[Date]

**Aleksandra Petkova - S6222041**

**Nour Aldin - P4302185**

**Victor Essien - W9104119**

Data Mining

**Case Study Report**

Contents

[Introduction 2](#_Toc39575373)

[Datasets 2](#_Toc39575374)

[1. Online Retail: time series, transactional data – 8 attributes, 541K records 2](#_Toc39575375)

[2. Groceries: customer recite data – 9K records 2](#_Toc39575376)

[3. MovieLens: movie rating data – 100K records 2](#_Toc39575377)

[Techniques 2](#_Toc39575378)

[*Association Rules Mining, Apriori (Core and Library) – Aleksandra Petkova, Nour Aldin* 2](#_Toc39575379)

[Brief 2](#_Toc39575380)

[Datasets Usage 2](#_Toc39575381)

[Algorithm (*Source code can be found in AssociationRulesCore.py*) 6](#_Toc39575382)

[Questions 8](#_Toc39575383)

[Experiments and Evaluation *(What experiments can be done to prove it works)* 9](#_Toc39575384)

[*Collaborative Filtering (Core and Library) – Victor Essien, Nour Aldin* 10](#_Toc39575385)

[Brief 10](#_Toc39575386)

[Datasets Usage 11](#_Toc39575387)

[Experiments and Evaluation *(What experiments can be done to prove it works)* 11](#_Toc39575388)

[Other Contributions 15](#_Toc39575389)

[Conclusion 15](#_Toc39575390)

[References 15](#_Toc39575391)

# Introduction

This project was developed for the Data Mining module at Teesside University with the aim to demonstrate and evaluate the use of popular computational techniques for data mining. The project contains implementations of Association Rules Mining, Collaborative Filtering and variety of datasets to test on.

# Datasets

### [Online Retail](https://archive.ics.uci.edu/ml/datasets/Online+Retail): time series, transactional data – 8 attributes, 541K records

Each row represents a whole sale purchase of an item along with details about the purchase such as: the invoice ID, stock code, item description, quantity purchased, price per unit, the ID and country of the customer who made the purchase.

### [Groceries](https://www.kaggle.com/irfanasrullah/groceries): customer recite data – 9K records

Each row represents a customer recite (list of the items bought) with no details about the purchase but the names of the items.

### [MovieLens](https://www.kaggle.com/prajitdatta/movielens-100k-dataset): movie rating data – 100K records

Each row represents one review by one customer.

# Techniques

## *Association Rules Mining, Apriori (Core and Library) – Aleksandra Petkova, Nour Aldin*

### Important Note

Aleksandra and Nour built 2 separate Apriori algorithms – from scratch/core python (Aleksandra) and with a library (Nour). The version from scratch was running a lot slower than the version with the library so much so that 50k records from online retail took 20 minutes to process. Considering the dataset has 500k records that dataset was dropped for the version from scratch. Despite that the library version of Apriori runs on both algorithms with processing time to complete for 500k records taking a bit under 30 min on the same machine that the other version of the algorithm was tested. The Apriori report section was fully written by Aleksandra along with the RegX pre-processing code for the Online Retail dataset.

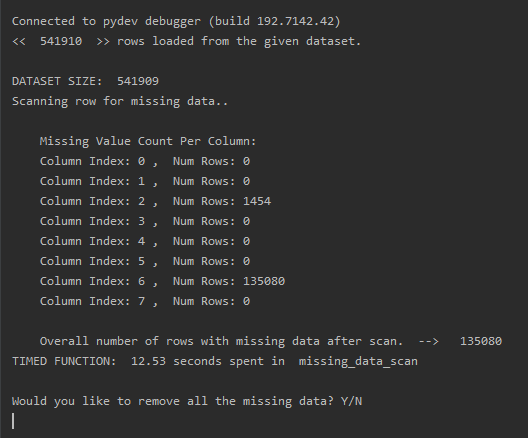
### Brief

The **Apriori Algorithm** is one of many algorithms for **Association Rules Mining**, other popular examples are the **Eclat Algorithm**, **OPUS Search** and **FP-growth algorithm**. We chose to work with **Apriori** because it is a standard and simple method for association rules mining and is also the one we are most familiar with.

