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 (https://www.kaggle.com/datasets/ashirwadsangwan/imdb-dataset)) although we have taken steps to pull this data into a publis s3 bucket:

- s3://cis9760-lecture9-movieanalysis/name.basics.tsv ---> (actors)
- s3://cis9760-lecture9-movieanalysis/title.basics.tsv ---> (genres)
- s3://cis9760-lecture9-movieanalysis/title.principals.tsv ---> (movie actors)
- s3://cis9760-lecture9-movieanalysis/title.ratings.tsv ---> (movie ratings)

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.

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

In [1]:

```
%%info
```

Current session configs: `{'conf': {'spark.pyspark.python': 'python3', 'spark.pyspark.virtualenv.enabled': 'true', 'spark.pyspark.virtualenv.type': 'native', 'spark.pyspark.virtualenv.bin.path': '/usr/bin/virtualenv'}, 'kind': 'pyspark'}`

No active sessions.

Let's install the necessary packages here

In [2]:

```
sc.install_pypi_package("pandas==1.0.3")
sc.install_pypi_package("matplotlib==3.2.1")
```

▸ Spark Job Progress

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	
3	application_1669595400244_0004	pyspark	idle	Link 62.ec2.internal:20888/proxy/application_16

```
SparkSession available as 'spark'.

Collecting pandas==1.0.3
  Using cached https://files.pythonhosted.org/packages/4a/6a/94b2
19b8ea0f2d580169e85ed1edc0163743f55aaeca8a44c2e8fc1e344e/pandas-
1.0.3-cp37-cp37m-manylinux1_x86_64.whl (https://files.pythonhoste
d.org/packages/4a/6a/94b219b8ea0f2d580169e85ed1edc0163743f55aaeca
8a44c2e8fc1e344e/pandas-1.0.3-cp37-cp37m-manylinux1_x86_64.whl)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/pyt
hon3.7/site-packages (from pandas==1.0.3)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib64/
python3.7/site-packages (from pandas==1.0.3)
Collecting python-dateutil>=2.6.1 (from pandas==1.0.3)
  Using cached https://files.pythonhosted.org/packages/36/7a/8783
7f39d0296e723bb9b62bbb257d0355c7f6128853c78955f57342a56d/python_d
ateutil-2.8.2-py2.py3-none-any.whl (https://files.pythonhosted.or
g/packages/36/7a/87837f39d0296e723bb9b62bbb257d0355c7f6128853c789
55f57342a56d/python_dateutil-2.8.2-py2.py3-none-any.whl)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python
3.7/site-packages (from python-dateutil>=2.6.1->pandas==1.0.3)
Installing collected packages: python-dateutil, pandas
Successfully installed pandas-1.0.3 python-dateutil-2.8.2

Collecting matplotlib==3.2.1
  Using cached https://files.pythonhosted.org/packages/b2/c2/71fc
f957710f3ba1f09088b35776a799ba7dd95f7c2b195ec800933b276b/matplotl
ib-3.2.1-cp37-cp37m-manylinux1_x86_64.whl (https://files.pythonho
sted.org/packages/b2/c2/71fcf957710f3ba1f09088b35776a799ba7dd95f7
c2b195ec800933b276b/matplotlib-3.2.1-cp37-cp37m-manylinux1_x86_6
4.whl)
Requirement already satisfied: python-dateutil>=2.1 in /mnt/tmp/1
669611076034-0/lib/python3.7/site-packages (from matplotlib==3.2.
1)
Collecting pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 (from matplot
lib==3.2.1)
  Using cached https://files.pythonhosted.org/packages/6c/10/a7d0
```

```
fa5baea8fe7b50f448ab742f26f52b80bfca85ac2be9d35cdd9a3246/pyparsin
g-3.0.9-py3-none-any.whl (https://files.pythonhosted.org/package
s/6c/10/a7d0fa5baea8fe7b50f448ab742f26f52b80bfca85ac2be9d35cdd9a3
246/pyparsing-3.0.9-py3-none-any.whl)
Collecting cycler>=0.10 (from matplotlib==3.2.1)
  Using cached https://files.pythonhosted.org/packages/5c/f9/695d
6bedebd747e5eb0fe8fad57b72fdf25411273a39791cde838d5a8f51/cycler-
0.11.0-py3-none-any.whl (https://files.pythonhosted.org/packages/
5c/f9/695d6bedebd747e5eb0fe8fad57b72fdf25411273a39791cde838d5a8f5
1/cycler-0.11.0-py3-none-any.whl)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib64/py
thon3.7/site-packages (from matplotlib==3.2.1)
Collecting kiwisolver>=1.0.1 (from matplotlib==3.2.1)
  Using cached https://files.pythonhosted.org/packages/ab/8f/8dbe
2d4efc4c0b08ec67d6efb7cc31fbfd688c80afad85f65980633b0d37/kiwisolv
er-1.4.4-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.whl (h
ttps://files.pythonhosted.org/packages/ab/8f/8dbe2d4efc4c0b08ec67
d6efb7cc31fbfd688c80afad85f65980633b0d37/kiwisolver-1.4.4-cp37-cp
37m-manylinux_2_5_x86_64.manylinux1_x86_64.whl)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python
3.7/site-packages (from python-dateutil>=2.1->matplotlib==3.2.1)
Collecting typing-extensions; python_version < "3.8" (from kiwiso
lver>=1.0.1->matplotlib==3.2.1)
  Using cached https://files.pythonhosted.org/packages/0b/8e/f1a0
a5a76cfef77e1eb6004cb49e5f8d72634da638420b9ea492ce8305e8/typing_e
xtensions-4.4.0-py3-none-any.whl (https://files.pythonhosted.org/
packages/0b/8e/f1a0a5a76cfef77e1eb6004cb49e5f8d72634da638420b9ea4
92ce8305e8/typing_extensions-4.4.0-py3-none-any.whl)
Installing collected packages: pyparsing, cycler, typing-extensio
ns, kiwisolver, matplotlib
Successfully installed cycler-0.11.0 kiwisolver-1.4.4 matplotlib-
3.2.1 pyparsing-3.0.9 typing-extensions-4.4.0
```

