

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 ---> (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]: %%configure -f
{ "conf":{
    "spark.pyspark.python": "python3",
    "spark.pyspark.virtualenv.enabled": "true",
    "spark.pyspark.virtualenv.bin.path":"/usr/bin/virtualenv",
    "spark.pyspark.virtualenv.type":"native",
    "spark.dynamicAllocation.enabled": "true",
    "spark.shuffle.service.enabled": "true",
    "spark.dynamicAllocation.minExecutors": "8",
    "spark.sql.sources.partitionOverwriteMode": "dynamic",
    "spark.sql.legacy.allowCreatingManagedTableUsingNonemptyLocation": "true"
}}
```

```
Current session configs: {'conf': {'spark.pyspark.python': 'python3', 'spark.pyspark.virtualenv.enabled': 'true', 'spark.pyspark.virtualenv.bin.path': '/usr/bin/virtualenv', 'spark.pyspark.virtualenv.type': 'native', 'spark.dynamicAllocation.enabled': 'true', 'spark.shuffle.service.enabled': 'true', 'spark.dynamicAllocation.minExecutors': '8', 'spark.sql.sources.partitionOverwriteMode': 'dynamic', 'spark.sql.legacy.allowCreatingManagedTableUsingNonemptyLocation': 'true'}, 'kind': 'pyspark'}
```

No active sessions.

```
In [2]: sc.install_pypi_package("matplotlib==3.2.1")
sc.install_pypi_package("pandas==1.0.3")
# sc.install_pypi_package("seaborn")
```

Starting Spark application

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
2	application_1669607535363_0003	pyspark	idle	Link	Link	✓

SparkSession available as 'spark'.

```
Collecting matplotlib==3.2.1
  Using cached https://files.pythonhosted.org/packages/b2/c2/71fcf957710f3ba1f09088b35776a799ba7dd95f7c2b195ec800933b276
b/matplotlib-3.2.1-cp37-cp37m-manylinux1_x86_64.whl
Collecting python-dateutil>=2.1 (from matplotlib==3.2.1)
  Using cached https://files.pythonhosted.org/packages/36/7a/87837f39d0296e723bb9b62bbb257d0355c7f6128853c78955f57342a56
d/python_dateutil-2.8.2-py2.py3-none-any.whl
Collecting pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 (from matplotlib==3.2.1)
  Using cached https://files.pythonhosted.org/packages/6c/10/a7d0fa5baea8fe7b50f448ab742f26f52b80bfca85ac2be9d35cdd9a324
6/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/f9/695d6bedebd747e5eb0fe8fad57b72fdf25411273a39791cde838d5a8f5
1/cycler-0.11.0-py3-none-any.whl
Requirement already satisfied: numpy>=1.11 in /usr/local/lib64/python3.7/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/8dbe2d4efc4c0b08ec67d6efb7cc31fbfd688c80afad85f65980633b0d3
7/kiwisolver-1.4.4-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.whl
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/site-packages (from python-dateutil>=2.1->matplotlib=
=3.2.1)
Collecting typing-extensions; python_version < "3.8" (from kiwisolver>=1.0.1->matplotlib==3.2.1)
  Using cached https://files.pythonhosted.org/packages/0b/8e/f1a0a5a76cfef77e1eb6004cb49e5f8d72634da638420b9ea492ce8305e
8/typing_extensions-4.4.0-py3-none-any.whl
Installing collected packages: python-dateutil, pyparsing, cycler, typing-extensions, kiwisolver, matplotlib
Successfully installed cycler-0.11.0 kiwisolver-1.4.4 matplotlib-3.2.1 pyparsing-3.0.9 python-dateutil-2.8.2 typing-exten
sions-4.4.0
```

```
Collecting pandas==1.0.3
  Using cached https://files.pythonhosted.org/packages/4a/6a/94b219b8ea0f2d580169e85ed1edc0163743f55aaeca8a44c2e8fc1e344
e/pandas-1.0.3-cp37-cp37m-manylinux1_x86_64.whl
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/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)
Requirement already satisfied: python-dateutil>=2.6.1 in /mnt/tmp/1669609599812-0/lib/python3.7/site-packages (from pandas==1.0.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/site-packages (from python-dateutil>=2.6.1->pandas==1.0.3)
Installing collected packages: pandas
  Successfully installed pandas-1.0.3
```

Let's install the necessary packages here

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

```
In [3]:  
import pandas as pd  
import matplotlib.pyplot as plt  
from pyspark.sql.functions import mean, stddev, col, abs, split, explode, round  
from pyspark.sql import functions as F
```

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', sep=r'\t', header=True)  
genres = spark.read.csv('s3://cis9760-lecture9-movieanalysis/title.basics.tsv', sep=r'\t', header=True)  
movie_actors = spark.read.csv('s3://cis9760-lecture9-movieanalysis/title.principals.tsv', sep=r'\t', header=True)  
movie_ratings = spark.read.csv('s3://cis9760-lecture9-movieanalysis/title.ratings.tsv', sep=r'\t', header=True)
```