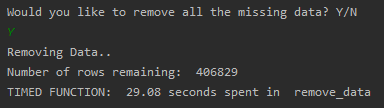
### Datasets Usage

1. *Online Retail –* Used for both **the algorithm** **written from scratch** and **the algorithm** **written** **with the MLextend Library**. It is the **largest dataset** from the available and has both **missing** and **erroneous** **data**, those are the key reasons it was first chosen to present the **Apriori Association Rules Mining** algorithm. Despite that the amount of data was too much for the **Association Rules** algorithm written from scratch so we had to swap to the **Groceries dataset** instead. The library version still has the ability to run on this algorithm. All the **pre-processing** data was done on it because there is no **pre-processing** to do for the **Groceries dataset**.
2. *Groceries* – Much smaller dataset, no cleaning required, very simple to work with when building an **Association Rules Mining** algorithm. That is why this is the dataset used to build and test the first prototype of the algorithm written from scratch. Around beginning of April, we reverted back to using that dataset and stuck with it until the end.
3. *Simple Dataset* – Is a random **sample** from the *Groceries* dataset used to **validate** each step of the algorithm.

**Pre-processing Done and Experiments - Online Retail Dataset**:

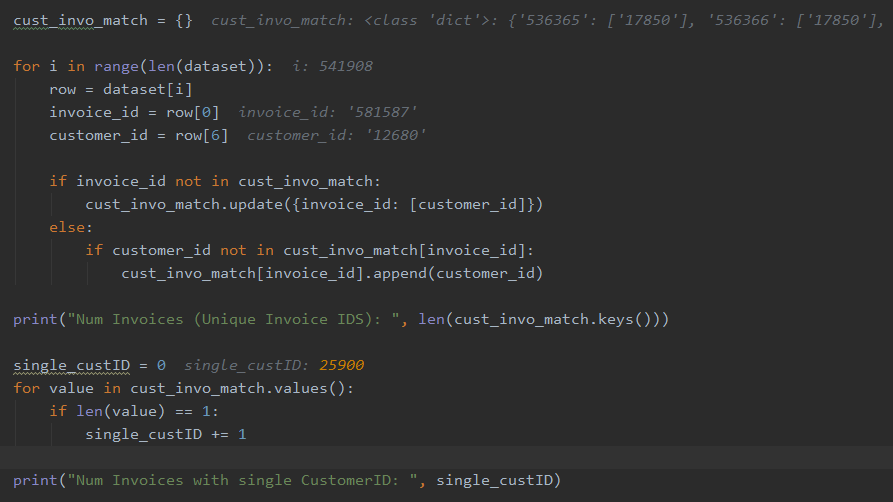


**Figure 1: Missing Data Scan**



**Figure 2: Missing Data Removal**

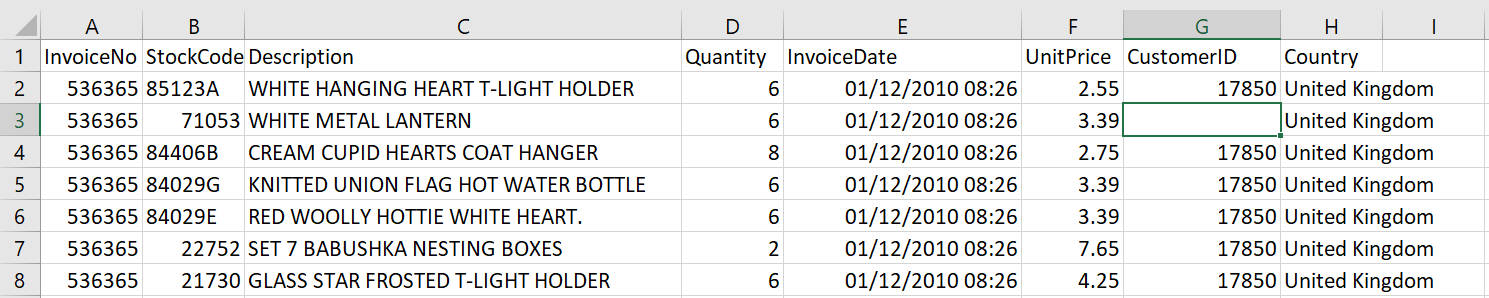
1. **Missing Data Scan**: As you can see from Figure 1, there are only 2 columns that contain missing data – **Column 2 (Description)** and **Column 6 (CustomerID)**. The rows missing in **Column 6** representabout a **5th** of the whole dataset so ideally it would be good to fill them. According to the dataset structure (more information in the [Datasets Section](#_Online_Retail:_time)) the invoice is always sent to the same customer, i.e. the **InvoiceID** will only ever be linked to one **CustomerID**. This is confirmed by the investigation on this and the results from the mining shown on Figure 3.



