Description of the MovieLens 20M Dataset:

The MovieLens 20M dataset is a publicly available dataset curated by GroupLens Research. It contains 20 million ratings and 465,000 tag applications applied to 27,000 movies by 138,000 users. The dataset is widely used in recommendation system research and provides an excellent foundation for building and testing collaborative filtering algorithms.

The dataset consists of multiple files, but for this project, we will primarily use the "movies.csv" and "ratings.csv" files:

movies.csv: This file contains information about movies, including the movie ID, title, and genres. Each movie is uniquely identified by a movie ID, which is essential for linking the movies to ratings.

ratings.csv: This file contains user ratings for the movies. It includes user IDs, movie IDs, ratings (ranging from 0.5 to 5.0), and timestamps. The ratings are explicit, representing the direct feedback provided by users.

The MovieLens 20M dataset is valuable due to its rich set of ratings, which allows us to explore user behavior, perform Exploratory Data Analysis (EDA), and build effective recommendation models. The scale and diversity of the data also make it ideal for testing the performance of machine learning techniques, such as collaborative filtering, in a real-world setting.

This dataset provides an ideal testbed for understanding user preferences, developing machine learning models, and evaluating recommendation systems on a large, diverse set of users and items.

```python
import os
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, count, avg
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql import functions as F
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Initialize Spark session
spark =
SparkSession.builder.appName("MovieLensRecommendation").getOrCreate()

# Cell 2: Define file paths and options
movies_location = "/content/movie.csv"  # Update with your DBFS path
ratings_location = "/content/rating.csv"  # Update with your DBFS path
file_type = "csv"

# CSV options
infer_schema = "true"
```

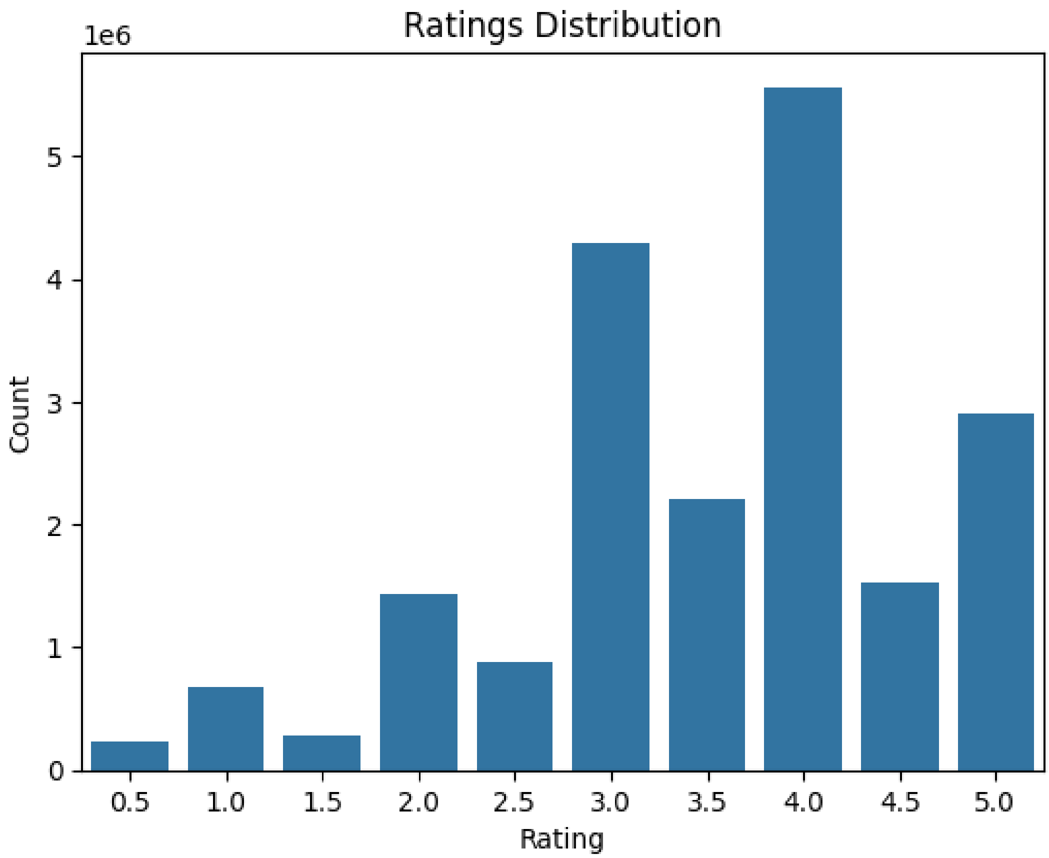
```python
first_row_is_header = "true"
delimiter = ","

#Load data files
movies_df = spark.read.format(file_type) \
    .option("inferSchema", infer_schema) \
    .option("header", first_row_is_header) \
    .option("sep", delimiter) \
    .load(movies_location)

ratings_df = spark.read.format(file_type) \
    .option("inferSchema", infer_schema) \
    .option("header", first_row_is_header) \
    .option("sep", delimiter) \
    .load(ratings_location)

# Display loaded data
print("Movies DataFrame:")
movies_df.show(5)
print("Ratings DataFrame:")
ratings_df.show(5)

# Show schema to verify data
movies_df.printSchema()
ratings_df.printSchema()
```

```
Movies DataFrame:
+-------+--------------------+--------------------+
|movieId|               title|              genres|
+-------+--------------------+--------------------+
|      1|    Toy Story (1995)|Adventure|Animati...|
|      2|      Jumanji (1995)|Adventure|Childre...|
|      3|Grumpier Old Men ...|      Comedy|Romance|
|      4|Waiting to Exhale...|Comedy|Drama|Romance|
|      5|Father of the Bri...|              Comedy|
+-------+--------------------+--------------------+
only showing top 5 rows

Ratings DataFrame:
+------+-------+------+-------------------+
|userId|movieId|rating|          timestamp|
+------+-------+------+-------------------+
|     1|      2|   3.5|2005-04-02 23:53:47|
|     1|     29|   3.5|2005-04-02 23:31:16|
|     1|     32|   3.5|2005-04-02 23:33:39|
|     1|     47|   3.5|2005-04-02 23:32:07|
|     1|     50|   3.5|2005-04-02 23:29:40|
+------+-------+------+-------------------+
only showing top 5 rows
```

```
root
 |-- movieId: integer (nullable = true)
 |-- title: string (nullable = true)
 |-- genres: string (nullable = true)

root
 |-- userId: integer (nullable = true)
 |-- movieId: integer (nullable = true)
 |-- rating: double (nullable = true)
 |-- timestamp: timestamp (nullable = true)
```

