МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА - Московский технологический университет»

РТУ МИРЭА

Институт Информационных Технологий
Кафедра Прикладной Математики (ПМ)

ПРАКТИЧЕСКАЯ РАБОТА № 13

Выполнил студент группы ИКБО-08-19

Борисов А.В. (_____)

подпись

Принял Ассистент кафедры ПМ

Высоцкая А.А. (_____)

подпись

Практическая работа выполнена «____» _____2022 г.

«Зачтено» «____» _____2022 г.

Москва 2022

1. *Загрузить данные Market_Basket_Optimisation.csv.*

```python
df = pd.read_csv('Market_Basket_Optimisation.csv')
```

Python

2. *Визуализировать данные (отразить на гистограммах относительную и фактическую частоту встречаемости для 20 наиболее популярных товаров).*

```python
df.stack().value_counts(normalize=True).head(20).plot(kind='bar')
```

Python

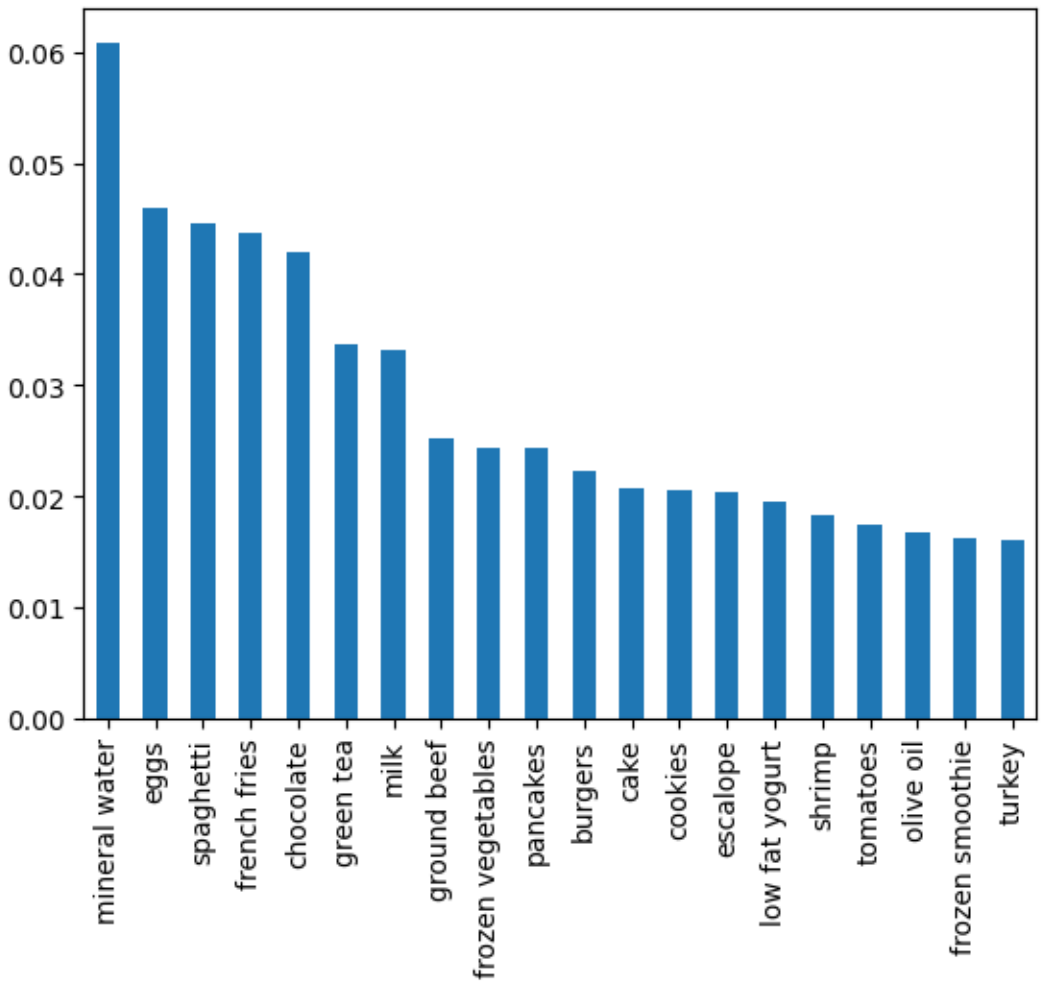

Рисунок 1 – 20 самых покупаемых товаров в датасете

3. *Применить алгоритм Apriori, используя 3 разные библиотеки (apriori_python, apyori, efficient_apriori).*

Apriori_python:

```python
from apriori_python import apriori
```

Python

```python
t = []
start = time.perf_counter()

t1, rules = apriori(transactions, minSup=0.03, minConf=0.3)

time1 = (time.perf_counter() - start)

t.append(time1)
```

Python

```python
rules
```

Python

```
[[{'chocolate'}, {'mineral water'}, 0.3213995117982099],
 [{'spaghetti'}, {'mineral water'}, 0.3430321592649311],
 [{'pancakes'}, {'mineral water'}, 0.3548387096774194],
 [{'milk'}, {'mineral water'}, 0.37037037037037035],
 [{'frozen vegetables'}, {'mineral water'}, 0.3748251748251748],
 [{'ground beef'}, {'spaghetti'}, 0.3989145183175034],
 [{'ground beef'}, {'mineral water'}, 0.41655359565807326]]
```

Apyori:

```python
from apyori import apriori
```

Python

```python
start = time.perf_counter()

rules = apriori(transactions=transactions, min_support=0.03,
                min_confidence=0.3, min_lift=1.0001)

results = list(rules)

time2 = (time.perf_counter() - start)

t.append(time2)
```

Python

```python
results
```

```
[RelationRecord(items=frozenset({'chocolate', 'mineral water'}), support=0.05266666666666667,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'chocolate'}), items_add=frozenset({'mineral
water'}), confidence=0.32139951179820997, lift=1.3489067367020564)]),
 RelationRecord(items=frozenset({'frozen vegetables', 'mineral water'}), support=0.03573333333333333,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'frozen vegetables'}),
items_add=frozenset({'mineral water'}), confidence=0.3748251748251748, lift=1.5731330784492508)]),
 RelationRecord(items=frozenset({'ground beef', 'mineral water'}), support=0.040933333333333335,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'ground beef'}), items_add=frozenset({'mineral
water'}), confidence=0.41655359565807326, lift=1.7482663499919135)]),
 RelationRecord(items=frozenset({'ground beef', 'spaghetti'}), support=0.0392, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'ground beef'}), items_add=frozenset({'spaghetti'}),
confidence=0.39891451831750335, lift=2.2908567284695827)]),
 RelationRecord(items=frozenset({'milk', 'mineral water'}), support=0.048, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'milk'}), items_add=frozenset({'mineral water'}),
confidence=0.3703703703703704, lift=1.5544363613753656)]),
 RelationRecord(items=frozenset({'pancakes', 'mineral water'}), support=0.03373333333333333,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'pancakes'}), items_add=frozenset({'mineral
water'}), confidence=0.3548387096774194, lift=1.489250320414463)]),
 RelationRecord(items=frozenset({'mineral water', 'spaghetti'}), support=0.05973333333333333,
ordered_statistics=[OrderedStatistic(items_base=frozenset({'spaghetti'}), items_add=frozenset({'mineral
water'}), confidence=0.3430321592649311, lift=1.4396984860027886)])]
```

Efficient_apriori:

```python
from efficient_apriori import apriori
```

```python
start = time.perf_counter()

itemsets, rules = apriori(transactions, min_support=0.03, min_confidence=0.3)

time3 = (time.perf_counter() - start)

t.append(time3)
```

```python
for i in range(len(rules)):
    print(rules[i])
```

```
{chocolate} -> {mineral water} (conf: 0.321, supp: 0.053, lift: 1.349, conv: 1.123)
{frozen vegetables} -> {mineral water} (conf: 0.375, supp: 0.036, lift: 1.573, conv: 1.218)
{ground beef} -> {mineral water} (conf: 0.417, supp: 0.041, lift: 1.748, conv: 1.306)
{ground beef} -> {spaghetti} (conf: 0.399, supp: 0.039, lift: 2.291, conv: 1.374)
{milk} -> {mineral water} (conf: 0.370, supp: 0.048, lift: 1.554, conv: 1.210)
{pancakes} -> {mineral water} (conf: 0.355, supp: 0.034, lift: 1.489, conv: 1.181)
{spaghetti} -> {mineral water} (conf: 0.343, supp: 0.060, lift: 1.440, conv: 1.159)
```

Параметры min support и min confidence были подобраны опытным путем, для получения оптимального количества правил. Во многом все три

библиотеки дали очень схожие результаты при одинаковых параметрах (алгоритм ведь используется один и тот же), однако количество пар ассоциаций варьируется. Наибольшее количество результатов дал ipyori.

4. *Применить алгоритм FP-Growth из библиотеки fpgrowth_py.*

```python
from fpgrowth_py import fpgrowth
```

Python

```python
start = time.perf_counter()

itemsets, rules = fpgrowth(transactions, minSupRatio=0.03, minConf=0.3)

time4 = (time.perf_counter() - start)

t.append(time4)
```

Python

```python
for i in range(len(rules)):
    print(rules[i])
```

Python

```
[{'pancakes'}, {'mineral water'}, 0.3548387096774194]
[{'frozen vegetables'}, {'mineral water'}, 0.3748251748251748]
[{'ground beef'}, {'spaghetti'}, 0.3989145183175034]
[{'ground beef'}, {'mineral water'}, 0.41655359565807326]
[{'milk'}, {'mineral water'}, 0.37037037037037035]
[{'chocolate'}, {'mineral water'}, 0.3213995117982099]
[{'spaghetti'}, {'mineral water'}, 0.3430321592649311]
```

При помощи FP-Growth так же был получен приемлемый результат. Параметры переданы те же, что и в предыдущем пункте.

В результате работы данных алгоритмов мы получили ассоциативные правила, которые можно извлечь из этого датасета. Большая часть из них пересекаются между решениями при помощи разных алгоритмов и библиотек, что может говорить о их верности.

5. *Сравнить время выполнения всех алгоритмов и построить гистограмму.*

```python
plt.bar(['apriori', 'apriori2', 'efficient_apriori', 'fpgrowth'], t)
```

Python

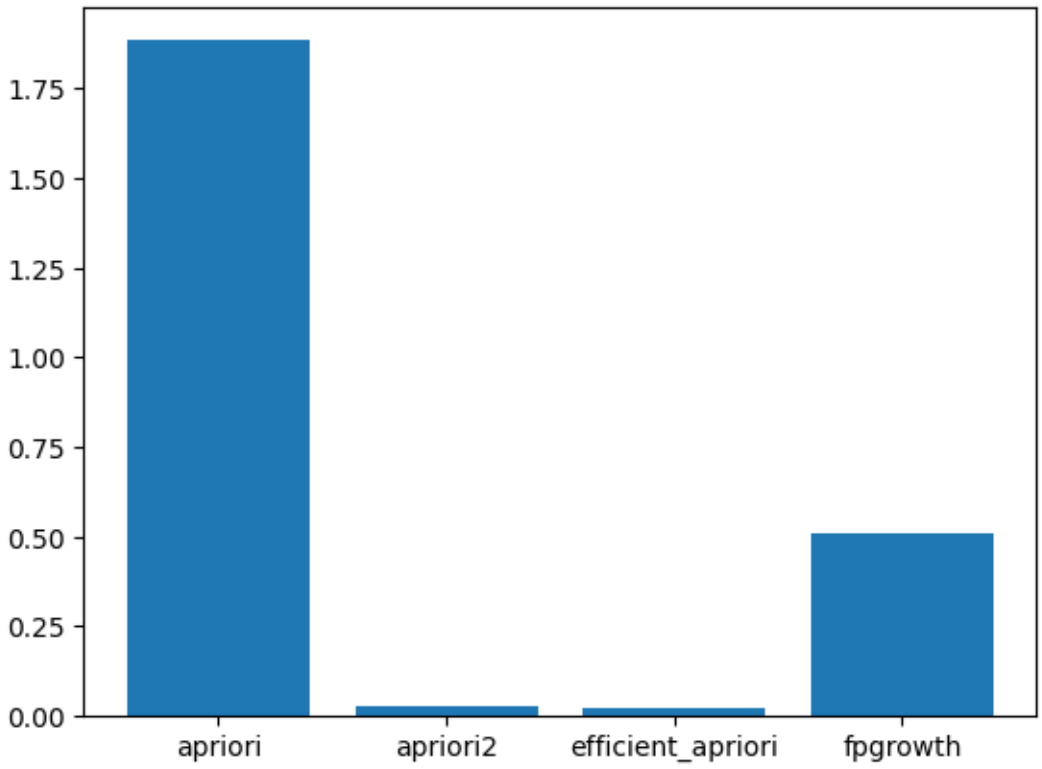

Рисунок 2 – гистограмма времени выполнения алгоритмов

Из результатов ясно видно, что дольше всего выполняется алгоритм из библиотеки apriori_python. Разница же между apyori и efficient_apriori практически незаметна (в районе нескольких тысячных секунды).

6. *Загрузить данные data.csv.*

```python
df = pd.read_csv('data.csv')
```

7. *Визуализировать данные (отразить на гистограммах относительную и фактическую частоту встречаемости для 20 наиболее популярных товаров).*

```python
df.stack().value_counts(normalize=True).head(20).plot(kind='bar')
```

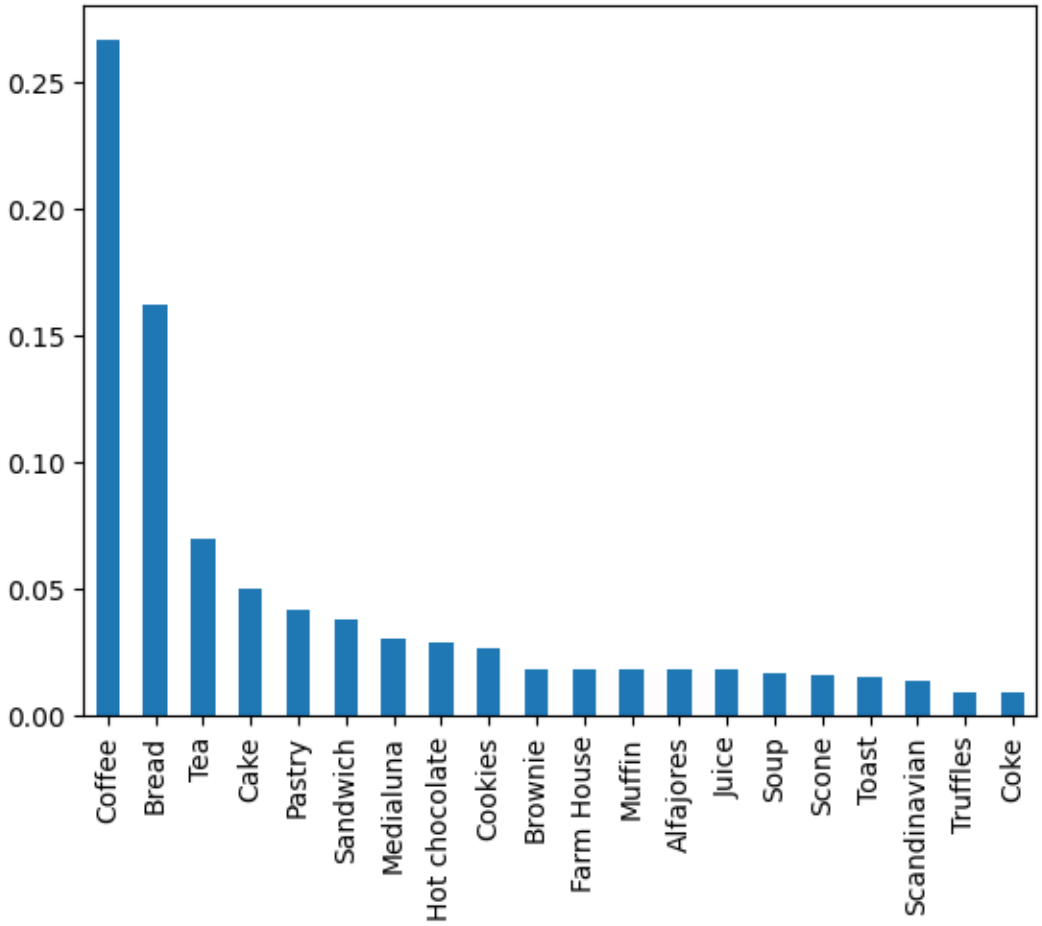

Рисунок 3 - 20 самых покупаемых товаров в датасете

8. *Применить алгоритм Apriori, используя 3 разные библиотеки (apriori_python, apyori, efficient_apriori).*

Apriori_python:

```
1  from apriori_python import apriori
```

✓ 0.4s Python

```
1  t = []
2  start = time.perf_counter()
3
4  t1, rules = apriori(transactions, minSup=0.03, minConf=0.1)
5
6  time1 = (time.perf_counter() - start)
7
8  t.append(time1)
```

✓ 0.5s Python

```
1  rules
```
✓ 0.9s Python

```
[[{'Coffee'}, {'Tea'}, 0.10424028268551237],
 [{'Coffee'}, {'Cake'}, 0.11439929328621908],
 [{'Coffee'}, {'Bread'}, 0.1881625441696113],
 [{'Bread'}, {'Coffee'}, 0.2751937984496124],
 [{'Tea'}, {'Coffee'}, 0.3496296296296296],
 [{'Cake'}, {'Coffee'}, 0.5269582909460834],
 [{'Sandwich'}, {'Coffee'}, 0.5323529411764706],
 [{'Pastry'}, {'Coffee'}, 0.5521472392638037],
 [{'Medialuna'}, {'Coffee'}, 0.5692307692307692]]
```

Apyori:

```
1  from apyori import apriori
```
✓ 0.5s Python

```
1  start = time.perf_counter()
2
3  rules = apriori(transactions=transactions, min_support=0.03,
4                  min_confidence=0.1, min_lift=1.0001)
5
6  results = list(rules)
7
8  time2 = (time.perf_counter() - start)
9
10 t.append(time2)
```
✓ 0.4s Python

```
1  results
```
✓ 0.2s Python

```
[RelationRecord(items=frozenset({'Cake', 'Coffee'}), support=0.05435466946484785, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'Cake'}), items_add=frozenset({'Coffee'}),
confidence=0.5269582909460834, lift=1.109079618532724), OrderedStatistic(items_base=frozenset({'Coffee'}),
items_add=frozenset({'Cake'}), confidence=0.11439929328621908, lift=1.109079618532724)]),
 RelationRecord(items=frozenset({'Medialuna', 'Coffee'}), support=0.034942287513116475, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'Medialuna'}), items_add=frozenset({'Coffee'}),
confidence=0.5692307692307693, lift=1.1980497417776572)]),
 RelationRecord(items=frozenset({'Pastry', 'Coffee'}), support=0.0472193074501574, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'Pastry'}), items_add=frozenset({'Coffee'}),
confidence=0.5521472392638037, lift=1.162094344121919)]),
 RelationRecord(items=frozenset({'Sandwich', 'Coffee'}), support=0.037985309548793283, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'Sandwich'}), items_add=frozenset({'Coffee'}),
confidence=0.5323529411764706, lift=1.1204336416545417)])]
```

Efficient_apriori:

```
1  from efficient_apriori import apriori
```
✓ 0.3s Python

```python
start = time.perf_counter()

itemsets, rules = apriori(transactions, min_support=0.03, min_confidence=0.1)

time3 = (time.perf_counter() - start)

t.append(time3)
```

✓ 0.3s Python

```python
for i in range(len(rules)):
    print(rules[i])
```

✓ 0.3s Python

```
{Coffee} -> {Bread} (conf: 0.188, supp: 0.089, lift: 0.579, conv: 0.832)
{Bread} -> {Coffee} (conf: 0.275, supp: 0.089, lift: 0.579, conv: 0.724)
{Coffee} -> {Cake} (conf: 0.114, supp: 0.054, lift: 1.109, conv: 1.013)
{Cake} -> {Coffee} (conf: 0.527, supp: 0.054, lift: 1.109, conv: 1.110)
{Medialuna} -> {Coffee} (conf: 0.569, supp: 0.035, lift: 1.198, conv: 1.218)
{Pastry} -> {Coffee} (conf: 0.552, supp: 0.047, lift: 1.162, conv: 1.172)
{Sandwich} -> {Coffee} (conf: 0.532, supp: 0.038, lift: 1.120, conv: 1.122)
{Tea} -> {Coffee} (conf: 0.350, supp: 0.050, lift: 0.736, conv: 0.807)
{Coffee} -> {Tea} (conf: 0.104, supp: 0.050, lift: 0.736, conv: 0.958)
```

Параметры min support и min confidence опять же были подобраны опытным путем, для получения оптимального количества правил. Снова все три библиотеки дали очень схожие результаты при одинаковых параметрах, однако количество пар ассоциаций варьируется. Наибольшее количество результатов опять дал ipyori.

9. *Применить алгоритм FP-Growth из библиотеки fpgrowth_py.*

```python
from fpgrowth_py import fpgrowth
```

✓ 0.4s Python

```python
start = time.perf_counter()

itemsets, rules = fpgrowth(transactions, minSupRatio=0.03, minConf=0.1)

time4 = (time.perf_counter() - start)

t.append(time4)
```

✓ 0.3s Python

```python
for i in range(len(rules)):
    print(rules[i])
```

✓ 0.2s Python

```
[{'Toast'}, {'Coffee'}, 0.7044025157232704]
[{'Cookies'}, {'Coffee'}, 0.5184466019417475]
[{'Hot chocolate'}, {'Coffee'}, 0.5072463768115942]
[{'Medialuna'}, {'Coffee'}, 0.5692307692307692]
[{'Sandwich'}, {'Coffee'}, 0.5323529411764706]
[{'Pastry'}, {'Bread'}, 0.33865030674846625]
[{'Pastry'}, {'Coffee'}, 0.5521472392638037]
[{'Cake'}, {'Coffee'}, 0.5269582909460834]
[{'Coffee'}, {'Cake'}, 0.11439929328621908]
[{'Tea'}, {'Bread'}, 0.19703703703703704]
[{'Tea'}, {'Coffee'}, 0.3496296296296296]
[{'Coffee'}, {'Tea'}, 0.10424028268551237]
[{'Coffee'}, {'Bread'}, 0.1881625441696113]
[{'Bread'}, {'Coffee'}, 0.2751937984496124]
```

При помощи FP-Growth так же снова был получен приемлемый результат. Параметры переданы те же, что и в предыдущем пункте.

Снова в результате работы алгоритмов мы получили ассоциативные правила, которые можно извлечь датасета. Опять же большая часть из них пересекаются между различными решениями.

10. *Сравнить время выполнения всех алгоритмов и построить гистограмму.*

```python
plt.bar(['apriori', 'apriori2', 'efficient_apriori', 'fpgrowth'], t);
```

✓ 0.2s Python

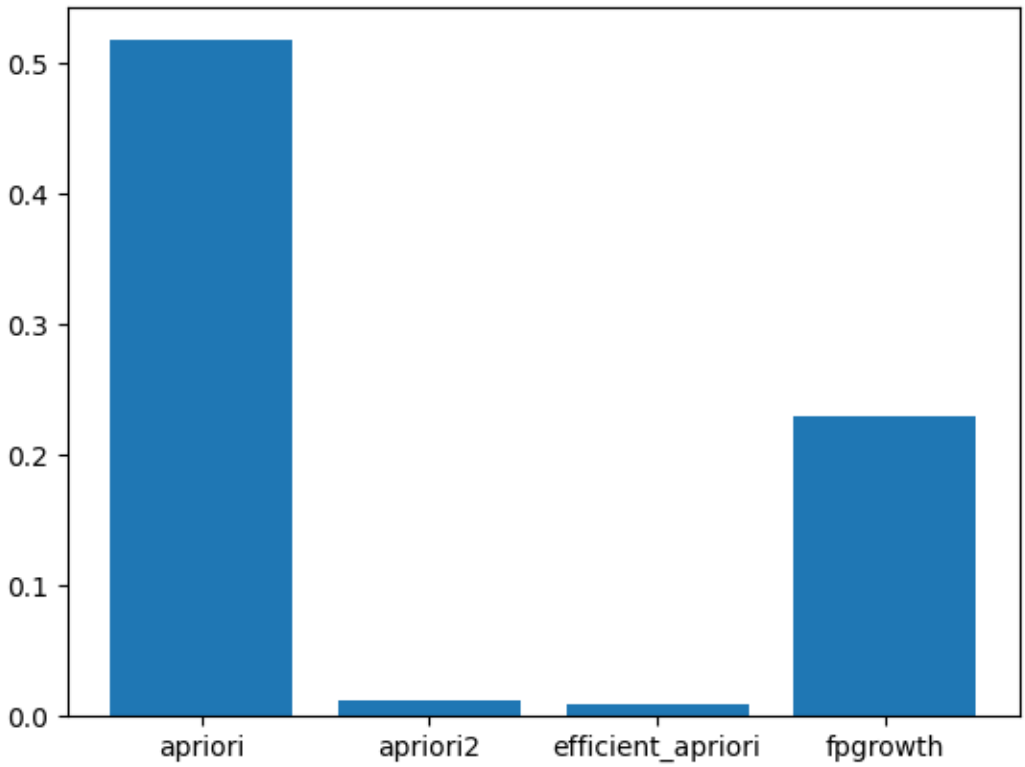

Рисунок 4 - гистограмма времени выполнения алгоритмов

Из результатов опять же видно, что дольше всего выполняется алгоритм из библиотеки apriori_python. Разница между apyori и efficient_apriori снова практически незаметна.

В целом, результаты работы с обоими датасетами дали схожие, в плане временных затрат и эффективности работы алгоритмов, результаты. Не смотря на то, что время работы первых трех алгоритмов по-идее должны быть очень похожи, на практике они имеют очень весомую разницу.

Алгоритм FP-Growth показал себя на более-менее среднем уровне (по временным затратам).