MEF UNIVERSITY

MARKET BASKET ANALYSIS USING APRIORI ALGORITHM

Capstone Project

Yıldırım Murat ŞİMŞEK

İSTANBUL, 2018

MEF UNIVERSITY

MARKET BASKET ANALYSIS USING APRIORI ALGORITHM

Capstone Project

Yıldırım Murat ŞİMŞEK

Advisor: Dr. Tuna ÇAKAR

İSTANBUL, 2018

MEF UNIVERSITY

Name of the project: Market Basket Analysis Using Apriori Algorithm
Name/Last Name of the Student: Yıldırım Murat ŞİMŞEK
Date of Thesis Defense: 15/01/2018

I hereby state that the graduation project prepared by Yıldırım Murat Şimşek has been completed under my supervision. I accept this work as a “Graduation Project”.

15/01/2018
Dr. Tuna Çakar

I hereby state that I have examined this graduation project by Yıldırım Murat Şimşek which is accepted by his supervisor. This work is acceptable as a graduation project and the student is eligible to take the graduation project examination.

15/01/2018
Prof. Dr. Özgür Özlük

We hereby state that we have held the graduation examination of __________ and agree that the student has satisfied all requirements.

THE EXAMINATION COMMITTEE

Committee Member	Signature
1. Dr. Tuna Çakar	………………………..
2. Prof. Dr. Özgür Özlük	………………………..

Academic Honesty Pledge

I promise not to collaborate with anyone, not to seek or accept any outside help, and not to give any help to others.

I understand that all resources in print or on the web must be explicitly cited.

In keeping with MEF University's ideals, I pledge that this work is my own and that I have neither given nor received inappropriate assistance in preparing it.

Name	Date	Signature
Yıldırım Murat ŞİMŞEK	15/01/2018	

ACKNOWLEDGEMENTS

I would like to present my deepest thanks to my capstone supervisor Dr. Tuna Çakar for his valuable guidance, motivation and support throughout this study.

I wish to express my sincere gratitude and appreciation to Prof. Dr. Özgür Özlük. You definitely provided the project that I needed to choose the right direction and successfully complete my capstone study.

In addition, I would like to thank Sanem Balkan from GittiGidiyor --the electronic commerce company-- for supplying the data set that I used in my capstone project.

EXECUTIVE SUMMARY

MARKET BASKET ANALYSIS USING APRIORI ALGORITHM

Yıldırım Murat ŞİMŞEK

Advisor: Dr. Tuna ÇAKAR

JANUARY 2018, 57

Predictive analysis is a branch of data engineering that predicts some occurrence or probabilities depend on the data. To make predictions about future events, predictive analytics uses data mining techniques. The process of these techniques involves an analysis of historic data and predicts the future events based on that analysis. Also using predictive analytics modelling techniques, a model can be created to predict. Depending on the data that they are using these predictive models can be varied. Predictive analytics is made of various statistical and analytical techniques used to develop models that will predict future occurrence, events or probabilities.

Market basket analysis is one of the data mining techniques that focusing on discovering purchasing pattern by extracting associations from a store's transactional data. The electronic commerce point-of-sale expanded the utilization and application of transactional data in Market Basket Analysis. The needs of the customers have to be known and adapted to them from the retailers. The retailers collect information about their customers and what they purchase with the help of the advanced technology. Analysing this information is extremely valuable for understanding purchasing behaviour in retail commerce. Market basket analysis is one possible way to discover which items can be sold together. This analysis gives retailer valuable information about related sales on a group of goods basis customers who buy bread often also buy several products related to bread like milk or butter. It makes sense that these groups are placed side by side in a store so that customers can reach them quickly. Market basket analysis is very useful technique for the related group of products that are bought together, and to reorganize the supermarket layout, and also to design promotional campaigns such that products' purchase can be improved.

The main aim of this capstone project is to find the co-occurring items in consumer shopping baskets in the data set that provided by GittiGidiyor E-Commerce Company with the help of the association rule mining algorithm; apriori. Mining association rules from transactional data will provide us with valuable information about co-occurrences and co-purchases of products. Such information can be used as a basis for decisions about marketing activity such as promotional support, inventory control and cross-sale campaigns.

Key Words: Association Rules, Market Basket Analysis, and Apriori Algorithm

ÖZET

APRIORİ ALGORİTMALARI KULLANARAK PİYASA SEPET ANALİZİ

Yıldırım Murat ŞİMŞEK

Tez Danışmanı: Dr. Tuna ÇAKAR

OCAK, 2018, 57

Tahminli analiz, bazı oluşumları veya olasılıkları verilere bağlı olarak tahmin eden bir veri mühendisliği dalıdır. Tahmini analitik, gelecekteki olaylarla ilgili tahminler yapmak için veri madenciliği tekniklerini kullanır. Bu tekniklerin süreci, tarihi verilerin analizini içerir ve gelecekteki olayları bu analiz temelinde öngörür. Tahmini analitik modelleme tekniklerini de kullanarak, tahmin etmek için bir model oluşturulabilir. Kullandıkları verilere bağlı olarak, bu tahmini modeller çeşitlenebilir. Tahminli analitik, gelecekte oluşumu, olayları veya olasılıkları tahmin edecek modeller geliştirmek için kullanılan çeşitli istatistiksel ve analitik tekniklerden yapılır.

Pazar sepeti analizi, bir mağazanın işlem verilerinden ilişkilendirmeler çıkartarak satın alma modelinin keşfedilmesine odaklanan veri madenciliği tekniklerinden biridir. Elektronik ticaret pazarı, Market Sepeti Analizi'nde işlem verilerinin kullanımını ve uygulanmasını genişletti. Müşterilerin ihtiyaçları bilinmeli ve perakendeciler tarafından uyarlanmalıdır. Perakendeciler, ileri teknoloji yardımı ile müşterileri ve satın aldıkları hakkında bilgi toplamaktadır. Bu bilgiyi analiz etmek, perakende ticarette satın alma davranışını anlamak için son derecede önemlidir. Pazar sepeti analizi, hangi ürünlerin birlikte satılabileceğini keşfetmenin bir yoludur. Bu analiz perakendeciye, bir grup mal bazında ilgili satışlar hakkında değerli bilgiler verir; ekmek satın alan müşterilerin çoğu; süt veya tere yağ gibi ekmekle ilgili birçok ürün satın alırlar. Müşterilerin hızlı bir şekilde ulaşabilmesi için bu grupların bir dükkanda yan yana yerleştirilmesi mantıklıdır. Pazar sepeti analizi, birlikte satın alınan ilgili ürün grubu için çok yararlı bir tekniktir ve süpermarket düzenini yeniden organize eder ve ayrıca ürünlerin satın alınmasının geliştirilebileceği şekilde tanıtım kampanyaları tasarlar.

Bu projenin temel amacı, GittiGidiyor E-Ticaret Şirketinin sağladığı veri setinde tüketici alışveriş sepetlerinde bir arada bulunan ürünleri, Apriori algoritması yardımıyla bulmaktır. İşlem verileri ile madencilik kuralları bize, ürünlerin birlikte gerçekleşme ve ortak satın alımları hakkında değerli bilgiler sağlayacaktır. Bu tür bilgiler, promosyon desteği, envanter kontrolü ve çapraz satış kampanyaları gibi pazarlama faaliyetleri ile ilgili kararlar için temel olarak kullanılabilir.

Anahtar Kelimeler: İlişki Kuralları, Alışveriş Sepet Analizi, Apriori Algoritması

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Overview

The human nature wants to know and predict what the future will be. Predictive analytics handles the prediction of future events depends on prior examined historical data by applying machine learning algorithms. The historical data is gathered and converted by using different techniques like filtering, associating the data, and etc. The major goal in data mining is to create and improve the certainty of predictive models, and a basic challenge lies in the discovery of new features, inputs or predictors.

Market basket analysis is one of the data mining techniques that focusing on discovering purchasing pattern by extracting associations from a store's transactional data. The electronic commerce point-of-sale expanded the utilization and application of transactional data in Market Basket Analysis. The needs of the customers have to be known and adapted to them from the retailers. The retailers collect information about their customers and what they purchase with the help of the advanced technology. Analysing this information is extremely valuable for understanding purchasing behaviour in retail commerce. Market basket analysis is one possible way to discover which items can be sold together. This analysis gives retailer valuable information about related sales on a group of goods basis customers who buy bread often also buy several products related to bread like milk or butter. It makes sense that these groups are placed side by side in a store so that customers can reach them quickly. Market basket analysis is very useful technique for the related group of products that are bought together, and to reorganize the supermarket layout, and also to design promotional campaigns such that products' purchase can be improved.

In this study, I will illustrate how rules generated from Market Basket Analysis may be utilized to improve predictive models.

1.2. Literature Review

Predictive analysis is a branch of data engineering that predicts some occurrence or probabilities depend on the data. To make predictions about future events, predictive analytics uses data mining techniques. Market basket analysis is one of the data mining techniques that focusing on discovering purchasing pattern by extracting associations from a store's transactional data as called mining association rules.

Market basket analysis can be analysed in two methods: explanatory and exploratory. The exploratory analysis is the discovering of purchase patterns from store data. Exploratory approaches do not include information on consumer demographics or marketing mix variables (Katrin Dippold, Harald Hruschka, 2010). Methods like association rules (Rakesh Agrawal, Sirkant Ramakrishnan, 1994) or collaborative filtering (Andreas Mild, Thomas Reutterer, 2003) summarize a large amount of data into meaningful rules. These methods are very useful for discovering unknown relationships between the items in the data. Besides, association rules and collaborative filtering are computationally basic and can be used for undirected data mining. But, exploratory approximations are not suitable for estimation and finding the root cause of complicated problems. These are used to expose selected cross-category interdependencies depended on some frequency patterns for items or product categories purchased together. Defining product category relationships by simple association measures is the common practice of these exploratory approximations.

The processes of generating association rules are the important part of research in the area of exploratory analysis. The important numbers of algorithms for mining models from market basket data have been published. Rakesh Agrawal and Ramakrishnan Srikant present two new algorithms that named Apriori and AprioriTid for exploring huge item sets in databases. Despite the Apriori and AprioriTid are similar in terms of a function that is used to specify the candidate item sets, there is a difference for the AprioriTid. AprioiriTid does not use the database for counting support after the first iteration but Apriori does multiple passes on the database. The outcomes of the study show that Apriori and AprioriTid achieve much better than the already known AIS (R. Agrawal, T. Imielinski, and A. Swami, 1993) and SETM (M. Houtsma and A. Swami, 1993) algorithms. The Apriori algorithm has been regarded the most useful and fast algorithm for finding frequent item set since it introduced. Many improvements have been made on the Apriori algorithm in order to increase its efficiency and effectiveness. (M.J.Zaki, M.Ogihara,S. Parthasarathy, 1996). A small number of algorithms have been developed without being dependent on Apriori, but they still have a speed issue. The papers (Eu-Hong Han, George Karypis, Vipin Kumar, 1999), (Jong Soo Park, Ming-Syan Chen, Philip S. Yu) recommend new algorithms, which are not depend on the Apriori, but these are being compared to Apriori in terms of execution time.

Exploratory models are very useful for discovering unknown relationships between the items in the data but are not suitable for estimation and finding the root cause of complicated problems. During the major aim of exploratory market basket analysis shows the hidden relationships between the product categories, explanatory models consider the explaining effects. The aim of explanatory models is to define and determine cross-category selection impacts of marketing variables, like price, promotion and other marketing features. (Andreas Mild, Thomas Reutterer, 2003) Most of the explanatory models rely greatly on regression analysis and multivariate logistic model.

As a result, given the quantitative nature of the field of data mining, most of the literature on that topic proposes different algorithms and techniques for optimised mining and generation of association rules. Different techniques are needed for different objectives.

2. ABOUT THE DATA

2.1 General Description of Data Set

GittiGidiyor was founded in 2001, and became Turkey's leading e-commerce marketplace over the next 16 years. After becoming a part of the global e-commerce giant eBay in 2011, the company further strengthened its position as the industry's leader. With 60 million monthly visits on average and nearly 19 million registered users, GittiGidiyor is the most preferred online shopping site in Turkey today. GittiGidiyor hosts millions of products at accessible prices as a secure shopping platform where individual sellers, as well as SMEs and large enterprises open stores and grow their businesses. Standing apart with over 15 million products in more than 50 categories, GittiGidiyor uses a "Zero Risk" payment and confirmation system to provide a 100% safe method for online transactions. The site where an item is sold almost every second receives 64 percent of its traffic from mobile, thanks to its mobile app that have been downloaded 5 million times and mobile-responsive shopping screens.

In this study, the used dataset was given from GittiGidiyor and that dataset encloses the shopping records of customers. The dataset includes seven months of data that all the value in the columns was masked because of the privacy of the company. From the given dataset have seven columns that detailed below:

i. Payment code: unique transaction id for a sale
ii. Member id: given a unique identity for each customer
iii. Product id: given a unique identity for each product
iv. Category id: given a unique identity for each category
v. Catalog id: given a unique identity for each catalogue
vi. Product retail variant id: not given too much information about that. (Out of scope of this study)
vii. Colit Sales: shows that the customer has bought the relevant product several times in a transaction. (Out of scope of this study)

2.2 Data Pre-Processing

I imported the raw data by using Pandas read_csv function. The details as seen below, I dropped two columns (Product_Retail_Variant_Id and Colit_Sales) that we did not interest. Then for the eye readability, I made all the columns lowercase and converted all the null catalog_id values to No_CatalogId. I printed out the first and last five rows as below.

```
# Importing the dataset
dataset = pd.read_csv('Data.csv')

# Drop a variable (column)
# axis=1 denotes that we are referring to a column, not a row
dataset = dataset.drop('PRODUCT_RETAIL_VARIANT_ID', axis = 1)
dataset = dataset.drop('COLIT_SALES', axis = 1)

# Makes all columns name lowercase
dataset.columns = map(str.lower, dataset.columns)

# Impute NaN's (Null)
dataset['catalog_id'] = dataset['catalog_id'].fillna("No_CatalogId")

# Print first 5 row
dataset.head()

# Print last 5 row
dataset.tail()
```

	member_id	payment_code	urun_id	catalog_id	catc
0	1074756	81019987	172042481	893	tc
1	8979977	77204612	199339503	5723	tc
2	1265467	74514720	213766121	4165	tc
3	7737895	74738058	213766121	4165	tc
4	5966188	75406341	217300036	5219	tc

	member_id	payment_code	urun_id	catalog_id	catc
485579	1005539835	81021917	278515965	5452	tc
485580	10258080	81021882	271382196	No_CatalogId	taf
485581	3100132	81021813	271722290	8909	tc
485582	865297	81021730	277734211	No_CatalogId	tan2
485583	1003336955	81021357	276902499	No_CatalogId	tah

Figure 1: Data Overview

In the dataset, we have approximately 486K rows that are the market-basket transactions that made from indistinct members.

```
dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 485584 entries, 0 to 485583
Data columns (total 5 columns):
member_id       485584 non-null int64
payment_code    485584 non-null int64
urun_id         485584 non-null int64
catalog_id      485584 non-null object
catc            485584 non-null object
dtypes: int64(3), object(2)
memory usage: 18.5+ MB
```

Figure 2: Data Inspection

<u>The inspection output tells:</u>

- **i.** It's an instance of a Data Frame.
- **ii.** Each row was assigned an index of 0 to N-1; 0 to 485583
- **iii.** There are 485584 rows.
- **iv.** Our dataset has five total columns, one of that is missing some values (catalog_id).
- **v.** The last data types of each column, but not necessarily in the corresponding order to the listed columns. As seen below, we should use the dtypes method to get the data type for each column.
- **vi.** An approximate amount of RAM used to hold the Data Frame: 18.5+ MB

As seen in figure 2, we saw that we have integer and object columns as a type. Before making analysis, we should change their types to categorical value because each value represents a string and we cannot calculate their mean, median etc. Below, you can find how I change the column data types.

```
dataset['member_id'] = dataset['member_id'].astype("category")
dataset['payment_code'] = dataset['payment_code'].astype("category")
dataset['urun_id'] = dataset['urun_id'].astype("category")
dataset['catalog_id'] = dataset['catalog_id'].astype("category")
dataset['catc'] = dataset['catc'].astype("category")

print ("The datatype for each column: \n\n" + str(dataset.dtypes))

The datatype for each column:

member_id       category
payment_code    category
urun_id         category
catalog_id      category
catc            category
dtype: object
```

Figure 3: Converting Quantitative Variables To Categorical Variables

From the outputs below, we can see that the count of member id and the count of payment code are different. That's means; one customer has more than one transactions. Which is why we should handle that problem in the later sections.

```
#Total count of unique values in the dataframe['member_id'] column
print ('\x1b[1;04;31;49m'+"Total count of unique member id:"+'\x1b[0m',len(dataset.member_id.unique()))

#Total count of unique values in the dataframe['payment_code'] column
print ('\x1b[1;04;31;49m'+"Total count of unique payment code:"+'\x1b[0m',len(dataset.payment_code.unique()))

#Total count of unique values in the dataframe['urun_id'] column
print ('\x1b[1;04;31;49m'+"Total count of unique urun id:"+'\x1b[0m',len(dataset.urun_id.unique()))

#Total count of unique values in the dataframe['catalog_id'] column
print ('\x1b[1;04;31;49m'+"Total count of unique catalog id:"+'\x1b[0m',len(dataset.catalog_id.unique()))

#Total count of unique values in the dataframe['catc'] column
print ('\x1b[1;04;31;49m'+"Total count of unique catc:"+'\x1b[0m',len(dataset.catc.unique()))
```

```
Total count of unique member id: 300252
Total count of unique payment code: 422591
Total count of unique urun id: 160897
Total count of unique catalog id: 2194
Total count of unique catc: 40
```

Figure 4: Total Unique Values For Each Column

3. PROJECT DEFINITION

3.1 Problem Statement

In the dataset, we have approximately 486K rows that are the market-basket transactions that made from indistinct members. Retailers' focus is to understand the dependencies between acquisitions. Consumers buy various product combinations on a single shopping trip, but the selection scenario does not seem random.

3.2 Problem Objectives

The main aim of this capstone project is to find the co-occurring items in consumer shopping baskets in the data set that provided by GittiGidiyor E-Commerce Company with the help of the association rule mining algorithm; apriori. Mining association rules from transactional data will provide us with valuable information about co-occurrences and co-purchases of products. Such information can be used as a basis for decisions about marketing activity like promotional support, inventory control and cross-sale campaigns.

3.3 Project Scope

Scope of the capstone:

- Making assumptions, identifying interesting approaches and gaining insight into data via EDA
- Finding products which are purchased together without machine learning
- Creating models depending on EDA result for Apriori algorithm
- Finding the co-occurring items in consumer shopping baskets by using Apriori algorithm via Python, Jupyter

4. METHODOLOGY

4.1 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is an approach/philosophy for data analysis that employs a variety of techniques (mostly graphical) to maximize insight into a data set, uncover underlying structure, extract important variables, detect outliers and anomalies, test underlying assumptions and develop parsimonious models.

Starting from this step, I'm going to provide insight on information from data by using graphs. Such as we will see most sold products, most product-buying customers and top selling categories.

4.1.1 The Customers Who Purchase The Max. Number of Products

To get the figure 5, first I grouped all the member_ids' by payment_code, urun_id, catalog_id and catc. Then, calculated total distinct counts for each column except null values and sorted the top 10 result descending by urun_id. As seen below, the member who the number is 5415872 bought 179 products.

	payment_code	urun_id	catalog_id	catc
member_id				
541572	45	179	72	4
660201	4	159	1	7
1000706717	48	124	4	14
2253846	7	120	1	8
591841	85	94	39	7
8911458	50	78	4	9
8243927	11	74	6	3
8840101	11	71	1	6
13577521	54	70	9	8
380412	62	70	45	3

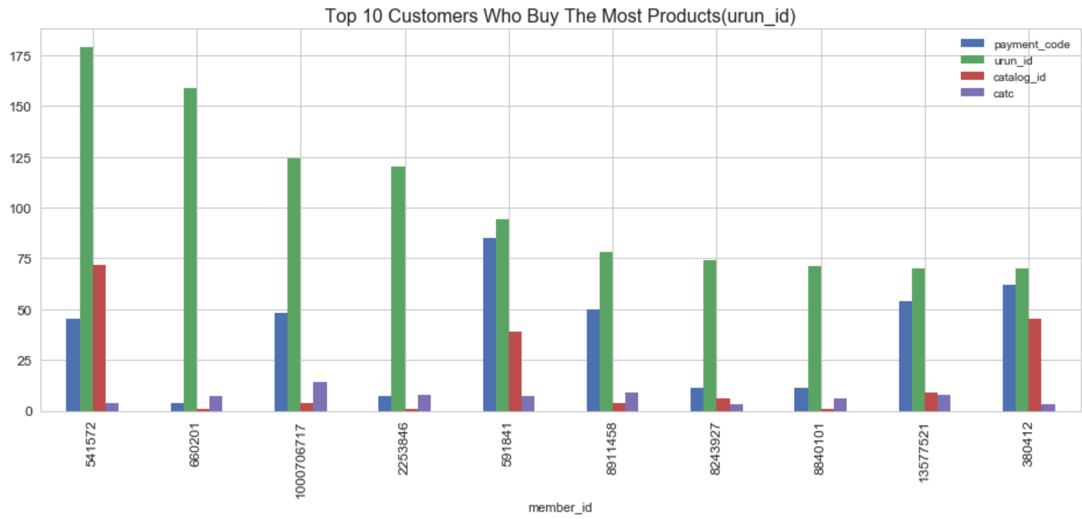

Figure 5: Top 10 Customers Who Buy The Most Products

In the figure 5, the customer who the member id is 660201 bought 159 products; he/she is the second customer who purchases the max. number of products but this customer bought these items from 1 catalog in 4 transactions. From this result, I may think that the products that associated with this catalog_id, is the top selling products.

