Assignment 2 Report

Part A Q1:

First, I uploaded the file integer.txt to /FileStore/tables/ in databricks. And then read the file to rdd. I converted each line to an integer and check parity. Then use aggregate operations to count odd and even numbers.

Below is the result, we can see that the numbers of odd numbers and even numbers are closed. It means no strong bias in the integer.txt.

```
Even numbers count: 514
Odd numbers count: 496
```

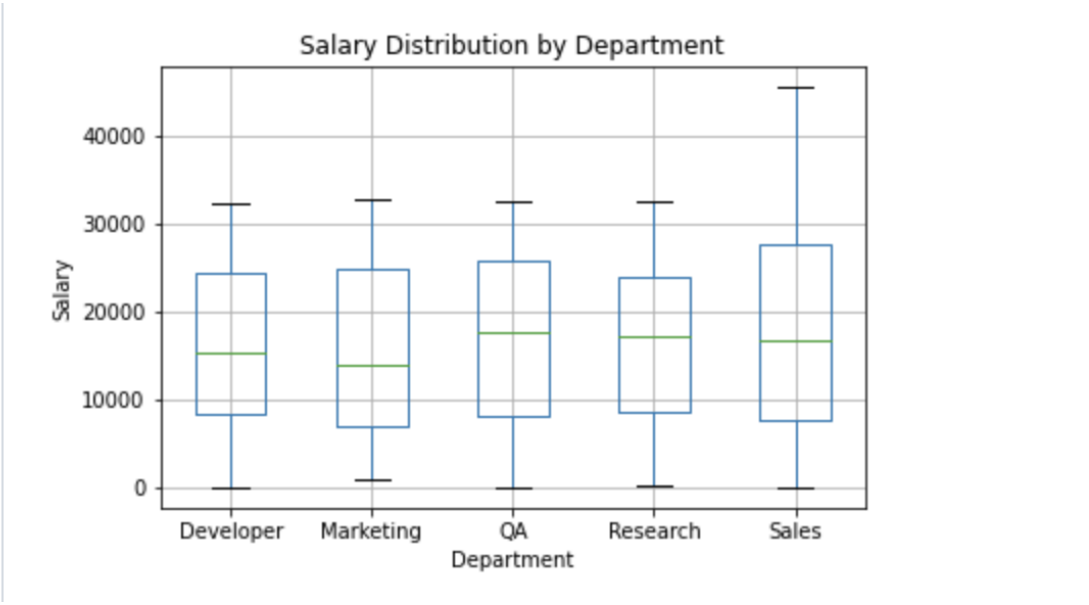
Part A Q2:

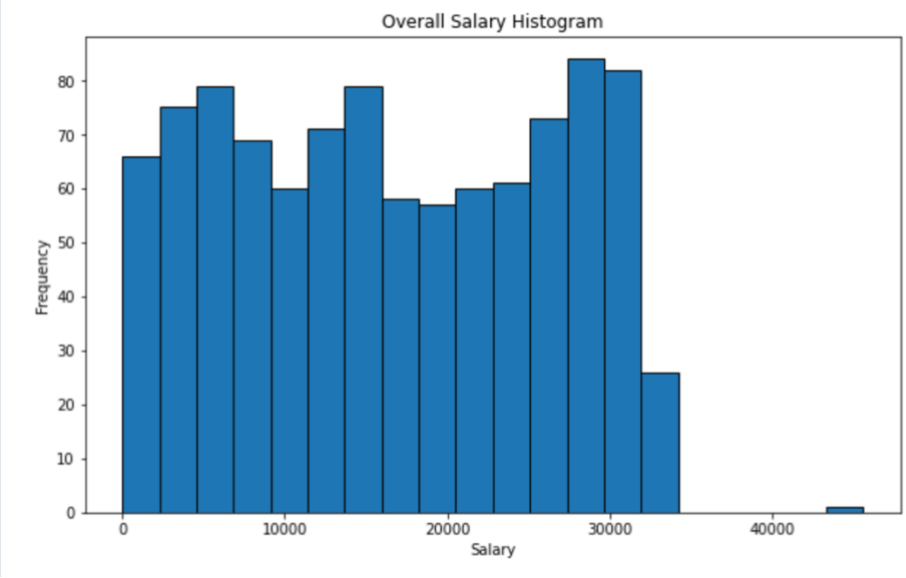
First, I uploaded the file salary.txt to /FileStore/tables/ in databricks. And then load the file using Sparkcontent and process the salary number. Then, I converted RDD to dataframe with columns " department" and " salary". I use pyspark.sql.function to calculate the mean, standard deviation, median, minimum and maximum salary per department. Below is the result:

Table

	department	mean_salary	stddev_salary	min_salary	max_salary	median_salary
1	Sales	17355.67661691542	10535.654471835174	14	45567	16292
2	Developer	16106.97	9138.15811076756	39	32220	14371
3	Research	16641.42	9189.53997971574	246	32542	17004
4	Marketing	15792.25	10002.990973181359	990	32758	13537
5	QA	16803.12	9854.836040366808	21	32429	17145

For the visualization, I converted spark dataframe to pandas dataframe so that I can do the plotting. Below are the box plot for salary distribution by department and the histogram for overall salary distribution.





From above analysis, we can see that sales department has the highest mean salary while marketing department has the lowest mean salary. And the sales department can reach the much higher salary than others sometimes.

Part A Q3

First, I uploaded the file sharesphere_1.txt to /FileStore/tables/ in databricks. And then read the file to define the target words. For convenience and uniformity, I made it all low-case. And then I use the MapReduce to count the target words. Below is the result:

```
gutenberg: 106
shakespeare: 123
world: 439
library: 104
college: 101
the: 13707
william: 169
what: 1969
lord: 1528
```

Part A Q4

First, I read the file using sparkcontent and then define a function to remove punctuation and convert to low-case. Then I create a RDD including all words. Then counting the occurrence of each word. Below are the top 10 words and the bottom words:

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

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

We can see that the top 10 words are some common words such as the, and. For the bottom 10 words, it includes some numbers.

PartB Q1

First, I loaded the movis.csv to /FileStore/tables/. And then inspect the schema and show first few rows so that we can see the structure and content of the data. Then I use pyspark.sql.function to calculate the average rating and count of rating for each movie. Below are the tables showing top 10movies with highest average ratings and top 10 users with the highest number of ratings.

Table

	movieId	avg_rating	num_ratings
1	32	2.9166666666666665	12
2	90	2.8125	16
3	30	2.5	14
4	94	2.473684210526316	19
5	23	2.466666666666667	15
6	49	2.4375	16
7	29	2.4	20
8	18	2.4	15
9	52	2.357142857142857	14
10	53	2.25	12

10 rows

```
Above are top 10 movies with the highest average ratings
```

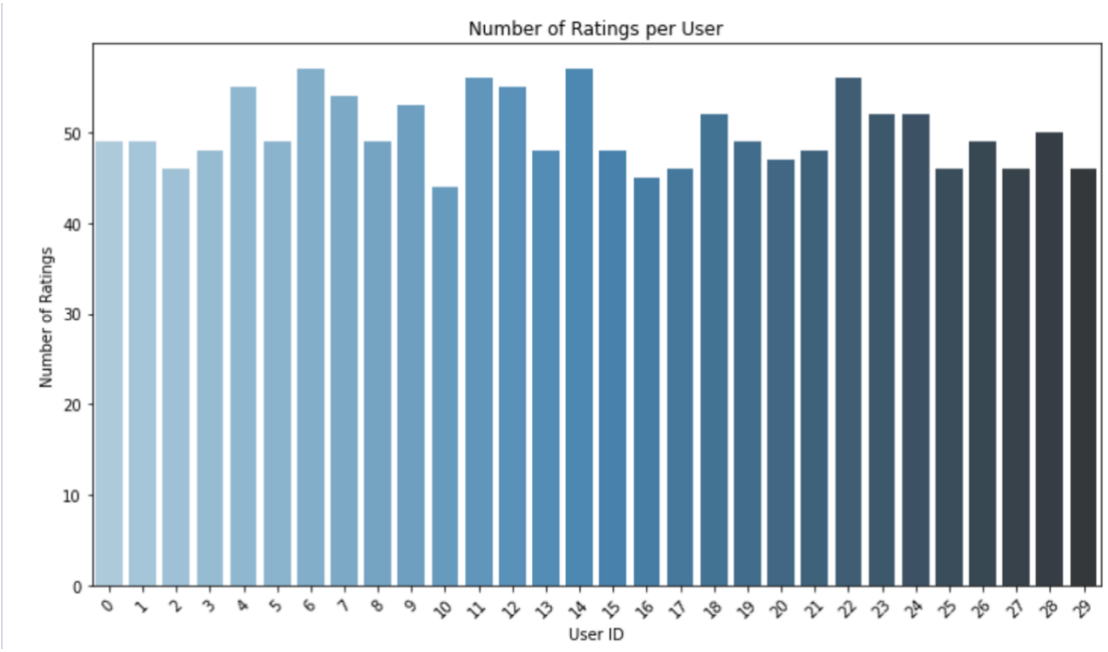
Table

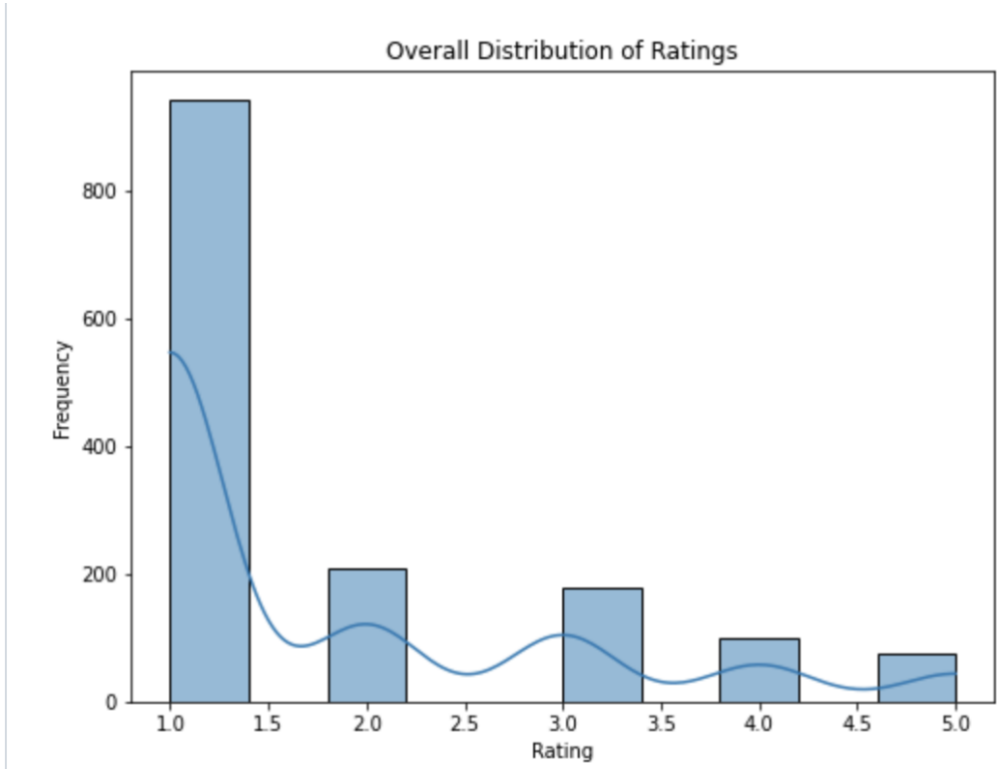
	userId	rating_count
1	6	57
2	14	57
3	22	56
4	11	56
5	12	55
6	4	55
7	7	54
8	9	53
9	23	52
10	24	52

10 rows

```
Above are top 10 users with the highest number of ratings
```

For the visualization, I used matplotlib to show how many ratings each user provides and overall distribution of ratings. Below are the results.





From the table 1, we can see that there are 3 headers in this movie dataset. They are movieId, rating and userId, which are integer. Table 2 shows top 10 movies with highest rating and table 3 shows top 10 user with most frequent rating. From the visualization of number of ratings per user, we can see that all users rated over 40 times. And there are 11 users rated over 50 times. From the visualization of overall distribution of ratings, we can see that the number of rating 1.0 is most rated by users while rating 5.0 is the least.

For the potential implications of marketing strategies, because the users all were actived and engaged, we can offer more services to the users such as group discussion about movie or other paid

service. In addtion to this, we can see that rate 1.0 are more than rate 5.0. We can ask the users why the movie is rate 1.0 to help develop better movie service. And we can also make personalized targeting. We can learn what categories are highly rated by each user, and then recommend the same or similar category of movies to them.

Part B Q2

First, I defined a list of split ratios [0.6, 0.7, 0.75, 0.8]. Then splitting the data using randomSplit into training and test set based on the ratios. And using ALS model with chosen hyperparameters. Last, I train the model on the training split using als.fit and generate predictions on the test set using transform. And then calculate RMSE for each ratio. Below is the result.

```
Training/Test Ratio 60/40 RMSE: 1.0901
Training/Test Ratio 70/30 RMSE: 1.0780
Training/Test Ratio 75/25 RMSE: 0.9704
Training/Test Ratio 80/20 RMSE: 0.9275
```

From above table, we can see that as the training set increases, the RMSE on test set decreases. It means the model is learning more effectively with additional training samples. Among the above tested ratios, because the 80/20 split has the lowest RMSE (0.9275), i think it is the most effective split configuration for this dataset.

Part B Q3

I am using the 80/20 split in this step because it shows the least RMSE on last step. And then computing the MAE, MSE. I defined a threshold which is ratings>=3.0 is consider relevant. And then computing Precision, Recall and F1 score. Below is the result.

```
RMSE: 0.9275246242850463
MAE: 0.6126140397648479
MSE: 0.8603019286551163
Precision: 0.75
Recall: 0.14516129032258066
F1 Score: 0.24324324324324328
```

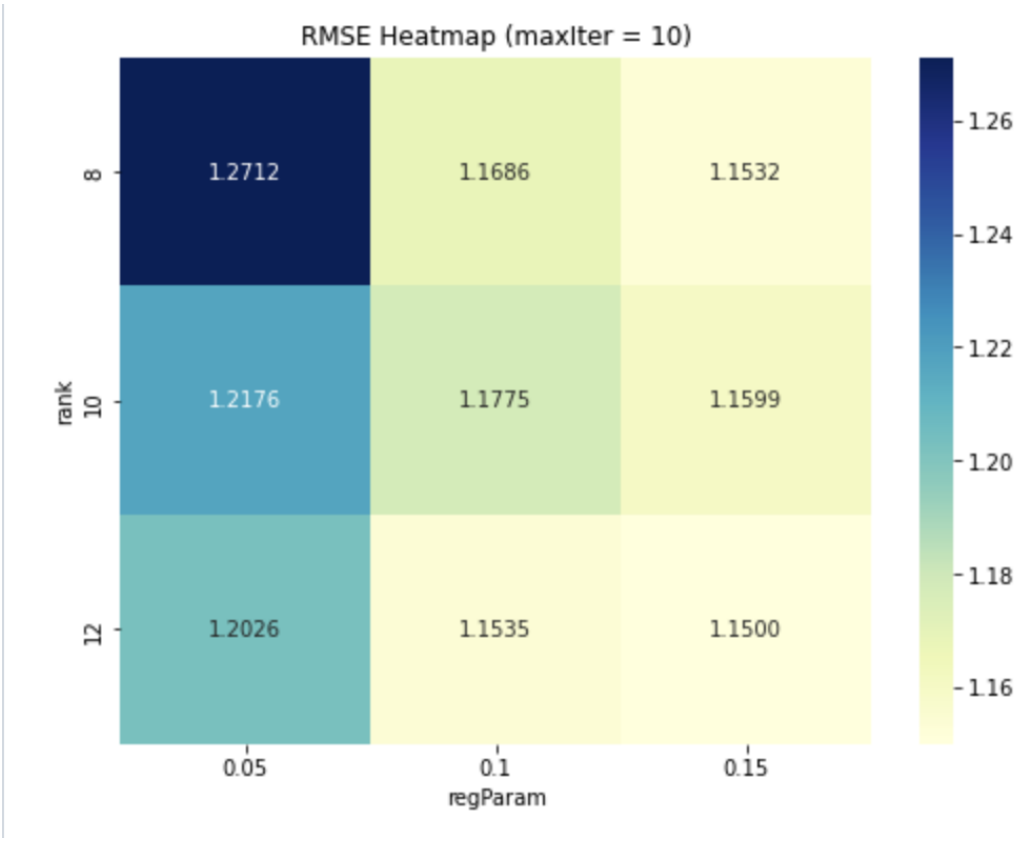
From above result, the RMSE is 0.9275. A lower RMSE indicates the model's predicted ratings are relatively close to actual ratings on average. The MAE is 0.6126. It is more robust to outliers than RMSE because large errors are not squared. The MSE is 0.8603. MSE is the average of the squared errors and is the square of the RMSE. The precision is 0.75, which means all items the model predicted as "liked" (≥ 3.0), 75% were truly liked by the user. The recall is 0.1452, which means in the items that were truly liked by the user, the model only identifies about 14.5% of them. It means the users may miss many movies. The F1 score is 0.2432. F1 score is harmonic mean of Precision and Recall. For this recommendation system, it relies solely on user ratings. RMSE, MAE, and MSE