Actors

Display the schema below:

```
In [5]:  
actors.printSchema()  
actors.show(10)  
actors.count()
```

```

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)

+-----+-----+-----+-----+-----+
| nconst| primaryName|birthYear|deathYear| primaryProfession| knownForTitles|
+-----+-----+-----+-----+-----+
| nm0000001| Fred Astaire| 1899| 1987| soundtrack,actor,...| tt0050419,tt00531...
| nm0000002| Lauren Bacall| 1924| 2014| actress,soundtrack| tt0071877,tt01170...
| nm0000003| Brigitte Bardot| 1934| \N| actress,soundtrac...| tt0054452,tt00491...
| nm0000004| John Belushi| 1949| 1982| actor,writer,soun...| tt0077975,tt00725...
| nm0000005| Ingmar Bergman| 1918| 2007| writer,director,a...| tt0069467,tt00509...
| nm0000006| Ingrid Bergman| 1915| 1982| actress,soundtrac...| tt0038109,tt00368...
| nm0000007| Humphrey Bogart| 1899| 1957| actor,soundtrack,...| tt0043265,tt00338...
| nm0000008| Marlon Brando| 1924| 2004| actor,soundtrack,...| tt0070849,tt00787...
| nm0000009| Richard Burton| 1925| 1984| actor,producer,so...| tt0057877,tt00878...
| nm0000010| James Cagney| 1899| 1986| actor,soundtrack,...| tt0035575,tt00318...
+-----+-----+-----+-----+-----+
only showing top 10 rows

```

9706922

Display the first 5 rows with the following columns:

- primaryName
- birthYear
- deathYear
- knownForTitles

In [6]: `actors.select("primaryName", "birthYear", "deathYear", "knownForTitles").show(5)`

```

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

```
+-----+-----+-----+
| 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').withColumn("titleType", explode(split("titleType", ","))).distinct().show()
```

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

tconst	ordering	nconst	category	job	characters
tt0000001	1	nm1588970	self	\N	["Herself"]
tt0000001	2	nm0005690	director	\N	\N
tt0000001	3	nm0374658	cinematographer	director of photo...	\N
tt0000002	1	nm0721526	director	\N	\N
tt0000002	2	nm1335271	composer	\N	\N
tt0000003	1	nm0721526	director	\N	\N
tt0000003	2	nm5442194	producer	producer	\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]: movie_ratings.sort(col("numVotes").desc()).show(10)
```

tconst	averageRating	numVotes
tt7430722	6.8	9999
tt4445154	8.1	9997
tt2229907	6.3	9996
tt0294097	8.0	9994
tt0264734	6.5	9993
tt2032572	5.2	9991
tt8860450	6.3	9991
tt0025173	6.6	999
tt0245247	8.3	999

```
|tt0062690|          6.1|     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 rows in Actors table: {len(actors.columns)}')
print(f'Number of columns in Actors table: {actors.count()}')

print(f'Number of rows in Genres table: {len(genres.columns)}')
print(f'Number of columns in Genres table: {genres.count()}')

print(f'Number of rows in Movie Actors table: {len(movie_actors.columns)}')
print(f'Number of columns in Movie Actors table: {movie_actors.count()}')

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

```
Number of rows in Actors table: 6
Number of columns in Actors table: 9706922
Number of rows in Genres table: 9
Number of columns in Genres table: 6321302
Number of rows in Movie Actors table: 6
Number of columns in Movie Actors table: 36468817
Number of rows in Movie Ratings table: 3
Number of columns 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(2)
```

```
+-----+-----+
| tconst|titleType|      genres|
+-----+-----+
|tt0000001|    short|Documentary,Short|
|tt0000002|    short| Animation,Short|
+-----+-----+
only showing top 2 rows
```

Display the first 10 rows of your association table below

```
In [16]: genres.select('tconst', 'titleType', 'genres').withColumn("genres", explode(split("genres", ","))).show(10)
```

```

+-----+-----+
| 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]: num_of_unique_genres = genres.select('titleType', 'genres').withColumn("genres", explode(split("genres", ","))).select('genres')
print(f'Number of Unique genres: {num_of_unique_genres}')
```

Number of Unique genres: 29

What are the unique genres available?

