МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)

Кафедра прикладной математики

ОТЧЁТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 8
по дисциплине «Технологии и инструментарий анализа больших данных»

Выполнил студент группы ИКБО-20-21 Сидоров С.Д.

Проверил ассистент кафедры ПМ ИИТ Тетерин Н.Н.

Москва 2024

Практическая работа

1. Загрузить данные Market_Basket_Optimisation.csv.

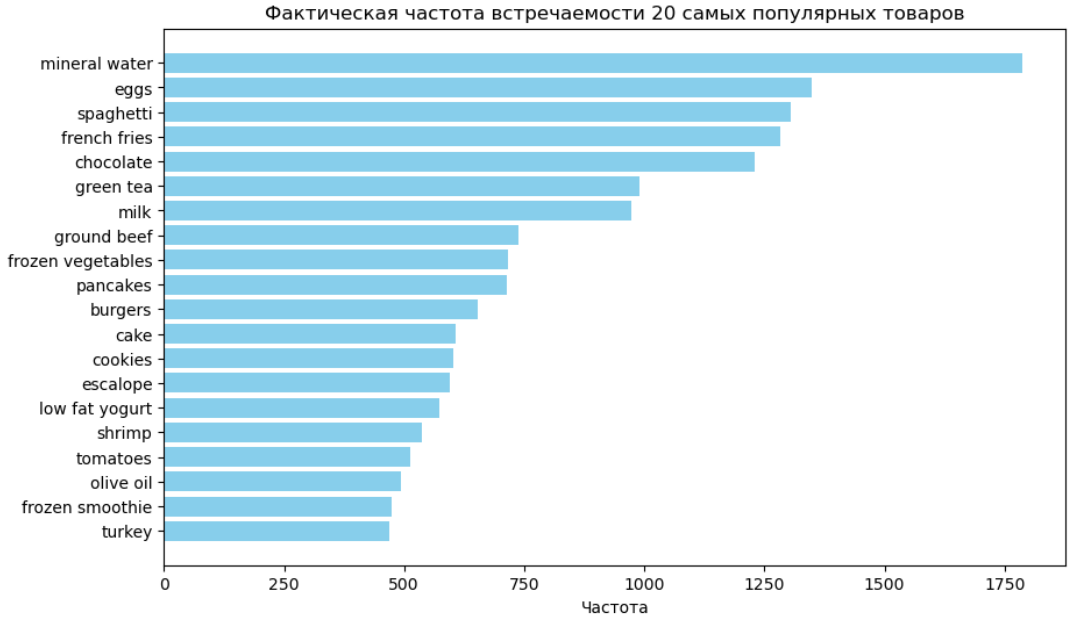
Листинг 1:

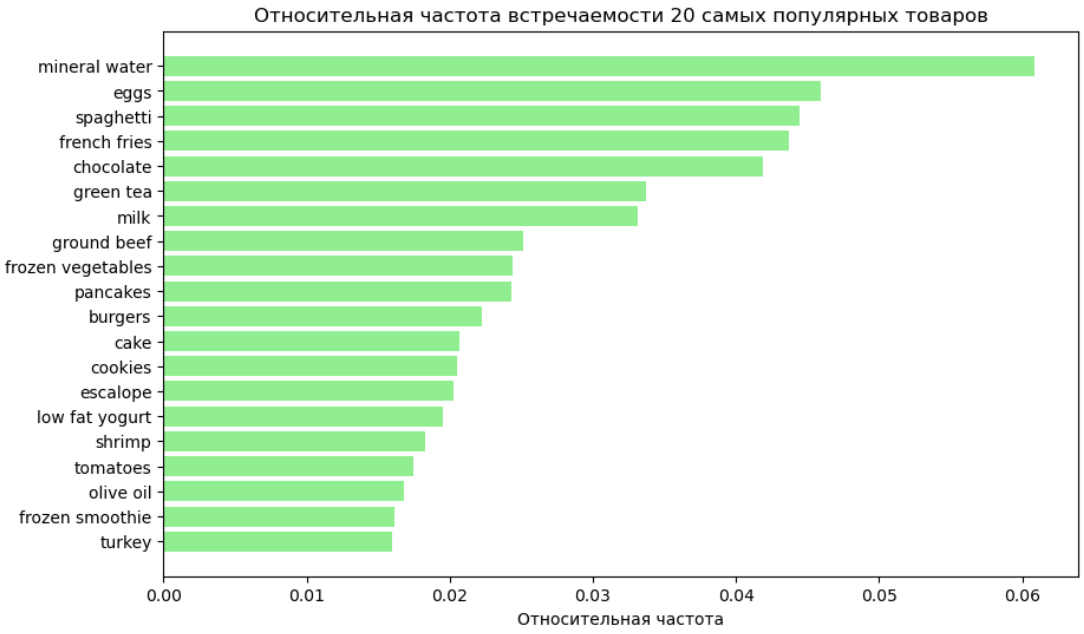
```
# Загрузка данных из файла Market_Basket_Optimisation.csv
file_path = 'Market_Basket_Optimisation.csv'
data = pd.read_csv(file_path, header=None)

# Отображение первых строк данных
data.head()
```