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

In [3]:

```
import pandas as pd
import matplotlib as plt
```

Loading Data

Load all data from S3 into a Spark dataframe object

In [4]:

```
actors = spark.read.csv('s3://cis9760-lecture9-movieanalysis/name.basics.tsv',
genres = spark.read.csv('s3://cis9760-lecture9-movieanalysis/title.basics.tsv'
movie_actors = spark.read.csv('s3://cis9760-lecture9-movieanalysis/title.princ
movie_ratings = spark.read.csv('s3://cis9760-lecture9-movieanalysis/title.rati
```

▸ Spark Job Progress

Actors

Display the schema below:

In [5]:

```
actors.printSchema()
```

```
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`

In [6]:

```
actors.select("primaryName","birthYear","deathYear","knownForTitles").show(5)
```

▸ Spark Job Progress

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

Genres

Display the first 10 rows with the following columns:

- `titleType`
- `primaryTitle`
- `genres`

In [7]:

```
genres.select("titleType","primaryTitle","genres").show(10)
```

▸ Spark Job Progress

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

Display the unique categories below:

In [8]:

```
genres.select("titleType").distinct().show()
```

▸ Spark Job Progress

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

Display the schema below:

In [9]:

```
genres.printSchema()
```

```
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)
```

Movie Actors

Display the schema below:

In [10]:

```
movie_actors.printSchema()
```

```
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 10 rows below

In [11]:

```
movie_actors.show(10)
```

▸ Spark Job Progress

```
+---------+--------+---------+---------------+-------------------
-+-----------+
|   tconst|ordering|   nconst|       category|                  jo
b| characters|
+---------+--------+---------+---------------+-------------------
-+-----------+
|tt0000001|       1|nm1588970|           self|
\N|["Herself"]|
|tt0000001|       2|nm0005690|       director|
\N|         \N|
|tt0000001|       3|nm0374658|cinematographer|director of phot
o...|         \N|
|tt0000002|       1|nm0721526|       director|
\N|         \N|
|tt0000002|       2|nm1335271|       composer|
\N|         \N|
|tt0000003|       1|nm0721526|       director|
\N|         \N|
|tt0000003|       2|nm5442194|       producer|            produce
r|         \N|
|tt0000003|       3|nm1335271|       composer|
\N|         \N|
|tt0000003|       4|nm5442200|         editor|
\N|         \N|
|tt0000004|       1|nm0721526|       director|
\N|         \N|
+---------+--------+---------+---------------+-------------------
-+-----------+
only showing top 10 rows
```

