MVP Engenharia de Dados (40530010057_20240_01)

NAME: Andre Nascimento

DATE: Jul/2024

CONTEXT / PROBLEM:

Recently I started looking for high ranked movies that are not usually recommended by streaming platforms. They are mostly older movies that I cherry-pick because their rating is good, and they fit important criteria to me (such as the genre of the picture).

For this, I always use IMDb as a reference. Usually, I find that the data base has a good score classification because it relies on a reasonable number of users reviews and thus, accounts for public consensus. Reading a database is not that simple: you have to possess some understanding of user behavior and where to find the right data inputs (such as rating and users' qualitative reviews). User behavior is particularly important. For example, if you take a comedy movie and it hits somewhere between 7 and 8, it's generally a good movie. Dramas tend to score higher notes, with the especially good ones reaching scores higher than 8.

It's useful to notice that these things change over time. Because users' preferences change and movie quality varies as genres gain or lose prominence. There's a public feeling that movies now have to be shorter, more action driven, and adequate themselves to public. Ratings also change over time.

I would like to that a look into that, and construct a film list which I can rely upon instead of using a streaming platform recommendation algorithm to give me inputs.

OBJECTIVE:

To tackle all of this, the present project will look at public movie data bases provided by IMDb in order to if movies' length and scores changed over time. I will address the following questions:

- What is the 90% percentile for the selected movie genres since 1985, considering intervals of 15 years?
- Did the runtime and top scores of these movies changed over time?
- If I were to select movies above the selected percentiles, what movies should I pick?

I will segment the data base in chunks of time, and only look at non-adult movies belonging to the genres 'Action', 'Drama', 'Comedy', and 'Thriller'. I will use Data Brick for the ETL and for the analysis processes, as suggested in the task instruction.

STEP-BY-STEP GUIDELINE:

1) DATA SELLECTION

The data was collected from IMDB (https://developer.imdb.com/non-commercial-datasets/). Since the variables were already well documented there, I was able to identify the data sets that would help me to address the project objective.

We sure need a Primary Key that can help us to join datasets. We can identify this key as what IMDB calls “tconst”, an alphanumeric string that serves as an unique identifier for movies. To answers the questions we listed, we would need: the (1) movie title, (2) its genre, (3) its release year, (4) its category, (5) its runtime, and (6) its rating. It would also be useful to know the (7) number of reviews used to attribute the movie's rating, so we can discard movies with little reviews, since their final rating is very prone to suffer from the effect of outliers and reviewer selection bias (for example, a movie that is well liked by a small niche of people, and received a very generous score that does not is coherent with its quality).

Scanning through the metadata, we can map the variables in the following database:

- title.basics.tsv.gz: (1) movie title, (2) its genre, (3) its release year, (4) its category, (5) its runtime
- title.ratings.tsv.gz: (6) its rating, (7) number of reviews used to attribute the movie's rating

The relationship between the data sets will be better explored bellow.

2) DATA EXTRACTION

I decided to gather the data directly from the source while loading it to Data Bricks. For this we will use some pyhton modules (pandas and io from BytesIO to read the .tsv file provided by IMDB, and requests to access it through an internet link). I think this is better than downloading the data and inputting it manually, because, if IMDB updates the data base, the code can account for that and serve as a pipeline if you run it periodically with a scheduled job.

The code will be available in the step 4 of this section.

3) DATA MODELING

Since I already know the metadata by reading the IMDB website, I could plan ahead. I used the following information:

- Title_basics variables as provided by IMDB:
  - '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. \\N 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'

- Ratings variables as provided by IMDB:
  - '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'

I could take this and build the dictionaries by using SQL to modify the schema tables in Data Bricks. Note that this process occurred after the data was load (which is the step 4 of this step-by-step guide). I added some information about the range (min and max) for each variable and also explained the categories for categorical data.

The coded used is presented below:

Just now (49s) 1

```sql
%sql
-- Creating dictionaries and identifing categories for categorical fields and range of values for numeric fields, when applicable.

-- Adding comments to variables on titles_basics SCHEMA:
ALTER TABLE titles_basics ALTER COLUMN tconst COMMENT 'Unique identifier for each movie';
ALTER TABLE titles_basics ALTER COLUMN titleType COMMENT 'Type/format of the title (e.g. movie, tvmovie, short, tvseries, tvepisode, video, etc). Only movie and tvMovie are used in the analysis';
ALTER TABLE titles_basics ALTER COLUMN primaryTitle COMMENT 'The title name used by movie / tvMovie';
ALTER TABLE titles_basics ALTER COLUMN originalTitle COMMENT 'Original title, in the original language';
ALTER TABLE titles_basics ALTER COLUMN isAdult COMMENT '0: non-adult title; 1: adult title';
ALTER TABLE titles_basics ALTER COLUMN startYear COMMENT 'Represents the release year of a title. In the case of TV Series, it is the series start year. The year has 4 digits and will range only from 1985 to 2024 in the final analysis.';
ALTER TABLE titles_basics ALTER COLUMN endYear COMMENT 'TV Series end year. Null for all other title types.';
ALTER TABLE titles_basics ALTER COLUMN runtimeMinutes COMMENT 'Primary runtime of the title, in minutes, Can range from 0 to three digits units.';
ALTER TABLE titles_basics ALTER COLUMN genres COMMENT 'Includes up to three genres associated with the title.'
```

(32) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame

OK

SQL cell result stored as PySpark data frame `_sqldf`. Learn more

Just now (<1s) 2 SQL

```sql
%sql

DESCRIBE titles_basics;
```

_sqldf: pyspark.sql.dataframe.DataFrame = [col_name: string, data_type: string ... 1 more field]

Table +

	col_name	data_type	comment
1	tconst	string	Unique identifier for each movie
2	titleType	string	Type/format of the title (e.g. movie, tvmovie, short, tvseries, tvepisode, video, etc). Only movie and tvMovie are used in the analysis
3	primaryTitle	string	The title name used by movie / tvMovie
4	originalTitle	string	Original title, in the original language
5	isAdult	bigint	0: non-adult title; 1: adult title
6	startYear	bigint	› Represents the release year of a title. In the case of TV Series, it is the series start year. The year has 4 digits and will range only from 1...
7	endYear	bigint	TV Series end year. Null for all other title types.
8	runtimeMinutes	bigint	Primary runtime of the title, in minutes, Can range from 0 to three digits units.
9	genres	string	Includes up to three genres associated with the title.