4.1.2 The Transactions Which Have The Max. Numbers of Products

To get the figure 6, first I grouped all the payment_codes' by urun_id, catalog_id and catc. Then, calculated total distinct counts for each column except null values and sorted the top 10 result descending by urun_id. As seen below, the transaction that is the payment code is 75101902 has the most products.

	urun_id	catalog_id	catc
payment_code			
75101902	94	1	2
75102517	54	1	4
69208430	54	1	3
79859572	50	1	5
80108186	49	1	6
72853008	39	1	3
80685065	37	1	2
80648891	37	1	1
75658025	36	1	2
69938419	36	1	2

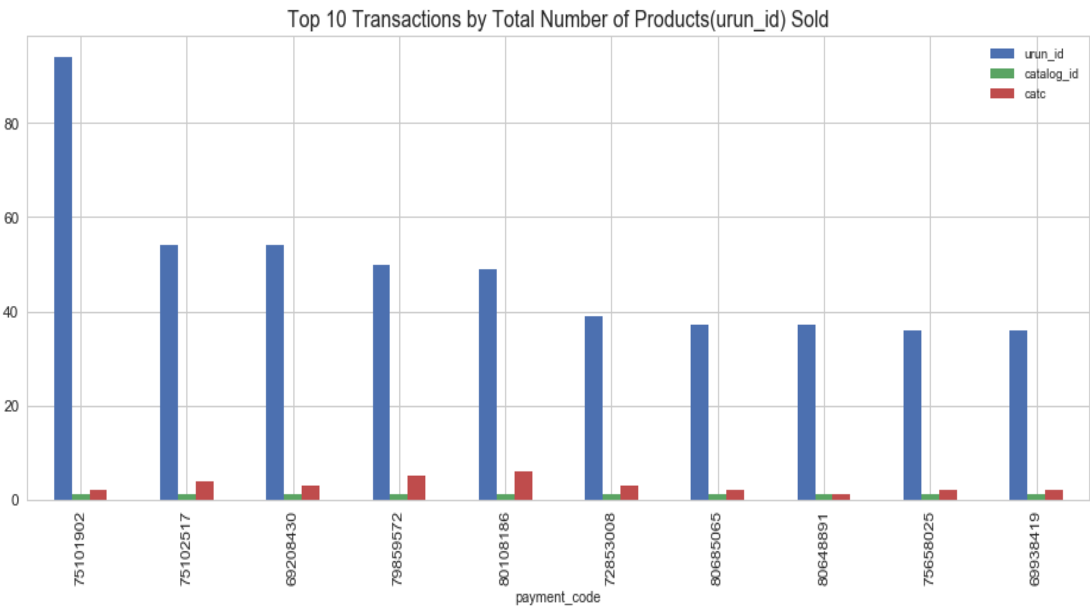

Figure 6: Top 10 Transactions by Total Number of Products

4.1.3 What Are The Top Selling Products?

To get the figure 7, first I grouped all the urun_ids' by payment_code, catalog_id and catc. Then, calculated total distinct counts for each column except null values and sorted the top 10 result descending by urun_id. As seen below, the top selling product is 250654478.

	payment_code	catalog_id	catc
urun_id			
250654478	1428	2	1
270093958	1332	3	1
259645425	1064	2	1
258381603	874	2	1
251511636	859	2	1
251875865	853	2	1
265964602	806	2	1
265622715	693	3	1
241000265	590	1	1
252466746	582	4	1

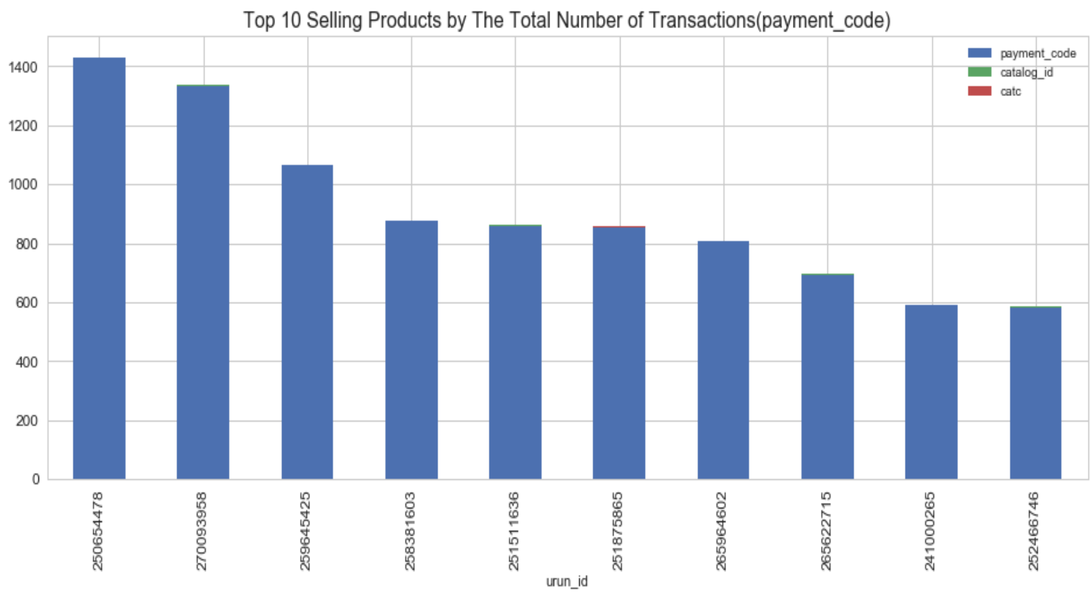

Figure 7: Top 10 Selling Products by Total Number of Transactions

4.1.4 What Are The Top Selling Categories?

To get the figure 8, first I grouped all the catc by payment_code, and urun_id. Then, calculated total distinct counts for each column except null values and sorted the top 10 result descending by urun_id. As seen below, the top selling category is "taf". And if we categorize the top selling categories by using the number of products and the number of payments, we can create the graph seen in figure 9 below.

	catc	Total_Count_of_Urun_id	Total_Count_of_payment_code
0	taf	78063	150830
1	tc	16304	117191
2	tah	19672	36560
3	tan5	9944	26244
4	tae1	9275	23821
5	tan2	3997	23523
6	taz	7313	15689
7	tada	1469	9439
8	tabc	1909	6704
9	ta1a	1809	6639

Figure 8: Top Selling Categories

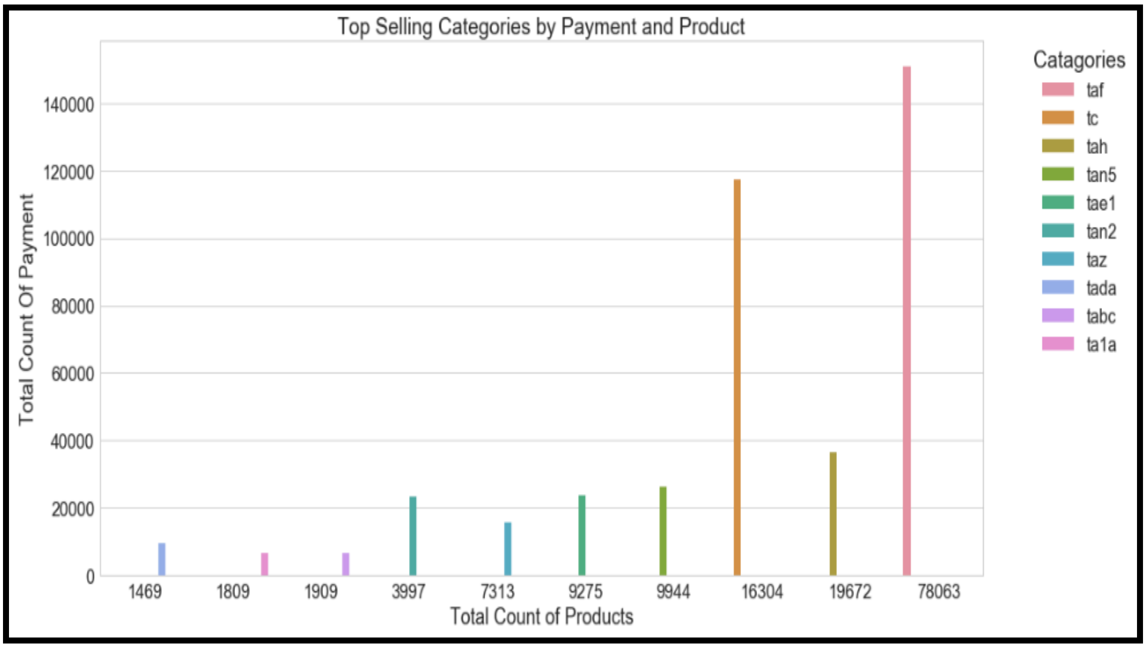

Figure 9: Top Selling Categories by Payment and Product

4.1.5 What Are The Top Selling Catalog Ids

To get the figure 10, first I grouped all the catalog_id by payment_code, and urun_id. Then, calculated total distinct counts for each column except null values and sorted the top 10 result descending by urun_id. As seen below, we can say that we don’t have catalog_id on the most of the transactions and products.

	catalog_id	Total_Count_of_Urun_id	Total_Count_of_payment_code
0	No_CatalogId	148739	332222
1	4168	155	1994
2	7284	120	125
3	908	119	251
4	355	113	2419
5	909	112	269
6	5246	108	978
7	907	106	307
8	929	102	595
9	7329	98	550

Figure 10: Top Selling Categories by Payment and Product

4.2 Finding Products Which Purchased Together Without Machine Learning

In this section, I will show how do I find the products that purchased together without machine learning by using transposing, feature extraction and binning.

4.2.1 Transposing and Feature Extraction From Dataset

When we look at the raw data set, we saw that we have more than one member ids' which are related the different payments in the member_id column. Because of this situation, I transposed the data set and extracted new columns for making good assumption and analysis. The code seen in figure 11 is taking the unique transactions (payment_code), creating new data set from each of them and then creating list of payments, list of products, list of unique products, list of catalogue ids and list of catalogues. With the help of this code, we have a list of unique transactions (payment_code) that have a list of purchased products. For example, the transaction (payment_code) 81020848 has the list of products which are [270203841, 273939543, 270945965] as seen from the figure 12 below.

```python
import time

dct={'member_id':None,'payment_code':None,'urun_id':None, 'urun_id_count':None,'distinct_urun_id': None,'distinct_urun
_id_count':None,'catalog_id':None,'distinct_catalog_id_count':None,'catc':None,'distinct_catc_count':None}
d = OrderedDict(dct)
dlist=[]

def combine_all(x):
        murat = dataset[dataset['payment_code'] == x]
        for i in murat:
            if (i == 'member_id') or (i == 'payment_code'):
                    d[i] = list(set(murat[i].values))[0]
            elif (i == 'urun_id'):
                    d[i] = list(murat[i].values)
                    d['urun_id_count'] = len(list(murat[i].values))
                    d['distinct_urun_id'] = set(list(murat[i].values))
                    d['distinct_urun_id_count'] = len(set(list(murat[i].values)))
            elif (i == 'catalog_id'):
                    d[i] = list(murat[i].values)
                    d['distinct_catalog_id_count'] = len(set(list(murat[i].values)))
            elif (i == 'catc'):
                    d[i] = list(murat[i].values)
                    d['distinct_catc_count'] = len(set(list(murat[i].values)))
        dlist.append(d.copy())
        d.clear()

start = time.time()
for j in dataset.payment_code.unique():
    combine_all(j)
end = time.time()

hours, rem = divmod(end - start,3600)
minutes, seconds = divmod(rem,60)

"Elapsed Time = {:0>2}:{:0>2}:{:05.2f}".format(int(hours),int(minutes),int(seconds))
transposed_dataset=pd.DataFrame.from_dict(dlist)

transposed_dataset
```

Figure 11: Transposing And Feature Extraction

	member_id	payment_code	urun_id	urun_id_count	distinct_urun_id	distinct_urun_id_count	catalog_id
0	1074756	81019987	[172042481]	1	{172042481}	1	[893.0]
1	8979977	77204612	[199339503]	1	{199339503}	1	[5723.0]
2	1265467	74514720	[213766121]	1	{213766121}	1	[4165.0]
3	7737895	74738058	[213766121]	1	{213766121}	1	[4165.0]

distinct_catalog_id_count	catc	distinct_catc_count
1	[tc]	1
1	[tc]	1
1	[tc]	1

...	...	...	...	...	...	...	...
422561	1337032	81020929	[277028190]	1	{277028190}	1	[No_CatalogId]
422562	13555919	81020869	[272377852]	1	{272377852}	1	[No_CatalogId]
422563	10882192	81020848	[270203841, 273939543, 270945965]	3	{270203841, 270945965, 273939543}	3	[No_CatalogId, No_CatalogId, No_CatalogId]

1	[tabc]	1
1	[tah]	1
1	[tae1, tae1, tan5]	2

Figure 12: Transposed And Feature Extracted Dataset

4.2.2 Bucketing (Binning) By Using Sold Products Count

The code below in figure 13 is creating two new columns which their names are "amount of products in the basket bin" and "most preferred amount of products in the basket bin" as seen in the right hand side of the table (Figure 14 and 15). To create these columns, first I created two bins by using product counts in transposed data set. Then, counted the transactions by using these two new columns. In the end, I saw that the huge number of transactions (421.935) was accomplished with minimum one product or maximum five products. If I focus the most preferred bucket that is [0,1,2,3,4,5,7,9,11], I saw that the huge number of transactions (382.698) were made with only one product. You can easily see these results below; I highlighted them with yellow in Figure 16.

```
def highlight_max(s):
    '''
    highlight the maximum in a Series yellow.
    '''
    is_max = s == s.max()
    return ['background-color: yellow' if v else '' for v in is_max]

#np.unique(transposed_dataset['distinct_urun_id_count'].values)
urun_bins = [0,5,10,15,20,25,30,35,40,45,50,100]
most_sold_urun_bins = [0,1,2,3,4,5,7,9,11]

dataset_urun_bins = transposed_dataset.copy(deep=True)
dataset_most_sold_urun_bins = transposed_dataset.copy(deep=True)

dataset_urun_bins['amountOfProducts_InThe_Basket_Bin'] = pd.cut(dataset_urun_bins['distinct_urun_id_count'],urun_bins)
dataset_most_sold_urun_bins['mostPreferredAmount_OfProducts_InThe_Basket_Bin'] = pd.cut(dataset_most_sold_urun_bins['d
istinct_urun_id_count'],most_sold_urun_bins)

x =dataset_urun_bins.groupby(['amountOfProducts_InThe_Basket_Bin']).agg({'amountOfProducts_InThe_Basket_Bin': ['count'
]})
x.columns = ["_".join(x) for x in x.columns.ravel()]

y = dataset_most_sold_urun_bins.groupby(['mostPreferredAmount_OfProducts_InThe_Basket_Bin']).agg({'mostPreferredAmount
_OfProducts_InThe_Basket_Bin': ['count']})
y.columns = ["_".join(x) for x in y.columns.ravel()]

dataset_urun_bins.head()
dataset_most_sold_urun_bins.head()
x.reset_index().rename(columns={'amountOfProducts_InThe_Basket_Bin_count':'count'}).style.apply(highlight_max)
y.reset_index().rename(columns={'mostPreferredAmount_OfProducts_InThe_Basket_Bin_count':'count'}).style.apply(highligh
t_max)
```

Figure 13: Bucketing By Using Sold Products Count

	member_id	payment_code	urun_id	urun_id_count	distinct_urun_id	distinct_urun_id_count	catalog_id	distinct_catalog_id_count
0	1074756	81019987	[172042481]	1	{172042481}	1	[893.0]	1
1	8979977	77204612	[199339503]	1	{199339503}	1	[5723.0]	1
2	1265467	74514720	[213766121]	1	{213766121}	1	[4165.0]	1
3	7737895	74738058	[213766121]	1	{213766121}	1	[4165.0]	1
4	5966188	75406341	[217300036]	1	{217300036}	1	[5219.0]	1

catc	distinct_catc_count	amountOfProducts_InThe_Basket_Bin
[tc]	1	(0, 5]
[tc]	1	(0, 5]
[tc]	1	(0, 5]
[tc]	1	(0, 5]
[tc]	1	(0, 5]

Figure 14: Amount of Products In The Basket Bin For Each Transaction

	member_id	payment_code	urun_id	urun_id_count	distinct_urun_id	distinct_urun_id_count	catalog_id	distinct_catalog_id_count
0	1074756	81019987	[172042481]	1	{172042481}	1	[893.0]	1
1	8979977	77204612	[199339503]	1	{199339503}	1	[5723.0]	1
2	1265467	74514720	[213766121]	1	{213766121}	1	[4165.0]	1
3	7737895	74738058	[213766121]	1	{213766121}	1	[4165.0]	1
4	5966188	75406341	[217300036]	1	{217300036}	1	[5219.0]	1

catc	distinct_catc_count	mostPreferredAmount_OfProducts_InThe_Basket_Bin
[tc]	1	(0, 1]
[tc]	1	(0, 1]
[tc]	1	(0, 1]
[tc]	1	(0, 1]
[tc]	1	(0, 1]

Figure 15: Most Preferred Amount of Products In The Basket Bin For Each Transaction

	amountOfProducts_InThe_Basket_Bin	count
0	(0, 5]	421935
1	(5, 10]	562
2	(10, 15]	55
3	(15, 20]	17
4	(20, 25]	4
5	(25, 30]	5
6	(30, 35]	3
7	(35, 40]	5
8	(40, 45]	0
9	(45, 50]	2
10	(50, 100]	3

	mostPreferredAmount_OfProducts_InThe_Basket_Bin	count
0	(0, 1]	382698
1	(1, 2]	30631
2	(2, 3]	6261
3	(3, 4]	1756
4	(4, 5]	589
5	(5, 7]	411
6	(7, 9]	123
7	(9, 11]	48

Figure 16: Bucketing Results

4.2.3 Bucketing (Binning) By Using Categories

The code in the figure 17 below is creating a new column which its names is "amount of categories in the basket bin" as seen in the right hand side of the table (Figure 18). To create that column, I used same logic seen on the one step before. After execution of this code, I saw that the huge number of transactions (399.692) was made with only one category. You can easily see that below, I highlighted them with yellow seen in figure 19.

```
def highlight_max(s):
    '''
    highlight the maximum in a Series yellow.
    '''
    is_max = s == s.max()
    return ['background-color: yellow' if v else '' for v in is_max]

#np.unique(transposed_dataset['distinct_catc_count'].values)
catc_bins = [0,1,2,3,4,5,6,7]

dataset_catc_bins = transposed_dataset.copy(deep=True)

dataset_catc_bins['amountOfCategories_InThe_Basket_Bin'] = pd.cut(dataset_catc_bins['distinct_catc_count'],catc_bins)

c =dataset_catc_bins.groupby(['amountOfCategories_InThe_Basket_Bin']).agg({'amountOfCategories_InThe_Basket_Bin': ['co
unt']})
c.columns = ["_".join(x) for x in c.columns.ravel()]

dataset_catc_bins.head()
c.reset_index().rename(columns={'amountOfCategories_InThe_Basket_Bin_count':'count'}).style.apply(highlight_max)
```

