CMP3749 BIG DATA ASSIGNMENT

Word Count: Task 1: 923, Task 2: 903, Task 3: 652

Vishal Maisuria - 26439978

26439978@students.lincoln.ac.uk

Contents

1. Task 1 - Pyspark Implementation of Recommendation System 923

The objective of task 1 was to develop a recommendation system, where I chose to implement collaborative filtering. This involved analysing movie rating data to predict user preferences. The core tool used for this task was Pyspark, a powerful library that handles large-scale data efficiently.

1.1. Design and Data Preparation

The first step was to load the data from the CSV file into a Spark DataFrame, ensuring they adhered to a schema that I predefined for consistency and ease of manipulation. After loading the data, I performed data cleaning operations to remove any null values, to ensure data quality and duplicates, to maintain data integrity and avoid skewing the model. To do this, I used PySpark's "dropna()" and "dropDuplicates()" functions. This resulted in an efficient preparation of datasets, considering that they were fairly large. Ratings had 1 duplicate, and Trust had none. Neither dataframe contained any null values. Doing this ensured the integrity of the data, and as a result crucial for the success of the model, as data of a poor quality can lead to distorted results.

```
35497 1853
Original ratings count:  35497
New ratings count after removing nulls:  35497
Original trust count:  1853
New trust count after removing nulls:  1853
Ratings count before removing duplicates:  35497
Ratings count after removing duplicates: 35496 (1)
Trust count before removing duplicates:  1853
Trust count after removing duplicates: 1853 (0)
```

Figure 1: Data Preparation

1.2. Data Analysis

To further understand the trust dataset I was working with, I conducted analysis of the data. This involved calculating the count, mean, standard deviation, as well as the quartiles.

- Most ratings are clustered between 3.0 and 4.0, with 4.0 having the most frequency (Figure 18). This shows that users prefer rating movies highly.
- The top 10 users, contributed many ratings of movies. Users 272 and 1187 contributed the most ratings (Figure 15). This could be due to their high activity, and could be noted for further marketing purporses.

- The top 10 movies are movies that received the most ratings, with movie 7 receiving the most ratings (Figure 16). Other movies such as 2 and 11 also recevied many ratings, but this does not determine whether they are positive or negative ratings. This could be used to determine the most liked or most disliked movies, and could be used for marketing purposes.

In the trust dataset, there was a mean Trustor of 775.4 and a mean Trustee of 782.2, with a standard deviation of approximately 447.7 for Trustor and 471.6 for Trustee. This shows that the trust values are spread out, and signifies a wide range of users involved. All trust values were uniformly 1.0, (Figure 17) which indicates a binary trust with no variability.

Trust Summary:

	Trustor	Trustee	Trust_Value
count	1853.000000	1853.000000	1853.0
mean	775.437669	782.192121	1.0
std	447.736536	471.614626	0.0
min	2.000000	2.000000	1.0
25%	410.000000	395.000000	1.0
50%	752.000000	716.000000	1.0
75%	1160.000000	1187.000000	1.0
max	1641.000000	1642.000000	1.0

Figure 2: Ratings Data Analysis

1.3. Training and Testing

The data was split into training and testing sets using an 80/20 split ratio, which ensured that the model was trained on a majority of the data and tested on a smaller subset to evaluate its performance. To do this, I utilised the "randomSplit" function in PySpark.

```
#split data into training and test sets
(trainingDF, testDF) = ratingsDF.randomSplit([0.8, 0.2], seed=42)
```

Figure 3: Data Splitting

The reason for the split ratio is derived from the Pareto principle, which states that for a wide variety of situations, about 80% of the outcome is caused by around 20% of causes (David, 2022). This split balances the need for a robust training set, and a meaningful evaluation dataset to assess performance.

I decided to implement collaborative filtering, which builds a model from a user's past behaviours as well as similar decisions made by other users (Liao, 2018a). To implement collaborative filtering, I used the Alternating Least

Squares (ALS) algorithm. This algorithm is beneficial due to its scalability and ability to handle large datasets efficiently.

Compared to content-based filtering, collaborative filtering is better for recommending items that are popular among similar users, introducing them to new movies. This is preferred in a situation like a movie recommender as content-based filtering would only recommend movies similar to the ones the user has already watched, which could limit the user's exposure to new movies (Glauber and Loula, 2019). On the other hand, content-based filtering offers a robustness in sparse data, as it does not rely on user-user similarity, compared to collaborative filtering, which has problems in performance when data is sparse.

To get the best of both worlds, it may be beneficial to implement a hybrid recommendation system, which combines collaborative and content-based filtering to provide a recommendation system with novelty, as well as robustness when encountering sparse datasets.

1.4. Implementation of Recommendation System

The ALS model was configured with the following hyperparameters:

- *MaxIter*: I also set this 10, which is the iterations to balance training time and performance.
- *RegParam*: I set this to 0.1, which is the regularisation parameter to prevent overfitting.
- *Rank*: I set to 10, which detemines the number of latent factors in the model.
- *Cold Start Strategy*: I set this to 'drop', which results in missing predictions being handled gracefully.