9 rows | 0.39 seconds runtime Refreshed now

SQL cell result stored as PySpark data frame `_sqldf`. Learn more

Just now (17s) 3

```sql
%sql

-- Add comments to variables on ratings SCHEMA:
ALTER TABLE ratings ALTER COLUMN tconst COMMENT 'Unique identifier of the title';
ALTER TABLE ratings ALTER COLUMN averageRating COMMENT 'Weighted average of all the individual user ratings. Can range from 0.0 to 10.0';
ALTER TABLE ratings ALTER COLUMN numVotes COMMENT 'Number of votes the title has received. Can range from 0 to infinite positive numbers'
```

(17) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame

OK

SQL cell result stored as PySpark data frame `_sqldf`. Learn more

Just now (<1s) 4 SQL

```sql
%sql

DESCRIBE ratings;
```

_sqldf: pyspark.sql.dataframe.DataFrame = [col_name: string, data_type: string ... 1 more field]

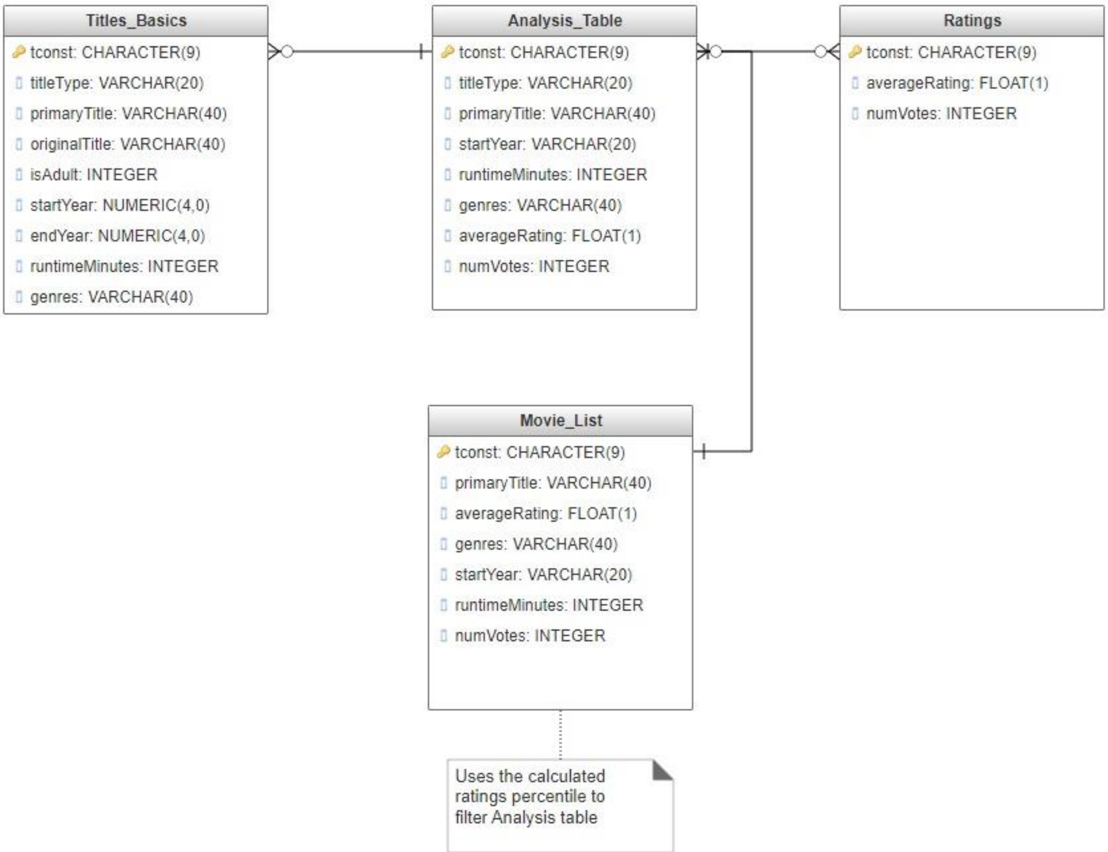
Table +

	col_name	data_type	comment
1	tconst	string	Unique identifier of the title
2	averageRating	double	Weighted average of all the individual user ratings. Can range from 0.0 to 10.0
3	numVotes	bigint	Number of votes the title has received. Can range from 0 to infinite positive numb...

3 rows | 0.40 seconds runtime Refreshed now

SQL cell result stored as PySpark data frame `_sqldf`. Learn more

To address the questions we listed in the project's objective, I will build a final data model that looks like that:



4) DATA LOADING

I started by importing the modules we need to capture the data from IMDB. We used pandas, requests and BytesIO.

After that, we provided the link for the title_basics table, ran the code and displayed the data set to see if the reading operation was running fine. We had to import all the data as strings, otherwise the execution would take too long, as the plataform would try to interpret the data by itself.