```
In [18]: genres.select('genres').withColumn("genres", explode(split("genres", ","))).select('genres').distinct().show(truncate=False)
```

```

+-----+
|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]: genres_ref = genres.select('genres').withColumn("genres", explode(split("genres", ","))).select('genres').where(genres.ge
genres_ref.distinct().show(truncate=False)
```

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

```
In [20]: num_of_unique_genres = genres_ref.distinct().count()
```

```
print(f'Number of Unique genres: {num_of_unique_genres}')
```

Number of Unique genres: 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
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]: genres_tconst = genres.select('tconst','genres','titleType').withColumn("genres", explode(split("genres", ",","")))\n    .select('tconst','genres','titleType') \\ \n    .filter(genres.genres != '\\N') \\ \n    .filter(genres.titleType == 'movie')\n\nmovie_ratings.show(2)
```

```
+-----+-----+-----+\n| tconst|averageRating|numVotes|\n+-----+-----+-----+\n|tt0000001|      5.6|    1550|\n|tt0000002|      6.1|     186|\n+-----+-----+-----+\nonly showing top 2 rows
```

```
In [22]: inner_join_genres = genres_tconst.join(movie_ratings, on=['tconst'], how='inner') \
    .select('genres', 'averageRating') \
    .withColumn("averageRating", col("averageRating").cast("float")) \
    .withColumnRenamed("averageRating", "avg_rating")

inner_join_genres.show(10)
```

genres	avg_rating
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 [23]: genres_avgrating_genre = inner_join_genres.groupBy('genres').agg(mean('avg_rating')) \
    .withColumn("avg(avg_rating)", round("avg(avg_rating)", 14)) \
    .sort("avg(avg_rating)", ascending=False)
genres_avgrating_genre1 = genres_avgrating_genre.withColumnRenamed("avg(avg_rating)", "avg_rating")

genres_avgrating_genre1.show(truncate=False)
```

genres	avg_rating
Short	7.25999994277954
Documentary	7.2454698053711
News	7.20091604094469
Biography	6.98363764304459
Game-Show	6.97499990463257
History	6.82271811719386
Music	6.75202020721459

```

| Film-Noir | 6.63624678050338 |
| Sport     | 6.60014519094339 |
| War       | 6.4838070362784 |
| Reality-TV| 6.37931037771291 |
| Animation | 6.32620374946744 |
| Drama     | 6.28808021109754 |
| Family    | 6.25056045269964 |
| Musical   | 6.20324605318532 |
| Romance   | 6.12571417929443 |
| Crime     | 6.02601333310915 |
| Western   | 5.94897099100506 |
| Comedy    | 5.94136310782223 |
| Mystery   | 5.94043753712632 |
+-----+
only showing top 20 rows

```

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
```

In [24]:

```
genres_avgrating_genre1.show(truncate=False)
```

```

+-----+
| genres      | avg_rating   |
+-----+
| Short       | 7.25999994277954 |
| Documentary | 7.2454698053711 |
| News        | 7.20091604094469 |
| Biography   | 6.98363764304459 |
| Game-Show   | 6.97499990463257 |
| History     | 6.82271811719386 |
| Music       | 6.75202020721459 |
| Film-Noir   | 6.63624678050338 |
| Sport       | 6.60014519094339 |
| War         | 6.4838070362784 |
| Reality-TV  | 6.37931037771291 |
| Animation   | 6.32620374946744 |
| Drama       | 6.28808021109754 |
| Family      | 6.25056045269964 |
| Musical     | 6.20324605318532 |

```

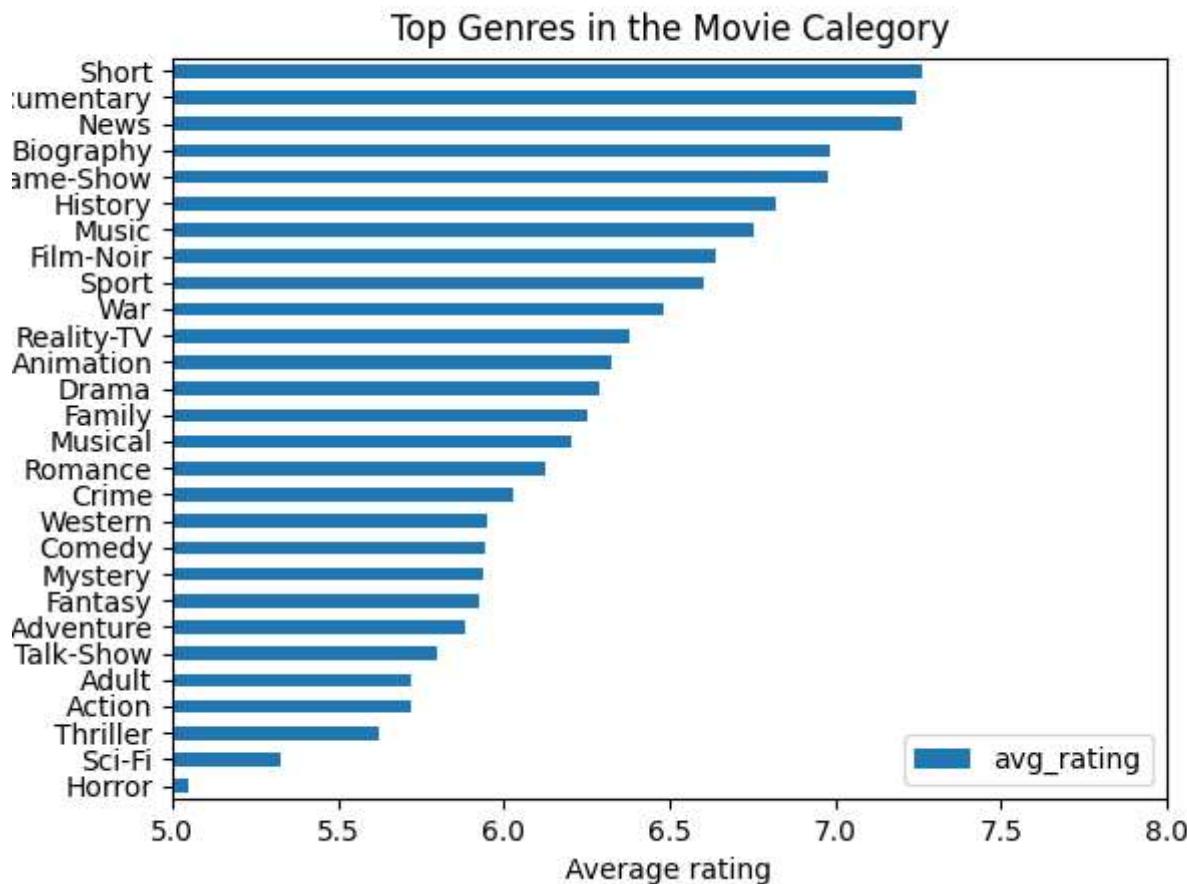
```
|Romance      |6.12571417929443|
|Crime        |6.02601333310915|
|Western       |5.94897099100506|
|Comedy        |5.94136310782223|
|Mystery       |5.94043753712632|
+-----+
only showing top 20 rows
```

