20230972_20230560_submission

June 24, 2024

1 Six Degrees of Kevin Bacon

Introduction - Six Degrees of Kevin Bacon is a game based on the "six degrees of separation" concept, which posits that any two people on Earth are six or fewer acquaintance links apart. Movie buffs challenge each other to find the shortest path between an arbitrary actor and prolific actor Kevin Bacon. It rests on the assumption that anyone involved in the film industry can be linked through their film roles to Bacon within six steps. The analysis of social networks can be a computationally intensive task, especially when dealing with large volumes of data. It is also a challenging problem to devise a correct methodology to infer an informative social network structure. Here, we will analyze a social network of actors and actresses that co-participated in movies. We will do some simple descriptive analysis, and in the end try to relate an actor/actress's position in the social network with the success of the movies in which they participate.

Rules & Notes - Please take your time to read the following points:

1. The submission deadline shall be set for the 10th of June at 23:59.
2. It is acceptable that you **discuss** with your colleagues different approaches to solve each step of the problem set. You are responsible for writing your own code, and analysing the results. Clear cases of cheating will be penalized with 0 points in this assignment;
3. After review of your submission files, and before a mark is attributed, you might be called to orally defend your submission;
4. You will be scored first and foremost by the number of correct answers, secondly by the logic used in the trying to approach each step of the problem set;
5. Consider skipping questions that you are stuck in, and get back to them later;
6. Expect computations to take a few minutes to finish in some of the steps.
7. **IMPORTANT** It is expected you have developed skills beyond writting SQL queries. Any question where you directly write a SQL query (then for example create a temporary table and use spark.sql to pass the query) will receive a 25% penalty. Using the Spark syntax (for example dataframe.select("*").where("conditions")) is acceptable and does not incur this penalty. Comment your code in a reasonable fashion.
8. **Questions** – Any questions about this assignment should be posted in the Forum@Moodle. The last class will be an open office session for anyone with questions concerning the assignment.
9. **Delivery** - To fulfil this activity you will have to upload the following materials to Moodle:
    1. An exported IPython notebook. The notebook should be solved (have results displayed), but should contain all neccesary code so that when the notebook is run in databricks it should also replicate these results. This means the all data downloading and processing should be done in this notebook. It is also important you clearly indicate where your final

answer to each question is when you are using multiple cells (for example you print "my final anwser is" before your answer or use cell comments). Please make sure to name your file in the following way: *[student_number1]_[student_number2]_submission.ipynb. *As an example:* 19740001_197400010_submission.ipynb*

2. **Delivery** - You will also need to provide a signed statement of authorship, which is present in the last page;
3. It is recommended you read the whole assignment before starting.
4. You can add as many cells as you like to answer the questions.
5. You can make use of caching or persisting your RDDs or Dataframes, this may speed up performance.
6. If you have trouble with graphframes in databricks (specifically the import statement) you need to make sure the graphframes package is installed on the cluster you are running. If you click home on the left, then click on the graphframes library, from where you can install the package on your cluster (check the graphframes checkbox and click install). Another installation option is using the JAR available on Moodle with the graphframes library.

10. **Note**: By including the name and student number of each group member in the submission notebook, this will be considered as a declaration of authorship.

Data Sources and Description We will use data from IMDB. You can download raw datafiles from https://datasets.imdbws.com. Note that the files are tab delimited (.tsv) You can find a description of the each datafile in https://www.imdb.com/interfaces/

1.1 Questions

1.1.1 Data loading and preperation

Review the file descriptions and load the necessary data onto your databricks cluser and into spark dataframes. You will need to use shell commands to download the data, unzip the data, load the data into spark. Note that the data might require parsing and preprocessing to be ready for the questions below.

Hints You can use 'gunzip' to unzip the .tz files. The data files will then be tab seperated (.tsv), which you can load into a dataframe using the tab seperated option instead of the comma seperated option we have typically used in class: `.option("sep","\t")`