Movie Ratings

Display the schema below:

In [12]:

```
movie_ratings.printSchema()
```

```
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

In [13]:

```
from pyspark.sql.functions import col
movie_ratings.select("tconst", "averageRating", "numVotes").sort(col("numVotes
```

▸ Spark Job Progress

```
+---------+-------------+--------+
|   tconst|averageRating|numVotes|
+---------+-------------+--------+
|tt7430722|          6.8|    9999|
|tt4445154|          8.1|    9997|
|tt2229907|          6.3|    9996|
|tt0294097|          8.0|    9994|
|tt0264734|          6.5|    9993|
|tt8860450|          6.3|    9991|
|tt2032572|          5.2|    9991|
|tt0664505|          8.4|     999|
|tt7508752|          7.9|     999|
|tt1077089|          7.3|     999|
+---------+-------------+--------+
only showing top 10 rows
```

Overview of Data

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

In [14]:

```
print(f"Number of columns in Actors table: {len(actors.dtypes)}")
print(f"Number of rows in Actors table: {actors.count()}\n")

print(f"Number of columns in Genres table: {len(genres.dtypes)}")
print(f"Number of rows in Genress table: {genres.count()}\n")

print(f"Number of columns in Movie Actors table: {len(movie_actors.dtypes)}")
print(f"Number of rows in Movie Actors table: {movie_actors.count()}\n")

print(f"Number of columns in Movie Ratings table: {len(movie_ratings.dtypes)}"
print(f"Number of rows in Movie Ratings table: {movie_ratings.count()}")
```

▸ Spark Job Progress

```
Number of columns in Actors table: 6
Number of rows in Actors table: 9706922

Number of columns in Genres table: 9
Number of rows in Genress table: 6321302

Number of columns in Movie Actors table: 6
Number of rows in Movie Actors table: 36468817

Number of columns in Movie Ratings table: 3
Number of rows in Movie Ratings table: 993153
```

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?

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

In [15]:

```
genres.select("tconst","titleType","genres").show(5,truncate=False)
```

▸ Spark Job Progress

```
+---------+---------+------------------------+
|tconst   |titleType|genres                  |
+---------+---------+------------------------+
|tt0000001|short    |Documentary,Short       |
|tt0000002|short    |Animation,Short         |
|tt0000003|short    |Animation,Comedy,Romance|
|tt0000004|short    |Animation,Short         |
|tt0000005|short    |Comedy,Short            |
+---------+---------+------------------------+
only showing top 5 rows
```

Display the first 10 rows of your association table below

In [16]:

```
import pyspark.sql.functions as F
genres_result = (genres
            .withColumn('genres', F.explode(F.split('genres',',')))
           )

genres_result.select("tconst", "titleType", "genres").show(10)
```

▸ Spark Job Progress

```
+---------+---------+-----------+
|   tconst|titleType|     genres|
+---------+---------+-----------+
|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|
+---------+---------+-----------+
only showing top 10 rows
```

Total Unique Genres

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

In [17]:

```
genres_result.filter(genres_result.titleType == "movie").select("genres").dist
```

▸ Spark Job Progress

```
29
```

What are the unique genres available?

```
In [18]:
```

```
genres_result.select("genres").distinct().show()
```

▸ Spark Job Progress

```
+-----------+
|     genres|
+-----------+
|    Mystery|
|    Musical|
|      Sport|
|     Action|
|  Talk-Show|
|    Romance|
|   Thriller|
|         \N|
| Reality-TV|
|     Family|
|    Fantasy|
|    History|
|  Animation|
|  Film-Noir|
|      Short|
|     Sci-Fi|
|       News|
|      Drama|
|Documentary|
|    Western|
+-----------+
only showing top 20 rows
```

Oops! Something is off!

In [19]:

```
nll= '\\N'
genres_result.select("genres").filter(col("genres") != nll).distinct().show()
```

▸ Spark Job Progress

```
+-----------+
|     genres|
+-----------+
|    Mystery|
|    Musical|
|      Sport|
|     Action|
|  Talk-Show|
|    Romance|
|   Thriller|
| Reality-TV|
|     Family|
|    Fantasy|
|    History|
|  Animation|
|  Film-Noir|
```

In [20]:

```
nll= '\\N'
genres_result.filter(genres_result.titleType == "movie").select("genres").filt
```

▸ Spark Job Progress

```
28
```

Top Genres by Movies

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

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

genre	averageRating
b	6.3
c	7.2

Or something to that effect.

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

In [21]:

```
_genre = genres_result.join(movie_ratings, on=["tconst"],how = "inner")
_genre.select("genres","averageRating").filter(F.col("genres") != nll).filter(r
```

▸ Spark Job Progress

```
+---------+-------------+
|   genres|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.