In [25]:

```
genres_avgrating_genre_aver_all = genres_avgrating_genre1.sort("avg_rating", ascending=False).toPandas().set_index('genre')

# plot
genres_avgrating_genre_aver_all.plot.barh().invert_yaxis()
plt.title('Top Genres in the Movie Category')
plt.xlabel("Average rating")
plt.ylabel("genres")
plt.xlim(5.0, 8.0)
plt.rcParams["figure.figsize"] = [20, 8]

%matplotlib plt
```



PART 3 - Analyzing Job Categories

Total Unique Job Categories

What is the total number of unique job categories?

In [26]:

```
movie_actors.select('tconst','category').show(5)
```

```
+-----+-----+
| tconst|    category|
+-----+-----+
```

```
|tt0000001|      self|
|tt0000001|    director|
|tt0000001|cinematographer|
|tt0000002|    director|
|tt0000002|    composer|
+-----+
only showing top 5 rows
```

In [27]: `movie_actors.select('category').distinct().count()`

12

What are the unique job categories available?

In [28]: `movie_actors.select('category').distinct().show(truncate=False)`

```
+-----+
|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
b	2
c	45

Or something to that effect.

```
In [29]: job_categories = movie_actors.select('tconst','category') \
    .groupBy('category').count().show(truncate=False)
```

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

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
```

```
In [30]: job_categories = movie_actors.select('tconst','category') \
    .groupBy('category').count().sort("count", ascending=False)

job_categories.show(truncate=False)
```