To train the model, I used the "fit()" function, which trained the model on the training set. Refining the hyperparameters was crucial, as each parameter was chosen to ensure optimal performance.

```
#initialise als model
als = ALS(
    maxIter=10,
    regParam=0.1,
    rank=10,
    userCol="userID",
    itemCol="FilmID",
    ratingCol="rating_value",
    coldStartStrategy="drop"
)
```

Figure 4: ALS Model Initialisation

Once trained, the ALS model predicted ratings for the test set. I used evaluation metrics such as Root Mean Squared Error (RMSE), which is useful when the goal is to minimise overall error, and Mean Absolute Error (MAE), which is useful when the goal is to minimise overall error, while avoiding large errors (Induraj, 2023).

The evaluation was conducted using PySpark's "RegressionEvaluator", which efficiently calculated the RMSE and MAE values. I got a RMSE value of around 0.87 and a MAE value of around 0.67, which indicates a reasonable level of prediction accuracy. Overall, PySpark's ability to efficiently handle large-scale data and its built in machine learning capabilities were vital in developing a robust recommendation system.

```
+------+------+------------+----------+
|userID|FilmID|rating_value|prediction|
+------+------+------------+----------+
|     1|     3|         3.5| 2.9791136|
|     1|     7|         3.5| 3.1295288|
|     1|     9|         2.5|  3.480434|
|     3|     6|         0.5|  1.697159|
|     3|    22|         2.0| 1.8464725|
+------+------+------------+----------+
only showing top 5 rows

Root mean squared error = 0.8672603356571297
mean absolute error = 0.6696552465020411
```

Figure 5: Recommendation System Evaluation

The trained model was used to generate recommendations:

- *User recommendations*: recommending five films for each user.
- *Film recommendations*: recommending five users for each film.

To do this, I utilised Pyspark's "recommendForAllUsers()" and "recommendForAllItems()" functions. These recommendations were based on the predicted ratings generated by the ALS model.

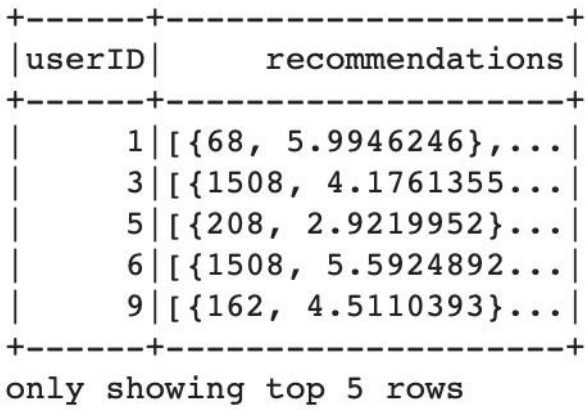

Figure 6: Film recommendations for each user

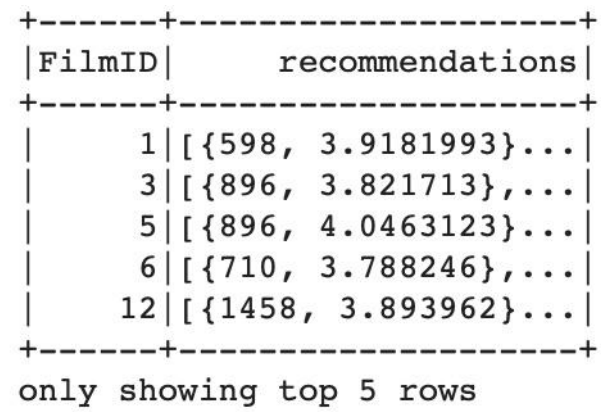

Figure 7: User recommendations for each film

2. Task 2 - MapReduce for Margie Travel Dataset 903

The second task revolved around utilising the MapReduce model to analyse the Margie Travel dataset. There were two files containing lists of data. The first file contained details of passengers that have flown between airports over a certain period, and the second file provided a list of airport data, such as the name, IATA/FAA code, and location. The goal was to use MapReduce, because of its simplicity, scalability and fault-tolerance (Maitrey and Jha, 2015).

2.1. Design and Data Preparation

The design and preparation stage was crucial to ensure the data was structured correctly and ready for analysis. By doing this, the results from the MapReduce operations were reliable and precise. The first step was to load the datasets into Pandas DataFrames, as this allowed for efficient data manipulation and

consistency. Here, for clarity, I renamed the columns which made the dataset easier to interpret and allowed for clarity when writing my mapper and reducer functions. Renaming my columns also minimised coding errors. To convert the time into HH:MM format, I designed a helper function to convert the epoch timestamps from the dataset into human-readable format. This resulted in a better interpretation of flight schedules.

2.2. Implementation of MapReduce

I implemented the MapReduce framework to address three goals: counting flights, converting epoch times to HH:MM, and calculatng the highest air miles travelled by passengers. To accomplish this, I utilised mapper and reducer functions to process the dataset to achieve the goals efficiently.

2.2.1. Counting Flights

The first MapReduce operation focused on counting the number of flights from each airport. To do this, I used a two-phase process with mapping and reducing.

The mapper function, iterated through the dataset, creating key-value pairs where keys were airport codes, and values were flight counts.

The reducer aggregated these counts, providing the total number of flights from each airport. This process was useful for identifying the busiest airports in the dataset, as well as those airports that were unused. This provided valuable data that can be used for resource optimisation and operational planning.

From a business perspective, identifying high and low-traffic airports can inform decisions about staffing, infrastructure investment and how particular services can be enhanced. Similarly, airports that are not utilised as much might present opportunities for new routes or partnerships that would make operations much more efficient.

```python
# Mapper: Count flights from each airport
def map_flights_from_airports(data):
    result = defaultdict(int)
    for _, row in data.iterrows():
        result[row['From Airport']] += 1
    return result

# Reducer: Aggregate counts
def reduce_flights_count(mapped_results):
    reduced_result = defaultdict(int)
    for result in mapped_results:
        for airport, count in result.items():
            reduced_result[airport] += count
    return reduced_result
```

Figure 8: Mapper and Reducer for Counting Flights

```
Number of Flights from Each Airport
{'JFK': 25, 'ORD': 33, 'DEN': 45, 'KUL': 33, 'MAD': 13, 'LHR': 25, 'CGK': 27, 'MUC': 14, 'AMS': 15, 'DFW': 11, 'MI
A': 11, 'CDG': 21, 'CAN': 37, 'IAH': 37, 'LAS': 17, 'CLT': 21, 'ATL': 36, 'PVG': 20, 'FCO': 15, 'BKK': 17, 'PEK': 1
3, 'HND': 13}

List of Unused Airports
['PHX', 'IST']
```

Figure 9: Number of Flights from Each Airport and Unused Airports

2.2.2. Converting Epoch time to HH:MM

It was ideal to convert the epoch time that was in the dataset into typical HH:MM format, which is human-readable. To do this, I created a helper function, which took in a single argument “epoch”, which is the time in seconds since January 1, 1970. “datetime.utcfromtimestamp()” is a method from Python’s “datetime” module, that converts the epoch timestamp into a UTC object, which is much more understandable to humans. After some formatting, and error handling, the time is now readable and better to interpret (Figure 11).

The ability to interpret flight schedules in a clear and accessible manner using HH:MM formart benefits multiple stakeholders. Airlines can use this to easily optimise schedules, passengers can plan travel effectively, and analysts can easily gain insights into behaviours of airlines and passengers. For example, identifying peak hours where many travellers fly can help with resource allocation to suit high-demand periods.

```
# Helper function to convert epoch time to HH:MM format
def convert_epoch_to_hhmm(epoch):
    try:
        return datetime.utcfromtimestamp(int(epoch)).strftime('%H:%M')
    except (ValueError, TypeError):
        return None
```

Figure 10: Helper function to Convert Epoch Time to HH:MM

After this, it was required to collect flight details, such as departure and arrival airports, flight times and passenger counts. I created another mapper function, which extracted and formatted this information, while the reducer aggregated the data. This operation was useful for identifying the most popular flight routes, as well as the busiest times for travel. This information that had been extracted could then be used for further analysis or improvement.

```
List of Flights with Details
{'Number of Passengers': 25, 'Flight ID': 'XXQ4064B', 'Departure Airport': 'JFK', 'Arrival Airport': 'FRA', 'Format
ted Departure Time': '17:05', 'Formatted Arrival Time': '06:27'}
{'Number of Passengers': 18, 'Flight ID': 'SOH3431A', 'Departure Airport': 'ORD', 'Arrival Airport': 'MIA', 'Format
ted Departure Time': '17:00', 'Formatted Arrival Time': '21:10'}
{'Number of Passengers': 18, 'Flight ID': 'PME8178S', 'Departure Airport': 'DEN', 'Arrival Airport': 'PEK', 'Format
ted Departure Time': '17:13', 'Formatted Arrival Time': '15:15'}
{'Number of Passengers': 16, 'Flight ID': 'MBA8071P', 'Departure Airport': 'KUL', 'Arrival Airport': 'PEK', 'Format
ted Departure Time': '17:04', 'Formatted Arrival Time': '02:36'}
{'Number of Passengers': 13, 'Flight ID': 'MOO1786A', 'Departure Airport': 'MAD', 'Arrival Airport': 'FRA', 'Format
ted Departure Time': '16:56', 'Formatted Arrival Time': '20:00'}
```

Figure 11: Flight Details

2.2.3. Highest Air Miles

The final task was to calculate the total air miles travelled by each passenger.

Initially, I needed to calculate the distance between each airport for each passenger. To accomplish this, I again used a mapper and reducer function, where the mapper calculates the distance traveled by each passenger for each flight, and the reducer totals the distances travelled that results in a total global distance.

```python
# Mapper: Calculate distances for each passenger
def map_passenger_distances(data):
    result = defaultdict(float)
    for _, row in data.iterrows():
        try:
            from_coords = airport_data.loc[airport_data['IATA/FAA
            Code'] == row['From Airport'], ['Latitude',
            'Longitude']].values[0]
            to_coords = airport_data.loc[airport_data['IATA/FAA
            Code'] == row['Destination Airport'], ['Latitude',
            'Longitude']].values[0]
            distance = geodesic(from_coords, to_coords).nautical
        except IndexError:
            distance = 0  # Skip invalid airports
        result[row['Passenger ID']] += distance
    return result

# Reducer: Sum distances for each passenger
def reduce_passenger_distances(mapped_results):
    reduced_result = defaultdict(float)
    for result in mapped_results:
        for passenger, distance in result.items():
            reduced_result[passenger] += distance
    return reduced_result

# Map
mapped_passenger_distances = map_passenger_distances
(passenger_data)

# Reduce
passenger_miles = reduce_passenger_distances
([mapped_passenger_distances])
```

Figure 12: Mapper and Reducer Functions for Miles Travelled

The aim of the task was to display the total miles calculated. To do this, I utilised Pandas' ability to create a dataframe using only the "Passenger ID" and "Total Miles" columns. After this, I select 10 rows with the highest "miles" values. "top_10_passengers" is a subset of the dataframe that only contains the top 10 passengers.

```
#Top Passengers by Air Miles
passenger_miles_df = pd.DataFrame(list(passenger_miles.items()), columns=['Passenger ID', 'Total Miles'])
top_10_passengers = passenger_miles_df.nlargest(10, 'Total Miles')

plt.figure(figsize=(12, 6))
top_10_passengers.plot(
    x='Passenger ID',
    y='Total Miles',
    kind='bar',
    legend=False,
    title='Top 10 Passengers by Total Air Miles',
    xlabel='Passenger ID',
    ylabel='Total Miles (nautical)',
    figsize=(12, 6)
)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Figure 13: Implementation of Passenger with Highest Air Miles

```
Passenger with the Highest Air Miles
Passenger ID: BWI0520BG6, Total Miles: 124942.56
```

Figure 14: Passenger Air Miles

Visualisation was key in this task, as it would be very beneficial for a wide range of scenarios to have the data presented in a way that is easy to understand by people of all abilities, and also easy to interpret for many reasons, whether that be business related or otherwise. This is why I created a bar chart displaying the appropriate information, Figure 19 to visualise the number of flights from each airport, and Figure 20 to visualise the highest air miles by passengers. Visualising this data demonstrates the potential for business optimisation, whether that be through marketing campaigns or the potential for loyalty programs. For example, passengers with exceptionally high miles could be offered incentives such as discounts or lounge access, demonstrating business appreciation and customer retention.

These operations demonstrates the power of MapReduce in decomposing problems into manageable tasks, which can be executed efficiently. Not only did these operations demonstrate the framework's ability, but also gathered valuable insights into the dataset:

- The task of counting flights provided a clear picture of airport activity, and identified potential areas for improvement.
- Calculating the miles travelled by passengers offered insights into their behaviour, allowing for targeted marketing strategies and loyalty programs.

3. Task 3 - Big Data Tools and Technology Appraisal 652

The methodologies used in tasks 1 and 2 reflect my understanding of big data concepts and techniques. This appraisal evaluates their effectiveness and scalability while providing recommendations for future improvements.

3.1. Analysis of PySpark in Task 1

Pyspark was instrumental in developing the recommendation system because of its ability to handle large datasets efficiently and faster than other options, for example sci-kit learn (Junaid Et al., 2022). I used the Alternating Least Squares (ALS) algorithm, which is well-suited for collaborative filtering tasks, and is highly scalable with the ability to handle large datasets effectively. The algorithm was straightforward to implement, and ALS minimises the error by adjusting the latent factors in the model, resulting in accurate predictions (Mishra, 2024). Unlike KNN-based approaches, ALS efficiently factorises the interaction matrix into latent features (Liao, 2018b). This scalability makes it a practical choice. The use of PySpark's DataFrame API streamlined the data manipulation process, which resulted in an efficient implementation considering my lack of experience with the tool.

However, the ALS algorithm had some limitations, one of which was the cold-start problem, which was the difficulty to recommend items to new users or items with no ratings, as there is nothing to go from. This limits the applications that this algorithm could be used for. A mitigation for this could be to use a hybrid recommendation system, which combines collaborative filtering with content-based filtering, to provide recommendations for new users or items (Liao, 2018a).

3.2. Analysis of MapReduce in Task 2

MapReduce is a useful programming model framework for large scale data processing (Hashem Et al., 2016). I used this tool in task 2 to process and aggregate flight data, and return the requested values. The tasks were to calculate the number of flights from each airport, airports that are unused in the dataset, and the details of flights. The simplicity of the programming model resulted in a straighforward implementation of the tasks. MapReduce's fault-tolerance ensured reliable execution of the tasks, even in the presence of failures. This resulted in a robust product, ideal for critical operations.

However, MapReduce had some limitations, for example the reliance on real-time data processing, which could be a bottleneck for some applications. Using modern abstractions like PySpark could mitigate these problems, allowing for better scalability and fault-tolerance. Another challenge was the lack of support for advanced analytical functions. This could be resolved by implementing tools like PySpark, which would provide better scalability.

3.3. Visualisation and Interpretation

Visualisation was key in both of the tasks, as it allowed me to display the results of the analysis in a clear and understandable way, which is important for many reasons. Good data visualisation can share summary analysis in a simple manner, along with better business insights (Rahman, 2021). By enabling quick visual displays, it helps identify trends and anomalies, for example it was quick identify that the higher rating values of films had a higher frequency (Figure 18).