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

Листинг 2:

```
# Преобразование данных в список покупок
transactions = data.values.flatten()
transactions = [item for item in transactions if str(item) != 'nan']

# Подсчет количества вхождений каждого товара
item_counts = Counter(transactions)

# Получение 20 самых популярных товаров
most_common_items = item_counts.most_common(20)
items, counts = zip(*most_common_items)

# Построение гистограммы фактической частоты
plt.figure(figsize=(10, 6))
plt.barh(items, counts, color='skyblue')
plt.xlabel('Частота')
plt.title('Фактическая частота встречаемости 20 самых популярных
товаров')
plt.gca().invert_yaxis()  # Инвертировать ось Y для правильного
отображения
plt.show()

# Построение гистограммы относительной частоты
total_items = sum(item_counts.values())
relative_freq = np.array(counts) / total_items

plt.figure(figsize=(10, 6))
plt.barh(items, relative_freq, color='lightgreen')
plt.xlabel('Относительная частота')
plt.title('Относительная частота встречаемости 20 самых популярных
товаров')
plt.gca().invert_yaxis()  # Инвертировать ось Y для правильного
отображения
plt.show()
```





3. Применить алгоритм Apriori, используя 3 разные библиотеки (apriori_python, apyori, efficient_apriori). Подобрать гиперпараметры для алгоритмов так, чтобы выводилось порядка 10 наилучших правил.