04:51 PM (6s) 1

```
from io import BytesIO
import requests
import pandas as pd
```

1 minute ago (1m) 2

```
#Creating Title_Basics Table

df_link = requests.get('https://datasets.imdbws.com/title.basics.tsv.gz')
df = pd.read_csv(BytesIO(df_link.content),compression='gzip', header=0, sep='\t',
                 dtype={'tconst': str,
                        'titleType': str,
                        'primaryTitle': str,
                        'originalTitle': str,
                        'isAdult': str,
                        'startYear': str,
                        'endYear': str,
                        'runtimeMinutes': str,
                        'genres': str})

# When importing I was running into errors and delays when the data types were no string, because of int values with '\\N' to signal null values.
# I will treat this before creating and storing the table in the workspace.

df.display()
```

(2) Spark Jobs

Table

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	genres
1	tt0000001	short	Carmencita	Carmencita	0	1894	\N	1	Documentary,Short
2	tt0000002	short	Le clown et ses chiens	Le clown et ses chiens	0	1892	\N	5	Animation,Short
3	tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	0	1892	\N	5	Animation,Comedy,
4	tt0000004	short	Un bon bock	Un bon bock	0	1892	\N	12	Animation,Short
5	tt0000005	short	Blacksmith Scene	Blacksmith Scene	0	1893	\N	1	Comedy,Short
6	tt0000006	short	Chinese Opium Den	Chinese Opium Den	0	1894	\N	1	Short
7	tt0000007	short	Corbett and Courtney Before the Kinetograph	Corbett and Courtney Before the Kinetograph	0	1894	\N	1	Short,Sport
8	tt0000008	short	Edison Kinetoscopic Record of a Sneeze	Edison Kinetoscopic Record of a Sneeze	0	1894	\N	1	Documentary,Short
9	tt0000009	movie	Miss Jerry	Miss Jerry	0	1894	\N	45	Romance
10	tt0000010	short	Leaving the Factory	La sortie de l'usine Lumière à Lyon	0	1895	\N	1	Documentary,Short
11	tt0000011	short	Akrobatisches Potpourri	Akrobatisches Potpourri	0	1895	\N	1	Documentary,Short
12	tt0000012	short	The Arrival of a Train	L'arrivée d'un train à La Ciotat	0	1896	\N	1	Documentary,Short
13	tt0000013	short	The Photographical Congress Arrives in Lyon	Le débarquement du congrès de photographie à Lyon	0	1895	\N	1	Documentary,Short
14	tt0000014	short	The Waterer Watered	L'arroseur arrosé	0	1895	\N	1	Comedy,Short

10,000+ rows | Truncated data due to row limit | 1.08 minutes runtime

Refreshed 1 minute ago

Then, I dropped null values and corrected the data types.

```
# We will select only the columns we will need. Since we will have to drop null values, we have to get rid of the 'EndYear' column, because only series and tvSeries have it.
# If we dropped null value before exclusing this column, we would lose all the other titleTypes, including the ones which are the focus of our analysis.

df = df[['tconst', 'titleType', 'primaryTitle', 'isAdult', 'startYear', 'runtimeMinutes', 'genres']]
```

```
# Dropping \\N interfering with data types, then changing data types as intented for the schema

df = df.replace('\\N', None)
df = df.dropna()
df.isAdult = df.isAdult.astype(int)
df.startYear = df.startYear.astype(int)
df.runtimeMinutes = df.runtimeMinutes.astype(int)

df.display()
```

(2) Spark Jobs

	tconst	titleType	primaryTitle	isAdult	startYear	runtimeMinutes	genres
1	tt0000001	short	Carmencita	0	1894	1	Documentary,Short
2	tt0000002	short	Le clown et ses chiens	0	1892	5	Animation,Short
3	tt0000003	short	Pauvre Pierrot	0	1892	5	Animation,Comedy,Romance
4	tt0000004	short	Un bon bock	0	1892	12	Animation,Short
5	tt0000005	short	Blacksmith Scene	0	1893	1	Comedy,Short
6	tt0000006	short	Chinese Opium Den	0	1894	1	Short
7	tt0000007	short	Corbett and Courtney Before the Kinetograph	0	1894	1	Short,Sport
8	tt0000008	short	Edison Kinetoscopic Record of a Sneeze	0	1894	1	Documentary,Short
9	tt0000009	movie	Miss Jerry	0	1894	45	Romance
10	tt0000010	short	Leaving the Factory	0	1895	1	Documentary,Short
11	tt0000011	short	Akrobatisches Potpourri	0	1895	1	Documentary,Short
12	tt0000012	short	The Arrival of a Train	0	1896	1	Documentary,Short
13	tt0000013	short	The Photographical Congress Arrives in Lyon	0	1895	1	Documentary,Short
14	tt0000014	short	The Waterer Watered	0	1895	1	Comedy,Short
15	tt0000015	short	Autour d'une cabine	0	1894	2	Animation,Short

10,000+ rows | Truncated data due to row limit | 11.11 seconds runtime

Refreshed now

Once I did that, I then created the table inside DataBricks using pySparks.

```
# Creating table

permanent_table_name = "Titles_Basics"
spark.createDataFrame(df).write.mode("overwrite").saveAsTable(permanent_table_name)
```

(10) Spark Jobs
- Job 4 View (Stages: 1/1)
- Job 5 View (Stages: 1/1, 1 skipped)
- Job 6 View (Stages: 1/1, 2 skipped)
- Job 7 View (Stages: 1/1)
- Job 8 View (Stages: 1/1)
- Job 9 View (Stages: 1/1)
- Job 10 View (Stages: 1/1)
- Job 11 View (Stages: 1/1)
- Job 12 View (Stages: 1/1, 1 skipped)
- Job 13 View (Stages: 1/1, 2 skipped)

After that, I checked the catalog to the see if the created table was available and created the data dictionary, as explained previously in the “Data Modeling” section.

default.titles_basics | Refresh

ANDRE NASCIMENTO's MVP

Details History

Description:
Created at: 2024-06-23 17:33:44
Last modified: 2024-07-01 20:07:07
Partition columns:
Number of files: 309
Size: 77.1 MB

Schema:

	col_name	data_type	comment
1	tconst	string	Unique identifier for each movie
2	titleType	string	Type/format of the title (e.g. movie, tvmovie, short, tvseries, tvepisode, video, etc). Only movie and tvMovie are used in the analysis
3	primaryTitle	string	The title name used by movie / tvMovie
4	originalTitle	string	Original title, in the original language
5	isAdult	bigint	0: non-adult title; 1: adult title
6	startYear	bigint	> Represents the release year of a title. In the case of TV Series, it is the series start year. The year has 4 digits and will range only from 1...
7	endYear	bigint	TV Series end year. Null for all other title types.
8	runtimeMinutes	bigint	Primary runtime of the title, in minutes, Can range from 0 to three digits units.
9	genres	string	Includes up to three genres associated with the title.