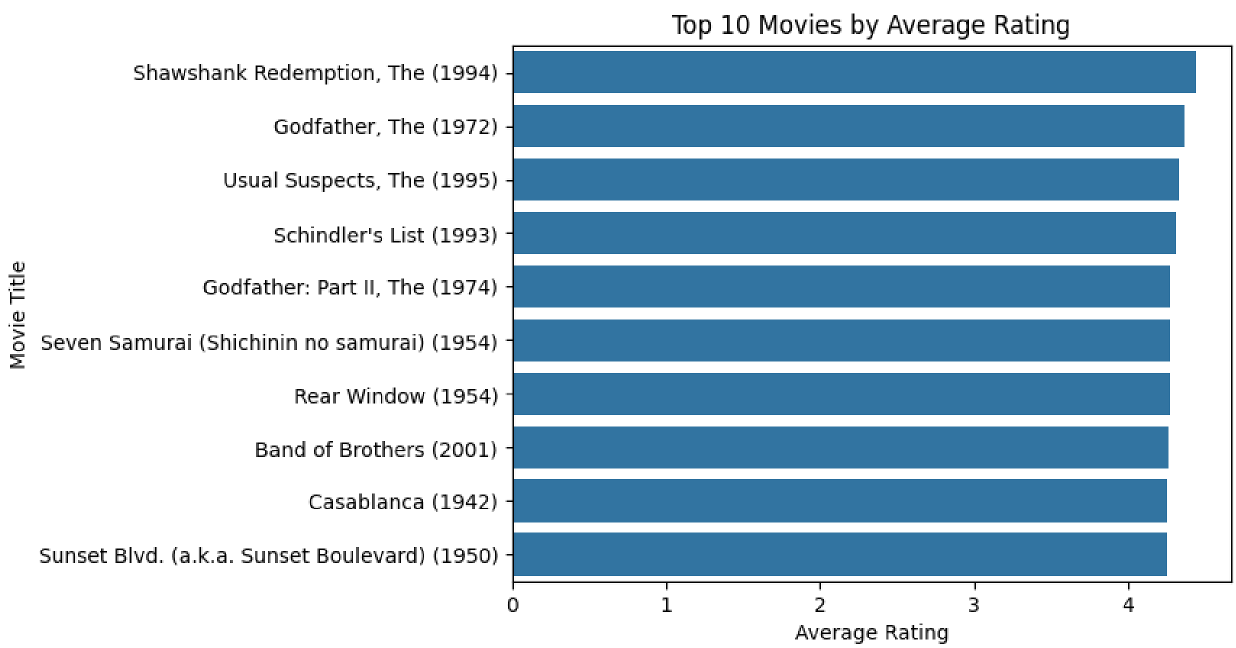
```
#  Perform basic analysis
print("Movies count:", movies_df.count())
print("Ratings count:", ratings_df.count())

# Ratings distribution analysis
ratings_distribution =
ratings_df.groupBy("rating").count().orderBy("rating")
ratings_distribution.show()

# Visualization: Ratings distribution
ratings_distribution_pd = ratings_distribution.toPandas()
sns.barplot(x="rating", y="count", data=ratings_distribution_pd)
plt.title("Ratings Distribution")
plt.xlabel("Rating")
plt.ylabel("Count")
plt.show()
```

```
Movies count: 27278
Ratings count: 20000263
+------+-------+
|rating|  count|
+------+-------+
|   0.5| 239125|
|   1.0| 680732|
|   1.5| 279252|
|   2.0|1430997|
|   2.5| 883398|
|   3.0|4291193|
|   3.5|2200156|
|   4.0|5561926|
|   4.5|1534824|
|   5.0|2898660|
+------+-------+
```




```
average_ratings_df = ratings_df.groupBy("movieId").agg(
    avg("rating").alias("avg_rating"),
    count("rating").alias("rating_count")
)
average_ratings_df.show(5)

# Filter movies with sufficient ratings
popular_movies_df = average_ratings_df.filter(col("rating_count") >
50)
popular_movies_df.show(5)

# Visualization: Average ratings for top movies
popular_movies_pd = popular_movies_df.join(movies_df,
"movieId").select("title", "avg_rating",
"rating_count").orderBy("avg_rating",
ascending=False).limit(10).toPandas()
sns.barplot(x="avg_rating", y="title", data=popular_movies_pd)
plt.title("Top 10 Movies by Average Rating")
plt.xlabel("Average Rating")
plt.ylabel("Movie Title")
plt.show()
```

```
+-------+------------------+------------+
|movieId|        avg_rating|rating_count|
+-------+------------------+------------+
|   3997|2.0703468490473864|        2047|
|   1580|  3.55831928049466|       35580|
|   3918| 2.918940609951846|        1246|
|   2366|3.5492681454655197|        6627|
|   3175| 3.600717102904267|       13945|
+-------+------------------+------------+
only showing top 5 rows

+-------+------------------+------------+
|movieId|        avg_rating|rating_count|
+-------+------------------+------------+
|   3997|2.0703468490473864|        2047|
|   1580|  3.55831928049466|       35580|
|   3918| 2.918940609951846|        1246|
|   2366|3.5492681454655197|        6627|
|   3175| 3.600717102904267|       13945|
+-------+------------------+------------+
only showing top 5 rows
```



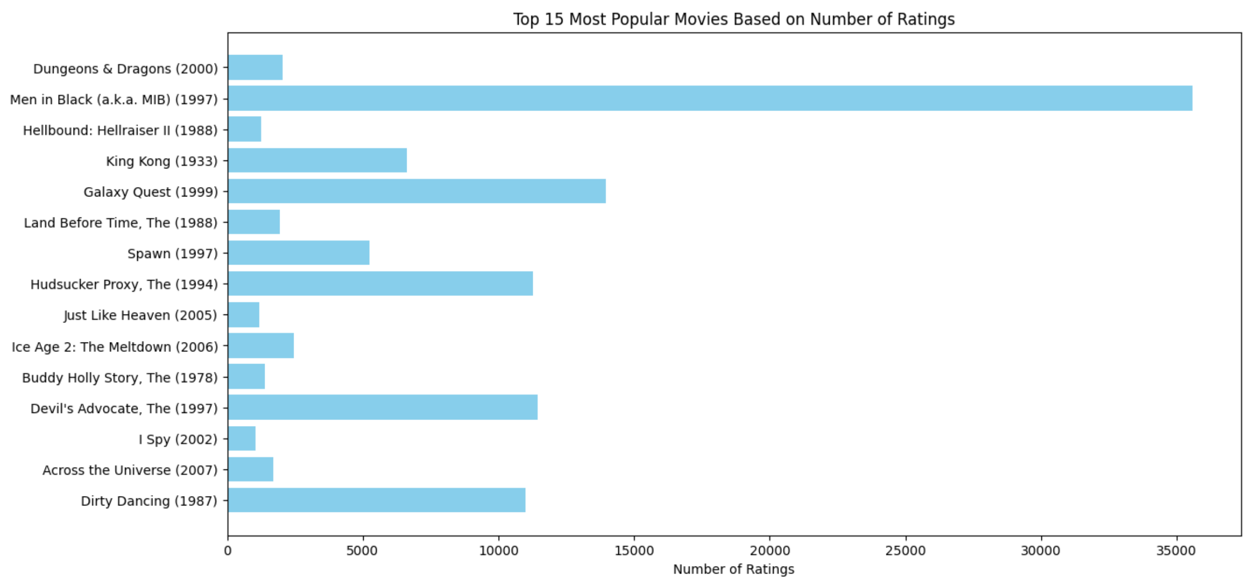
Most popular movies

```
from pyspark.sql.functions import *

most_popular = ratings_df\
```

```
.groupBy("movieId")\
.agg(count("userId"))\
.withColumnRenamed("count(userId)", "num_ratings")\
.sort(desc("num_ratings"))

# This DataFrame contains only the movieId and num_ratings. The actual
title of the movie is stored in the movies DataFrame. To get the movie
titles, we can join our most_popular DataFrame with the movies
DataFrame on movieId

most_popular_movies = most_popular.join(movies_df,
most_popular.movieId == movies_df.movieId)
most_popular_movies.show(15, truncate=False)

+-------+-----------+-------+----------------------------------
+-----------------------------------+
|movieId|num_ratings|movieId|title                             |genres
|
+-------+-----------+-------+----------------------------------
+-----------------------------------+
|3997   |2047       |3997   |Dungeons & Dragons (2000)         |
Adventure|Fantasy                   |
|1580   |35580      |1580   |Men in Black (a.k.a. MIB) (1997)|Action|
Comedy|Sci-Fi                |
|3918   |1246       |3918   |Hellbound: Hellraiser II (1988) |Horror
|
|2366   |6627       |2366   |King Kong (1933)                |Action|
Adventure|Fantasy|Horror     |
|3175   |13945      |3175   |Galaxy Quest (1999)             |
Adventure|Comedy|Sci-Fi             |
|4519   |1936       |4519   |Land Before Time, The (1988)    |
Adventure|Animation|Children|Fantasy|
|1591   |5255       |1591   |Spawn (1997)                    |Action|
Adventure|Sci-Fi|Thriller    |
|471    |11268      |471    |Hudsucker Proxy, The (1994)     |Comedy
|
|36525  |1169       |36525  |Just Like Heaven (2005)         |Comedy|
Fantasy|Romance              |
|44022  |2465       |44022  |Ice Age 2: The Meltdown (2006)  |
Adventure|Animation|Children|Comedy |
|2866   |1407       |2866   |Buddy Holly Story, The (1978)   |Drama
|
|1645   |11458      |1645   |Devil's Advocate, The (1997)    |Drama|
Mystery|Thriller              |
|5803   |1046       |5803   |I Spy (2002)                    |Action|
Adventure|Comedy|Crime       |
|54190  |1687       |54190  |Across the Universe (2007)      |Drama|
Fantasy|Musical|Romance       |
|1088   |11013      |1088   |Dirty Dancing (1987)            |Drama|
Musical|Romance               |
```

```
+-------+----------+-------+--------------------------------
+--------------------------------+
only showing top 15 rows
```

```
most_popular_movies_pd = most_popular_movies.limit(15).toPandas()

# Plot the data using matplotlib
plt.figure(figsize=(14, 7))
plt.barh(most_popular_movies_pd['title'],
most_popular_movies_pd['num_ratings'], color='skyblue')
plt.xlabel('Number of Ratings')
plt.title('Top 15 Most Popular Movies Based on Number of Ratings')
plt.gca().invert_yaxis()  # Invert y-axis to have the most popular
movie at the top
plt.show()
```