[ ]:
```
%sh

wget "https://datasets.imdbws.com/name.basics.tsv.gz" -O /tmp/name.basics.tsv.gz
wget "https://datasets.imdbws.com/title.akas.tsv.gz" -O /tmp/title.akas.tsv.gz
wget "https://datasets.imdbws.com/title.basics.tsv.gz" -O /tmp/title.basics.tsv.
↪gz
wget "https://datasets.imdbws.com/title.crew.tsv.gz" -O /tmp/title.crew.tsv.gz
wget "https://datasets.imdbws.com/title.episode.tsv.gz" -O /tmp/title.episode.
↪tsv.gz
wget "https://datasets.imdbws.com/title.principals.tsv.gz" -O /tmp/title.
↪principals.tsv.gz
```

```
wget "https://datasets.imdbws.com/title.ratings.tsv.gz" -O /tmp/title.ratings.
↪tsv.gz
```

```
--2024-06-09 07:53:56--  https://datasets.imdbws.com/name.basics.tsv.gz
Resolving datasets.imdbws.com (datasets.imdbws.com)… 18.245.253.55,
18.245.253.117, 18.245.253.85, …
Connecting to datasets.imdbws.com (datasets.imdbws.com)|18.245.253.55|:443…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 267637948 (255M) [binary/octet-stream]
Saving to: '/tmp/name.basics.tsv.gz'

     0K … … … … …  0% 15.2M 17s
    50K … … … … …  0% 20.8M 15s
   100K … … … … …  0% 19.6M 14s
   150K … … … … …  0% 21.1M 14s
   200K … … … … …  0% 21.8M 13s
   250K … … … … …  0% 91.1M 11s
   300K … … … … …  0% 68.3M 10s
   350K … … … … …  0% 32.4M 10s
   400K … … … … …  0% 45.4M 10s
   450K … … … … …  0% 30.1M 9s
   500K … … … … …  0% 49.4M 9s
   550K … … … … …  0%  115M 8s
   600K … … … … …  0% 66.6M 8s
   650K … … … … …  0%  205M 8s
   700K … … … … …  0% 76.7M 7s
   750K … … … … …  0% 74.5M 7s
   800K … … … … …  0%  114M 7s
   850K … … … … …  0% 87.5M 7s
   900K … … … … …  0% 76.2M 6s
   950K … … … … …  0%  338M 6s
  1000K … … … … …  0% 67.4M 6s
  1050K … … … … …  0% 87.7M 6s
  1100K … … … … …  0% 84.4M 6s
  1150K … … … … …  0% 66.8M 6s
  1200K … … … … …  0%  368M 5s
  1250K … … … … …  0% 91.2M 5s
  1300K … … … … …  0%  111M 5s
  1350K … … … … …  0% 86.4M 5s
  1400K … … … … …  0%  126M 5s
  1450K … … … … …  0%  445M 5s
  1500K … … … … …  0%  110M 5s
  1550K … … … … …  0% 73.7M 5s
  1600K … … … … …  0%  138M 5s
  1650K … … … … …  0%  109M 5s
  1700K … … … … …  0% 87.7M 5s
  1750K … … … … …  0%  382M 4s
```

```
1800K … … … … …  0%  114M 4s
1850K … … … … …  0%  120M 4s
1900K … … … … …  0%  127M 4s
1950K … … … … …  0%  296M 4s
2000K … … … … …  0%  103M 4s
2050K … … … … …  0%  129M 4s
2100K … … … … …  0%  117M 4s
2150K … … … … …  0%  105M 4s
2200K … … … … …  0%  329M 4s
2250K … … … … …  0%  104M 4s
2300K … … … … …  0%  181M 4s
2350K … … … … …  0%  153M 4s
2400K … … … … …  0%  126M 4s
2450K … … … … …  0%  213M 4s
2500K … … … … …  0%  190M 4s
2550K … … … … …  0%  142M 4s
2600K … … … … …  1%  117M 4s
2650K … … … … …  1%  150M 4s
2700K … … … … …  1%  410M 4s
2750K … … … … …  1%  111M 3s
2800K … … … … …  1%  142M 3s
2850K … … … … …  1%  165M 3s
2900K … … … … …  1%  151M 3s
2950K … … … … …  1%  331M 3s
3000K … … … … …  1%  150M 3s
3050K … … … … …  1%  156M 3s
3100K … … … … …  1%  155M 3s
3150K … … … … …  1%  404M 3s
3200K … … … … …  1%  137M 3s
3250K … … … … …  1%  185M 3s
3300K … … … … …  1%  162M 3s
3350K … … … … …  1%  168M 3s
3400K … … … … …  1%  152M 3s
3450K … … … … …  1%  365M 3s
3500K … … … … …  1%  172M 3s
3550K … … … … …  1%  132M 3s
3600K … … … … …  1%  207M 3s
3650K … … … … …  1%  538M 3s
3700K … … … … …  1%  144M 3s
3750K … … … … …  1%  230M 3s
3800K … … … … …  1%  150M 3s
3850K … … … … …  1%  207M 3s
3900K … … … … …  1%  431M 3s
3950K … … … … …  1%  154M 3s
4000K … … … … …  1%  275M 3s
4050K … … … … …  1%  228M 3s
4100K … … … … …  1%  203M 3s
4150K … … … … …  1%  221M 3s
```

```
4200K … … … … …  1%  329M 3s
4250K … … … … …  1%  196M 3s
4300K … … … … …  1%  210M 3s
4350K … … … … …  1%  190M 3s
4400K … … … … …  1%  219M 3s
4450K … … … … …  1%  300M 3s
4500K … … … … …  1%  190M 3s
4550K … … … … …  1%  264M 3s
4600K … … … … …  1%  225M 3s
4650K … … … … …  1%  343M 3s
4700K … … … … …  1%  198M 3s
4750K … … … … …  1%  224M 3s
4800K … … … … …  1% 64.2M 3s
4850K … … … … …  1%  304M 3s
4900K … … … … …  1%  363M 3s
4950K … … … … …  1%  107M 3s
5000K … … … … …  1%  176M 3s
5050K … … … … …  1%  388M 2s
5100K … … … … …  1%  156M 2s
5150K … … … … …  1%  261M 2s
5200K … … … … …  2%  564M 2s
5250K … … … … …  2%  184M 2s
5300K … … … … …  2%  243M 2s
5350K … … … … …  2%  158M 2s
5400K … … … … …  2%  323M 2s
5450K … … … … …  2%  233M 2s
5500K … … … … …  2%  573M 2s
5550K … … … … …  2%  241M 2s
5600K … … … … …  2%  263M 2s
5650K … … … … …  2%  390M 2s
5700K … … … … …  2%  429M 2s
5750K … … … … …  2%  230M 2s
5800K … … … … …  2%  250M 2s
5850K … … … … …  2%  287M 2s
5900K … … … … …  2%  326M 2s
5950K … … … … …  2%  272M 2s
6000K … … … … …  2%  277M 2s
6050K … … … … …  2%  308M 2s
6100K … … … … …  2%  308M 2s
6150K … … … … …  2%  330M 2s
6200K … … … … …  2%  400M 2s
6250K … … … … …  2%  298M 2s
6300K … … … … …  2%  364M 2s
6350K … … … … …  2%  318M 2s
6400K … … … … …  2%  391M 2s
6450K … … … … …  2%  358M 2s
6500K … … … … …  2%  421M 2s
6550K … … … … …  2%  347M 2s
```

```
6600K … … … … …  2%  420M 2s
6650K … … … … …  2%  314M 2s
6700K … … … … …  2%  427M 2s
6750K … … … … …  2%  344M 2s
6800K … … … … …  2%  358M 2s
6850K … … … … …  2%  410M 2s
6900K … … … … …  2%  439M 2s
6950K … … … … …  2%  381M 2s
7000K … … … … …  2%  358M 2s
7050K … … … … …  2%  347M 2s
7100K … … … … …  2%  418M 2s
7150K … … … … …  2%  390M 2s
7200K … … … … …  2%  319M 2s
7250K … … … … …  2%  419M 2s
7300K … … … … …  2%  396M 2s
7350K … … … … …  2%  326M 2s
7400K … … … … …  2%  362M 2s
7450K … … … … …  2%  348M 2s
7500K … … … … …  2%  392M 2s
7550K … … … … …  2%  359M 2s
7600K … … … … …  2%  308M 2s
7650K … … … … …  2%  389M 2s
7700K … … … … …  2%  351M 2s
7750K … … … … …  2%  364M 2s
7800K … … … … …  3%  391M 2s
7850K … … … … …  3%  310M 2s
7900K … … … … …  3%  394M 2s
7950K … … … … …  3%  318M 2s
8000K … … … … …  3%  350M 2s
8050K … … … … …  3%  398M 2s
8100K … … … … …  3%  278M 2s
8150K … … … … …  3%  384M 2s
8200K … … … … …  3%  343M 2s
8250K … … … … …  3%  339M 2s
8300K … … … … …  3%  339M 2s
8350K … … … … …  3%  359M 2s
8400K … … … … …  3%  377M 2s
8450K … … … … …  3%  342M 2s
8500K … … … … …  3%  368M 2s
8550K … … … … …  3%  299M 2s
8600K … … … … …  3%  362M 2s
8650K … … … … …  3%  273M 2s
8700K … … … … …  3%  337M 2s
8750K … … … … …  3%  365M 2s
8800K … … … … …  3%  355M 2s
8850K … … … … …  3%  374M 2s
8900K … … … … …  3%  324M 2s
8950K … … … … …  3%  362M 2s
```

```
  9000K … … … … …  3%  373M 2s
  9050K … … … … …  3%  387M 2s
  9100K … … … … …  3%  316M 2s
  9150K … … … … …  3%  346M 2s
  9200K … … … … …  3%  326M 2s
  9250K … … … … …  3%  431M 2s
  9300K … … … … …  3%  314M 2s
  9350K … … … … …  3%  421M 2s
  9400K … … … … …  3%  387M 2s
  9450K … … … … …  3%  295M 2s
  9500K … … … … …  3%  341M 2s
  9550K … … … … …  3%  345M 2s
  9600K … … … … …  3%  314M 2s
  9650K … … … … …  3%  335M 2s
  9700K … … … … …  3%  388M 2s
  9750K … … … … …  3%  401M 2s
  9800K … … … … …  3%  366M 2s
  9850K … … … … …  3%  366M 2s
  9900K … … … … …  3%  393M 2s
  9950K … … … … …  3%  383M 2s
 10000K … … … … …  3%  284M 2s
 10050K … … … … …  3%  388M 2s
 10100K … … … … …  3%  296M 2s
 10150K … … … … …  3%  343M 2s
 10200K … … … … …  3%  368M 2s
 10250K … … … … …  3%  299M 2s
 10300K … … … … …  3%  344M 2s
 10350K … … … … …  3%  323M 2s
 10400K … … … … …  3%  354M 2s
 10450K … … … … …  4%  308M 2s
 10500K … … … … …  4%  331M 2s
 10550K … … … … …  4%  336M 2s
 10600K … … … … …  4%  329M 2s
 10650K … … … … …  4%  325M 2s
 10700K … … … … …  4%  345M 2s
 10750K … … … … …  4%  259M 2s
 10800K … … … … …  4%  319M 2s
 10850K … … … … …  4%  341M 2s
 10900K … … … … …  4%  353M 2s
 10950K … … … … …  4%  311M 2s
 11000K … … … … …  4%  265M 2s
 11050K … … … … …  4%  337M 2s
 11100K … … … … …  4%  327M 2s
 11150K … … … … …  4%  335M 2s
 11200K … … … … …  4%  277M 2s
 11250K … … … … …  4%  330M 1s
 11300K … … … … …  4%  337M 1s
 11350K … … … … …  4%  332M 1s
```

```
11400K … … … … …  4%  283M 1s
11450K … … … … …  4%  295M 1s
11500K … … … … …  4%  323M 1s
11550K … … … … …  4%  353M 1s
11600K … … … … …  4%  344M 1s
11650K … … … … …  4%  282M 1s
11700K … … … … …  4%  341M 1s
11750K … … … … …  4%  349M 1s
11800K … … … … …  4%  380M 1s
11850K … … … … …  4%  302M 1s
11900K … … … … …  4%  315M 1s
11950K … … … … …  4%  383M 1s
12000K … … … … …  4%  339M 1s
12050K … … … … …  4%  305M 1s
12100K … … … … …  4%  360M 1s
12150K … … … … …  4%  337M 1s
12200K … … … … …  4%  410M 1s
12250K … … … … …  4%  299M 1s
12300K … … … … …  4%  347M 1s
12350K … … … … …  4%  327M 1s
12400K … … … … …  4%  370M 1s
12450K … … … … …  4%  322M 1s
12500K … … … … …  4%  326M 1s
12550K … … … … …  4%  304M 1s
12600K … … … … …  4%  329M 1s
12650K … … … … …  4%  319M 1s
12700K … … … … …  4%  310M 1s
12750K … … … … …  4%  247M 1s
12800K … … … … …  4%  350M 1s
12850K … … … … …  4%  360M 1s
12900K … … … … …  4%  399M 1s
12950K … … … … …  4%  487M 1s
13000K … … … … …  4%  330M 1s
13050K … … … … …  5%  365M 1s
13100K … … … … …  5%  347M 1s
13150K … … … … …  5%  327M 1s
13200K … … … … …  5%  317M 1s
13250K … … … … …  5%  273M 1s
13300K … … … … …  5%  409M 1s
13350K … … … … …  5%  372M 1s
13400K … … … … …  5%  326M 1s
13450K … … … … …  5%  412M 1s
13500K … … … … …  5%  389M 1s
13550K … … … … …  5%  377M 1s
13600K … … … … …  5%  367M 1s
13650K … … … … …  5%  314M 1s
13700K … … … … …  5%  394M 1s
13750K … … … … …  5%  425M 1s
```

```
13800K … … … … …  5%  327M 1s
13850K … … … … …  5%  356M 1s
13900K … … … … …  5%  308M 1s
13950K … … … … …  5%  362M 1s
14000K … … … … …  5%  289M 1s
14050K … … … … …  5%  417M 1s
14100K … … … … …  5%  440M 1s
14150K … … … … …  5%  321M 1s
14200K … … … … …  5%  280M 1s
14250K … … … … …  5%  315M 1s
14300K … … … … …  5%  362M 1s
14350K … … … … …  5%  326M 1s
14400K … … … … …  5%  326M 1s
14450K … … … … …  5%  315M 1s
14500K … … … … …  5%  358M 1s
14550K … … … … …  5%  316M 1s
14600K … … … … …  5%  359M 1s
14650K … … … … …  5%  277M 1s
14700K … … … … …  5%  330M 1s
14750K … … … … …  5%  275M 1s
14800K … … … … …  5%  351M 1s
14850K … … … … …  5%  346M 1s
14900K … … … … …  5%  272M 1s
14950K … … … … …  5%  298M 1s
15000K … … … … …  5%  337M 1s
15050K … … … … …  5%  310M 1s
15100K … … … … …  5%  347M 1s
15150K … … … … …  5%  279M 1s
15200K … … … … …  5%  336M 1s
15250K … … … … …  5%  357M 1s
15300K … … … … …  5%  305M 1s
15350K … … … … …  5%  331M 1s
15400K … … … … …  5%  321M 1s
15450K … … … … …  5%  338M 1s
15500K … … … … …  5%  319M 1s
15550K … … … … …  5%  320M 1s
15600K … … … … …  5%  384M 1s
15650K … … … … …  6%  322M 1s
15700K … … … … …  6%  411M 1s
15750K … … … … …  6%  424M 1s
15800K … … … … …  6%  339M 1s
15850K … … … … …  6%  365M 1s
15900K … … … … …  6%  334M 1s
15950K … … … … …  6%  361M 1s