Sample Data:

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	genres
1	tt0017606	short	Alice the Beach Nut	null	0	1927	null	7	Animation,Comedy,Short
2	tt0017607	short	Alice the Collegiate	null	0	1927	null	7	Animation,Comedy,Short
3	tt0017608	short	Alice the Golf Bug	null	0	1927	null	7	Animation,Comedy,Short
4	tt0017609	short	Alice the Whaler	null	0	1927	null	6	Animation,Comedy,Family
5	tt0017610	short	Alice's Auto Race	null	0	1927	null	7	Animation,Comedy,Short
6	tt0017612	short	Alice's Circus Daze	null	0	1927	null	7	Animation,Comedy,Short
7	tt0017613	short	Alice's Knaughty Knight	null	0	1927	null	7	Animation,Comedy,Short
8	tt0017614	short	Alice's Medicine Show	null	0	1927	null	7	Animation,Comedy,Short
9	tt0017615	short	Alice's Picnic	null	0	1927	null	7	Animation,Comedy,Short
10	tt0017617	movie	All Aboard	null	0	1927	null	60	Comedy,Romance
11	tt0017618	short	All Wet	null	0	1927	null	7	Animation,Comedy,Short
12	tt0017619	movie	Almost Human	null	0	1927	null	60	Drama,Romance
13	tt0017621	movie	A Daughter of Destiny	null	0	1928	null	108	Drama,Fantasy,Horror
14	tt0017622	movie	Altars of Desire	null	0	1927	null	70	Drama,Romance
15	tt0017623	movie	Am Rande der Welt	null	0	1927	null	104	Drama,War

Then, I did the same thing the ratings table.

Just now (41s) 7 Python

```
# Creating Ratings Table

df_link = requests.get('https://datasets.imdbws.com/title.ratings.tsv.gz')
df = pd.read_csv(BytesIO(df_link.content),compression='gzip', header=0, sep='\t',
                 dtype={'tconst': str,
                        'averageRating': float,
                        'numVotes': int})

permanent_table_name = "Ratings"

spark.createDataFrame(df).write.mode("overwrite").saveAsTable(permanent_table_name)
```

(11) Spark Jobs

default.ratings | Refresh

ANDRE NASCIMENTO's MVP

Details History

Description:
Created at: 2024-06-18 20:38:19
Last modified: 2024-07-01 20:14:10
Partition columns:
Number of files: 8
Size: 10.9 MB

Schema:

	col_name	data_type	comment
1	tconst	string	Unique identifier of the title
2	averageRating	double	Weighted average of all the individual user ratings. Can range from 0.0 to 10.0
3	numVotes	bigint	Number of votes the title has received. Can range from 0 to infinite positive numb...

Sample Data:

	tconst	1.2 averageRating	numVotes
1	tt0000001	5.7	2059
2	tt0000002	5.6	278
3	tt0000003	6.5	2028
4	tt0000004	5.4	179
5	tt0000005	6.2	2795
6	tt0000006	5.1	189
7	tt0000007	5.4	876
8	tt0000008	5.4	2207
9	tt0000009	5.4	211
10	tt0000010	6.8	7612
11	tt0000011	5.2	389
12	tt0000012	7.4	12957
13	tt0000013	5.7	1978
14	tt0000014	7.1	5886
15	tt0000015	6.1	1187

Notice that I didn't have to import all the data as strings, since all the columns had an consistent high quality and there was no null values interfering with the data type classification (there was no null fields classified as "\\N" in this table).

So, the data loading phase finished and the analysis phase could take place.

5) DATA ANALYSIS

a. Data quality

I trackled most data quality issues during the loading phase. There were problems with **(1) null values**, **(2) low quality data** and **(3) data types**. Null values show concerns about the data sets, we dropped them to make the analysis more consistent. To dimmish low quality data records, we also abandoned movie entries with a low number of votes. Also, some data types were originally given as strings. That would have prevent us from doing operations, so we changed them to numeric to allow operations to take place.

There was also a problem related to **(4) data classification inconsistencies**, which was not treated because it did not interfere with the analysis. There are a lot of genres and some of them have multiple values such as ('Drama, Thriller' or 'Drama, History, Biography'). We addressed that during the analysis by searching if this field contained a given word. This is a solution in a sense that, if 'Drama' and 'Drama, History, Biography' both contained Drama, they will be identified as 'Drama'. But we also have be aware that, if we were to make calculations segmenting the genres instead of segmenting the data base by year, movies would be accounted in more than one category of analysis. This did not affect the analysis, since we used a view segmented by time frame.

b. Problem solving

We conducted the analysis using pySpark, so I started importing the relevant modules to read the stored tables.

```
Just now (<1s)                                  1                                  Python
# Impoting modules and loading the datasets

from pyspark.sql import SparkSession

Titles_Basics = spark.read.table("titles_basics")
Ratings = spark.read.table("Ratings")

Titles_Basics: pyspark.sql.dataframe.DataFrame = [tconst: string, titleType: string ... 7 more fields]
Ratings: pyspark.sql.dataframe.DataFrame = [tconst: string, averageRating: double ... 1 more field]
```

Then we filtered the data just to select the subset that was relevant to answer the MVP's objective questions. All the operations were commented in the code. We removed adult movies from the sample, selected movies only from 1985 onwards, selected only 'movie' and 'tvMovies' from titleTypes, and selected only the genres we intended to work with (Drama, Comedy, Action, Thriller).

