Data Translation Challenge Final Project

Professor Name: Dr James J. Lee
Student Name: Aditi Somani
Student ID: 4181152
Subject: BUAN 5315 02 23SQ Big Data Analysis
Total Points: 200

Use Case Name: Ripe Pumpkins - Movie Review-Aggregation Service

Use Case URL: https://seattleu.instructure.com/courses/1608168/assignments/7115702

Introduction:
Ripe Pumpkins is a startup business that aims to provide a movie review-aggregation service called Pumpkinmeter. The goal is to develop a collaborative recommendation system that calculates Pumpkinmeter scores based on user ratings and preferences. This system will help millions of fans discover movies they are likely to enjoy. The board of directors wants to assess the potential of Pumpkinmeter by analyzing two test scenarios using the MovieLens dataset.

Dataset used:
Name: MovieLens
Source Address: MovieLens website
Summary: The dataset includes ratings and tag applications for movies, collected from the MovieLens website. For this exercise, we will use the Full dataset, which consists of 27 million ratings, 1.1 million tag applications, and information about 58,000 movies provided by 280,000 users.

Technical Details:
The main aspect of Spark applied in this project is collaborative filtering, a technique used to make recommendations by analyzing user preferences and similarities. Collaborative filtering leverages the assumption that users who have similar tastes on one issue are likely to have similar tastes on other issues.

Debugging Details:

1. Handling Large Dataset: One of the challenges encountered was dealing with the large MovieLens dataset, especially the full dataset with 27 million ratings. Processing such a massive amount of data required efficient memory management and optimization techniques. Partitioning the data, utilizing distributed computing capabilities of Spark, and implementing data caching were some of the approaches used to overcome this challenge.
2. Optimizing Recommendation Algorithm: Generating accurate and relevant movie recommendations for millions of users involved optimizing the recommendation algorithm. This required fine-tuning the collaborative filtering techniques, such as matrix factorization or neighborhood-based approaches, to improve the quality of recommendations. Experimenting with different hyperparameters and model configurations helped achieve better results.
3. Preprocessing and Filtering Data: The MovieLens dataset required preprocessing and filtering to extract relevant information for the recommendation engine. This involved handling missing values, transforming categorical variables, and applying appropriate filtering criteria based on the scenario requirements. Modifying the data preprocessing pipeline and applying filtering conditions were necessary to ensure the data was in the desired format for analysis.
4. Achievements: Successfully implementing the collaborative recommendation system and generating 15 personalized movie recommendations for two users were significant achievements. The system was able to leverage user preferences and similarity metrics to provide accurate and relevant movie suggestions. Achieving this functionality at scale using Spark demonstrated the effectiveness of the Pumpkinmeter score and its potential for Ripe Pumpkins' business.

Key Features of Coding Practices:

1. Modularity: The project followed a modular approach, separating different components such as data preprocessing, recommendation algorithm implementation, and result generation. This modular structure allows for better code organization, reusability, and maintainability.
2. Code Documentation: The code was well-documented, including comments and clear explanations of functions, classes, and variables. This practice improves code readability and helps other developers understand the implementation logic.
3. Optimization Techniques: The code implemented optimization techniques to handle the large dataset efficiently. This included using Spark's distributed computing capabilities, data partitioning, and caching mechanisms to enhance processing speed and memory utilization.

4. Error Handling and Exception Management: The code incorporated robust error handling and exception management to handle unexpected scenarios or data inconsistencies. Proper exception handling ensures the code gracefully handles errors and provides informative error messages for debugging.
5. Testing and Validation: The code was thoroughly tested with different test cases and datasets to ensure the accuracy and correctness of the recommendation system. This practice helps identify any potential issues or inconsistencies in the implementation.
6. Scalability Considerations: The code was designed with scalability in mind, utilizing Spark's scalability features to handle large datasets and accommodate future growth. This ensures that the recommendation system can handle increasing user demands and data volumes.

Results:

User 1: Aditi Somani

```
new_user_ID = 0

# The format of each line is (userID, movieID, rating)
new_user_ratings = [
        (0,190863,3), # 3 Musketeers (2011)
        (0,147246,4), # Adventures of Mowgli (1973)
        (0,165723,2), # Ae Dil Hai Mushkil (2016)
        (0,147300,5), # Adventures Of Sherlock Holmes And Dr. Watson: The Twentieth Century Approaches (1986)
        (0,52938,5), # Adventures of Mark Twain, The (1986)
        (0,93923,1), # Agent Vinod (2012)
        (0,151459,3), # Airlift (2016)
        (0,175559,2), # Aiyyaa (2012)
        (0,158236,5), # Ali Baba and the Forty Thieves (1954)
        (0,142382,4), # Agneepath (2012)
    ]
new_user_ratings_RDD = sc.parallelize(new_user_ratings)
print ('New user ratings: {}'.format(new_user_ratings_RDD.take(10)))
```