16000K … … … … …  6%  342M 1s
16050K … … … … …  6%  278M 1s
16100K … … … … …  6%  311M 1s
16150K … … … …
```

```
*** WARNING: max output size exceeded, skipping output. ***

… 98%  341M 0s
643750K … … … … … 98% 1.92M 0s
643800K … … … … … 98%  317M 0s
643850K … … … … … 98%  390M 0s
643900K … … … … … 98%  380M 0s
643950K … … … … … 98%  328M 0s
644000K … … … … … 98%  386M 0s
644050K … … … … … 98%  367M 0s
644100K … … … … … 98%  273M 0s
644150K … … … … … 98%  353M 0s
644200K … … … … … 98%  386M 0s
644250K … … … … … 98%  339M 0s
644300K … … … … … 98%  371M 0s
644350K … … … … … 98%  377M 0s
644400K … … … … … 98%  319M 0s
644450K … … … … … 98%  378M 0s
644500K … … … … … 98%  392M 0s
644550K … … … … … 98%  368M 0s
644600K … … … … … 98%  318M 0s
644650K … … … … … 98%  351M 0s
644700K … … … … … 98%  339M 0s
644750K … … … … … 98%  343M 0s
644800K … … … … … 98%  370M 0s
644850K … … … … … 98%  367M 0s
644900K … … … … … 98%  323M 0s
644950K … … … … … 98%  431M 0s
645000K … … … … … 98%  393M 0s
645050K … … … … … 98%  388M 0s
645100K … … … … … 98%  345M 0s
645150K … … … … … 98%  377M 0s
645200K … … … … … 98%  367M 0s
645250K … … … … … 98%  454M 0s
645300K … … … … … 98%  385M 0s
645350K … … … … … 98%  351M 0s
645400K … … … … … 98%  342M 0s
645450K … … … … … 98%  377M 0s
645500K … … … … … 98%  345M 0s
645550K … … … … … 98%  365M 0s
645600K … … … … … 98%  256M 0s
645650K … … … … … 98%  384M 0s
645700K … … … … … 98%  315M 0s
645750K … … … … … 98%  339M 0s
645800K … … … … … 98%  378M 0s
645850K … … … … … 98%  343M 0s
645900K … … … … … 98%  383M 0s
```

```
645950K … … … … … 98%  378M 0s
646000K … … … … … 98%  355M 0s
646050K … … … … … 98%  372M 0s
646100K … … … … … 99%  359M 0s
646150K … … … … … 99%  397M 0s
646200K … … … … … 99%  359M 0s
646250K … … … … … 99%  325M 0s
646300K … … … … … 99%  368M 0s
646350K … … … … … 99%  357M 0s
646400K … … … … … 99%  346M 0s
646450K … … … … … 99%  348M 0s
646500K … … … … … 99%  384M 0s
646550K … … … … … 99%  359M 0s
646600K … … … … … 99%  380M 0s
646650K … … … … … 99%  323M 0s
646700K … … … … … 99%  387M 0s
646750K … … … … … 99%  378M 0s
646800K … … … … … 99%  315M 0s
646850K … … … … … 99%  375M 0s
646900K … … … … … 99%  352M 0s
646950K … … … … … 99%  374M 0s
647000K … … … … … 99%  383M 0s
647050K … … … … … 99%  309M 0s
647100K … … … … … 99%  368M 0s
647150K … … … … … 99%  331M 0s
647200K … … … … … 99%  363M 0s
647250K … … … … … 99%  367M 0s
647300K … … … … … 99%  389M 0s
647350K … … … … … 99%  382M 0s
647400K … … … … … 99%  393M 0s
647450K … … … … … 99%  408M 0s
647500K … … … … … 99%  455M 0s
647550K … … … … … 99%  338M 0s
647600K … … … … … 99%  424M 0s
647650K … … … … … 99%  354M 0s
647700K … … … … … 99%  375M 0s
647750K … … … … … 99%  340M 0s
647800K … … … … … 99%  347M 0s
647850K … … … … … 99%  310M 0s
647900K … … … … … 99%  409M 0s
647950K … … … … … 99%  381M 0s
648000K … … … … … 99%  332M 0s
648050K … … … … … 99%  267M 0s
648100K … … … … … 99%  360M 0s
648150K … … … … … 99%  381M 0s
648200K … … … … … 99%  367M 0s
648250K … … … … … 99%  276M 0s
648300K … … … … … 99%  371M 0s
```

```
648350K … … … … … 99%  344M 0s
648400K … … … … … 99%  389M 0s
648450K … … … … … 99%  305M 0s
648500K … … … … … 99%  285M 0s
648550K … … … … … 99%  324M 0s
648600K … … … … … 99%  363M 0s
648650K … … … … … 99%  300M 0s
648700K … … … … … 99%  369M 0s
648750K … … … … … 99%  335M 0s
648800K … … … … … 99%  309M 0s
648850K … … … … … 99%  367M 0s
648900K … … … … … 99%  282M 0s
648950K … … … … … 99%  370M 0s
649000K … … … … … 99%  378M 0s
649050K … … … … … 99%  376M 0s
649100K … … … … … 99% 1.17M 0s
649150K … … … … … 99%  466M 0s
649200K … … … … … 99%  356M 0s
649250K … … … … … 99%  511M 0s
649300K … … … … … 99%  466M 0s
649350K … … … … … 99%  419M 0s
649400K … … … … … 99%  381M 0s
649450K … … … … … 99%  433M 0s
649500K … … … … … 99%  346M 0s
649550K … … … … … 99%  445M 0s
649600K … … … … … 99%  444M 0s
649650K … … … … … 99%  486M 0s
649700K … … … … … 99%  425M 0s
649750K … … … … … 99%  474M 0s
649800K … … … … … 99%  476M 0s
649850K … … … … … 99%  497M 0s
649900K … … … … … 99%  421M 0s
649950K … … … … … 99%  474M 0s
650000K … … … … … 99%  392M 0s
650050K … … … … … 99%  467M 0s
650100K … … … … … 99%  377M 0s
650150K … … … … … 99%  442M 0s
650200K … … … … … 99%  462M 0s
650250K … … … … … 99%  444M 0s
650300K … … … … … 99%  451M 0s
650350K … … … … … 99%  393M 0s
650400K … … … … … 99%  471M 0s
650450K … … … … … 99%  390M 0s
650500K … … … … … 99%  459M 0s
650550K … … … … … 99%  477M 0s
650600K … … … … … 99%  367M 0s
650650K … … … … … 99%  452M 0s
650700K … … … … … 99%  444M 0s
```

```
650750K … … … … … 99%  445M 0s
650800K … … … … … 99%  474M 0s
650850K … … … … … 99%  386M 0s
650900K … … … … … 99%  566M 0s
650950K … … … … … 99%  518M 0s
651000K … … … … … 99%  436M 0s
651050K … … … … … 99%  440M 0s
651100K … … … … … 99%  490M 0s
651150K … … … … … 99%  433M 0s
651200K … … … … … 99%  411M 0s
651250K … … … … … 99%  322M 0s
651300K … … … … … 99%  267M 0s
651350K … … … … … 99%  258M 0s
651400K … … … … … 99%  311M 0s
651450K … … … … … 99%  381M 0s
651500K … … … … … 99%  611M 0s
651550K … … … … … 99%  623M 0s
651600K … … … … … 99%  549M 0s
651650K … … … … … 99%  627M 0s
651700K … … … … … 99%  557M 0s
651750K … … … … … 99%  606M 0s
651800K … … … … … 99%  650M 0s
651850K … … … … … 99%  560M 0s
651900K … … … … … 99%  635M 0s
651950K … … … … … 99%  574M 0s
652000K … … … … … 99%  617M 0s
652050K … … … … … 99%  537M 0s
652100K … … … … … 99%  569M 0s
652150K … … … … … 99%  635M 0s
652200K … … … … … 99%  569M 0s
652250K … … … … … 99%  631M 0s
652300K … … … … … 99%  586M 0s
652350K … … … … … 99%  559M 0s
652400K … … … … … 99%  637M 0s
652450K … … … … … 99%  585M 0s
652500K … … … … … 99%  570M 0s
652550K … … … … … 99%  575M 0s
652600K … … … … … 99%  445M 0s
652650K … …                                        100%  558M=6.5s

2024-06-09 07:54:12 (98.6 MB/s) - ‘/tmp/title.principals.tsv.gz’ saved
[668327000/668327000]

--2024-06-09 07:54:12--  https://datasets.imdbws.com/title.ratings.tsv.gz
Resolving datasets.imdbws.com (datasets.imdbws.com)… 18.245.253.55,
18.245.253.70, 18.245.253.85, …
Connecting to datasets.imdbws.com (datasets.imdbws.com)|18.245.253.55|:443…
connected.
```

```
HTTP request sent, awaiting response… 200 OK
Length: 7261619 (6.9M) [binary/octet-stream]
Saving to: ‘/tmp/title.ratings.tsv.gz’

     0K … … … … …  0% 21.0M 0s
    50K … … … … …  1% 17.8M 0s
   100K … … … … …  2% 22.0M 0s
   150K … … … … …  2%  108M 0s
   200K … … … … …  3% 59.1M 0s
   250K … … … … …  4% 87.5M 0s
   300K … … … … …  4% 59.4M 0s
   350K … … … … …  5%  166M 0s
   400K … … … … …  6% 83.3M 0s
   450K … … … … …  7% 68.3M 0s
   500K … … … … …  7%  180M 0s
   550K … … … … …  8%  109M 0s
   600K … … … … …  9%  327M 0s
   650K … … … … …  9% 73.5M 0s
   700K … … … … … 10%  301M 0s
   750K … … … … … 11%  248M 0s
   800K … … … … … 11%  329M 0s
   850K … … … … … 12% 69.0M 0s
   900K … … … … … 13% 77.8M 0s
   950K … … … … … 14% 85.6M 0s
  1000K … … … … … 14%  276M 0s
  1050K … … … … … 15%  243M 0s
  1100K … … … … … 16% 60.0M 0s
  1150K … … … … … 16%  244M 0s
  1200K … … … … … 17%  153M 0s
  1250K … … … … … 18%  364M 0s
  1300K … … … … … 19%  329M 0s
  1350K … … … … … 19%  296M 0s
  1400K … … … … … 20%  371M 0s
  1450K … … … … … 21% 4.32M 0s
  1500K … … … … … 21%  251M 0s
  1550K … … … … … 22%  270M 0s
  1600K … … … … … 23%  300M 0s
  1650K … … … … … 23%  271M 0s
  1700K … … … … … 24%  301M 0s
  1750K … … … … … 25%  246M 0s
  1800K … … … … … 26%  295M 0s
  1850K … … … … … 26%  302M 0s
  1900K … … … … … 27%  311M 0s
  1950K … … … … … 28%  282M 0s
  2000K … … … … … 28%  273M 0s
  2050K … … … … … 29%  313M 0s
  2100K … … … … … 30%  335M 0s
  2150K … … … … … 31%  251M 0s
```

```
2200K … … … … … 31% 87.8M 0s
2250K … … … … … 32%  102M 0s
2300K … … … … … 33%  295M 0s
2350K … … … … … 33%  333M 0s
2400K … … … … … 34%  310M 0s
2450K … … … … … 35%  169M 0s
2500K … … … … … 35%  123M 0s
2550K … … … … … 36% 74.7M 0s
2600K … … … … … 37%  124M 0s
2650K … … … … … 38%  108M 0s
2700K … … … … … 38%  291M 0s
2750K … … … … … 39% 23.3M 0s
2800K … … … … … 40%  320M 0s
2850K … … … … … 40%  193M 0s
2900K … … … … … 41%  323M 0s
2950K … … … … … 42%  332M 0s
3000K … … … … … 43%  285M 0s
3050K … … … … … 43%  327M 0s
3100K … … … … … 44%  290M 0s
3150K … … … … … 45%  340M 0s
3200K … … … … … 45%  294M 0s
3250K … … … … … 46%  353M 0s
3300K … … … … … 47%  334M 0s
3350K … … … … … 47%  322M 0s
3400K … … … … … 48%  335M 0s
3450K … … … … … 49%  257M 0s
3500K … … … … … 50%  291M 0s
3550K … … … … … 50%  315M 0s
3600K … … … … … 51%  332M 0s
3650K … … … … … 52%  345M 0s
3700K … … … … … 52%  368M 0s
3750K … … … … … 53%  368M 0s
3800K … … … … … 54%  328M 0s
3850K … … … … … 54%  369M 0s
3900K … … … … … 55%  401M 0s
3950K … … … … … 56%  286M 0s
4000K … … … … … 57%  321M 0s
4050K … … … … … 57%  311M 0s
4100K … … … … … 58%  324M 0s
4150K … … … … … 59%  284M 0s
4200K … … … … … 59%  334M 0s
4250K … … … … … 60%  311M 0s
4300K … … … … … 61%  322M 0s
4350K … … … … … 62%  306M 0s
4400K … … … … … 62%  302M 0s
4450K … … … … … 63%  302M 0s
4500K … … … … … 64%  331M 0s
4550K … … … … … 64%  329M 0s
```

```
4600K … … … … … 65%  318M 0s
4650K … … … … … 66%  325M 0s
4700K … … … … … 66%  328M 0s
4750K … … … … … 67%  276M 0s
4800K … … … … … 68%  296M 0s
4850K … … … … … 69%  292M 0s
4900K … … … … … 69%  301M 0s
4950K … … … … … 70%  301M 0s
5000K … … … … … 71%  304M 0s
5050K … … … … … 71%  308M 0s
5100K … … … … … 72%  287M 0s
5150K … … … … … 73%  125M 0s
5200K … … … … … 74%  179M 0s
5250K … … … … … 74%  287M 0s
5300K … … … … … 75%  292M 0s
5350K … … … … … 76%  291M 0s
5400K … … … … … 76%  322M 0s
5450K … … … … … 77%  296M 0s
5500K … … … … … 78%  325M 0s
5550K … … … … … 78%  266M 0s
5600K … … … … … 79% 61.9M 0s
5650K … … … … … 80%  237M 0s
5700K … … … … … 81%  194M 0s
5750K … … … … … 81%  254M 0s
5800K … … … … … 82%  289M 0s
5850K … … … … … 83%  213M 0s
5900K … … … … … 83%  225M 0s
5950K … … … … … 84%  293M 0s
6000K … … … … … 85%  335M 0s
6050K … … … … … 86%  292M 0s
6100K … … … … … 86%  281M 0s
6150K … … … … … 87%  334M 0s
6200K … … … … … 88%  392M 0s
6250K … … … … … 88%  325M 0s
6300K … … … … … 89%  502M 0s
6350K … … … … … 90%  317M 0s
6400K … … … … … 90%  402M 0s
6450K … … … … … 91%  328M 0s
6500K … … … … … 92%  273M 0s
6550K … … … … … 93%  350M 0s
6600K … … … … … 93%  343M 0s
6650K … … … … … 94%  283M 0s
6700K … … … … … 95%  308M 0s
6750K … … … … … 95%  307M 0s
6800K … … … … … 96%  333M 0s
6850K … … … … … 97% 1.12M 0s
6900K … … … … … 98%  238M 0s
6950K … … … … … 98%  445M 0s
```

```
7000K … … … … … 99%  358M 0s
7050K … … … … .         100%  267M=0.1s

2024-06-09 07:54:12 (72.5 MB/s) - ‘/tmp/title.ratings.tsv.gz’ saved
[7261619/7261619]
```