Листинг 3:

```
# Преобразование данных в список транзакций без NaN
transactions = data.values.tolist()

# Убираем значения NaN, None, и float (оставляем только строки)
transactions = [[str(item) for item in transaction if isinstance(item,
str)] for transaction in transactions]

# Применение алгоритма Apriori из библиотеки apyori
rules_apyori = list(apriori(transactions, min_support=0.01,
```

```
min_confidence=0.2))
print("Применение алгоритма Apriori из библиотеки apyori")
# Вывод 10 наилучших правил
results_apyori = sorted(rules_apyori, key=lambda x: x[2],
reverse=True)[:10]
# for rule in results_apyori:
#     print(rule, '\n')
for result in results_apyori:
    for subset in result[2]:
        print(subset[0], subset[1])
        print("Support: {0}; Confidence: {1}; Lift:
{2};".format(result[1], subset[2],subset[3]))
        print()
# Применение алгоритма Apriori из библиотеки efficient_apriori
itemsets, rules_efficient = efficient_apriori(transactions,
min_support=0.01, min_confidence=0.2)

# Вывод 10 наилучших правил, отсортированных по уверенности
(confidence)
rules_sorted = sorted(rules_efficient, key=lambda rule:
rule.confidence, reverse=True)[:10]
print("Применение алгоритма Apriori из библиотеки efficient_apriori")
# Печать правил
if rules_sorted:
    for rule in rules_sorted:
        print(rule)
else:
    print("Правила не найдены.")

# Применение алгоритма Apriori из библиотеки apriori_python
t3, rules_apriori = apriory_py(transactions, minSup=0.01, minConf=0.2)
print("Применение алгоритма Apriori из библиотеки apriori_python")
# Выводим одно правило для проверки структуры
# Вывод 10 наилучших правил
for rule in rules_apriori[:10]:
    print(rule, "\n")
```

```
Применение алгоритма Apriori из библиотеки apyori
frozenset({'avocado'}) frozenset({'mineral water'})
Support: 0.011598453539528063; Confidence: 0.348; Lift: 1.4599261744966443;

frozenset({'burgers'}) frozenset({'eggs'})
Support: 0.02879616051193174; Confidence: 0.33027522935779813; Lift: 1.8378297443715457;

frozenset({'burgers'}) frozenset({'french fries'})
Support: 0.021997067057725635; Confidence: 0.25229357798165136; Lift: 1.4761732671141707;

frozenset({'burgers'}) frozenset({'green tea'})
Support: 0.0174643380882549; Confidence: 0.2003058103975535; Lift: 1.5161391360161947;

frozenset({'burgers'}) frozenset({'milk'})
Support: 0.01786428476203173; Confidence: 0.20489296636085627; Lift: 1.581175041844427;

frozenset({'burgers'}) frozenset({'mineral water'})
Support: 0.024396747100386616; Confidence: 0.2798165137614679; Lift: 1.1738834841861134;

frozenset({'burgers'}) frozenset({'spaghetti'})
Support: 0.021463804826023197; Confidence: 0.24617737003058102; Lift: 1.4139176513012162;

frozenset({'cake'}) frozenset({'eggs'})
Support: 0.019064124783362217; Confidence: 0.23519736842105263; Lift: 1.308765178431985;

frozenset({'cake'}) frozenset({'french fries'})
Support: 0.01786428476203173; Confidence: 0.22039473684210528; Lift: 1.289532699729042;

frozenset({'cake'}) frozenset({'mineral water'})
Support: 0.027463004932675644; Confidence: 0.33881578947368424; Lift: 1.4213966649005065;
```

```
Применение алгоритма Apriori из библиотеки efficient_apriori
{eggs, ground beef} -> {mineral water} (conf: 0.507, supp: 0.010, lift: 2.126, conv: 1.544)
{ground beef, milk} -> {mineral water} (conf: 0.503, supp: 0.011, lift: 2.110, conv: 1.533)
{chocolate, ground beef} -> {mineral water} (conf: 0.474, supp: 0.011, lift: 1.988, conv: 1.448)
{frozen vegetables, milk} -> {mineral water} (conf: 0.469, supp: 0.011, lift: 1.967, conv: 1.434)
{soup} -> {mineral water} (conf: 0.456, supp: 0.023, lift: 1.915, conv: 1.401)
{pancakes, spaghetti} -> {mineral water} (conf: 0.455, supp: 0.011, lift: 1.909, conv: 1.398)
{olive oil, spaghetti} -> {mineral water} (conf: 0.448, supp: 0.010, lift: 1.878, conv: 1.379)
{milk, spaghetti} -> {mineral water} (conf: 0.444, supp: 0.016, lift: 1.861, conv: 1.369)
{chocolate, milk} -> {mineral water} (conf: 0.436, supp: 0.014, lift: 1.828, conv: 1.350)
{ground beef, spaghetti} -> {mineral water} (conf: 0.435, supp: 0.017, lift: 1.826, conv: 1.349)
```

```
Применение алгоритма Apriori из библиотеки apriori_python
[{'burgers'}, {'green tea'}, 0.20030581039755352]

[{'soup'}, {'chocolate'}, 0.20052770448548812]

[{'green tea'}, {'spaghetti'}, 0.20080726538849647]

[{'spaghetti', 'mineral water'}, {'frozen vegetables'}, 0.20089285714285715]

[{'french fries'}, {'chocolate'}, 0.20124804992199688]

[{'mineral water'}, {'milk'}, 0.20134228187919462]

[{'eggs'}, {'french fries'}, 0.20252225519287834]

[{'chocolate'}, {'eggs'}, 0.20260374288039057]

[{'whole wheat rice'}, {'milk'}, 0.20273348519362186]

[{'eggs'}, {'spaghetti'}, 0.2032640949554896]
```

4. Применить алгоритм FP-Growth из библиотеки fpgrowth_py. Подобрать гиперпараметры для алгоритма так, чтобы выводилось порядка 10 наилучших правил.