**Figure 3: Code written to prove theory and output below.**

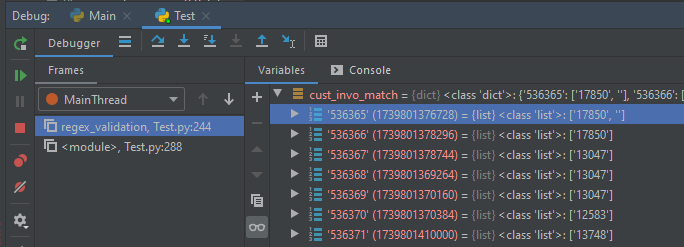
**Output:**

If there was a case where one or more **CustomerIDs** existed in an Invoice with missing **CustomerIDs** that would mean the **cust\_invo\_match dictionary** would store the existing **CustomerID** and the empty string - **“”**. In the example shown below (Figure 4) I manually removed one of the customer IDs from an invoice to test how the scan will react to this.



**Figure 4, Experiment: Manually removing a CustomerID.**

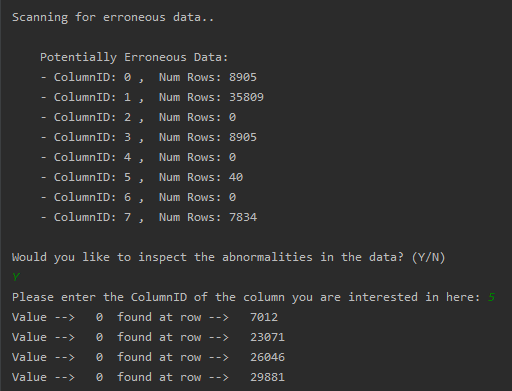
The empty string was indeed added as an option for **CustomerID** for the current **InvoiceID** (Figure 5). The fact that every invoice in the dataset only has one option for **CustomerID**, as shown on Figure 3, that means that there are no cases where at least one **CustomerID** is present on an invoice with one or more missing **Customer IDs**.



**Figure 5: Experiment results after removing a customer ID.**

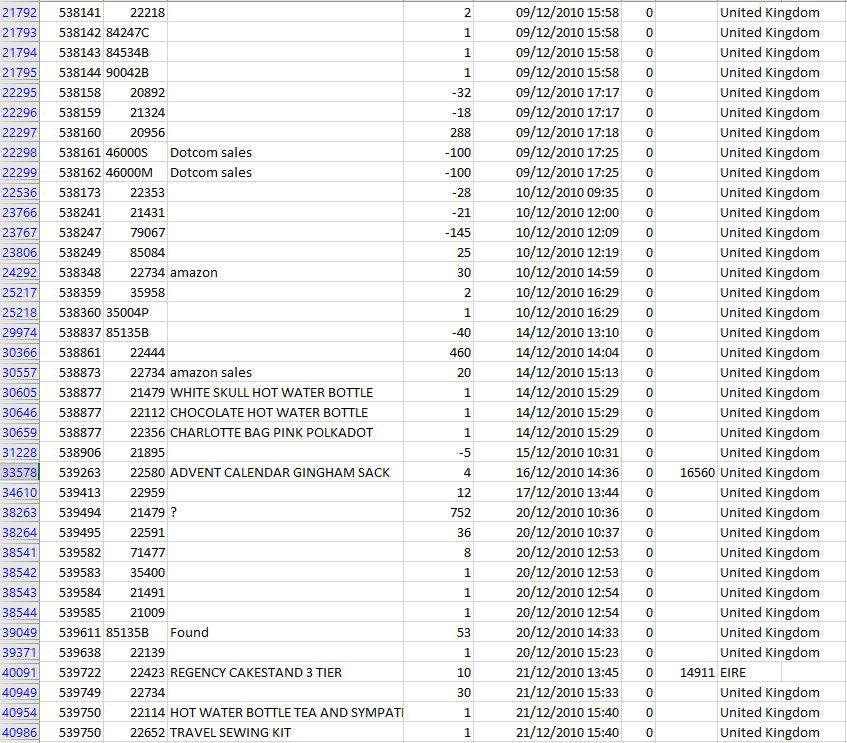
This disproves the idea of filling the data when at least one **CustomerID** exists in an invoice that contains missing **CustomerIDs**. I could not find any other ways to fill the missing data so I removed all of it including the records with missing data in column 2.

