20230972_20230560_submission

August 7, 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
```

[ ]:
```
%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"),
```

```python
    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).

[ ]:
```
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)
```

[ ]:
```python
# 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|
```

```
+---------+---------+------------+------------+-------+---------+-------+------
--------+----------------+
```

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

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

```
[ ]: # 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.

```
[ ]: 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
```

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

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

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

```
[ ]: # 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()]
```

```
[ ]: # 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
```

[ ]:
```
# 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))
```

[ ]:
```
# 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
outdegrees_df = outdegrees_with_names_rdd.map(lambda row:␣
 ↪Row(name_nconst=row[0], outdegree=row[1][0])).toDF()
```

[ ]:
```
# 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"))
```

[ ]:
```
# 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")
```

[ ]:
```
# 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)
```

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

[ ]:
```
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)
```

[ ]:
```
# 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="scaledFeatures", labelCol="averageRating")
```

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

[ ]:
```
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")
```

[ ]:
```
# 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()
```

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

[ ]:
```
# 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)
```

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

[ ]:
```
# 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)
```

[ ]:
```
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
```

[ ]:
```
# 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
```