In [22]:

```
nll = '\\N'
rating_and_genre.select("genres","averageRating")\
    .withColumn("averageRating", F.col("averageRating").cast("float"))\
    .filter(F.col("genres") != nll)\
    .filter(F.col("titleType") == "movie")\
    .groupBy("genres").agg(F.avg("averageRating").alias("avg_rating"),)\
    .show(truncate=False)
```

▸ Spark Job Progress

```
+-----------+-----------------+
|genres     |avg_rating       |
+-----------+-----------------+
|Mystery    |5.940437537126316|
|Musical    |6.203246053185319|
|Action     |5.718734067904495|
|Sport      |6.600145190943391|
|Talk-Show  |5.800000190734863|
|Romance    |6.125714179294426|
|Thriller   |5.625967567519544|
|Reality-TV |6.379310377712907|
|Family     |6.250560452699635|
|Fantasy    |5.924820762891499|
|History    |6.822718117193864|
|Animation  |6.326203749467441|
|Film-Noir  |6.636246780503378|
```

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!

```
%matplot plt
```

In [23]:

```
nll = '\\N'
rating_and_genre.select("genres","averageRating")\
    .withColumn("averageRating", F.col("averageRating").cast("float"))\
    .filter(F.col("genres") != nll)\
    .filter(F.col("titleType") == "movie")\
    .groupBy("genres").agg(F.avg("averageRating").alias("avg_rating"),)\
    .sort(F.desc("avg_rating"))\
    .show(truncate=False)
```