[ ]:
```
%sh

gunzip -f /tmp/name.basics.tsv.gz
gunzip -f /tmp/title.akas.tsv.gz
gunzip -f /tmp/title.basics.tsv.gz
gunzip -f /tmp/title.crew.tsv.gz
gunzip -f /tmp/title.episode.tsv.gz
gunzip -f /tmp/title.principals.tsv.gz
gunzip -f /tmp/title.ratings.tsv.gz
```

[ ]:
```
dbutils.fs.mkdirs("/mnt/data/test")
```

```
Out[3]: True
```

[ ]:
```
dbutils.fs.mv("file:/tmp/name.basics.tsv", "dbfs:/mnt/data/test")
dbutils.fs.mv("file:/tmp/title.akas.tsv", "dbfs:/mnt/data/test")
dbutils.fs.mv("file:/tmp/title.basics.tsv", "dbfs:/mnt/data/test")
dbutils.fs.mv("file:/tmp/title.crew.tsv", "dbfs:/mnt/data/test")
dbutils.fs.mv("file:/tmp/title.episode.tsv", "dbfs:/mnt/data/test")
dbutils.fs.mv("file:/tmp/title.principals.tsv", "dbfs:/mnt/data/test")
dbutils.fs.mv("file:/tmp/title.ratings.tsv", "dbfs:/mnt/data/test")
```

```
Out[4]: True
```

[ ]:
```
name_basics_df = spark.read.option('inferSchema', 'true').option('header',␣
↪'true').option('sep', '\t').csv('dbfs:/mnt/data/test/name.basics.tsv')
title_akas_df = spark.read.option('inferSchema', 'true').option('header',␣
↪'true').option('sep', '\t').csv('dbfs:/mnt/data/test/title.akas.tsv')
title_basics_df = spark.read.option('inferSchema', 'true').option('header',␣
↪'true').option('sep', '\t').csv('dbfs:/mnt/data/test/title.basics.tsv')
title_crew_df = spark.read.option('inferSchema', 'true').option('header',␣
↪'true').option('sep', '\t').csv('dbfs:/mnt/data/test/title.crew.tsv')
title_episode_df = spark.read.option('inferSchema', 'true').option('header',␣
↪'true').option('sep', '\t').csv('dbfs:/mnt/data/test/title.episode.tsv')
title_principals_df = spark.read.option('inferSchema', 'true').option('header',␣
↪'true').option('sep', '\t').csv('dbfs:/mnt/data/test/title.principals.tsv')
title_ratings_df = spark.read.option('inferSchema', 'true').option('header',␣
↪'true').option('sep', '\t').csv('dbfs:/mnt/data/test/title.ratings.tsv')
```

1.1.2 Network Inference, Let's build a network

In the following questions you will look to summarise the data and build a network. We want to examine a network that abstracts how actors and actress are related through their co-participation in movies. To that end perform the following steps:

Q1 Create a DataFrame that combines **all the information** on each of the titles (i.e., movies, tv-shows, etc …) and **all of the information** the participants in those movies (i.e., actors, directors, etc …), make sure the actual names of the movies and participants are included. It may be worth reviewing the following questions to see how this dataframe will be used.

How many rows does your dataframe have?

[ ]:
```
from pyspark.sql.functions import col
```

[ ]:
```
# Aliasing DataFrames for clarity
titles_df = title_basics_df.alias("titles")
principals_df = title_principals_df.alias("principals")
names_df = name_basics_df.alias("names")
ratings_df = title_ratings_df.alias("ratings")

# Join titles with principals on tconst
titles_with_principals = titles_df.join(
    principals_df,
    col("titles.tconst") == col("principals.tconst"),
    "left"
).select(
    col("titles.tconst").alias("title_tconst"),
    col("titles.*"),
    col("principals.*")
)

# Join the result with names on nconst
titles_principals_names = titles_with_principals.join(
    names_df,
    col("principals.nconst") == col("names.nconst"),
    "left_outer"
).select(
    col("title_tconst"),
    col("titles.*"),
    col("principals.nconst").alias("principal_nconst"),
    col("principals.*"),
    col("names.*")
)

# Join with ratings on tconst
full_title_info = titles_principals_names.join(
    ratings_df,
    col("title_tconst") == col("ratings.tconst"),
```

```
    "left"
).select(
    col("title_tconst"),
    col("titles.*"),
    col("principal_nconst"),
    col("names.nconst").alias("name_nconst"),
    col("names.*"),
    col("ratings.*"),
    col("principals.*")
)

# Final selection to avoid duplicate columns
final_df = full_title_info.select(
    col("title_tconst").alias("tconst"),
    col("principal_nconst"),
    col("name_nconst"),
    col("ratings.averageRating"),
    col("ratings.numVotes"),
    col("names.primaryName"),
    col("names.birthYear"),
    col("names.deathYear"),
    col("names.primaryProfession"),
    col("titles.primaryTitle"),
    col("titles.originalTitle"),
    col("titles.isAdult"),
    col("titles.startYear"),
    col("titles.endYear"),
    col("titles.runtimeMinutes"),
    col("titles.genres"),
    col("principals.category"),
    col("principals.job"),
    col("principals.characters")
)

# Show some of the data to verify correctness
final_df.show(5)
```

```
+---------+----------------+-----------+-------------+--------+----------------
-+---------+---------+--------------------+--------------------+----------------
----+-------+---------+-------+--------------+---------------+--------+--------
+------------------+
|   tconst|principal_nconst|name_nconst|averageRating|numVotes|
primaryName|birthYear|deathYear|   primaryProfession|        primaryTitle|
originalTitle|isAdult|startYear|endYear|runtimeMinutes|         genres|category|
job|       characters|
+---------+----------------+-----------+-------------+--------+----------------
-+---------+---------+--------------------+--------------------+----------------
----+-------+---------+-------+--------------+---------------+--------+--------
```

```
+------------------+
|tt0000658|       nm0169871|  nm0169871|          6.4|     292|        Émile
Cohl|     1857|      1938|director,animatio…|The Puppet's Nigh…|Le cauchemar
de F…|      0|      1908|     \N|             2|Animation,Short|director|
\N|               \N|
|tt0000839|       nm0294276|  nm0294276|         null|    null|      Theo
Frenkel|     1871|      1956|director,actor,wr…|  The Curse of Money|  The
Curse of Money|      0|      1909|     \N|            \N|
Drama,Short|director|director|               \N|
|tt0000839|       nm0378408|  nm0378408|         null|    null| Cecil M.
Hepworth|     1873|      1953|producer,cinemato…|  The Curse of Money|  The
Curse of Money|      0|      1909|     \N|            \N|
Drama,Short|producer|producer|               \N|
|tt0001170|       nm1400009|  nm1400009|         null|    null|William A.
Russell|     1878|      1914|                actor|A Cowboy's Vindic…|A Cowboy's
Vindic…|      0|      1910|     \N|             \N|  Short,Western|   actor|
\N|               \N|
|tt0001170|       nm0355582|  nm0355582|         null|    null|     Franklyn
Hall|     1886|        \N|actor,writer,dire…|A Cowboy's Vindic…|A Cowboy's
Vindic…|      0|      1910|     \N|             \N|  Short,Western|   actor|
\N|["Will Morrison"]|
+---------+----------------+-----------+-------------+--------+-----------------
-+---------+---------+--------------------+--------------------+----------------
----+-------+---------+-------+--------------+---------------+--------+--------
+------------------+
only showing top 5 rows
```

[ ]:
```python
# Q1 FINAL ANSWER

# Show the number of rows
print(f"My dataframe has {final_df.count()} rows.")
```

```
My dataframe has 87282924 rows.
```

Q2 Create a new DataFrame based on the previous step, with the following removed: 1. Any participant that is not an actor or actress (as measured by the category column); 1. All adult movies; 1. All dead actors or actresses; 1. All actors or actresses born before 1920 or with no date of birth listed; 1. All titles that are not of the type movie.

How many rows does your dataframe have?

[ ]:
```python
# Filter for actors and actresses
filtered_df = final_df.filter(col("category").isin(["actor", "actress"]))

# Filter out adult movies
filtered_df = filtered_df.filter(col("isAdult") == 0)

# Filter out dead actors/actresses
```

```
filtered_df = filtered_df.filter(col("deathYear")=="\\N")

# Filter out actors/actresses born before 1920 or with no date of birth listed
filtered_df = filtered_df.filter((col("birthYear") >= 1920) & (col("birthYear")␣
↪!= "\\N"))

# Filter for movie titles only
filtered_df = filtered_df.filter(col("titleType") == "movie")
```

[ ]:
```
# Q2 FINAL ANSWER

# Show the number of rows
print(f"The filtered dataframe has {filtered_df.count()} rows.")
```

```
The filtered dataframe has 930698 rows.
```

Q3 Convert the above Dataframe to an RDD. Use map and reduce to create a paired RDD which counts how many movies each actor / actress appears in.

Display names of the top 10 actors/actresses according to the number of movies in which they appeared. Be careful to deal with different actors / actresses with the same name, these could be different people.

[ ]:
```
# Convert DataFrame to RDD
filtered_rdd = filtered_df.rdd
```

[ ]:
```
# Create a paired RDD (actor_id, 1) and reduce by key to count the movies
actor_movie_counts_rdd = filtered_rdd.map(lambda row: (row['name_nconst'], 1)).
↪reduceByKey(lambda a, b: a + b)
```

[ ]:
```
# Convert the filtered DataFrame to RDD to get actor names
actor_names_rdd = filtered_df.select("name_nconst", "primaryName").distinct().
↪rdd.map(lambda row: (row.name_nconst, row.primaryName))
```

[ ]:
```
# Join actor_movie_rdd with actor_names_rdd
actor_movie_count_with_names_rdd = actor_movie_counts_rdd.join(actor_names_rdd)
```

[ ]:
```
# Sort by movie count in descending order and take the top 10
top_10_actors_rdd = actor_movie_count_with_names_rdd.sortBy(lambda x: x[1][0],␣
↪ascending=False).take(10)
```

[ ]:
```
# Q3 FINAL ANSWER

# Display names of the top 10 actors/actresses
for actor, (count, name) in top_10_actors_rdd:
    print(f"Actor: {name}, Movie Count: {count}")
```

```
Actor: Brahmanandam, Movie Count: 1130
Actor: Jagathy Sreekumar, Movie Count: 659
```

```
Actor: Shakti Kapoor, Movie Count: 600
Actor: Eric Roberts, Movie Count: 492
Actor: Aruna Irani, Movie Count: 467
Actor: Nassar, Movie Count: 440
Actor: Mammootty, Movie Count: 437
Actor: Helen, Movie Count: 433
Actor: Tanikella Bharani, Movie Count: 412
Actor: Anupam Kher, Movie Count: 409
```

Q4 Start with the dataframe from Q2. Generate a DataFrame that lists all links of your network. Here we shall consider that a link connects a pair of actors/actresses if they participated in at least one movie together (actors / actresses should be represented by their unique ID's). For every link we then need anytime a pair of actors were together in a movie as a link in each direction (A -> B and B -> A). However links should be distinct we do not need duplicates when two actors worked together in several movies.

