1150 NEWS – FP GROWTH

The Dataset

The dataset comprises a total of 1150 news articles distributed across five distinct news categories. Each category consists of 230 articles, including economy, entertainment, health, politics, and sports news. There are 39,700 features representing the content of the articles.

- Number of classes in the dataset: 5
- Number of samples in the dataset: 1150
- Average word count in the texts: 204

The dataset is structured with 1150 rows and 39700 columns. Each row represents a specific news article, and each column denotes different features describing the content of the news articles. The last column of each row indicates the category to which the news article belongs, such as economy, entertainment, health, politics, or sports.

```python
data=pd.read_csv("1150haber.arff.csv")
data
```

Python

	oz0	oz1	oz2	oz3	oz4	oz5	oz6	oz7	oz8	oz9	...	oz39690	oz39691	oz39692	oz39693	oz39694	oz39695	oz39696	oz39697	oz39698	oz39699
0	28	342	2376	19	329	0.082	0.144	0.085	1.040	0	...	0	0	0	0	0	0	0	0	0	ekonomi
1	4	76	546	4	83	0.053	0.139	0.048	0.916	0	...	0	0	0	0	0	0	0	0	0	ekonomi
2	4	138	947	2	138	0.029	0.146	0.029	1.000	0	...	0	0	0	0	0	0	0	0	0	ekonomi
3	12	176	1152	10	153	0.068	0.153	0.078	1.150	0	...	0	0	0	0	0	0	0	0	0	ekonomi
4	43	718	5011	38	691	0.060	0.143	0.062	1.039	0	...	0	0	0	0	0	0	0	0	0	ekonomi
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1145	19	195	1224	12	156	0.097	0.159	0.122	1.250	0	...	0	0	0	0	0	0	0	0	0	spor
1146	85	695	4470	31	648	0.122	0.155	0.131	1.073	13	...	0	0	0	0	0	0	0	0	0	spor
1147	26	355	2257	23	285	0.073	0.157	0.091	1.246	0	...	0	0	0	0	0	0	0	0	0	spor
1148	31	337	2103	23	291	0.092	0.160	0.107	1.158	0	...	0	0	0	0	0	0	0	0	0	spor
1149	35	289	1927	23	272	0.121	0.150	0.129	1.063	3	...	0	0	0	0	0	0	0	0	1	spor

150 rows × 39700 columns

Click here to ask Blackbox to help you code faster

```python
print(data.head())
```

✓ 0.0s

```
   soz0  oz1   oz2  oz3  oz4    oz5    oz6    oz7    oz8  oz9  ...  oz39690  \
0    28  342  2376   19  329  0.082  0.144  0.085  1.040    0  ...        0
1     4   76   546    4   83  0.053  0.139  0.048  0.916    0  ...        0
2     4  138   947    2  138  0.029  0.146  0.029  1.000    0  ...        0
3    12  176  1152   10  153  0.068  0.153  0.078  1.150    0  ...        0
4    43  718  5011   38  691  0.060  0.143  0.062  1.039    0  ...        0

   oz39691  oz39692  oz39693  oz39694  oz39695  oz39696  oz39697  oz39698  \
0        0        0        0        0        0        0        0        0
1        0        0        0        0        0        0        0        0
2        0        0        0        0        0        0        0        0
3        0        0        0        0        0        0        0        0
4        0        0        0        0        0        0        0        0

   oz39699
0  ekonomi
1  ekonomi
2  ekonomi
3  ekonomi
4  ekonomi

[5 rows x 39700 columns]
```

Click here to ask Blackbox to help you code faster

```python
print(data.info())
```