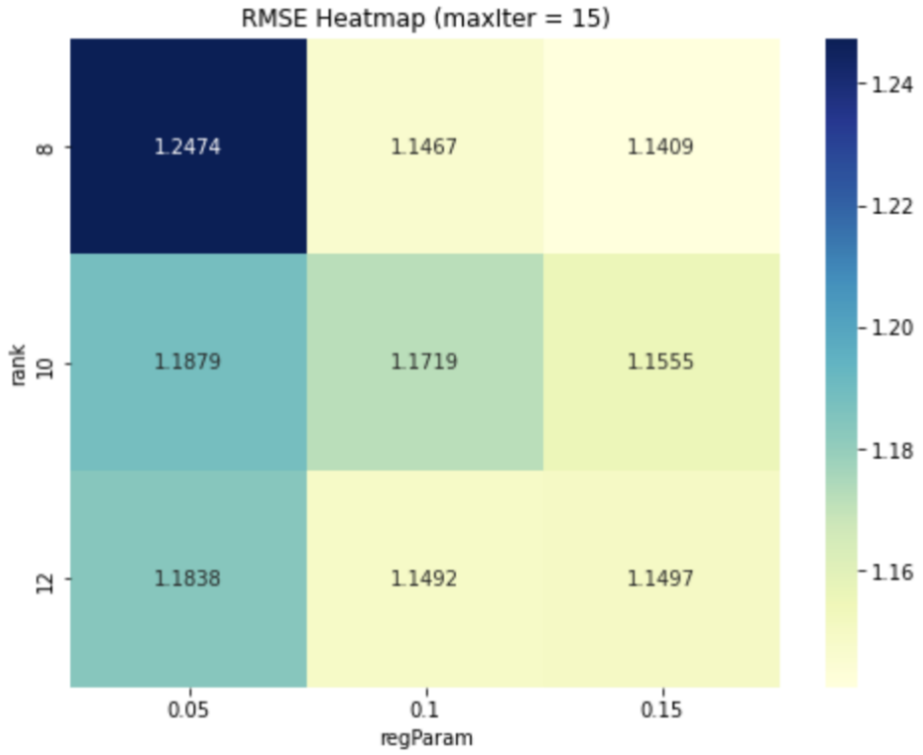
are regression metrics measure how close predicted ratings are to the actual ones, but they don't show whether users truly "like" a movie. Classification metrics: Precision, Recall, and F1, which are better for assessing how well the recommended list captures user preferences, especially under a binary rule like "rating ≥ 3.0." In imbalanced datasets, Recall often ends up low, meaning the system may miss many items users would actually like. If broaden recommendations to boost Recall, Precision tends to drop. Ultimately, which metrics we prioritize depends on our goals: if we want to accurately predict the exact rating, focus on lowering RMSE or MAE; if we want users to see as many liked items as possible, raise Recall but also keep an eye on Precision to balance coverage and accuracy.

Part B Q4

In this step, I built the parameter grid for tuning: als.rank [8, 10, 12], als.regParam [0.05, 0.1, 0.15], als.maxIter[10, 15]. And then set up 3-fold cross-validation with the grid. Last, I retrieve the best model and evaluate it on the test set. Below is the result and visualization.

```
Best RMSE from Cross-Validation: 0.9667
    rank  regParam  maxIter      rmse
0      8      0.05       10  1.271221
1      8      0.05       15  1.247408
2      8      0.10       10  1.168558
3      8      0.10       15  1.146662
4      8      0.15       10  1.153218
5      8      0.15       15  1.140907
6     10      0.05       10  1.217581
7     10      0.05       15  1.187946
8     10      0.10       10  1.177480
9     10      0.10       15  1.171860
10    10      0.15       10  1.159906
11    10      0.15       15  1.155503
12    12      0.05       10  1.202587
13    12      0.05       15  1.183751
14    12      0.10       10  1.153493
15    12      0.10       15  1.149208
16    12      0.15       10  1.149953
17    12      0.15       15  1.149677
```





From the first heatmap (maxIter=15), we can see that when rank=8 and regParam=0.05, the RMSE is as high as 1.2474. As regParam increases to 0.15, the RMSE drops to 1.1409, indicating that appropriately increasing regularization helps alleviate overfitting and improves accuracy. When

rank=10 or rank=12, the RMSE ranges roughly between 1.18 and 1.14, and setting regParam to 0.1 or 0.15 yields a relatively low RMSE (around 1.15). Comparing this with the second heatmap (maxIter=10), under fewer iterations, if rank=8 and regParam=0.05, the RMSE rises to 1.2712, which is worse. However, with rank=12 and regParam=0.1 or 0.15, the RMSE can be kept near 1.15, suggesting that increasing rank along with moderate regularization can maintain decent accuracy under fewer iterations—though it's still slightly higher than the best RMSE (about 1.14) achieved when maxIter=15. Overall, maxIter=15 allows the model to converge further, especially with a higher rank and moderate regParam, achieving a lower RMSE but at a higher training cost. If you need to balance training time and accuracy, you might consider reducing the number of iterations or choosing a moderate rank and moderate regularization to find an optimal trade-off between performance and efficiency.

PartB Q5

First, I created a usersID' s Dataframe and use the best model in step4 to recommend 5 movies for user 11 and user 21. For the base line, I just pick the top 5 movies with highest average rating.

Below is the recommendation using best model:

	userId	recommendations
1	21	array 0: {"movieId": 29, "rating": 3.9365473} 1: {"movieId": 52, "rating": 3.6982603} 2: {"movieId": 53, "rating": 3.4719462} 3: {"movieId": 63, "rating": 3.2726026} 4: {"movieId": 76, "rating": 3.2282004}
2	11	array 0: {"movieId": 30, "rating": 4.500776} 1: {"movieId": 18, "rating": 4.47132} 2: {"movieId": 27, "rating": 4.356862} 3: {"movieId": 23, "rating": 4.168791} 4: {"movieId": 32, "rating": 3.9722664}

Below is the baseline recommendation:

	movieId	avg_rating	num_ratings
1	32	2.9166666666666665	12
2	90	2.8125	16
3	30	2.5	14
4	94	2.473684210526316	19
5	23	2.466666666666667	15

Collaborative filtering can generate personalized recommendations by leveraging user ratings to find patterns and similarities between users. It works well in identifying niche preferences but faces challenges in data-sparse situations or new users and items. Enhancing collaborative filtering with additional features like movie genres, temporal dynamics, or contextual data can improve

personalization, while incorporating better ranking metrics can refine future iterations for even more effective recommendations. While Collaborative filtering provides more personalized suggestions compared to the method we analysis before such as just recommending the most popular movies, it may struggle in sparse datasets or imbalanced ratings.