User 1, Scenario 1:

Scenario 1 - FULL dataset, filtering out movies with less than 25 ratings (meaning 25 or more ratings)

top 15 recommended movies for each user

```
top_movies = new_user_recommendations_rating_title_and_count_RDD.filter(lambda r: r[2]>=25).takeOrdered(15, key=lambda

print ('TOP 15 recommended movies (with more than 25 reviews):\n{}'.format('\n'.join(map(str, top_movies))))
```

```
TOP 15 recommended movies (with more than 25 reviews):
('"Lonely Wife', 5.870108142013315, 43)
('Music for One Apartment and Six Drummers (2001)', 5.864985506876503, 31)
('"Things I Like', 5.771641506766851, 30)
('Cranford (2007)', 5.754259462579273, 35)
('The Garden of Sinners - Chapter 5: Paradox Paradigm (2008)', 5.721277947682164, 27)
("Won't You Be My Neighbor? (2018)", 5.70748342393759, 83)
('Slaying the Badger', 5.689480971449388, 25)
('Alone in the Wilderness (2004)', 5.658184439605291, 343)
('Mei and the Kittenbus (2002)', 5.637207192507796, 44)
('My Love (2006)', 5.635059975005253, 32)
('Hamlet (Gamlet) (1964)', 5.606352083856942, 37)
('"Civil War', 5.605288714379184, 431)
('"Crucified Lovers', 5.5999938654481465, 25)
('Life (2009)', 5.595653871758609, 166)
('Olive Kitteridge (2014)', 5.591718109650474, 211)
```

User 1, Scenario 2:

Scenario 2 - FULL dataset, filtering out movies with less than 100 ratings (meaning 100 or more ratings)

top 15 recommended movies for each user

```
top_movies = new_user_recommendations_rating_title_and_count_RDD.filter(lambda r: r[2]>=100).takeOrdered(15, key=lambda

print ('TOP 15 recommended movies (with more than 100 reviews):\n{}'.format('\n'.join(map(str, top_movies))))
```

```
TOP 15 recommended movies (with more than 100 reviews):
('Alone in the Wilderness (2004)', 5.658184439605291, 343)
('"Civil War', 5.605288714379184, 431)
('Life (2009)', 5.595653871758609, 166)
('Olive Kitteridge (2014)', 5.591718109650474, 211)
('"Decalogue', 5.579240000970557, 547)
('"Best of Youth', 5.554732763576486, 548)
("Smiley's People (1982)", 5.553740799105896, 116)
('Blue Planet II (2017)', 5.552545805339198, 349)
('56 Up (2012)', 5.546527359315515, 193)
('Frozen Planet (2011)', 5.527124650868659, 402)
('Planet Earth (2006)', 5.526898391082175, 1384)
('Ikiru (1952)', 5.5156413672066416, 1551)
('Casablanca (1942)', 5.509163187436604, 31095)
('Promises (2001)', 5.501614878969943, 149)
('Harakiri (Seppuku) (1962)', 5.498013350397949, 679)
```

User 2: Abhishek Somani

USER 2

```
new_user_ID = 0

# The format of each line is (userID, movieID, rating)
new_user_ratings = [
        (0,173175,4), # Badrinath Ki Dulhania (2017)
        (0,82906,5), # Baghban (2003)
        (0,139148,3), # Bajrangi Bhaijaan (2015)
        (0,89848,4), # Badmaash Company (2010)
        (0,146256,5), # Bade Miyan Chote Miyan (1998)
        (0,98956,5), # Barfi! (2012)
        (0,158408,4), # Ajab Prem Ki Ghazab Kahani (2009)
        (0,164600,2), # Akira (2016)
        (0,158697,1), # Aitraaz (2004)
        (0,184551,3), # Aiyaary (2018)
    ]
new_user_ratings_RDD = sc.parallelize(new_user_ratings)
print ('New user ratings: {}'.format(new_user_ratings_RDD.take(10)))
```

```
New user ratings: [(0, 173175, 4), (0, 82906, 5), (0, 139148, 3), (0, 89848, 4), (0, 146256, 5), (0, 98956, 5), (0, 1
58408, 4), (0, 164600, 2), (0, 158697, 1), (0, 184551, 3)]
```

User 2, Scenario 1:

