 8.1

Lowest Rating in the Harry Potter Franchise: Harry Potter and the Chamber of Secrets with a rating of 7.4

[]: