Masket Basket Analysis: find frequent itemsets

Introduction

The following paper has the aim to explain how to implement a system finding frequent itemset, managing a large-scale dataset with a technique that scales up with the data size. In order to make clear theoretical concepts behind it, the paper will show an experiment carried out on the IMDB dataset published on Kaggle.

The report starts explaining what a largescale dataset is and useful techniques to manage it.

It continues presenting the dataset used to carry on the experiment, useful information about the variables and how it can be downloaded directly from Kaggle using the Kaggle Token API.

Identified the dataset and made it scalable, the paper shows some pre-processing techniques that can be applied to explore, clean and prepare data in order to identify the best possible solution to find frequent itemset.

So, finally there are shown theoretical concepts and practical implementations about the Algorithm used.

To carry on the experiment, the following steps have been executed:

1. Import the dataset
2. Implementation of the spark framework
3. Pre-processing techniques and investigation on the dataset
4. Market Basket Analysis implementation
5. Association Rule and confidence evaluation
6. Results interpretation

The code implementation is available on my GitHub repository: one can easily access to it following the link to "market_basket_analysis" repository and opening the notebook created with Colaboratory (marketbasketanalysis_AMDproject.ipynb).

Chapter 1: Import the dataset

Large-scale Dataset in a nutshell

A large-scale dataset is a dataset that contains very large amount of data. Generally speaking, any dataset with more than 1 TB of data is considered a large-scale dataset. In order to satisfy needs of a LSDS, the dataset storage needs to have low access latency and contention, high throughput and high availability.

IMDB Dataset

The IMDB dataset used in order to carry out the experiment is, as said before, a large-scale dataset.

The IMDB dataset, available on Kaggle, contains 5 datasets and each one has tab-separated values.

The first dataset is title.akas.tsv, with information for titles:

-*Title ID* (string) – the unique identifier of the title (alphanumeric);

-*ordering* (integer) – variable that allows to uniquely identify rows for a given ID;

-*title* (string) – the localized title;

-*region* (string) - the region for the version of the title presented;

-*language* (string) - the language of the title;

-*types* (array) - set of attributes for this alternative;

-*isOriginalTitle* (boolean) – 0: not original title; 1: original title.

The second dataset is title.basics.tsv, with the following information:

-*tconst* (string) - unique identifier of the title (alphanumeric);

-*titleType* (string) – the type of the title;

-*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) -> it assumes value 0 if it refers to a non-adult title; it has value 1 when it describes an adult title;

-*startYear* (YYYY) – represents the year in which a title has been realised;

-*endYear* (YYYY) – end year;

-*runtimeMinutes* – primary runtime of the title, in minutes.

-*genres* (string array) – the variable values describe genres associated with the title.

The third dataset is title.principals.tsv which the principal cast for titles:

-*tconst* (string) - unique identifier of the title;

-*ordering* (integer) – a number to uniquely identify rows for a given

titleID;

-*nconst* (string) - alphanumeric unique identifier of the person;

-*category* (string) - the category of job that person was in;

-*job* (string) - the specific job title

-*characters* (string) - the name of the character played;

The fourth is title.ratings.tsv which contains the IMDB rating and votes information for titles:

-*tconst* (string) - unique identifier of the title (alphanumeric);

-*averageRating* – weighted average of all the individual user ratings;

-*numVotes* - number of votes the title has received;

The fiveth is name.basics.tsv which contains the information about names of the persons involved:

-*nconst* (string) - unique identifier of the person;

-*primaryName* (string)– name of the person;

-*birthYear* – in YYYY format;

-*deathYear* – in YYYY format (not always applicable);

-*primaryProfession* (array of strings)– the top-3 professions of the person;

-*knownForTitles* (array of tconsts) – titles the person is known for;

Import the dataset

In order to import the dataset, one could follow lines of codes indicated in the "marketbasketanalysis.ypdb" notebook (view introduction to find hyperlink) and download the dataset directly from Kaggle, using the Kaggle token API.

The Kaggle Token API allows to import the dataset directly in the Collaboratory Notebook without having the dataset saved on the local machine, just inserting the Kaggle API associated with the Kaggle Account and then import the dataset copying the API Command from the IMDB dataset in the platform.

Chapter 2: Spark Framework for scalability

In order to collect, store and analyse the huge amount of data, the scalable solution adopted exploits the Spark Framework.

Spark is defined as a fast and general engine for large-scale data processing. It is faster than Hadoop Framework since Spark store RDD in the main memory; Hadoop, on the other hand stores data in the mass memory; the main memory allows high accessibility.

It's written in JavaScript and the great advantage is due to the possibility of making computation on distributed systems.

In the notebook one can see some of the various possible computations made possible by using Spark: running distributed SQL, creating data pipelines, running algorithms and working with graphs.

Since the Spark framework is written in JavaScript, in order to make it readable in Python 3 one needs to install some packages that let Python to communicate with Spark.

Useful Packages

To compute required operations, PySpark library has been installed.

PySpark is a Python API for Spark. It helps to interface with Resilient Distributed Datasets (RDDs) or DataFrames (tabular representation similar to relational database representation).

Among all possible PySpark packeges, in the experiment *Spark PySpark.SQL* and *PySparkML* have been required: from the first module has been imported the *SparkSession* method and from the second *FP-Growth.*

Other useful packages

-Matplotlib for some basic graphical representations;

-*Seaborn* for advanced data visualization;

-Pandas to read SQL queries in a clear way;

Recall Spark functions in Python 3

The Spark methods called along all the notebook have been summed in the "spark" object:

```
spark = SparkSession.builder.enableHiveSupport().getOrCreate()
```

Explanation :

-SparkSession: method that helps to translate and execute JavaScript Functions in Python. It's very similar to Spark Context, the difference it's related to the possibility of executing SQL queries. For this reason, for the experiments carried out in the notebook, SparkSession has been preferred.

-HiveSupport: method for the catalogue implementation; it enables the sequel views memorization.

-GetOrCreate: to create or recall an object that can be recognised by Java.

Scaling up the datasets size using Spark

Once created the Spark framework in Python 3, it is possible to scale up the datasets with the framework created.

Using the **Spark.read.csv** function *tsv* files (csv with fields delimited by tab) are read in the spark framework.

The results output are DataFrames.

In the experiment described in this notebook, it has been created a DataFrame for each input dataset, but it is also possible to group all data in the same distributed datasets. It depends on the desirable results.

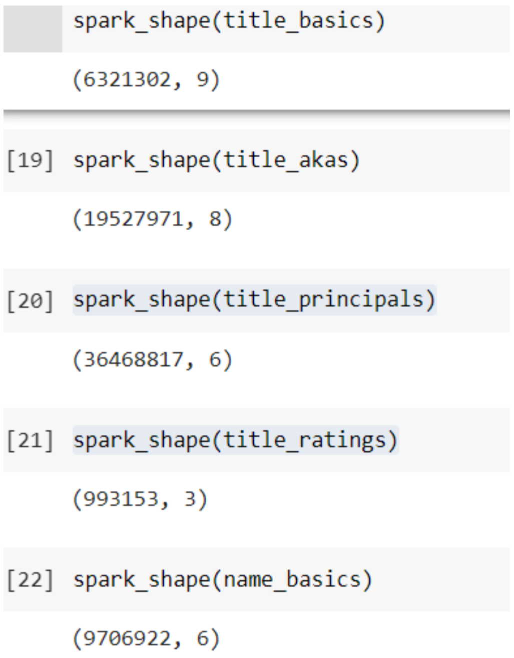
Chapter 3: Pre-processing techniques

After required the Spark framework to manage data, one can adopts some pre-processing techniques exploting the functions made available by PySpark library and other packages.

In the experiments have been used the techniques described in the following paragraphs.

Check on the structure and the dimensions of the DataFrame

Towards inspect data imported, one can firstly look at the DataFrame Shapes:



```
     spark_shape(title_basics)

     (6321302, 9)

[19] spark_shape(title_akas)

     (19527971, 8)

[20] spark_shape(title_principals)

     (36468817, 6)

[21] spark_shape(title_ratings)

     (993153, 3)

[22] spark_shape(name_basics)

     (9706922, 6)
```

-

As we can see from the output reported here, using a Spark framework is necessary for very big DataFrames.

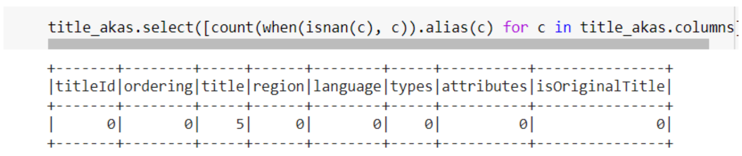
Having a look at the notebook it is also possible to view characteristics of each variable (and types).