1. **Regular Expression Validation System:** Catches any data that does not fit the set rules and displays insights to the user in the form of console output. **Example Regex**: **UnitPrice (Column 5)** has the regular expression **“^£{0,1}([0-9]+\.[0-9]\*[1-9]+)|([1-9][0-9]\*)”** which states, the data in the column may have 0 or 1 of £ symbol at the start of the column data – **“^£{0,1}”**, followed by either a floating point number that is > 0.0 – **“[0-9]+\.[0-9]\*[1-9]+”**, or a whole number that is > 0 – **“[1-9][0-9]\*”.** If the data in the **column fails** this **validation**, it will be recorded into the **compromised\_rows** **dictionary**, which the user has the option to inspect right after the full scan is complete. The output after the scan is shown in the screenshot on Figure 6. If the user types Y or y, they will be asked to **select a column** by entering the **column ID**. As you can see, I selected **Column 5** to follow up on the example given. The regex caught one issue, with 40 occurrences/rows, which is the unit price being 0. Depending on the owner of the dataset this may be intentional or it could be something to remove, but to make a choice I would have to look at the full row containing this data.



**Figure 6: Regex Validation System Output**

Below (Figure 7) you can find a screenshot of what I found by manually inspecting the compromised rows. Here you can see that there are generally all kinds of issues with these rows, like empty cell for **CustomerID**, negative value for quantity, missing item description, etc. At this point we found out that the association Rules written from scratch is impossibly slow at dealing with frequency counting of itemsets larger than 1 element per set (on **50K rows** has **900k combinations** of **2-itemsets** and takes **52 minutes** to process). So, we decided to swap to the **Groceries dataset**, which has very feasible processing time. The library version the **Groceries dataset** does not require any cleaning though which is why we kept the cleaning for **Online Retail**.



**Figure 7: Rows with UnitPrice 0**

### Algorithm (*Source code can be found in AssociationRulesCore.py*)

1. Find all **unique items** by transforming the loaded dataset from 2D to 1D and applying a set cast on the whole dataset to remove any repetitions. The unique items are then sorted to allow for easier management of the data.
2. Create an item to number **map/dictionary**, with the itemset as the key and the item ID as the value to **improve performance** as string comparisons are expensive.
3. **Generate mapped data**
   1. Reformat the **unique items** using the **unique items map** so that each **1-itemsets** is represented by its **ID**.
   2. Reformat the **dataset** using the **unique items map** so that the algorithm is only working with item **IDs**.
4. Count the **frequency** of each unique item.
5. Calculate the **minimum support** by multiplying the **minimum relative support** by the highest **frequency**. (**Example**: if the highest frequency is 6 and the relative support is 0.5 (50%) the minimum support/frequency will be 3.)
6. Apply the **Apriori property** to filter out the **infrequent** 1-itemsets.
7. Append the survived 1-itemsets dictionary to the complete\_survival\_subsets\_list.
8. Start main Association Rules Loop and keep running until there are no more itemsets left to work with.

8.1. Generate a k+1 itemsets list from the survived k-itemsets dictionary.

8.1.1 Sets contain only unique values so {0, 0}, {1,1}, {2,2}, etc should not exist! Also, multiple variations of the same subset/itemset should not be checked at this stage because they may not and most likely will not be frequent. Example: if {0,1} is checked {1,0} should not be checked unless for sure {0,1} is frequent. Then we want to see which item implies the other so we will try each variation. Below there are tables of the acceptable 2, 3, 4-itemsets that implement the rules mentioned. You can see that there is a clear pattern which was used to optimise the process for generating k+1 itemsets.

based on a much smaller dataset (**simple\_dataset.csv**) that is also available in the “**Data Repository**” folder. The example tables do not show the Apriori property, they simply visualise the outputs assuming all subsets survive.