Just now (7s) 2 Python

```python
# Filtering out values we will not need before merging tables.

# From titles, we will select only the columsn we will need
# From titles, we will drop movies that are Adult movies (isAdult != 0)
# From titles, we will drop movies that were released before 1985
# From titles, we will select only the titleTypes 'movie' and 'tvMovie'
# From titles, we will drop movies which do not contain Drama, Comedy, Action or Thriller in their 'genre'
# From titles, we will select only the columns we need (we do not need isAdult any longer)

Title_Basics_Updated = Titles_Basics[['tconst', 'titleType', 'primaryTitle', 'isAdult', 'startYear', 'runtimeMinutes', 'genres']]

Title_Basics_Updated = Title_Basics_Updated.filter(Titles_Basics.isAdult == 0)
Title_Basics_Updated = Title_Basics_Updated.filter(Titles_Basics.startYear >= 1985)

Title_Basics_Updated = Title_Basics_Updated.filter((Title_Basics_Updated.titleType == 'movie') | (Title_Basics_Updated.titleType == 'tvMovie'))

Title_Basics_Updated = Title_Basics_Updated.filter(
    Title_Basics_Updated.genres.contains('Drama') |
    Title_Basics_Updated.genres.contains('Comedy') |
    Title_Basics_Updated.genres.contains('Action') |
    Title_Basics_Updated.genres.contains('Thriller')
)

Title_Basics_Updated = Title_Basics_Updated.drop('isAdult')

Title_Basics_Updated.display()
```

(3) Spark Jobs

```
Title_Basics_Updated: pyspark.sql.dataframe.DataFrame
    tconst: string
    titleType: string
    primaryTitle: string
    startYear: long
    runtimeMinutes: long
    genres: string
```

Table

	tconst	titleType	primaryTitle	startYear	runtimeMinutes	genres
1	tt29419040	movie	Seima	2023	92	Comedy
2	tt29419533	movie	A Way Out 3	2024	91	Comedy
3	tt29419593	movie	All the Silence	2023	82	Drama
4	tt29419926	movie	Vessel	2023	126	Thriller
5	tt29420033	movie	Le Retour Tout Court	2007	117	Comedy

So we named this temporary table as Title_Basics_Updated. We then, manipulated the Ratings table by excluding movies with less than 2000 number of votes and named it Ratings_Updated.

Just now (2s) 3

```python
# From Ratings, we will drop movies with less than 2.000 votes

Ratings_Updated = Ratings.filter(Ratings.numVotes >= 2000)
Ratings_Updated.display()
```

(3) Spark Jobs

Ratings_Updated: pyspark.sql.dataframe.DataFrame = [tconst: string, averageRating: double ... 1 more field]

Table

	tconst	1.2 averageRating	numVotes
1	tt0000001	5.7	2059
2	tt0000003	6.5	2028
3	tt0000005	6.2	2795
4	tt0000008	5.4	2207
5	tt0000010	6.8	7612
6	tt0000012	7.4	12957
7	tt0000014	7.1	5886
8	tt0000029	5.9	3537
9	tt0000041	6.7	2012
10	tt0000070	6.4	2785
11	tt0000075	6.3	2070
12	tt0000091	6.7	4027
13	tt0000211	7.4	4786
14	tt0000359	7.1	3205
15	tt0000417	8.1	56233

10,000+ rows | Truncated data due to row limit | 1.90 seconds runtime

Refreshed now

And finally, we joined the tables, was we planned in the data model.

Just now (39s) 4 Python

```python
# Joining datables on a analysis table

Analysis_table = Title_Basics_Updated.join(Ratings_Updated, Title_Basics_Updated.tconst == Ratings_Updated.tconst, how='inner')

Analysis_table.display()
```

(5) Spark Jobs

Analysis_table: pyspark.sql.dataframe.DataFrame = [tconst: string, titleType: string ... 7 more fields]

Table

	tconst	titleType	primaryTitle	startYear	runtimeMinutes	genres	tconst	averageRating	numVotes
1	tt2980210	movie	A Hologram for the King	2016	98	Comedy,Drama,Romance	tt2980210	6.1	47655
2	tt13957560	movie	Dumb Money	2023	105	Biography,Comedy,Drama	tt13957560	6.9	45619
3	tt7026672	movie	Every Day	2018	97	Drama,Fantasy,Romance	tt7026672	6.4	23773
4	tt1582271	movie	The Good Doctor	2011	93	Drama,Mystery,Thriller	tt1582271	5.5	7303
5	tt0349113	movie	Ax 'Em	1992	71	Comedy,Horror,Mystery	tt0349113	1.3	2115
6	tt0353496	movie	Godfather	1991	150	Comedy,Drama,Romance	tt0353496	8.6	4263
7	tt0363988	movie	Secret Window	2004	96	Drama,Mystery,Thriller	tt0363988	6.5	210456
8	tt1323973	movie	The Trials of Cate McCall	2013	89	Crime,Drama,Mystery	tt1323973	6.2	7140
9	tt0311926	tvMovie	Screwed in Tallinn	1999	59	Comedy,Drama	tt0311926	7.8	5679
10	tt3203890	movie	Pulling Strings	2013	111	Comedy,Romance	tt3203890	6.1	2010
11	tt16085592	movie	Tall Girl 2	2022	97	Comedy,Drama,Family	tt16085592	4.7	6377
12	tt1611004	movie	Autograph	2010	131	Drama	tt1611004	7.4	2532
13	tt11143108	movie	Oh My Kadavule	2020	151	Comedy,Fantasy,Romance	tt11143108	8.1	4918
14	tt0327698	tvMovie	The Even Stevens Movie	2003	93	Comedy,Family	tt0327698	6.2	7530
15	tt0338706	movie	Alexandra's Project	2003	103	Drama,Mystery,Thriller	tt0338706	6.5	5231

10,000+ rows | Truncated data due to row limit | 38.69 seconds runtime — Refreshed now

Then, I created three subsets of this table. One with all the movies from 1985 to 2000 (this is until the end of 1999); a second one with all the movies ranging from 2000 to 2015 (until the end of 2014); and a third one with movies created in 2015 or more recent years. I checked if the code worked fine before proceeding.