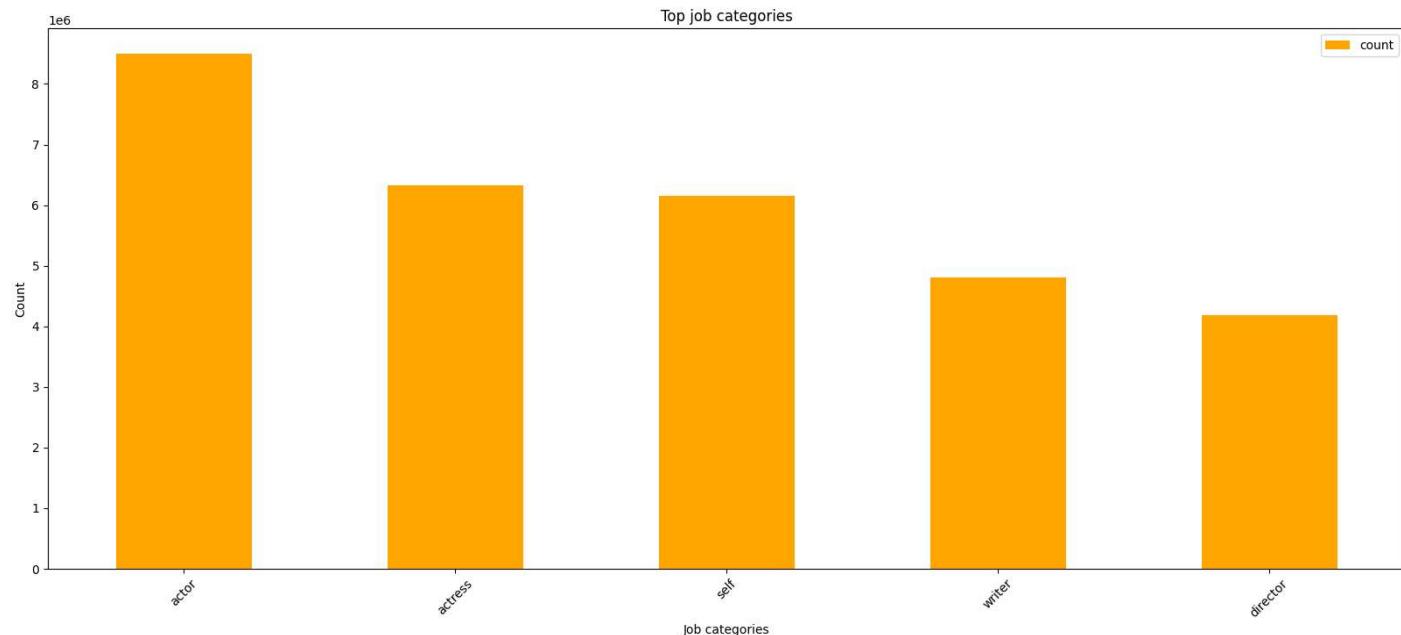
category	count
actor	8493701
actress	6325097
self	6153089
writer	4811596
director	4179106
producer	2197866
composer	1313187
cinematographer	1300404
editor	1197669
production_designer	285924
archive_footage	209035
archive_sound	2143

```
In [31]: job_categories_count_top20 = job_categories.sort("count", ascending=False).limit(5)

job_categories_count_top20_pandas = job_categories_count_top20.toPandas().set_index('category')

job_categories_count_top20_pandas.plot(kind="bar", color='orange')
plt.xticks(rotation=45, horizontalalignment="center")
plt.ylim = [0,9]
plt.title("Top job categories")
plt.xlabel("Job categories")
plt.ylabel("Count")
plt.rcParams["figure.figsize"] = [15, 9]

%matplotlib plt
```



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 [32]:

```
two_join_data = genres.join(movie_actors, on=['tconst'], how='inner') \
    .filter(genres.titleType=='movie') \
    .select(genres.primaryTitle, genres.titleType, movie_actors.nconst, genres.startYear)
three_join_data = two_join_data.join(actors, on=['nconst'], how='inner')

Johnny_Depp_movie = three_join_data.filter(three_join_data.primaryName == 'Johnny Depp').select('primaryTitle', 'primaryNa
Helena_Bonham_Carter_movie = three_join_data.filter(three_join_data.primaryName == 'Helena Bonham Carter').select('primary

data = Helena_Bonham_Carter_movie.join(Johnny_Depp_movie, Johnny_Depp_movie.primaryTitle == Helena_Bonham_Carter_movie.pri
data.show(truncate=False)
```

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

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

In [33]:

```
three_join_data.show(1)
```

```
+-----+-----+-----+-----+-----+-----+-----+
--+
| nconst| primaryTitle|titleType|startYear|      primaryName|birthYear|deathYear|      primaryProfession|      knownForTitl
es|
+-----+-----+-----+-----+-----+-----+-----+
--+
|nm0000059|Three Sisters|    movie|     1970|Laurence Olivier|     1907|     1989|actor,producer,di...|tt0069281,tt0040
4...
+-----+-----+-----+-----+-----+-----+-----+
--+
only showing top 1 row
```

In [34]:

```
Brad_Pitt_movie = three_join_data.filter(three_join_data.primaryName =='Brad Pitt') \
    .filter(three_join_data.startYear > 2010) \
    .select('primaryTitle', 'startYear') \
    .orderBy(three_join_data.startYear.asc()) \
    .show(truncate=False)
```

```
+-----+-----+
|primaryTitle          |startYear|
+-----+-----+
|Moneyball              |2011
|The Tree of Life       |2011
|Killing Them Softly    |2012
|Kick-Ass 2             |2013
|12 Years a Slave       |2013
|World War Z            |2013
+-----+
```

Fury	2014
By the Sea	2015
The Big Short	2015
Hitting the Apex	2015
Allied	2016
Voyage of Time: Life's Journey	2016
War Machine	2017
Vice	2018
The King	2019
Ad Astra	2019
Once Upon a Time ... in Hollywood	2019
Irresistible	2020
Kajillionaire	2020
Babylon	2021

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

```
In [35]: Zendaya_movie = three_join_data.filter(three_join_data.primaryName =='Zendaya') \
    .filter(three_join_data.startYear !='\N') \
    .select('primaryTitle', 'startYear') \
    .groupBy(three_join_data.startYear).count() \
    .show(truncate=False)
```

startYear	count
2020	1
2018	2
2017	1

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

```
In [ ]:
```

```
In [36]: ratings = movie_ratings.withColumn("averageRating", col("averageRating").cast("float")) \
    .sort("averageRating", ascending=False)
```

```

ratings = ratings.filter(ratings.averageRating > 9.7 )

genres_select = genres.withColumn("genres", explode(split("genres", ","))) \
    .withColumn("startYear", col("startYear").cast("int")) \
    .filter(genres.genres != '\N') \
    .filter(genres.titleType == 'movie')

genres_select = genres_select.filter(genres_select.startYear == 2019)

genres_join_movie_rating = genres_select.join(ratings, on=['tconst'], how='inner')

genres_join_movie_rating.select('primaryTitle', 'averageRating') \
    .distinct() \
    .sort("averageRating", ascending=False) \
    .show(truncate=False)