Table :  **All possible set combinations 1-itemset 🡪 2-itemset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | {0} | {1} | {2} | {3} | {4} |
| {0} | X | X | X | X | X |
| {1} | {0, 1} | X | X | X | X |
| {2} | {0, 2} | {1, 2} | X | X | X |
| {3} | {0, 3} | {1, 3} | {2, 3} | X | X |
| {4} | {0, 4} | {1, 4} | {2, 4} | {3, 4} | X |

Table 2: **All possible set combinations** **2-itemset 🡪 3-itemset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | {0} | {1} | {2} | {3} | {4} |
| {0, 1} | X | X | X | X | X |
| {0, 2} | X | X | X | X | X |
| {0, 3} | X | X | X | X | X |
| {0, 4} | X | X | X | X | X |
| {1, 2} | {0,1,2} | X | X | X | X |
| {1, 3} | {0,1,3} | X | X | X | X |
| {1, 4} | {0,1,4} | X | X | X | X |
| {2, 3} | {0,2,3} | {1,2,3} | X | X | X |
| {2, 4} | {0,2,4} | {1,2,4} | X | X | X |
| {3, 4} | {0,3,4} | {1,3,4} | {2,3,4} | X | X |

Table 3: **All possible set combinations 3-itemset 🡪 4-itemset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | {0} | {1} | {2} | {3} | {4} |
| {0,1,2} | X | X | X | X | X |
| {0,1,3} | X | X | X | X | X |
| {0,1,4} | X | X | X | X | X |
| {0,2,3} | X | X | X | X | X |
| {0,2,4} | X | X | X | X | X |
| {0,3,4} | X | X | X | X | X |
| {1,2,3} | {0,1,2,3} | X | X | X | X |
| {1,2,4} | {0,1,2,4} | X | X | X | X |
| {1,3,4} | {0,1,3,4} | X | X | X | X |
| {2,3,4} | {0,2,3,4} | {1,2,3,4} | X | X | X |

8.2. Count the **frequency** of each itemset.

8.3. Apply the **Apriori property** to filter out the **infrequent** current k-itemsets.

1. Generate Rules out of the frequent itemsets.

9.1. Ignore the 1-itemsets as no rules can be generated from 1 item.

9.2. Split the itemset into two parts (X and Y).

9.3. Calculate **confidence** of the rule **X implies Y or Y implies X**.

9.4. If it is below the **minimum confidence** given, discard the rule.

1. Display rules

10.1. Open the file where the rules will be written - **association\_rules.txt** (the “with” keyword was used to make sure Python’s garbage collector cleans the file object)

10.2. Go through every rule in the rules dictionary

10.2.1. Extract the X and Y subsets, their frequencies/supports and the confidence of the rule.

10.2.2. Construct a string out of that data to display the results.

**Example string:** **{pip fruit, root vegetables}**, **sup(153), rel sup(2%) ---> {whole milk}, sup(2513), rel sup(26%) - conf(58%)**

1. Count the number of rules and itemsets and print that to the console.

### Questions

1. How many **unique items** are there? - **169 unique items** found.
2. What is the **frequency** of each **unique item**?

- Please refer to **frequent\_itemsets.txt** file, located in the **outputs folder**.

1. Which items have the **highest frequency** and which one has the **lowest**?
2. Which items are very **often bought together**? (Association rules with **relative support higher than 50%**)

* Please refer to **association\_rules.txt** file, located in the outputs folder.

1. Are there any **associations** in the dataset/s with **high confidence**? (**Rule confidence > 50%**)

* Please refer to **Experiments and Evaluation** tables.

1. How many association rules have **high confidence**? (**Rule confidence > 50%**)

* Please refer to **Experiments and Evaluation** tables.

### Experiments and Evaluation *(What experiments can be done to prove it works)*

1. Display results with different support values, **highest and lowest support are calculated based of the itemsets that have survived the minimum support cap**. **All changes made from previous experiment are marked in red!**

Table 4, Exp1: Results Comparison of the algorithm - from scratch (Core Python) and with library (MLextend)

|  |  |
| --- | --- |
| Parameter | Result |
| Minimum Support | 63 occurrences |
| Lowest Support | 77 occurrences |
| Highest Support | 2513 occurrences |
| Minimum Relative Support | 2.5% |
| Minimum Confidence | 50% |
| Lift | N/A |
| Number of Survived Itemsets | 688, with sup > 63 |
| Number of Rules | 52 rules |

Table 5, Exp2: Changed Minimum relative support to 10%, with minimum rule confidence still at 50%.

|  |  |
| --- | --- |
| Parameter | Result |
| Minimum Support | **251 occurrences** |
| Lowest Support | **736 occurrences** |
| Highest Support | 2513 occurrences |
| Minimum Relative Support | **10%** |
| Minimum Confidence | 50% |
| Lift | N/A |
| Number of Survived Itemsets | **86, with sup > 251** |
| Number of Rules | **0 rules** |

Table 6, Exp3: The only change is to Minimum Rule Confidence, which was brought down from 50% to 10%

|  |  |
| --- | --- |
| Parameter | Result |
| Minimum Support | 251 occurrences |
| Lowest Support | 736 occurrences |
| Highest Support | 2513 occurrences |
| Minimum Relative Support | 10% |
| Minimum Confidence | **10%** |
| Lift | N/A |
| Number of Survived Itemsets | 86, with sup > 251 |
| Number of Rules | **66 rules** |

Table 7, Exp4: Changed both Minimum relative support and minimum confidence to 5% and 50%.

|  |  |
| --- | --- |
| Parameter | Result |
| Minimum Support | **126 occurrences** |
| Lowest Support | **228 occurrences** |
| Highest Support | 2513 occurrences |
| Minimum Relative Support | **5%** |
| Minimum Confidence | **50%** |
| Lift | N/A |
| Number of Survived Itemsets | **228, with sup > 126** |
| Number of Rules | **5 rules** |

## *Collaborative Filtering (Core and Library) – Victor Essien, Nour Aldin*

**Important Note:** Both Nour and Victor built 2 separate CF algorithms – From scratch/core python (Victor) and with a library (Nour) but the report section for Collaborative Filtering was fully written by Nour.

### Brief

The algorithm has been developed both **from scratch** and with the [**Surprise**](http://surprise.readthedocs.io/en/stable/getting_started.html) **Library** to make recommendation based on Collaborative Filtering and predict user rating for a movie. Another popular algorithm for Collaborative Filtering is **Nearest Neighbour**, we chose to write it using the [**Surprise**](http://surprise.readthedocs.io/en/stable/getting_started.html) **library** because it has default implementation for a variety of CF algorithms and to write one from scratch to get a better understanding of the technique.

### Datasets Usage

1. *Movielands*: This dataset (ml-latest-small) describes a movie recommendation service. It contains 100836 ratings and 3683 tag applications across 9742 movies. The data contains ratings from 610 unique users made between March 29, 1996 and September 24, 2018. The dataset was generated on September 26, 2018. The users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.

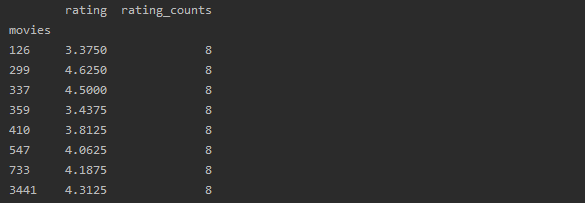
* User Ids: MovieLens users were selected at random for inclusion
* Movie Ids: Only movies with at least one rating or tag are included in the dataset.
* Ratings: Each line of this file after the header row represents one rating of one movie by one user.

The lines within this file are ordered first by ushered, then, within user, by movieId. Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars).

### Experiments and Evaluation *(What experiments can be done to prove it works)*

1. Find the average rating and number of ratings for the movie?

To find the average rating of each movie. To do so, we grouped the dataset by the title of the movie and then calculate the mean of the rating for each movie. Then we sorted the ratings in the descending order of their average ratings, then we have added add the number of ratings for a movie to the ratings\_mean\_count data frame as showing in figure 8.



*Figure 8 average rating and number of ratings for the movie*

To have a clear visualization we have plotted a histogram chart for the number of ratings represented by the "rating\_counts" column in the above data frame. This can be seen in Figure 9.

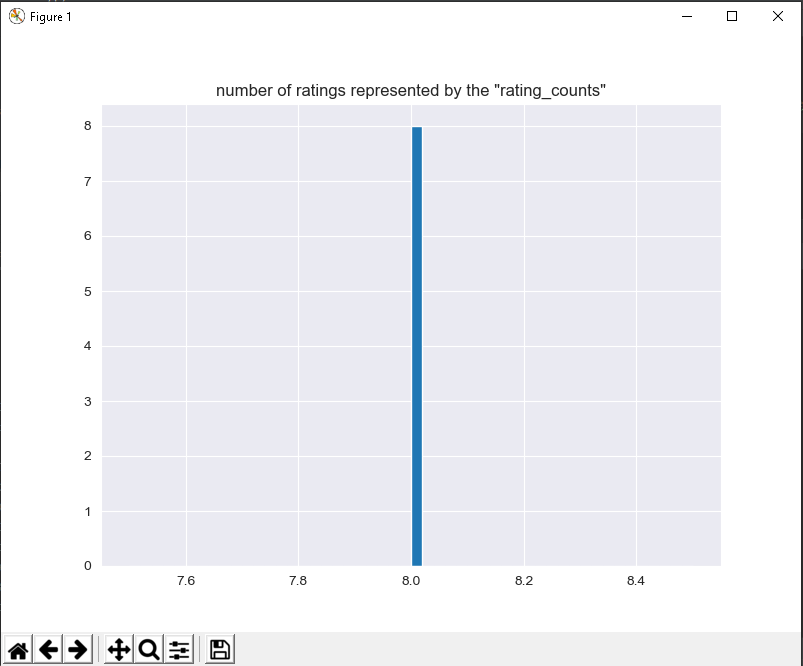


Figure 9. Number of ratings represented by the “rating\_counts”

The output in figure 9, clearly shows that most of the movies have received 8 ratings. To view the average ratings for each movie, we have plotted a histogram chart as showing in Figure 10.

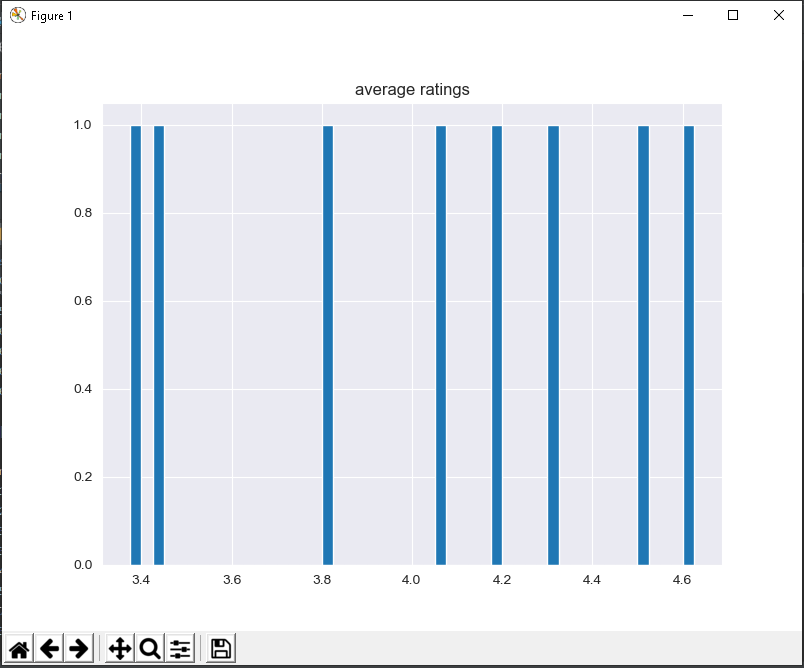


Figure 10. Average ratings.

1. How can we find the predicted rating?

As in many other Machine Learning algorithms training is used to optimize its predictions to match as closely as possible the actual results. So, in the context of collaborative filtering, our algorithm will try to predict the rating of a certain user-movie combination and it will compare that prediction to the actual prediction. The difference between the actual and the predicted rating is measured using classical error measurements such as Root mean squared error (RMSE) and Mean absolute error (MAE).

In the **Surprise library** we have a wide choice of algorithms to use and a wide choice of parameters to tune in each algorithm. One of the famous available algorithms mentioned is **SVD**. For the purpose of our example we will use the SVD algorithm. For the sake of simplicity, we will use the algorithm as is without adding many customisations through the available parameters.

The **Surprise library** also supports the **RMSE** and **MAE** measurements so we will use those to measure the performance of our algorithm below in Figure 11.

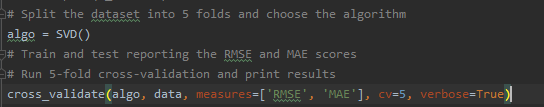


Figure 11. **optimization code**

Figure 12. displays the performance of our algorithm through the use of 5-fold Cross Validation.

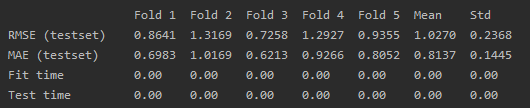


Figure 12. SVD Performance

**Predicting**

Finally, we are interested in the predicted rating for a certain user-movie combination so we know whether the user will like this movie or not.

To do this, first we will train on the whole data set without splitting the data to get the best results possible as showing in Figure 13.

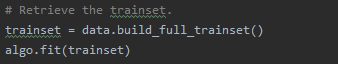


Figure 13. Train on the whole data.

To predict a rating, we give the algorithm the users, movies and the actual rating data as follows in Figure 14.

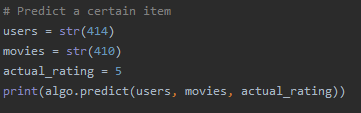


Figure 14. Predict the rating of a certain item.

And from the results we can see that the estimated rating is 4.04 compared to the actual rating of 5 as displaying in figure 15.



Figure 15. Predicted vs actual rating.

# Contributions and Peer Mark

Nour

* Actively attending group meetings and discussions except for one of the weeks.
* Built two different algorithms with libraries – Apriori and Collaborative Filtering.
* Wrote the presentation and the Collaborative Filtering section of the report.

Aleksandra

* Setup the Trello Boards and Github.
* Actively attending group meetings and discussions.
* Built the Apriori algorithm from scratch and wrote the whole report except the Collaborative Filtering and the Conclusion sections.
* Supported the team when there are issues.
* Proof reading the report and presentation.

Victor

* Actively attending group meetings when not ill.
* Wrote the Collaborative Filtering Algorithm from scratch.

Table : Peer Mark (13 peer marks)

|  |  |  |
| --- | --- | --- |
|  | Aleksandra | Nour |
| Aleksandra | 7 – I believe this is a fair mark for me because I was always there to support everyone despite the fact that I also have my own work to do; built an algorithm from scratch and spent a long time making sure it all works as intended and improving on the code. Despite the fact that performance wise it could not compare to the library version I believe I learnt a lot. | 6 – I believe this a fair mark because despite being new to programming and data science, he showed initiative and improvement, always asked for help when stuck. Sometimes it seemed like there was not much effort put to fix the issue by himself before asking for help which is not too great but understandable at times as there is only so much time one can afford to spend on fixing bugs. |
| Nour |  |  |

# Conclusion

Association rule mining algorithms such as Apriori are very useful for finding simple associations between our data items. They are easy to implement and have high explain-ability. However, for more advanced insights, such those used by Google or Amazon etc. In this project, we studied what a recommender system and association rules mining are and how we can create them in Python using many different libraries and also from scratch.

# References

Garg, A., 2018. *Complete guide to Association Rules (1/2).* [Online]   
Available at: https://towardsdatascience.com/association-rules-2-aa9a77241654  
[Accessed Febuary 2020].

Malaeb, M., 2017. *The easy guide for building python collaborative filtering recommendation system.* [Online]   
Available at: https://medium.com/@m\_n\_malaeb/the-easy-guide-for-building-python-collaborative-filtering-recommendation-system-in-2017-d2736d2e92a8  
[Accessed April 2020].

Ng, A., 2016. *Association Rules and the Apriori Algorithm: A Tutorial.* [Online]   
Available at: https://www.kdnuggets.com/2016/04/association-rules-apriori-algorithm-tutorial.html  
[Accessed Febuary 2020].