1 minute ago (<1s) 5

```python
Analysis_table_1985 = Analysis_table.filter(Analysis_table.startYear < 2000)
Analysis_table_2000 = Analysis_table.filter((Analysis_table.startYear >= 2000) & (Analysis_table.startYear < 2015))
Analysis_table_2015 = Analysis_table.filter(Analysis_table.startYear >= 2015)
```

Analysis_table_1985: pyspark.sql.dataframe.DataFrame = [tconst: string, titleType: string ... 7 more fields]
Analysis_table_2000: pyspark.sql.dataframe.DataFrame = [tconst: string, titleType: string ... 7 more fields]
Analysis_table_2015: pyspark.sql.dataframe.DataFrame = [tconst: string, titleType: string ... 7 more fields]

Just now (38s) 6 Python

```python
Analysis_table_1985.display()
```

(6) Spark Jobs

Table

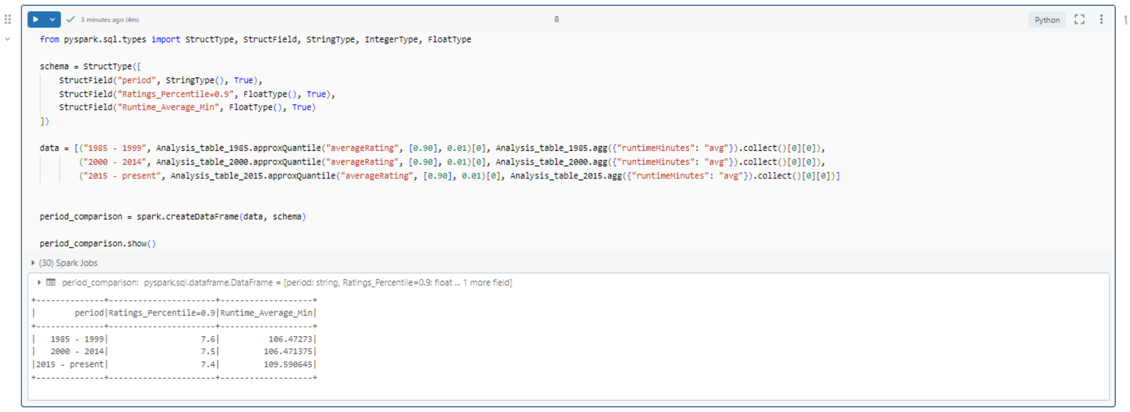
	tconst	titleType	primaryTitle	startYear	runtimeMinutes	genres	tconst	averageRating	numVotes
1	tt0349113	movie	Ax 'Em	1992	71	Comedy,Horror,Mystery	tt0349113	1.3	2115
2	tt0353496	movie	Godfather	1991	150	Comedy,Drama,Romance	tt0353496	8.6	4263
3	tt0311926	tvMovie	Screwed in Tallinn	1999	59	Comedy,Drama	tt0311926	7.8	5679
4	tt0359715	movie	My Best Friend's Birthday	1987	69	Comedy	tt0359715	5.6	4811
5	tt0313573	movie	Selamsiz's Band	1987	107	Comedy,Drama,Music	tt0313573	7.7	3976
6	tt1619856	tvMovie	Mulholland Dr.	1999	88	Drama,Mystery,Thriller	tt1619856	8.2	9970
7	tt0353975	movie	Sandesham	1991	138	Comedy,Drama	tt0353975	9	5290
8	tt0354148	movie	Vadakkunokkiyantram	1989	120	Comedy,Drama	tt0354148	8.3	2216
9	tt0316078	movie	Kadhalukku Mariyadhai	1997	165	Drama,Romance	tt0316078	8.2	5079
10	tt3189256	movie	Om	1995	150	Action,Crime,Drama	tt3189256	8.9	3954
11	tt0353935	movie	Ramji Rao Speaking	1989	150	Comedy,Drama,Thriller	tt0353935	8.5	3592
12	tt0337611	movie	Dilwale	1994	172	Action,Drama,Romance	tt0337611	5.6	3195
13	tt3178292	movie	A: Film by Upendra	1998	150	Drama	tt3178292	8.7	2363
14	tt1517561	movie	Thulladha Manamum Thullum	1999	170	Drama,Romance	tt1517561	8.3	6568
15	tt0321715	tvMovie	La classe américaine	1993	70	Comedy	tt0321715	7.9	2767

4,364 rows | 38.01 seconds runtime — Refreshed now

Now, I could answer the two first questions of the objective:

- What is the 90% percentile for the selected movie genres since 1985, considering intervals of 15 years?
- Did the runtime and top scores of these movies changed over time?

To do that we had to import new modules to create a schema so that we could present the percentile calculations and the runtime aggregation within a table format.



```
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, FloatType

schema = StructType([
    StructField("period", StringType(), True),
    StructField("Ratings_Percentile=0.9", FloatType(), True),
    StructField("Runtime_Average_Min", FloatType(), True)
])

data = [("1985 - 1999", Analysis_table_1985.approxQuantile("averageRating", [0.90], 0.01)[0], Analysis_table_1985.agg({"runtimeMinutes": "avg"}).collect()[0][0]),
        ("2000 - 2014", Analysis_table_2000.approxQuantile("averageRating", [0.90], 0.01)[0], Analysis_table_2000.agg({"runtimeMinutes": "avg"}).collect()[0][0]),
        ("2015 - present", Analysis_table_2015.approxQuantile("averageRating", [0.90], 0.01)[0], Analysis_table_2015.agg({"runtimeMinutes": "avg"}).collect()[0][0])]


period_comparison = spark.createDataFrame(data, schema)

period_comparison.show()
```

```
(30) Spark Jobs
period_comparison: pyspark.sql.dataframe.DataFrame = [period: string, Ratings_Percentile=0.9: float ... 1 more field]
+--------------+----------------------+-------------------+
|        period|Ratings_Percentile=0.9|Runtime_Average_Min|
+--------------+----------------------+-------------------+
|   1985 - 1999|                   7.6|          106.47273|
|   2000 - 2014|                   7.5|         106.471375|
|2015 - present|                   7.4|         109.590645|
+--------------+----------------------+-------------------+
```

We can see the percentile 90% for movie ratings is steadily decreasing throughout the periods analyzed, but the runtime isn’t. So, in a high level, we might infer the quality of movies is dropping and that’s not attribute to shorter lengths, since there is not a noticeable decrease in runtimes in recent periods.