Scenario 1 - FULL dataset, filtering out movies with less than 25 ratings (meaning 25 or more ratings)

top 15 recommended movies for each user

```
]: op_movies = new_user_recommendations_rating_title_and_count_RDD.filter(lambda r: r[2]>=25).takeOrdered(15, key=lambda x:

   int ('TOP 15 recommended movies (with more than 25 reviews):\n{}'.format('\n'.join(map(str, top_movies))))
```

```
TOP 15 recommended movies (with more than 25 reviews):
("India's Daughter (2015)", 6.33296756728441, 25)
('Out in the Dark (2012)', 6.23684469759667, 27)
('Always (2011)', 5.947683051392968, 28)
('"Lord of the Rings: The Return of the King', 5.639035487784753, 57378)
('Bridegroom (2013)', 5.637766977192641, 36)
('The Red Pill (2016)', 5.589525603543304, 46)
('"Lord of the Rings: The Fellowship of the Ring', 5.587376033278204, 61883)
('"Things I Like', 5.579380106463301, 30)
('"Lord of the Rings: The Two Towers', 5.5730645122969875, 56696)
('N.H 10 (2015)', 5.557907634446291, 35)
('Jimmy Carr: Live (2004)', 5.488937260082846, 28)
('Udta Punjab (2016)', 5.4353618781713, 37)
('Holding the Man (2015)', 5.429306257218158, 39)
('Queen (2014)', 5.419449759710034, 82)
('"Misérables in Concert', 5.407820872248806, 43)
```

User 2, Scenario 2:

Scenario 2 - FULL dataset, filtering out movies with less than 100 ratings (meaning 100 or more ratings)

top 15 recommended movies for each user

```
6]: top_movies = new_user_recommendations_rating_title_and_count_RDD.filter(lambda r: r[2]>=100).takeOrdered(15, key=lambda

    print ('TOP 15 recommended movies (with more than 100 reviews):\n{}'.format('\n'.join(map(str, top_movies))))
```

```
TOP 15 recommended movies (with more than 100 reviews):
('"Lord of the Rings: The Return of the King', 5.639035487784753, 57378)
('"Lord of the Rings: The Fellowship of the Ring', 5.587376033278204, 61883)
('"Lord of the Rings: The Two Towers', 5.5730645122969875, 56696)
('Prayers for Bobby (2009)', 5.198114519581836, 102)
('The Blue Planet (2001)', 5.078366014461743, 421)
('Blue Planet II (2017)', 5.07280128557607, 349)
('Doctor Who: The Husbands of River Song (2015)', 5.068229288321994, 239)
('Frozen Planet (2011)', 5.058000585103578, 402)
('Gangs of Wasseypur (2012)', 5.05719203931173, 167)
('Doctor Who: The Time of the Doctor (2013)', 5.043155067985097, 394)
('Life (2009)', 5.031069417476978, 166)
('Doctor Who: Last Christmas (2014)', 5.021954923432752, 228)
('Harry Potter and the Deathly Hallows: Part 2 (2011)', 4.995154400492718, 13262)
('"Swades: We', 4.985371442155568, 163)
('Voices from the List (2004)', 4.983503029912285, 1800)
```

Insights:

The Pumpkinmeter score and collaborative recommendation system have significant business implications for Ripe Pumpkins. It can drive user engagement, retention, and loyalty, differentiate the platform from competitors, and provide valuable insights for data-driven decision making, ultimately leading to business growth and success in the movie review-aggregation service industry.

1. Personalized Movie Recommendations
    - Utilize the Pumpkinmeter score to offer personalized movie recommendations to users

- Enhance user experience and engagement by tailoring recommendations to individual tastes

2. Improved User Retention and Loyalty
   - Understand individual customer preferences through the Pumpkinmeter score
   - Increase user satisfaction, retention, and loyalty by providing customized movie recommendations
3. Competitor Differentiation
   - Stand out from competitors in the movie review-aggregation service industry
   - Unique selling point with the Pumpkinmeter score to attract new users and retain existing ones
4. Enhanced Customer Engagement
   - Actively engage users by offering movie recommendations aligned with their preferences
   - Increase user interaction, exploration, and discovery on the platform
5. Higher User Satisfaction
   - Implement collaborative filtering techniques for accurate movie suggestions
   - Improve user satisfaction by helping users discover movies that match their tastes
6. Data-driven Decision Making
   - Leverage insights from the Pumpkinmeter score for data-driven decision making
   - Make informed choices regarding content curation, marketing strategies, and platform enhancements
7. Business Growth and Success
   - Drive user engagement, retention, and loyalty to fuel business growth
   - Utilize the Pumpkinmeter score as a tool for continuous improvement and success in the industry

References:

1. https://www.canva.com/design/DAFkzW-_-4o/8OsnKP9focKKzEho8KA6aw/edit?utm_content=DAFkzW-_-4o&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton
2. https://seattleu.instructure.com/courses/1608168/assignments/7115702
3. https://grouplens.org/datasets/movielens/latest/
4. https://seattleu.instructure.com/courses/1608168/assignments/7115702

Appendix:

1. Collaborative filtering: https://en.wikipedia.org/wiki/Collaborative_filtering
2. Spark documentation: https://spark.apache.org/documentation.html
3. https://www.codementor.io/spark/tutorial/building-a-recommender-with-apache-spark-python-example-app-part1