PART A: Advanced Data Analysis with PySpark or SQL Spark (35 points)

1. Count Odd and Even Numbers [Marks: 5]

- **Using the provided integer.txt file, develop a spark script to count the number of odd and even integers.**

See Integer_Count.ipynb

Result

```
# Reduce by key
result = reduceRDD.reduceByKey(lambda x, y: x + y)

#result
print(f"Even number: {result.collect()[0][1]}")
print(f"Odd number: {result.collect()[1][1]}")
```

▸ (2) Spark Jobs

```
Even number: 461
Odd number: 549
```

2. Salary Aggregation with Statistical Analysis [Marks: 10]

- **Analyze the salary.txt file to compute total salaries per department. Expand your analysis to investigate trends or discrepancies in salary distributions.**
- **Requirements:**
  § Utilize statistical measures (e.g. mean, median, standard deviation) and visualizations (e.g. box plots, histograms) to convey your findings.

See Salary_Aggregation.ipynb

statistical measures

Sales
Count: 201 Sum: 3488491 Mean: 17355.68 Median: 16698.00 Standard Deviation: 10509.41

Research
Count: 200 Sum: 3328284 Mean: 16641.42 Median: 17188.00 Standard Deviation: 9166.54

Developer
Count: 200 Sum: 3221394 Mean: 16106.97 Median: 15231.50 Standard Deviation: 9115.28

QA
Count: 200 Sum: 3360624 Mean: 16803.12 Median: 17737.50 Standard Deviation: 9830.17

Marketing
Count: 200 Sum: 3158450 Mean: 15792.25 Median: 13939.00 Standard Deviation: 9977.95

```
results = statsRDD.collect()
for dept, stats in results:
    print(f"{dept}")
    print(f"Count: {stats['count']}")
    print(f"Sum: {stats['sum']}")
    print(f"Mean: {stats['mean']:.2f}")
    print(f"Median: {stats['median']:.2f}")
    print(f"Standard Deviation: {stats['stddev']:.2f}")
    print("---------------")
```

▸ (1) Spark Jobs

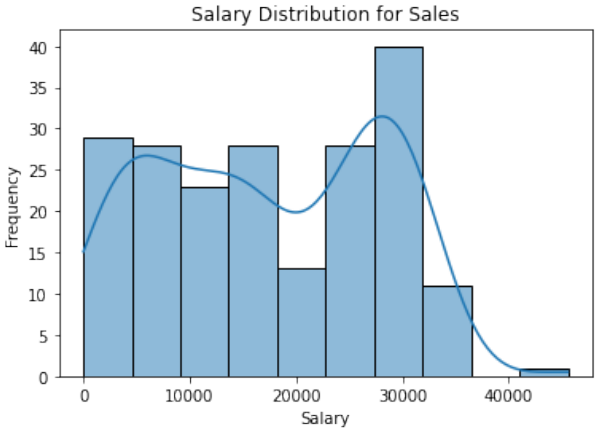
```
Sales
Count: 201
Sum: 3488491
Mean: 17355.68
Median: 16698.00
Standard Deviation: 10509.41
---------------
Research
Count: 200
Sum: 3328284
Mean: 16641.42
Median: 17188.00
Standard Deviation: 9166.54
---------------
Developer
Count: 200
Sum: 3221394
Mean: 16106.97
Median: 15231.50
Standard Deviation: 9115.28
---------------
```

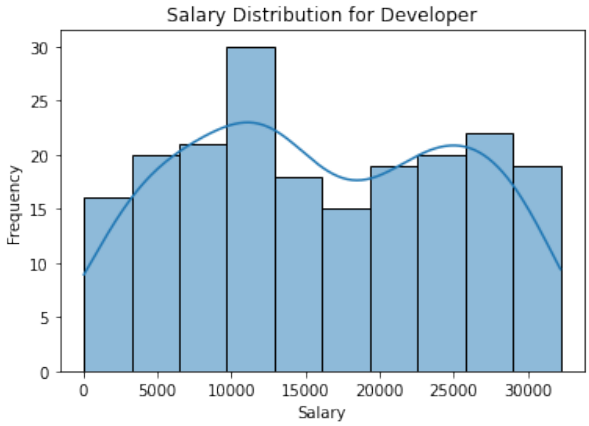
```
    print(f"Mean: {stats['mean']:.2f}")
    print(f"Median: {stats['median']:.2f}")
    print(f"Standard Deviation: {stats['stddev']:.2f}")
    print("---------------")
```