The last task was to respond the third question listed in the MVP's objective:

- If I were to select movies above the selected percentiles, what movies should I pick?

To do that, we need to filter the subsets of the table by the calculated percentiles and then stack them on the top of each other. We ran the code in the 1985 – 2000 subset, checked if it was fine and replicated the same process for the other two periods.

2 minutes ago (1m) 9 Python

```
Movie_list_1985_2000 = Analysis_table_1985.filter(Analysis_table_1985.averageRating >= Analysis_table_1985.approxQuantile("averageRating", [0.90], 0.01)[0])
Movie_list_1985_2000 = Movie_list_1985_2000.orderBy(Movie_list_1985_2000['numVotes'].desc())
Movie_list_1985_2000.display()
```

(8) Spark Jobs

Movie_list_1985_2000: pyspark.sql.dataframe.DataFrame = [tconst: string, titleType: string ... 7 more fields]

Table

	tconst	titleType	primaryTitle	startYear	runtimeMinutes	genres	tconst	averageRating	numVotes
1	tt0111161	movie	The Shawshank Redemption	1994	142	Drama	tt0111161	9.3	2910355
2	tt0137523	movie	Fight Club	1999	139	Drama	tt0137523	8.8	2343638
3	tt0109830	movie	Forrest Gump	1994	142	Drama,Romance	tt0109830	8.8	2275504
4	tt0110912	movie	Pulp Fiction	1994	154	Crime,Drama	tt0110912	8.9	2237125
5	tt0133093	movie	The Matrix	1999	136	Action,Sci-Fi	tt0133093	8.7	2068377
6	tt0114369	movie	Se7en	1995	127	Crime,Drama,Mystery	tt0114369	8.6	1812931
7	tt0102926	movie	The Silence of the Lambs	1991	118	Crime,Drama,Thriller	tt0102926	8.6	1559887
8	tt0120815	movie	Saving Private Ryan	1998	169	Drama,War	tt0120815	8.6	1507452
9	tt0108052	movie	Schindler's List	1993	195	Biography,Drama,History	tt0108052	9	1461188
10	tt0120689	movie	The Green Mile	1999	189	Crime,Drama,Fantasy	tt0120689	8.6	1418190
11	tt0088763	movie	Back to the Future	1985	116	Adventure,Comedy,Sci-Fi	tt0088763	8.5	1316665
12	tt0120338	movie	Titanic	1997	194	Drama,Romance	tt0120338	7.9	1290173
13	tt0099685	movie	Goodfellas	1990	145	Biography,Crime,Drama	tt0099685	8.7	1266631
14	tt0110413	movie	Léon: The Professional	1994	110	Action,Crime,Drama	tt0110413	8.5	1254014
15	tt0169547	movie	American Beauty	1999	122	Drama	tt0169547	8.3	1216341

525 rows | 1.08 minutes runtime Refreshed 2 minutes ago

Just now (1m) 10 Python

```
Movie_list_2000_2015 = Analysis_table_2000.filter(Analysis_table_2000.averageRating >= Analysis_table_2000.approxQuantile("averageRating", [0.90], 0.01)[0])
Movie_list_2000_2015 = Movie_list_2000_2015.orderBy(Movie_list_2000_2015['numVotes'].desc())
Movie_list_2000_2015.display()
```

(8) Spark Jobs

Movie_list_2000_2015: pyspark.sql.dataframe.DataFrame = [tconst: string, titleType: string ... 7 more fields]

Table

	tconst	titleType	primaryTitle	startYear	runtimeMinutes	genres	tconst	averageRating	numVotes
1	tt0468569	movie	The Dark Knight	2008	152	Action,Crime,Drama	tt0468569	9	2891568
2	tt1375666	movie	Inception	2010	148	Action,Adventure,Sci-Fi	tt1375666	8.8	2569070
3	tt0816692	movie	Interstellar	2014	169	Adventure,Drama,Sci-Fi	tt0816692	8.7	2124710
4	tt0120737	movie	The Lord of the Rings: The Fellowship of the Ring	2001	178	Action,Adventure,Drama	tt0120737	8.9	2021450
5	tt0167260	movie	The Lord of the Rings: The Return of the King	2003	201	Action,Adventure,Drama	tt0167260	9	1992825
6	tt1345836	movie	The Dark Knight Rises	2012	164	Action,Drama,Thriller	tt1345836	8.4	1839531
7	tt0167261	movie	The Lord of the Rings: The Two Towers	2002	179	Action,Adventure,Drama	tt0167261	8.8	1796635
8	tt1853728	movie	Django Unchained	2012	165	Comedy,Drama,Western	tt1853728	8.5	1709051
9	tt0172495	movie	Gladiator	2000	155	Action,Adventure,Drama	tt0172495	8.5	1632759
10	tt0993846	movie	The Wolf of Wall Street	2013	180	Biography,Comedy,Crime	tt0993846	8.2	1598206
11	tt0361748	movie	Inglourious Basterds	2009	153	Adventure,Drama,War	tt0361748	8.4	1597709
12	tt0372784	movie	Batman Begins	2005	140	Action,Crime,Drama	tt0372784	8.2	1588812
13	tt1130884	movie	Shutter Island	2010	138	Drama,Mystery,Thriller	tt1130884	8.2	1469967
14	tt0848228	movie	The Avengers	2012	143	Action,Sci-Fi	tt0848228	8	1466099
15	tt0482571	movie	The Prestige	2006	130	Drama,Mystery,Sci-Fi	tt0482571	8.5	1451091

1,104 rows | 1.06 minutes runtime Refreshed now

1 minute ago (1m) 11 Python

```
Movie_list_2015_present = Analysis_table_2015.filter(Analysis_table_2015.averageRating >= Analysis_table_2015.approxQuantile("averageRating", [0.90], 0.01)[0])
Movie_list_2015_present = Movie_list_2015_present.orderBy(Movie_list_2015_present['numVotes'].desc())
Movie_list_2015_present.display()
```

(8) Spark Jobs

Movie_list_2015_present: pyspark.sql.dataframe.DataFrame = [tconst: string, titleType: string ... 7 more fields]

Table

	tconst	titleType	primaryTitle	startYear	runtimeMinutes	genres	tconst	averageRating	numVotes
1	tt7286456	movie	Joker	2019	122	Crime,Drama,Thriller	tt7286456	8.4	1506753
2	tt4154796	movie	Avengers: Endgame	2019	181	Action,Adventure,Drama	tt4154796	8.4	1278574
3	tt4154756	movie	Avengers: Infinity War	2018	149	Action,Adventure,Sci-Fi	tt4154756	8.4	1215305
4	tt1431045	movie	Deadpool	2016	108	Action,Comedy	tt1431045	8	1135483
5	tt1392190	movie	Mad Max: Fury Road	2015	120	Action,Adventure,Sci-Fi	tt1392190	8.1	1112467
6	tt2488496	movie	Star Wars: Episode VII - The Force Awakens	2015	138	Action,Adventure,Sci-Fi	tt2488496	7.8	978447
7	tt6751668	movie	Parasite	2019	132	Drama,Thriller	tt6751668	8.5	973257
8	tt3659388	movie	The Martian	2015	144	Adventure,Drama,Sci-Fi	tt3659388	8	932273
9	tt10872600	movie	Spider-Man: No Way Home	2021	148	Action,Adventure,Fantasy	tt10872600	8.2	890085
10	tt1160419	movie	Dune	2021	155	Action,Adventure,Drama	tt1160419	8	882837
11	tt1663202	movie	The Revenant	2015	156	Action,Adventure,Drama	tt1663202	8	880529
12	tt7131622	movie	Once Upon a Time... in Hollywood	2019	161	Comedy,Drama	tt7131622	7.6	857988
13	tt3498820	movie	Captain America: Civil War	2016	147	Action,Sci-Fi	tt3498820	7.8	855820
14	tt3315342	movie	Logan	2017	137	Action,Drama,Sci-Fi	tt3315342	8.1	838917
15	tt3501632	movie	Thor: Ragnarok	2017	130	Action,Adventure,Comedy	tt3501632	7.9	821871

1,039 rows | 1.06 minutes runtime Refreshed now

Finally, we built the final recommendation movie table, stacking the subsets.

1 minute ago (1m) 12 Python

```python
Movie_list = Movie_list_1985_2000.union(Movie_list_2000_2015)
Movie_list = Movie_list.union(Movie_list_2015_present)
Movie_list = Movie_list.orderBy(Movie_list['numVotes'].desc())
Movie_list = Movie_list.select('primaryTitle', 'averageRating', 'genres', 'startYear', 'runtimeMinutes', 'numVotes')
Movie_list.display()
```

(10) Spark Jobs

Movie_list: pyspark.sql.dataframe.DataFrame = [primaryTitle: string, averageRating: double ... 4 more fields]

Table

	primaryTitle	1.2 averageRating	genres	startYear	runtimeMinutes	numVotes
1	The Shawshank Redemption	9.3	Drama	1994	142	2910355
2	The Dark Knight	9	Action,Crime,Drama	2008	152	2891568
3	Inception	8.8	Action,Adventure,Sci-Fi	2010	148	2569070
4	Fight Club	8.8	Drama	1999	139	2343638
5	Forrest Gump	8.8	Drama,Romance	1994	142	2275504
6	Pulp Fiction	8.9	Crime,Drama	1994	154	2237125
7	Interstellar	8.7	Adventure,Drama,Sci-Fi	2014	169	2124710
8	The Matrix	8.7	Action,Sci-Fi	1999	136	2068377
9	The Lord of the Rings: The Fellowship of the Ring	8.9	Action,Adventure,Drama	2001	178	2021450
10	The Lord of the Rings: The Return of the King	9	Action,Adventure,Drama	2003	201	1992825
11	The Dark Knight Rises	8.4	Action,Drama,Thriller	2012	164	1839531
12	Se7en	8.6	Crime,Drama,Mystery	1995	127	1812931
13	The Lord of the Rings: The Two Towers	8.8	Action,Adventure,Drama	2002	179	1796635
14	Django Unchained	8.5	Comedy,Drama,Western	2012	165	1709051
15	Gladiator	8.5	Action,Adventure,Drama	2000	155	1632759

2,668 rows | 1.20 minutes runtime

Refreshed now

Notice that we ordered the result by Number of Votes, assuming that more popular movies are a better pick, even if their score is lower than other movies with lower votes. We also selected the order in which the columns would be seen and chose to not present the primary key in the final table, because this variable is not relevant for an external user trying to consume its content.

SELF-EVALUATION

The solution worked and it was useful for me already. The conclusions brought insights I was not expecting (about movie runtimes) and I identified some high-quality movies I intend to watch. Overall, despite its simplicity, I think it is a better source to select movies than the recommendation algorithm of streaming platforms for viewers that resemble me.

From a technical point of view, I had to learned a lot to deliver the MVP. Loading the data was the hardest part, because I wanted to structure this in a way that guaranteed that if the source of the data changes, the loading would account for that.

As I said before, the analysis was simple. It could have been deeper if I had more time to work on it. Also, some things could have been better explored, such as: (1) programming languages, (2) data quality treatments, and (3) cloud platforms.

Concerning the first issue, I could have used other languages, but I preferred pySparks, because I think I generates a step-by-step view that is easier to read than SQL.

Secondly, I selected a data base that was well documented, and this saved me from some real-world recurring issues, such as lack of knowledge about the data. But dealing with these issues was not the focus of this endeavor. My main concern was to make a solution I would use and that would be also useful to other people, so it was mandatory that I selected well-documented data.

Lastly, I could have used some cloud solutions to build the pipelines, such as AWS, or Google Cloud, but I chose to proceed the way I did to avoid potential financial budget constraints.

Overall, this endeavor was relevant and solidified some building blocks I needed to become better at engineering and analyzing data. As this course goes on, in the future, it would be nice to revisit this project and add a Business Intelligence interface to the movie recommendation list.