**Collaborative Movie Recommendation Based on** **Alternating Least Squares (ALS) with Apache Spark**

****

|  |  |
| --- | --- |
| Name | Subrata Das |
| Pnumber | P2681987 |
| Subject Code | IMAT5322 |

Table of Contents

[Abstract of the report 3](#_Toc103001491)

[Introduction to the report 4](#_Toc103001492)

[The data statistics and schema representations 5](#_Toc103001493)

[Data range, scope, and accuracy 7](#_Toc103001494)

[Data format 7](#_Toc103001495)

[Data processing 7](#_Toc103001496)

[Primary data processing for visualization 7](#_Toc103001497)

[Visualizations of the data 11](#_Toc103001498)

[User count by month 11](#_Toc103001499)

[Rating count by days of the week 12](#_Toc103001500)

[Rating count by an hour of the day 13](#_Toc103001501)

[Average rating vs count of rating 15](#_Toc103001502)

[Movie count by genres 16](#_Toc103001503)

[Released movie count by year – time-series 17](#_Toc103001504)

[Histogram of movies by the runtime 18](#_Toc103001505)

[Observations from visualizations 18](#_Toc103001506)

[Collaborative filtering 19](#_Toc103001507)

[Matrix Factorization 19](#_Toc103001508)

[Coding for ALS 21](#_Toc103001509)

[Splitting the dataframe df\_ratings 21](#_Toc103001510)

[Using Spark ML functions to model our algorithm 21](#_Toc103001511)

[Hyper-parameter tuning 21](#_Toc103001512)

[Best model selection 22](#_Toc103001513)

[Results and Discussion 22](#_Toc103001514)

[Comparison of predicted rating vs actual ratings 23](#_Toc103001515)

[Creating the best 8 movie recommendations for each user 23](#_Toc103001516)

[User-specific recommendation – comparison of output 24](#_Toc103001517)

[Generating the movie homepage links for the user 25](#_Toc103001518)

[Movie and user recommendation 25](#_Toc103001519)

[Limitations and future scope of the work 25](#_Toc103001520)

[Conclusion of the report 26](#_Toc103001521)

[Appendix 27](#_Toc103001522)

[Configuration 27](#_Toc103001523)

[Import Pixiedust 27](#_Toc103001524)

[Data accuracy 28](#_Toc103001525)

[Dropping unwanted columns 31](#_Toc103001526)

[Registering dataframe as table 32](#_Toc103001527)

[Changing numeric values into string equivalent data 32](#_Toc103001528)

[Table 1 datasets 4](#_Toc102849711)

[Figure 1 Primary data processing for visualization 11](#_Toc102849767)

[Figure 2 User count and percentage by month 12](#_Toc102849768)

[Figure 3 Rating count by days of the week 13](#_Toc102849769)

[Figure 4 Rating count by an hour of the day 14](#_Toc102849770)

[Figure 5 Movie count by genres 16](#_Toc102849771)

[Figure 6 Released movie count by year – time-series 17](#_Toc102849772)

[Figure 7 Histogram of movies by the runtime 18](#_Toc102849773)

[Figure 8 Matrix factorization 20](#_Toc102849774)

[Figure 9 Matrix transformation after ALS 20](#_Toc102849775)

# 

# Abstract of the report

In today’s technology world movie business earns a lot from the online streaming platforms or Over The Top (OTT). These platforms provide facilities for their subscribers to view the content over the internet. Netflix, Amazon Prime Video, and Hulu are the most popular platforms.

The biggest reason behind the success of these OTT platforms is the recommendation systems working at the backend to provide recommendations for individual users. There are mainly three types of machine learning approaches present.

1. Collaborative recommendation system.

2. Content-based recommendation system.

3. Popularity-based recommendation system.

In this report, we are mainly focusing on the Collaborative recommendation system which is based on Alternating Least Squares (ALS) algorithm using the Movie Lens (Anon., 2021) dataset. This approach may be used to create recommenders that provide recommendations to a user based on the likes and dislikes of other users based on the rating system. Most of the biggest name in OTT-business uses this approach for their recommendation engines.