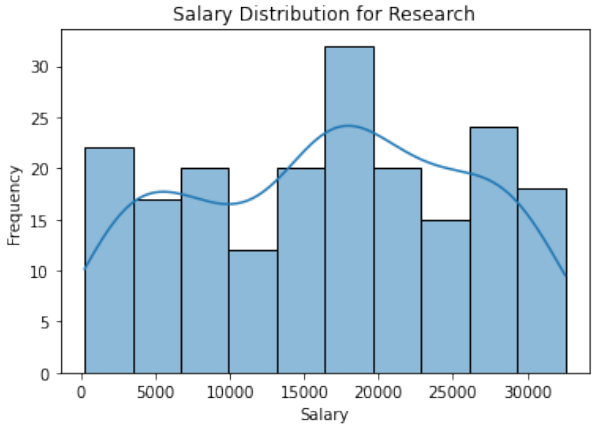
▸ (1) Spark Jobs

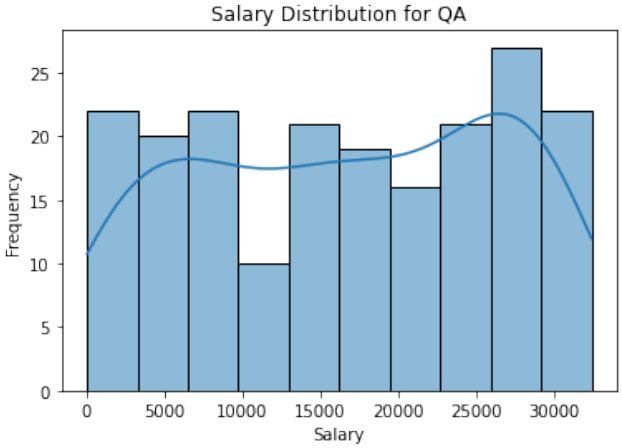
```
Developer
Count: 200
Sum: 3221394
Mean: 16106.97
Median: 15231.50
Standard Deviation: 9115.28
---------------
QA
Count: 200
Sum: 3360624
Mean: 16803.12
Median: 17737.50
Standard Deviation: 9830.17
---------------
Marketing
Count: 200
Sum: 3158450
Mean: 15792.25
Median: 13939.00
Standard Deviation: 9977.95
---------------
```

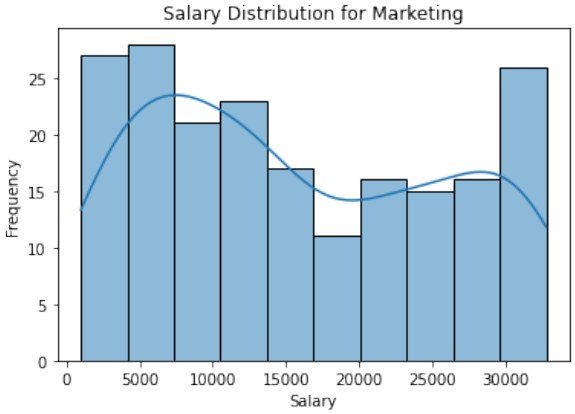
Graph (box plots, histograms)











3. Implement an Optimized MapReduce [Marks: 10]

- **Utilize the shakespeare.txt file to implement an optimized MapReduce operation that**
- **counts specific terms, allowing for case-insensitivity and punctuation removal.**
- **Requirements:**
  § Show how many times these particular words appear in the document: Shakespeare, What, The, Lord, Library, GUTENBERG, WILLIAM, COLLEGE and WORLD. (Count exact words only)

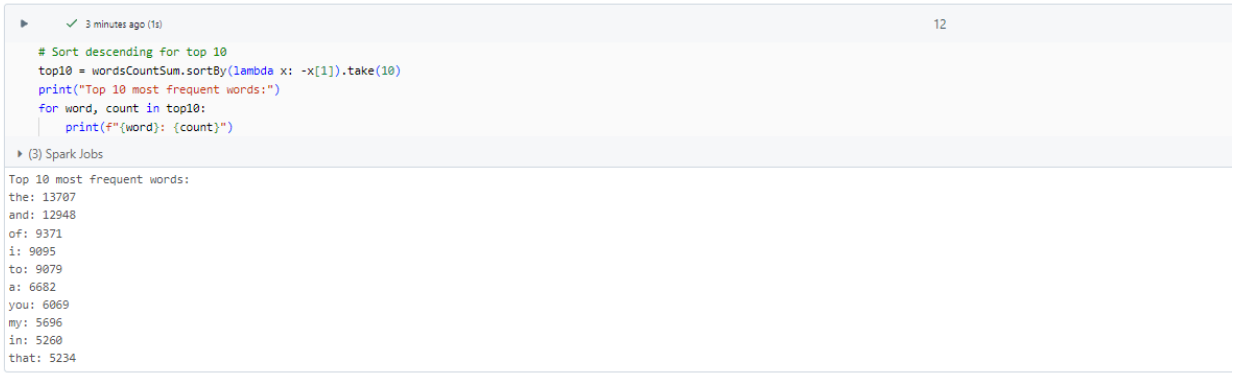
See Shakespeare_MapReduce_Frequency_Distribuition_Analysis.ipynb

4. Word Frequency and Distribution Analysis [Marks: 10]

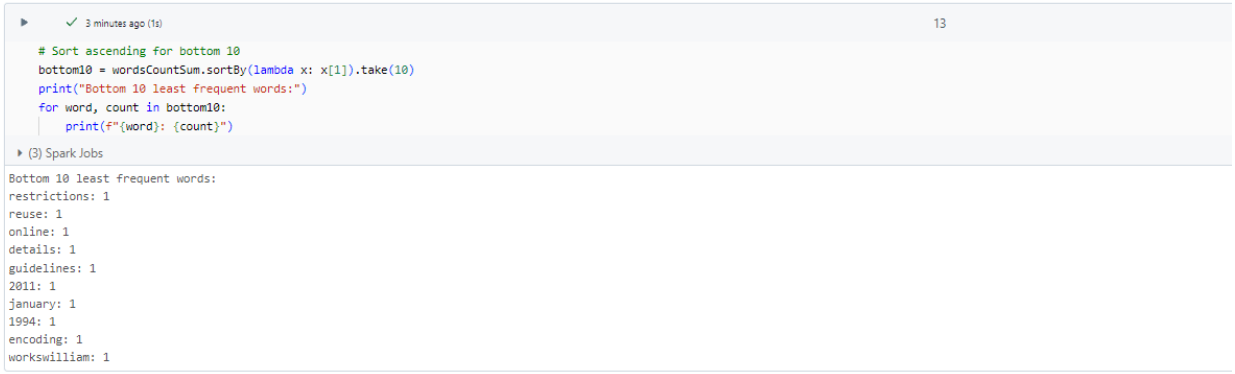
- **From shakespeare.txt, calculate top 10 and bottom 10 words. Show 10 words with most count and 10 words with least count.**

See Shakespeare_MapReduce_Frequency_Distribuition_Analysis.ipynb (Cont')

Top and bottom10 words



```
✓ 3 minutes ago (1s)                                                    12

# Sort descending for top 10
top10 = wordsCountSum.sortBy(lambda x: -x[1]).take(10)
print("Top 10 most frequent words:")
for word, count in top10:
    print(f"{word}: {count}")

▸ (3) Spark Jobs

Top 10 most frequent words:
the: 13707
and: 12948
of: 9371
i: 9095
to: 9079
a: 6682
you: 6069
my: 5696
in: 5260
that: 5234
```



```
✓ 3 minutes ago (1s)                                                    13

# Sort ascending for bottom 10
bottom10 = wordsCountSum.sortBy(lambda x: x[1]).take(10)
print("Bottom 10 least frequent words:")
for word, count in bottom10:
    print(f"{word}: {count}")

▸ (3) Spark Jobs

Bottom 10 least frequent words:
restrictions: 1
reuse: 1
online: 1
details: 1
guidelines: 1
2011: 1
january: 1
1994: 1
encoding: 1
workswilliam: 1
```

PART B: Advanced Recommender System with Apache Spark (65 points)

The objective of this part is to develop a sophisticated distributed recommender system using Apache Spark's capabilities, emphasizing accuracy, efficiency, and comparative evaluations.

Data Input: Utilize the provided movies.csv dataset, available for download from Quercus.

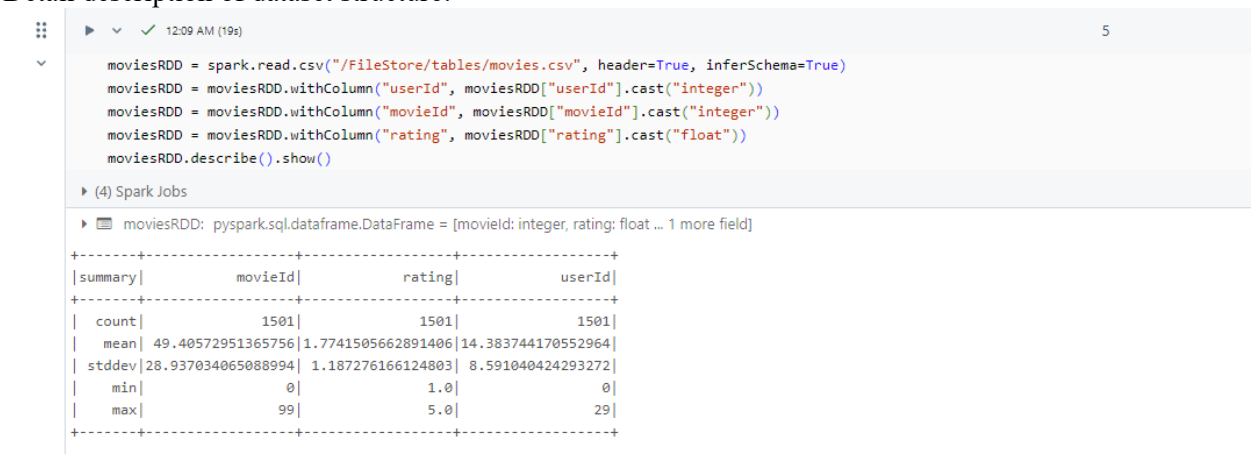
Implementation Steps: Load the dataset and import the necessary libraries. Address the below questions, applying advanced techniques and theoretical principles.

See Advanced_Recommender_System_with_Apache_Spark.ipynb

1. Data Description and Insights Analysis [Marks: 10]

- **Description: Provide a detailed description of the dataset's structure and contents. Analyze the distribution of ratings across each movie and identify the top 10 movies with the highest average ratings.**
- **Requirements:**
  § Identify the top 10 users who have contributed the most ratings (not just high ratings) and discuss their influence on the dataset.
  § Conduct exploratory data analysis, such as visualizing the distribution of ratings, to uncover patterns in user behaviors (e.g., how many ratings each user provides) and preferences.
  § Discuss any potential implications for marketing strategies based on user engagement and rating tendencies.

Detail description of dataset structure:



Top 10 user contributed to the most ratings;(Next pape)
They can shape the predominant distribution of ratings and have potential influence on the model's learning pattern.

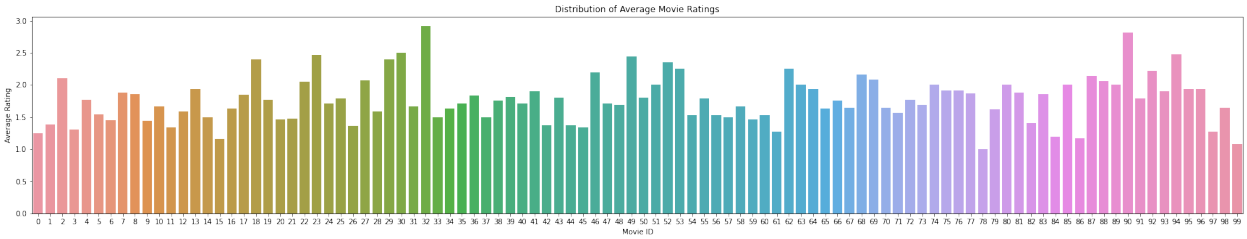
```
12:09 AM (2s)

# find the top user that has most rating
user_counts = moviesRDD.groupBy("userId").agg(count("*").alias("rating_count"))

user_counts.orderBy("rating_count", ascending=False).show()
```

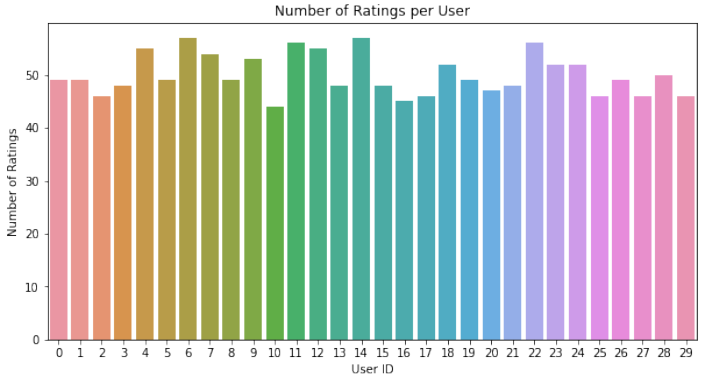
(2) Spark Jobs

user_counts: pyspark.sql.dataframe.DataFrame = [userId: integer, rating_count: long]

userId	rating_count
6	57
14	57
22	56
11	56
12	55
4	55
7	54
9	53
23	52
24	52
18	52
28	50
26	49
1	49
5	49
19	49
8	49

Exploratory analysis





Potential implications for marketing strategies based on user engagement and rating tendencies

For users that are active and contributed to more ratings can be targeted with more personalized recommendations and offer for more ratings as an approach to maintain their engagement. Meanwhile, it is meaningful to understand the patters to help promote the products to wider and potential users.

2. Split Dataset and Performance Assessment [Marks: 10]

- **Experiment: Split the dataset into training and testing subsets using 2 different ratios (for e.g. 60/40, 70/30, 75/25, and 80/20). Implement stratified sampling to ensure users are represented proportionally across the splits.**
- **Requirements:**
  § Report on how different splits influence the performance of your collaborative filtering model (you can use one of the evaluation metrics to show this).
  § Represent the performance variation of the model based on each split ratio, identifying the most effective configuration based on empirical findings.

Used 70/30 and 80/20 separation, I will use MSE, RMSE, MAE metrics, the performance of the collaborative filtering model changes slightly with different train-test splits.

```
Yesterday (24s)

# Model for 70/30
model_70_30 = als.fit(training_70)
predictions_70_30 = model_70_30.transform(testing_30)
predictions_70_30.show()
```

```
Yesterday (19s)

# Model for 80/20
model_80_20 = als.fit(training_80)
predictions_80_20 = model_80_20.transform(testing_20)
predictions_80_20.show()

(5) Spark Jobs
==========
rmse on 70/30 split: 1.0309514632232468
rmse on 80/20  split: 1.0098783605804091
==========
mae on 70/30 split: 0.752810019089879
mae on 80/20  split: 0.7218527526525822
==========
mse on 70/30 split: 1.0628609195221534
mse on 80/20  split: 1.0198543031685747
==========
```

Based on RMSE, MAE , MSE, the 80/20 split achieved a lower error compared to the 70/30 split as usually a better prediction accuracy could be achieved when more data for training is available. Thus, to reach the better overall performance, allocating more data for training is of great significance.

Based on these empirical results, the 80/20 split appears to be the more effective configuration for this dataset.

3. In-Depth Evaluation of Error Metrics [Marks: 10]

- **Metrics: Define and explain key metrics for evaluation: MSE, RMSE and MAE. Introduce advanced metrics like Precision, Recall, and F1 Score specifically tailored for recommendations.**
- **Requirements:**

§ Provide a detailed evaluation of each models performance using these metrics, discussing the strengths and weaknesses of each in the context of a recommendation system focused solely on user ratings.
§ Make observations about the trade-offs involved when selecting different metrics, particularly in scenarios of sparse data or imbalanced ratings.

MSE, RMSE and MAE are illustrated in previous questions.

```
#function to output performance metrics
def rank_eval(name, rank, k):
    ranking_evaluator = RankingEvaluator(labelCol="actual_items", predictionCol="predicted_items", k=k)

    precision = ranking_evaluator.setMetricName("precisionAtK").evaluate(rank)
    recall = ranking_evaluator.setMetricName("recallAtK").evaluate(rank)
    f1 = 2 * (precision * recall) / (precision + recall)

    print("-" * 10)
    print(f"Performance on {name}")
    print(f"Precision@{k}: {precision:.4f}")
    print(f"Recall@{k}: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("-" * 10)
```

```
✓ 23 hours ago (6s)                                                    22
rank_eval("70-30 Training Test Spilit", rank_70, 10)
▸ (17) Spark Jobs
----------
Performance on 70-30 Training Test Spilit
Precision@10: 0.9833
Recall@10: 0.7220
F1 Score: 0.8327
----------
```

```
✓ 23 hours ago (5s)                                                    23
rank_eval("80-20 Training Test Spilit", rank_80, 10)
▸ (19) Spark Jobs
----------
Performance on 80-20 Training Test Spilit
Precision@10: 0.8933
Recall@10: 0.9029
F1 Score: 0.8981
----------
```

- In the 70/30 split, the model achieved higher precision (0.9833) while it has lower recall (0.7220), this suggests that it tends to recommend highly relevant items but would misses some potential recommendations.
- The 80/20 split showed slightly lower precision (0.8933) but higher recall (0.9029), indicating better coverage of relevant items resulted from the existence of more training data.
- The F1 score was higher in the 80/20 split compared to 70/30 split, showing a better overall balance between precision and recall.

Trade-off:

Precision focuses on recommending a smaller set of highly accurate items, it will minimize those irrelevant recommendations, on the other hand, recall would more focus on the more relevant items. Thus, when training the model, optimizing the precision would reduce the recommendation diversity while, optimizing the precision would reduce the relevance of the recommendations. F1 Score is a balanced measurement between precision and recall.

4. Hyperparameter Tuning Using Cross-Validation Techniques [Marks: 20]

- **Tuning: Conduct systematic hyperparameter tuning for at least two parameters of the collaborative**
- **filtering algorithm, such as rank, regularization, or iterations etc. utilizing methods like grid search or randomized search combined with cross-validation.**
- **Requirements:**
  §Visualize the impact of different hyperparameter configurations on model performance (e.g., varying RMSE scores) and provide a rationale for your tuning choices based on your findings.
  §Discuss how each parameter affects model performance and overall training time, including insights on how to balance complexity against performance.

Yesterday (<1s)

```
# parameter grid
paramGrid = ParamGridBuilder() \
    .addGrid(als.rank, [5, 10, 15]) \
    .addGrid(als.regParam, [0.01, 0.1, 1.0]) \
    .addGrid(als.maxIter, [5, 10]) \
    .build()

#build cross validator
crossval = CrossValidator(estimator=als,
                          estimatorParamMaps=paramGrid,
                          evaluator=eval_cv,
                          numFolds=3)
```

(5) Spark Jobs

```
----------
Best Rank: 15
Best RegParam: 0.1
Best MaxIter: 10
----------
```

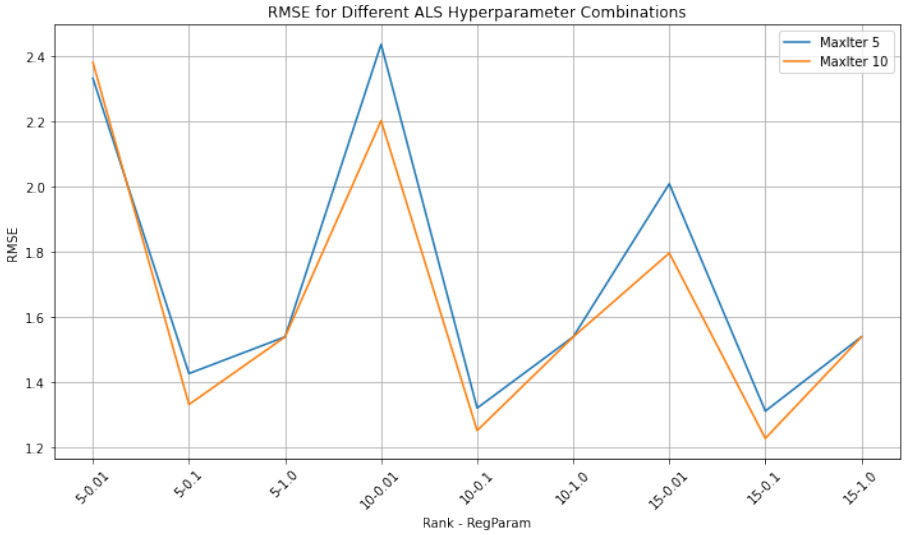
Yesterday (3s)

```
#test on testing set
final_predictions = bestModel.transform(testing_30)
final_rmse = eval_cv.evaluate(final_predictions)
print("Test RMSE on best model:", final_rmse)
print("-"*10)
```