Duplicates and null values management

In order to manage null values, in the notebook there has been implemented the *select* function: it literally selects columns in DataFrames and let to detect and count null values in each column.

The following output shows how many null values are contained in each variable of the DataFrames:

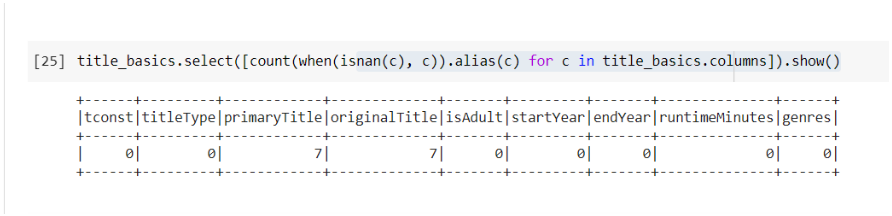
-the title_akas DataFrame contains only 5 null values over 19527971 observations;



```
title_akas.select([count(when(isnan(c), c)).alias(c) for c in title_akas.columns

+-------+--------+-----+------+--------+-----+----------+---------------+
|titleId|ordering|title|region|language|types|attributes|isOriginalTitle|
+-------+--------+-----+------+--------+-----+----------+---------------+
|      0|       0|    5|     0|       0|    0|         0|              0|
+-------+--------+-----+------+--------+-----+----------+---------------+
```

-the title_basics DataFrame includes, among 6321302 observations, only 14 null values divided between primaryTitle and originalTitle:

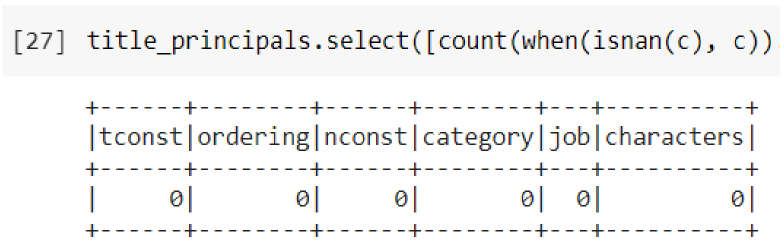


```
[25] title_basics.select([count(when(isnan(c), c)).alias(c) for c in title_basics.columns]).show()

+------+---------+------------+-------------+-------+---------+-------+--------------+------+
|tconst|titleType|primaryTitle|originalTitle|isAdult|startYear|endYear|runtimeMinutes|genres|
+------+---------+------------+-------------+-------+---------+-------+--------------+------+
|     0|        0|           7|            7|      0|        0|      0|             0|     0|
+------+---------+------------+-------------+-------+---------+-------+--------------+------+
```

-

-the title_principals DataFrame doesn't contain null values

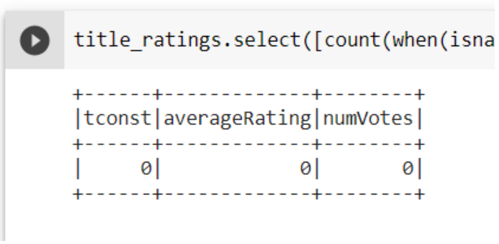


```
[27] title_principals.select([count(when(isnan(c), c))

+------+--------+------+--------+---+----------+
|tconst|ordering|nconst|category|job|characters|
+------+--------+------+--------+---+----------+
|     0|       0|     0|       0|  0|         0|
+------+--------+------+--------+---+----------+
```

-

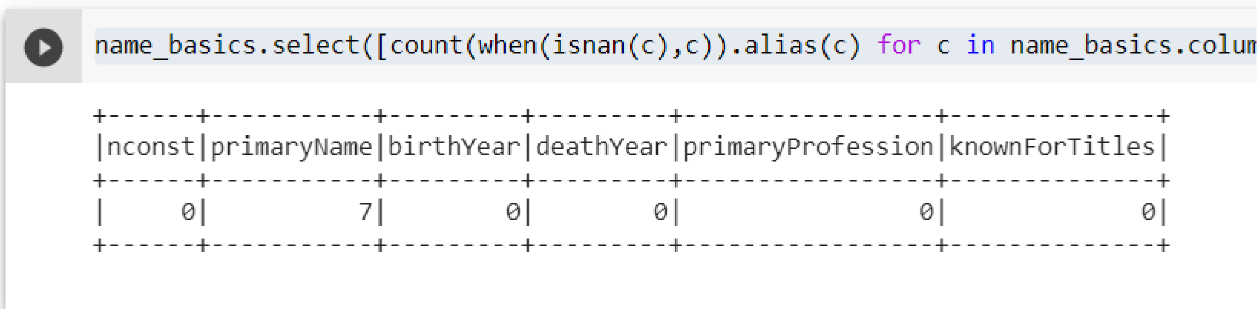
-the title_ratings DataFrame doesn't contain null values:



```
title_ratings.select([count(when(isna

+------+-------------+--------+
|tconst|averageRating|numVotes|
+------+-------------+--------+
|     0|            0|       0|
+------+-------------+--------+
```

-the name_basics DataFrame contains only 7 null values



```
name_basics.select([count(when(isnan(c),c)).alias(c) for c in name_basics.colum
```

```
+------+-----------+---------+---------+-----------------+--------------+
|nconst|primaryName|birthYear|deathYear|primaryProfession|knownForTitles|
+------+-----------+---------+---------+-----------------+--------------+
|     0|          7|        0|        0|                0|             0|
+------+-----------+---------+---------+-----------------+--------------+
```

It could be important to underline that many values have '\N' as value. In this latter case it is not a problem, since we can exclude these values during truly computations.

<u>Cleaning the DataFrames:</u>

Identified null values, the activity done in the notebook is cleaning up the dataframes:

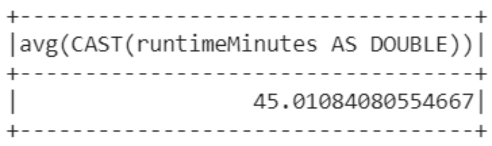
-dropping null values; to be honest, this activity could be also omitted, since the incidence of null value on overall observations has not any significance;

-dropping duplicates;

Explorative analysis

To further extract information about data, the notebook shows some queries runned exploiting the advantages of PySparkSQL (and SparkSession).

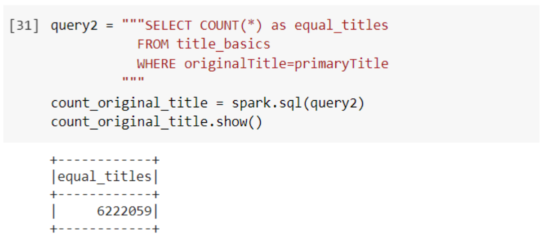
<u>1.Detect average runtime:</u>



```
+----------------------------------+
|avg(CAST(runtimeMinutes AS DOUBLE))|
+----------------------------------+
|                 45.01084080554667|
+----------------------------------+
```

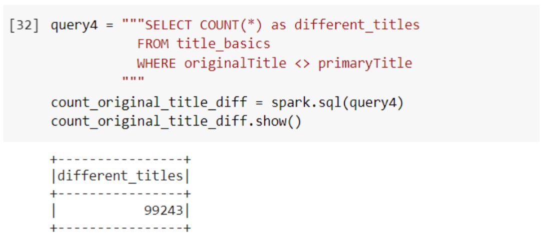
The runtime it is expressed in minutes; so, on average, movies last 45 minutes.

2. Identify how many original titles (titles written in original language) are equal to the primary titles (titles chosen by filmmakers) and how much of them differ:



```
[31] query2 = """SELECT COUNT(*) as equal_titles
               FROM title_basics
               WHERE originalTitle=primaryTitle
             """
     count_original_title = spark.sql(query2)
     count_original_title.show()

     +------------+
     |equal_titles|
     +------------+
     |     6222059|
     +------------+
```



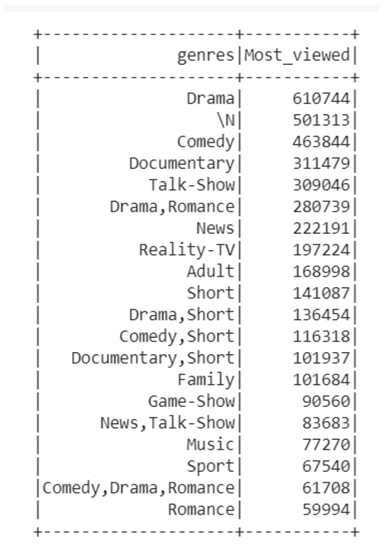
```
[32] query4 = """SELECT COUNT(*) as different_titles
               FROM title_basics
               WHERE originalTitle <> primaryTitle
             """
     count_original_title_diff = spark.sql(query4)
     count_original_title_diff.show()

     +----------------+
     |different_titles|
     +----------------+
     |           99243|
     +----------------+
```

The originalTitle and primaryTitle are equal for the 98,43% of the observations;

In the notebook, has been taken a lot of attention to the differences between these two variables, since it would be very important for the market basket analysis implementation;

3. Identify the 20 most frequent genres and shelf space taken by each genre:



```
+--------------------+-----------+
|              genres|Most_viewed|
+--------------------+-----------+
|               Drama|     610744|
|                  \N|     501313|
|              Comedy|     463844|
|         Documentary|     311479|
|           Talk-Show|     309046|
|       Drama,Romance|     280739|
|                News|     222191|
|          Reality-TV|     197224|
|               Adult|     168998|
|               Short|     141087|
|         Drama,Short|     136454|
|        Comedy,Short|     116318|
|   Documentary,Short|     101937|
|              Family|     101684|
|           Game-Show|      90560|
|      News,Talk-Show|      83683|
|               Music|      77270|
|               Sport|      67540|
|Comedy,Drama,Romance|      61708|
|             Romance|      59994|
+--------------------+-----------+
```

Some useful insights:

-the most common genre of movies is the dramatic one ;

-the lowest part of movies belongs to the Romance genre only;

-for 501313 movies there is no genre associated;

Chapter 4: Finding Similar items with Market Basket Analysis

With the implementation the market basket analysis, it is reached the aim of the entire experiment: *detect pairs of similar objects considering movies as baskets and actors as items, then find most frequent itemsets.*

(Note that “similar” has a different meaning with respect to “equal”, “similar” word considers also small variations that have to be detected.)

A **frequent itemset** is a set or documents that have common words or common sentences.

One can use the frequent itemsets for building the so-called association rules with the form: *I -> j* where I is the set of items and j is a single item.

The “Market Basket Analysis” could be defined as the technique which identifies **the strength of association** between pairs of products purchased together and identify patterns of co-occurrence.

The strength of association is defined by association rules: if in a basket are contained all the elements belonging to *I*, probably also the item *j* will be contained in this basket, creating a IF-ELSE scenario where:

-the IF part of the rule is known as antecedent;

-the THEN part of the rule is known as consequent;

Of course, each association rule needs some **measure of goodness.**

There are three common ways to measure the association: **support, confidence and lift.**

Support

Support measure gives information on popularity of an itemset and it is measured by the proportion of transactions in which an itemset appears.

Choosing support= 0.001, one infers that we consider in our basket the 0.01% of total item *j* that possibly can appear.

Confidence of the rule

The confidence of the rule - $conf(I \rightarrow j)$ indicates the number of baskets which contain all the elements belonging to the set *I* that also contain the further element *j*. It is number contained between 0 and 1.

It is expressed by the following formula:

$$conf(I \rightarrow j) = \frac{supp(I \cup \{j\})}{supp(I)}$$

where:

- $supp(I)$: count the number of baskets that contain all the elements in I;

- $supp(I \cup \{j\})$: it counts the number of baskets that contains all the elements in I, including the single item j.

Sometimes, having high confidence might be not a relevant information, indeed once one gets an association rule with high confidence, he/she should perform an additional check by computing the lift or the interest of that rule.

Lift

Lift let infers how likely an item could be in the basket when an other item is in the basket.

Interpretation:

-a lift value equal to 1 implies no association between items;

-a lift value greater than 1 means that an item is likely to be added in the basket if another item is in the basket;

-a lift value less than 1 means that item is unlikely to be added to the basket if the other item occurs.

Interest

Another important measure of goodness not included in the experiment is the so-called interest; it is explained by the following formula:

$$INT(I \rightarrow j) = conf\ (I \rightarrow j) - \frac{supp(\{j\})}{tot.numb\ of\ baskets}$$

Where:

- $INT(I \rightarrow j) \approx 0$: the itemset I has no influence on j;

- $INT(I \rightarrow j) > 0$: baskets that contain all items belonging to I, tend to contain also item j;

- $INT(I \rightarrow j) < 0$: baskets that contain all items belonging to I, tend not to contain also item j;

Practical implementation of the market basket analysis

Towards find similar items, the algorithm used in the experiment is the Frequent Pattern Growth Algorithm using the Apache Spark MLlib.

FP-growth algorithm

The Frequent Pattern Growth Algorithm (also known as FP-growth algorithm) is the tool exploited in the experiment to detect frequent itemsets.

It represents the association between itemsets, since the FP-growth Algorithm creates a frequent pattern tree (also known as FP tree) over the DataFrame.

Firstly, the dataframe is fragmented using one frequent item; the fragmented part is called "pattern fragment".

Then, the itemsets of these fragmented patterns are analysed reducing comparatively the search for frequent itemsets.

In the experiments the FP-growth algorithm has been preferred to the Apriori Algorithms since the Apriori Algorithm needs a generation of candidate itemsets to generate frequent patterns.

Using the FP-Growth, it is possible to reduces steps of the algorithm and, as a consequence, the computational time.

Steps:

1. Creating baskets

In the experiment, as it has been underlined above, the movies are considered as baskets and the actors are the items.

Extracted actors' name (items) and the movie identifiers (baskets), it has been created a basket grouping actors by movie identifiers.

The basket is created extracting data running a SQL query matching the *title_basics, title_principals* and *name_basics* dataframes.

2. FP-growth algorithm implementation building the FP-tree. In order to construct the tree it has been set up:

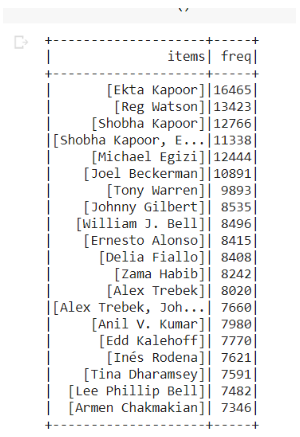
-minSupport (the minimum support for an itemset to be identified as frequent) equal to 0,001. This means that only the 0,1% of the total observations has been considered to find similar items;

-minConfidence that is the minimum confidence for generating the association rule, where confidence is an indication of how often an association rule has been found to be true.

The FP-Growth model (fitted on the basket generated at step 1) in the experiment has been called "fp_MBA".

The model provides *frequentItems, associationRules and transform* methods.

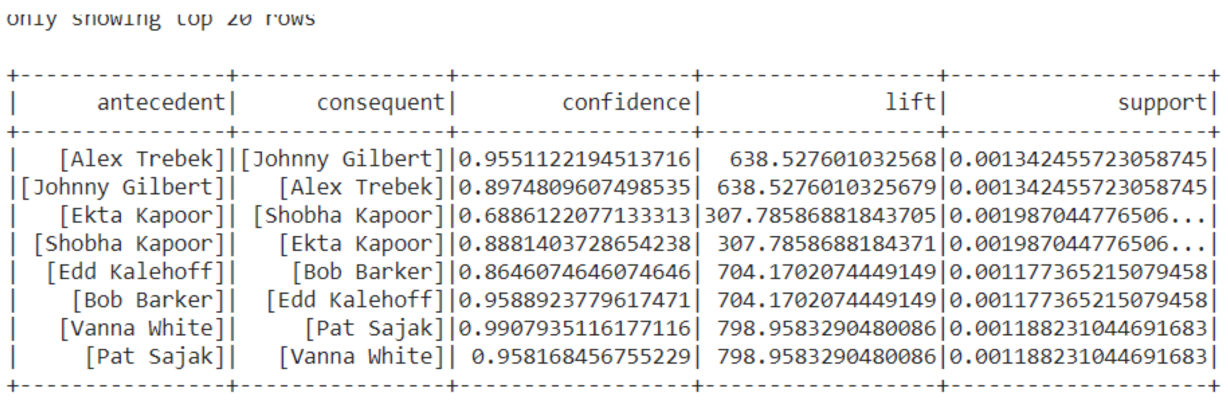
a) Applying the frequentItems method it has been extracted the information about most frequent actors:



```
+--------------------+-----+
|               items| freq|
+--------------------+-----+
|       [Ekta Kapoor]|16465|
|        [Reg Watson]|13423|
|     [Shobha Kapoor]|12766|
|[Shobha Kapoor, E...|11338|
|     [Michael Egizi]|12444|
|    [Joel Beckerman]|10891|
|       [Tony Warren]| 9893|
|    [Johnny Gilbert]| 8535|
|   [William J. Bell]| 8496|
|    [Ernesto Alonso]| 8415|
|      [Delia Fiallo]| 8408|
|        [Zama Habib]| 8242|
|       [Alex Trebek]| 8020|
|[Alex Trebek, Joh...| 7660|
|     [Anil V. Kumar]| 7980|
|      [Edd Kalehoff]| 7770|
|       [Inés Rodena]| 7621|
|    [Tina Dharamsey]| 7591|
|  [Lee Phillip Bell]| 7482|
|  [Armen Chakmakian]| 7346|
+--------------------+-----+
```

From the attached output, it is visible that Ekta Kapoor is the actors that appears more frequently: he took part in 16465 movies.

b) Applying the AssociationRules method it has been generated the association rules



```
only showing top 20 rows

+---------------+---------------+------------------+------------------+--------------------+
|     antecedent|     consequent|        confidence|              lift|             support|
+---------------+---------------+------------------+------------------+--------------------+
|  [Alex Trebek]|[Johnny Gilbert]|0.9551122194513716|  638.527601032568|0.001342455723058745|
|[Johnny Gilbert]|   [Alex Trebek]|0.8974809607498535| 638.5276010325679|0.001342455723058745|
|  [Ekta Kapoor]| [Shobha Kapoor]|0.6886122077133313|307.78586881843705|0.001987044776506...|
| [Shobha Kapoor]|   [Ekta Kapoor]|0.8881403728654238| 307.7858688184371|0.001987044776506...|
| [Edd Kalehoff]|    [Bob Barker]|0.8646074646074646| 704.1702074449149|0.001177365215079458|
|   [Bob Barker]|  [Edd Kalehoff]|0.9588923779617471| 704.1702074449149|0.001177365215079458|
|  [Vanna White]|     [Pat Sajak]|0.9907935116177116| 798.9583290480086|0.001188231044691683|
|    [Pat Sajak]|   [Vanna White]| 0.958168456755229| 798.9583290480086|0.001188231044691683|
+---------------+---------------+------------------+------------------+--------------------+
```

Where:

-antecedent(if) and consequent(then) are the two parts of the association rule; the antecedent is the item found in the DataFrame and the consequent is the item found in combination with the antecedent;

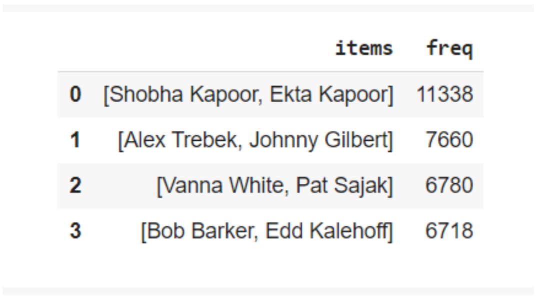
-confidence, lift and support are the criteria used to identify the most important relationships; in this way one could know how frequently the items appear in the data (support), the number of times the if-then statements are found true (confidence) and how many times an if-then statement is expected to be found true (lift)

From the output shown above, one can infer that if Alex Trebek takes part in the movie, also Johnny Gilbert will be an actor of that movie with 95,5% of confidence. As well as, with 89,7% of confidence, if Girlbert takes part in the movie also Trebek will be in that cast.

As one can deduce looking at the output, the most important relationship is between Vanna White as antecedent and Pat Sajak as consequent: with 99% of confidence and the higher value for lift.

Conclusions

Starting from some useful preprocessing techniques and exploting the Spark Framework for obtaining scalability in the application of the FP-Growth Algorithm, in the report it have been shown the most frequent Itemsets in the basket:



	items	freq
0	[Shobha Kapoor, Ekta Kapoor]	11338
1	[Alex Trebek, Johnny Gilbert]	7660
2	[Vanna White, Pat Sajak]	6780
3	[Bob Barker, Edd Kalehoff]	6718

Nonetheless the confidence and the lift are lower (view results about association rules), Shobha Kapoor and Ekta Kapoor appear together 11338 times, more than Vanna White and Pat Sajak and this is due to the higher value of support.

So, depending on the aim of the analysis, one has to consider all the three measures and evaluate their relevancy.