To visualise the data, I used the *pandas*, *matplotlib* and *seaborn* libraries. These libraries allowed me to create bar charts and line graphs, which provided valuable insights into the dataset's characteristics.

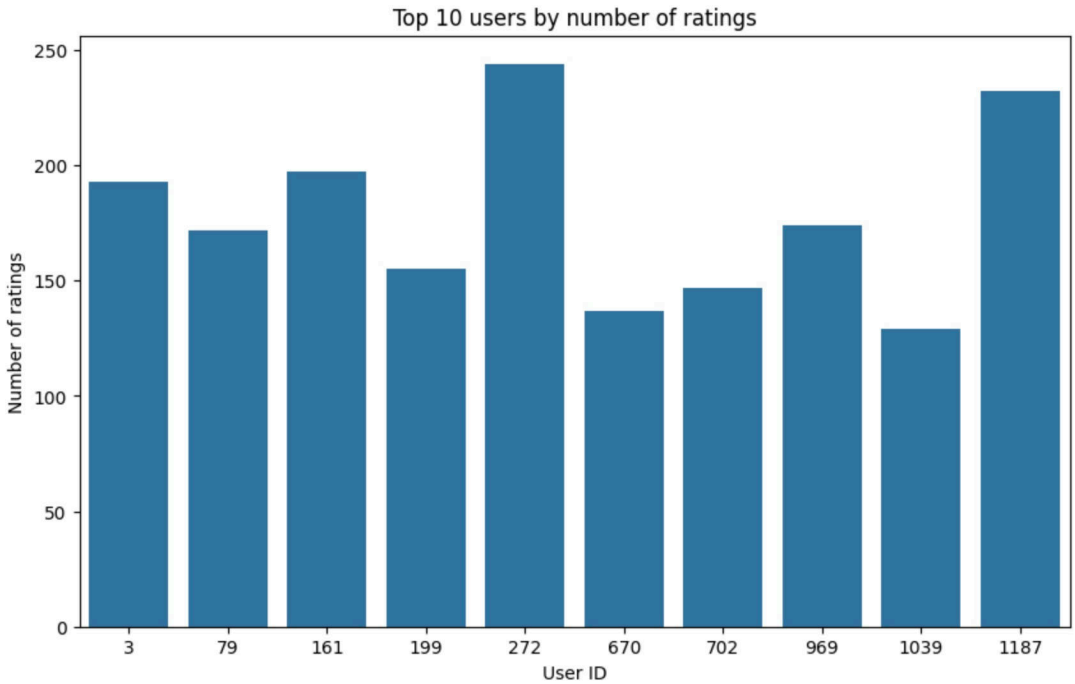

Figure 15: Top 10 Users with Most Ratings

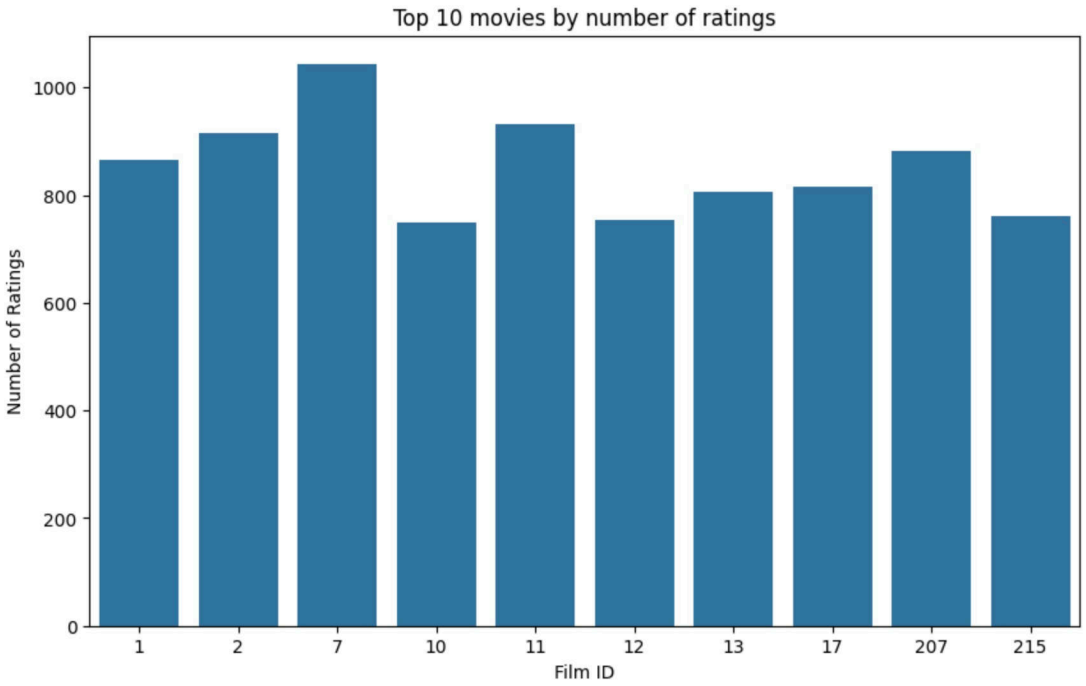

Figure 16: Top 10 Movies with Most Ratings

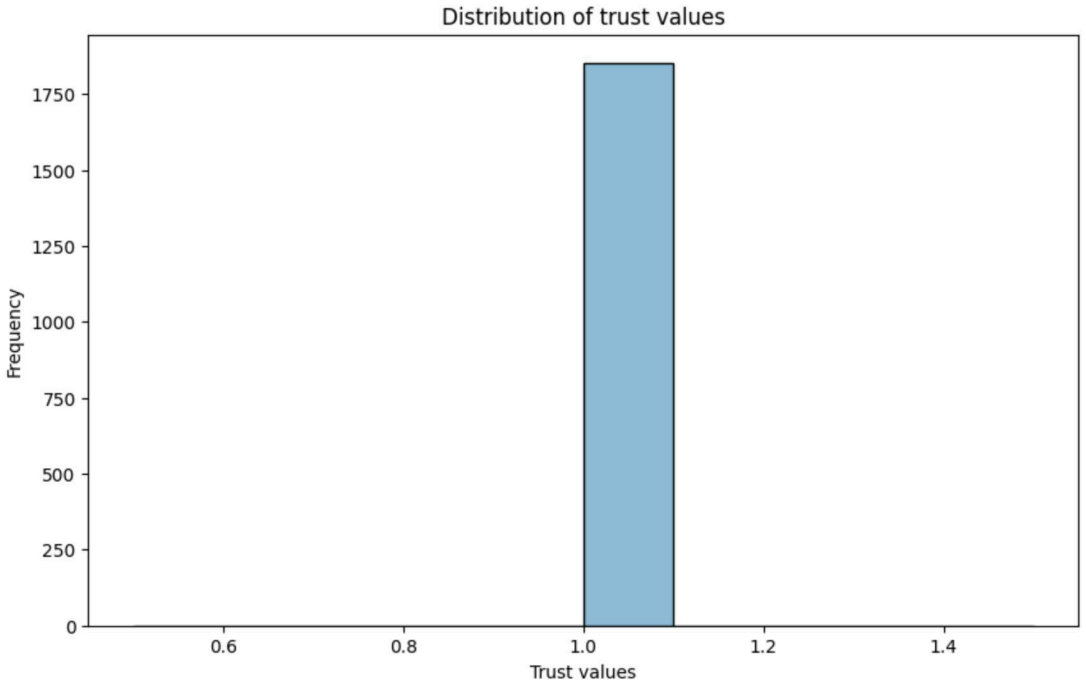

Figure 17: Trust Value Distribution

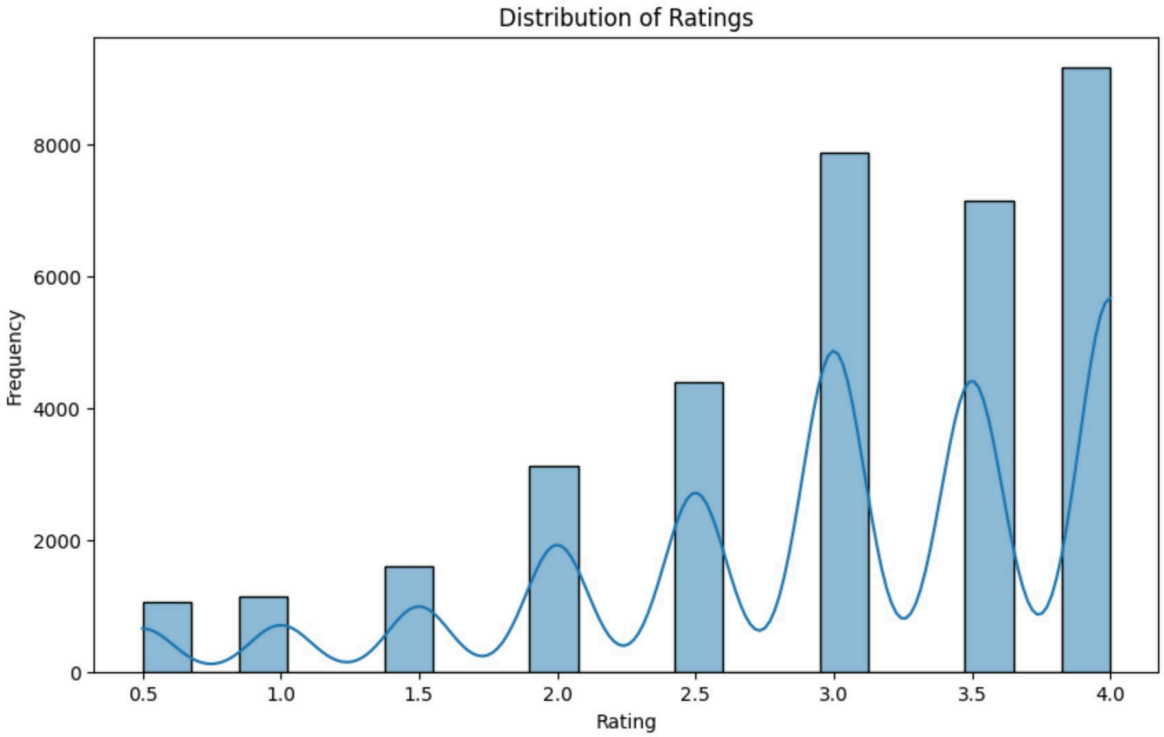

Figure 18: Visualisation of Ratings Data

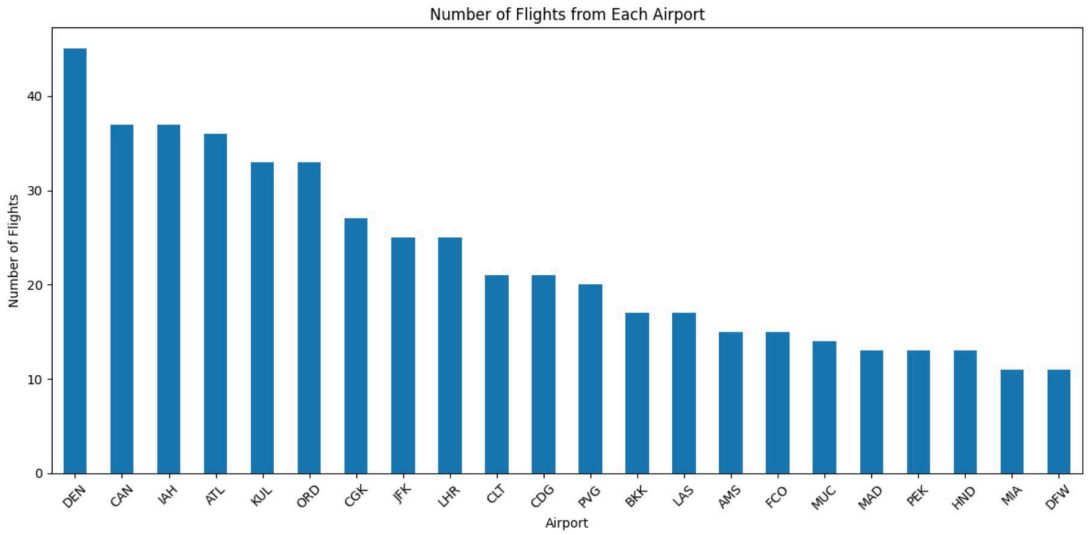

Figure 19: Number of Flights from Each Airport

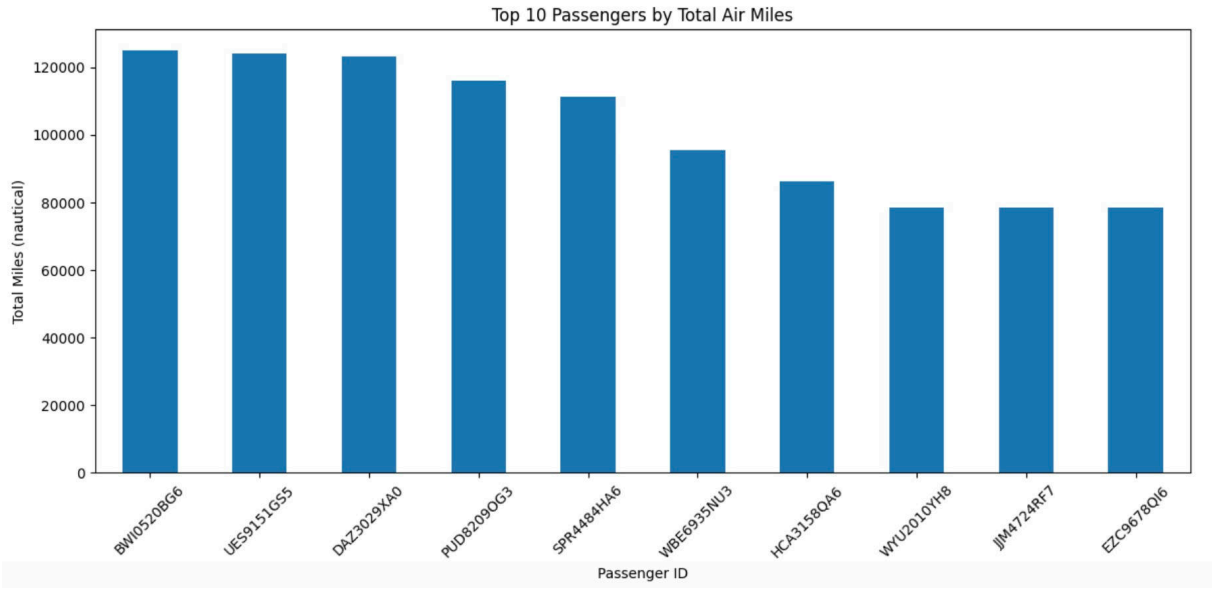

Figure 20: Visualisation of Passenger Air Miles

3.4. Future improvements

After evaluation of my current implementation and the tools that I used, I have identified some ways that this could be improved. Improving the recommendation system would result in a more efficient and accurate model that would be able to work on a wider range of data, along with not struggling with cold-start problems.

To further enhance scalability, I could improve my current implementation by using cloud-based services like AWS. Cloud-based platforms offer distributed storage and computing resources, enabling efficient handling of datasets. This would allow me to process and access data remotely, which would be beneficial in terms of resource management and cost-effectiveness. Also, this would futureproof programs, as the datasets increase in size.

To conclude, both Pyspark and MapReduce have strengths and weaknesses for different situations. PySpark excels in iterative tasks, while MapReduce is better suited for batch processing. After understanding these strenths and weaknesses, I can embrace hybrid implementations, getting the best of both worlds and achieving a more efficient and effective solution.

References

David, E. (2022) *The Pareto principle — Game-changer to ML Engineers in 2022* Available from https://medium.com/analytics-vidhya/the-pareto-principle-game-changer-to-ml-engineers-in-2022-72c53673cb91 [accessed 1 January 2025]

Glauber, R. and Loula, A. (2019) *Collaborative Filtering vs. Content-Based Filtering: differences and similarities* Available from http://arxiv.org/abs/1912.08932 [accessed 2 January 2025]

Hashem, I. A. T., Anuar, N. B., Gani, A., Yaqoob, I., Xia, F. and Khan, S. U. (2016) MapReduce: Review and open challenges. *Scientometrics,* 109(1) 389–422. Available from http://link.springer.com/10.1007/s11192-016-1945-y [accessed 29 December 2024]

Induraj (2023) *Which Metrics in Regression matter the most? MSE|RMSE|MAE|R2|Adj R2- Advantages/Disadvantages* Available from https://induraj2020.medium.com/which-metrics-in-regression-matter-the-most-mse-rmse-mae-r2-adj-r2-advantages-disadvantages-55740cb873ec [accessed 22 December 2024]

Junaid, M., Ali, S., Siddiqui, I. F., Nam, C., Qureshi, N. M. F., Kim, J. and Shin, D. R. (2022) Performance Evaluation of Data-driven Intelligent Algorithms for Big data Ecosystem. *Wireless Personal Communications,* 126(3) 2403–2423. Available from https://link.springer.com/10.1007/s11277-021-09362-7 [accessed 29 December 2024]

Koren, Y., Bell, R. and Volinsky, C. (2009) Matrix Factorization Techniques for Recommender Systems. *Computer,* 42(8) 30–37. Available from http://ieeexplore.ieee.org/document/5197422/ [accessed 29 December 2024]

Liao, K. (2018a) *Prototyping a Recommender System Step by Step Part 1: KNN Item-Based Collaborative Filtering* Available from https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea [accessed 22 December 2024]

Liao, K. (2018b) *Prototyping a Recommender System Step by Step Part 2: Alternating Least Square (ALS) Matrix…* Available from https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1 [accessed 22 December 2024]

Maitrey, S. and Jha, C. K. (2015) MapReduce: Simplified Data Analysis of Big Data. *Procedia Computer Science,* 57 563–571. Available from https://www.sciencedirect.com/science/article/pii/S1877050915019213 [accessed 28 December 2024]

Mishra, A. (2024) *Movie Recommendation System Using Alternating Least Squares* Available from https://medium.com/@anurag.tech/movie-recommendation-system-using-alternating-least-squares-14c7bed98f08 [accessed 29 December 2024]

PySpark Overview — PySpark 3.5.4 documentation (n.d.) Available from https://spark.apache.org/docs/latest/api/python/index.html [accessed 22 December 2024]

Rahman, K. (2021) *Python Data Visualization Essentials Guide: Become a Data Visualization expert by building strong proficiency in Pandas, Matplotlib, Seaborn, Plotly, Numpy, and Bokeh (English Edition)*. BPB Publications