```

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

In []:

Extra Credit - 1

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 [37]:

```
movie_ratings = movie_ratings.withColumn("averageRating", col("averageRating").cast("float")) \
    .sort("averageRating", ascending=False)

genres = genres.withColumn("genres", explode(split("genres", ","))) \
    .withColumn("startYear", col("startYear").cast("int")) \
    .filter(genres.genres != '\N') \
    .filter(genres.titleType == 'movie')

genres_join_movie_rating = genres.join(movie_ratings, on=['tconst'], how='inner')

genres_join_movie_rating.select('primaryTitle', 'averageRating') \
    .distinct() \
    .sort("averageRating", ascending=False) \
    .show(truncate=False)
```

primaryTitle	averageRating
Our Scripted Life	10.0
Love in Kilmurry	10.0
A Medicine for the Mind	10.0
Girls Loving Girls	10.0
The Butcher Baronet	10.0
Petersburg's tram	10.0
The Twilight Zone: A 60th Anniversary Celebration	10.0
The Maltese Phallus	10.0
Filthy Frank Final Full Lore Movie	10.0
Callao	10.0
Centro de Dia	10.0
Bu Can Var Oldugu Sürece	10.0
Yellow Scare	10.0
Running 4 President	10.0
A Grunt's Life	10.0
Kirket	10.0
L'Enfant Terrible	10.0
The Cardinal	9.9
Puritan: All of Life to The Glory of God	9.9

```
| Home
```

```
| 9.9
```

```
+-----+-----+
only showing top 20 rows
```

```
In [38]:
```

```
total = genres_join_movie_rating.count()

less_5 = genres_join_movie_rating.filter(genres_join_movie_rating.averageRating <= 5).count()
print(f'Percentage Difference <= 5%: {less_5}, Percentage: {(less_5/total)*100}%')

greate_5_less_6 = genres_join_movie_rating.filter(genres_join_movie_rating.averageRating > 5).filter(genres_join_movie_rating.averageRating <= 6)
print(f'Percentage Difference >5 and <= 6%: {greate_5_less_6}, Percentage: {(greate_5_less_6/total)*100}%')

greate_6_less_7 = genres_join_movie_rating.filter(genres_join_movie_rating.averageRating > 6).filter(genres_join_movie_rating.averageRating <= 7)
print(f'Percentage Difference >6 and <= 7%: {greate_6_less_7}, Percentage: {(greate_6_less_7/total)*100}%')

greate_7_less_8 = genres_join_movie_rating.filter(genres_join_movie_rating.averageRating > 7).filter(genres_join_movie_rating.averageRating <= 8)
print(f'Percentage Difference >7 and <= 8%: {greate_7_less_8}, Percentage: {(greate_7_less_8/total)*100}%')

greate_8_less_9 = genres_join_movie_rating.filter(genres_join_movie_rating.averageRating > 8).filter(genres_join_movie_rating.averageRating <= 9)
print(f'Percentage Difference >8 and <= 9%: {greate_8_less_9}, Percentage: {(greate_8_less_9/total)*100}%')

greate_9_less_11 = genres_join_movie_rating.filter(genres_join_movie_rating.averageRating > 9).filter(genres_join_movie_rating.averageRating <= 11)
print(f'Percentage Difference >9 and <= 11%: {greate_9_less_11}, Percentage: {(greate_9_less_11/total)*100}%')
```

```
Percentage Difference <= 5%: 74800, Percentage: 18.542665483373575%
Percentage Difference >5 and <= 6%: 94460, Percentage: 23.416312587693422%
Percentage Difference >6 and <= 7%: 135235, Percentage: 33.52429634550836%
Percentage Difference >7 and <= 8%: 77258, Percentage: 19.151995319712242%
Percentage Difference >8 and <= 9%: 19814, Percentage: 4.911823180315027%
Percentage Difference >9 and <= 11%: 1827, Percentage: 0.45290708339737323%
```

```
In [39]:
```

```
df_averageRating_Percentage = spark.createDataFrame(
    [
        ('[0,5]', (less_5/total)*100),
        ('(5,6]', (greate_5_less_6/total)*100),
        ('(6,7]', (greate_6_less_7/total)*100),
        ('(7,8]', (greate_7_less_8/total)*100),
        ('(8,9]', (greate_8_less_9/total)*100),
        ('(9,11)', (greate_9_less_11/total)*100),
    ],
    ['AverageRating_Percentage_Range', 'Percentage']
)
```

```
df_averageRating_Percentage.show()
```