(7) Spark Jobs

final_predictions: pyspark.sql.dataframe.DataFrame = [movieId: integer, rating: float ... 2 more fields]

```
Test RMSE on best model: 1.0011152605208538
----------
```



Tuning Rank and regParam

Rationale for tuning choice:
Both rank and regParam have significant affect the models RMSE. Lower values of regParam (0.1) combined with higher rank values generally led to better performance, with the best RMSE when setting rank=15, regParam=0.1, and maxIter=10, these values setting could help ALS model better capturing user-rating-movie relationships.

How each parameter affects model performance and overall training time:

Rank: Higher rank allows the model to capture more complex latent features, it can potentially improve the recommendation accuracy, but meanwhile, it will also increase model complexity and training time.
MaxIter: More iterations could let the model converge to better solutions; however, after a certain iteration, it will not provide any more improvement but extending training time.
RegParam: Lower regularization could values help the model fit the data more closely and improve the accuracy, but it can increase the risk of overfitting; higher values reduce overfitting but may underfit the data.

5. Personalized Recommendations and Analysis for Selected Users [Marks: 15]

- **Recommendations: Generate personalized movie recommendations for user IDs 11 and 21 based on their rating preferences.**
- **Requirements:**
  § Discuss how the collaborative filtering approach utilizes user ratings to generate these recommendations and the effectiveness of this technique for a dataset with limited features.
  § Compare performance outcomes between the refined recommendations for these users and any baseline recommendations generated earlier in your analysis. Discuss potential enhancements or features that could be added for improved personalization in future iterations

```
✓ 24 hours ago (10s)

#user 21
generate_recommendation(recommender_system, 21)

▸ (2) Spark Jobs

Top recommendations for user 21:
+-------+---------+
|movieId|   rating|
+-------+---------+
|     53| 4.348518|
|     87| 3.068904|
|     70|2.7150736|
|     58|2.6514935|
|     93|2.6494384|
|     77|2.6446126|
|     96|2.5969944|
|     52|2.5543914|
|     29|2.5361338|
|     63| 2.471595|
+-------+---------+
```

```
✓ 24 hours ago (10s)

#user 11
generate_recommendation(recommender_system, 11)

▸ (2) Spark Jobs

Top recommendations for user 11:
+-------+---------+
|movieId|   rating|
+-------+---------+
|     32|  4.71984|
|     27|4.6241703|
|     18|4.6194463|
|     23| 4.517104|
|     69|4.4659357|
|     79| 4.299311|
|     19| 3.686703|
|     30|3.6323361|
|     13|3.5621607|
|     66|  3.54922|
+-------+---------+
```

The collaborative filtering approach uses historical user ratings for the movies to identify patterns of similarity between users and movies; the system would tend to recommend movie to distinct users as similar users show interest (high ratings) while the target user has not interacted with. Despite the dataset having limited features, the ALS model could still generate the personalized recommendations.

Compared to baseline models without hyperparameter tuning, the model after tunning illustrates a better prediction score in RMSE as it will have a more accurate recommendation.

Potential enhancements or features that could be added for improved personalization in future iterations. More features like movie timestamps, user geographical locations, or demographics information could be added to the model for training so that model could capture more contextual information and have better recommendation diversity and relevance.