In a nutshell, we are focusing on the whole process starting from getting the data from a public forum to providing user-based recommendations. This is an end-to-end solution for the recommender system. The same approach can also be applied to books, songs, web series, and other products provided that the user is already present in the dataset and has some historic rating data for those particular products or items.

# Introduction to the report

The MovieLens (Chang, 2015) datasets are very common datasets that are used in a lot of areas such as research, education, and industry. It is downloaded by thousands of people each year which depicts the widespread use in coding, online and traditional applications. Users’ preferences were recorded in the form of a rating system for a particular movie at a particular time. These preferences were recorded by the website called “Movielens website1” to provide them with personalized movie recommendations.

This dataset is vastly used to create different movie recommender systems and Collaborative filtering is one of the most preferred approaches for the same with the help of ALS. This dataset is comprised of the below Comma Separated Value files:

|  |  |  |  |
| --- | --- | --- | --- |
| File Name | Purpose | Count of rows | Used? |
| genome\_scores.csv | Contains genome scores | Not used | No |
| genome\_tags.csv | Contain genome tags | Not used | No |
| rating.csv | Contain rating details | 9139592 | Yes |
| movie.csv | Contains movie details | 58098 | Yes |
| tag.csv | Contains tags | Not used | No |
| link.csv | Contain links to movies | 58098 | Yes |
| Additional metadata files used | | | |
| Movies\_metadata.csv | Contains all movie information | 45572 | Yes |
| Language\_code.csv (Anon., n.d.) | Contains language code and language | 184 | Yes |

Table 1 datasets

We have only used above mentioned datasets for our analysis. Rest datasets can be used for future scope and will be discussed later. Movies\_metadata.csv is not as accurate as above and for the same reason, we have performed data cleaning and pre-processing for the same.

## The data statistics and schema representations

In this section, we will discuss the structures of the dataframes which were generated from the original CSV files.

1. rating.csv 🡪 df\_ratings

|  |  |
| --- | --- |
|  | Primary key = movieId |
|  | |

1. movie.csv 🡪 df\_movies

|  |  |
| --- | --- |
|  | Primary key = movieId |
|  | |

1. Movies\_metadata.csv 🡪 df\_movies\_metadata

|  |  |
| --- | --- |
|  | Primary key = title |

1. Language\_code.csv 🡪df\_language\_code

|  |  |
| --- | --- |
|  | Primary key = code |
|  | |

1. Links.csv 🡪df\_links

|  |  |
| --- | --- |
|  | Primary key=movieId |
|  | |

From the above screenshots, we can observe the basic structures with variable names and the data types. Also, we can see the basic summary of the data in each screenshot respectively.

Note: The basic summary statistics are provided for the df\_movies\_metadata dataset because this dataframe contains 24 columns and most of the column has not been used in this analysis. Cleaned dataframe details will be provided later in this report.

## Data range, scope, and accuracy

Apart from the df\_movies\_metadata dataframe all the other three dataframes are 100% accurate with no missing values and need no cleaning and pre-processing. Dataframes can be directly used for our purpose. Although, we have done pre-processing on the “Timestamp” column of the rating dataset to explore more visualizations.

df\_movies\_metadata dataframe is the only dataframe that required dropping columns that were not necessary and few steps for pre-processing.

For the Machine learning algorithm “df\_ratings” dataframe has been used and because of the accuracy, no pre-processing or cleaning was required.

Detailed screenshots can be found in Appendix A.

## Data format

All the data files are in comma-separated values (CSV) format which means that the values are delimited by commas. All the files have headers present in the 1st row.

# Data processing

## Primary data processing for visualization

1. Df\_ratings dataframe

At this point, we have imported all the raw datasets and are ready for data cleansing and pre-processing. After checking the accuracy of the data, we have converted the df\_ratings dataframe into the df\_ratings\_summary dataframe by aggregating the column values:

|  |  |
| --- | --- |
|  | |
|  | The dataframe has the average rating and the rating count for each movieId. |

|  |
| --- |
|  |
| This snippet helps to convert the UNIX timestamp provided in the df\_ratings dataframe to extract the exact date and time of the same entry. We can see that corresponding dates are transformed from the original timestamp in the above screenshot. |
|  |
| This snippet helps us to extract the exact date, month, year, and hour data from the date\_of\_rating files using the substring function and proper positions of the data. |
|  |
| In this section, we have extracted the day of the week (week\_day\_abb) from the date\_of\_rating using the “date\_format” function. Also, once that is achieved, we can easily drop the original “date\_of\_rating” and “timestamp” columns. |