Figure 17: Bucketing By Using Categories

	member_id	payment_code	urun_id	urun_id_count	distinct_urun_id	distinct_urun_id_count	catalog_id	distinct_catalog_id_count	catc	distinct_catc_count	amountOfCategories_InThe_Basket_Bin
0	1074756	81019987	[172042481]	1	{172042481}	1	[893.0]	1	[tc]	1	(0, 1]
1	8979977	77204612	[199339503]	1	{199339503}	1	[5723.0]	1	[tc]	1	(0, 1]
2	1265467	74514720	[213766121]	1	{213766121}	1	[4165.0]	1	[tc]	1	(0, 1]
3	7737895	74738058	[213766121]	1	{213766121}	1	[4165.0]	1	[tc]	1	(0, 1]
4	5966188	75406341	[217300036]	1	{217300036}	1	[5219.0]	1	[tc]	1	(0, 1]

Figure 18: Amount of Categories In The Basket Bin

	amountOfCategories_InThe_Basket_Bin	count
0	(0, 1]	399692
1	(1, 2]	20323
2	(2, 3]	2195
3	(3, 4]	317
4	(4, 5]	45
5	(5, 6]	16
6	(6, 7]	3

Figure 19: Bucketing Results

4.2.4 Which Two Products Sold Together?

At this step, I focused the most preferred amount of products sold in the baskets that it is 2. Depending on the figure 16, 30.631 transactions, only two products were sold. The code in the figure 20 is creating tuples where distinct product count is 2. After that, creating list of distinct products then searching and appending the products that sold together.

```
# Creating tuples where distinct_urun_id_count is 2 or more
transposed_dataset.loc[transposed_dataset.distinct_urun_id_count >= 2 ,'distinct_urun_id'] = transposed_dataset.loc[:,
'distinct_urun_id'].apply(tuple)

# the transactions which 2 products were sold
distinct_urun_id = transposed_dataset[transposed_dataset.distinct_urun_id_count == 2].sort_values(['distinct_urun_id_c
ount'], ascending = False)['distinct_urun_id'].tolist()
urun_id_dict_list = {}
urun_id_list = []
list_tuple_items = []

for a,b in distinct_urun_id:
    list_tuple_items.append(a)
    list_tuple_items.append(b)

start = time.time()
for x in list(set(list_tuple_items)):
    for a,b in distinct_urun_id:
        if a == x:
            urun_id_list.append(tuple([a,b]))
    urun_id_dict_list[x] = urun_id_list.copy()
    urun_id_list.clear()
end = time.time()

hours, rem = divmod(end - start,3600)
minutes, seconds = divmod(rem,60)
"Elapsed Time = {:0>2}:{:0>2}:{:05.2f}".format(int(hours),int(minutes),int(seconds))
```

Figure 20: Creating tuples where distinct product count is 2

As seen in the figure 21 and 22, I printed out the each product that sold together. We can read this output like that the product "275306762" sold together with these 3 products 267890068, 277637271, 274524382.

```
listOfproducts = {k: v for k,v in urun_id_dict_list.items() if len(v) >= 2}
pd.set_option('display.max_colwidth', -1)
pd.DataFrame(pd.Series(listOfproducts), columns=['urun_ids\' sold together']).tail(100)
```

	urun_ids' sold together
275306762	[(275306762, 267890068), (275306762, 277637271), (275306762, 274524382)]
275329469	[(275329469, 274308351), (275329469, 276737494)]
275329472	[(275329472, 277861716), (275329472, 277861716)]
275450273	[(275450273, 272031658), (275450273, 272748094)]
275450368	[(275450368, 275451948), (275450368, 274881825)]
275461760	[(275461760, 257784086), (275461760, 271203427)]
275489948	[(275489948, 270744260), (275489948, 275598829)]
275491877	[(275491877, 268960318), (275491877, 275021310), (275491877, 268960318), (275491877, 275021310), (275491877, 274517871), (275491877, 275021310), (275491877, 273219239), (275491877, 275021310)]

Figure 21: Creating tuples where distinct product count is 2

276053523	[(276053523, 260492636), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058804), (276053523, 276058804), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 260491195), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 260491195), (276053523, 260491195), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276058803), (276053523, 276368630), (276053523, 276058803)]
276058803	[(276058803, 273830109), (276058803, 260491195), (276058803, 260491195), (276058803, 260491195)]
276058824	[(276058824, 273600357), (276058824, 272321984)]
276058834	[(276058834, 276058803), (276058834, 276058803), (276058834, 276058803), (276058834, 276058803), (276058834, 276058803)]
276080601	[(276080601, 271042365), (276080601, 272639956), (276080601, 277996022), (276080601, 274608122), (276080601, 272050423), (276080601, 271042365), (276080601, 272639956)]
...	...

Figure 22: Creating tuples where distinct product count is 2

4.2.5 Visualizing The Products Which Purchased Together

The code seen below in figure 23 is drawing tree graph for the products that sold together. That code is using the outputs of figure 21 and 22 as an input. I showed the relations between products below in figure 24, 25 and 26. We can say that these products purchased together and may be purchased together in the future.

```
import networkx as nx
import warnings
warnings.filterwarnings('ignore')
def hierarchy_pos(G,root,width=1., vert_gap=0.2, vert_loc=0,xcenter= 0.5,pos=None,parent=None):
    if pos ==None:
        pos = {root:(xcenter,vert_loc)}
    else:
        pos[root] = (xcenter, vert_loc)
    neighbors = G.neighbors(root)
    if parent != None:
        neighbors.remove(parent)
    if len(neighbors) != 0:
        dx =width/len(neighbors)
        nextx = xcenter -width/2 -dx/2
        for neighbor in neighbors:
            nextx += dx
            pos = hierarchy_pos(G,neighbor,width = dx, vert_gap = vert_gap, vert_loc= vert_loc-vert_gap,xcenter=nextx,
pos=pos,parent=root)
    return pos
```

Figure 23: Drawing tree graph

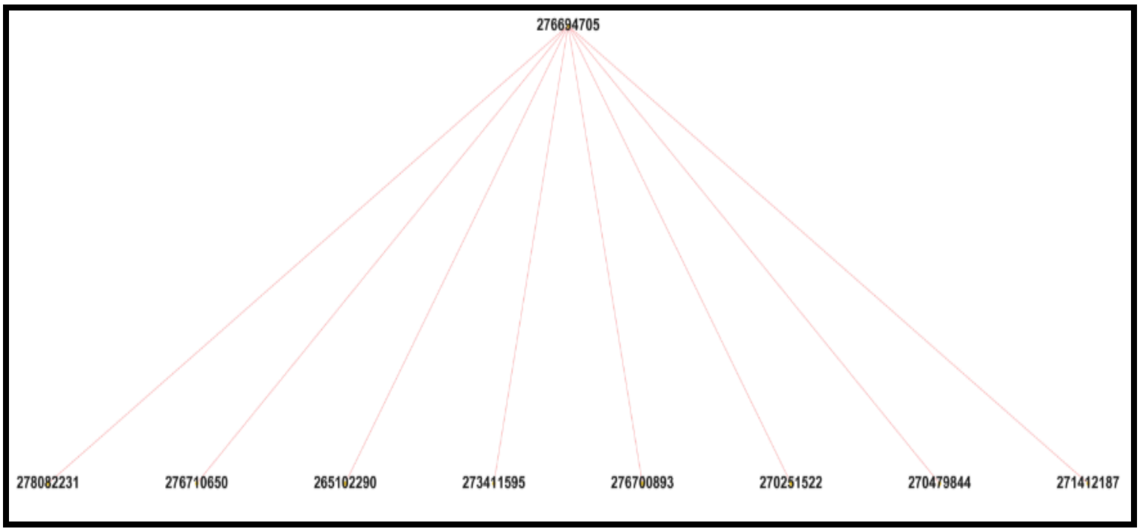

Figure 24: Who Bought 276694705, Also Bought The Other Products In The Tree

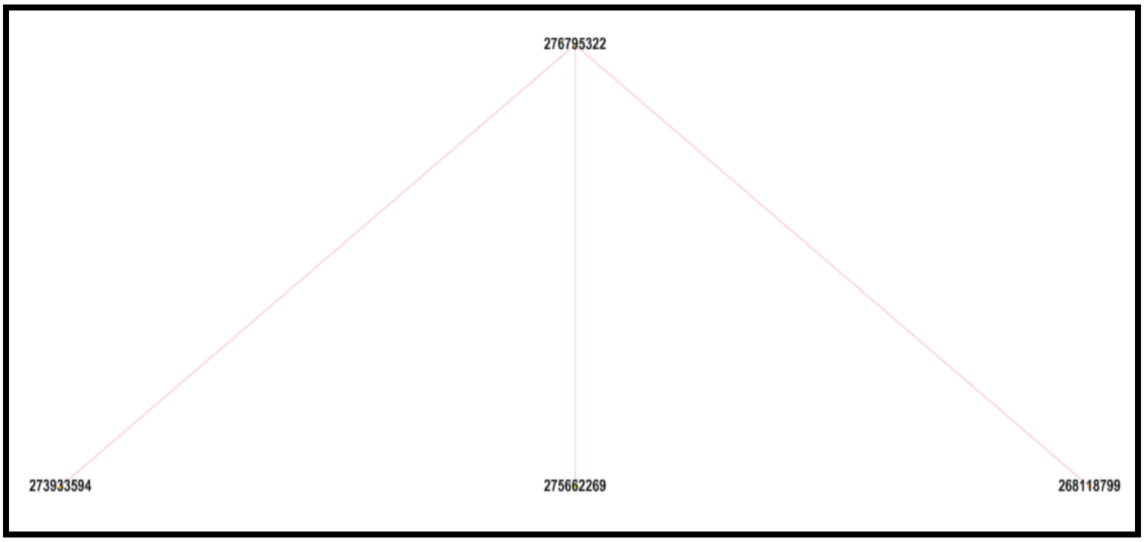

Figure 25: Who Bought 276795322, Also Bought The Other Products In The Tree

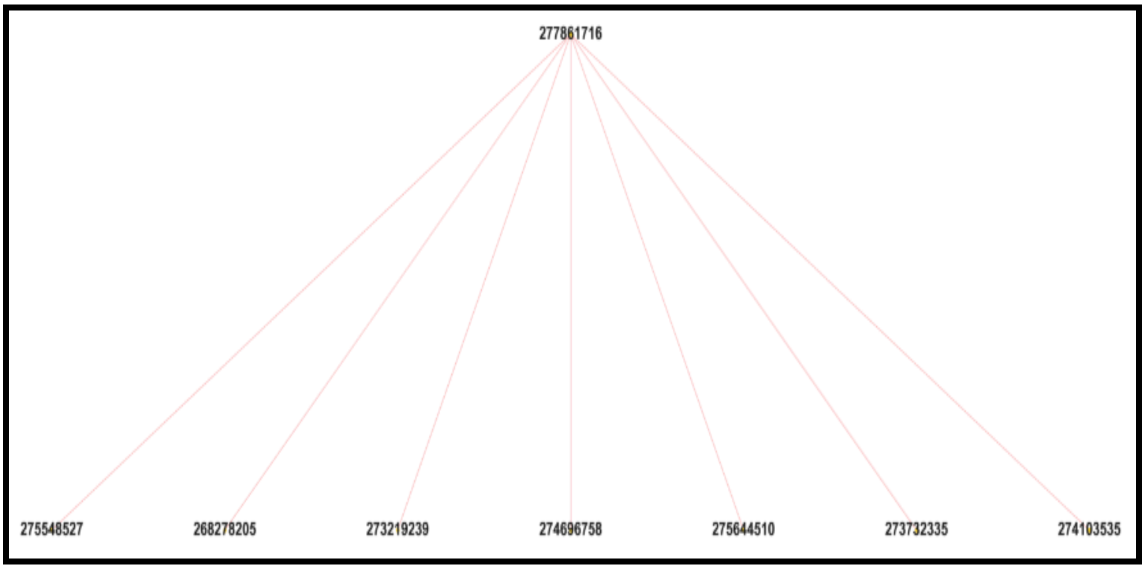

Figure 26: Who Bought 277861716, Also Bought The Other Products In The Tree

4.3 Machine Learning with Apriori Algorithm

Let’s say we have a store that is located in one of the most popular places in Istanbul. So a lot of people go into the store and we know this place is a very convivial place, a very friendly place where people love to hang out, relax, talk to each other. And so these people come very often to this store because even if it’s not buying something it’s a place to meet their friend.

Manager of this store wants to optimize the placement of its different products to optimize the sales. The manager noticed and calculated that on average each customer goes and buy something to the store once a week. This data set contains the 486K transactions of all the different customers that bought a basket of products in a whole 7 months. Indeed, the manager took it as the basis of its analysis because since each customer is going an average once a week to the store then the transaction registered over a week is quite representative of what customers want to buy. So based on my story and all these 486K transactions which algorithm should I use to implement machine-learning model?

After all the steps told above sections, I came through to the association analysis that is useful for discovering interesting relationships hidden in large data sets. I used Apriori algorithm from association rule learning models such as FP-Growth, DHP etc. In Apriori, the uncovered relationships can be represented in the form of association rules. It is one of the earliest algorithms to have successfully addressed the combinatorial explosion of frequent item set generation. It achieves this by applying the Apriori principle to prune the exponential search space. Despite its significant performance improvement, the algorithm still incurs considerable I/O overhead since it requires making several passes over the transaction data set.

So, my Apriori model is going to learn the different associations it can make to actually understand the rules. Such as if customers buy this product then they are likely to buy this other set of products. As a result, that's what I want to figure out and that's what my model will tell us.

4.3.1 How Apriori Algorithm Works?

Association analysis is a job that is finding hidden relationships in large data sets. The hidden relationships are then described as a collection of association rules and frequent item sets. A collection of items that frequently occurred together is called frequent item sets. And association rules suggest a strong relationship that exists between two items. Let's illustrate these two concepts with an example.

Transaction ID	Products Purchased
TRX98283	soy milk, lettuce
TRX00456	lettuce, diapers, wine, chard
TRX09456	soy milk, diapers, wine, orange juice
TRX87456	Lettuce, soy milk, diapers, wine

Table 1: List of Transactions

A list of transactions from a grocery store is shown in the table above. Frequent items are a list of items that commonly appear together. One example is {wine, diapers, soy milk}. From the data set we can also find an association rule such as diapers → wine. This means that if someone buys diapers, there is a good chance they will buy wine.

How do we define the relationships and what is interesting? When we are looking for frequent item sets or association rules, we must look two parameters: the support of an item set and the confidence.

The support is defined as the percentage of the data set that contains this item set. From the table 1 above the support of {soy milk} is 3/4. The support of {soy milk, diapers} is 2/4 because of the four transactions, two contained both soy milk and diapers. Support applies to an item set, so we can define a minimum support and only get the item sets that meet that minimum support.

The confidence is defined for an association rule like diapers → wine. The confidence for this rule is defined as support ({diapers, wine}) / support (diapers). From the table 1 above, support of {diapers, wine} is 3/4. The support for diapers is 3/4, so the confidence for diaper → wine is: 1. That means in 100% of the items in our data set containing diapers our rule is correct. The support and confidence are ways we can quantify the success of our association analysis. To sum up:

$$\textbf{support(i)} = \frac{(number\ of\ transactions\ (i))}{number\ of\ transactions}$$

$$\textbf{confidence(i1}\dashrightarrow\textbf{i2)} = \frac{(number\ of\ transactions\ (i1\ and\ i2))}{number\ of\ transactions\ containing\ i1}$$

$$\textbf{Lift(i1}\dashrightarrow\textbf{i2)} = \frac{(confidence\ (i1\ and\ i2))}{support(i2)}$$

Let's assume that we are running a market store. We are interested in finding out which items were purchased together. We have only four items: item0, item1, item2 and item3. What are all the possible combinations that can be purchased? We can have one item, say item0 alone, or two items, or three items, or all of the items together. If someone purchased two of item0 and four of item2, we don't care. We are concerned only that they purchased one or more of an item. Figure 27 is showing all the possible combinations of the items. To make easier to interpret, we only use the item number such as 0 instead of item0. The first set is a big Ø, which means the null set or a set containing no items. Lines connecting item sets indicate that two or more sets can be combined to form a larger set.

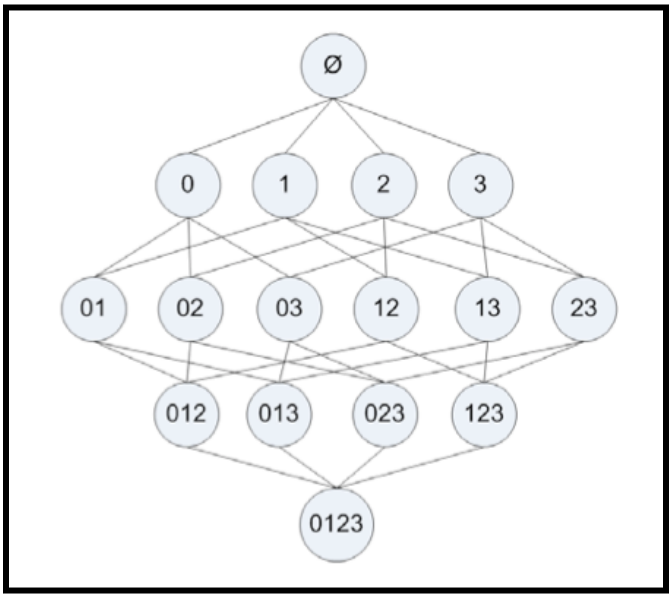

Figure 27: Combinations of Items

Our goal is to find sets of items that are purchased together frequently. The support of a set counted the percentage of transactions that contained that set. If we want to calculate this support for a given set, say {0,3}, we go through every transaction and ask, ”Did this transaction contain 0 and 3?” If the transaction did contain both those items, we increment the total. After scanning all of our data, we divide the total by the number of transactions and we have our support. This result is for only one set: {0,3}. We will have to do this many times to get the support for every possible set.

We can count the sets in Figure 28 below and see that for four items, we have to go over the data 15 times. This number gets large quickly. A data set that contains N possible items can generate 2N-1 possible item sets. Stores selling 10,000 or more items are not uncommon. Even a store selling 100 items can generate 1.26 x 1030 possible item sets. This would take a very, very long time to compute on a modern computer.

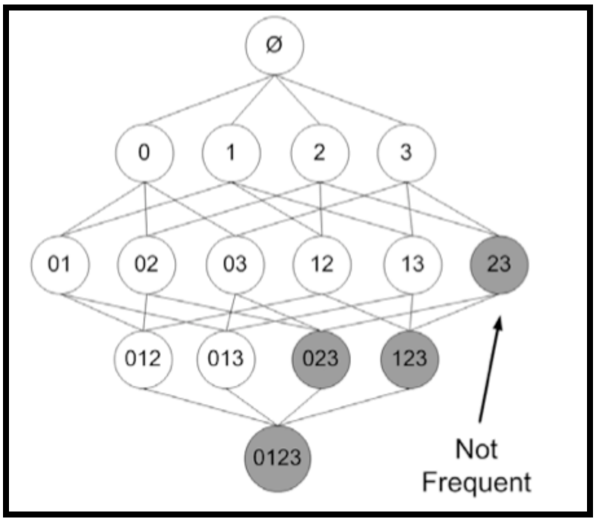

Figure 28: Not Frequent Items

To reduce the time needed to compute this value, researchers identified something called the Apriori principle. The Apriori principle helps us reduce the number of possible interesting item sets. The Apriori principle is: if an itemset is frequent, then all of its subsets are frequent. In Figure 28 above this means that if {0,1} is frequent then {0} and {1} have to be frequent. This rule as it is doesn’t really help us, but if we turn it inside out it will help us. The rule turned around reads: if an item set is infrequent then its supersets are also infrequent, as shown in Figure 28. As you can observe, the shaded item set {2,3} is known to be infrequent. From this knowledge, we know that item sets {0,2,3}, {1,2,3}, and {0,1,2,3} are also infrequent. This tells us that once we have computed the support of {2,3}, we don’t have to compute the support of {0,2,3}, {1,2,3}, and {0,1,2,3} because we know they won’t meet our requirements. Using this principle, we can halt the exponential growth of item sets and in a reasonable amount of time compute a list of frequent item sets. The way to find frequent item sets is the Apriori algorithm. The Apriori algorithm needs a minimum support level as an input and a data set. The algorithm will generate a list of all candidate item sets with one item. The transaction data set will then be scanned to see which sets meet the minimum support level. Sets that don’t meet the minimum support level will get tossed out. The remaining sets will then be combined to make item sets with two elements. Again, the transaction dataset will be scanned and item sets not meeting the minimum support level will get tossed.

To sum up, steps of Apriori algorithm:

i. Set a min. support and confidence.
ii. Take all the subsets in transactions having higher support than min. support.
iii. Take all the rules of these subsets having higher confidence than min. confidence.
iv. Sort the rules by decreasing lift.

4.3.2 Creating Data Set For Model 1 and Usage of Apriori Algorithm

Now, I will create the first dataset of the Apriori algorithm from transposed dataset. As you know, I already transposed the raw dataset before in the section 4.2.1. The first dataset can be summarized as "payment_code - urun_id" which it has the list of purchased product from unique transactions.

```
dataset_of_transaction_list = transposed_dataset['urun_id'].tolist()

print ('\x1b[1;04;31;49m'+"Total Unique Transactions(List of urun_ids):"+'\x1b[0m',len(dataset_of_transaction_list),"
\n")
print ('\x1b[1;04;31;49m'+"Last 10 Items In The Transactions List:"+'\x1b[0m', dataset_of_transaction_list[-10:])

Total Unique Transactions(List of urun_ids): 422591

Last 10 Items In The Transactions List: [[275678672], [276317808], [270720950], [277423270], [269943577, 269840257],
 [278515965], [271382196], [271722290], [277734211], [276902499]]
```

Figure 29: Data set for model 1

As seen in the figure 29, our data set has 422591 unique transactions. If we look the last 10 transactions, we see that we have list of array like [[275678672], [276317808], [270720950], [277423270], [269943577, 269840257], [278515965], [271382196], [271722290], [277734211], [276902499]]. Then, the code below is taking this transactions list as an input to execute the Apriori algorithm and create list of rules. As seen, the apriori fuction is using some keyword arguments like min_support, confidence and lift. I already explained what they are, but now I will give more detail about them, why I chose 0,2 as a min_confidence or 3 as a min_lift etc.

```
from apyori
import apriori rules = apriori(dataset_of_transaction_list,min_support = 0.00001,
min_confidence = 0.2,min_lift =3 ,min_length = 2)
```

Apriori keyword arguments are very important because when we use the Apriori model for our business problem, these arguments will actually depend on our business problem. They will depend on our data set, the number of observations we have. Because min. support is not going to be the same when we have 1000 transactions or 100000 transactions and it is same for the confidence and the lift. So we have to set min. support, then min. confidence, also part of the Apriori model we need to set a min. lift.

Lift is a very important argument because it is one of the best criterion of the strength and the relevance of a rule. Then, we have the min length argument that allows us to set the min. number of products we want to have in our rules. I am using min. length because I want to make associations between at least two different products.

min_support: The support of a set of items i is equal to the number of transactions contained in this set of items i divided by the total number of transactions and the support argument that is actually min. support we want to have in our rules. That means that the items that are going to appear in our rules will have a higher support than min. support. So we must ask ourselves; what supports do we want to have our different items in the rules so that the rules are relevant and show them how to choose to support.

We need to look at the products that are purchase frequently like at least 3 or 4 times a day. Again it depends on our business goal, but if we manage to find some strong rules about items that are bought at least 3 or 4 times a day then by associating them and placing them together, customers will more likely to put them in their basket. Therefore, more of these products will be purchased and the sales will increase. So, to set the min. support, we should consider the products that are purchased at least 3 or 4 times a day and then we will look at rules. If we are not convinced by the rules, we will change this value of the support that is how we work with the Apriori model. We try different values of support, different values of confidence until we are satisfied with the rules, until we think it makes sense. We can also try these rules within a certain period of time, then if we look at the impact on the revenue, we don't observe a meaningful increase in the sales revenue, we can change the support and the confidence to change the rules and then experience again until we find the strongest rules that optimize the sales.

min_confidence: If we follow the R tutorials the Apriori model is implemented in a package called 'arules'. This package contains some default value for the confidence. It started with the default value is 0.8 but we got two obvious rules because the confidence was too high. The confidence of 0.8 means that the rules has to be correct in 80% of the time which is in 80 % of the cases 4 times out of 5. So we get some of these rules like we get some rules containing some products that are most purchased overall and so they are purchased together not because they are well associated but simply each of the products is one of the most purchased products. So they end up in the same basket. Not for the right reason because they associate well together but they are purchased all the time. There is no logical association between these two products and that's why it's not very relevant and unfortunately that's what we will get if we set the confidence too high. So, 0.8 is too high confidence and besides noticed that with 0.8 confidence we actually did not get any rule because there was no rule being correct 80% of the time. What I did afterwards is that I divided this default confidence of 80% by 2 then divided again the confidence by 2 to obtain 0.2 confidence.

min_lift: We can try different values of the min. lift but what we would like to have is some rules that have lift at least higher than 3. If we get some rules that have lift above 3, these are actually some good rules because we know the lift is a great insight of the relevance and the strength of rule and we know we are hoping to find some rules having lift equals to 4, 5 or even 6. So I set this min lift to 3.

4.3.3 Creating Data Set For Model 2 and Usage of Apriori Algorithm

I will create the second dataset of Apriori algorithm from transposed dataset seen in figure 31. It can be summarized as "member_id - urun_id" which it has the list of purchased products from unique member_ids. Then, use same logic before I told about above for Apriori algorithm, the only diffrence is member_id. You can see the first 5 sample of the second dataset in figure 30.

	member_id	payment_code	urun_id	urun_id_count	distinct_urun_id	distinct_urun_id_count	catalog_id
0	1074756	[81019987, 81020231, 70339623, 70339923, 70340...	[172042481, 274971337, 247464804, 247464802, 2...	7	{247464802, 247464804, 274971337, 246335275, 1...	6	[893.0, 6485.0, No_CatalogId, No_CatalogId, No...
1	8979977	[77204612]	[199339503]	1	{199339503}	1	[5723.0]
2	1265467	[74514720, 69438493]	[213766121, 245248154]	2	{213766121, 245248154}	2	[4165.0, No_CatalogId]
3	7737895	[74738058, 71614451, 72030865, 73718586, 74738...	[213766121, 251552445, 248662593, 250413976, 2...	12	{248662593, 263722338, 261209764, 275116839, 2...	12	[4165.0, No_CatalogId, No_CatalogId, No_Catalo...
4	5966188	[75406341, 70311121, 75089099, 77943469]	[217300036, 238769932, 258967328, 265867028]	4	{258967328, 265867028, 217300036, 238769932}	4	[5219.0, No_CatalogId, No_CatalogId, No_Catalo...

Figure 30: Data set for model 2

```
import time

dct={'member_id':None,'payment_code':None,'urun_id':None, 'urun_id_count':None,'distinct_urun_id': None,'distinct_urun
_id_count':None,'catalog_id':None,'distinct_catalog_id_count':None,'catc':None,'distinct_catc_count':None}
d = OrderedDict(dct)
dlist=[]

def combine_all(x):
        murat = dataset[dataset['member_id'] == x]
        for i in murat:
            if (i == 'member_id'):
                    d[i] = list(set(murat[i].values))[0]
            elif (i == 'payment_code'):
                    d[i] = list(murat[i].values)
            elif (i == 'urun_id'):
                    d[i] = list(murat[i].values)
                    d['urun_id_count'] = len(list(murat[i].values))
                    d['distinct_urun_id'] = set(list(murat[i].values))
                    d['distinct_urun_id_count'] = len(set(list(murat[i].values)))
            elif (i == 'catalog_id'):
                    d[i] = list(murat[i].values)
                    d['distinct_catalog_id_count'] = len(set(list(murat[i].values)))
            elif (i == 'catc'):
                    d[i] = list(murat[i].values)
                    d['distinct_catc_count'] = len(set(list(murat[i].values)))
        dlist.append(d.copy())
        d.clear()

start = time.time()
for j in dataset.member_id.unique():
    combine_all(j)
end = time.time()

hours, rem = divmod(end - start,3600)
minutes, seconds = divmod(rem,60)

"Elapsed Time = {:0>2}:{:0>2}:{:05.2f}".format(int(hours),int(minutes),int(seconds))
transposed_dataset_for_member_id=pd.DataFrame.from_dict(dlist)

transposed_dataset_for_member_id
```

Figure 31: Creation of Dataset 2

5. RESULTS

5.1 Results for Model No 1: Payment code to Product Id

I set min_support =0.00001, min_confidence=0.2, min_lift=3 and min_length= 2 and got satisfied the rules for model 1. The results in the table 2 makes sense to us because we have greater than min_lift=3 values. You can see all outputs in the appendix; you can see the first top 10 rules that have the highest lift value and the confidence value is 1. From table 2 results, we can say that if the consumer buys 243983529, he/she may also buy a 248475563 with %100 confidence. By the way, during this study, I did all my developments on python-jupyter so you can see the all detailed results with nice and human readable format on jupyter html file that I provided before.

#	Rule	Support	LeftHandSide => RightHandSide	Confidence	Lift
1	frozenset({243983529,248475563})	0.000012	frozenset({243983529}) => frozenset({248475563})	1.0	84518,20
2	frozenset({243983529,248475564})	0.000012	frozenset({243983529}) => frozenset({248475564})	1.0	84518,20
3	frozenset({248475563,248475564})	0.000012	frozenset({248475563}) => frozenset({248475564})	1.0	84518,20
4	frozenset({243983529,248475563,248475564})	0.000012	frozenset({243983529;248475563}) => frozenset({248475564})	1.0	84518,20
5	frozenset({243983529,251346844})	0.000012	frozenset({243983529}) => frozenset({251346844})	1.0	70431,83
6	frozenset({243983529,252944451})	0.000012	frozenset({243983529}) => frozenset({252944451})	1.0	70431,83
7	frozenset({248687298,248378431})	0.000012	frozenset({248378431}) => frozenset({248687298})	1.0	70431,83
8	frozenset({248475563,251346844})	0.000012	frozenset({248475563}) => frozenset({251346844})	1.0	70431,83
9	frozenset({252944451,248475563})	0.000012	frozenset({248475563}) => frozenset({252944451})	1.0	70431,83
10	frozenset({251346844,248475564})	0.000012	frozenset({248475564}) => frozenset({251346844})	1.0	70431,83

Table 2: Top 10 rules that have the highest lift value

5.2 Results for Model No 2: Member Id to Product Id

I set min_support =0.00001, min_confidence=0.2, min_lift=3 and min_length= 2 and got satisfied the rules for model 1. The results in the table 3 makes sense to us because we have greater than min_lift=3 values. You can see all outputs in the appendix; you can see the first 10 rules that have the highest lift value. From table 3 results, we can say that if the consumer buys 243983529 and 248475563, he/she may also buy a 248475564 with %100 confidence. By the way, during this study, I did all my developments on python-jupyter so you can see the all detailed results with nice and human readable format on jupyter html file that I provided before.

#	Rule	Support	LeftHandSide => RightHandSide	Confidence	Lift
1	frozenset({243983529,248475563})	0.000012	frozenset({243983529}) => frozenset({248475563})	1.0	84518,20
2	frozenset({243983529,248475564})	0.000012	frozenset({243983529}) => frozenset({248475564})	1.0	84518,20
3	frozenset({248475563,248475564})	0.000012	frozenset({248475563}) => frozenset({248475564})	1.0	84518,20
4	frozenset({243983529,248475563,248475564})	0.000012	frozenset({243983529,248475563}) => frozenset({248475564})	1.0	84518,20
5	frozenset({243983529,251346844})	0.000012	frozenset({243983529}) => frozenset({251346844})	1.0	70431,83
6	frozenset({243983529,252944451})	0.000012	frozenset({243983529}) => frozenset({252944451})	1.0	70431,83
7	frozenset({248687298,248378431})	0.000012	frozenset({248378431}) => frozenset({248687298})	1.0	70431,83
8	frozenset({248475563,251346844})	0.000012	frozenset({248475563}) => frozenset({251346844})	1.0	70431,83
9	frozenset({252944451;248475563})	0.000012	frozenset({248475563}) => frozenset({252944451})	1.0	70431,83
10	frozenset({251346844,248475564})	0.000012	frozenset({248475564}) => frozenset({251346844})	1.0	70431,83

Table 3: Top 10 rules that have the highest lift value

6. CONCLUSION, LIMITATIONS AND FUTURE WORK

Apriori algorithm effectively generates highly informative frequent itemsets and association rules for GittiGidiyor. In this capstone project, we found the co-occurring items in consumer shopping baskets in the data set with the help of the association rule mining algorithm; apriori. Mining association rules from transactional data provided us with valuable information about co-occurrences and co-purchases of products. Such information can be used as a basis for decisions about marketing activity such as promotional support, inventory control and cross-sale campaigns.

Limitation of this study was the memory leak of python list function. I used that list function to get the Apriori results as a list. Without this operation, we cannot read the Apriori results because it returns the results as an object map. During object to list process, python list function is using 32GB memory for our data set. Because of that, I had to work on a workstation that has 8-core i5, 64GB ram.

For future work, we can focus on parallelization of Apriori algorithm on the GPU. The algorithm in GPU shows a speedup over existing Apriori algorithm. GPU parallel computing will provide compelling benefits for data mining applications. When the size of dataset increases, speedup also increases. So we can get rid of our memory leak problem.

REFERENCES

1- Katrin Dippold, Harald Hruschka. (2010). Variable Selection for Market Basket Analysis. University of Regensburg Working Papers in Business, Economics and Management Information Systems.
2- Verma Dipti, Nashine Rakesh. (2012). Data Mining: Next Generation Challenges and Future Directions - International Journal of Modeling and Optimization. pp 603
3- R. Agrawal, T. Imielinski, and A. Swami. (1993). Mining Association Rules Between Sets of Items in Large Databases. Proc. of the ACM SIGMOD Conference on Management of Data. Washington D.C.
4- M.J.Zaki, M.Ogihara, S. Parthasarathy. (1996). Parallel Data Mining for Association Rules on Shared Memory Multi-processors. New York: University of Rochester.
5- Eu-Hong (Sam) Han, George Karypis, Vipin Kumar. (1999). Scalable Parallel Data Mining for Association Rules. IEEE Transactions on Knowledge and Data Engineering, vol.20.
6- Andreas Mild, Thomas Reutterer. (2003). An improved collaborative filtering approach for predicting cross-category purchases based on binary market basket data. Journal of Retailing and Consumer Services vol.10, 123-133.
7- K. Spandana, D.Sirisha, S. Shahida. (2016). Parallelizing Apriori Algorithm on GPU. Journal of Computer Applications vol.155
8- http://adataanalyst.com/machine-learning/apriori-algorithm-python-3-0/
9- https://www-users.cs.umn.edu/~kumar/dmbook/ch6.pdf
10- https://pypi.python.org/pypi/apyori/1.1.1
11- https://sivaanalytics.wordpress.com/tag/market-basket-analysis/
12- https://discourse.snowplowanalytics.com/t/market-basket-analysis-identifying-products-and-content-that-go-well-together/1132
13- https://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/retail-big-data-analytics-solution-blueprint.pdf

APPENDIX A: RESULTS OF MODEL NO 1

#	Rule	Support	LeftHandSide => RightHandSide	Confidence	Lift
1	frozenset({243983529 ;248475563})	0.000012	frozenset({243983529}) => frozenset({248475563})	1.0	84518.2
2	frozenset({243983529 ;248475564})	0.000012	frozenset({243983529}) => frozenset({248475564})	1.0	84518.2
3	frozenset({243983529 ;248683252})	0.000012	frozenset({243983529}) => frozenset({248683252})	1.0	60.370.143
4	frozenset({243983529 ;251346844})	0.000012	frozenset({243983529}) => frozenset({251346844})	1.0	70.431.833
5	frozenset({243983529 ;252944451})	0.000012	frozenset({243983529}) => frozenset({252944451})	1.0	70.431.833
6	frozenset({248683252 ;248378431})	0.000012	frozenset({248378431}) => frozenset({248683252})	1.0	60.370.143
7	frozenset({248687298 ;248378431})	0.000012	frozenset({248378431}) => frozenset({248687298})	1.0	70.431.833
8	frozenset({248475563 ;248475564})	0.000012	frozenset({248475563}) => frozenset({248475564})	1.0	84518.2
9	frozenset({248475563 ;248683252})	0.000012	frozenset({248475563}) => frozenset({248683252})	1.0	60.370.143
10	frozenset({248475563 ;251346844})	0.000012	frozenset({248475563}) => frozenset({251346844})	1.0	70.431.833
11	frozenset({252944451 ;248475563})	0.000012	frozenset({248475563}) => frozenset({252944451})	1.0	70.431.833
12	frozenset({248683252 ;248475564})	0.000012	frozenset({248475564}) => frozenset({248683252})	1.0	60.370.143
13	frozenset({251346844 ;248475564})	0.000012	frozenset({248475564}) => frozenset({251346844})	1.0	70.431.833
14	frozenset({252944451 ;248475564})	0.000012	frozenset({248475564}) => frozenset({252944451})	1.0	70.431.833
15	frozenset({256099537 ;256099250})	0.000026	frozenset({256099250}) => frozenset({256099537})	1.0	38.417.364
16	frozenset({256099250 ;256099918})	0.000026	frozenset({256099250}) => frozenset({256099918})	1.0	38.417.364
17	frozenset({256099250 ;256100125})	0.000026	frozenset({256099250}) => frozenset({256100125})	1.0	35.215.917
18	frozenset({256099250 ;256100381})	0.000026	frozenset({256099250}) => frozenset({256100381})	1.0	32507.0
19	frozenset({256099250 ;256101039})	0.000026	frozenset({256099250}) => frozenset({256101039})	1.0	35.215.917
20	frozenset({256101896 ;256099250})	0.000026	frozenset({256099250}) => frozenset({256101896})	1.0	35.215.917
21	frozenset({256099537 ;256099918})	0.000026	frozenset({256099537}) => frozenset({256099918})	1.0	38.417.364
22	frozenset({256099537 ;256100125})	0.000026	frozenset({256099537}) => frozenset({256100125})	1.0	35.215.917
23	frozenset({256099537 ;256100381})	0.000026	frozenset({256099537}) => frozenset({256100381})	1.0	32507.0
24	frozenset({256099537 ;256101039})	0.000026	frozenset({256099537}) => frozenset({256101039})	1.0	35.215.917
25	frozenset({256101896 ;256099537})	0.000026	frozenset({256099537}) => frozenset({256101896})	1.0	35.215.917
26	frozenset({256100125 ;256099918})	0.000026	frozenset({256099918}) => frozenset({256100125})	1.0	35.215.917

27	frozenset({256100381;256099918})	0.000026	frozenset({256099918}) => frozenset({256100381})	1.0	32507.0
28	frozenset({256099918;256101039})	0.000026	frozenset({256099918}) => frozenset({256101039})	1.0	35.215.917
29	frozenset({256101896;256099918})	0.000026	frozenset({256099918}) => frozenset({256101896})	1.0	35.215.917
30	frozenset({258567979;256908963})	0.000012	frozenset({256908963}) => frozenset({258567979})	1.0	16903.64
31	frozenset({266842569;266842564})	0.000017	frozenset({266842564}) => frozenset({266842569})	1.0	38.417.364
32	frozenset({266842564;266842573})	0.000017	frozenset({266842564}) => frozenset({266842573})	1.0	38.417.364
33	frozenset({266842579;266842564})	0.000017	frozenset({266842564}) => frozenset({266842579})	1.0	35.215.917
34	frozenset({266842569;266842573})	0.000026	frozenset({266842569}) => frozenset({266842573})	1.0	38.417.364
35	frozenset({266842569;266842579})	0.000026	frozenset({266842569}) => frozenset({266842579})	1.0	35.215.917
36	frozenset({266842579;266842573})	0.000026	frozenset({266842573}) => frozenset({266842579})	1.0	35.215.917
37	frozenset({266842577;266842579})	0.000024	frozenset({266842577}) => frozenset({266842579})	1.0	35.215.917
38	frozenset({243983529;248475563;248475564})	0.000012	frozenset({243983529;248475563}) => frozenset({248475564})	1.0	84518.2
39	frozenset({243983529;248475563;248683252})	0.000012	frozenset({243983529;248475563}) => frozenset({248683252})	1.0	60.370.143
40	frozenset({243983529;248475563;251346844})	0.000012	frozenset({243983529;248475563}) => frozenset({251346844})	1.0	70.431.833
41	frozenset({243983529;252944451;248475563})	0.000012	frozenset({243983529;248475563}) => frozenset({252944451})	1.0	70.431.833
42	frozenset({243983529;248683252;248475564})	0.000012	frozenset({243983529;248475564}) => frozenset({248683252})	1.0	60.370.143
43	frozenset({243983529;248475564;251346844})	0.000012	frozenset({243983529;248475564}) => frozenset({251346844})	1.0	70.431.833
44	frozenset({243983529;252944451;248475564})	0.000012	frozenset({243983529;248475564}) => frozenset({252944451})	1.0	70.431.833
45	frozenset({243983529;248683252;251346844})	0.000012	frozenset({243983529;248683252}) => frozenset({251346844})	1.0	70.431.833
46	frozenset({243983529;252944451;248683252})	0.000012	frozenset({243983529;248683252}) => frozenset({252944451})	1.0	70.431.833
47	frozenset({243983529;252944451;251346844})	0.000012	frozenset({243983529;251346844}) => frozenset({252944451})	1.0	70.431.833
48	frozenset({248687298;248683252;248378431})	0.000012	frozenset({248683252;248378431}) => frozenset({248687298})	1.0	70.431.833
49	frozenset({248475563;248475564;248683252})	0.000012	frozenset({248475563;248475564}) => frozenset({248683252})	1.0	60.370.143
50	frozenset({251346844	0.000012	frozenset({248475563;248475564})	1.0	70.431.833

	;248475563;248475564})		=> frozenset({251346844})		
51	frozenset({252944451;248475563;248475564})	0.000012	frozenset({248475563;248475564}) => frozenset({252944451})	1.0	70.431.833
52	frozenset({251346844;248475563;248683252})	0.000012	frozenset({248475563;248683252}) => frozenset({251346844})	1.0	70.431.833
53	frozenset({252944451;248475563;248683252})	0.000012	frozenset({248475563;248683252}) => frozenset({252944451})	1.0	70.431.833
54	frozenset({252944451;248475563;251346844})	0.000012	frozenset({248475563;251346844}) => frozenset({252944451})	1.0	70.431.833
55	frozenset({251346844;248683252;248475564})	0.000012	frozenset({248683252;248475564}) => frozenset({251346844})	1.0	70.431.833
56	frozenset({252944451;248683252;248475564})	0.000012	frozenset({248683252;248475564}) => frozenset({252944451})	1.0	70.431.833
57	frozenset({251346844;252944451;248475564})	0.000012	frozenset({251346844;248475564}) => frozenset({252944451})	1.0	70.431.833
58	frozenset({256099537;256099250;256099918})	0.000026	frozenset({256099537;256099250}) => frozenset({256099918})	1.0	38.417.364
59	frozenset({256099537;256099250;256100125})	0.000026	frozenset({256099537;256099250}) => frozenset({256100125})	1.0	35.215.917
60	frozenset({256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250}) => frozenset({256100381})	1.0	32507.0
61	frozenset({256099537;256099250;256101039})	0.000026	frozenset({256099537;256099250}) => frozenset({256101039})	1.0	35.215.917
62	frozenset({256101896;256099537;256099250})	0.000026	frozenset({256099537;256099250}) => frozenset({256101896})	1.0	35.215.917
63	frozenset({256099250;256100125;256099918})	0.000026	frozenset({256099250;256099918}) => frozenset({256100125})	1.0	35.215.917
64	frozenset({256099250;256100381;256099918})	0.000026	frozenset({256099250;256099918}) => frozenset({256100381})	1.0	32507.0
65	frozenset({256099250;256099918;256101039})	0.000026	frozenset({256099250;256099918}) => frozenset({256101039})	1.0	35.215.917
66	frozenset({256101896;256099250;256099918})	0.000026	frozenset({256099250;256099918}) => frozenset({256101896})	1.0	35.215.917
67	frozenset({256100381;256099250;256100125})	0.000026	frozenset({256099250;256100125}) => frozenset({256100381})	1.0	32507.0
68	frozenset({256099250;256100125;256101039})	0.000026	frozenset({256099250;256100125}) => frozenset({256101039})	1.0	35.215.917
69	frozenset({256101896;256099250;256100125})	0.000026	frozenset({256099250;256100125}) => frozenset({256101896})	1.0	35.215.917

70	frozenset({256099250;256100381;256101039})	0.000026	frozenset({256099250;256100381}) => frozenset({256101039})	1.0	35.215.917
71	frozenset({256101896;256099250;256100381})	0.000026	frozenset({256099250;256100381}) => frozenset({256101896})	1.0	35.215.917
72	frozenset({256101896;256099250;256101039})	0.000026	frozenset({256099250;256101039}) => frozenset({256101896})	1.0	35.215.917
73	frozenset({256099537;256100125;256099918})	0.000026	frozenset({256099537;256099918}) => frozenset({256100125})	1.0	35.215.917
74	frozenset({256099537;256100381;256099918})	0.000026	frozenset({256099537;256099918}) => frozenset({256100381})	1.0	32507.0
75	frozenset({256099537;256099918;256101039})	0.000026	frozenset({256099537;256099918}) => frozenset({256101039})	1.0	35.215.917
76	frozenset({256101896;256099537;256099918})	0.000026	frozenset({256099537;256099918}) => frozenset({256101896})	1.0	35.215.917
77	frozenset({256099537;256100381;256100125})	0.000026	frozenset({256099537;256100125}) => frozenset({256100381})	1.0	32507.0
78	frozenset({256099537;256100125;256101039})	0.000026	frozenset({256099537;256100125}) => frozenset({256101039})	1.0	35.215.917
79	frozenset({256101896;256099537;256100125})	0.000026	frozenset({256099537;256100125}) => frozenset({256101896})	1.0	35.215.917
80	frozenset({256099537;256100381;256101039})	0.000026	frozenset({256099537;256100381}) => frozenset({256101039})	1.0	35.215.917
81	frozenset({256101896;256099537;256100381})	0.000026	frozenset({256099537;256100381}) => frozenset({256101896})	1.0	35.215.917
82	frozenset({256101896;256099537;256101039})	0.000026	frozenset({256099537;256101039}) => frozenset({256101896})	1.0	35.215.917
83	frozenset({256100381;256100125;256099918})	0.000026	frozenset({256100125;256099918}) => frozenset({256100381})	1.0	32507.0
84	frozenset({256100125;256099918;256101039})	0.000026	frozenset({256100125;256099918}) => frozenset({256101039})	1.0	35.215.917
85	frozenset({256101896;256100125;256099918})	0.000026	frozenset({256100125;256099918}) => frozenset({256101896})	1.0	35.215.917
86	frozenset({256100381;256099918;256101039})	0.000026	frozenset({256100381;256099918}) => frozenset({256101039})	1.0	35.215.917
87	frozenset({256101896;256100381;256099918})	0.000026	frozenset({256100381;256099918}) => frozenset({256101896})	1.0	35.215.917
88	frozenset({256101896;256099918;256101039})	0.000026	frozenset({256099918;256101039}) => frozenset({256101896})	1.0	35.215.917
89	frozenset({256100381;256100125;256101039})	0.000026	frozenset({256100381;256100125}) => frozenset({256101039})	1.0	35.215.917

90	frozenset({256101896;256100381;256100125})	0.000026	frozenset({256100381;256100125}) => frozenset({256101896})	1.0	35.215.917
91	frozenset({256101896;256100125;256101039})	0.000026	frozenset({256100125;256101039}) => frozenset({256101896})	1.0	35.215.917
92	frozenset({256101896;256100381;256101039})	0.000026	frozenset({256100381;256101039}) => frozenset({256101896})	1.0	35.215.917
93	frozenset({266842569;266842564;266842573})	0.000017	frozenset({266842569;266842564}) => frozenset({266842573})	1.0	38.417.364
94	frozenset({266842569;266842579;266842564})	0.000017	frozenset({266842569;266842564}) => frozenset({266842579})	1.0	35.215.917
95	frozenset({266842579;266842564;266842573})	0.000017	frozenset({266842564;266842573}) => frozenset({266842579})	1.0	35.215.917
96	frozenset({266842577;266842579;266842564})	0.000014	frozenset({266842577;266842564}) => frozenset({266842579})	1.0	35.215.917
97	frozenset({266842569;266842579;266842573})	0.000026	frozenset({266842569;266842573}) => frozenset({266842579})	1.0	35.215.917
98	frozenset({266842569;266842579;266842577})	0.000024	frozenset({266842569;266842577}) => frozenset({266842579})	1.0	35.215.917
99	frozenset({266842577;266842579;266842573})	0.000024	frozenset({266842577;266842573}) => frozenset({266842579})	1.0	35.215.917
100	frozenset({243983529;248475563;248475564;248683252})	0.000012	frozenset({243983529;248475563;248475564}) => frozenset({248683252})	1.0	60.370.143
101	frozenset({243983529;248475563;248475564;251346844})	0.000012	frozenset({243983529;248475563;248475564}) => frozenset({251346844})	1.0	70.431.833
102	frozenset({243983529;252944451;248475563;248475564})	0.000012	frozenset({243983529;248475563;248475564}) => frozenset({252944451})	1.0	70.431.833
103	frozenset({243983529;248475563;248683252;251346844})	0.000012	frozenset({243983529;248475563;248683252}) => frozenset({251346844})	1.0	70.431.833
104	frozenset({243983529;252944451;248475563;248683252})	0.000012	frozenset({243983529;248475563;248683252}) => frozenset({252944451})	1.0	70.431.833
105	frozenset({243983529;252944451;248475563;251346844})	0.000012	frozenset({243983529;248475563;251346844}) => frozenset({252944451})	1.0	70.431.833
106	frozenset({243983529;248683252;248475564;251346844})	0.000012	frozenset({243983529;248683252;248475564}) => frozenset({251346844})	1.0	70.431.833
107	frozenset({243983529;252944451;248683252;248475564})	0.000012	frozenset({243983529;248683252;248475564}) => frozenset({252944451})	1.0	70.431.833
108	frozenset({243983529;252944451;248475564;251346844})	0.000012	frozenset({243983529;248475564;251346844}) => frozenset({252944451})	1.0	70.431.833
109	frozenset({243983529;252944451;248683252;251346844})	0.000012	frozenset({243983529;248683252;251346844}) => frozenset({252944451})	1.0	70.431.833

110	frozenset({251346844;248475563;248475564;248683252})	0.000012	frozenset({248475563;248475564;248683252}) => frozenset({251346844})	1.0	70.431.833
111	frozenset({252944451;248475563;248475564;248683252})	0.000012	frozenset({248475563;248475564;248683252}) => frozenset({252944451})	1.0	70.431.833
112	frozenset({251346844;252944451;248475563;248475564})	0.000012	frozenset({251346844;248475563;248475564}) => frozenset({252944451})	1.0	70.431.833
113	frozenset({251346844;252944451;248475563;248683252})	0.000012	frozenset({251346844;248475563;248683252}) => frozenset({252944451})	1.0	70.431.833
114	frozenset({251346844;252944451;248683252;248475564})	0.000012	frozenset({251346844;248683252;248475564}) => frozenset({252944451})	1.0	70.431.833
115	frozenset({256099537;256099250;256100125;256099918})	0.000026	frozenset({256099537;256099250;256099918}) => frozenset({256100125})	1.0	35.215.917
116	frozenset({256099537;256099250;256100381;256099918})	0.000026	frozenset({256099537;256099250;256099918}) => frozenset({256100381})	1.0	32507.0
117	frozenset({256099537;256099250;256099918;256101039})	0.000026	frozenset({256099537;256099250;256099918}) => frozenset({256101039})	1.0	35.215.917
118	frozenset({256101896;256099537;256099250;256099918})	0.000026	frozenset({256099537;256099250;256099918}) => frozenset({256101896})	1.0	35.215.917
119	frozenset({256099537;256099250;256100381;256100125})	0.000026	frozenset({256099537;256099250;256100125}) => frozenset({256100381})	1.0	32507.0
120	frozenset({256099537;256099250;256100125;256101039})	0.000026	frozenset({256099537;256099250;256100125}) => frozenset({256101039})	1.0	35.215.917
121	frozenset({256101896;256099537;256099250;256100125})	0.000026	frozenset({256099537;256099250;256100125}) => frozenset({256101896})	1.0	35.215.917
122	frozenset({256099537;256099250;256100381;256101039})	0.000026	frozenset({256099537;256099250;256100381}) => frozenset({256101039})	1.0	35.215.917
123	frozenset({256101896;256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250;256100381}) => frozenset({256101896})	1.0	35.215.917
124	frozenset({256101896;256099537;256099250;256101039})	0.000026	frozenset({256099537;256099250;256101039}) => frozenset({256101896})	1.0	35.215.917
125	frozenset({256100381;256099250;256100125;256099918})	0.000026	frozenset({256099250;256100125;256099918}) => frozenset({256100381})	1.0	32507.0
126	frozenset({256099250;256100125;256099918;256101039})	0.000026	frozenset({256099250;256100125;256099918}) => frozenset({256101039})	1.0	35.215.917
127	frozenset({256101896;256099250;256100125;256099918})	0.000026	frozenset({256099250;256100125;256099918}) => frozenset({256101896})	1.0	35.215.917
128	frozenset({256099250;256100381;256099918;256101039})	0.000026	frozenset({256099250;256100381;256099918}) => frozenset({256101039})	1.0	35.215.917
129	frozenset({256101896;256099250;256100381;256099918})	0.000026	frozenset({256099250;256100381;256099918}) => frozenset({256101896})	1.0	35.215.917

130	frozenset({256101896;256099250;256099918;256101039})	0.000026	frozenset({256099250;256099918;256101039}) => frozenset({256101896})	1.0	35.215.917
131	frozenset({256100381;256099250;256100125;256101039})	0.000026	frozenset({256100381;256099250;256100125}) => frozenset({256101039})	1.0	35.215.917
132	frozenset({256101896;256100381;256099250;256100125})	0.000026	frozenset({256100381;256099250;256100125}) => frozenset({256101896})	1.0	35.215.917
133	frozenset({256101896;256099250;256100125;256101039})	0.000026	frozenset({256099250;256100125;256101039}) => frozenset({256101896})	1.0	35.215.917
134	frozenset({256101896;256099250;256100381;256101039})	0.000026	frozenset({256099250;256100381;256101039}) => frozenset({256101896})	1.0	35.215.917
135	frozenset({256099537;256100381;256100125;256099918})	0.000026	frozenset({256099537;256100125;256099918}) => frozenset({256100381})	1.0	32507.0
136	frozenset({256099537;256100125;256099918;256101039})	0.000026	frozenset({256099537;256100125;256099918}) => frozenset({256101039})	1.0	35.215.917
137	frozenset({256101896;256099537;256100125;256099918})	0.000026	frozenset({256099537;256100125;256099918}) => frozenset({256101896})	1.0	35.215.917
138	frozenset({256099537;256100381;256099918;256101039})	0.000026	frozenset({256099537;256100381;256099918}) => frozenset({256101039})	1.0	35.215.917
139	frozenset({256101896;256099537;256100381;256099918})	0.000026	frozenset({256099537;256100381;256099918}) => frozenset({256101896})	1.0	35.215.917
140	frozenset({256101896;256099537;256099918;256101039})	0.000026	frozenset({256099537;256099918;256101039}) => frozenset({256101896})	1.0	35.215.917
141	frozenset({256099537;256100381;256100125;256101039})	0.000026	frozenset({256099537;256100381;256100125}) => frozenset({256101039})	1.0	35.215.917
142	frozenset({256101896;256099537;256100381;256100125})	0.000026	frozenset({256099537;256100381;256100125}) => frozenset({256101896})	1.0	35.215.917
143	frozenset({256101896;256099537;256100125;256101039})	0.000026	frozenset({256099537;256100125;256101039}) => frozenset({256101896})	1.0	35.215.917
144	frozenset({256101896;256099537;256100381;256101039})	0.000026	frozenset({256099537;256100381;256101039}) => frozenset({256101896})	1.0	35.215.917
145	frozenset({256100381;256100125;256099918;256101039})	0.000026	frozenset({256100381;256100125;256099918}) => frozenset({256101039})	1.0	35.215.917
146	frozenset({256101896;256100381;256100125;256099918})	0.000026	frozenset({256100381;256100125;256099918}) => frozenset({256101896})	1.0	35.215.917
147	frozenset({256101896;256100125;256099918;256101039})	0.000026	frozenset({256100125;256099918;256101039}) => frozenset({256101896})	1.0	35.215.917
148	frozenset({256101896;256100381;256099918;256101039})	0.000026	frozenset({256100381;256099918;256101039}) => frozenset({256101896})	1.0	35.215.917
149	frozenset({256101896;256100381;256100125;256101039})	0.000026	frozenset({256100381;256100125;256101039}) => frozenset({256101896})	1.0	35.215.917

150	frozenset({266842569;266842579;266842564;266842573})	0.000017	frozenset({266842569;266842564;266842573}) => frozenset({266842579})	1.0	35.215.917
151	frozenset({266842569;266842579;266842564;266842577})	0.000014	frozenset({266842569;266842564;266842577}) => frozenset({266842579})	1.0	35.215.917
152	frozenset({266842577;266842579;266842564;266842573})	0.000014	frozenset({266842577;266842564;266842573}) => frozenset({266842579})	1.0	35.215.917
153	frozenset({266842569;266842579;266842577;266842573})	0.000024	frozenset({266842569;266842577;266842573}) => frozenset({266842579})	1.0	35.215.917
154	frozenset({243983529;248475563;248475564;248683252;251346844})	0.000012	frozenset({243983529;248475563;248475564;248683252}) => frozenset({251346844})	1.0	70.431.833
155	frozenset({252944451;243983529;248475563;248475564;248683252})	0.000012	frozenset({243983529;248475563;248475564;248683252}) => frozenset({252944451})	1.0	70.431.833
156	frozenset({252944451;243983529;248475563;248475564;251346844})	0.000012	frozenset({243983529;248475563;248475564;251346844}) => frozenset({252944451})	1.0	70.431.833
157	frozenset({252944451;243983529;248475563;248683252;251346844})	0.000012	frozenset({243983529;248475563;248683252;251346844}) => frozenset({252944451})	1.0	70.431.833
158	frozenset({252944451;243983529;248475564;248683252;251346844})	0.000012	frozenset({243983529;248683252;248475564;251346844}) => frozenset({252944451})	1.0	70.431.833
159	frozenset({252944451;248475563;248475564;248683252;251346844})	0.000012	frozenset({251346844;248475563;248475564;248683252}) => frozenset({252944451})	1.0	70.431.833
160	frozenset({256100125;256099918;256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250;256100125;256099918}) => frozenset({256100381})	1.0	32507.0
161	frozenset({256099918;256101039;256099537;256099250;256100125})	0.000026	frozenset({256099537;256099250;256100125;256099918}) => frozenset({256101039})	1.0	35.215.917
162	frozenset({256101896;256099918;256099537;256099250;256100125})	0.000026	frozenset({256099537;256099250;256100125;256099918}) => frozenset({256101896})	1.0	35.215.917
163	frozenset({256099918;256101039;256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250;256100381;256099918}) => frozenset({256101039})	1.0	35.215.917
164	frozenset({256101896;256099918;256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250;256100381;256099918}) => frozenset({256101896})	1.0	35.215.917
165	frozenset({256101896;256099918;256101039;256099537;256099250})	0.000026	frozenset({256099537;256099250;256099918;256101039}) => frozenset({256101896})	1.0	35.215.917

166	frozenset({256100125;256101039;256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250;256100381;256100125}) => frozenset({256101039})	1.0	35.215.917
167	frozenset({256101896;256100125;256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250;256100381;256100125}) => frozenset({256101896})	1.0	35.215.917
168	frozenset({256101896;256101039;256099537;256099250;256100125})	0.000026	frozenset({256099537;256099250;256100125;256101039}) => frozenset({256101896})	1.0	35.215.917
169	frozenset({256101896;256101039;256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250;256100381;256101039}) => frozenset({256101896})	1.0	35.215.917
170	frozenset({256100125;256099918;256101039;256099250;256100381})	0.000026	frozenset({256100381;256099250;256100125;256099918}) => frozenset({256101039})	1.0	35.215.917
171	frozenset({256101896;256100125;256099918;256099250;256100381})	0.000026	frozenset({256100381;256099250;256100125;256099918}) => frozenset({256101896})	1.0	35.215.917
172	frozenset({256101896;256099918;256101039;256099250;256100125})	0.000026	frozenset({256099250;256100125;256099918;256101039}) => frozenset({256101896})	1.0	35.215.917
173	frozenset({256101896;256099918;256101039;256099250;256100381})	0.000026	frozenset({256099250;256100381;256099918;256101039}) => frozenset({256101896})	1.0	35.215.917
174	frozenset({256101896;256100125;256101039;256099250;256100381})	0.000026	frozenset({256100381;256099250;256100125;256101039}) => frozenset({256101896})	1.0	35.215.917
175	frozenset({256100125;256099918;256101039;256099537;256100381})	0.000026	frozenset({256099537;256100381;256100125;256099918}) => frozenset({256101039})	1.0	35.215.917
176	frozenset({256101896;256100125;256099918;256099537;256100381})	0.000026	frozenset({256099537;256100381;256100125;256099918}) => frozenset({256101896})	1.0	35.215.917
177	frozenset({256101896;256099918;256101039;256099537;256100125})	0.000026	frozenset({256099537;256100125;256099918;256101039}) => frozenset({256101896})	1.0	35.215.917
178	frozenset({256101896;256099918;256101039;256099537;256100381})	0.000026	frozenset({256099537;256100381;256099918;256101039}) => frozenset({256101896})	1.0	35.215.917
179	frozenset({256101896;256100125;256101039;256099537;256100381})	0.000026	frozenset({256099537;256100381;256100125;256101039}) => frozenset({256101896})	1.0	35.215.917
180	frozenset({256101896;256100125;256099918;256101039;256100381})	0.000026	frozenset({256100381;256100125;256099918;256101039}) => frozenset({256101896})	1.0	35.215.917

181	frozenset({266842564;266842569;266842573;266842577;266842579})	0.000014	frozenset({266842577;266842569;266842564;266842573}) => frozenset({266842579})	1.0	35.215.917
182	frozenset({252944451;243983529;248475563;248475564;248683252;251346844})	0.000012	frozenset({243983529;248475563;248475564;248683252;251346844}) => frozenset({252944451})	1.0	70.431.833
183	frozenset({256100125;256099918;256101039;256099537;256099250;256100381})	0.000026	frozenset({256100125;256099918;256099537;256099250;256100381}) => frozenset({256101039})	1.0	35.215.917
184	frozenset({256101896;256100125;256099918;256099537;256099250;256100381})	0.000026	frozenset({256100125;256099918;256099537;256099250;256100381}) => frozenset({256101896})	1.0	35.215.917
185	frozenset({256101896;256099918;256101039;256099537;256099250;256100125})	0.000026	frozenset({256099918;256101039;256099537;256099250;256100125}) => frozenset({256101896})	1.0	35.215.917
186	frozenset({256101896;256099918;256101039;256099537;256099250;256100381})	0.000026	frozenset({256099918;256101039;256099537;256099250;256100381}) => frozenset({256101896})	1.0	35.215.917
187	frozenset({256101896;256100125;256101039;256099537;256099250;256100381})	0.000026	frozenset({256100125;256101039;256099537;256099250;256100381}) => frozenset({256101896})	1.0	35.215.917
188	frozenset({256101896;256100125;256099918;256101039;256099250;256100381})	0.000026	frozenset({256100125;256099918;256101039;256099250;256100381}) => frozenset({256101896})	1.0	35.215.917
189	frozenset({256101896;256100125;256099918;256101039;256099537;256100381})	0.000026	frozenset({256100125;256099918;256101039;256099537;256100381}) => frozenset({256101896})	1.0	35.215.917
190	frozenset({256101896;256100125;256099918;256101039;256099537;256099250;256100381})	0.000026	frozenset({256100125;256099918;256101039;256099537;256099250;256100381}) => frozenset({256101896})	1.0	35.215.917
191	frozenset({256100381;256100125})	0.000026	frozenset({256100125}) => frozenset({256100381})	0.917	29.798.083
192	frozenset({256100125;256101039})	0.000026	frozenset({256100125}) => frozenset({256101039})	0.917	32.281.257
193	frozenset({256101896;256100125})	0.000026	frozenset({256100125}) => frozenset({256101896})	0.917	32.281.257
194	frozenset({256101896;256101039})	0.000026	frozenset({256101039}) => frozenset({256101896})	0.917	32.281.257
195	frozenset({266842569;266842577})	0.000024	frozenset({266842569}) => frozenset({266842577})	0.909	38.417.364
196	frozenset({266842577;266842573})	0.000024	frozenset({266842573}) => frozenset({266842577})	0.909	38.417.364
197	frozenset({266842569;266842577;266842573})	0.000024	frozenset({266842569;266842573}) => frozenset({266842577})	0.909	38.417.364
198	frozenset({255732329;256050591})	0.000019	frozenset({256050591}) => frozenset({255732329})	0.889	1.020.751
199	frozenset({228993720;248669138})	0.000017	frozenset({228993720}) => frozenset({248669138})	0.875	30.813.927

200	frozenset({251346844;248683252})	0.000014	frozenset({248683252}) => frozenset({251346844})	0.857	60.370.143
201	frozenset({252944451;248683252})	0.000014	frozenset({248683252}) => frozenset({252944451})	0.857	60.370.143
202	frozenset({266842577;266842564})	0.000014	frozenset({266842564}) => frozenset({266842577})	0.857	36.222.086
203	frozenset({266842569;266842564;266842577})	0.000014	frozenset({266842569;266842564}) => frozenset({266842577})	0.857	36.222.086
204	frozenset({266842577;266842564;266842573})	0.000014	frozenset({266842564;266842573}) => frozenset({266842577})	0.857	36.222.086
205	frozenset({266842577;266842569;266842564;266842573})	0.000014	frozenset({266842569;266842564;266842573}) => frozenset({266842577})	0.857	36.222.086
206	frozenset({256100381;256101039})	0.000026	frozenset({256100381}) => frozenset({256101039})	0.846	29.798.083
207	frozenset({256101896;256100381})	0.000026	frozenset({256100381}) => frozenset({256101896})	0.846	29.798.083
208	frozenset({236992197;239797102})	0.000012	frozenset({236992197}) => frozenset({239797102})	0.833	70.431.833
209	frozenset({252944451;251346844})	0.000012	frozenset({251346844}) => frozenset({252944451})	0.833	58.693.194
210	frozenset({256901074;258567979})	0.000012	frozenset({256901074}) => frozenset({258567979})	0.833	14.086.367
211	frozenset({251346844;252944451;248683252})	0.000012	frozenset({251346844;248683252}) => frozenset({252944451})	0.833	58.693.194
212	frozenset({255922043;255922044;255935181})	0.000024	frozenset({255922043;255922044}) => frozenset({255935181})	0.769	2.731.681
213	frozenset({252703306;246620796})	0.000026	frozenset({246620796}) => frozenset({252703306})	0.733	748.551
214	frozenset({248687298;248683252})	0.000012	frozenset({248683252}) => frozenset({248687298})	0.714	50.308.452
215	frozenset({251511636;260320446})	0.000019	frozenset({260320446}) => frozenset({251511636})	0.667	327.971
216	frozenset({246648362;250654478})	0.000047	frozenset({246648362}) => frozenset({250654478})	0.667	197.288
217	frozenset({265964602;259694237})	0.000031	frozenset({259694237}) => frozenset({265964602})	0.65	340.799
218	frozenset({255732329;260704449})	0.000012	frozenset({260704449}) => frozenset({255732329})	0.625	717.716
219	frozenset({256811658;258567979;256901791})	0.000012	frozenset({256811658;256901791}) => frozenset({258567979})	0.625	10.564.775
220	frozenset({251562499;255734364})	0.000012	frozenset({251562499}) => frozenset({255734364})	0.556	790.481
221	frozenset({255732329;251946684})	0.000012	frozenset({251946684}) => frozenset({255732329})	0.556	637.97
222	frozenset({252528723;250654478})	0.000026	frozenset({252528723}) => frozenset({250654478})	0.55	162.763
223	frozenset({248438052;250114941})	0.000014	frozenset({250114941}) => frozenset({248438052})	0.545	441.579
224	frozenset({258567979;256914095})	0.000014	frozenset({256914095}) => frozenset({258567979})	0.545	9.220.167
225	frozenset({265363377;265363094})	0.000014	frozenset({265363094}) => frozenset({265363377})	0.545	38.417.364

226	frozenset({268827043 ;268118799})	0.000014	frozenset({268827043}) => frozenset({268118799})	0.545	527.47
227	frozenset({252528723 ;252528854;2506544 78})	0.000014	frozenset({252528723;250654478}) => frozenset({252528854})	0.545	8.537.192
228	frozenset({261632776 ;261632873})	0.000017	frozenset({261632776}) => frozenset({261632873})	0.538	12.641.611
229	frozenset({252528854 ;250654478})	0.000033	frozenset({252528854}) => frozenset({250654478})	0.519	153.446
230	frozenset({249582384 ;251562499})	0.000012	frozenset({249582384}) => frozenset({251562499})	0.5	23.477.278
231	frozenset({249582384 ;255734364})	0.000012	frozenset({249582384}) => frozenset({255734364})	0.5	711.433
232	frozenset({254712410 ;250654478})	0.000033	frozenset({254712410}) => frozenset({250654478})	0.5	147.966
233	frozenset({255732329 ;259348732})	0.000014	frozenset({259348732}) => frozenset({255732329})	0.5	574.173
234	frozenset({258424107 ;260333814})	0.000024	frozenset({260333814}) => frozenset({258424107})	0.5	785.485
235	frozenset({265363569 ;265363506})	0.000024	frozenset({265363506}) => frozenset({265363569})	0.5	14.086.367
236	frozenset({268301345 ;270093958})	0.000085	frozenset({268301345}) => frozenset({270093958})	0.474	150.281
237	frozenset({255732329 ;258377850})	0.000017	frozenset({258377850}) => frozenset({255732329})	0.467	535.894
238	frozenset({273309576 ;270283868})	0.000033	frozenset({270283868}) => frozenset({273309576})	0.467	342.377
239	frozenset({256914095 ;256901791})	0.000012	frozenset({256914095}) => frozenset({256901791})	0.455	3.369.944
240	frozenset({258569352 ;258424107})	0.000012	frozenset({258569352}) => frozenset({258424107})	0.455	714.077
241	frozenset({258750424 ;259918909})	0.000012	frozenset({258750424}) => frozenset({259918909})	0.455	1.402.094
242	frozenset({252528723 ;252528854})	0.000021	frozenset({252528723}) => frozenset({252528854})	0.45	7.043.183
243	frozenset({268325338 ;269864739})	0.000019	frozenset({268325338}) => frozenset({269864739})	0.444	3.078.987
244	frozenset({250094344 ;250654478})	0.000014	frozenset({250094344}) => frozenset({250654478})	0.429	126.828
245	frozenset({256811658 ;256901791})	0.000019	frozenset({256811658}) => frozenset({256901791})	0.421	3.121.633
246	frozenset({270934433 ;269376483})	0.000012	frozenset({270934433}) => frozenset({269376483})	0.417	1.189.727
247	frozenset({274995841 ;275491877})	0.000012	frozenset({274995841}) => frozenset({275491877})	0.417	1.333.936
248	frozenset({259603152 ;255734364})	0.000031	frozenset({259603152}) => frozenset({255734364})	0.406	578.039
249	frozenset({242452616 ;242452613})	0.000017	frozenset({242452613}) => frozenset({242452616})	0.389	3.100.773
250	frozenset({260508220 ;260508230})	0.000019	frozenset({260508220}) => frozenset({260508230})	0.381	10061.69
251	frozenset({265020401 ;264770118})	0.000019	frozenset({265020401}) => frozenset({264770118})	0.381	336.09
252	frozenset({245721171 ;245299869})	0.000059	frozenset({245299869}) => frozenset({245721171})	0.357	1.886.567
253	frozenset({270734826 ;269366242})	0.000012	frozenset({270734826}) => frozenset({269366242})	0.357	4.573.496

254	frozenset({270093958 ;269903495})	0.000012	frozenset({269903495}) => frozenset({270093958})	0.357	113.307
255	frozenset({255734364 ;262796583})	0.000040	frozenset({262796583}) => frozenset({255734364})	0.347	493.647
256	frozenset({273579892 ;271537559})	0.000028	frozenset({273579892}) => frozenset({271537559})	0.343	499.615
257	frozenset({256723761 ;256901791})	0.000012	frozenset({256723761}) => frozenset({256901791})	0.333	2.471.292
258	frozenset({256723761 ;258568010})	0.000012	frozenset({256723761}) => frozenset({258568010})	0.333	10.835.667
259	frozenset({256723761 ;259037031})	0.000012	frozenset({256723761}) => frozenset({259037031})	0.333	7.825.759
260	frozenset({258338519 ;256922831})	0.000012	frozenset({258338519}) => frozenset({256922831})	0.333	378.666
261	frozenset({268779328 ;268839792})	0.000014	frozenset({268839792}) => frozenset({268779328})	0.333	4.024.676
262	frozenset({273309576 ;273914888})	0.000014	frozenset({273914888}) => frozenset({273309576})	0.333	244.555
263	frozenset({262613127 ;262613039})	0.000021	frozenset({262613039}) => frozenset({262613127})	0.321	4.244.776
264	frozenset({269542586 ;266531818})	0.000017	frozenset({269542586}) => frozenset({266531818})	0.318	2.860.868
265	frozenset({255922043 ;255922044})	0.000031	frozenset({255922043}) => frozenset({255922044})	0.317	1.595.146
266	frozenset({261217960 ;255732329})	0.000014	frozenset({261217960}) => frozenset({255732329})	0.316	362.635
267	frozenset({256811658 ;258567979})	0.000014	frozenset({256811658}) => frozenset({258567979})	0.316	5.337.992
268	frozenset({270730650 ;274036554})	0.000012	frozenset({270730650}) => frozenset({274036554})	0.312	3.569.181
269	frozenset({274308369 ;273309787})	0.000012	frozenset({274308369}) => frozenset({273309787})	0.312	886.307
270	frozenset({259645425 ;259607046})	0.000019	frozenset({259607046}) => frozenset({259645425})	0.308	122.207
271	frozenset({255732329 ;264465657})	0.000026	frozenset({264465657}) => frozenset({255732329})	0.306	350.883
272	frozenset({260492124 ;260492126})	0.000045	frozenset({260492124}) => frozenset({260492126})	0.302	2832.18
273	frozenset({254975672 ;251511636})	0.000014	frozenset({254975672}) => frozenset({251511636})	0.3	147.587
274	frozenset({256906985 ;256901791})	0.000014	frozenset({256906985}) => frozenset({256901791})	0.3	2.224.163
275	frozenset({245627217 ;245629726})	0.000012	frozenset({245629726}) => frozenset({245627217})	0.294	4.009.402
276	frozenset({254407643 ;254409123})	0.000012	frozenset({254407643}) => frozenset({254409123})	0.294	13.810.163
277	frozenset({270275813 ;270275815})	0.000040	frozenset({270275813}) => frozenset({270275815})	0.293	1.629.775
278	frozenset({250831675 ;250654478})	0.000047	frozenset({250831675}) => frozenset({250654478})	0.286	84.552
279	frozenset({270091995 ;273579892})	0.000024	frozenset({273579892}) => frozenset({270091995})	0.286	588.977
280	frozenset({245299869 ;245293983})	0.000026	frozenset({245293983}) => frozenset({245299869})	0.282	1.702.748
281	frozenset({259037031 ;256901791})	0.000012	frozenset({259037031}) => frozenset({256901791})	0.278	2059.41
282	frozenset({256901582 ;256901791})	0.000014	frozenset({256901582}) => frozenset({256901791})	0.273	2.021.967

283	frozenset({256905190 ;256901582})	0.000014	frozenset({256901582}) => frozenset({256905190})	0.273	12.805.788
284	frozenset({268775315 ;268838660})	0.000017	frozenset({268775315}) => frozenset({268838660})	0.269	7.584.967
285	frozenset({270283868 ;276737494})	0.000019	frozenset({270283868}) => frozenset({276737494})	0.267	1.043.435
286	frozenset({256464638 ;250654478})	0.000080	frozenset({256464638}) => frozenset({250654478})	0.264	77.998
287	frozenset({266852977 ;255767438})	0.000031	frozenset({266852977}) => frozenset({255767438})	0.26	1.408.637
288	frozenset({242452616 ;242452614})	0.000021	frozenset({242452614}) => frozenset({242452616})	0.25	1.993.354
289	frozenset({263088225 ;255734364})	0.000017	frozenset({263088225}) => frozenset({255734364})	0.25	355.716
290	frozenset({258567979 ;256901791})	0.000033	frozenset({256901791}) => frozenset({258567979})	0.246	4.151.771
291	frozenset({265838984 ;265964602})	0.000026	frozenset({265838984}) => frozenset({265964602})	0.244	128.164
292	frozenset({255922043 ;255935181})	0.000024	frozenset({255922043}) => frozenset({255935181})	0.244	866.143
293	frozenset({266531818 ;269366242})	0.000019	frozenset({269366242}) => frozenset({266531818})	0.242	2.179.709
294	frozenset({258568010 ;258567979})	0.000014	frozenset({258567979}) => frozenset({258568010})	0.24	7801.68
295	frozenset({265366674 ;257246527})	0.000014	frozenset({257246527}) => frozenset({265366674})	0.231	308.611
296	frozenset({268779328 ;268775315})	0.000014	frozenset({268775315}) => frozenset({268779328})	0.231	2.786.314
297	frozenset({261925305 ;261926809})	0.000012	frozenset({261925305}) => frozenset({261926809})	0.227	3.557.163
298	frozenset({269521842 ;269366242})	0.000012	frozenset({269521842}) => frozenset({269366242})	0.227	2.910.406
299	frozenset({269542586 ;269537491})	0.000012	frozenset({269542586}) => frozenset({269537491})	0.227	2.182.805
300	frozenset({243493373 ;245088925})	0.000050	frozenset({243493373}) => frozenset({245088925})	0.226	1.004.461
301	frozenset({258567677 ;256901791})	0.000014	frozenset({258567677}) => frozenset({256901791})	0.222	1.647.528
302	frozenset({256916065 ;256901791})	0.000012	frozenset({256916065}) => frozenset({256901791})	0.217	1.611.712
303	frozenset({255733792 ;255733780})	0.000024	frozenset({255733780}) => frozenset({255733792})	0.208	2.379.454
304	frozenset({265622715 ;268790069})	0.000014	frozenset({268790069}) => frozenset({265622715})	0.207	126.165
305	frozenset({272188933 ;271842798})	0.000014	frozenset({272188933}) => frozenset({271842798})	0.207	1.231.445
306	frozenset({273309576 ;274518714})	0.000024	frozenset({274518714}) => frozenset({273309576})	0.204	149.728
307	frozenset({255733794 ;255733797})	0.000031	frozenset({255733794}) => frozenset({255733797})	0.203	1.341.231
308	frozenset({255922044 ;255935181})	0.000040	frozenset({255922044}) => frozenset({255935181})	0.202	718.692
309	frozenset({250208729 ;250654478})	0.000014	frozenset({250208729}) => frozenset({250654478})	0.2	59.186

APPENDIX B: RESULTS OF MODEL NO 2

#	Rule	Support	LeftHandSide => RightHandSide	Confidence	Lift
1	frozenset({243983529;248475563})	0.000012	frozenset({243983529}) => frozenset({248475563})	1.0	84518,20
2	frozenset({243983529;248475564})	0.000012	frozenset({243983529}) => frozenset({248475564})	1.0	84518,20
3	frozenset({243983529;248683252})	0.000012	frozenset({243983529}) => frozenset({248683252})	1.0	60370,14
4	frozenset({243983529;251346844})	0.000012	frozenset({243983529}) => frozenset({251346844})	1.0	70431,83
5	frozenset({243983529;252944451})	0.000012	frozenset({243983529}) => frozenset({252944451})	1.0	70431,83
6	frozenset({248683252;248378431})	0.000012	frozenset({248378431}) => frozenset({248683252})	1.0	60370,14
7	frozenset({248687298;248378431})	0.000012	frozenset({248378431}) => frozenset({248687298})	1.0	70431,83
8	frozenset({248475563;248475564})	0.000012	frozenset({248475563}) => frozenset({248475564})	1.0	84518,20
9	frozenset({248475563;248683252})	0.000012	frozenset({248475563}) => frozenset({248683252})	1.0	60370,14
10	frozenset({248475563;251346844})	0.000012	frozenset({248475563}) => frozenset({251346844})	1.0	70431,83
11	frozenset({252944451;248475563})	0.000012	frozenset({248475563}) => frozenset({252944451})	1.0	70431,83
12	frozenset({248683252;248475564})	0.000012	frozenset({248475564}) => frozenset({248683252})	1.0	60370,14
13	frozenset({251346844;248475564})	0.000012	frozenset({248475564}) => frozenset({251346844})	1.0	70431,83
14	frozenset({252944451;248475564})	0.000012	frozenset({248475564}) => frozenset({252944451})	1.0	70431,83
15	frozenset({256099537;256099250})	0.000026	frozenset({256099250}) => frozenset({256099537})	1.0	38417,36
16	frozenset({256099250;256099918})	0.000026	frozenset({256099250}) => frozenset({256099918})	1.0	38417,36
17	frozenset({256099250;256100125})	0.000026	frozenset({256099250}) => frozenset({256100125})	1.0	35,215,917
18	frozenset({256099250;256100381})	0.000026	frozenset({256099250}) => frozenset({256100381})	1.0	32507,00
19	frozenset({256099250;256101039})	0.000026	frozenset({256099250}) => frozenset({256101039})	1.0	35215,92
20	frozenset({256101896;256099250})	0.000026	frozenset({256099250}) => frozenset({256101896})	1.0	35215,92
21	frozenset({256099537;256099918})	0.000026	frozenset({256099537}) => frozenset({256099918})	1.0	38417,36
22	frozenset({256099537;256100125})	0.000026	frozenset({256099537}) => frozenset({256100125})	1.0	35215,92
23	frozenset({256099537;256100381})	0.000026	frozenset({256099537}) => frozenset({256100381})	1.0	32507,00
24	frozenset({256099537;256101039})	0.000026	frozenset({256099537}) => frozenset({256101039})	1.0	35215,92

25	frozenset({256101896;256099537})	0.000026	frozenset({256099537}) => frozenset({256101896})	1.0	35215,92
26	frozenset({256100125;256099918})	0.000026	frozenset({256099918}) => frozenset({256100125})	1.0	35215,92
27	frozenset({256100381;256099918})	0.000026	frozenset({256099918}) => frozenset({256100381})	1.0	32507,00
28	frozenset({256099918;256101039})	0.000026	frozenset({256099918}) => frozenset({256101039})	1.0	35215,92
29	frozenset({256101896;256099918})	0.000026	frozenset({256099918}) => frozenset({256101896})	1.0	35215,92
30	frozenset({258567979;256908963})	0.000012	frozenset({256908963}) => frozenset({258567979})	1.0	16903,64
31	frozenset({266842569;266842564})	0.000017	frozenset({266842564}) => frozenset({266842569})	1.0	38417,36
32	frozenset({266842564;266842573})	0.000017	frozenset({266842564}) => frozenset({266842573})	1.0	38417,36
33	frozenset({266842579;266842564})	0.000017	frozenset({266842564}) => frozenset({266842579})	1.0	35215,92
34	frozenset({266842569;266842573})	0.000026	frozenset({266842569}) => frozenset({266842573})	1.0	38417,36
35	frozenset({266842569;266842579})	0.000026	frozenset({266842569}) => frozenset({266842579})	1.0	35215,92
36	frozenset({266842579;266842573})	0.000026	frozenset({266842573}) => frozenset({266842579})	1.0	35215,92
37	frozenset({266842577;266842579})	0.000024	frozenset({266842577}) => frozenset({266842579})	1.0	35215,92
38	frozenset({243983529;248475563;248475564})	0.000012	frozenset({243983529;248475563}) => frozenset({248475564})	1.0	84518,20
39	frozenset({243983529;248475563;248683252})	0.000012	frozenset({243983529;248475563}) => frozenset({248683252})	1.0	60370,14
40	frozenset({243983529;248475563;251346844})	0.000012	frozenset({243983529;248475563}) => frozenset({251346844})	1.0	70431,83
41	frozenset({243983529;252944451;248475563})	0.000012	frozenset({243983529;248475563}) => frozenset({252944451})	1.0	70431,83
42	frozenset({243983529;248683252;248475564})	0.000012	frozenset({243983529;248475564}) => frozenset({248683252})	1.0	60370,14
43	frozenset({243983529;248475564;251346844})	0.000012	frozenset({243983529;248475564}) => frozenset({251346844})	1.0	70431,83
44	frozenset({243983529;252944451;248475564})	0.000012	frozenset({243983529;248475564}) => frozenset({252944451})	1.0	70431,83
45	frozenset({243983529;248683252;251346844})	0.000012	frozenset({243983529;248683252}) => frozenset({251346844})	1.0	70431,83
46	frozenset({243983529;252944451;248683252})	0.000012	frozenset({243983529;248683252}) => frozenset({252944451})	1.0	70431,83

47	frozenset({243983529;252944451;251346844})	0.000012	frozenset({243983529;251346844}) => frozenset({252944451})	1.0	70431,83
48	frozenset({248687298;248683252;248378431})	0.000012	frozenset({248683252;248378431}) => frozenset({248687298})	1.0	70431,83
49	frozenset({248475563;248475564;248683252})	0.000012	frozenset({248475563;248475564}) => frozenset({248683252})	1.0	60370,14
50	frozenset({251346844;248475563;248475564})	0.000012	frozenset({248475563;248475564}) => frozenset({251346844})	1.0	70431,83
51	frozenset({252944451;248475563;248475564})	0.000012	frozenset({248475563;248475564}) => frozenset({252944451})	1.0	70431,83
52	frozenset({251346844;248475563;248683252})	0.000012	frozenset({248475563;248683252}) => frozenset({251346844})	1.0	70431,83
53	frozenset({252944451;248475563;248683252})	0.000012	frozenset({248475563;248683252}) => frozenset({252944451})	1.0	70431,83
54	frozenset({252944451;248475563;251346844})	0.000012	frozenset({248475563;251346844}) => frozenset({252944451})	1.0	70431,83
55	frozenset({251346844;248683252;248475564})	0.000012	frozenset({248683252;248475564}) => frozenset({251346844})	1.0	70431,83
56	frozenset({252944451;248683252;248475564})	0.000012	frozenset({248683252;248475564}) => frozenset({252944451})	1.0	70431,83
57	frozenset({251346844;252944451;248475564})	0.000012	frozenset({251346844;248475564}) => frozenset({252944451})	1.0	70431,83
58	frozenset({256099537;256099250;256099918})	0.000026	frozenset({256099537;256099250}) => frozenset({256099918})	1.0	38417,36
59	frozenset({256099537;256099250;256100125})	0.000026	frozenset({256099537;256099250}) => frozenset({256100125})	1.0	35215,92
60	frozenset({256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250}) => frozenset({256100381})	1.0	32507,00
61	frozenset({256099537;256099250;256101039})	0.000026	frozenset({256099537;256099250}) => frozenset({256101039})	1.0	35215,92
62	frozenset({256101896;256099537;256099250})	0.000026	frozenset({256099537;256099250}) => frozenset({256101896})	1.0	35215,92
63	frozenset({256099250;256100125;256099918})	0.000026	frozenset({256099250;256099918}) => frozenset({256100125})	1.0	35215,92
64	frozenset({256099250;256100381;256099918})	0.000026	frozenset({256099250;256099918}) => frozenset({256100381})	1.0	32507,00

65	frozenset({256099250;256099918;256101039})	0.000026	frozenset({256099250;256099918}) => frozenset({256101039})	1.0	35215,92
66	frozenset({256101896;256099250;256099918})	0.000026	frozenset({256099250;256099918}) => frozenset({256101896})	1.0	35215,92
67	frozenset({256100381;256099250;256100125})	0.000026	frozenset({256099250;256100125}) => frozenset({256100381})	1.0	32507,00
68	frozenset({256099250;256100125;256101039})	0.000026	frozenset({256099250;256100125}) => frozenset({256101039})	1.0	35215,92
69	frozenset({256101896;256099250;256100125})	0.000026	frozenset({256099250;256100125}) => frozenset({256101896})	1.0	35215,92
70	frozenset({256099250;256100381;256101039})	0.000026	frozenset({256099250;256100381}) => frozenset({256101039})	1.0	35215,92
71	frozenset({256101896;256099250;256100381})	0.000026	frozenset({256099250;256100381}) => frozenset({256101896})	1.0	35215,92
72	frozenset({256101896;256099250;256101039})	0.000026	frozenset({256099250;256101039}) => frozenset({256101896})	1.0	35215,92
73	frozenset({256099537;256100125;256099918})	0.000026	frozenset({256099537;256099918}) => frozenset({256100125})	1.0	35215,92
74	frozenset({256099537;256100381;256099918})	0.000026	frozenset({256099537;256099918}) => frozenset({256100381})	1.0	32507,00
75	frozenset({256099537;256099918;256101039})	0.000026	frozenset({256099537;256099918}) => frozenset({256101039})	1.0	35215,92
76	frozenset({256101896;256099537;256099918})	0.000026	frozenset({256099537;256099918}) => frozenset({256101896})	1.0	35215,92
77	frozenset({256099537;256100381;256100125})	0.000026	frozenset({256099537;256100125}) => frozenset({256100381})	1.0	32507,00
78	frozenset({256099537;256100125;256101039})	0.000026	frozenset({256099537;256100125}) => frozenset({256101039})	1.0	35215,92
79	frozenset({256101896;256099537;256100125})	0.000026	frozenset({256099537;256100125}) => frozenset({256101896})	1.0	35215,92
80	frozenset({256099537;256100381;256101039})	0.000026	frozenset({256099537;256100381}) => frozenset({256101039})	1.0	35215,92
81	frozenset({256101896;256099537;256100381})	0.000026	frozenset({256099537;256100381}) => frozenset({256101896})	1.0	35215,92
82	frozenset({256101896;256099537;256101039})	0.000026	frozenset({256099537;256101039}) => frozenset({256101896})	1.0	35215,92

83	frozenset({256100381;256100125;256099918})	0.000026	frozenset({256100125;256099918}) => frozenset({256100381})	1.0	32507,00
84	frozenset({256100125;256099918;256101039})	0.000026	frozenset({256100125;256099918}) => frozenset({256101039})	1.0	35215,92
85	frozenset({256101896;256100125;256099918})	0.000026	frozenset({256100125;256099918}) => frozenset({256101896})	1.0	35215,92
86	frozenset({256100381;256099918;256101039})	0.000026	frozenset({256100381;256099918}) => frozenset({256101039})	1.0	35215,92
87	frozenset({256101896;256100381;256099918})	0.000026	frozenset({256100381;256099918}) => frozenset({256101896})	1.0	35215,92
88	frozenset({256101896;256099918;256101039})	0.000026	frozenset({256099918;256101039}) => frozenset({256101896})	1.0	35215,92
89	frozenset({256100381;256100125;256101039})	0.000026	frozenset({256100381;256100125}) => frozenset({256101039})	1.0	35215,92
90	frozenset({256101896;256100381;256100125})	0.000026	frozenset({256100381;256100125}) => frozenset({256101896})	1.0	35215,92
91	frozenset({256101896;256100125;256101039})	0.000026	frozenset({256100125;256101039}) => frozenset({256101896})	1.0	35215,92
92	frozenset({256101896;256100381;256101039})	0.000026	frozenset({256100381;256101039}) => frozenset({256101896})	1.0	35215,92
93	frozenset({266842569;266842564;266842573})	0.000017	frozenset({266842569;266842564}) => frozenset({266842573})	1.0	38417,36
94	frozenset({266842569;266842579;266842564})	0.000017	frozenset({266842569;266842564}) => frozenset({266842579})	1.0	35215,92
95	frozenset({266842579;266842564;266842573})	0.000017	frozenset({266842564;266842573}) => frozenset({266842579})	1.0	35215,92
96	frozenset({266842577;266842579;266842564})	0.000014	frozenset({266842577;266842564}) => frozenset({266842579})	1.0	35215,92
97	frozenset({266842569;266842579;266842573})	0.000026	frozenset({266842569;266842573}) => frozenset({266842579})	1.0	35215,92
98	frozenset({266842569;266842579;266842577})	0.000024	frozenset({266842569;266842577}) => frozenset({266842579})	1.0	35215,92
99	frozenset({266842577;266842579;266842573})	0.000024	frozenset({266842577;266842573}) => frozenset({266842579})	1.0	35215,92
100	frozenset({243983529;248475563;248475564;248683252})	0.000012	frozenset({243983529;248475563;248475564}) => frozenset({248683252})	1.0	60370,14

101	frozenset({243983529;248475563;248475564;251346844})	0.000012	frozenset({243983529;248475563;248475564}) => frozenset({251346844})	1.0	70431,83
102	frozenset({243983529;252944451;248475563;248475564})	0.000012	frozenset({243983529;248475563;248475564}) => frozenset({252944451})	1.0	70431,83
103	frozenset({243983529;248475563;248683252;251346844})	0.000012	frozenset({243983529;248475563;248683252}) => frozenset({251346844})	1.0	70431,83
104	frozenset({243983529;252944451;248475563;248683252})	0.000012	frozenset({243983529;248475563;248683252}) => frozenset({252944451})	1.0	70431,83
105	frozenset({243983529;252944451;248475563;251346844})	0.000012	frozenset({243983529;248475563;251346844}) => frozenset({252944451})	1.0	70431,83
106	frozenset({243983529;248683252;248475564;251346844})	0.000012	frozenset({243983529;248683252;248475564}) => frozenset({251346844})	1.0	70431,83
107	frozenset({243983529;252944451;248683252;248475564})	0.000012	frozenset({243983529;248683252;248475564}) => frozenset({252944451})	1.0	70431,83
108	frozenset({243983529;252944451;248475564;251346844})	0.000012	frozenset({243983529;248475564;251346844}) => frozenset({252944451})	1.0	70431,83
109	frozenset({243983529;252944451;248683252;251346844})	0.000012	frozenset({243983529;248683252;251346844}) => frozenset({252944451})	1.0	70431,83
110	frozenset({251346844;248475563;248475564;248683252})	0.000012	frozenset({248475563;248475564;248683252}) => frozenset({251346844})	1.0	70431,83
111	frozenset({252944451;248475563;248475564;248683252})	0.000012	frozenset({248475563;248475564;248683252}) => frozenset({252944451})	1.0	70431,83
112	frozenset({251346844;252944451;248475563;248475564})	0.000012	frozenset({251346844;248475563;248475564}) => frozenset({252944451})	1.0	70431,83
113	frozenset({251346844;252944451;248475563;248683252})	0.000012	frozenset({251346844;248475563;248683252}) => frozenset({252944451})	1.0	70431,83
114	frozenset({251346844;252944451;248683252;248475564})	0.000012	frozenset({251346844;248683252;248475564}) => frozenset({252944451})	1.0	70431,83
115	frozenset({256099537;256099250;256100125;256099918})	0.000026	frozenset({256099537;256099250;256099918}) => frozenset({256100125})	1.0	35215,92
116	frozenset({256099537;256099250;256100381;256099918})	0.000026	frozenset({256099537;256099250;256099918}) => frozenset({256100381})	1.0	32507,00
117	frozenset({256099537;256099250;256099918;256101039})	0.000026	frozenset({256099537;256099250;256099918}) => frozenset({256101039})	1.0	35215,92
118	frozenset({256101896;256099537;256099250;256099918})	0.000026	frozenset({256099537;256099250;256099918}) => frozenset({256101896})	1.0	35,215,917

119	frozenset({256099537;256099250;256100381;256100125})	0.000026	frozenset({256099537;256099250;256100125}) => frozenset({256100381})	1.0	32507,00
120	frozenset({256099537;256099250;256100125;256101039})	0.000026	frozenset({256099537;256099250;256100125}) => frozenset({256101039})	1.0	35215,92
121	frozenset({256101896;256099537;256099250;256100125})	0.000026	frozenset({256099537;256099250;256100125}) => frozenset({256101896})	1.0	35215,92
122	frozenset({256099537;256099250;256100381;256101039})	0.000026	frozenset({256099537;256099250;256100381}) => frozenset({256101039})	1.0	35215,92
123	frozenset({256101896;256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250;256100381}) => frozenset({256101896})	1.0	35215,92
124	frozenset({256101896;256099537;256099250;256101039})	0.000026	frozenset({256099537;256099250;256101039}) => frozenset({256101896})	1.0	35215,92
125	frozenset({256100381;256099250;256100125;256099918})	0.000026	frozenset({256099250;256100125;256099918}) => frozenset({256100381})	1.0	32507,00
126	frozenset({256099250;256100125;256099918;256101039})	0.000026	frozenset({256099250;256100125;256099918}) => frozenset({256101039})	1.0	35215,92
127	frozenset({256101896;256099250;256100125;256099918})	0.000026	frozenset({256099250;256100125;256099918}) => frozenset({256101896})	1.0	35215,92
128	frozenset({256099250;256100381;256099918;256101039})	0.000026	frozenset({256099250;256100381;256099918}) => frozenset({256101039})	1.0	35215,92
129	frozenset({256101896;256099250;256100381;256099918})	0.000026	frozenset({256099250;256100381;256099918}) => frozenset({256101896})	1.0	35215,92
130	frozenset({256101896;256099250;256099918;256101039})	0.000026	frozenset({256099250;256099918;256101039}) => frozenset({256101896})	1.0	35215,92
131	frozenset({256100381;256099250;256100125;256101039})	0.000026	frozenset({256100381;256099250;256100125}) => frozenset({256101039})	1.0	35215,92
132	frozenset({256101896;256100381;256099250;256100125})	0.000026	frozenset({256100381;256099250;256100125}) => frozenset({256101896})	1.0	35215,92
133	frozenset({256101896;256099250;256100125;256101039})	0.000026	frozenset({256099250;256100125;256101039}) => frozenset({256101896})	1.0	35215,92
134	frozenset({256101896;256099250;256100381;256101039})	0.000026	frozenset({256099250;256100381;256101039}) => frozenset({256101896})	1.0	35215,92
135	frozenset({256099537;256100381;256100125;256099918})	0.000026	frozenset({256099537;256100125;256099918}) => frozenset({256100381})	1.0	32507,00
136	frozenset({256099537;256100125;256099918;256101039})	0.000026	frozenset({256099537;256100125;256099918}) => frozenset({256101039})	1.0	35215,92

137	frozenset({256101896;256099537;256100125;256099918})	0.000026	frozenset({256099537;256100125;256099918}) => frozenset({256101896})	1.0	35215,92
138	frozenset({256099537;256100381;256099918;256101039})	0.000026	frozenset({256099537;256100381;256099918}) => frozenset({256101039})	1.0	35215,92
139	frozenset({256101896;256099537;256100381;256099918})	0.000026	frozenset({256099537;256100381;256099918}) => frozenset({256101896})	1.0	35215,92
140	frozenset({256101896;256099537;256099918;256101039})	0.000026	frozenset({256099537;256099918;256101039}) => frozenset({256101896})	1.0	35215,92
141	frozenset({256099537;256100381;256100125;256101039})	0.000026	frozenset({256099537;256100381;256100125}) => frozenset({256101039})	1.0	35215,92
142	frozenset({256101896;256099537;256100381;256100125})	0.000026	frozenset({256099537;256100381;256100125}) => frozenset({256101896})	1.0	35215,92
143	frozenset({256101896;256099537;256100125;256101039})	0.000026	frozenset({256099537;256100125;256101039}) => frozenset({256101896})	1.0	35215,92
144	frozenset({256101896;256099537;256100381;256101039})	0.000026	frozenset({256099537;256100381;256101039}) => frozenset({256101896})	1.0	35215,92
145	frozenset({256100381;256100125;256099918;256101039})	0.000026	frozenset({256100381;256100125;256099918}) => frozenset({256101039})	1.0	35215,92
146	frozenset({256101896;256100381;256100125;256099918})	0.000026	frozenset({256100381;256100125;256099918}) => frozenset({256101896})	1.0	35215,92
147	frozenset({256101896;256100125;256099918;256101039})	0.000026	frozenset({256100125;256099918;256101039}) => frozenset({256101896})	1.0	35215,92
148	frozenset({256101896;256100381;256099918;256101039})	0.000026	frozenset({256100381;256099918;256101039}) => frozenset({256101896})	1.0	35215,92
149	frozenset({256101896;256100381;256100125;256101039})	0.000026	frozenset({256100381;256100125;256101039}) => frozenset({256101896})	1.0	35215,92
150	frozenset({266842569;266842579;266842564;266842573})	0.000017	frozenset({266842569;266842564;266842573}) => frozenset({266842579})	1.0	35215,92
151	frozenset({266842569;266842579;266842564;266842577})	0.000014	frozenset({266842569;266842564;266842577}) => frozenset({266842579})	1.0	35215,92
152	frozenset({266842577;266842579;266842564;266842573})	0.000014	frozenset({266842577;266842564;266842573}) => frozenset({266842579})	1.0	35215,92
153	frozenset({266842569;266842579;266842577;266842573})	0.000024	frozenset({266842569;266842577;266842573}) => frozenset({266842579})	1.0	35215,92
154	frozenset({243983529;248475563;248475564;248683252;2	0.000012	frozenset({243983529;248475563;248475564;248683252}) => frozenset({251346844})	1.0	70431,83

	51346844})				
155	frozenset({252944451;243983529;248475563;248475564;248683252})	0.000012	frozenset({243983529;248475563;248475564;248683252}) => frozenset({252944451})	1.0	70431,83
156	frozenset({252944451;243983529;248475563;248475564;251346844})	0.000012	frozenset({243983529;248475563;248475564;251346844}) => frozenset({252944451})	1.0	70431,83
157	frozenset({252944451;243983529;248475563;248683252;251346844})	0.000012	frozenset({243983529;248475563;248683252;251346844}) => frozenset({252944451})	1.0	70431,83
158	frozenset({252944451;243983529;248475564;248683252;251346844})	0.000012	frozenset({243983529;248683252;248475564;251346844}) => frozenset({252944451})	1.0	70431,83
159	frozenset({252944451;248475563;248475564;248683252;251346844})	0.000012	frozenset({251346844;248475563;248475564;248683252}) => frozenset({252944451})	1.0	70431,83
160	frozenset({256100125;256099918;256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250;256100125;256099918}) => frozenset({256100381})	1.0	32507,00
161	frozenset({256099918;256101039;256099537;256099250;256100125})	0.000026	frozenset({256099537;256099250;256100125;256099918}) => frozenset({256101039})	1.0	35215,92
162	frozenset({256101896;256099918;256099537;256099250;256100125})	0.000026	frozenset({256099537;256099250;256100125;256099918}) => frozenset({256101896})	1.0	35215,92
163	frozenset({256099918;256101039;256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250;256100381;256099918}) => frozenset({256101039})	1.0	35215,92
164	frozenset({256101896;256099918;256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250;256100381;256099918}) => frozenset({256101896})	1.0	35215,92
165	frozenset({256101896;256099918;256101039;256099537;256099250})	0.000026	frozenset({256099537;256099250;256099918;256101039}) => frozenset({256101896})	1.0	35215,92
166	frozenset({256100125;256101039;256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250;256100381;256100125}) => frozenset({256101039})	1.0	35215,92
167	frozenset({256101896;256100125;256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250;256100381;256100125}) => frozenset({256101896})	1.0	35215,92
168	frozenset({2561018	0.000026	frozenset({256099537;256099250	1.0	35215,92

	96;256101039;256099537;256099250;256100125})		;256100125;256101039}) => frozenset({256101896})		
169	frozenset({256101896;256101039;256099537;256099250;256100381})	0.000026	frozenset({256099537;256099250;256100381;256101039}) => frozenset({256101896})	1.0	35215,92
170	frozenset({256100125;256099918;256101039;256099250;256100381})	0.000026	frozenset({256100381;256099250;256100125;256099918}) => frozenset({256101039})	1.0	35215,92
171	frozenset({256101896;256100125;256099918;256099250;256100381})	0.000026	frozenset({256100381;256099250;256100125;256099918}) => frozenset({256101896})	1.0	35215,92
172	frozenset({256101896;256099918;256101039;256099250;256100125})	0.000026	frozenset({256099250;256100125;256099918;256101039}) => frozenset({256101896})	1.0	35215,92
173	frozenset({256101896;256099918;256101039;256099250;256100381})	0.000026	frozenset({256099250;256100381;256099918;256101039}) => frozenset({256101896})	1.0	35215,92
174	frozenset({256101896;256100125;256101039;256099250;256100381})	0.000026	frozenset({256100381;256099250;256100125;256101039}) => frozenset({256101896})	1.0	35215,92
175	frozenset({256100125;256099918;256101039;256099537;256100381})	0.000026	frozenset({256099537;256100381;256100125;256099918}) => frozenset({256101039})	1.0	35215,92
176	frozenset({256101896;256100125;256099918;256099537;256100381})	0.000026	frozenset({256099537;256100381;256100125;256099918}) => frozenset({256101896})	1.0	35215,92
177	frozenset({256101896;256099918;256101039;256099537;256100125})	0.000026	frozenset({256099537;256100125;256099918;256101039}) => frozenset({256101896})	1.0	35215,92
178	frozenset({256101896;256099918;256101039;256099537;256100381})	0.000026	frozenset({256099537;256100381;256099918;256101039}) => frozenset({256101896})	1.0	35215,92
179	frozenset({256101896;256100125;256101039;256099537;256100381})	0.000026	frozenset({256099537;256100381;256100125;256101039}) => frozenset({256101896})	1.0	35215,92
180	frozenset({256101896;256100125;256099918;256101039;256100381})	0.000026	frozenset({256100381;256100125;256099918;256101039}) => frozenset({256101896})	1.0	35215,92
181	frozenset({266842564;266842569;266842573;266842577;266842579})	0.000014	frozenset({266842577;266842569;266842564;266842573}) => frozenset({266842579})	1.0	35215,92

182	frozenset({252944451;243983529;248475563;248475564;248683252;251346844})	0.000012	frozenset({243983529;248475563;248475564;248683252;251346844}) => frozenset({252944451})	1.0	70431,83
183	frozenset({256100125;256099918;256101039;256099537;256099250;256100381})	0.000026	frozenset({256100125;256099918;256099537;256099250;256100381}) => frozenset({256101039})	1.0	35215,92
184	frozenset({256101896;256100125;256099918;256099537;256099250;256100381})	0.000026	frozenset({256100125;256099918;256099537;256099250;256100381}) => frozenset({256101896})	1.0	35215,92
185	frozenset({256101896;256099918;256101039;256099537;256099250;256100125})	0.000026	frozenset({256099918;256101039;256099537;256099250;256100125}) => frozenset({256101896})	1.0	35215,92
186	frozenset({256101896;256099918;256101039;256099537;256099250;256100381})	0.000026	frozenset({256099918;256101039;256099537;256099250;256100381}) => frozenset({256101896})	1.0	35215,92
187	frozenset({256101896;256100125;256101039;256099537;256099250;256100381})	0.000026	frozenset({256100125;256101039;256099537;256099250;256100381}) => frozenset({256101896})	1.0	35215,92
188	frozenset({256101896;256100125;256099918;256101039;256099250;256100381})	0.000026	frozenset({256100125;256099918;256101039;256099250;256100381}) => frozenset({256101896})	1.0	35215,92
189	frozenset({256101896;256100125;256099918;256101039;256099537;256100381})	0.000026	frozenset({256100125;256099918;256101039;256099537;256100381}) => frozenset({256101896})	1.0	35215,92
190	frozenset({256101896;256100125;256099918;256101039;256099537;256099250;256100381})	0.000026	frozenset({256100125;256099918;256101039;256099537;256099250;256100381}) => frozenset({256101896})	1.0	35215,92
191	frozenset({256100381;256100125})	0.000026	frozenset({256100125}) => frozenset({256100381})	0.917	29798,08
192	frozenset({256100125;256101039})	0.000026	frozenset({256100125}) => frozenset({256101039})	0.917	32281,26
193	frozenset({256101896;256100125})	0.000026	frozenset({256100125}) => frozenset({256101896})	0.917	32281,26
194	frozenset({256101896;256101039})	0.000026	frozenset({256101039}) => frozenset({256101896})	0.917	32281,26
195	frozenset({266842569;266842577})	0.000024	frozenset({266842569}) => frozenset({266842577})	0.909	38417,36

196	frozenset({266842577;266842573})	0.000024	frozenset({266842573}) => frozenset({266842577})	0.909	38417,36
197	frozenset({266842569;266842577;266842573})	0.000024	frozenset({266842569;266842573}) => frozenset({266842577})	0.909	38417,36
198	frozenset({255732329;256050591})	0.000019	frozenset({256050591}) => frozenset({255732329})	0.889	1020,75
199	frozenset({228993720;248669138})	0.000017	frozenset({228993720}) => frozenset({248669138})	0.875	30813,93
200	frozenset({251346844;248683252})	0.000014	frozenset({248683252}) => frozenset({251346844})	0.857	60370,14
201	frozenset({252944451;248683252})	0.000014	frozenset({248683252}) => frozenset({252944451})	0.857	60370,14
202	frozenset({266842577;266842564})	0.000014	frozenset({266842564}) => frozenset({266842577})	0.857	36222,09
203	frozenset({266842569;266842564;266842577})	0.000014	frozenset({266842569;266842564}) => frozenset({266842577})	0.857	36222,09
204	frozenset({266842577;266842564;266842573})	0.000014	frozenset({266842564;266842573}) => frozenset({266842577})	0.857	36222,09
205	frozenset({266842577;266842569;266842564;266842573})	0.000014	frozenset({266842569;266842564;266842573}) => frozenset({266842577})	0.857	36222,09
206	frozenset({256100381;256101039})	0.000026	frozenset({256100381}) => frozenset({256101039})	0.846	29798,08
207	frozenset({256101896;256100381})	0.000026	frozenset({256100381}) => frozenset({256101896})	0.846	29798,08
208	frozenset({236992197;239797102})	0.000012	frozenset({236992197}) => frozenset({239797102})	0.833	70431,83
209	frozenset({252944451;251346844})	0.000012	frozenset({251346844}) => frozenset({252944451})	0.833	58693,19
210	frozenset({256901074;258567979})	0.000012	frozenset({256901074}) => frozenset({258567979})	0.833	14086,37
211	frozenset({251346844;252944451;248683252})	0.000012	frozenset({251346844;248683252}) => frozenset({252944451})	0.833	58693,19
212	frozenset({255922043;255922044;255935181})	0.000024	frozenset({255922043;255922044}) => frozenset({255935181})	0.769	2731,68
213	frozenset({252703306;246620796})	0.000026	frozenset({246620796}) => frozenset({252703306})	0.733	748,55
214	frozenset({248687298;248683252})	0.000012	frozenset({248683252}) => frozenset({248687298})	0.714	50308,45
215	frozenset({251511636;260320446})	0.000019	frozenset({260320446}) => frozenset({251511636})	0.667	327,97
216	frozenset({246648362;250654478})	0.000047	frozenset({246648362}) => frozenset({250654478})	0.667	197,29
217	frozenset({265964602;259694237})	0.000031	frozenset({259694237}) => frozenset({265964602})	0.65	340,80
218	frozenset({255732329;260704449})	0.000012	frozenset({260704449}) => frozenset({255732329})	0.625	717,72
219	frozenset({2568116	0.000012	frozenset({256811658;256901791	0.625	10564,78

	58;258567979;256901791})		}) => frozenset({258567979})		
220	frozenset({251562499;255734364})	0.000012	frozenset({251562499}) => frozenset({255734364})	0.556	790,48
221	frozenset({255732329;251946684})	0.000012	frozenset({251946684}) => frozenset({255732329})	0.556	637,97
222	frozenset({252528723;250654478})	0.000026	frozenset({252528723}) => frozenset({250654478})	0.55	162,76
223	frozenset({248438052;250114941})	0.000014	frozenset({250114941}) => frozenset({248438052})	0.545	441,58
224	frozenset({258567979;256914095})	0.000014	frozenset({256914095}) => frozenset({258567979})	0.545	9220,17
225	frozenset({265363377;265363094})	0.000014	frozenset({265363094}) => frozenset({265363377})	0.545	38417,36
226	frozenset({268827043;268118799})	0.000014	frozenset({268827043}) => frozenset({268118799})	0.545	527,47
227	frozenset({252528723;252528854;250654478})	0.000014	frozenset({252528723;250654478}) => frozenset({252528854})	0.545	8537,19
228	frozenset({261632776;261632873})	0.000017	frozenset({261632776}) => frozenset({261632873})	0.538	12641,61
229	frozenset({252528854;250654478})	0.000033	frozenset({252528854}) => frozenset({250654478})	0.519	153,45
230	frozenset({249582384;251562499})	0.000012	frozenset({249582384}) => frozenset({251562499})	0.5	23477,28
231	frozenset({249582384;255734364})	0.000012	frozenset({249582384}) => frozenset({255734364})	0.5	711,43
232	frozenset({254712410;250654478})	0.000033	frozenset({254712410}) => frozenset({250654478})	0.5	147,97
233	frozenset({255732329;259348732})	0.000014	frozenset({259348732}) => frozenset({255732329})	0.5	574,17
234	frozenset({258424107;260333814})	0.000024	frozenset({260333814}) => frozenset({258424107})	0.5	785,49
235	frozenset({265363569;265363506})	0.000024	frozenset({265363506}) => frozenset({265363569})	0.5	14086,37
236	frozenset({268301345;270093958})	0.000085	frozenset({268301345}) => frozenset({270093958})	0.474	150,28
237	frozenset({255732329;258377850})	0.000017	frozenset({258377850}) => frozenset({255732329})	0.467	535,89
238	frozenset({273309576;270283868})	0.000033	frozenset({270283868}) => frozenset({273309576})	0.467	342,38
239	frozenset({256914095;256901791})	0.000012	frozenset({256914095}) => frozenset({256901791})	0.455	3369,94
240	frozenset({258569352;258424107})	0.000012	frozenset({258569352}) => frozenset({258424107})	0.455	714,08
241	frozenset({258750424;259918909})	0.000012	frozenset({258750424}) => frozenset({259918909})	0.455	1402,09
242	frozenset({252528723;252528854})	0.000021	frozenset({252528723}) => frozenset({252528854})	0.45	7043,18
243	frozenset({268325338;269864739})	0.000019	frozenset({268325338}) => frozenset({269864739})	0.444	3078,99
244	frozenset({250094344;250654478})	0.000014	frozenset({250094344}) => frozenset({250654478})	0.429	126,83

245	frozenset({256811658;256901791})	0.000019	frozenset({256811658}) => frozenset({256901791})	0.421	3121,63
246	frozenset({270934433;269376483})	0.000012	frozenset({270934433}) => frozenset({269376483})	0.417	1189,73
247	frozenset({274995841;275491877})	0.000012	frozenset({274995841}) => frozenset({275491877})	0.417	1333,94
248	frozenset({259603152;255734364})	0.000031	frozenset({259603152}) => frozenset({255734364})	0.406	578,04
249	frozenset({242452616;242452613})	0.000017	frozenset({242452613}) => frozenset({242452616})	0.389	3100,77
250	frozenset({260508220;260508230})	0.000019	frozenset({260508220}) => frozenset({260508230})	0.381	10061,69
251	frozenset({265020401;264770118})	0.000019	frozenset({265020401}) => frozenset({264770118})	0.381	336,09
252	frozenset({245721171;245299869})	0.000059	frozenset({245299869}) => frozenset({245721171})	0.357	1886,57
253	frozenset({270734826;269366242})	0.000012	frozenset({270734826}) => frozenset({269366242})	0.357	4573,50
254	frozenset({270093958;269903495})	0.000012	frozenset({269903495}) => frozenset({270093958})	0.357	113,31
255	frozenset({255734364;262796583})	0.000040	frozenset({262796583}) => frozenset({255734364})	0.347	493,65
256	frozenset({273579892;271537559})	0.000028	frozenset({273579892}) => frozenset({271537559})	0.343	499,62
257	frozenset({256723761;256901791})	0.000012	frozenset({256723761}) => frozenset({256901791})	0.333	2471,29
258	frozenset({256723761;258568010})	0.000012	frozenset({256723761}) => frozenset({258568010})	0.333	10835,67
259	frozenset({256723761;259037031})	0.000012	frozenset({256723761}) => frozenset({259037031})	0.333	7825,76
260	frozenset({258338519;256922831})	0.000012	frozenset({258338519}) => frozenset({256922831})	0.333	378,67
261	frozenset({268779328;268839792})	0.000014	frozenset({268839792}) => frozenset({268779328})	0.333	4024,68
262	frozenset({273309576;273914888})	0.000014	frozenset({273914888}) => frozenset({273309576})	0.333	244,56
263	frozenset({262613127;262613039})	0.000021	frozenset({262613039}) => frozenset({262613127})	0.321	4244,78
264	frozenset({269542586;266531818})	0.000017	frozenset({269542586}) => frozenset({266531818})	0.318	2860,87
265	frozenset({255922043;255922044})	0.000031	frozenset({255922043}) => frozenset({255922044})	0.317	1595,15
266	frozenset({261217960;255732329})	0.000014	frozenset({261217960}) => frozenset({255732329})	0.316	362,64
267	frozenset({256811658;258567979})	0.000014	frozenset({256811658}) => frozenset({258567979})	0.316	5337,99
268	frozenset({270730650;274036554})	0.000012	frozenset({270730650}) => frozenset({274036554})	0.312	3569,18
269	frozenset({274308369;273309787})	0.000012	frozenset({274308369}) => frozenset({273309787})	0.312	886,31
270	frozenset({259645425;259607046})	0.000019	frozenset({259607046}) => frozenset({259645425})	0.308	122,21

271	frozenset({255732329;264465657})	0.000026	frozenset({264465657}) => frozenset({255732329})	0.306	350,88
272	frozenset({260492124;260492126})	0.000045	frozenset({260492124}) => frozenset({260492126})	0.302	2832,18
273	frozenset({254975672;251511636})	0.000014	frozenset({254975672}) => frozenset({251511636})	0.3	147,59
274	frozenset({256906985;256901791})	0.000014	frozenset({256906985}) => frozenset({256901791})	0.3	2224,16
275	frozenset({245627217;245629726})	0.000012	frozenset({245629726}) => frozenset({245627217})	0.294	4009,40
276	frozenset({254407643;254409123})	0.000012	frozenset({254407643}) => frozenset({254409123})	0.294	13810,16
277	frozenset({270275813;270275815})	0.000040	frozenset({270275813}) => frozenset({270275815})	0.293	1629,78
278	frozenset({250831675;250654478})	0.000047	frozenset({250831675}) => frozenset({250654478})	0.286	84,55
279	frozenset({270091995;273579892})	0.000024	frozenset({273579892}) => frozenset({270091995})	0.286	588,98
280	frozenset({245299869;245293983})	0.000026	frozenset({245293983}) => frozenset({245299869})	0.282	1702,75
281	frozenset({259037031;256901791})	0.000012	frozenset({259037031}) => frozenset({256901791})	0.278	2059,41
282	frozenset({256901582;256901791})	0.000014	frozenset({256901582}) => frozenset({256901791})	0.273	2021,97
283	frozenset({256905190;256901582})	0.000014	frozenset({256901582}) => frozenset({256905190})	0.273	12805,79
284	frozenset({268775315;268838660})	0.000017	frozenset({268775315}) => frozenset({268838660})	0.269	7584,97
285	frozenset({270283868;276737494})	0.000019	frozenset({270283868}) => frozenset({276737494})	0.267	1043,44
286	frozenset({256464638;250654478})	0.000080	frozenset({256464638}) => frozenset({250654478})	0.264	78,00
287	frozenset({266852977;255767438})	0.000031	frozenset({266852977}) => frozenset({255767438})	0.26	1408,64
288	frozenset({242452616;242452614})	0.000021	frozenset({242452614}) => frozenset({242452616})	0.25	1993,35
289	frozenset({263088225;255734364})	0.000017	frozenset({263088225}) => frozenset({255734364})	0.25	355,72
290	frozenset({258567979;256901791})	0.000033	frozenset({256901791}) => frozenset({258567979})	0.246	4151,77
291	frozenset({265838984;265964602})	0.000026	frozenset({265838984}) => frozenset({265964602})	0.244	128,16
292	frozenset({255922043;255935181})	0.000024	frozenset({255922043}) => frozenset({255935181})	0.244	866,14
293	frozenset({266531818;269366242})	0.000019	frozenset({269366242}) => frozenset({266531818})	0.242	2179,71
294	frozenset({258568010;258567979})	0.000014	frozenset({258567979}) => frozenset({258568010})	0.24	7801,68
295	frozenset({265366674;257246527})	0.000014	frozenset({257246527}) => frozenset({265366674})	0.231	308,61
296	frozenset({268779328;268775315})	0.000014	frozenset({268775315}) => frozenset({268779328})	0.231	2786,31

297	frozenset({261925305;261926809})	0.000012	frozenset({261925305}) => frozenset({261926809})	0.227	3557,16
298	frozenset({269521842;269366242})	0.000012	frozenset({269521842}) => frozenset({269366242})	0.227	2910,41
299	frozenset({269542586;269537491})	0.000012	frozenset({269542586}) => frozenset({269537491})	0.227	2182,81
300	frozenset({243493373;245088925})	0.000050	frozenset({243493373}) => frozenset({245088925})	0.226	1004,46
301	frozenset({258567677;256901791})	0.000014	frozenset({258567677}) => frozenset({256901791})	0.222	1647,53
302	frozenset({256916065;256901791})	0.000012	frozenset({256916065}) => frozenset({256901791})	0.217	1611,71
303	frozenset({255733792;255733780})	0.000024	frozenset({255733780}) => frozenset({255733792})	0.208	2379,45
304	frozenset({265622715;268790069})	0.000014	frozenset({268790069}) => frozenset({265622715})	0.207	126,17
305	frozenset({272188933;271842798})	0.000014	frozenset({272188933}) => frozenset({271842798})	0.207	1231,45
306	frozenset({273309576;274518714})	0.000024	frozenset({274518714}) => frozenset({273309576})	0.204	149,73
307	frozenset({255733794;255733797})	0.000031	frozenset({255733794}) => frozenset({255733797})	0.203	1341,23
308	frozenset({255922044;255935181})	0.000040	frozenset({255922044}) => frozenset({255935181})	0.202	718,69
309	frozenset({250208729;250654478})	0.000014	frozenset({250208729}) => frozenset({250654478})	0.2	59,19