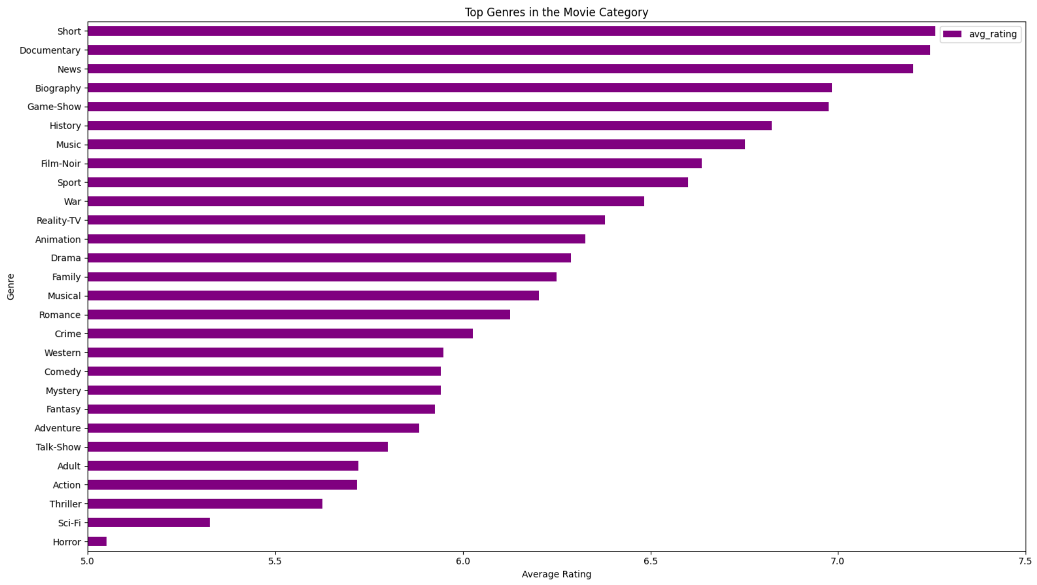
▸ Spark Job Progress

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

In [24]:

```
import matplotlib.pyplot as plt

nll = '\\N'
x = rating_and_genre.select("genres","averageRating")\
    .withColumn("averageRating", F.col("averageRating").cast("float"))\
    .filter(F.col("genres") != nll)\
    .filter(F.col("titleType") == "movie")\
    .groupBy("genres").agg(F.avg("averageRating").alias("avg_rating"),)\
    .sort(F.asc("avg_rating")).toPandas().plot.barh(color='purple')
x.set_xlabel("Average Rating")
x.set_ylabel("Genre")
x.set_title("Top Genres in the Movie Category")

temp = rating_and_genre.select("genres","averageRating")\
    .withColumn("averageRating", F.col("averageRating").cast("float"))\
    .filter(F.col("genres") != nll)\
    .filter(F.col("titleType") == "movie")\
    .groupBy("genres").agg(F.avg("averageRating").alias("avg_rating"),)\
    .sort(F.asc("avg_rating"))

labels = temp.select('genres').collect()
labels = [labels[i]["genres"] for i in range(len(labels))]

x.set_yticklabels(labels)
x.set_xlim([5, 7.5])

fig = plt.gcf()
fig.set_size_inches(18.5, 10.5)
%matplot plt
```

▸ Spark Job Progress



PART 3 - Analyzing Job Categories

Total Unique Job Categories

What is the total number of unique job categories?

In [25]:

```
movie_actors.select("tconst","category").distinct().show(5)
```

▸ Spark Job Progress

```
+---------+---------------+
|   tconst|       category|
+---------+---------------+
|tt0000826|cinematographer|
|tt0001014|cinematographer|
|tt0001150|        actress|
|tt0002234|         writer|
|tt0002401|       director|
+---------+---------------+
only showing top 5 rows
```

In [26]:

```
movie_actors.select("category").distinct().count()
```

▸ Spark Job Progress

```
12
```

What are the unique job categories available?

In [27]:

```
movie_actors.select("category").distinct().show()
```

▸ Spark Job Progress

```
+-------------------+
|           category|
+-------------------+
|            actress|
|           producer|
|production_designer|
|             writer|
|              actor|
|    cinematographer|
|      archive_sound|
|    archive_footage|
|               self|
|             editor|
|           composer|
|           director|
+-------------------+
```

Top Job Categories

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

Counts of Titles / Job Category

The expected output should be:

category	count
a	15

category	count
b	2
c	45

Or something to that effect.

In [28]:

```
from pyspark.sql.functions import mean, stddev, col, abs, split, explode

job_categories = movie_actors.select('tconst','category').withColumn("category
print(job_categories.groupBy('category').count().show())
```

▸ Spark Job Progress

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

None
```

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!

```
%matplot plt
```

In [29]:

```
job_categories_count = job_categories.groupBy('category').count()
job_categories_top_5 = job_categories_count.sort("count", ascending=False).lim
job_categories_top_5.show()
```

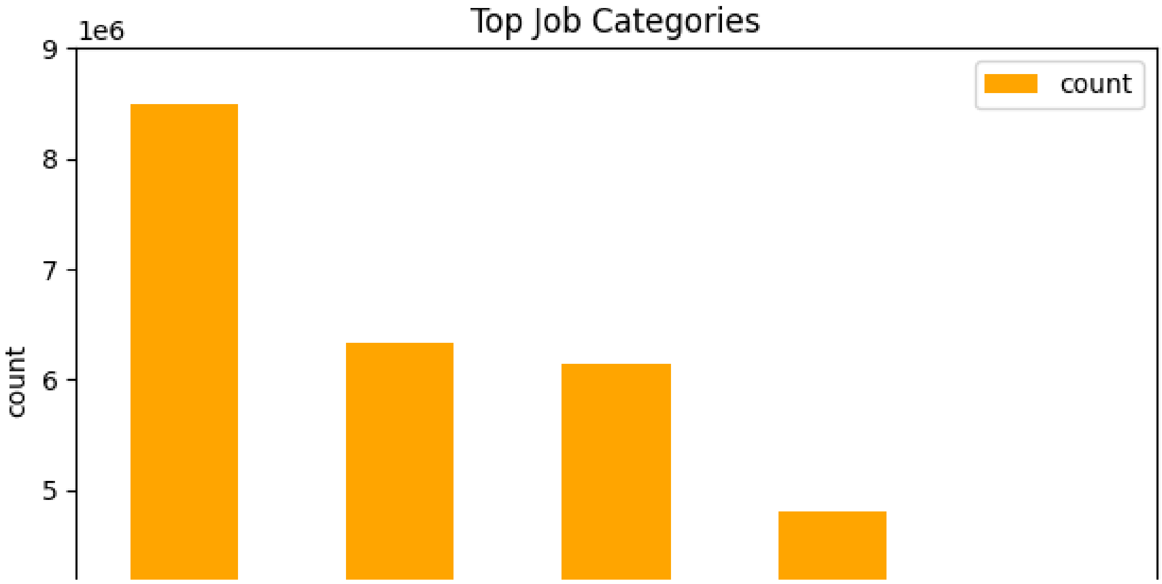
▸ Spark Job Progress