1. df\_movies\_metadata

|  |
| --- |
|  |
| In this piece of code, we have just dropped all the unnecessary columns for our analyses but there is a lot of insightful information resides in those columns’ data, that can be used for future  scope. |
|  |
| After joining df\_movies and df\_ratings\_summary dataframe using primary key=”movieId”, df\_movies\_ratings\_summary is created. In this snippet, we have removed the trailing year data from the “title” column. |
|  |
| In this section, we have just rearranged all the columns as per our convenience. |
|  |
| Column “runtime” had string values so we have explicitly typecast that to an integer. |

In the below, diagram we can see the whole data preparation flow. All the joins and transformations are depicted in the same.

|  |
| --- |
|  |

Figure 1 Primary data processing for visualization

We have not included the df\_language\_codes dataframe in the diagram because this dataframe is only used to replace the language codes in the df\_movies\_metadata dataframe with its original language and it is not significant.

Note: We have not done any preprocessing or data cleaning for the df\_ratings dataframe for the machine learning part as the data is perfect and has no missing values. If there were any such values we could have deleted that if the numbers were less in percentage or just replace the values of the columns with the mean values. This approach can minimize the error.

# Visualizations of the data

Pixiedust has been used here to create the visualizations. Any other tool or library can be used to create visualization such as Matplotlib, Seaborn, Tablue, etc.

In this section, we have explored descriptive analytics.

## User count by month

|  |  |
| --- | --- |
|  | |
| Description:  This bar chart represents the total number of users who provided ratings throughout the whole timespan. This data is also segregated by months in ascending order. | Inference:  **November, December, and January** are the favorite months here for people to watch movies if we assume that every user rated the movies immediately after watching.  On the other side, **February to May** is the opposite. People tend to watch fewer movies and rate. |

Figure 2 User count and percentage by month

## Rating count by days of the week

|  |  |
| --- | --- |
|  | |
| Description:  This bar chart represents the total number of ratings provided by users throughout the whole timespan. This data is also segregated by days of the week in ascending order. | Inference:  People tend to watch **more** movies on **Sundays, Mondays, and Tuesdays** as depicted in the bar chart.  On the other side, people use less time on other days.  One of the possible reasons is people dedicate less time as the week progresses towards the weekends and spend more time in a **social gatherings**. |

Figure 3 Rating count by days of the week

## Rating count by an hour of the day

|  |  |
| --- | --- |
|  | |
| Description:  This line chart shows the user involved in providing ratings during the hours of the day. | Inference:  This line graph clearly shows that the users tend to dedicate more time after **15:00 hours and the number increases with time till 21:00 hours.** After that, it again starts to decline as the night progresses. |

Figure 4 Rating count by an hour of the day

## Average rating vs count of rating

|  |  |
| --- | --- |
|  | |
| Description:  This scatters plot shows the average rating vs rating count. Additionally, the histogram on the top shows the distribution of average ratings. | Inference:  We can see observe that the most rating given are in the **rating zone of 2.5 to 3.5**.  We can also see some **outliers** are also present near 0 and 5 ratings.  The above histogram shows the **normal** **distribution** if we just ignore those outliers. |

## Movie count by genres

|  |  |
| --- | --- |
|  | |
| Description:  This bar chart depicts the movies released as per genre. | Inference:  We have the **most** movie released in the genre of **Drama** and this is the most commonly tagged genre in most of the movies.  **Comedy, thriller, romance, and action are following respectively.**  **Rest** genres are quite narrow in the movies and particular to a movie type and cannot be generalized like other genres. **This is the reason we have fewer movies in these categories**. |

Figure 5 Movie count by genres

## Released movie count by year – time-series

|  |  |
| --- | --- |
|  | |
| Description:  This time-series graph shows the count of movies released by year. | Inference:  We have data for 1880-2018 and the same is reflected in the above. A gradual increase in movie production can be observed throughout history. The sudden **jump** in the increase can be observed in the last **two decades**. |

Figure 6 Released movie count by year – time-series

## Histogram of movies by the runtime

|  |  |
| --- | --- |
|  | |
| Description:  Histogram of movies by the runtime | Inference:  We can see that most of the density is around 50-180 minutes range which is the normal time for short movies and full-length movies. Apart from that, we can also see some outliers are near 1200 minutes but the number is too less. |

Figure 7 Histogram of movies by the runtime

# Observations from visualizations

Assumptions: we consider that the user rates the movies as soon as they watch.

1. Most movies watched from November to January (10%).
2. Least movies are watched in September and May (7%).
3. Most movies have a runtime between 50-180 minutes.
4. Last two decades has a spike in movie production and the growth is noticeable.
5. We have the most movie released in the genre of Drama and this is the most commonly tagged genre in most of the movies. Comedy, thriller, romance, and action are following respectively.
6. The most common rating provided with the most counts is in the zone of 2.5 to 3.5.
7. Users’ movie watch time increases from 15:00 hours to 21:00 hours. It is least during the office hours in the mooring which is 09:00 hours and stays low around that time.
8. People tend to watch more movies on Sundays, Mondays, and Tuesdays, and at least on Thursdays and Fridays. The possible reason may be a social gathering.

Movie recommender system

The technique used in this report is collaborative filtering which is based on the algorithm of matrix factorization. This algorithm creates a rating matrix for all the users and all the movies and later using ALS we will factorize the main matrix into a small depiction matrix.

We have concluded more than one recommendation by using different approaches.

1. Movie recommendation for the user
2. User recommendations for movies
3. Popularity-based movie recommendations for new users.

This is part of prescriptive and cognitive analytics.

## Collaborative filtering

Collaborative filtering is commonly employed in recommender engines. The missing elements are predicted using matrix factorization. The rating for movies implemented using Spark-ML, now employ the model-based approach.  Users or goods are modeled in a collaborative framework that can predict missing entries due to latent variables. Furthermore, the ALS (alternating least squared) method was implemented to discover these hidden aspects.

## Matrix Factorization

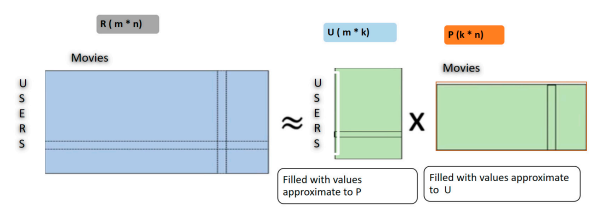
To tackle the sparsity problem in collaborative filtering, matrix factorization is utilized. In this problem, user-provided rating values are being considered account. to build aneffective recommender system we need to choose a few parameters which will analyze the data and provide final recommendations. In this report, we have used Apache Spark to build our recommendation engine based on the ALS algorithm.Figure 8 shows the suggested smaller matrices and it is based on user and item level approaches and integrated with the matrix factorization approach, ALS, and Latent features to partition the rating dataset into an item-based and user-based matrix. As a consequence, we generate a new user matrix when we combine them again which should have all the values populated even for the missing values based on the similarities between the user and the object.

Figure 8 Matrix factorization

The root means squared error (RMSE) should be used to rate the prediction, and the model will be trained.

This report is based on machine learning and the ALS algorithm, which is a type of regression analysis in which a line of data points is presented and utilized to draw through regression. The sum of squares of the distance between these shown data points reduces the distance in this fashion. These lines are used to anticipate the value of the function, which is subsequently fulfilled by the independent variables.

Let us assume we have the main matrix R which is factorized into two smaller matrices, U and P. These U and P matrices should provide an approximation of the original matrix R. ALS starts with a random set of values for U and P matrices and calculate the error for the random matrices; then by changing the values of matrix U and matrix P, ALS tries to reduce the error using the RMSE. ALS continues this process till the error is minimized. Once the error is minimized the matrices are multiplied together and we receive again the original matrix with some error values but in the process, we get all the missing values replaced with predicted ratings of the users that were missing in the original matrix R (Yu, et al., December 2012). The same is depicted in Figure 9.

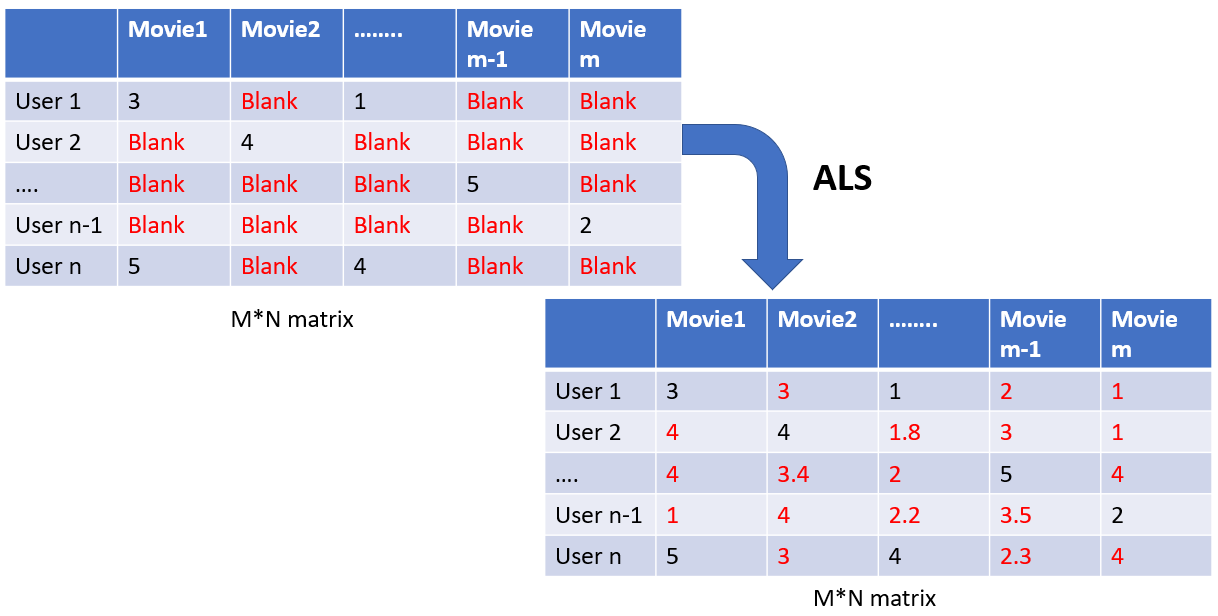


Figure 9 Matrix transformation after ALS

## Coding for ALS

All the codes are uploaded along with this report but it is required to focus on a few cases we have used here.

### Splitting the dataframe df\_ratings

|  |
| --- |
|  |
| The dataset is large and because of that reason we chose 70% data to be our training data and the rest are test data. |

### Using Spark ML functions to model our algorithm

|  |
| --- |
|  |
| ALS is the Spark ML function that can be directly called to train our training data. This is supervised learning as we have our “rating” as a label to build our model.  **Initially**, we are working with a random set of parameters just to see how the model performs and we can see the root mean square error is **0.8638** which is not a bad result.  Initial parameters:  maxIter=5  regParam=0.01  Also,  userCol is “userId” which is the actual user for which the rating will be predicted.  itemCol is “movieId” which is the item or movie to be suggested.  ratingCol is “rating” which is the prediction for each movie.  The output predictions are in the prediction dataframe and we can extract the required information from the same. |

### Hyper-parameter tuning

|  |
| --- |
|  |
| Now we will tune the hyper-parameters and chose between multiple models to select the best one.  Param\_grid is the reference matrix for the different models. We will be testing the model with 27 different sets of values made out of the combination of the below parameter range:  maxIter[5,10,15]  rank[5,10,15]  regParam[0.01,0.05,0.1] |

### Best model selection

|  |
| --- |
|  |
| This is the best model with the lowest RMSE in the set of **27 parameter combinations**.  **Rank=15**  **MaxIter=15**  **RegParam=0.1**  **Best RMSE=0.8225**  Just to mention: **This code runs for a longer time and sometimes throws an OutOfMemory error for Java heap memory. Kindly keep patients running the code or refer to the above screenshots.** |