✓ 0.9s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1150 entries, 0 to 1149
Columns: 39700 entries, soz0 to oz39699
dtypes: float64(4), int64(39695), object(1)
memory usage: 348.3+ MB
None
```

Using the FP-Growth algorithm, frequent relationships or recurring patterns within the news articles in this database will be identified.

FP-Growth Algoritms

FP-Growth is an algorithm used to find frequent patterns in association analysis. It operates more efficiently compared to some other algorithms, particularly offering advantages in handling large datasets. Developed to overcome certain challenges of the Apriori algorithm, FP-Growth maintains the entire database within a smaller and denser data structure known as the FP-Tree, which enhances its efficiency.

The FP-Growth algorithm scans the database only twice. The first scan calculates the support value for each item, while the second scan is dedicated to constructing the FP-Tree structure. This method eliminates the candidate generation step, providing a significant advantage for large databases.

The algorithm computes the support value for each item in the database and sorts the items based on their support values from high to low. Subsequently, it arranges the items within each transaction in accordance with their support values. To create the FP-Tree, it begins by creating a 'root' node and then inserts each transaction sequence into the FP-Tree structure. If an item from the dataset isn't in the tree, a new node is generated for it with a support value of 1. Support values are maintained alongside the items. If an item already exists, only the support value of that node is incremented by 1.

Once the FP-Growth algorithm constructs the tree, it starts processing from the item with the least frequency. For each item, it determines the paths where the item occurs, and its support value is set as the total observed value along those paths. These paths form the conditional pattern base for the item.

For each conditional pattern base, a conditional pattern tree is created, and the algorithm operates recursively. This process is repeated for each item, resulting in the identification of the frequent itemsets. This method employs a divide-and-conquer approach, dividing the main task into subtasks, thereby ensuring the algorithm's efficiency and speed.

Transaction ID	Items
T1	$\{E, K, M, N, O, Y\}$
T2	$\{D, E, K, N, \mathbf{O}, Y\}$
T3	$\{A, E, K, M\}$
T4	$\{C, K, M, U, Y\}$
T5	$\{C, E, I, K, O, O\}$

Item	Frequency
A	1
C	2
D	1
E	4
I	1
K	5
M	3
N	2
0	3
U	1
Y	3

After insertion of the relevant items, the set L looks like this :

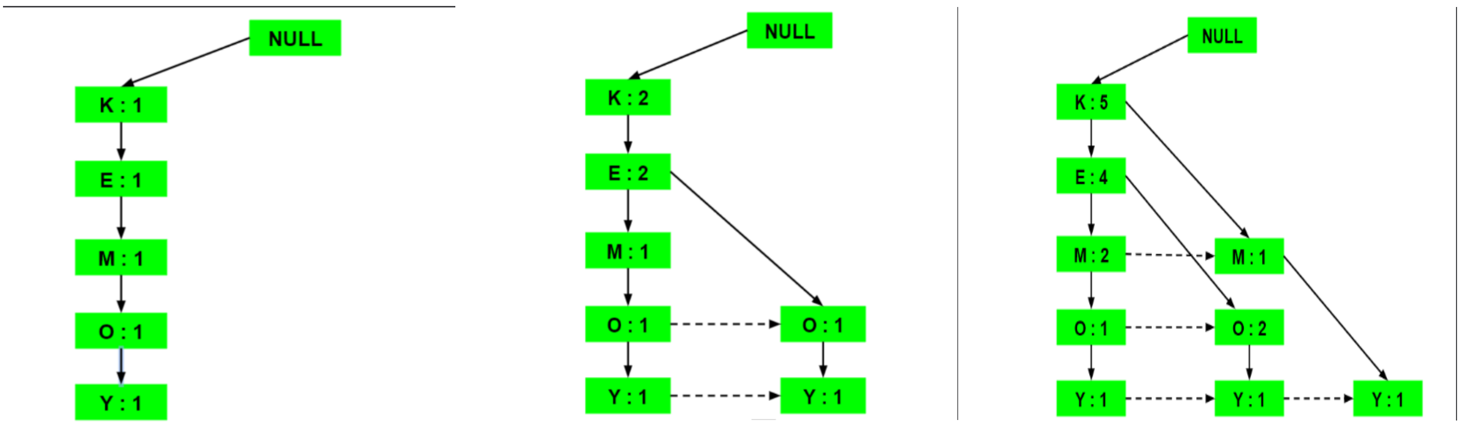
L = {K : 5, E : 4, M : 3, O : 3, Y : 3}

If the current item is contained, the item is inserted in the Ordered-Item set for the current transaction. The following table is built for all the transactions:

Transaction ID	Items	Ordered-Item Set
T1	$\{E, K, M, N, O, Y\}$	$\{K, E, M, O, Y\}$
T2	$\{D, E, K, N, O, Y\}$	$\{K, E, O, Y\}$
T3	$\{ A, E, K, M\}$	$\{K, E, M\}$
T4	$\{C, K, M, U, Y\}$	$\{K, M, Y\}$
T5	$\{C, E, I, K, O, O\}$	$\{K, E, O\}$

All the items are simply linked one after the other in the order of occurrence in the set and initialize the support count for each item as 1.

As the same element is added, the number of supports increases by 1.



For each item, the Conditional Frequent Pattern Tree is built. It is done by taking the set of elements that is common in all the paths in the Conditional Pattern Base of that item and calculating its support count by summing the support counts of all the paths in the Conditional Pattern Base.

Items	Conditional Pattern Base	Conditional Frequent Pattern Tree
Y	{{K,E,M,O : 1}, {K,E,O : 1}, {K,M : 1}}	{K : 3}
O	{{K,E,M : 1}, {K,E : 2}}	{K,E : 3}
M	{{K,E : 2}, {K : 1}}	{K : 3}
E	{K : 4}	{K : 4}
K		

Assume that the operations are carried out in the following order:

T1: A B C

T2: A B C E F

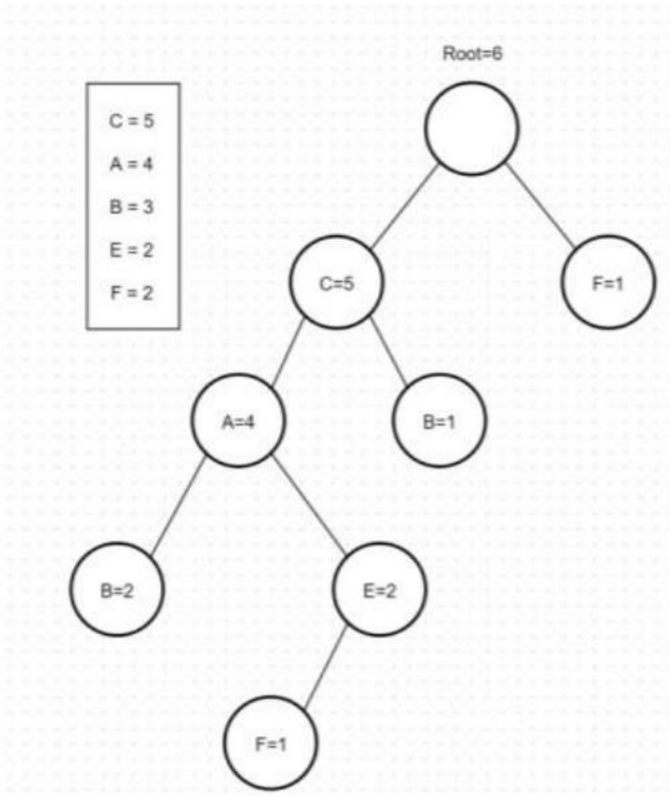
T3: D F

T4: A B C

T5: A C E G

T6: B C

Based on the size of the product support value in each transaction, the product is placed as shown in the rectangular box in the figure and the frequent pattern tree (Fp-Tree) proceeds based on this.



Advantages of the Fp-Growth Algorithm:

- Faster than the Apriori Algorithm
- Passes through the data in the database only twice
- No candidate cluster generation

Disadvantages of the Fp-Growth algorithm:

- Fp-Tree may not fit in memory
- Fp-Tree is costly to build

Usage Areas of FP-Growth Algorithm:

- Market Basket Analysis
- Data mining
- Recommendation systems
- Web traffic analysis
- It is frequently used in bioinformatics and many other fields.

In a nutshell,

The FP-Growth algorithm is an effective method that utilizes the FP-Tree structure to extract patterns that occur more frequently than a specified support threshold within a dataset. By generating Frequent Itemsets from data samples, it identifies commonly occurring combinations of features. The FP-Tree structure builds a tree-like representation containing frequent elements in the dataset, summarizing the patterns and detecting frequent patterns through its Conditional Pattern Base, a set of sub-trees. The resulting frequent patterns are filtered based on a particular support value, highlighting significant features or combinations within the dataset. The FP-Growth algorithm aids in discovering valuable insights in datasets, particularly in fields such as data mining, recommendation systems, and market basket analysis.

SPOR

[1]:
```
import pandas as pd
```

[2]:
```
data=pd.read_csv("1150haber.csv")
data
```

[2]:

	oz0	oz1	oz2	oz3	oz4	oz5	oz6	oz7	oz8	oz9	...	oz39690	oz39691	oz39692	oz39693	oz39694	oz39695	oz39696	o
0	28	342	2376	19	329	0.082	0.144	0.085	1.040	0	...	0	0	0	0	0	0	0	
1	4	76	546	4	83	0.053	0.139	0.048	0.916	0	...	0	0	0	0	0	0	0	
2	12	176	1152	10	153	0.068	0.153	0.078	1.150	0	...	0	0	0	0	0	0	0	
3	22	194	1273	8	155	0.113	0.152	0.142	1.252	0	...	0	0	0	0	0	0	0	
4	6	55	334	1	40	0.109	0.165	0.150	1.375	0	...	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1145	7	74	412	4	47	0.095	0.180	0.149	1.574	0	...	0	0	0	0	0	0	0	
1146	29	348	2297	21	313	0.083	0.152	0.093	1.112	0	...	0	0	0	0	0	0	0	
1147	47	466	2885	37	400	0.101	0.162	0.118	1.165	0	...	0	0	0	0	0	0	0	
1148	79	709	4540	63	599	0.111	0.156	0.132	1.184	0	...	0	0	0	0	0	0	0	
1149	5	123	766	3	38	0.041	0.161	0.132	3.237	0	...	0	0	0	0	0	0	0	

1150 rows × 39700 columns

It loads the Pandas library into the code. Reads a CSV file and loads its contents into a DataFrame named 'data'.

[3]:
```
spor=data[data["oz39699"]=="spor"]
spor
```

[3]:

	oz0	oz1	oz2	oz3	oz4	oz5	oz6	oz7	oz8	oz9	...	oz39690	oz39691	oz39692	oz39693	oz39694	oz39695	oz39696	c
608	27	248	1630	19	213	0.109	0.152	0.127	1.164	0	...	0	0	0	0	0	0	0	
609	26	252	1540	17	202	0.103	0.164	0.129	1.248	0	...	0	0	0	0	0	0	0	
610	7	84	462	7	51	0.083	0.182	0.137	1.647	0	...	0	0	0	0	0	0	0	
611	65	691	4349	51	604	0.094	0.159	0.108	1.144	0	...	0	0	0	0	0	0	0	
612	33	331	2205	25	315	0.100	0.150	0.105	1.051	0	...	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1144	16	119	689	4	57	0.134	0.173	0.281	2.088	0	...	0	0	0	0	0	0	0	
1145	7	74	412	4	47	0.095	0.180	0.149	1.574	0	...	0	0	0	0	0	0	0	
1146	29	348	2297	21	313	0.083	0.152	0.093	1.112	0	...	0	0	0	0	0	0	0	
1147	47	466	2885	37	400	0.101	0.162	0.118	1.165	0	...	0	0	0	0	0	0	0	
1148	79	709	4540	63	599	0.111	0.156	0.132	1.184	0	...	0	0	0	0	0	0	0	

230 rows × 39700 columns

It selects the rows in the Data Frame 'data' where the column "oz39699" has the value "sport" and assigns these rows to a DataFrame variable named 'sports.

[5]:
```
spor.shape
```

[5]: (230, 39700)

[6]:
```
spor = spor.drop(spor.columns[:34], axis=1)
```

[7]:
```
spor = spor.drop(spor.columns[8206:39698], axis=1)
```

[8]:
```
spor = spor.loc[:, (spor != 0).any(axis=0)]

spor.shape
```

[8]: (230, 4054)

The 5. the line shows the size of the sports data we have.

The 6. and 7. deletes unnecessary data on December lines from 0-33 to 8206-39698.

The 6. it checks each element in the row and determines which ones are different from zero. Then, after deleting unnecessary data, it shows the data size.

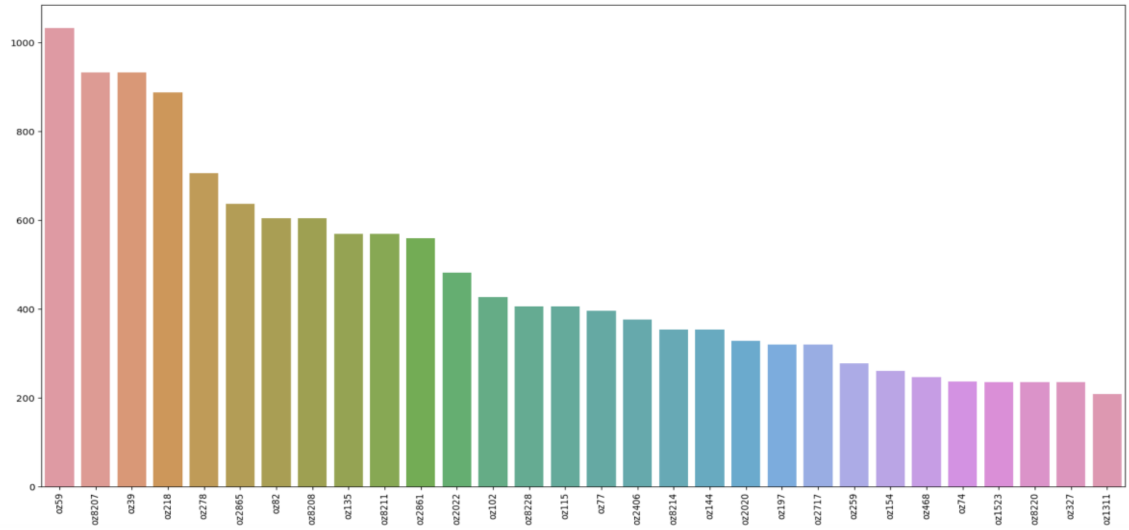
[9]:

	oz34	oz35	oz36	oz39	oz41	oz42	oz43	oz44	oz45	oz46	...	oz8230	oz8231	oz8232	oz8233	oz8234	oz8235	oz8236	oz823
608	0	0	0	6	0	0	0	0	0	1	...	0	2	1	0	0	1	0	
609	0	0	0	3	0	2	0	0	0	0	...	2	2	0	0	0	0	0	
610	0	0	0	3	0	0	0	0	0	0	...	0	0	1	0	0	0	0	
611	0	1	1	8	0	0	0	0	0	0	...	4	1	5	9	2	5	0	
612	0	0	0	4	0	0	0	1	0	0	...	0	0	4	0	1	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1144	3	0	0	1	0	0	0	0	1	0	...	0	0	0	0	0	0	0	
1145	0	0	0	2	0	0	0	0	0	0	...	0	0	0	1	0	0	0	
1146	0	0	0	8	0	0	0	0	1	0	...	1	0	2	1	0	0	0	
1147	0	0	0	5	0	0	0	0	0	0	...	2	1	4	4	0	1	2	
1148	0	0	0	16	0	0	0	0	0	1	...	3	3	2	1	0	0	1	

The view after the data has been cleaned.

[10]:

```python
import matplotlib.pyplot as plt
import seaborn as sns
r=spor.sum(axis=0).sort_values(ascending=False)[:30]
plt.figure(figsize=(20,10))
s=sns.barplot(x=r.index,y=r.values)
s.set_xticklabels(s.get_xticklabels(), rotation=90)
plt.show()
```



It takes the total values of the columns in the DATAFRAME, sorts these totals from large to small, and then visualizes the columns containing the largest 30 total values.

[11]:

```python
#spor[spor != 0] = 1

spor = spor.astype(bool)
spor.head()
```

[11]:

	oz34	oz35	oz36	oz39	oz41	oz42	oz43	oz44	oz45	oz46	...	oz8230	oz8231	oz8232	oz8233	oz8234	oz8235	oz8236	oz823
608	False	False	False	True	False	False	False	False	False	True	...	False	True	True	False	False	True	False	Fals
609	False	False	False	True	False	True	False	False	False	False	...	True	True	False	False	False	False	False	Fals
610	False	False	False	True	False	False	False	False	False	False	...	False	False	True	False	False	False	False	Fals
611	False	True	True	True	False	False	False	False	False	False	...	True	True	True	True	True	True	False	Tru
612	False	False	False	True	False	False	False	True	False	False	...	False	False	True	False	True	False	False	Tru

5 rows × 4054 columns

Converts the values in the table to boolean values.

[13]:

```python
from mlxtend.frequent_patterns import fpgrowth
dv = fpgrowth(spor, min_support=0.8, max_len=2,use_colnames=True)
dv
```

[13]:

	support	itemsets
0	0.908696	(oz8207)
1	0.908696	(oz39)
2	0.856522	(oz59)
3	0.908696	(oz8207, oz39)

[14]:

```python
# dv değeri item değerinin yada {oz39, oz59} kaç defa göründüğü/Satır sayısı
dv.loc[:20,:]
```

[14]:

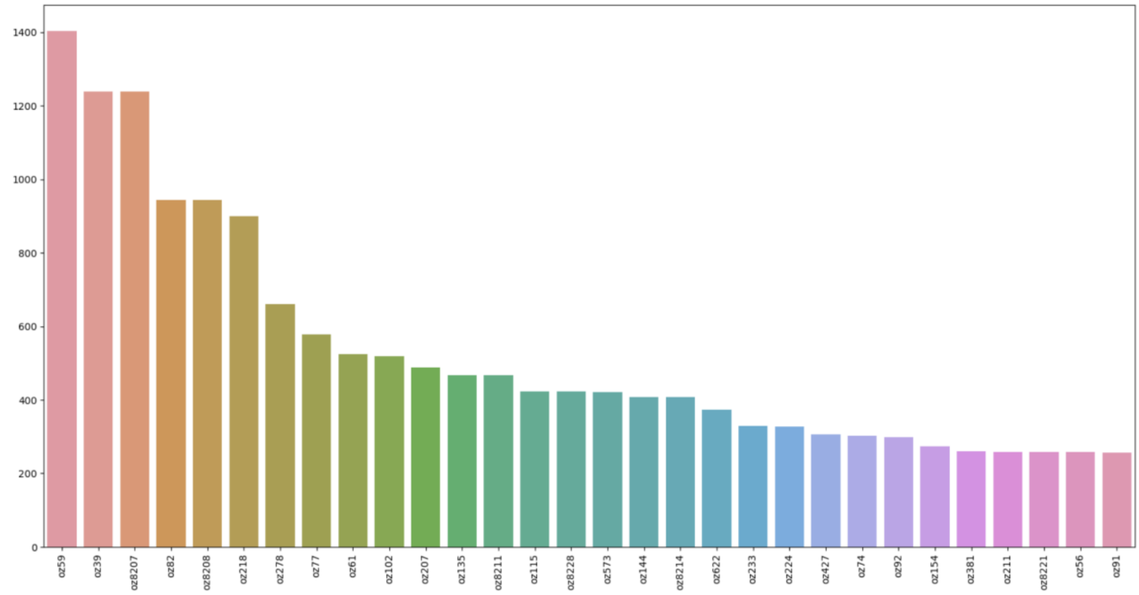
	support	itemsets
0	0.908696	(oz8207)
1	0.908696	(oz39)
2	0.856522	(oz59)
3	0.908696	(oz8207, oz39)

'support': A support value that indicates how often a particular pattern appears in the dataset.

'itemsets': A list of items that represent patterns.

ECONOMY

```
•[10]: import matplotlib.pyplot as plt
import seaborn as sns
r=ekonomi.sum(axis=0).sort_values(ascending=False)[:30]
plt.figure(figsize=(20,10))
s=sns.barplot(x=r.index,y=r.values)
s.set_xticklabels(s.get_xticklabels(), rotation=90)
plt.show()
```



DataFrame named "ekonomi", sorts these values in descending order, and selects the top 30 most frequent features.

```
[13]: from mlxtend.frequent_patterns import fpgrowth
dv = fpgrowth(ekonomi, min_support=0.4, max_len=1,use_colnames=True)
dv
```

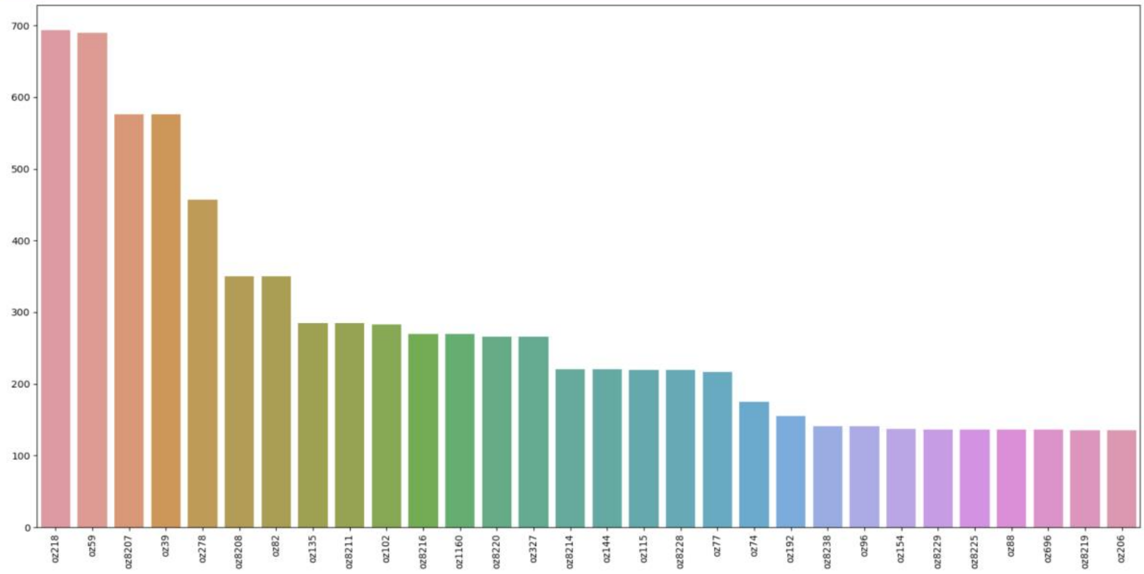
[13]:

	support	itemsets
0	0.943478	(oz59)
1	0.939130	(oz39)
2	0.939130	(oz8207)
3	0.791304	(oz82)
4	0.791304	(oz8208)
5	0.739130	(oz77)
6	0.739130	(oz102)
7	0.721739	(oz135)
8	0.721739	(oz8211)
9	0.695652	(oz115)
10	0.695652	(oz8228)
11	0.643478	(oz8214)
12	0.643478	(oz144)
13	0.560870	(oz74)
14	0.539130	(oz61)
15	0.517391	(oz95)
16	0.495652	(oz162)
17	0.495652	(oz154)
18	0.495652	(oz142)
18	0.495652	(oz142)
19	0.491304	(oz56)
20	0.469565	(oz130)
21	0.456522	(oz165)
22	0.456522	(oz8234)
23	0.447826	(oz127)
24	0.413043	(oz96)
25	0.413043	(oz8238)
26	0.791304	(oz218)
27	0.517391	(oz211)
28	0.517391	(oz8221)
29	0.473913	(oz207)
30	0.439130	(oz192)
31	0.756522	(oz278)
32	0.434783	(oz224)
33	0.465217	(oz468)
34	0.460870	(oz622)
35	0.421739	(oz263)
36	0.408696	(oz377)
37	0.408696	(oz8226)

It indicates that the patterns should have at least a 40% support level.

Magazine

```
[10]: import matplotlib.pyplot as plt
import seaborn as sns
r=magazin.sum(axis=0).sort_values(ascending=False)[:30]
plt.figure(figsize=(20,10))
s=sns.barplot(x=r.index,y=r.values)
s.set_xticklabels(s.get_xticklabels(), rotation=90)
plt.show()
```



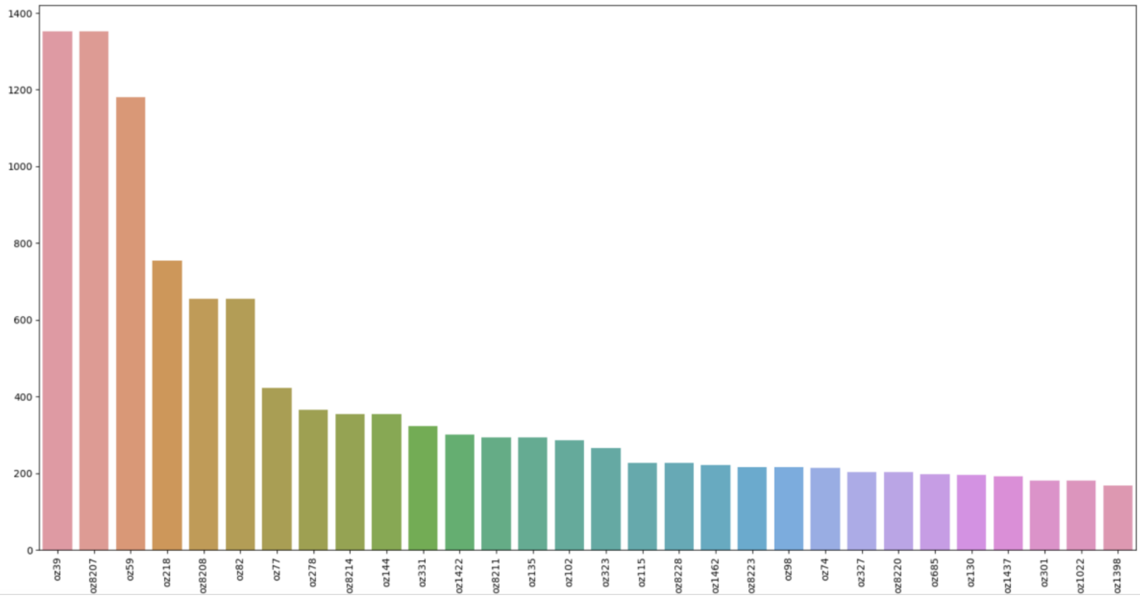

```
[13]: from mlxtend.frequent_patterns import fpgrowth
dv = fpgrowth(magazin, min_support=0.6, max_len=2,use_colnames=True)
dv
```

[13]:

	support	itemsets
0	0.826087	(oz59)
1	0.786957	(oz218)
2	0.734783	(oz8207)
3	0.734783	(oz39)
4	0.686957	(oz278)
5	0.669565	(oz218, oz59)
6	0.634783	(oz8207, oz59)
7	0.734783	(oz39, oz8207)
8	0.634783	(oz39, oz59)
9	0.604348	(oz278, oz59)

Health

[10]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
r=saglik.sum(axis=0).sort_values(ascending=False)[:30]
plt.figure(figsize=(20,10))
s=sns.barplot(x=r.index,y=r.values)
s.set_xticklabels(s.get_xticklabels(), rotation=90)
plt.show()
```



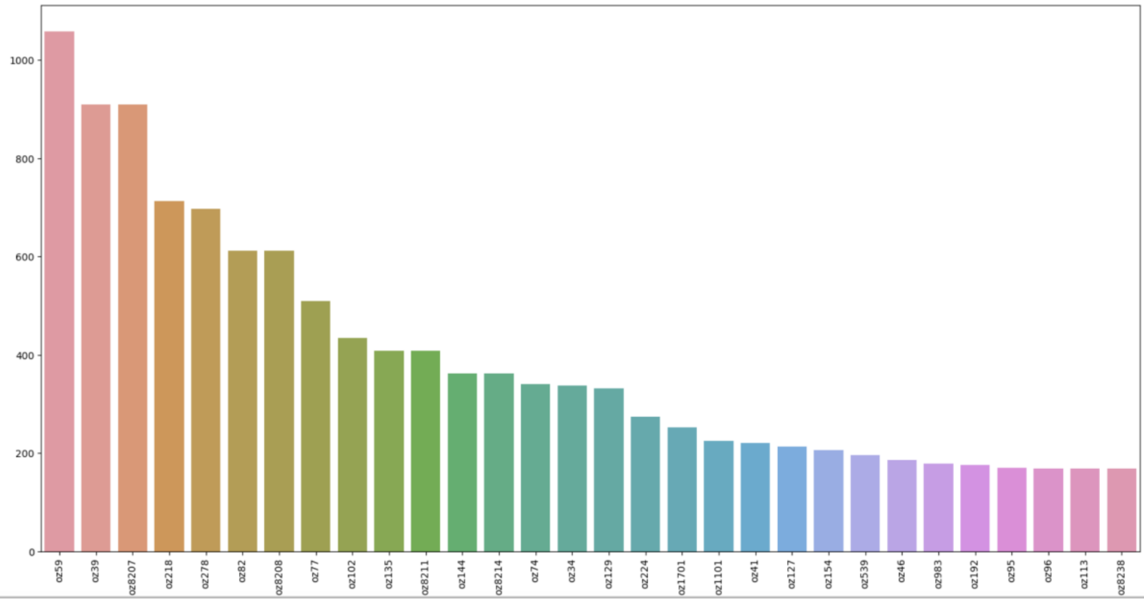
[13]:
```python
from mlxtend.frequent_patterns import fpgrowth
dv = fpgrowth(saglik, min_support=0.6, max_len=1,use_colnames=True)
dv
```

[13]:

	support	itemsets
0	0.969565	(oz8207)
1	0.969565	(oz39)
2	0.921739	(oz59)
3	0.830435	(oz218)
4	0.756522	(oz8208)
5	0.756522	(oz82)
6	0.708696	(oz77)
7	0.639130	(oz278)
8	0.634783	(oz8214)
9	0.634783	(oz144)

<u>**Political**</u>

[10]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
r=siyasi.sum(axis=0).sort_values(ascending=False)[:30]
plt.figure(figsize=(20,10))
s=sns.barplot(x=r.index,y=r.values)
s.set_xticklabels(s.get_xticklabels(), rotation=90)
plt.show()
```



[13]:
```python
from mlxtend.frequent_patterns import fpgrowth
dv = fpgrowth(siyasi, min_support=0.7, max_len=2,use_colnames=True)
dv
```

[13]:

	support	itemsets
0	0.908696	(oz59)
1	0.891304	(oz8207)
2	0.891304	(oz39)
3	0.839130	(oz278)
4	0.813043	(oz218)
5	0.747826	(oz8208)
6	0.747826	(oz82)
7	0.743478	(oz77)
8	0.730435	(oz102)
9	0.704348	(oz8211)
10	0.704348	(oz135)
11	0.817391	(oz59, oz8207)
12	0.891304	(oz39, oz8207)
13	0.817391	(oz39, oz59)
14	0.782609	(oz59, oz278)
15	0.765217	(oz39, oz278)
16	0.765217	(oz278, oz8207)
17	0.760870	(oz59, oz218)
18	0.739130	(oz39, oz218)
19	0.739130	(oz218, oz8207)
20	0.717391	(oz278, oz218)
21	0.704348	(oz59, oz8208)
22	0.747826	(oz82, oz8208)
23	0.704348	(oz82, oz59)
24	0.704348	(oz135, oz8211)

REFERENCES

- https://www.geeksforgeeks.org/frequent-pattern-growth-algorithm/
- https://medium.com/@anilcogalan/fp-growth-algorithm-how-to-analyze-user-behavior-and-outrank-your-competitors-c39af08879db
- https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/fpgrowth/
- https://www.kaggle.com/code/rjmanoj/fp-growth-algorithm-frequent-itemset-patternhttps://www.kaggle.com/code/rjmanoj/fp-growth-algorithm-frequent-itemset-pattern
- https://github.com/topics/fp-growth-algorithm
- https://towardsdatascience.com/fp-growth-frequent-pattern-generation-in-data-mining-with-python-implementation-244e561ab1c3
- https://thinkingneuron.com/how-to-do-association-rule-mining-using-fp-growth-in-python/
- https://www.sonarsource.com/lp/knowledge/languages/python/?gads_campaign=SQ-Mroi-Generic&gads_ad_group=Python&gads_keyword=python%20analysis&cq_src=google_ads&cq_cmp=19265130410&cq_con=155008917058&cq_term=python%20analysis&cq_med=&cq_plac=&cq_net=g&cq_pos=&cq_plt=gp&gad_source=1&gclid=CjwKCAiAnL-sBhBnEiwAJRGighKEiMxtEvZ79mCAKNsFW2TQneRxQf65XgOkDDpnNji_q8EceRYpTxoC2D0QAvD_BwE
- http://www.kemik.yildiz.edu.tr/veri_kumelerimiz.html

SİMGE LALE – 2018556050

KEVSER KOÇ – 2018556048