[Date]

**Aleksandra Petkova - S6222041**

**Nour Aldin**

**Victor Essien**

Data Mining

**Case Study Report**

Contents

[Abstract 2](#_Toc36898964)

[Introduction 2](#_Toc36898965)

[Datasets 2](#_Toc36898966)

[1. Online Retail: time series, transactional data – 8 attributes, 541K records 2](#_Toc36898967)

[2. Groceries: customer recite data – 9K records 2](#_Toc36898968)

[3. MovieLens: movie rating data – 100K records 2](#_Toc36898969)

[Techniques 2](#_Toc36898970)

[*Association Rules Mining, Apriori – Aleksandra Petkova, Nour Aldin* 2](#_Toc36898971)

[Brief 2](#_Toc36898972)

[Datasets Usage 3](#_Toc36898973)

[Algorithm (*Source code can be found in AssociationRulesCore.py*) 6](#_Toc36898974)

[Questions 8](#_Toc36898975)

[Experiments and Evaluation *(What experiments can be done to prove it works)* 9](#_Toc36898976)

[*Collaborative Filtering (Library) – Victor Essien* 9](#_Toc36898977)

[Brief 9](#_Toc36898978)

[Datasets 9](#_Toc36898979)

[Algorithm 9](#_Toc36898980)

[Experiments and Evaluation *(What experiments can be done to prove it works)* 9](#_Toc36898981)

[Conclusion 9](#_Toc36898982)

[References 9](#_Toc36898983)

**PLEASE DO NOT REMOVE COMENTS UNTIL THE FINAL DRAFT IS COMPLETE!**

# Abstract

“““Talk about..

Given data:  
Important Parameters:   
Analysis techniques used:   
Results/Output/Evaluation: ”””

# ABSTRACT HERE

# 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

# More details here

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

# More details here

# Techniques

## *Association Rules Mining, Apriori – Aleksandra Petkova, Nour Aldin*

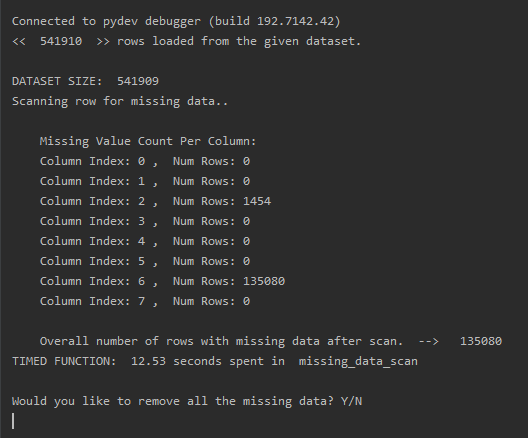
### Brief

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.

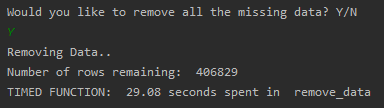
### Datasets Usage

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. 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**:

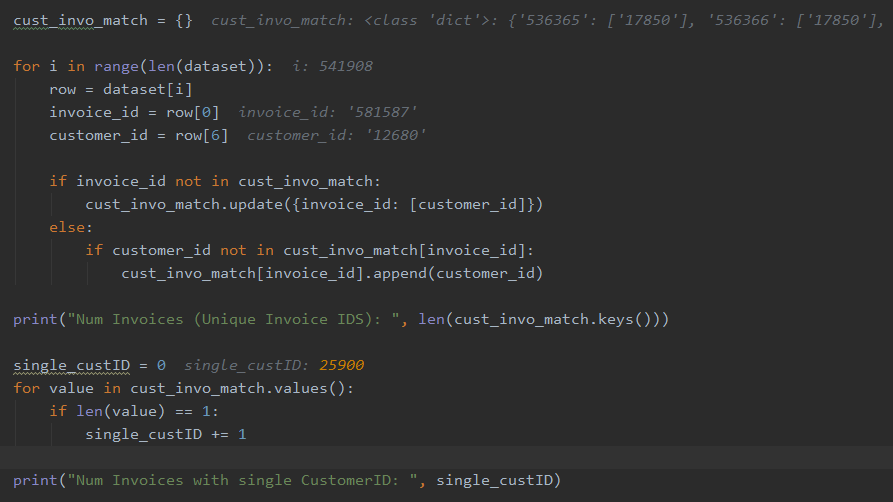


**Figure 1: Missing Data Scan**



**Figure 2: Missing Data Removal**

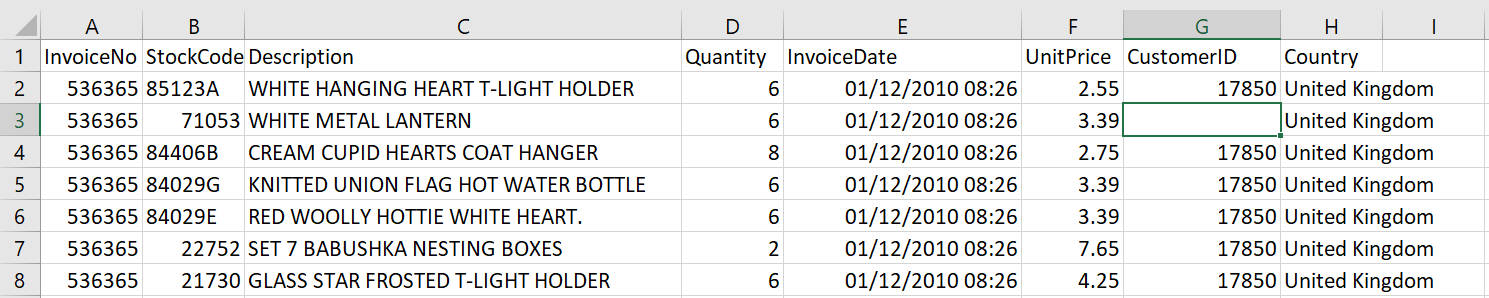
1. **Missing Data Scan**: As you can see from Figure 1, there are only 2 columns that contain missing data – **Column 2 (Description)** and **Column 6 (CustomerID)**. The rows missing in **Column 6** representabout a **5th** of the whole dataset so ideally it would be good to fill them. According to 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.**

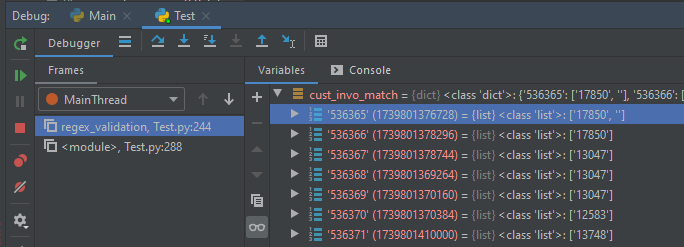
**Output:**

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.**

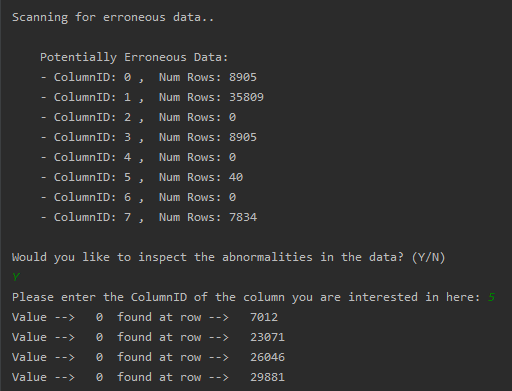
The empty string was indeed added as an option for **CustomerID** for the current **InvoiceID** (Figure 5). The fact that every invoice in the dataset only has one option for **CustomerID**, as shown on Figure 3, that means that there are no cases where at least one **CustomerID** is present on an invoice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



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

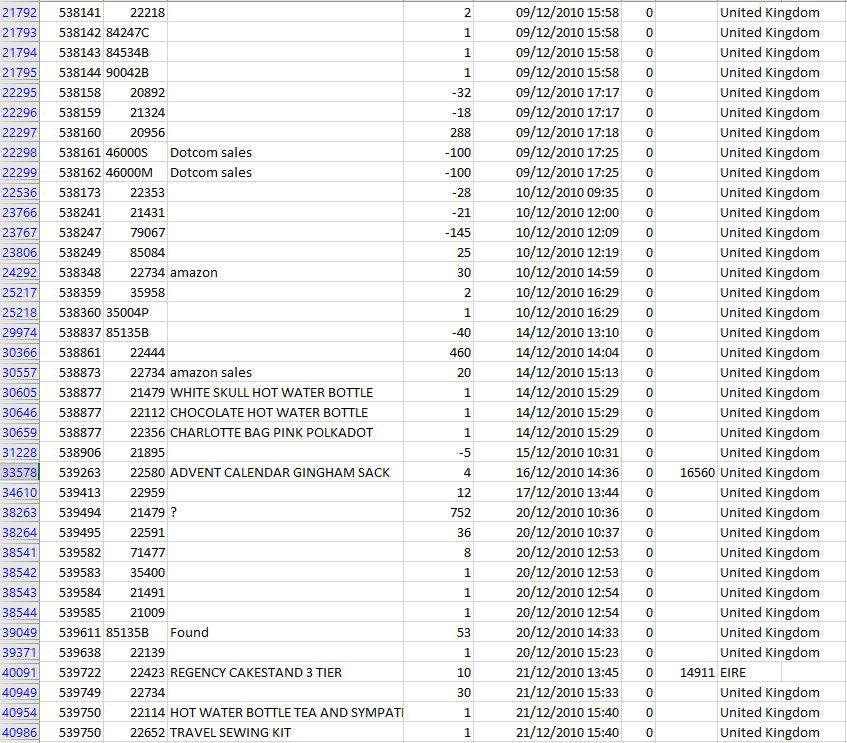
This disproves the idea of filling the data when at least one **CustomerID** exists in an invoice that contains missing **CustomerIDs**. I could not find any other ways to fill the missing data so I removed all of it including the records with missing data in column 2.

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**?



1. Which items are very **often bought together**? (Association rules with **relative support higher than 50%**)
2. Are there any **associations** in the dataset/s with **high confidence**? (**Rule confidence > 50%**)
3. How many association rules have **high confidence**? (**Rule confidence > 50%**)

### 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)

|  |  |  |
| --- | --- | --- |
|  | Core Python | Python Library |
| 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%.

|  |  |  |
| --- | --- | --- |
|  | Core Python | Python Library |
| 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%

|  |  |  |
| --- | --- | --- |
|  | Core Python | Python Library |
| 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%.

|  |  |  |
| --- | --- | --- |
|  | Core Python | Python Library |
| 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** |  |

1. Applied algorithm on a very small dataset (**simple\_dataset.csv**) so that results can easily be verified.
2. When given an **itemset** can the algorithm **return all it’s association rules**?

## *Collaborative Filtering (Library) – Victor Essien*

### Brief

* What is this technique normally used for?
* What other techniques exist for this problem?
* Why is this technique suitable for the selected dataset/ Why is this technique suitable for the selected dataset?

### Datasets

1. *MovieLens*

### Algorithm

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

1. Q1
2. Q2

# Conclusion