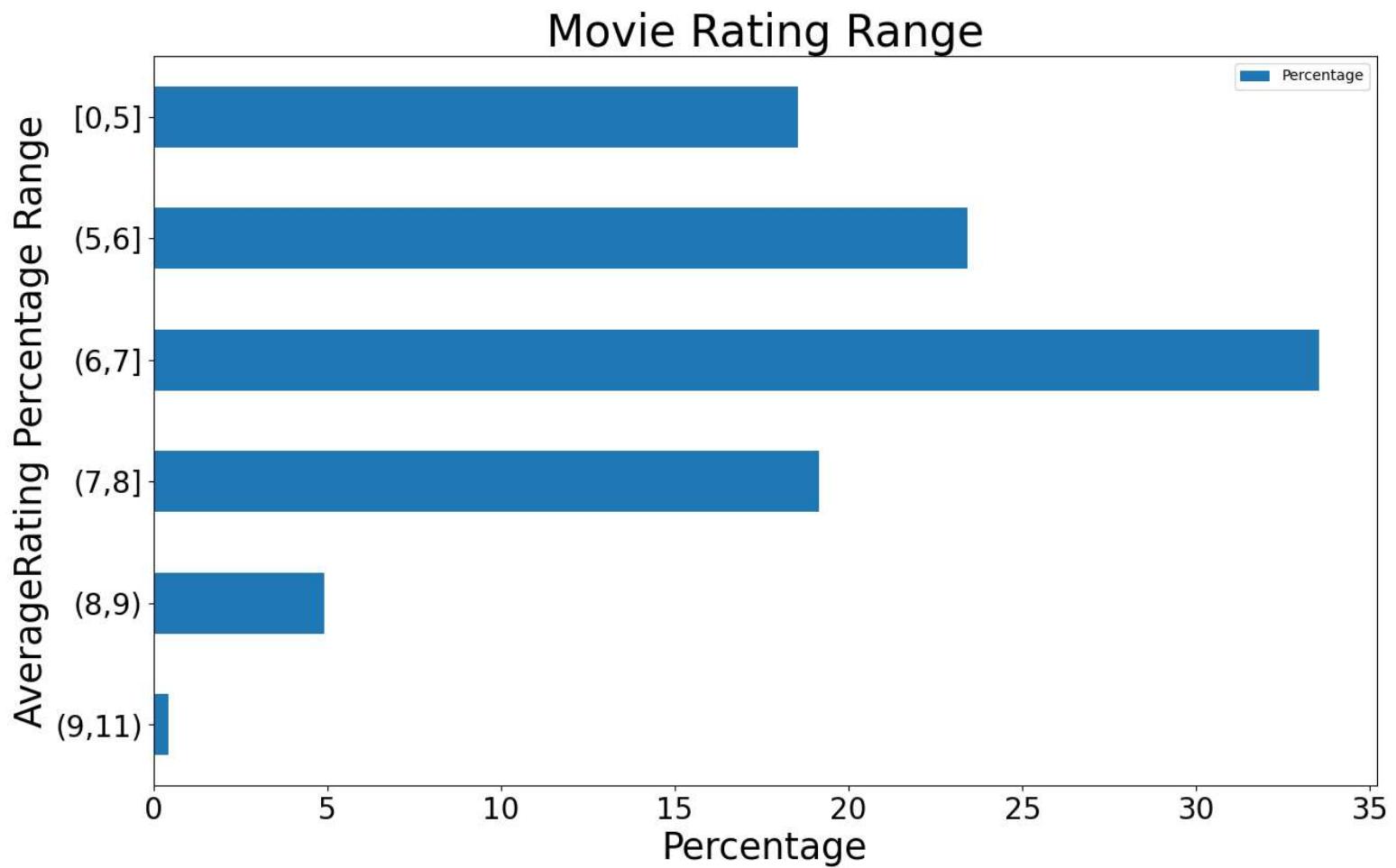
AverageRating_Percentage_Range	Percentage
[0,5]	18.542665483373575
(5,6]	23.416312587693422
(6,7]	33.52429634550836
(7,8]	19.151995319712242
(8,9]	4.911823180315027
(9,11)	0.45290708339737323

In [40]:

```
df_averageRating_Percentage_pandas = df_averageRating_Percentage.toPandas()
df_averageRating_Percentage_pandas = df_averageRating_Percentage_pandas.set_index('AverageRating_Percentage_Range')
```

In [41]:

```
df_averageRating_Percentage_pandas.plot.barh().invert_yaxis()
plt.title('Movie Rating Range', fontsize=30)
plt.xlabel('Percentage', fontsize=25)
plt.ylabel('AverageRating Percentage Range', fontsize=25)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
%matplotlib plt
```



Extra Credit - 2

```
In [42]: actors_movie = (actors.join(movie_actors, on=['nconst'], how='inner')).join(genres, on=['tconst'], how='inner') \
    .withColumn("startYear", col("startYear").cast("int"))
```

```
In [43]: actors_movie_1 = actors_movie.select('primaryTitle', 'startYear') \
    .filter(actors_movie.startYear > 2010) \
```

```
.filter(actors_movie.startYear < 2020) \
.distinct()

actors_movie_1.show(truncate=False)
```

primaryTitle	startYear
Stefan Arngrim - On Giants and Feeding Darkness	2019
Off the Rez	2011
Muhammad Ali's Greatest Fight	2013
The Bloody Rise of Axis Video	2014
Born River Bye	2017
P.A.I.N.	2018
O Mein Gott	2018
When I Walk	2013
Vampire	2011
Finding Me: Truth	2011
In April the Following Year, There Was a Fire	2012
Express Raja	2016
The Unknown Mountaineer	2018
The Mirror of the Muse: My Pretty Doll	2012
Daily Lives of High School Boys	2013
Broken Eggs Film	2013
La lune est tombée	2015
The Traveler: The Missing Link	2014
Africa Rising	2011
Flash	2018

only showing top 20 rows

```
In [44]: actors_movie_2=actors_movie_1.groupBy(['startYear']) \
                    .agg(mean('primaryTitle').alias('count_primaryTitle'))
actors_movie_2.show()
```

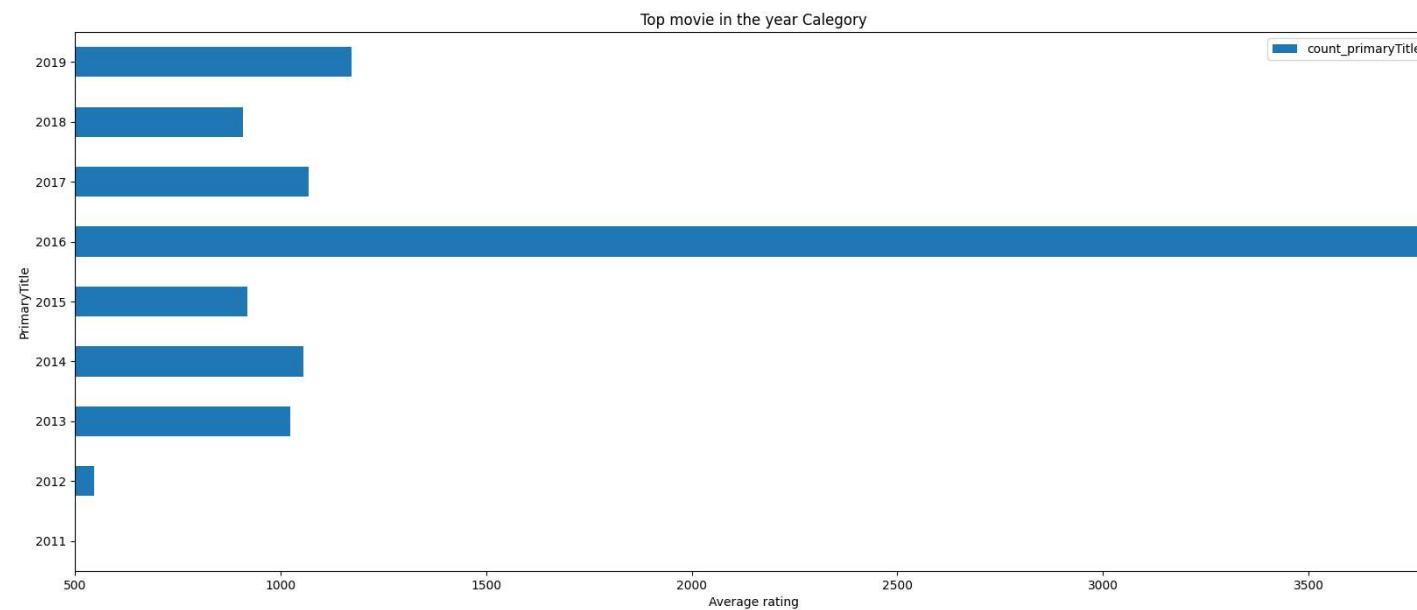
startYear	count_primaryTitle
2018	907.859999999999
2011	Infinity
2017	1069.45
2013	1023.0240526315789
2015	919.5238095238095
2014	1055.0396428571428

```
| 2012 | 547.4615384615385 |
| 2016 | 3793.777272727273 |
| 2019 | 1171.9766666666667 |
+-----+
```

```
In [46]: actors_movie1_all = actors_movie_2.sort("startYear", ascending=False).toPandas().set_index('startYear')

# plot
actors_movie1_all.plot.barh().invert_yaxis()
plt.title('Top movie in the year Category')
plt.xlabel("Average rating")
plt.ylabel("PrimaryTitle")
plt.xlim(500, 3800)
plt.rcParams["figure.figsize"] = [20, 8]

%matplotlib plt
```



```
In [ ]:
```