```
+--------+-------+
|category|  count|
+--------+-------+
|   actor|8493701|
| actress|6325097|
|    self|6153089|
|  writer|4811596|
|director|4179106|
+--------+-------+
```

In [30]:

```
job_categories_top_5_pd = job_categories_top_5.toPandas().set_index('category'

job_categories_top_5_pd.plot.bar(color='orange')
plt.title('Top Job Categories')
plt.ylabel("count")
plt.xlabel("categories")
plt.xticks(rotation = 45)
plt.ylim([3e6,9e6])
plt.tight_layout()
%matplot plt
```

▸ Spark Job Progress



PART 4 - Answer to the following questions:

1) Find all the "movies" featuring "Johnny Depp" and "Helena Bonham Carter".

First join actors, genres, and movie actors on each other

In [31]:

```
movie_actor_genre_join = movie_actors.join(genres, on=["tconst"],how = "inner"
move_actor_join = movie_actor_genre_join.join(actors, on=["nconst"],how = "inn

actor_johnnyd = move_actor_join.select("primaryTitle","primaryName").filter(mo

actor_helena = move_actor_join.select("primaryTitle","primaryName").filter(mov

dfmix = actor_johnnyd.join(actor_helena, on=["primaryTitle"],how = "inner")
dfmix.select("primaryTitle").show(truncate=False)
```

▸ Spark Job Progress

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

2) Find all the "movies" featuring "Brad Pitt" after 2010.

In [32]:

```python
nll = '\\N'
actor_bradp = move_actor_join.select("primaryTitle","primaryName","startYear")
actor_bradp.select("primaryTitle","startYear").sort(F.desc("startYear")).show(
```

▸ Spark Job Progress

```
+---------------------------------+---------+
|primaryTitle                     |startYear|
+---------------------------------+---------+
|Babylon                          |2021     |
|Kajillionaire                    |2020     |
|Irresistible                     |2020     |
|Ad Astra                         |2019     |
|The King                         |2019     |
|Once Upon a Time ... in Hollywood|2019     |
|Vice                             |2018     |
|War Machine                      |2017     |
|Voyage of Time: Life's Journey   |2016     |
|Allied                           |2016     |
|By the Sea                       |2015     |
|The Big Short                    |2015     |
|Hitting the Apex                 |2015     |
|Fury                             |2014     |
|12 Years a Slave                 |2013     |
|Kick-Ass 2                       |2013     |
|World War Z                      |2013     |
|Killing Them Softly              |2012     |
|Moneyball                        |2011     |
|The Tree of Life                 |2011     |
+---------------------------------+---------+
```

3) What is the number of "movies" "acted" by "Zendaya" per year?

In [33]:

```
actor_zendaya = move_actor_join.select("primaryTitle","primaryName","startYear

actor_zendaya.select('startYear')
print(actor_zendaya.groupBy('startYear').count().show())
```

▸ Spark Job Progress

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

None
```

4) What are the "movies" by average rating greater than "9.7" and released in "2019"?

In [34]:

```
movie_rating_join = move_actor_join.join(movie_ratings, on=["tconst"],how = "i
ratings= movie_rating_join.select("primaryTitle","startYear","averageRating").

ratings.select("primaryTitle","averageRating").sort(F.asc("averageRating")).sh
```

▸ Spark Job Progress

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

Extra Credit - Analysis of your choice

Try and analyze some interesting dimension to this data. You should specify the question in your Project2_Analysis.ipynb.

You must join at least two datasets.

In [ ]: