

ProjectAnalysisI_Rashi

December 13, 2023

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.tsv` —> Name Basics
- `s3://cis9760-lecture9-movieanalysis/title.basics.tsv` —> Title Basics
- `s3://cis9760-lecture9-movieanalysis/title.principals.tsv` —> Title Principals
- `s3://cis9760-lecture9-movieanalysis/title.ratings.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

```
[1]: %%info
```

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

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

```
[1]:
```

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

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

Let's install the necessary packages here

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

```
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%'),...
```

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-2.8.2-py2.py3-none-any.whl (247 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-2.8.2

Collecting matplotlib==3.2.1

Downloading matplotlib-3.2.1-cp37-cp37m-manylinux1_x86_64.whl (12.4 MB)

Collecting cyclor>=0.10

```

    Downloading cyciler-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-3.1.1-py3-none-any.whl (103 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-
aef38d60-f248-4a68-9a5b-c1cff85ce9c5/lib/python3.7/site-packages (from
matplotlib==3.2.1) (2.8.2)
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: cyciler, pyparsing, typing-extensions, kiwisolver,
matplotlib
Successfully installed cyciler-0.11.0 kiwisolver-1.4.5 matplotlib-3.2.1
pyparsing-3.1.1 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.

[2]:

```

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%'),...

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-2.8.2-py2.py3-none-any.whl (247 kB)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/site-

```

```
packages (from pandas==1.0.5) (2023.3.post1)
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-2.8.2
```

```
Collecting matplotlib==3.2.1
  Downloading matplotlib-3.2.1-cp37-cp37m-manylinux1_x86_64.whl (12.4 MB)
Collecting cyclor>=0.10
  Downloading cyclor-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-3.1.1-py3-none-any.whl (103 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-29f5af80-7504-4a54-81fa-e5b663c6f2c3/lib/python3.7/site-packages
(from matplotlib==3.2.1) (2.8.2)
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: cyclor, pyparsing, typing-extensions, kiwisolver,
matplotlib
Successfully installed cyclor-0.11.0 kiwisolver-1.4.5 matplotlib-3.2.1
pyparsing-3.1.1 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.

```
[3]: 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%'),...
```

[3]:

```
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

```
[4]: namedf = spark.read.csv('s3://cis9760-lecture9-movieanalysis/name.basics.tsv',  
↳header='true', sep="\t")  
titledf = spark.read.csv('s3://cis9760-lecture9-movieanalysis/title.basics.  
↳tsv', header='true', sep="\t")  
titleprincdf = spark.read.csv('s3://cis9760-lecture9-movieanalysis/title.  
↳principals.tsv', header='true', sep="\t")  
titleratdf = spark.read.csv('s3://cis9760-lecture9-movieanalysis/title.ratings.  
↳tsv', header='true', sep="\t")
```

```
VBox()
```

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

[4]:

```
VBox()
```

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

4.1 Name Basics

Display the schema below:

```
[5]: namedf.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)

```

[5]:

```

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 5 rows with the following columns:

- primaryName
- birthYear
- deathYear
- knownForTitles

[6]: `namedf.select('primaryName', 'birthYear', 'deathYear', 'knownForTitles').show(5)`

```

VBox()

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

+-----+-----+-----+-----+
| primaryName|birthYear|deathYear|knownForTitles|
+-----+-----+-----+-----+
| Fred Astaire|1899|1987|tt0050419,tt00531...|
| Lauren Bacall|1924|2014|tt0071877,tt01170...|
|Brigitte Bardot|1934|\N|tt0054452,tt00491...|
| John Belushi|1949|1982|tt0077975,tt00725...|
| Ingmar Bergman|1918|2007|tt0069467,tt00509...|
+-----+-----+-----+-----+

only showing top 5 rows

```

[6]:

```

VBox()

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

```

```
+-----+-----+-----+-----+
| primaryName|birthYear|deathYear|      knownForTitles|
+-----+-----+-----+-----+
| Fred Astaire|    1899|    1987|tt0050419,tt00531...|
| Lauren Bacall|    1924|    2014|tt0071877,tt01170...|
|Brigitte Bardot|    1934|      \N|tt0054452,tt00491...|
| John Belushi|    1949|    1982|tt0077975,tt00725...|
| Ingmar Bergman|    1918|    2007|tt0069467,tt00509...|
+-----+-----+-----+-----+
only showing top 5 rows
```

4.2 Title Basics

Display the first 15 rows with the following columns:

- titleType
- primaryTitle
- genres

```
[7]: titledf.select('titleType', 'primaryTitle', 'genres').show(15)
```

```
VBox()
```

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

```
+-----+-----+-----+-----+
|titleType|      primaryTitle|      genres|
+-----+-----+-----+-----+
| short| Carmencita| Documentary,Short|
| short|Le clown et ses c...| Animation,Short|
| short| Pauvre Pierrot|Animation,Comedy,...|
| short| Un bon bock| Animation,Short|
| short| Blacksmith Scene| Comedy,Short|
| short| Chinese Opium Den| Short|
| short|Corbett and Court...| Short,Sport|
| short|Edison Kinetoscop...| Documentary,Short|
| movie| Miss Jerry| Romance|
| short| Exiting the Factory| Documentary,Short|
| short|Akrobatisches Pot...| Documentary,Short|
| short|The Arrival of a ...|Action,Documentar...|
| short|The Photographica...| Documentary,Short|
| short|The Sprinkler Spr...| Comedy,Short|
| short| Autour d'une cabine| Animation,Short|
+-----+-----+-----+-----+
only showing top 15 rows
```

```
[7]:
```

VBox()