## Results and Discussion

### Comparison of predicted rating vs actual ratings

|  |  |
| --- | --- |
|  | This table shows the random user rating for a particular movieId the actual rating and the predicted rating.  We can see we are predicting the ratings with a little error which can reduce by training larger data. |

### Creating the best 8 movie recommendations for each user

|  |
| --- |
|  |
| We have created a custom function here to retrieve the userId, movieId, and predicted ratings for the top 8 movies. We can generate any number of movies by changing the argument to other numbers from 8 here in the function. |

### User-specific recommendation – comparison of output

|  |
| --- |
|  |
| This code helps to find the top 8 movies to recommend to users by any particular user.  Here we have demonstrated with userId=10.  Final\_df contains the predicted rating of the top-recommended movies  Df\_test contains the original data with the highest rated movies of user 10.  We can match the genres of the movies and from those two tables, we can conclude that both the users spend time watching movies of drama, comedy, and romance. Hence our prediction is accurate. |

### Generating the movie homepage links for the user

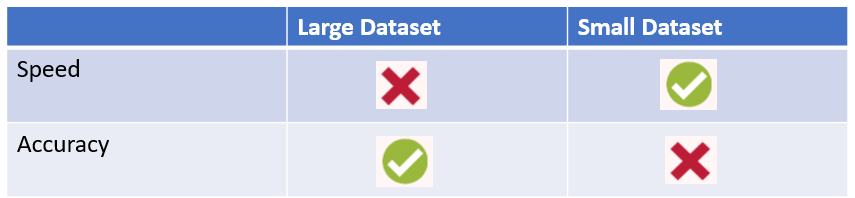
|  |
| --- |
|  |
| Using the df\_links dataframe we generated the TMDB homepages for the movies. We have only considered 3 entries here. |

### Movie and user recommendation

|  |  |
| --- | --- |
|  |  |
| This code suggests the top 5 movies for each user.  It is the user-based approach. | This code suggests the top 5 users for each movie. This is item-based approach.  If any person watches any movie, then based on that we can recommend this. |

# Limitations and future scope of the work

ALS is one of the most reliable algorithms for recommender systems and it provides fair accuracy in predicting the target variable but whenever any new user or item comes into the ecosystem it fails to provide a recommendation. As there is no prior rating or user behavior the model cannot predict ratings or outcomes. This issue is called data **sparsity and cold-start problem.**



Also, ALS lacks performance with the growing data volume. If the user or item count increases the performance drops drastically. **Scalability** is one of the major concerns of collaborative filtering.

In the end, the users will always get recommendations on old items based on the behavior of old users who are similar. New products/movies will never be recommended until and unless they become old and rated by lots of users. This can be a real **problem for new products**.

But now all the OTT platforms are collaborating with multiple recommender systems to provide recommendations. In the future, we can also use Natural Language Processing (NLP) to analyze the text data which are ported by users as reviews to provide fresh recommendations. Apart from the Content-based recommendations can also be included. We can employ deep learning-based algorithms to apply movie recommendations in the future. Multimodal-based techniques with efficient deep learning and the BigDL framework have inspired several recent studies (Aljunid, 2020). Furthermore, the suggested approach may be used to recommend books or news.

With the combination of all the above approaches, we can create a highly effective dashboard or homepage for every user.

# Conclusion of the report

This report started with the initial information about the recommender system and walked through the process of creating an actual one. Initial data exploration helped to decide the goals and paths for the result. Data cleaning and pre-processing helped to build a model on accurate and cleaned data. In the end, we have achieved the recommendations using the ALS algorithm. We have also used cross-validation to decide our best model for the algorithm. Best parameter selection is the key for any model building. In this report, we also suggested different types of recommendations such as item-based, user-based and popularity-based.

To conclude even though there are some limitations present for the ALS method but it is still a reliable algorithm to predict recommendations, but we can achieve better results if we collaborate with different algorithms. In the future, this work can be continued with the ideas mentioned in the future scope section.

# Appendix

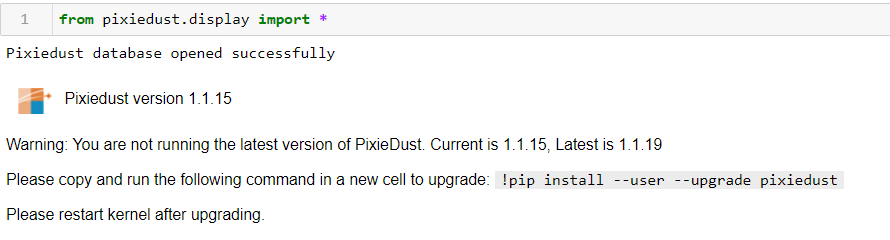
## Configuration

Below configurations were used to set up the environment for Spark and libraries for using different methods provided by Spark and Python.

|  |  |
| --- | --- |
|  | ML libraries are included to utilize the pre-build Spark ML functionalities for the ALS method. |

## Import Pixiedust

We have used Pixiedust here for the visualization but any other library can be used such as Matplotlib or Seaborn.



## Data accuracy

Initial dataset

|  |  |
| --- | --- |
| df\_movies | Dataframe contains zero missing value |
|  |  |
| df\_ratings | Dataframe contains zero missing value |
|  |  |
| df\_language\_code | Dataframe contains zero missing value |
|  |  |

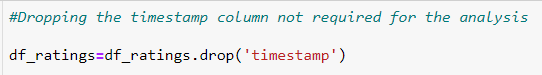
Final dataset after multiple joins

|  |  |
| --- | --- |
| Df\_ratings\_summary | Aggregated dataframes generated from df\_ratings dataframe.  Dataframe contains zero missing value |
|  |  |
| df\_all\_data\_movies | Joined dataframe of below dataframes:   1. Df\_ratings\_summary 2. Df\_movies\_metadata   Dataframe contains some missing values which are insignificant compared to the total count. |
|  |  |
|  |  |
|  |  |
|  |  |
| Df\_language\_codes |  |
|  |  |

## Dropping unwanted columns

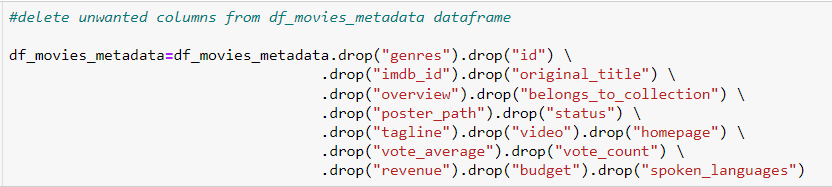
Below code snippets provide the details for dropped columns:

1. Df\_ratings.drop



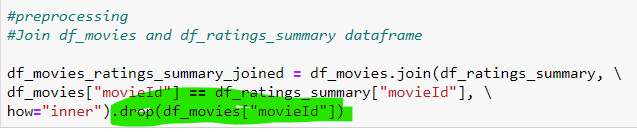
1. Df\_movies\_metadata

We have dropped the below columns because these are not going to use in our analysis.



1. Df\_movies\_ratings\_summary\_joined

We have dropped ambiguous columns.



## Registering dataframe as table

## Changing numeric values into string equivalent data

In this below code snippet, we have converted the month numeric values into corresponding month names.



# References

Aljunid, M. a. D. M., 2020. An efficient deep learning approach for a collaborative filtering recommender system. *Procedia Computer Science,* pp. 171, pp.829-836.

Anon., 2021. *Movie Lens Dataset.* [Online]   
Available at: https://grouplens.org/datasets/movielens/  
[Accessed 2022].

Anon., n.d. *Data Hub.* [Online]   
Available at: https://datahub.io/core/language-codes  
[Accessed 29 4 2022].

Chang, S. H. F. a. T. L., 2015. Chang, S., Harper, F.M. and Terveen, L., 2015, February. Using groups of items for preference elicitation in recommender systems. In Proceedings of the. *18th ACM Conference on Computer Supported Cooperative Work & Social Computing ,* pp. pp. 1258-1269.

UK department of transport and road safety, 2022. [Online]   
Available at: https://data.gov.uk/dataset/cb7ae6f0-4be6-4935-9277-47e5ce24a11f/road-safety-data  
[Accessed 31-03-22 Mar 2022].

Yu, H.-F., Hsieh, C.-J., Si, S. & Dhillon, I., December 2012. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. *In Proceedings of the 2012 IEEE 12th International Conference on Data Mining,* pp. 765-774.