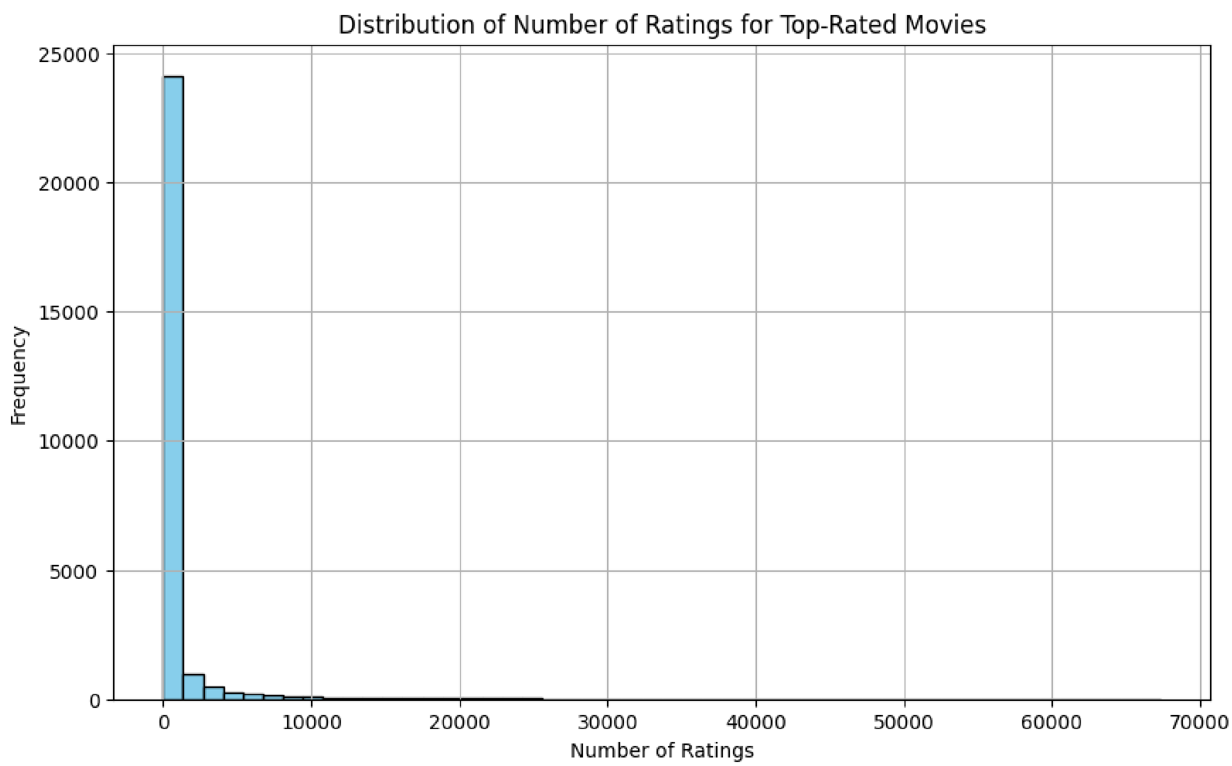


Top rated movies

```
# The movies listed here appear to be quite niche. We want to focus on
top rated movies that also have a decent number of ratings, so want to
take into account both the average rating and the number of ratings.
We can easily create a DataFrame which has both of these columns by
specifying multiple expressions within one agg() call.

top_rated = ratings_df\
.groupBy("movieId")\
.agg(count("userId"), avg(col("rating")))\
.withColumnRenamed("count(userId)", "num_ratings")\
.withColumnRenamed("avg(rating)", "avg_rating")
```

```
top_rated_movies = top_rated.join(movies_df, top_rated.movieId ==
movies_df.movieId).sort(desc("avg_rating"), desc("num_ratings"))

top_rated_movies.show(15)

+-------+-----------+----------+-------+--------------------
+--------------------+
|movieId|num_ratings|avg_rating|movieId|               title|
genres|
+-------+-----------+----------+-------+--------------------
+--------------------+
| 103871|          2|       5.0| 103871|Consuming Kids: T...|
Documentary|
| 108527|          2|       5.0| 108527|  Catastroika (2012)|
Documentary|
| 114214|          1|       5.0| 114214|Mishen (Target) (...|
Drama|Sci-Fi|
| 129295|          1|       5.0| 129295|A Gun for Jennife...|Crime|
Drama|Thriller|
| 129516|          1|       5.0| 129516|       Poison (1951)|
Comedy|
| 128506|          1|       5.0| 128506|   Rent-a-Cat (2012)|
Comedy|Drama|
|  98761|          1|       5.0|  98761|Shaolin Temple 2:...|
Action|Comedy|
| 121029|          1|       5.0| 121029|No Distance Left ...|
Documentary|
|  92956|          1|       5.0|  92956|Little Criminals ...|
Crime|Drama|
|  79866|          1|       5.0|  79866|Schmatta: Rags to...|
Documentary|
| 125599|          1|       5.0| 125599|Always for Pleasu...|  (no
genres listed)|
| 129243|          1|       5.0| 129243|Afstiros katallil...|
Comedy|
| 129034|          1|       5.0| 129034| Serving Life (2011)|
Documentary|
| 129478|          1|       5.0| 129478|A Blank on the Ma...|
Documentary|
| 112790|          1|       5.0| 112790|Going Down in LA-...|Comedy|
Drama|Romance|
+-------+-----------+----------+-------+--------------------
+--------------------+
only showing top 15 rows


# We see that all of the movies with an average rating of exactly 5.0
have 2 or less ratings. We would like to only consider movies that
have achieved some minimum number of ratings. To determine an
appropriate threshold, we should investigate the distribution of
```

```
num_ratings. We can do this by calculating some summary statistics
within Spark.

# Calculate average, minimum, and maximum of num_ratings
top_rated_movies.select([mean('num_ratings'), min('num_ratings'),
max('num_ratings')]).show(1)

num_ratings_data = top_rated_movies.select("num_ratings").toPandas()


#Top 15 Movies Based on Average Rating (with a threshold)
# Plot histogram of num_ratings
plt.figure(figsize=(10, 6))
plt.hist(num_ratings_data['num_ratings'], bins=50, color='skyblue',
edgecolor='black')
plt.title('Distribution of Number of Ratings for Top-Rated Movies')
plt.xlabel('Number of Ratings')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

+----------------+----------------+----------------+
| avg(num_ratings)|min(num_ratings)|max(num_ratings)|
+----------------+----------------+----------------+
|747.8411232425965|               1|           67310|
+----------------+----------------+----------------+
```



```
top_rated_movies.where("num_ratings > 500").show(5, truncate=False)
```

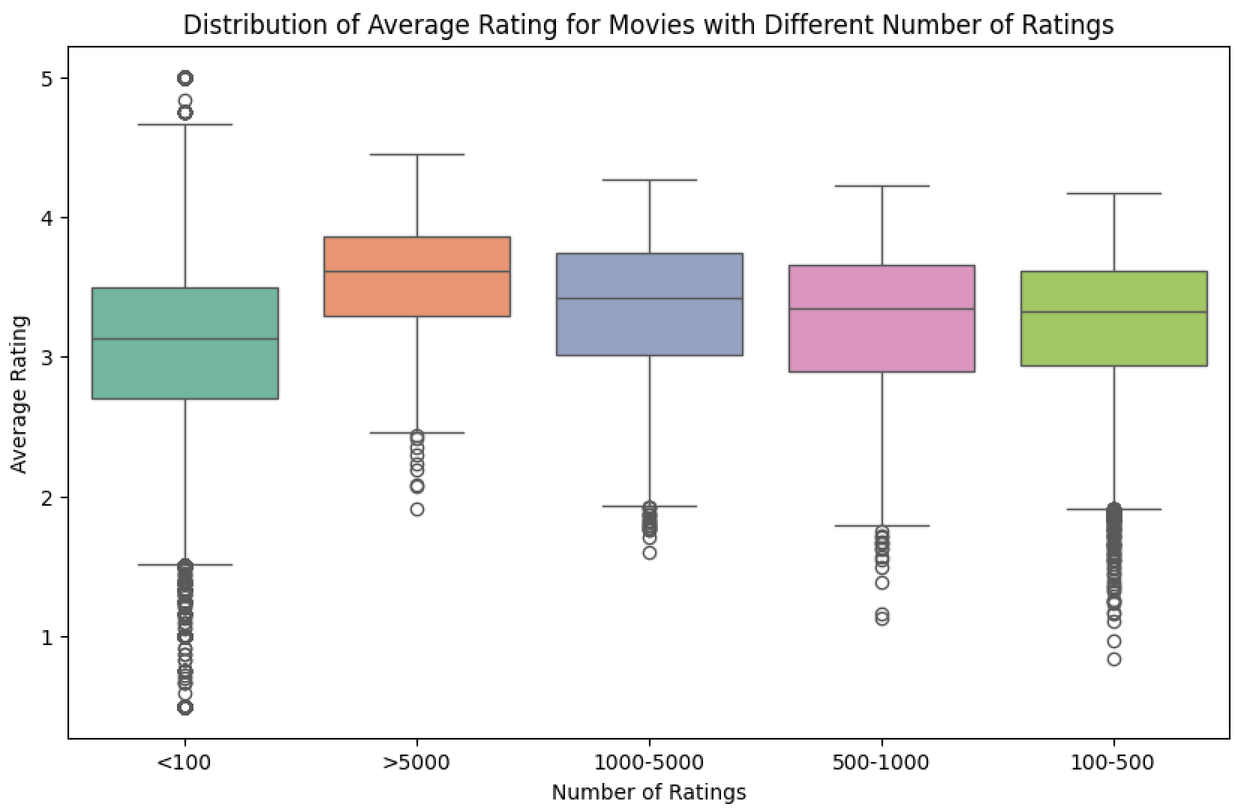
```
+-------+-----------+-----------------+-------
+--------------------------------+----------------------+
|movieId|num_ratings|avg_rating       |movieId|title
|genres                |
+-------+-----------+-----------------+-------
+--------------------------------+----------------------+
|318    |63366      |4.446990499637029|318    |Shawshank Redemption,
The (1994)|Crime|Drama           |
|858    |41355      |4.364732196832306|858    |Godfather, The (1972)
|Crime|Drama           |
|50     |47006      |4.334372207803259|50     |Usual Suspects, The
(1995)      |Crime|Mystery|Thriller|
|527    |50054      |4.310175010988133|527    |Schindler's List (1993)
|Drama|War             |
|1221   |27398      |4.275640557704942|1221   |Godfather: Part II, The
(1974)  |Crime|Drama           |
+-------+-----------+-----------------+-------
+--------------------------------+----------------------+
only showing top 5 rows
```

Boxplot of Average Rating Distribution for Movies with Different Rating Counts

```
# Visualization 4: Boxplot of Average Rating by Number of Ratings
Categories
# Create bins for the number of ratings to categorize the movies
bins = [0, 100, 500, 1000, 5000, 10000]
labels = ['<100', '100-500', '500-1000', '1000-5000', '>5000']
top_rated_movies = top_rated_movies.withColumn('rating_bins',
F.when(top_rated_movies['num_ratings'] < 100, '<100')
    .when((top_rated_movies['num_ratings'] >= 100) &
(top_rated_movies['num_ratings'] < 500), '100-500')
    .when((top_rated_movies['num_ratings'] >= 500) &
(top_rated_movies['num_ratings'] < 1000), '500-1000')
    .when((top_rated_movies['num_ratings'] >= 1000) &
(top_rated_movies['num_ratings'] < 5000), '1000-5000')
    .otherwise('>5000'))

# Convert to Pandas for visualization
top_rated_movies_pd = top_rated_movies.toPandas()

# Create a boxplot of avg_rating across the different bins of
num_ratings
plt.figure(figsize=(10, 6))
```

```
sns.boxplot(x='rating_bins', y='avg_rating', data=top_rated_movies_pd,
palette='Set2')
plt.title('Distribution of Average Rating for Movies with Different
Number of Ratings')
plt.xlabel('Number of Ratings')
plt.ylabel('Average Rating')
plt.show()
```

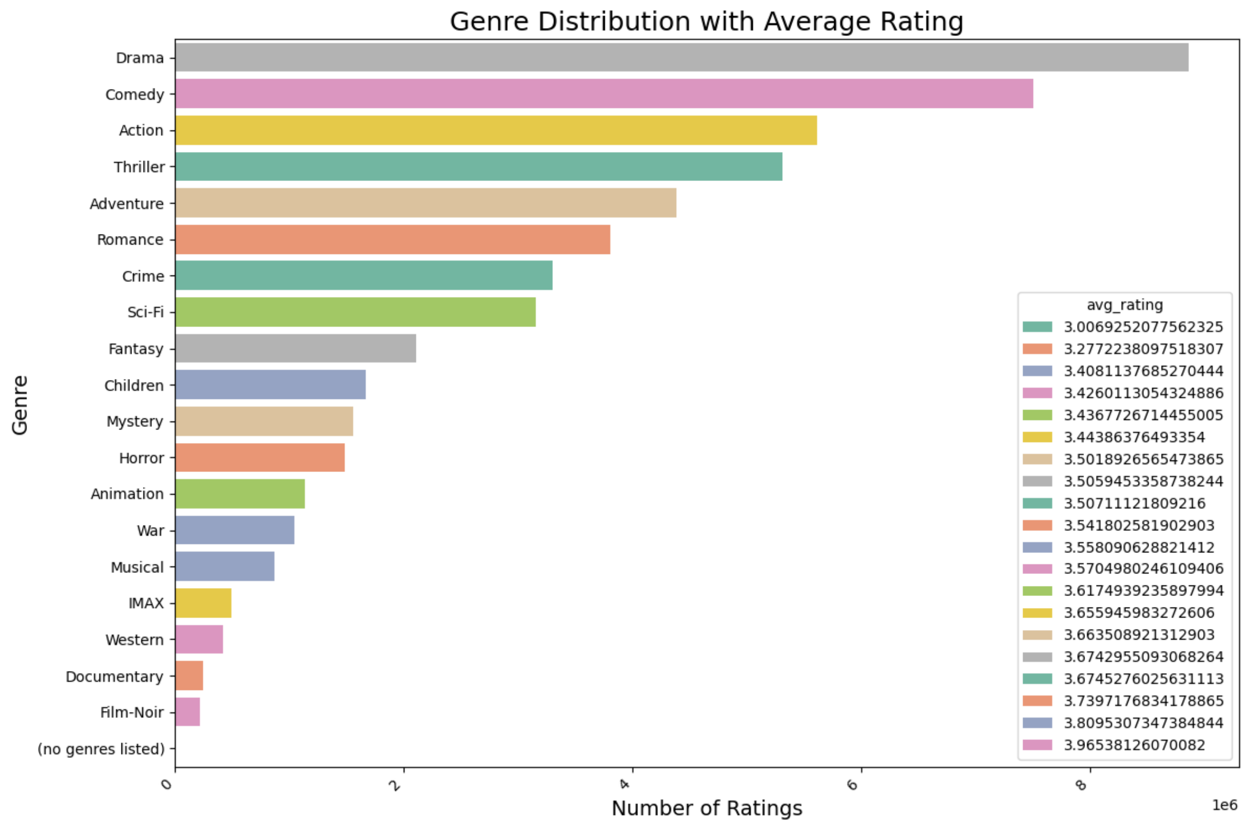
```
<ipython-input-14-1e7e65b427bb>:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='rating_bins', y='avg_rating',
data=top_rated_movies_pd, palette='Set2')
```



Top 10 Genres with Average Rating Distribution

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from pyspark.sql.functions import explode, split, col
```

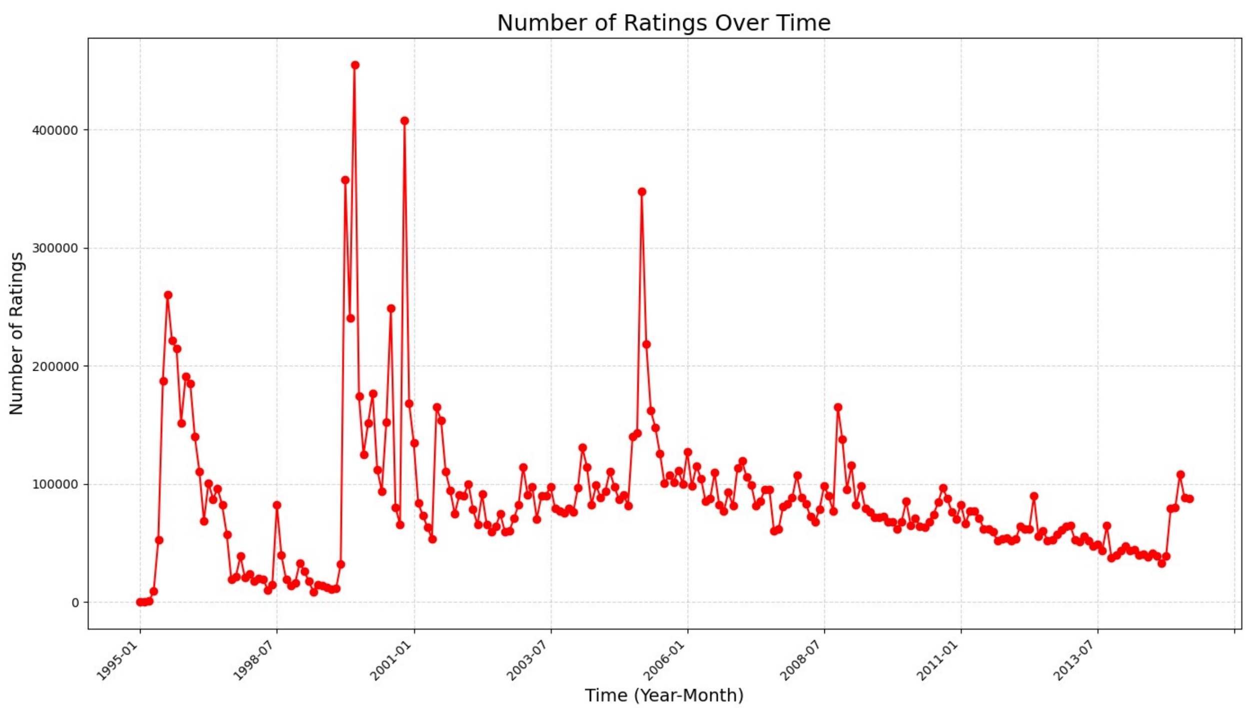
```python
genres_df = movies_df.withColumn("genre",
explode(split(movies_df['genres'], '\|')))

# Join with ratings to get the count of ratings and average rating by
genre
ratings_by_genre = genres_df.join(ratings_df, genres_df.movieId ==
ratings_df.movieId)

# Calculate the number of ratings and average rating for each genre
genre_stats = ratings_by_genre.groupBy("genre").agg(
    (F.count("rating")).alias("num_ratings"),
    (F.avg("rating")).alias("avg_rating")
).orderBy("num_ratings", ascending=False)

# Convert to Pandas for visualization
genre_stats_pd = genre_stats.toPandas()

# Plotting
plt.figure(figsize=(12, 8))

# Barplot with distinct colors for each genre
sns.barplot(x="num_ratings", y="genre", data=genre_stats_pd,
palette="Set2", hue="avg_rating", dodge=False)

# Title and labels with improved readability
plt.title('Genre Distribution with Average Rating', fontsize=18)
plt.xlabel('Number of Ratings', fontsize=14)
plt.ylabel('Genre', fontsize=14)

# Adjusting label visibility
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Time-Based Line Plot

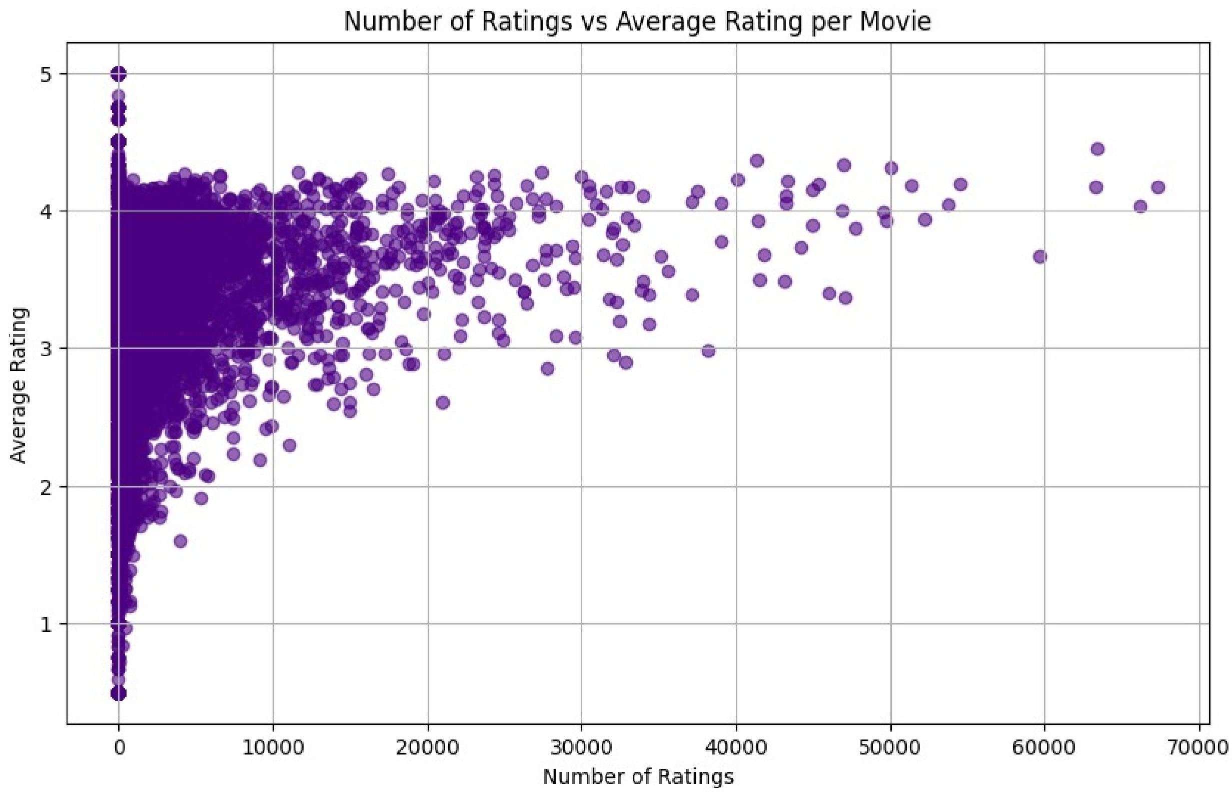
```
import matplotlib.pyplot as plt
import pandas as pd
from pyspark.sql import functions as F
from pyspark.sql.types import TimestampType
from matplotlib.ticker import MaxNLocator

# Assuming ratings_df is already loaded

# Convert timestamp to datetime format
ratings_df = ratings_df.withColumn("timestamp",
F.col("timestamp").cast(TimestampType()))

# Extract year and month from timestamp
ratings_df = ratings_df.withColumn("year_month",
F.date_format("timestamp", "yyyy-MM"))

# Group by year_month and count the number of ratings per month
ratings_by_month =
ratings_df.groupBy("year_month").agg(F.count("rating").alias("num_rati
ngs")).orderBy("year_month")

# Convert to Pandas for visualization
ratings_by_month_pd = ratings_by_month.toPandas()
```

```
# Plotting
plt.figure(figsize=(14, 8))

# Line plot showing number of ratings per month
plt.plot(ratings_by_month_pd['year_month'],
ratings_by_month_pd['num_ratings'], marker='o', color='red')

# Title and labels
plt.title('Number of Ratings Over Time', fontsize=18)
plt.xlabel('Time (Year-Month)', fontsize=14)
plt.ylabel('Number of Ratings', fontsize=14)

# Adjust the x-axis labels for readability
plt.xticks(rotation=45, ha='right')  # Rotate labels
plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True,
prune='both', nbins=10))  # Limit number of ticks

# Tidy up the layout
plt.tight_layout()
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
```



4. Correlation between Number of Ratings and Average Rating per Movie

```
# Count the number of ratings per movie and rename the column to
'num_ratings'
movie_rating_count =
ratings_df.groupBy('movieId').count().withColumnRenamed('count',
'num_ratings')

# Get the average rating for each movie
movie_avg_ratings =
ratings_df.groupBy('movieId').agg(F.avg('rating').alias('avg_rating'))

# Join the two DataFrames on 'movieId'
movie_data = movie_rating_count.join(movie_avg_ratings, on='movieId')

# Convert to Pandas DataFrame for visualization
movie_data_pd = movie_data.toPandas()

# Scatter plot: Plot 'num_ratings' against 'avg_rating'
plt.figure(figsize=(10, 6))
plt.scatter(movie_data_pd['num_ratings'], movie_data_pd['avg_rating'],
alpha=0.6, color='indigo')
plt.title('Number of Ratings vs Average Rating per Movie')
plt.xlabel('Number of Ratings')
plt.ylabel('Average Rating')
plt.grid(True)
plt.show()
```



5. Rating Frequency vs. Movie Rating (Hexbin Plot)

```
# Get number of ratings and average rating per movie
movie_rating_data = movie_rating_count.join(movie_avg_ratings,
on='movieId')

# Convert to Pandas for visualization
movie_rating_data_pd = movie_rating_data.toPandas()

# Hexbin plot
plt.figure(figsize=(10, 6))
plt.hexbin(movie_rating_data_pd['num_ratings'],
movie_rating_data_pd['avg_rating'], gridsize=50, cmap='Blues')
plt.title('Hexbin Plot: Rating Frequency vs Movie Rating')
plt.xlabel('Number of Ratings')
plt.ylabel('Average Rating')
plt.colorbar(label='Frequency')
plt.show()
```

Splitting in Train, Test and Validation dataset

```
# Split the dataset into training, validation, and test sets
split_ratios = [0.6, 0.2, 0.2]
train_df, validation_df, test_df =
ratings_df.randomSplit(split_ratios, 4)

# Display counts of each dataset
print(f"Training set count: {train_df.count()}\nValidation set count:
{validation_df.count()}\nTest set count: {test_df.count()}\n")

# Display sample records from each dataset
print("Sample records from Training dataset:")
train_df.show(4, truncate=False)

print("Sample records from Validation dataset:")
validation_df.show(4, truncate=False)

print("Sample records from Test dataset:")
test_df.show(4, truncate=False)

Training set count: 11999231
Validation set count: 4000853
Test set count: 4000179
```

```
Sample records from Training dataset:
+------+-------+------+-------------------+
|userId|movieId|rating|timestamp          |
+------+-------+------+-------------------+
|1     |29     |3.5   |2005-04-02 23:31:16|
|1     |32     |3.5   |2005-04-02 23:33:39|
|1     |47     |3.5   |2005-04-02 23:32:07|
|1     |50     |3.5   |2005-04-02 23:29:40|
+------+-------+------+-------------------+
only showing top 4 rows

Sample records from Validation dataset:
+------+-------+------+-------------------+
|userId|movieId|rating|timestamp          |
+------+-------+------+-------------------+
|1     |260    |4.0   |2005-04-02 23:33:46|
|1     |318    |4.0   |2005-04-02 23:33:18|
|1     |541    |4.0   |2005-04-02 23:30:03|
|1     |589    |3.5   |2005-04-02 23:45:57|
+------+-------+------+-------------------+
only showing top 4 rows

Sample records from Test dataset:
+------+-------+------+-------------------+
|userId|movieId|rating|timestamp          |
+------+-------+------+-------------------+
|1     |2      |3.5   |2005-04-02 23:53:47|
|1     |223    |4.0   |2005-04-02 23:46:13|
|1     |367    |3.5   |2005-04-02 23:53:00|
|1     |924    |3.5   |2004-09-10 03:06:38|
+------+-------+------+-------------------+
only showing top 4 rows
```

Colloborative filtering

```
# ALS Model Training and Evaluation Setup

# Import necessary libraries
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator

# Initialize ALS learner
# ALS (Alternating Least Squares) is used for collaborative filtering
recommendations
als = ALS(
    maxIter=5,  # Number of iterations to run
    seed=4,  # Random seed for reproducibility
```

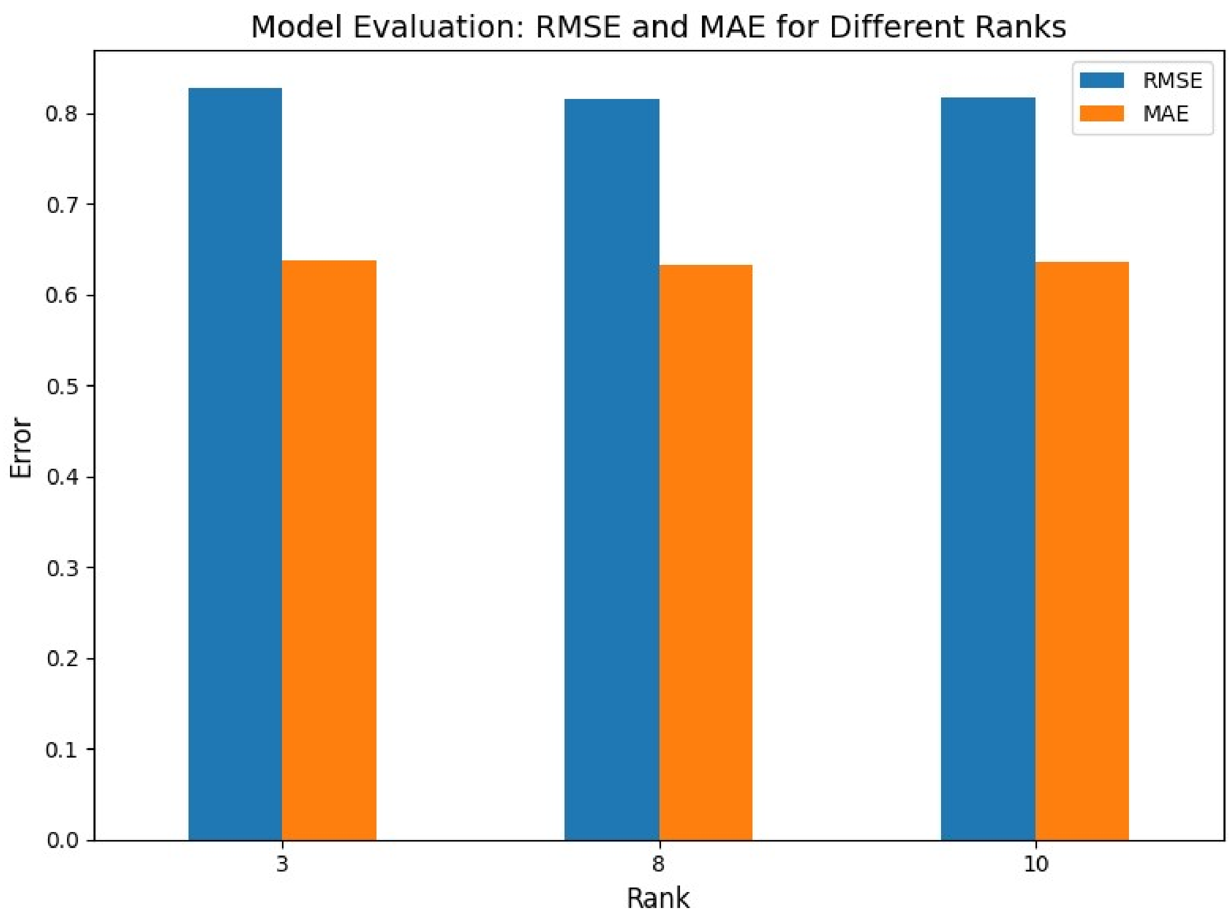
```python
    regParam=0.1,  # Regularization parameter to prevent overfitting
    userCol="userId",  # Column representing user IDs
    itemCol="movieId",  # Column representing item (movie) IDs
    ratingCol="rating"  # Column representing ratings given by users
)

# Initialize RMSE evaluator
# RegressionEvaluator calculates the Root Mean Squared Error (RMSE) to
evaluate model accuracy
tolerance = 0.03  # Acceptable error margin
reg_eval = RegressionEvaluator(
    predictionCol="prediction",  # Column containing predicted ratings
    labelCol="rating",  # Column containing actual ratings
    metricName="rmse"  # Metric to evaluate
)

mae_eval = RegressionEvaluator(
    predictionCol="prediction",  # Column containing predicted ratings
    labelCol="rating",  # Column containing actual ratings
    metricName="mae"  # Metric to evaluate
)

# Define different ranks for evaluation
# Rank determines the number of latent features used by ALS
ranks = [3, 8, 10]
errors = []  # List to store RMSE errors for each rank
models = []  # List to store models for each rank
min_error = float("inf")  # Initialize minimum error
best_rank = -1  # Initialize the best rank variable

# Initialize lists to store errors for both metrics
mae_errors = []  # List to store MAE errors
rmse_errors = []  # List to store RMSE errors
models = []  # List to store models for each rank

# Initialize variables for tracking the best model
min_error = float("inf")  # Initialize minimum error
best_rank = -1  # Initialize the best rank variable

# Loop through each rank to train and evaluate the ALS model
for rank in ranks:
    print(f"Evaluating model with rank: {rank}")

    # Set the rank for ALS
    als.setRank(rank)

    # Train the ALS model on the training dataset
    model = als.fit(train_df)

    # Predict ratings for the validation dataset
```

```
    predict_df = model.transform(validation_df)

    # Filter out NaN predictions
    predicted_ratings_df = predict_df.filter(predict_df.prediction != float('nan'))

    # Calculate RMSE and MAE
    rmse_error = reg_eval.evaluate(predicted_ratings_df)  # RMSE for this rank
    mae_error = mae_eval.evaluate(predicted_ratings_df)  # MAE for this rank

    # Append errors for this rank
    mae_errors.append(mae_error)
    rmse_errors.append(rmse_error)

    # Print the errors for the current rank
    print(f"For rank {rank}, the RMSE is {rmse_error:.4f}, MAE is {mae_error:.4f}")

    # Update the best model if this rank yields a lower RMSE
    if rmse_error < min_error:
        min_error = rmse_error
        best_rank = rank

    # Store the model for the current rank
    models.append(model)

# Output the best rank based on RMSE
print(f"The best model was trained with rank: {best_rank}")

# Set ALS to use the best rank and save the corresponding model
als.setRank(best_rank)
my_model = models[ranks.index(best_rank)]
```

```
Evaluating model with rank: 3
For rank 3, the RMSE is 0.8274, MAE is 0.6381
Evaluating model with rank: 8
For rank 8, the RMSE is 0.8149, MAE is 0.6324
Evaluating model with rank: 10
For rank 10, the RMSE is 0.8165, MAE is 0.6354
The best model was trained with rank: 8
```

```
# Create a DataFrame to visualize the errors
error_df = pd.DataFrame({
    'Rank': ranks,
    'RMSE': rmse_errors,
    'MAE': mae_errors
})

# Set the Rank column as the index for better visualization
```

```
error_df.set_index('Rank', inplace=True)

# Plot RMSE and MAE comparison
fig, ax = plt.subplots(figsize=(8, 6))
error_df.plot(kind='bar', ax=ax)

# Title and labels
ax.set_title('Model Evaluation: RMSE and MAE for Different Ranks',
fontsize=14)
ax.set_xlabel('Rank', fontsize=12)
ax.set_ylabel('Error', fontsize=12)
ax.legend(['RMSE', 'MAE'])

# Display the plot
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

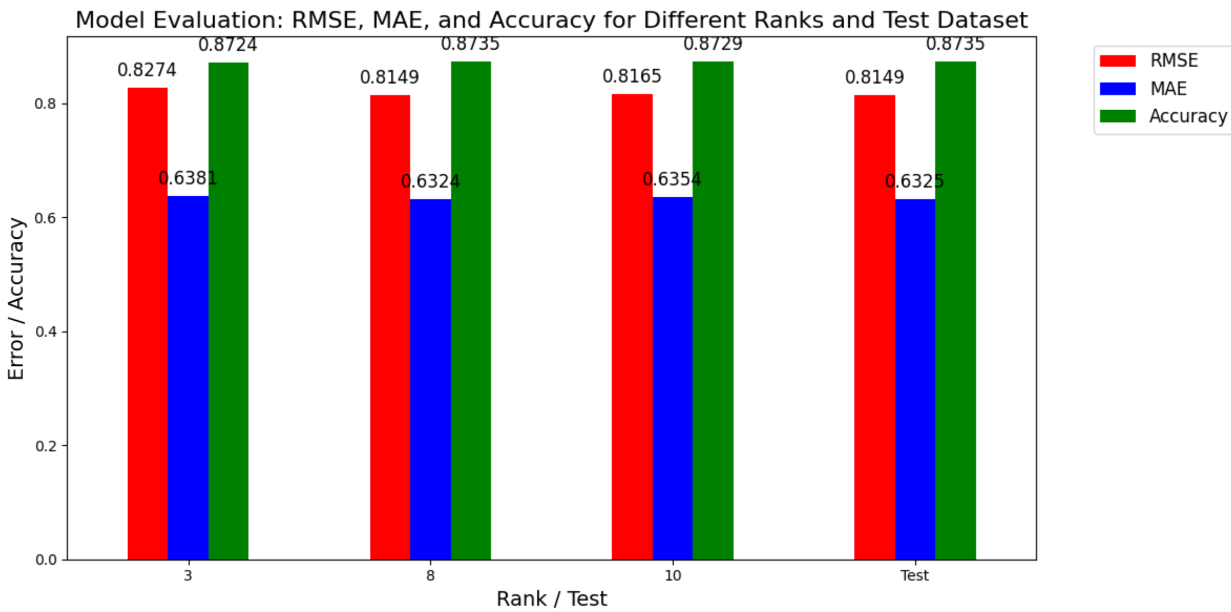



```
# Generate predictions for the test dataset using the best model
(my_model)
test_predictions = my_model.transform(test_df)
```

```python
# Filter out NaN predictions (to handle SPARK-14489 issue)
valid_test_predictions =
test_predictions.filter(test_predictions.prediction != float('nan'))

# Evaluate RMSE for the test dataset
test_rmse = reg_eval.evaluate(valid_test_predictions)
print(f"The model achieved an RMSE of {test_rmse:.4f} on the test
dataset.")

# Evaluate MAE for the test dataset
test_mae = mae_eval.evaluate(valid_test_predictions)
print(f"The model achieved an MAE of {test_mae:.4f} on the test
dataset.")

max_rating = 5
error_df['Accuracy'] = 1 - (error_df['MAE'] / max_rating)
accuracy = 1 - (test_mae / max_rating)

# Add accuracy to the error_df for display
error_df.loc['Test', 'Accuracy'] = accuracy
error_df.loc['Test', 'RMSE'] = test_rmse
error_df.loc['Test', 'MAE'] = test_mae

# Visualize RMSE, MAE, and Accuracy
fig, ax = plt.subplots(figsize=(12, 6))  # Increase figure width for a
wider view

# Plot RMSE, MAE, and Accuracy with increased bar width and spacing
error_df.plot(kind='bar', ax=ax, color=['red', 'blue', 'green'])  #
Increased width and spacing

# Title and labels
ax.set_title('Model Evaluation: RMSE, MAE, and Accuracy for Different
Ranks and Test Dataset', fontsize=16)
ax.set_xlabel('Rank / Test', fontsize=14)
ax.set_ylabel('Error / Accuracy', fontsize=14)

# Adjust the legend position to completely move it to the right
(outside the plot)
ax.legend(['RMSE', 'MAE', 'Accuracy'], loc='upper left', fontsize=12,
bbox_to_anchor=(1.05, 1))

# Display RMSE and MAE values on top of the bars
for p in ax.patches:
    ax.annotate(f'{p.get_height():.4f}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                xytext=(0, 5), textcoords='offset points',
                ha='center', va='bottom', fontsize=12)
```

```
# Adjust layout for better spacing and visibility
plt.xticks(rotation=0)
plt.tight_layout()

# Display the plot
plt.show()

# Display the updated error table with accuracy
print(error_df)
```

```
The model achieved an RMSE of 0.8149 on the test dataset.
The model achieved an MAE of 0.6325 on the test dataset.
```




```
          RMSE       MAE  Accuracy
Rank
3     0.827379  0.638127  0.872375
8     0.814857  0.632360  0.873528
10    0.816535  0.635383  0.872923
Test  0.814907  0.632535  0.873493
```

Prediction of a movie to user based on his past ratings

```
from pyspark.sql import Row

# Specify user ID for personalized ratings
my_user_id = 0

# Define movie ratings in the format (userId, movieId, rating)
# Note: The movie ID is the actual movie identifier, not the number of
ratings.
```

```
my_rated_movies = [
    (my_user_id, 318, 3),  # User rated movie 318 as 3
    (my_user_id, 1203, 5), # User rated movie 1203 as 5
    (my_user_id, 356, 5),  # User rated movie 356 as 5
    (my_user_id, 858, 2)   # User rated movie 858 as 2
]

# Create a DataFrame for the user's movie ratings
my_ratings_df = spark.createDataFrame(my_rated_movies, ['userId',
'movieId', 'rating'])

# Display the user's ratings
print("My movie ratings:")
my_ratings_df.show()
```

```
My movie ratings:
+------+-------+------+
|userId|movieId|rating|
+------+-------+------+
|     0|    318|     3|
|     0|   1203|     5|
|     0|    356|     5|
|     0|    858|     2|
+------+-------+------+
```

```
train_df= train_df.drop("timestamp")
training_with_my_ratings_df = train_df.unionAll(my_ratings_df)

als.setPredictionCol("prediction") \
   .setMaxIter(5) \
   .setSeed(4) \
   .setRegParam(0.1) \
   .setUserCol("userId") \
   .setItemCol("movieId") \
   .setRatingCol("rating") \
   .setRank(8)  # Optimal rank determined from earlier tuning

# Fit the ALS model with training data containing user-specific
ratings
my_ratings_model = als.fit(training_with_my_ratings_df)

# Generate predictions for the test dataset
predictions_with_my_ratings = my_ratings_model.transform(test_df)

# Filter out NaN predictions (if any)
predictions_cleaned =
predictions_with_my_ratings.filter(predictions_with_my_ratings.predict
ion != float('nan'))

# Evaluate the model using RMSE on the cleaned predictions
```

```
test_rmse_with_my_ratings = reg_eval.evaluate(predictions_cleaned)

# Print the RMSE result
print(f"The model achieved an RMSE of {test_rmse_with_my_ratings:.4f}
on the test set including my ratings.")

The model achieved an RMSE of 0.8165 on the test set including my
ratings.

from pyspark.sql import functions as f

# Create a list of movie IDs that I have already rated
my_rated_movie_ids = [movie[1] for movie in my_rated_movies]

# Filter out movies that I have already rated
# The '~' operator ensures these movies are excluded
unrated_movies_df =
movies_df.filter(~movies_df['movieId'].isin(my_rated_movie_ids))

# Prepare the DataFrame for prediction by renaming the column and
adding userId
prepared_unrated_movies_df = unrated_movies_df.withColumn('userId',
f.lit(my_user_id))

# Predict ratings for the movies I have not rated using the trained
model
raw_predictions_df =
my_ratings_model.transform(prepared_unrated_movies_df)

# Filter out invalid predictions (NaN values)
predicted_ratings_df =
raw_predictions_df.filter(raw_predictions_df['prediction'] !=
float('nan'))


# Display the top predictions for unrated movies
predicted_ratings_df.show(10, truncate=False)

+-------+----------------------------------
+-------------------------------------------+------+----------+
|movieId|title                             |genres
|userId|prediction|
+-------+----------------------------------
+-------------------------------------------+------+----------+
|1      |Toy Story (1995)                  |Adventure|Animation|
Children|Comedy|Fantasy|0     |3.4656773 |
|2      |Jumanji (1995)                    |Adventure|Children|Fantasy
|0     |3.3102617 |
|3      |Grumpier Old Men (1995)           |Comedy|Romance
|0     |3.2494984 |
|4      |Waiting to Exhale (1995)          |Comedy|Drama|Romance
```

```
|0      |2.9852548 |
|5       |Father of the Bride Part II (1995)|Comedy
|0      |3.341408  |
|6       |Heat (1995)                       |Action|Crime|Thriller
|0      |3.260615  |
|7       |Sabrina (1995)                    |Comedy|Romance
|0      |3.7643006 |
|8       |Tom and Huck (1995)               |Adventure|Children
|0      |3.425904  |
|9       |Sudden Death (1995)               |Action
|0      |2.4517527 |
|10      |GoldenEye (1995)                  |Action|Adventure|Thriller
|0      |3.1364052 |
+-------+----------------------------------
+--------------------------------------------+-------+-----------+
only showing top 10 rows
```