Display a DataFrame with the first 10 edges.

[ ]:
```python
from pyspark.sql.functions import col, collect_list, explode

# Filtered DataFrame from Q2
df_q2 = filtered_df
```

[ ]:
```python
# Select necessary columns
actor_movie_df = filtered_df.select("tconst", "name_nconst")

# Group by movie to get the list of actors in each movie
movie_actors_df = actor_movie_df.groupBy("tconst").
 ↪agg(collect_list("name_nconst").alias("actors"))

# Explode the list of actors to create actor pairs
# First create a new DataFrame where each row contains the movie ID and each␣
 ↪pair of actors
def create_pairs(actors):
    pairs = []
    for i in range(len(actors)):
        for j in range(i + 1, len(actors)):
            pairs.append((actors[i], actors[j]))
            pairs.append((actors[j], actors[i]))
    return pairs

# Register the function as a UDF
from pyspark.sql.types import ArrayType, StructType, StructField, StringType
from pyspark.sql.functions import udf

pair_schema = ArrayType(StructType([
    StructField("src", StringType(), False),
    StructField("dst", StringType(), False)
```

```
]))

create_pairs_udf = udf(create_pairs, pair_schema)

# Apply the UDF to create pairs
actor_pairs_df = movie_actors_df.withColumn("pairs",␣
↪explode(create_pairs_udf(col("actors"))))

# Separate src and dst
bidirectional_pairs_df = actor_pairs_df.select(col("pairs.src").alias("src"),␣
↪col("pairs.dst").alias("dst")).distinct()
```

[ ]:
```
# Q4 FINAL ANSWER

# Show the first 10 edges
bidirectional_pairs_df.show(10)
```

```
+---------+---------+
|      src|      dst|
+---------+---------+
|nm0180228|nm0639684|
|nm0172237|nm0149883|
|nm0231942|nm0639684|
|nm0013690|nm0013672|
|nm0180228|nm0231942|
|nm0639684|nm0231942|
|nm0231942|nm0180228|
|nm0571763|nm0177320|
|nm0639684|nm0180228|
|nm0013672|nm0013690|
+---------+---------+
only showing top 10 rows
```

Q5 Compute the page rank of each actor. This can be done using GraphFrames or by using RDDs and the iterative implementation of the PageRank algorithm. Do not take more than 5 iterations and use reset probility = 0.1.

List the top 10 actors / actresses by pagerank.

[ ]:
```
from graphframes import GraphFrame
from pyspark.sql.functions import col
```

[ ]:
```
# Create vertices DataFrame
vertices_df = filtered_df.select(col("name_nconst").alias("id")).distinct()
```

[ ]:
```
# Create edges DataFrame from bidirectional_pairs_df
edges_df = bidirectional_pairs_df.select(col("src").alias("src"), col("dst").
↪alias("dst"))
```

```
[ ]: # Initialize GraphFrame
g = GraphFrame(vertices_df, edges_df)
```

```
/databricks/spark/python/pyspark/sql/dataframe.py:170: UserWarning:
DataFrame.sql_ctx is an internal property, and will be removed in future
releases. Use DataFrame.sparkSession instead.
  warnings.warn(
```

```
[ ]: # Compute PageRank
pagerank_results = g.pageRank(resetProbability=0.1, maxIter=5)
```

```
/databricks/spark/python/pyspark/sql/dataframe.py:149: UserWarning: DataFrame
constructor is internal. Do not directly use it.
  warnings.warn("DataFrame constructor is internal. Do not directly use it.")
```

```
[ ]: # Join PageRank results with filtered_df to get actor names
pagerank_with_names = pagerank_results.vertices.join(filtered_df,␣
 ↪pagerank_results.vertices.id == filtered_df.name_nconst, "inner") \
                                                .select("id", "pagerank",␣
 ↪"primaryName") \
                                                .distinct()
```

```
[ ]: # Q5 FINAL ANSWER

# Display top 10 actors by PageRank
top_10_actors = pagerank_with_names.orderBy(col("pagerank").desc()).
 ↪select("id", "primaryName", "pagerank").limit(10)
top_10_actors.show()
```

```
+---------+----------------+------------------+
|       id|     primaryName|          pagerank|
+---------+----------------+------------------+
|nm0000616|    Eric Roberts|62.777102793531824|
|nm0000514|  Michael Madsen| 33.85372574721415|
|nm0001803|     Danny Trejo|26.530499578166594|
|nm0202966|     Keith David|24.796193603716564|
|nm0001595|    Michael Paré|24.302493820642567|
|nm0261724|     Joe Estevez|23.867127734262173|
|nm0726223|  Richard Riehle| 22.93143628629502|
|nm0000532|Malcolm McDowell| 22.83158457241578|
|nm0442207|   Lloyd Kaufman| 22.66740153494526|
|nm0000448| Lance Henriksen|22.253312809786298|
+---------+----------------+------------------+
```

Q6: Create an RDD with the number of outDegrees for each actor. Display the top 10 by outdegrees.

```
[ ]: # Convert DataFrame to RDD
pairs_rdd = bidirectional_pairs_df.rdd
```

```
[ ]: # Map to paired RDD with (actor_id, 1) for each out-degree connection
outdegrees_rdd = pairs_rdd.map(lambda row: (row.src, 1)).reduceByKey(lambda a,␣
↪b: a + b)
```

```
[ ]: # Convert the filtered DataFrame to RDD to get actor names
actor_names_rdd = filtered_df.select("name_nconst", "primaryName").distinct().
↪rdd.map(lambda row: (row.name_nconst, row.primaryName))
```

```
[ ]: # Join outDegrees with actor names
outdegrees_with_names_rdd = outdegrees_rdd.join(actor_names_rdd)
```

```
[ ]: # Sort by outDegrees in descending order
top_10_outdegrees = outdegrees_with_names_rdd.sortBy(lambda x: x[1][0],␣
↪ascending=False).take(10)
```

```
[ ]: # Q6 FINAL ANSWER

# Display the top 10 actors by outDegrees with names
for actor, (outdegree, name) in top_10_outdegrees:
    print(f"Actor: {name}, OutDegrees: {outdegree}")
```

```
Actor: Eric Roberts, OutDegrees: 1338
Actor: Michael Madsen, OutDegrees: 842
Actor: Anupam Kher, OutDegrees: 761
Actor: Keith David, OutDegrees: 708
Actor: Renji Ishibashi, OutDegrees: 704
Actor: Nassar, OutDegrees: 689
Actor: Gérard Depardieu, OutDegrees: 678
Actor: Danny Trejo, OutDegrees: 664
Actor: Akira Emoto, OutDegrees: 659
Actor: Prakash Raj, OutDegrees: 649
```

1.1.3 Let's play Kevin's own game

Q7 Start with the graphframe / dataframe you developed in the previous questions. Using Spark GraphFrame and/or Spark Core library perform the following steps:

1. Identify the id of Kevin Bacon, there are two actors named 'Kevin Bacon', we will use the one with the highest degree, that is, the one that participated in most titles;
2. Estimate the shortest path between every actor in the database actors and Kevin Bacon, keep a dataframe with this information as you will need it later;
3. Summarise the data, that is, count the number of actors at each number of degress from kevin bacon (you will need to deal with actors unconnected to kevin bacon, if not connected to Kevin Bacon given these actors / actresses a score/degree of 20).

[ ]:
```python
from graphframes import GraphFrame
from pyspark.sql.functions import col, explode, when, lit

# Identify the ID of Kevin Bacon with the highest degree
filtered_df.filter(col("primaryName") == "Kevin Bacon").show()
```

```
+----------+----------------+-----------+-------------+--------+-----------+----
-----+---------+--------------------+--------------------+--------------------+-
------+---------+-------+--------------+--------------------+--------+---+------
--------------+
|    tconst|principal_nconst|name_nconst|averageRating|numVotes|primaryName|birt
hYear|deathYear|   primaryProfession|        primaryTitle|
originalTitle|isAdult|startYear|endYear|runtimeMinutes|
genres|category|job|          characters|
+----------+----------------+-----------+-------------+--------+-----------+----
-----+---------+--------------------+--------------------+--------------------+-
------+---------+-------+--------------+--------------------+--------+---+------
--------------+
| tt0373450|       nm0000102|  nm0000102|          6.4|   18805|Kevin Bacon|
1958|       \N|actor,producer,di…|Where the Truth Lies|Where the Truth Lies|
0|     2005|     \N|           107|Crime,Mystery,Thr…|   actor| \N|
["Lanny"]|
| tt0119896|       nm0000102|  nm0000102|          5.5|   21522|Kevin Bacon|
1958|       \N|actor,producer,di…|     Picture Perfect|     Picture Perfect|
0|     1997|     \N|           101|Comedy,Drama,Romance|   actor| \N|
["Sam"]|
| tt0361127|       nm0000102|  nm0000102|          7.1|   35445|Kevin Bacon|
1958|       \N|actor,producer,di…|        The Woodsman|        The Woodsman|
0|     2004|     \N|            87|               Drama|   actor| \N|
["Walter"]|
|tt14502344|       nm0000102|  nm0000102|          4.0|   11597|Kevin Bacon|
1958|       \N|actor,producer,di…|           They/Them|           They/Them|
0|     2022|     \N|           104|Drama,Horror,Mystery|   actor| \N|
["Owen"]|
| tt0164181|       nm0000102|  nm0000102|          6.9|   87830|Kevin Bacon|
1958|       \N|actor,producer,di…|      Stir of Echoes|      Stir of Echoes|
0|     1999|     \N|            99|Horror,Mystery,Th…|   actor| \N|
["Tom"]|
| tt6317762|       nm0000102|  nm0000102|          5.5|    1369|Kevin Bacon|
1958|       \N|actor,producer,di…|        Space Oddity|        Space Oddity|
0|     2022|     \N|            92|Comedy,Romance,Sc…|   actor| \N| ["Jeff
McAllister"]|
| tt0093403|       nm0000102|  nm0000102|          6.3|     340|Kevin Bacon|
1958|       \N|actor,producer,di…|           Lemon Sky|           Lemon Sky|
0|     1988|     \N|           106|               Drama|   actor| \N|
["Alan"]|
| tt0790736|       nm0000102|  nm0000102|          5.6|  144006|Kevin Bacon|
1958|       \N|actor,producer,di…|            R.I.P.D.|            R.I.P.D.|
```

```
0|     2013|     \N|            96|Action,Adventure,…|   actor| \N|
["Hayes"]|
|tt13075730|       nm0000102|  nm0000102|          4.1|    1864|Kevin Bacon|
1958|       \N|actor,producer,di…|             One Way|             One Way|
0|     2022|     \N|            95|     Action,Thriller|   actor| \N|["Fred
Sullivan S…|
| tt1512235|       nm0000102|  nm0000102|          6.7|   83837|Kevin Bacon|
1958|       \N|actor,producer,di…|               Super|               Super|
0|     2010|     \N|            96| Action,Comedy,Crime|   actor| \N|
["Jacques"]|
| tt1578882|       nm0000102|  nm0000102|          5.0|   11115|Kevin Bacon|
1958|       \N|actor,producer,di…|      Elephant White|      Elephant White|
0|     2011|     \N|            91|Action,Crime,Thri…|   actor| \N|
["Jimmy"]|
| tt8201852|       nm0000102|  nm0000102|          5.4|   26106|Kevin Bacon|
1958|       \N|actor,producer,di…|You Should Have Left|You Should Have Left|
0|     2020|     \N|            93|Horror,Mystery,Th…|   actor| \N|
["Theo"]|
| tt8201852|       nm0000102|  nm0000102|          5.4|   26106|Kevin Bacon|
1958|       \N|actor,producer,di…|You Should Have Left|You Should Have Left|
0|     2020|     \N|            93|Horror,Mystery,Th…|   actor| \N|
["Stetler"]|
| tt1270798|       nm0000102|  nm0000102|          7.7|  726015|Kevin Bacon|
1958|       \N|actor,producer,di…|  X-Men: First Class|      X: First Class|
0|     2011|     \N|           131|       Action,Sci-Fi|   actor| \N|
["Sebastian Shaw"]|
| tt0822849|       nm0000102|  nm0000102|          6.7|    4277|Kevin Bacon|
1958|       \N|actor,producer,di…|        Rails & Ties|        Rails & Ties|
0|     2007|     \N|           101|               Drama|   actor| \N|
["Tom Stark"]|
| tt0080761|       nm0000102|  nm0000102|          6.4|  158162|Kevin Bacon|
1958|       \N|actor,producer,di…|     Friday the 13th|     Friday the 13th|
0|     1980|     \N|            95|Horror,Mystery,Th…|   actor| \N|
["Jack"]|
| tt0094318|       nm0000102|  nm0000102|          6.2|    5946|Kevin Bacon|
1958|       \N|actor,producer,di…|  White Water Summer|  White Water Summer|
0|     1987|     \N|            90|     Adventure,Drama|   actor| \N|
["Vic"]|
| tt0120303|       nm0000102|  nm0000102|          6.2|    2432|Kevin Bacon|
1958|       \N|actor,producer,di…|Telling Lies in A…|Telling Lies in A…|
0|     1997|     \N|           101|         Drama,Music|   actor| \N|
["Billy Magic"]|
| tt0096094|       nm0000102|  nm0000102|          5.9|   13936|Kevin Bacon|
1958|       \N|actor,producer,di…| She's Having a Baby| She's Having a Baby|
0|     1988|     \N|           106|Comedy,Drama,Romance|   actor| \N|     ["Jake
Briggs"]|
| tt0120890|       nm0000102|  nm0000102|          6.6|  131436|Kevin Bacon|
1958|       \N|actor,producer,di…|         Wild Things|         Wild Things|
```

```
0|     1998|     \N|           108| Crime,Drama,Mystery|   actor| \N|    ["Ray
Duquette"]|
+----------+----------------+-----------+-------------+--------+-----------+----
-----+---------+--------------------+--------------------+--------------------+-
------+---------+-------+--------------+--------------------+--------+---+------
--------------+
only showing top 20 rows
```

[ ]:
```python
graph = GraphFrame(vertices_df, edges_df)

# Kevin Bacon's ID
kevin_bacon_id = "nm0000102"

# Estimate shortest paths from Kevin Bacon to all other actors
shortest_paths = graph.shortestPaths(landmarks=[kevin_bacon_id])

# Extract and process the shortest path distances
shortest_paths = shortest_paths.select("id", col("distances").
 ↪getItem(kevin_bacon_id).alias("distance"))

# Handle unconnected actors by assigning a distance of 20
shortest_paths = shortest_paths.withColumn("distance", col("distance").
 ↪cast("int"))
shortest_paths = shortest_paths.na.fill(20, subset=["distance"])

# Cache the shortest paths DataFrame as it will be used later
shortest_paths.cache()
```

```
Out[34]: DataFrame[id: string, distance: int]
```

[ ]:
```python
# Q7 FINAL ANSWER
degree_summary = shortest_paths.groupBy("distance").count().orderBy("distance")

# Show the degree summary
degree_summary.show(15)
```

```
+--------+-----+
|distance|count|
+--------+-----+
|       0|    1|
|       1|  354|
|       2|14170|
|       3|58560|
|       4|42462|
|       5| 4842|
|       6|  510|
|       7|   56|
```

```
|       8|   20|
|       9|    3|
|      20|15330|
+--------+-----+
```

1.1.4 Exploring the data with RDD's

Using RDDs and (not dataframes) answer the following questions (if you loaded your data into spark in a dataframe you can convert to an RDD of rows easily using `.rdd`):

Q8 Movies can have multiple genres. Considering only titles of the type 'movie' what is the combination of genres that is the most popluar (as measured by number of reviews). Hint: paired RDD's will be useful.

```
[ ]: from itertools import combinations
     from pyspark.sql.functions import col

     # Convert DataFrame to RDD and filter out movies with null numVotes
     movies_rdd = final_df.filter((col("titleType") == "movie") & (col("numVotes").
      ↪isNotNull())).select("genres", "numVotes").rdd
```

```
[ ]: # Function to create genre combinations
     def create_genre_combinations(row):
         genres = row.genres.split(',')
         combinations = []
         for i in range(len(genres)):
             for j in range(i + 1, len(genres)):
                 combinations.append((f"{genres[i]},{genres[j]}", row.numVotes))
         return combinations
```

```
[ ]: # Create paired RDD with genre combinations and review counts
     genre_combinations_rdd = movies_rdd.flatMap(create_genre_combinations)
```

```
[ ]: # Sum the number of reviews for each genre combination
     genre_combination_counts = genre_combinations_rdd.reduceByKey(lambda a, b: a +␣
      ↪b)
```

```
[ ]: # Find the most popular genre combination
     most_popular_genre_combination = genre_combination_counts.max(lambda x: x[1])
```

```
[ ]: # Q8 FINAL ANSWER

     # Display the most popular genre combination and the number of reviews
     print(f"Most Popular Genre Combination: {most_popular_genre_combination[0]}.␣
      ↪Number of Reviews: {most_popular_genre_combination[1]}")
```

```
Most Popular Genre Combination: Action,Adventure. Number of Reviews: 4404499959
```

Q9 Movies can have multiple genres. Considering only titles of the type 'movie', and movies with more than 400 ratings, what is the combination of genres that has the highest **average movie rating** (you can average the movie rating for each movie in that genre combination). Hint: paired RDD's will be useful.

[ ]:
```python
from itertools import combinations
from pyspark.sql import Row

# Convert DataFrames to RDDs
title_basics_rdd = title_basics_df.rdd
title_ratings_rdd = title_ratings_df.rdd
```

[ ]:
```python
# Filter for movies only and movies with more than 400 ratings
movies_rdd = title_basics_rdd.filter(lambda row: row['titleType'] == 'movie')
highly_rated_movies_rdd = title_ratings_rdd.filter(lambda row: row['numVotes']␣
↪> 400)
```

[ ]:
```python
# Join RDDs on tconst to get movie ratings
movie_ratings_rdd = movies_rdd.map(lambda row: (row['tconst'], row)) \
                              .join(highly_rated_movies_rdd.map(lambda row:␣
↪(row['tconst'], (row['averageRating'], row['numVotes'])))) \
                              .map(lambda x: (x[1][0]['genres'], x[1][1]))
```

[ ]:
```python
# Function to create genre combinations and map to ratings
def genre_combinations(row):
    genres = row[0]
    rating = row[1][0]
    if genres:
        genre_list = genres.split(',')
        for i in range(1, len(genre_list) + 1):
            for combo in combinations(genre_list, i):
                yield (','.join(sorted(combo)), (rating, 1))
```

[ ]:
```python
# Create genre combinations and map to ratings
genre_combinations_rdd = movie_ratings_rdd.flatMap(genre_combinations)
```

[ ]:
```python
# Reduce by key to calculate the sum of ratings and the count for each genre␣
↪combination
genre_ratings_count_rdd = genre_combinations_rdd.reduceByKey(lambda a, b: (a[0]␣
↪+ b[0], a[1] + b[1]))
```

[ ]:
```python
# Calculate the average rating for each genre combination
genre_avg_ratings_rdd = genre_ratings_count_rdd.mapValues(lambda v: v[0] / v[1])
```

[ ]:
```python
# Find the genre combination with the highest average rating
highest_avg_rating_genre_combo = genre_avg_ratings_rdd.sortBy(lambda x: x[1],␣
↪ascending=False).take(1)
```

```
[ ]: # Q9 FINAL ANSWER

# Show the highest average rating genre combination
for combo, avg_rating in highest_avg_rating_genre_combo:
    print(f"Highest Average Rating Genre Combination: {combo} with an average␣
 ↪rating of {avg_rating}")
```

```
Highest Average Rating Genre Combination: Action,Documentary,Mystery with an
average rating of 8.3
```

Q10 Movies can have multiple genres. What is **the individual genre** which is the most popular as meaured by number of votes. Votes for multiple genres count towards each genre listed. Hint: flatmap and pairedRDD's will be useful here.

```
[ ]: # Convert DataFrames to RDDs
title_basics_rdd = title_basics_df.rdd
title_ratings_rdd = title_ratings_df.rdd
```

```
[ ]: # Filter for movies only
movies_rdd = title_basics_rdd.filter(lambda row: row['titleType'] == 'movie')
```

```
[ ]: # Join RDDs on tconst to get number of votes
movie_votes_rdd = movies_rdd.map(lambda row: (row['tconst'], row)) \
                             .join(title_ratings_rdd.map(lambda row:␣
 ↪(row['tconst'], row['numVotes']))) \
                             .map(lambda x: (x[1][0]['genres'], x[1][1]))
```

```
[ ]: # Function to create individual genre records and map to votes
def explode_genres(row):
    genres = row[0]
    numVotes = row[1]
    if genres:
        genre_list = genres.split(',')
        for genre in genre_list:
            yield (genre, numVotes)
```

```
[ ]: # Create individual genre records and map to votes
genres_votes_rdd = movie_votes_rdd.flatMap(explode_genres)
```

```
[ ]: # Reduce by key to sum the number of votes for each genre
genre_votes_rdd = genres_votes_rdd.reduceByKey(lambda a, b: a + b)
```

```
[ ]: # Find the most popular genre by votes
most_popular_genre = genre_votes_rdd.sortBy(lambda x: x[1], ascending=False).
 ↪take(1)
```

```
[ ]: # Q10 FINAL ANSWER
```

```
# Show the most popular genre
for genre, votes in most_popular_genre:
    print(f"Most Popular Genre: {genre} with {votes} votes")
```

```
Most Popular Genre: Drama with 572360704 votes
```

1.2 Engineering the perfect cast

We have created a number of potential features for predicting the rating of a movie based on its cast. Use sparkML to build a simple linear model to predict the rating of a movie based on the following features:

1. The total number of movies in which the actors / actresses have acted (based on Q3)
2. The average pagerank of the cast in each movie (based on Q5)
3. The average outDegree of the cast in each movie (based on Q6)
4. The average value for for the cast of degrees of Kevin Bacon (based on Q7).

You will need to create a dataframe with the required features and label. Use a pipeline to create the vectors required by sparkML and apply the model. Remember to split your dataset, leave 30% of the data for testing, when splitting your data use the option seed=0.

Q11 Provide the coefficients of the regression and the accuracy of your model on that test dataset according to RSME.

[ ]:
```
from pyspark.sql.functions import col, avg
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
```

[ ]:
```
# Extract features
# Q3: Total number of movies each actor has acted in
actor_movie_counts_df = actor_movie_counts_rdd.toDF(["name_nconst",␣
 ↪"total_movies"])
```

[ ]:
```
# Q5: Average PageRank of the cast in each movie
pagerank_df = pagerank_results.vertices.select(col("id").alias("name_nconst"),␣
 ↪col("pagerank"))
```

[ ]:
```
# Q6: Average outDegree of the cast in each movie
# Extracting the first element of the struct which contains the numeric value
outdegrees_df = outdegrees_with_names_rdd.map(lambda row: (row[0], row[1][0],␣
 ↪row[1][1])).toDF(["name_nconst", "outdegree", "primaryName"])
```

[ ]:
```
# Q7: Average degrees of separation from Kevin Bacon
shortest_paths_df = shortest_paths.select(col("id").alias("name_nconst"),␣
 ↪col("distance").alias("distance_to_kevin_bacon"))
```

[ ]:
```
# Convert columns to numeric types where necessary
actor_movie_counts_df = actor_movie_counts_df.withColumn("total_movies",␣
↪col("total_movies").cast("double"))
pagerank_df = pagerank_df.withColumn("pagerank", col("pagerank").cast("double"))
outdegrees_df = outdegrees_df.withColumn("outdegree", col("outdegree").
↪cast("double"))
shortest_paths_df = shortest_paths_df.withColumn("distance_to_kevin_bacon",␣
↪col("distance_to_kevin_bacon").cast("double"))
```

[ ]:
```
# Combine all features for each movie
features_df = filtered_df.select("tconst", "name_nconst").distinct()
features_df = features_df.join(actor_movie_counts_df, "name_nconst", "left")
features_df = features_df.join(pagerank_df, "name_nconst", "left")
features_df = features_df.join(outdegrees_df, "name_nconst", "left")
features_df = features_df.join(shortest_paths_df, "name_nconst", "left")
```

[ ]:
```
# Aggregate features for each movie
aggregated_features_df = features_df.groupBy("tconst").agg(
    avg("total_movies").alias("avg_total_movies"),
    avg("pagerank").alias("avg_pagerank"),
    avg("outdegree").alias("avg_outdegree"),
    avg("distance_to_kevin_bacon").alias("avg_distance_to_kevin_bacon")
)
```

[ ]:
```
# Fill null values with 0
aggregated_features_df = aggregated_features_df.na.fill(0)
```

[ ]:
```
# Join with ratings to get the labels
ratings_df = final_df.select("tconst", "averageRating").distinct()
data = aggregated_features_df.join(ratings_df, "tconst")
```

[ ]:
```
# Check for and handle any remaining null values
data = data.na.fill(0)
```

[ ]:
```
# Prepare features and label
assembler = VectorAssembler(
    inputCols=["avg_total_movies", "avg_pagerank", "avg_outdegree",␣
↪"avg_distance_to_kevin_bacon"],
    outputCol="features"
)
```

[ ]:
```
# Split data into training and test sets
train_data, test_data = data.randomSplit([0.7, 0.3], seed=0)
```

[ ]:
```
# Define linear regression model
lr = LinearRegression(featuresCol="features", labelCol="averageRating")
```

```
[ ]: # Create pipeline
pipeline = Pipeline(stages=[assembler, lr])
```

```
[ ]: # Train the model
model = pipeline.fit(train_data)
```

```
[ ]: # Make predictions
predictions = model.transform(test_data)
```

```
[ ]: # Evaluate the model
evaluator = RegressionEvaluator(labelCol="averageRating",␣
 ↪predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
```

```
[ ]: # Get model coefficients
lr_model = model.stages[-1]
coefficients = lr_model.coefficients
intercept = lr_model.intercept
```

```
[ ]: # Q11 FINAL ANSWER

# Display the results
print(f"Coefficients: {coefficients}")
print(f"Intercept: {intercept}")
print(f"RMSE: {rmse}")
```

```
Coefficients: [-0.012659237327238589,-0.12336328797899417,0.012558636061121412,-
0.0525044127818446]
Intercept: 4.168227402209818
RMSE: 2.9026865848009455
```

Q12 What score would your model predict for the 1997 movie Titanic.

```
[ ]: # Filter for the title "Titanic" and year 1997
titanic_df = title_basics_df.filter((col("primaryTitle") == "Titanic") &␣
 ↪(col("startYear") == "1997"))

# Show the filtered result
titanic_df.show()

# Collect the ID for Titanic movie
titanic_id = titanic_df.select("tconst").collect()[0][0]
```

```
+---------+---------+------------+-------------+-------+---------+-------+------
--------+-----------------+
|   tconst|titleType|primaryTitle|originalTitle|isAdult|startYear|endYear|runtim
eMinutes|           genres|
+---------+---------+------------+-------------+-------+---------+-------+------
--------+-----------------+
```

```
|tt0120338|    movie|     Titanic|      Titanic|      0|     1997|     \N|
194|    Drama,Romance|
|tt0594950|tvEpisode|     Titanic|      Titanic|      0|     1997|     \N|
\N|Documentary,Short|
|tt5722820|tvEpisode|     Titanic|      Titanic|      0|     1997|     \N|
\N| Documentary,News|
+---------+---------+------------+------------+-------+---------+-------+------
--------+-----------------+
```

[ ]:
```python
# Filter the data for Titanic
titanic_features = data.filter(col("tconst") == titanic_id)
```

[ ]:
```python
# Step 3: Use the trained model to predict Titanic's rating
titanic_prediction = model.transform(titanic_features)
```

[ ]:
```python
# Q12 FINAL ANSWER

# Extract and print the predicted rating
predicted_rating = titanic_prediction.select("prediction").collect()[0][0]
print(f"Predicted rating for Titanic (1997): {round(predicted_rating,2)}")
```

```
Predicted rating for Titanic (1997): 5.88
```

Q13 Create dummy variables for each of the top 10 movie genres for Q10. These variable should have a value of 1 if the movie was rated with that genre and 0 otherwise. For example the 1997 movie Titanic should have a 1 in the dummy variable column for Romance, and a 1 in the dummy variable column for Drama, and 0's in all the other dummy variable columns.

Does adding these variable to the regression improve your results? What is the new RMSE and predicted rating for the 1997 movie Titanic.

[ ]:
```python
from pyspark.sql.functions import col, when, split, sum as _sum
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
```

[ ]:
```python
# Filter for movies only
movies_df = final_df.filter(col("titleType") == "movie")
```

[ ]:
```python
# Explode genres
movies_with_genres_df = movies_df.withColumn("genre",␣
 ↪explode(split(col("genres"), ",")))
```

[ ]:
```python
# Calculate total votes for each genre
genre_votes_df = movies_with_genres_df.groupBy("genre").agg(_sum("numVotes").
 ↪alias("total_votes"))
```

[ ]:
```python
# Sort by total votes and select top 10 genres
top_10_genres_df = genre_votes_df.orderBy(col("total_votes").desc()).limit(10)
top_10_genres = [row['genre'] for row in top_10_genres_df.collect()]
```

[ ]:
```python
# Display the top 10 genres
print("Top 10 genres by number of votes:")
for genre in top_10_genres:
    print(genre)
```

```
Top 10 genres by number of votes:
Drama
Action
Comedy
Adventure
Crime
Thriller
Sci-Fi
Romance
Mystery
Horror
```

[ ]:
```python
# Create dummy variables for the top 10 genres in final_df
for genre in top_10_genres:
    final_df = final_df.withColumn(f"genre_{genre}", when(col("genres").
 ↪contains(genre), 1).otherwise(0))
```

[ ]:
```python
# Q3: Total number of movies each actor has acted in
actor_movie_counts_df = actor_movie_counts_rdd.toDF(["name_nconst",␣
 ↪"total_movies"])
```

[ ]:
```python
# Q5: Average PageRank of the cast in each movie
pagerank_df = pagerank_results.vertices.select(col("id").alias("name_nconst"),␣
 ↪col("pagerank"))
```

[ ]:
```python
# Q6: Average outDegree of the cast in each movie
outdegrees_df = outdegrees_with_names_rdd.map(lambda row:␣
 ↪Row(name_nconst=row[0], outdegree=row[1][0])).toDF()
```

[ ]:
```python
# Q7: Average degrees of separation from Kevin Bacon
shortest_paths_df = shortest_paths.select(col("id").alias("name_nconst"),␣
 ↪col("distance").alias("distance_to_kevin_bacon"))
```

[ ]:
```python
# Combine all features for each movie
features_df = filtered_df.select("tconst", "name_nconst").distinct()
features_df = features_df.join(actor_movie_counts_df, "name_nconst", "left")
features_df = features_df.join(pagerank_df, "name_nconst", "left")
features_df = features_df.join(outdegrees_df, "name_nconst", "left")
features_df = features_df.join(shortest_paths_df, "name_nconst", "left")
```

[ ]:
```python
# Aggregate features for each movie
aggregated_features_df = features_df.groupBy("tconst").agg(
    avg("total_movies").alias("avg_total_movies"),
    avg("pagerank").alias("avg_pagerank"),
    avg("outdegree").alias("avg_outdegree"),
    avg("distance_to_kevin_bacon").alias("avg_distance_to_kevin_bacon")
)
```

[ ]:
```python
# Fill null values with 0
aggregated_features_df = aggregated_features_df.na.fill(0)
```

[ ]:
```python
# Join with ratings to get the labels
ratings_df = final_df.select("tconst", "averageRating").distinct()
data = aggregated_features_df.join(ratings_df, "tconst")
```

[ ]:
```python
# Join with dummy variables for genres
for genre in top_10_genres:
    genre_col = f"genre_{genre}"
    genre_df = final_df.select("tconst", genre_col).distinct()
    data = data.join(genre_df, "tconst", "left").na.fill(0)
```

[ ]:
```python
# Check for and handle any remaining null values
data = data.na.fill(0)
```

[ ]:
```python
from pyspark.ml.feature import VectorAssembler, StandardScaler

# Prepare feature set
assembler = VectorAssembler(
    inputCols=["avg_total_movies", "avg_pagerank", "avg_outdegree",␣
 ↪"avg_distance_to_kevin_bacon"] + [f"genre_{genre}" for genre in␣
 ↪top_10_genres],
    outputCol="features"
)
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures",␣
 ↪withStd=True, withMean=False)
```

[ ]:
```python
# Split data into training and test sets
train_data, test_data = data.randomSplit([0.7, 0.3], seed=0)
```

[ ]:
```python
# Define linear regression model
lr = LinearRegression(featuresCol="scaledFeatures", labelCol="averageRating")
```

[ ]:
```python
# Create pipeline
pipeline = Pipeline(stages=[assembler, scaler, lr])
```

[ ]:
```python
model = pipeline.fit(train_data)
```

[ ]:
```
# Make predictions
predictions = model.transform(test_data)
```

[ ]:
```
# Evaluate the model
evaluator = RegressionEvaluator(labelCol="averageRating",␣
 ↪predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
```

[ ]:
```
# Q13 PART 1 FINAL ANSWER

# Display the RMSE
print(f"New RMSE with genre dummy variables: {round(rmse,4)}")
print(f"This is an improvement in relation to the model without genre dummies")
```

```
New RMSE with genre dummy variables: 2.792
This is an improvement in relation to the model without genre dummies
```

[ ]:
```
# Predict the rating for Titanic using the updated model
titanic_id = final_df.filter(col("primaryTitle") == "Titanic").select("tconst").
 ↪distinct().first()[0]
```

[ ]:
```
# Extract features for Titanic
titanic_features_df = data.filter(col("tconst") == titanic_id)
```

[ ]:
```
titanic_features_df.drop("averageRating")
```

```
Out[110]: DataFrame[tconst: string, avg_total_movies: double, avg_pagerank:
double, avg_outdegree: double, avg_distance_to_kevin_bacon: double, genre_Drama:
int, genre_Action: int, genre_Comedy: int, genre_Adventure: int, genre_Crime:
int, genre_Thriller: int, genre_Sci-Fi: int, genre_Romance: int, genre_Mystery:
int, genre_Horror: int]
```

[ ]:
```
# Fill null values with 0
titanic_features_df = titanic_features_df.na.fill(0)
```

[ ]:
```
# Prepare features for prediction
titanic_features_vector = assembler.transform(titanic_features_df)
```

[ ]:
```
# Ensure no conflicting columns for prediction
titanic_features_vector = titanic_features_vector.drop("features")
```

[ ]:
```
# Assemble features for Titanic again
titanic_assembler = VectorAssembler(
    inputCols=["avg_total_movies", "avg_pagerank", "avg_outdegree",␣
 ↪"avg_distance_to_kevin_bacon"] + [f"genre_{genre}" for genre in␣
 ↪top_10_genres],
    outputCol="features"
)
```

[ ]:
```
titanic_features_vector = titanic_assembler.transform(titanic_features_df)
```

[ ]:
```
# Making sure there is no conflict as we were getting an error in the next cell␣
↪saying "column features already exists"
titanic_features_vector = titanic_features_vector.drop("features")
```

[ ]:
```
# Make prediction for Titanic
titanic_prediction = model.transform(titanic_features_vector)
```

[ ]:
```
# Extract and print the predicted rating
predicted_rating = titanic_prediction.select("prediction").collect()
```

[ ]:
```
# Q13 PART 2 FINAL ANSWER
print(f"Predicted rating for Titanic (1997): {predicted_rating[0][0]}")
```

```
Predicted rating for Titanic (1997): 5.187839660493992
```

Q14 - Open Question: Improve your model by testing different machine learning algorithms, using hyperparameter tuning on these algorithms, changing the included features. What is the RMSE of you final model and what rating does it predict for the 1997 movie Titanic.

[ ]:
```
from pyspark.sql.functions import col, avg, when, lit
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.regression import LinearRegression, RandomForestRegressor,␣
↪GBTRegressor
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
```

[ ]:
```
# Convert RDDs to DataFrames
actor_movie_counts_df = actor_movie_counts_rdd.toDF(["name_nconst",␣
↪"total_movies"])
pagerank_df = pagerank_results.vertices.select(col("id").alias("name_nconst"),␣
↪col("pagerank"))
outdegrees_df = outdegrees_with_names_rdd.map(lambda row:␣
↪Row(name_nconst=row[0], outdegree=row[1][0])).toDF()
shortest_paths_df = shortest_paths.select(col("id").alias("name_nconst"),␣
↪col("distance").alias("distance_to_kevin_bacon"))
```

[ ]:
```
# Combine all features for each movie
features_df = filtered_df.select("tconst", "name_nconst").distinct()
features_df = features_df.join(actor_movie_counts_df, "name_nconst", "left")
features_df = features_df.join(pagerank_df, "name_nconst", "left")
features_df = features_df.join(outdegrees_df, "name_nconst", "left")
features_df = features_df.join(shortest_paths_df, "name_nconst", "left")
```

[ ]:
```
# Combine all features for each movie
features_df = filtered_df.select("tconst", "name_nconst").distinct()
```

```
features_df = features_df.join(actor_movie_counts_df, "name_nconst", "left")
features_df = features_df.join(pagerank_df, "name_nconst", "left")
features_df = features_df.join(outdegrees_df, "name_nconst", "left")
features_df = features_df.join(shortest_paths_df, "name_nconst", "left")
```

[ ]:
```
# Aggregate features for each movie
aggregated_features_df = features_df.groupBy("tconst").agg(
    avg("total_movies").alias("avg_total_movies"),
    avg("pagerank").alias("avg_pagerank"),
    avg("outdegree").alias("avg_outdegree"),
    avg("distance_to_kevin_bacon").alias("avg_distance_to_kevin_bacon")
)
```

[ ]:
```
# Fill null values with 0
aggregated_features_df = aggregated_features_df.na.fill(0)
```

[ ]:
```
# Join with ratings to get the labels
ratings_df = final_df.select("tconst", "averageRating").distinct()
data = aggregated_features_df.join(ratings_df, "tconst")
```

[ ]:
```
# Check for and handle any remaining null values
data = data.na.fill(0)
```

[ ]:
```
# Prepare feature set
assembler = VectorAssembler(
    inputCols=["avg_total_movies", "avg_pagerank", "avg_outdegree",␣
 ↪"avg_distance_to_kevin_bacon"],
    outputCol="features4"
)
scaler_model = StandardScaler(inputCol="features4",␣
 ↪outputCol="scaledFeatures4", withStd=True, withMean=False)
```

[ ]:
```
# Split data into training and test sets
train_data, test_data = data.randomSplit([0.7, 0.3], seed=0)
```

[ ]:
```
# Define models
lr = LinearRegression(featuresCol="scaledFeatures4", labelCol="averageRating")
rf = RandomForestRegressor(featuresCol="scaledFeatures4",␣
 ↪labelCol="averageRating")
gbt = GBTRegressor(featuresCol="scaledFeatures4", labelCol="averageRating")
```

[ ]:
```
# Define parameter grids for hyperparameter tuning
paramGridLR = ParamGridBuilder().addGrid(lr.regParam, [0.01, 0.1, 0.5]).build()
paramGridRF = ParamGridBuilder().addGrid(rf.numTrees, [20, 50, 100]).build()
paramGridGBT = ParamGridBuilder().addGrid(gbt.maxIter, [10, 20, 50]).build()
```

[ ]:
```python
# Define evaluators
evaluator = RegressionEvaluator(labelCol="averageRating", predictionCol="prediction", metricName="rmse")
```

[ ]:
```python
# Cross-validation for each model
crossvalLR = CrossValidator(estimator=Pipeline(stages=[assembler, scaler_model, lr]),
                            estimatorParamMaps=paramGridLR,
                            evaluator=evaluator,
                            numFolds=5)
crossvalRF = CrossValidator(estimator=Pipeline(stages=[assembler, scaler_model, rf]),
                            estimatorParamMaps=paramGridRF,
                            evaluator=evaluator,
                            numFolds=5)
crossvalGBT = CrossValidator(estimator=Pipeline(stages=[assembler, scaler_model, gbt]),
                             estimatorParamMaps=paramGridGBT,
                             evaluator=evaluator,
                             numFolds=5)
```

[ ]:
```python
# Fit models
cvModelLR = crossvalLR.fit(train_data)
cvModelRF = crossvalRF.fit(train_data)
cvModelGBT = crossvalGBT.fit(train_data)
```

[ ]:
```python
# Evaluate models
predictionsLR = cvModelLR.transform(test_data)
predictionsRF = cvModelRF.transform(test_data)
predictionsGBT = cvModelGBT.transform(test_data)

rmseLR = evaluator.evaluate(predictionsLR)
rmseRF = evaluator.evaluate(predictionsRF)
rmseGBT = evaluator.evaluate(predictionsGBT)
```

[ ]:
```python
print(f"RMSE for Linear Regression: {rmseLR}")
print(f"RMSE for Random Forest: {rmseRF}")
print(f"RMSE for Gradient Boosted Trees: {rmseGBT}")
```

```
RMSE for Linear Regression: 2.902608241094834
RMSE for Random Forest: 2.8266434489955774
RMSE for Gradient Boosted Trees: 2.7850901282218037
```

[ ]:
```python
# Q14 PART 1 FINAL ANSWER

# Choose the best model
print(f"Our best model is Gradient Boosted Trees with an RMSE of {rmseGBT}.")
```

Our best model is Gradient Boosted Trees with an RMSE of 2.7850901282218037.

```
[ ]: # Predict the rating for Titanic using the best model

     # Extract features for Titanic
     titanic_features_df = filtered_df.filter(col("tconst") == titanic_id).
      ↪select("tconst", "name_nconst").distinct()
```

```
[ ]: # Perform the joins
     titanic_features_df = titanic_features_df.join(actor_movie_counts_df,␣
      ↪"name_nconst", "left")
     titanic_features_df = titanic_features_df.join(pagerank_df, "name_nconst",␣
      ↪"left")
     titanic_features_df = titanic_features_df.join(outdegrees_df, "name_nconst",␣
      ↪"left")
     titanic_features_df = titanic_features_df.join(shortest_paths_df,␣
      ↪"name_nconst", "left")

     # Aggregate features for Titanic
     titanic_aggregated_features_df = titanic_features_df.groupBy("tconst").agg(
         avg("total_movies").alias("avg_total_movies"),
         avg("pagerank").alias("avg_pagerank"),
         avg("outdegree").alias("avg_outdegree"),
         avg("distance_to_kevin_bacon").alias("avg_distance_to_kevin_bacon")
     )
```

```
[ ]: # Add dummy variables for Titanic
     for genre in top_10_genres:
         titanic_aggregated_features_df = titanic_aggregated_features_df.
      ↪withColumn(f"genre_{genre}", lit(1) if genre in ["Romance", "Drama"] else␣
      ↪lit(0))

     # Fill null values with 0
     titanic_aggregated_features_df = titanic_aggregated_features_df.na.fill(0)
```

```
[ ]: # Prepare feature set
     assembler = VectorAssembler(
         inputCols=["avg_total_movies", "avg_pagerank", "avg_outdegree",␣
      ↪"avg_distance_to_kevin_bacon"] + [f"genre_{genre}" for genre in␣
      ↪top_10_genres],
         outputCol="raw_features"
     )

     # Assemble the features
     assembled_data = assembler.transform(titanic_aggregated_features_df)
```

```
[ ]: # Prepare features for prediction
     # Scale the features
     scaler3 = StandardScaler(inputCol="raw_features", outputCol="scaledFeatures4",␣
      ↪withStd=True, withMean=False)
```

```
[ ]: # Making sure there is no conflict as we were getting an error in the next cell␣
      ↪saying "column features4 already exists"
     titanic_features_vector = titanic_features_vector.drop("scaledFeatures4")
```

```
[ ]: # Make prediction for Titanic
     titanic_prediction = cvModelGBT.transform(titanic_features_vector)
```

```
[ ]: # Extract and print the predicted rating
     predicted_rating = titanic_prediction.select("prediction").collect()
```

```
[ ]: # Q14 FINAL ANSWER PART 2

     # Display the prediction for Titanic
     print(f"Predicted Rating for Titanic with the best model (Gradient Boosted␣
      ↪Trees): {round(predicted_rating[0][0],2)}")
```

```
Predicted Rating for Titanic with the best model (Gradient Boosted Trees): 4.77
```

2 Statement of Authorship

André Filipe Silva - 20230972 João Gonçalves - 20230560