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

```
+-----+-----+-----+  
|titleType|      primaryTitle|      genres|  
+-----+-----+-----+  
|  short|      Carmencita| Documentary,Short|  
|  short|Le clown et ses c...| Animation,Short|  
|  short|      Pauvre Pierrot|Animation,Comedy,...|  
|  short|      Un bon bock| Animation,Short|  
|  short|      Blacksmith Scene| Comedy,Short|  
|  short|      Chinese Opium Den| Short|  
|  short|Corbett and Court...| Short,Sport|  
|  short|Edison Kinetoscop...| Documentary,Short|  
| movie|      Miss Jerry| Romance|  
|  short|Exiting the Factory| Documentary,Short|  
|  short|Akrobatisches Pot...| Documentary,Short|  
|  short|The Arrival of a ...|Action,Documentar...|  
|  short|The Photographica...| Documentary,Short|  
|  short|The Sprinkler Spr...| Comedy,Short|  
|  short|Autour d'une cabine| Animation,Short|  
+-----+-----+-----+
```

only showing top 15 rows

Display the unique titles below:

```
[8]: titledf.select('titleType').distinct().show()
```

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|  
+-----+
```


[8]:

```
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|
+-----+
```

Display the schema below:

[9]: titledf.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)
```

[9]:

```
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)

```

4.3 Title Principles

Display the schema below:

```
[10]: titleprincdf.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)

```

```
[10]:
```

```
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 “self”

```
[11]: titleprincdf.filter(titleprincdf.category == 'self').show(15)
```

```
VBox()
```

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

```

+-----+-----+-----+-----+-----+
|  tconst|ordering|  nconst|category|job|          characters|
+-----+-----+-----+-----+-----+
|tt0000001|      1|nm1588970|  self| \N|      ["Herself"]|
|tt0000012|      1|nm2880396|  self| \N|      ["Herself"]|
|tt0000012|      2|nm9735580|  self| \N|      ["Himself"]|
|tt0000012|      3|nm0525900|  self| \N|      ["Herself"]|
|tt0000012|      4|nm9735581|  self| \N|      ["Herself"]|
|tt0000012|      7|nm9735579|  self| \N|      ["Herself"]|
|tt0000012|      8|nm9653419|  self| \N|      ["Herself"]|
|tt0000013|      1|nm0525908|  self| \N|      ["Himself"]|
|tt0000013|      2|nm1715062|  self| \N|      ["Himself"]|
|tt0000016|      1|nm0525900|  self| \N|["Herself (on the...|
|tt0000016|      2|nm9735581|  self| \N|["Herself (on the...|
|tt0000024|      1|nm0256651|  self| \N|["Herself - Empre...|
|tt0000024|      2|nm0435118|  self| \N|["Himself - Emper...|
|tt0000028|      1|nm2350838|  self| \N|      ["Himself"]|
|tt0000028|      2|nm0525908|  self| \N|      ["Himself"]|
+-----+-----+-----+-----+-----+
only showing top 15 rows

```

[11]:

```

VBox()

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

+-----+-----+-----+-----+-----+
|  tconst|ordering|  nconst|category|job|          characters|
+-----+-----+-----+-----+-----+
|tt0000001|      1|nm1588970|  self| \N|      ["Herself"]|
|tt0000012|      1|nm2880396|  self| \N|      ["Herself"]|
|tt0000012|      2|nm9735580|  self| \N|      ["Himself"]|
|tt0000012|      3|nm0525900|  self| \N|      ["Herself"]|
|tt0000012|      4|nm9735581|  self| \N|      ["Herself"]|
|tt0000012|      7|nm9735579|  self| \N|      ["Herself"]|
|tt0000012|      8|nm9653419|  self| \N|      ["Herself"]|
|tt0000013|      1|nm0525908|  self| \N|      ["Himself"]|
|tt0000013|      2|nm1715062|  self| \N|      ["Himself"]|
|tt0000016|      1|nm0525900|  self| \N|["Herself (on the...|
|tt0000016|      2|nm9735581|  self| \N|["Herself (on the...|
|tt0000024|      1|nm0256651|  self| \N|["Herself - Empre...|
|tt0000024|      2|nm0435118|  self| \N|["Himself - Emper...|
|tt0000028|      1|nm2350838|  self| \N|      ["Himself"]|
|tt0000028|      2|nm0525908|  self| \N|      ["Himself"]|
+-----+-----+-----+-----+-----+
only showing top 15 rows

```

4.4 Title Ratings

Display the schema below:

```
[12]: titleratdf.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)
```

```
[12]:
```

```
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

```
[13]: numVotes = titleratdf.numVotes.cast("float")
      titleratdf = titleratdf.withColumn("numVotes", numVotes)
      titleratdf.orderBy(titleratdf.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|2159745.0|
|tt0468569|          9.0|2134602.0|
|tt1375666|          8.8|1892958.0|
|tt0137523|          8.8|1725444.0|
|tt0110912|          8.9|1695159.0|
|tt0109830|          8.8|1662609.0|
|tt0944947|          9.4|1606096.0|
|tt0133093|          8.7|1554345.0|
|tt0120737|          8.8|1548920.0|
|tt0167260|          8.9|1533632.0|
```

```
+-----+-----+-----+
only showing top 10 rows
```

[13]:

```
VBox()

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

```
+-----+-----+-----+
|  tconst|averageRating| numVotes|
+-----+-----+-----+
|tt0111161|          9.3|2159745.0|
|tt0468569|          9.0|2134602.0|
|tt1375666|          8.8|1892958.0|
|tt0137523|          8.8|1725444.0|
|tt0110912|          8.9|1695159.0|
|tt0109830|          8.8|1662609.0|
|tt0944947|          9.4|1606096.0|
|tt0133093|          8.7|1554345.0|
|tt0120737|          8.8|1548920.0|
|tt0167260|          8.9|1533632.0|
|tt0068646|          9.2|1482225.0|
|tt1345836|          8.4|1421506.0|
|tt0167261|          8.7|1387208.0|
|tt0816692|          8.6|1348184.0|
|tt0114369|          8.6|1327441.0|
+-----+-----+-----+
only showing top 15 rows
```

5 Overview of Data

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

```
[14]: print("Number of columns in Name Basics table: ", len(namedf.columns))
      print("Number of rows in Name Basics table: ", namedf.count(), '\n')

      print("Number of columns in Title Basics table: ", len(titledf.columns))
      print("Number of rows in Title Basics table: ", titledf.count(), '\n')

      print("Number of columns in Title Principals table: ", len(titleprincdf.columns))
      print("Number of rows in Title Principals table: ", titleprincdf.count(), '\n')

      print("Number of columns in Title Ratings table: ", len(titleratdf.columns))
      print("Number of rows in Title Ratings table: ", titleratdf.count(), '\n')
```

```
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: 9706922
```

```
Number of columns in Title Basics table: 9
```

```
Number of rows in Title Basics table: 6321302
```

```
Number of columns in Title Principals table: 6
```

```
Number of rows in Title Principals table: 36468817
```

```
Number of columns in Title Ratings table: 3
```

```
Number of rows in Title Ratings table: 993153
```

[14]:

```
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: 9706922
```

```
Number of columns in Title Basics table: 9
```

```
Number of rows in Title Basics table: 6321302
```

```
Number of columns in Title Principles table: 6
```

```
Number of rows in Title Principles table: 36468817
```

```
Number of columns in Title Ratings table: 3
```

```
Number of rows in Title Ratings table: 993153
```

6 PART 2 - Analyzing Genres

Let's now answer this question: how many unique 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

```
[15]: titledf.select("tconst", "titleType", "genres").show(5)
```

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,...|  
|tt0000004|  short|  Animation,Short|  
|tt0000005|  short|  Comedy,Short|  
+-----+-----+-----+
```

only showing top 5 rows

```
[15]:
```

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,...|
```

tt00000004	short	Animation,Short
tt00000005	short	Comedy,Short

+-----+-----+-----+

only showing top 5 rows

Display the first 10 rows of your association table below

```
[16]: from pyspark.sql.functions import *
# Split genres column into a list
df_with_genre_list = titledf.withColumn("genres", split("genres", ","))

# Explode genres list in different rows
separate_genre_df = df_with_genre_list.withColumn("genre", explode("genres"))

separate_genre_df.select("tconst", "titleType", "genre").show(10)
```

VBox()

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

	tconst	titleType	genre
--	--------	-----------	-------

+-----+-----+-----+

tt00000001	short	Documentary
tt00000001	short	Short
tt00000002	short	Animation
tt00000002	short	Short
tt00000003	short	Animation
tt00000003	short	Comedy
tt00000003	short	Romance
tt00000004	short	Animation
tt00000004	short	Short
tt00000005	short	Comedy

+-----+-----+-----+

only showing top 10 rows

[16]:

VBox()

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

	tconst	titleType	genre
--	--------	-----------	-------

+-----+-----+-----+

tt00000001	short	Documentary
tt00000001	short	Short
tt00000002	short	Animation

tt00000002	short	Short
tt00000003	short	Animation
tt00000003	short	Comedy
tt00000003	short	Romance
tt00000004	short	Animation
tt00000004	short	Short
tt00000005	short	Comedy

+-----+-----+-----+

only showing top 10 rows

6.2 Total Unique Genres

What is the total number of unique genres available in the movie title type?

```
[17]: movie_data= separate_genre_df.filter("titleType = 'movie'").
      ↪select("genre", "tconst")

num_unique_genres_movie = movie_data.select("genre").distinct().count()

print("Total number of unique genres in movie title type:",
      ↪num_unique_genres_movie)
```

VBox()

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

Total number of unique genres in movie title type: 29

[17]:

VBox()

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

29

What are the unique genres available?

```
[20]: movie_data.select("genre").distinct().show(29, 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	
\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	
+-----+	

[18]:

```

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!

```
[21]: movie_data=movie_data.filter(col("genre") != "\\N")
      movie_data.select("genre").distinct().show(29, 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      |
+-----+

```

[19]:

```
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

Or something to that effect.

First, let's join our two dataframes (movie ratings and genres) by tconst

```
[22]: joined_df = titlertatdf.join(movie_data, "tconst")
      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|          4.2|
|Biography|        4.1|
|  Drama|          4.1|
|  History|        4.1|
|  Drama|          5.7|
|  Drama|          4.6|
|  History|        4.6|
|Biography|        6.3|
|  Drama|          6.3|
```

```
+-----+-----+
only showing top 10 rows
```

[20]:

```
VBox()

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

```
+-----+-----+
|   genre|averageRating|
+-----+-----+
|   Drama|          4.2|
|   Drama|          4.2|
|Biography|         4.1|
|   Drama|          4.1|
|  History|          4.1|
|   Drama|          5.7|
|   Drama|          4.6|
|  History|          4.6|
|Biography|          6.3|
|   Drama|          6.3|
+-----+-----+
only showing top 10 rows
```

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

[23]: `genre_avg_rating_df = joined_df.groupby("genre").agg(avg("averageRating").`
`↳alias("averageRating"))`
`genre_avg_rating_df.show(truncate=False)`

```
VBox()

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

```
+-----+-----+
|genre      |averageRating      |
+-----+-----+
|Mystery    |5.9404375359815775|
|Musical    |6.2032460545193695|
|Action     |5.718734069668649 |
|Sport      |6.600145190562615 |
|Romance    |6.1257141803973605|
|Thriller   |5.625967566447334 |
|Family     |6.250560452715202 |
|Fantasy    |5.92482076283338  |
|History    |6.822718115605145 |
```

Animation	6.326203750633554	
Film-Noir	6.636246786632392	
Sci-Fi	5.325150008571917	
News	7.2009160305343505	
Drama	6.2880802103879025	
Documentary	7.245469798657718	
Western	5.948970989337962	
Comedy	5.94136310800413	
Crime	6.026013332684543	
War	6.483807030665668	
Adult	5.721742584745762	

+-----+-----+

only showing top 20 rows

[21]:

VBox()

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

genre	avg_rating	
-------	------------	--

+-----+-----+

Mystery	5.940437537126316	
Musical	6.203246053185319	
Action	5.718734067904495	
Sport	6.600145190943391	
Romance	6.125714179294426	
Thriller	5.625967567519544	
Family	6.250560452699635	
Fantasy	5.924820762891499	
History	6.822718117193864	
Animation	6.326203749467441	
Film-Noir	6.636246780503378	
Sci-Fi	5.325150006900168	
News	7.200916040944689	
Drama	6.288080211097538	
Documentary	7.245469805371099	
Western	5.948970991005059	
Comedy	5.941363107822231	
Crime	6.026013333109149	
War	6.483807036278403	
Adult	5.721742579082059	

+-----+-----+

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
```

```
[24]: genre_avg_rating_df.orderBy(col("averageRating").desc()).show(truncate=False)
```

```
VBox()
```

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

```
+-----+-----+  
|genre      |averageRating      |  
+-----+-----+  
|Short      |7.2600000000000001 |  
|Documentary|7.245469798657718 |  
|News       |7.2009160305343505 |  
|Biography  |6.983637640449436 |  
|Game-Show  |6.975              |  
|History    |6.822718115605145 |  
|Music      |6.752020202020201 |  
|Film-Noir  |6.636246786632392 |  
|Sport      |6.600145190562615 |  
|War        |6.483807030665668 |  
|Reality-TV |6.379310344827586 |  
|Animation  |6.326203750633554 |  
|Drama      |6.2880802103879025 |  
|Family     |6.250560452715202 |  
|Musical    |6.2032460545193695 |  
|Romance    |6.1257141803973605 |  
|Crime      |6.026013332684543 |  
|Western    |5.948970989337962 |  
|Comedy     |5.94136310800413 |  
|Mystery    |5.9404375359815775 |  
+-----+-----+
```

only showing top 20 rows

```
[22]:
```

```
VBox()
```

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

```
+-----+-----+  
|genre      |avg_rating          |  
+-----+-----+
```


Short	7.259999942779541
Documentary	7.245469805371099
News	7.200916040944689
Biography	6.983637643044585
Game-Show	6.974999904632568
History	6.822718117193864
Music	6.752020207214588
Film-Noir	6.636246780503378
Sport	6.600145190943391
War	6.483807036278403
Reality-TV	6.379310377712907
Animation	6.326203749467441
Drama	6.288080211097538
Family	6.250560452699635
Musical	6.203246053185319
Romance	6.125714179294426
Crime	6.026013333109149
Western	5.948970991005059
Comedy	5.941363107822231
Mystery	5.940437537126316

+-----+-----+

only showing top 20 rows

```
[25]: # Convert the PySpark DataFrame to a Pandas DataFrame
genre_avg_rating_pd_df = genre_avg_rating_df.toPandas()

plt.figure(figsize=(12, 8))

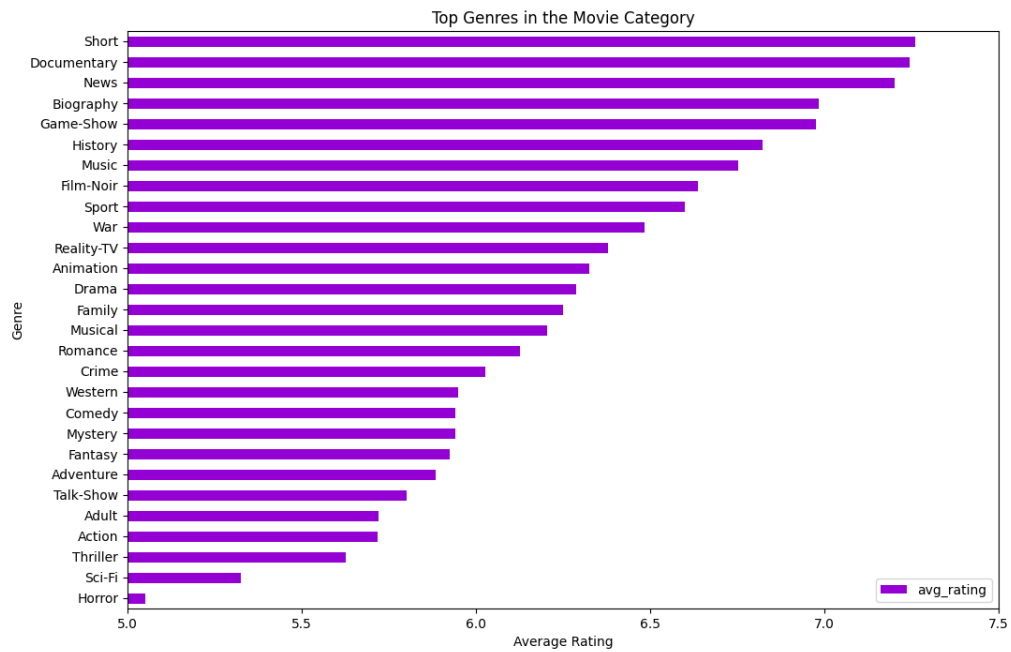
# Sort the DataFrame by 'averageRating'
genre_avg_rating_sorted_pd_df = genre_avg_rating_pd_df.
    ↪sort_values(by="averageRating", ascending=False)

movie_plot = plt.barh(genre_avg_rating_sorted_pd_df['genre'],
    ↪genre_avg_rating_sorted_pd_df['averageRating'], color='darkviolet', height=0.
    ↪5)

plt.xlabel("Average Rating")
plt.ylabel("Genre")
plt.title("Top Genres in the Movie Category")
plt.xlim(5.0, 7.5)
plt.gca().invert_yaxis()
plt.ylim(len(genre_avg_rating_sorted_pd_df['genre']) - 0.5, -0.5)
plt.legend([movie_plot], ['avg_rating'], loc='lower right')
%matplotlib plt
```

VBox()

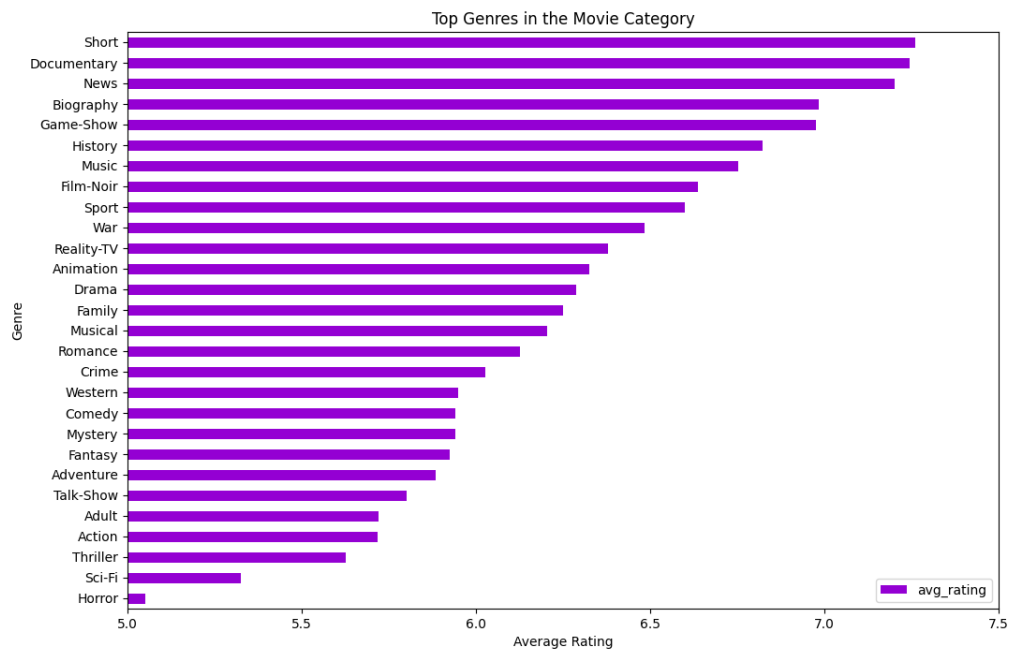
```
FloatProgress(value=0.0, bar_style='info', description='Progress:',
↳layout=Layout(height='25px', width='50%'),...
```



[23] :

```
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?

```
[26]: titleprincdf.select('tconst', 'category').show(5)
```

```
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|
+-----+-----+
only showing top 5 rows
```

[24]:

```
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|
+-----+-----+
only showing top 5 rows
```

```
[27]: unique_cat = titleprincdf.select("category").distinct().count()

print("Total number of unique job categories:", unique_cat)
```

```
VBox()

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

Total number of unique job categories: 12
```

[25]:

```
VBox()

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

12

What are the unique job categories available?
```

```
[28]: titleprincdf.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           |
+-----+
```

[26]:

```
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

category	count
c	45

Or something to that effect.

```
[29]: category_counts = titleprincdf.groupby("category").count()
category_counts.show(truncate=False)
```

VBox()

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

```
+-----+-----+
|category      |count  |
+-----+-----+
|actress       |6325097|
|producer      |2197866|
|production_designer|285924 |
|writer        |4811596|
|actor         |8493701|
|cinematographer|1300404|
|archive_sound  |2143   |
|archive_footage|209035 |
|self          |6153089|
|editor        |1197669|
|composer      |1313187|
|director      |4179106|
+-----+-----+
```

```
[27]:
```

VBox()

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

```
+-----+-----+
|category      |count  |
+-----+-----+
|actress       |6325097|
|producer      |2197866|
|production_designer|285924 |
|writer        |4811596|
|actor         |8493701|
|cinematographer|1300404|
|archive_sound  |2143   |
|archive_footage|209035 |
|self          |6153089|
```

editor	1197669
composer	1313187
director	4179106

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

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
```

```
[30]: category_counts.orderBy(col("count").desc()).show(truncate=False)
```

```
VBox()
```

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

category	count
----------	-------

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

actor	8493701
actress	6325097
self	6153089
writer	4811596
director	4179106
producer	2197866
composer	1313187
cinematographer	1300404
editor	1197669
production_designer	285924
archive_footage	209035
archive_sound	2143

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

```
[28]:
```

```
VBox()
```

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

category	count
----------	-------

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

actor	8493701
actress	6325097
self	6153089

writer	4811596
director	4179106
producer	2197866
composer	1313187
cinematographer	1300404
editor	1197669
production_designer	285924
archive_footage	209035
archive_sound	2143

+-----+-----+

```
[31]: # Convert the PySpark DataFrame to a Pandas DataFrame
category_counts_pd = category_counts.toPandas()

plt.figure(figsize=(12, 8))

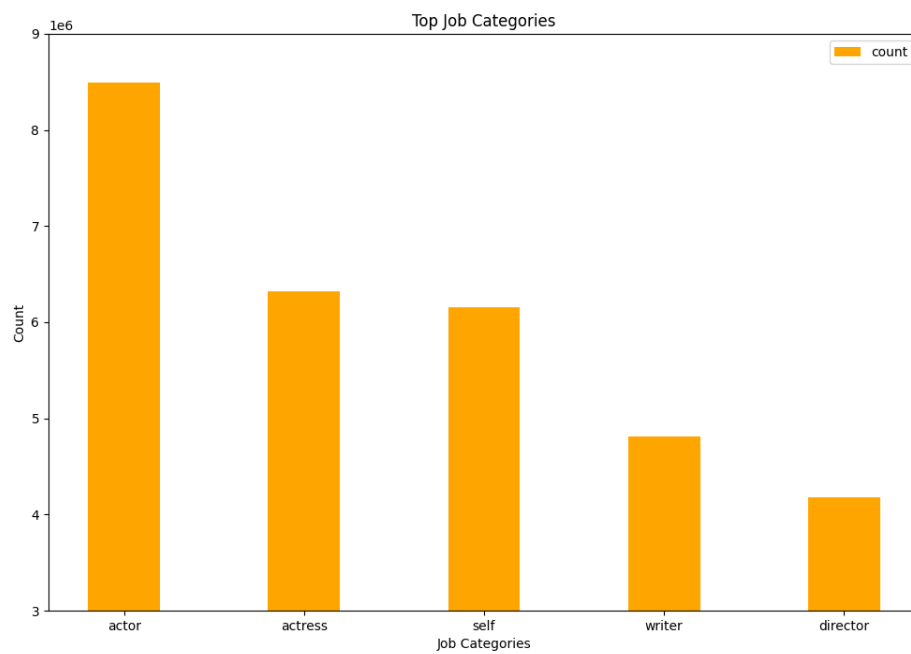
# Sort the DataFrame by 'averageRating'
category_counts_pd = category_counts_pd.sort_values(by="count",
    ↪ascending=False).head(5)

job_plot = plt.bar(category_counts_pd['category'], category_counts_pd['count'],
    ↪color='orange', width=0.4)

plt.xlabel("Job Categories")
plt.ylabel("Count")
plt.title("Top Job Categories")
plt.ylim(3e6, 9e6)
plt.legend([job_plot], ['count'], loc='upper right')
%matplotlib plt
```

VBox()

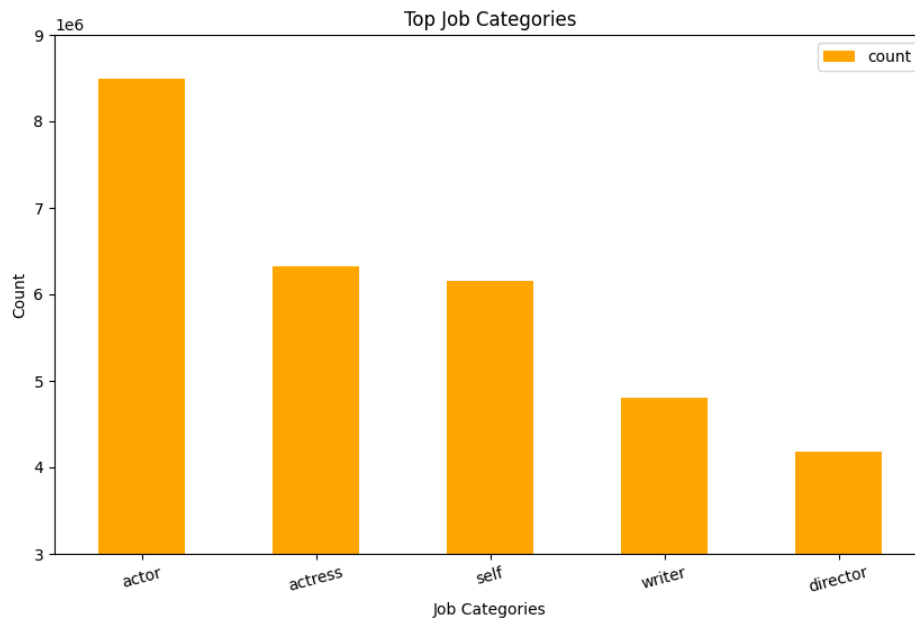
```
FloatProgress(value=0.0, bar_style='info', description='Progress:',
    ↪layout=Layout(height='25px', width='50%'),...
```

[29]:

VBox()

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



7 PART 4 - Answer to the following questions:

You will need to join tables to answer the following questions. Your code should meet all the requirements given in the question.

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

```
[32]: # Filter namedf for Johnny Depp and Helena Bonham Carter
johnny_depp_id = namedf.filter(namedf.primaryName == "Johnny Depp").
    ↳select("nconst").distinct()
helena_bonham_carter_id = namedf.filter(namedf.primaryName == "Helena Bonham_
    ↳Carter").select("nconst").distinct()

# Get the movie titles (tconst) where Johnny Depp acted
johnny_depp_titles = titleprincdf.join(johnny_depp_id, "nconst").
    ↳select("tconst").distinct()

# Get the movie titles (tconst) where Helena Bonham Carter acted
helena_bonham_carter_titles = titleprincdf.join(helena_bonham_carter_id,
    ↳"nconst").select("tconst").distinct()
```

```
# Find common titles in which both acted
common_titles = johnny_depp_titles.intersect(helena_bonham_carter_titles)

# Join with titledf to get movie names
movies_both_acted = common_titles.join(titledf, "tconst").
    ↪filter(col("titleType")=="movie").select("primaryTitle")

movies_both_acted.show(truncate=False)
```

VBox()

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

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

[30]:

VBox()

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

```
+-----+
|primaryTitle|
+-----+
|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|
+-----+
```

7.2 2) What are the movies in which Brad Pitt has acted since 2010?

```
[34]: # Filter for Brad Pitt's ID
brad_pitt_id = namedf.filter(namedf.primaryName == "Brad Pitt").
    ↳select("nconst").distinct()

# Get movie titles where Brad Pitt acted
brad_pitt_titles = titleprincdf.join(brad_pitt_id, "nconst").
    ↳filter(col("category") == "actor").select("tconst")

# Filter titledf for movies since 2010 and where startYear is not '\N'
movies_since_2010 = titledf.filter((col("titleType") == "movie") &↳
    ↳(col("startYear") > "2010") & (col("startYear") != r"\N"))

# Join to get Brad Pitt's movies since 2010
brad_pitt_movies_since_2010 = brad_pitt_titles.join(movies_since_2010,↳
    ↳"tconst").select("primaryTitle", "startYear")

brad_pitt_movies_since_2010.orderBy(col("startYear").desc()).
    ↳show(truncate=False)
```

VBox()

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

```
+-----+-----+
|primaryTitle          |startYear|
+-----+-----+
|Babylon                |2021    |
|Once Upon a Time ...  |2019    |
|Ad Astra               |2019    |
|War Machine            |2017    |
|Allied                 |2016    |
|The Big Short          |2015    |
|By the Sea             |2015    |
|Fury                   |2014    |
|World War Z            |2013    |
|12 Years a Slave       |2013    |
|Killing Them Softly    |2012    |
|The Tree of Life       |2011    |
|Moneyball              |2011    |
+-----+-----+
```

[31]:

VBox()

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

primaryTitle	startYear
Babylon	2021
Once Upon a Time ... in Hollywood	2019
Ad Astra	2019
War Machine	2017
Allied	2016
The Big Short	2015
By the Sea	2015
Fury	2014
World War Z	2013
12 Years a Slave	2013
Killing Them Softly	2012
The Tree of Life	2011
Moneyball	2011

7.3 3) How many movies has Zendaya acted in each year?

```
[36]: # Filter for Zendaya's ID
zendaya_id = namedf.filter(namedf.primaryName == "Zendaya").select("nconst").
↳distinct()

# Get movie titles where Zendaya acted
zendaya_titles = titleprincdf.join(zendaya_id, "nconst").select("tconst")

# Filter titledf for movies and join with Zendaya's movie titles
zendaya_movies = titledf.filter((col("titleType") == "movie") &
↳(col("startYear") != r"\N")).join(zendaya_titles, "tconst").select("tconst",
↳"startYear")

# Group by year and count the number of movies
movies_per_year = zendaya_movies.groupBy("startYear").count().
↳orderBy("startYear")

movies_per_year.orderBy(col("startYear").desc()).show(truncate=False)
```

```
VBox()
```

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

startYear	count
-----------	-------

```

+-----+-----+
|2020    |1    |
|2018    |2    |
|2017    |1    |
+-----+-----+

```

[32]:

```

VBox()

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

```

```

+-----+-----+
|startYear|count|
+-----+-----+
|2020     |1    |
|2018     |2    |
|2017     |1    |
+-----+-----+

```

7.4 4) Which movies, released in 2019, have an average rating exceeding 9.7?

```

[40]: # Filter titledf for movies released in 2019 and where startYear is not '\N'
movies_2019 = titledf.filter((col("titleType") == "movie") & (col("startYear")
↳== "2019") & (col("startYear") != r"\N"))

# Join with titleratdf to get the ratings
movies_2019_with_ratings = movies_2019.join(titleratdf, "tconst")

# Changing average rating to float
movies_2019_with_ratings = movies_2019_with_ratings.withColumn("averageRating",
↳col("averageRating").cast("float")).select("primaryTitle", "averageRating")

# Filter for movies with an average rating greater than 9.7
highly_rated_movies_2019 = movies_2019_with_ratings.filter(col("averageRating")
↳> 9.7)

# Sort by averageRating
highly_rated_movies_2019_sorted = highly_rated_movies_2019.
↳orderBy(col("averageRating").desc())

highly_rated_movies_2019_sorted.show(truncate=False)

```

```

VBox()

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

```

primaryTitle	averageRating
Bu Can Var Oldugu Sürece	10.0
L'Enfant Terrible	10.0
A Grunt's Life	10.0
Kirket	10.0
Our Scripted Life	10.0
The Twilight Zone: A 60th Anniversary Celebration	10.0
The Butcher Baronet	10.0
A Medicine for the Mind	10.0
Love in Kilnerry	10.0
Superhombre	9.9
Puritan: All of Life to The Glory of God	9.9
The Cardinal	9.9
Square One	9.8
Kamen Rider Zi-0: Over Quartzer	9.8
Time and motion	9.8
We Shall Not Die Now	9.8
From Shock to Awe	9.8
Randhawa	9.8
Gini Helida Kathe	9.8

[33]:

VBox()

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

PrimaryTitle	averageRating
Bu Can Var Oldugu Sürece	10.0
L'Enfant Terrible	10.0
A Grunt's Life	10.0
Kirket	10.0
Our Scripted Life	10.0
The Twilight Zone: A 60th Anniversary Celebration	10.0
The Butcher Baronet	10.0
A Medicine for the Mind	10.0
Love in Kilnerry	10.0
Superhombre	9.9
Puritan: All of Life to The Glory of God	9.9
The Cardinal	9.9
Square One	9.8
Kamen Rider Zi-0: Over Quartzer	9.8
Time and motion	9.8

We Shall Not Die Now	9.8	
From Shock to Awe	9.8	
Randhawa	9.8	
Gini Helida Kathe	9.8	
+-----+	+-----+	+-----+

7.5 5) Among the titles in which Clint Eastwood and Harrison Ford have acted, who has the higher average rating?

First, calculate the average rating of each actor and then show the actor with higher average rating.

```
[45]: # Filter for Clint Eastwood and Harrison Ford's IDs
actors_ids = namedf.filter(namedf.primaryName.isin("Clint Eastwood", "Harrison_
↳Ford")).select("nconst", "primaryName")

# Get titles for each actor
actor_titles = titleprincdf.join(actors_ids, "nconst").select("tconst",
↳"primaryName")

# Join with titleratdf to get ratings
actor_titles_with_ratings = actor_titles.join(titleratdf, "tconst")

# Group by actor and calculate average rating
average_ratings_per_actor = actor_titles_with_ratings.groupBy("primaryName").
↳agg(round(avg("averageRating"),2).alias("averageRating"))

average_ratings = average_ratings_per_actor.collect()

for actor in average_ratings:
    print(f"The average rating of {actor['primaryName']} is_
↳{actor['averageRating']}")

if average_ratings[0]['averageRating'] > average_ratings[1]['averageRating']:
    higherRatedActor = average_ratings[0]['primaryName']
else:
    higherRatedActor = average_ratings[1]['primaryName']

print(f"{higherRatedActor} has a higher average rating")
```

VBox()

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

The average rating of Clint Eastwood is 7.32

The average rating of Harrison Ford is 6.76

Clint Eastwood has a higher average rating

[34]:

```
VBox()

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

The average rating of Harrison Ford is 6.75
The average rating of Clint Eastwood is 7.71
Clint Eastwood has a higher average rating
```

7.6 6) What is the movie(s) with the highest average rating among those in which Chris Evans has acted?

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

[41]:

```
# Filter for Chris Evans's ID
chris_evans_id = namedf.filter(namedf.primaryName == "Chris Evans").
↳select("nconst").distinct()

# Get titles where Chris Evans acted
chris_evans_titles = titleprincdf.join(chris_evans_id, "nconst").
↳select("tconst")

# Join with titledf to filter for movies and exclude '\N' in startYear
chris_evans_movies = chris_evans_titles.join(titledf.filter((col("titleType")
↳== "movie") & (col("startYear") != r"\N")), "tconst")

# Join with titleratdf to get the ratings
chris_evans_movies_with_ratings = chris_evans_movies.join(titleratdf, "tconst").
↳select("primaryTitle", "averageRating")

# Find the highest average rating
max_rating = chris_evans_movies_with_ratings.agg(max("averageRating")).
↳first()[0]

# Filter for movies with the highest average rating
top_chris_evans_movies = chris_evans_movies_with_ratings.
↳filter(col("averageRating") == max_rating)

top_chris_evans_movies.show(truncate=False)
```

```
VBox()

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

```

+-----+-----+
|primaryTitle      |averageRating|
+-----+-----+
|Avengers: Infinity War|8.5          |
|Avengers: Endgame    |8.5          |
+-----+-----+

```

[35]:

VBox()

```

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

```

```

+-----+-----+
|primaryTitle      |averageRating|
+-----+-----+
|Avengers: Infinity War|8.5          |
|Avengers: Endgame    |8.5          |
+-----+-----+

```

7.7 7) What is the percentage of adult titles in which actors and actresses have acted?

```
[44]: # Join two dataframes
all_info = titledf.join(titleprincdf, "tconst")

actors_temp = all_info.select('category', 'isAdult', 'primaryTitle')\
    .filter(col('category') == 'actor')
actors_count = actors_temp.count()

actress_temp = all_info.select('category', 'isAdult', 'primaryTitle')\
    .filter(col('category') == 'actress')
actress_count = actress_temp.count()

actors_adult = actors_temp.select('isAdult', 'primaryTitle')\
    .filter(col('isAdult') == '1')\
    .count()
actress_adult = actress_temp.select('isAdult', 'primaryTitle')\
    .filter(col('isAdult') == '1')\
    .count()

perc_actor = actors_adult/actors_count*100
perc_actress = actress_adult/actress_count*100

print(f'The percentage of adult titles actors have acted is: {perc_actor:.2f}%')
print(f'The percentage of adult titles actresses have acted is: {perc_actress:.2f}%')
```

VBox()

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

The percentage of adult titles actors have acted is: 3.00%

The percentage of adult titles actresses have acted is: 7.18%

[36]:

VBox()

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

The percentage of adult titles actors have acted is: 2.14%

The percentage of adult titles actresses have acted is: 3.27%

7.8 8) What are the top 10 movie genres with the shortest average runtime?

You do not need to join tables in this question.

```
[43]: # Filter titledf for movies and valid runtimes
movies_with_runtime = titledf.filter((col("titleType") == "movie") &
    ↪(col("runtimeMinutes").cast("int").isNotNull()))

# Split the 'genres' column into individual genres and explode it into separate
    ↪rows
exploded_genres = movies_with_runtime.withColumn("genre",
    ↪explode(split(col("genres"), ",")))

exploded_genres=exploded_genres.filter(col("genre") != "\\N")

# Calculate the average runtime for each genre
average_runtime_per_genre = exploded_genres.groupBy("genre").
    ↪agg(round(avg(col("runtimeMinutes").cast("int")),2).alias("averageRuntime"))

# Sort by average runtime
top_10_genres_shortest_runtime = average_runtime_per_genre.
    ↪orderBy("averageRuntime")

top_10_genres_shortest_runtime.show(10)
```

VBox()

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

```
+-----+-----+
|      genre|averageRuntime|
+-----+-----+
|      Short|          18.67|
|      News|          66.85|
|Documentary|          72.68|
| Reality-TV|          76.28|
|      Adult|          77.18|
|    Western|          79.36|
|  Talk-Show|          79.43|
| Animation|          80.75|
| Biography|          81.05|
|  Film-Noir|          83.76|
+-----+-----+
only showing top 10 rows
```

[37]:

VBox()

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

genre	avg(runtimeMinutes)
Short	18.67
News	66.85
Documentary	72.68
Reality-TV	76.28
Adult	77.18
Western	79.36
Talk-Show	79.43
Animation	80.75
Biography	81.05
Film-Noir	83.76

only showing top 10 rows

7.9 9) What are the most common character names for actors and actresses in Romance movies?

```
[51]: # Filter for Romance movies
romance_titles = titledf.filter(col("genres").like("%Romance%") &
    ↳ (col("titleType") == "movie"))

# Join with titleprincdf for actor and actress roles
actor_roles = titleprincdf.join(romance_titles, "tconst").
    ↳ filter(col("category") == "actor")
actress_roles = titleprincdf.join(romance_titles, "tconst").
    ↳ filter(col("category") == "actress")

# Explode the characters column and clean up the names
actor_roles_exploded = actor_roles.withColumn("character",
    ↳ explode(split(col("characters"), r"\s*,\s*")))
    ↳ filter(col("character") !=
    ↳ "\N")
actress_roles_exploded = actress_roles.withColumn("character",
    ↳ explode(split(col("characters"), r"\s*,\s*")))
    ↳ filter(col("character") !=
    ↳ "\N")

# Group by character name and count, then order by count and added category
    ↳ column
top_actor = actor_roles_exploded.groupBy("character").count().
    ↳ orderBy(col("count").desc()).limit(1).withColumn("category", lit("actor"))
top_actress = actress_roles_exploded.groupBy("character").count().
    ↳ orderBy(col("count").desc()).limit(1).withColumn("category", lit("actress"))

# Union the top actor and actress results into one DataFrame
```

```
top_characters = top_actress.union(top_actor).select("category", "count",  
↳ "character")  
  
top_characters.show()
```

VBox()

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

```
+-----+-----+-----+  
|category|count|character|  
+-----+-----+-----+  
| actress|  228| ["Anna"] |  
|   actor|  182| ["David"] |  
+-----+-----+-----+
```

[38]:

VBox()

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

```
+-----+-----+-----+  
|category|count|characters|  
+-----+-----+-----+  
| actress|  228| ["Anna"] |  
|   actor|  182| ["David"] |  
+-----+-----+-----+
```