Листинг 4:

```
# Преобразование данных в список транзакций без NaN
transactions = data.values.tolist()

# Убираем значения NaN, None, и float (оставляем только строки)
transactions = [[str(item) for item in transaction if isinstance(item,
str)] for transaction in transactions]

# Применение алгоритма FP-Growth
min_support = 0.01  # Минимальная поддержка
min_confidence = 0.2  # Минимальная уверенность

# Генерация частых наборов и правил
freqItemSet, rules = fpgrowth(transactions, minSupRatio=min_support,
minConf=min_confidence)

# Вывод 10 наилучших правил
print("Применение алгоритма FP-Growth из библиотеки fpgrowth_py")
for rule in rules[:10]:
    print(rule)
```

```
Применение алгоритма FP-Growth из библиотеки fpgrowth_py
[{'cereals'}, {'mineral water'}, 0.39896373056994816]
[{'red wine'}, {'spaghetti'}, 0.36492890995260663]
[{'red wine'}, {'mineral water'}, 0.3886255924170616]
[{'avocado'}, {'mineral water'}, 0.348]
[{'salmon'}, {'chocolate'}, 0.2507836990595611]
[{'salmon'}, {'spaghetti'}, 0.3166144200626959]
[{'salmon'}, {'mineral water'}, 0.4012539184952978]
[{'fresh bread'}, {'mineral water'}, 0.30959752321981426]
[{'champagne'}, {'chocolate'}, 0.24786324786324787]
[{'honey'}, {'spaghetti'}, 0.25]
```

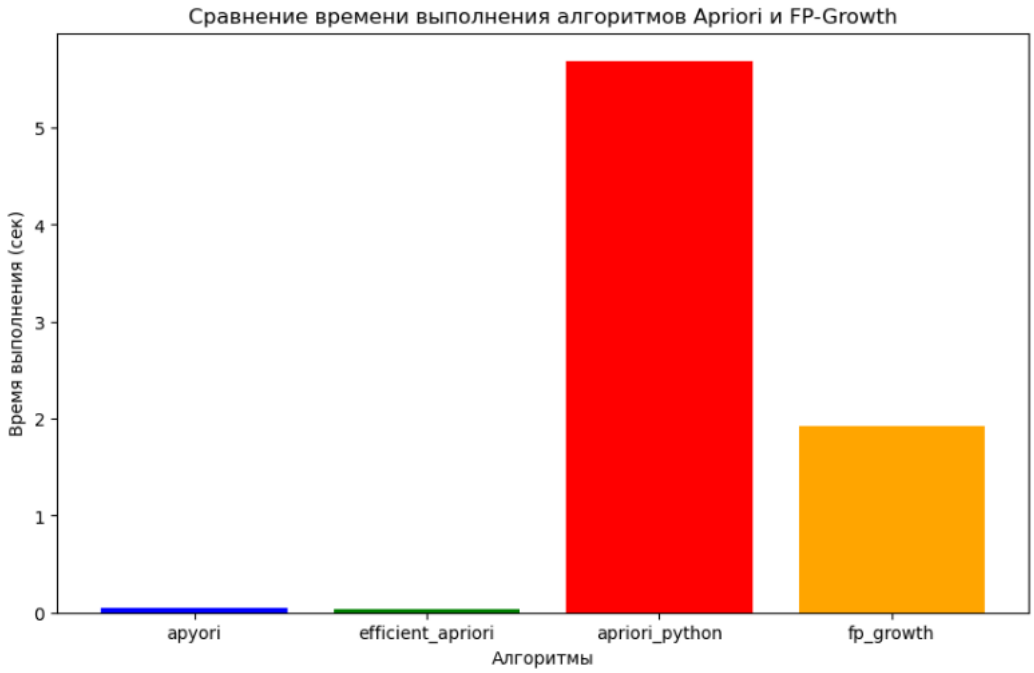
5. Сравнить время выполнения всех алгоритмов и построить гистограмму.

Листинг 5:

```
# Преобразование данных в список транзакций без NaN
transactions = data.values.tolist()
transactions = [[str(item) for item in transaction if isinstance(item,
str)] for transaction in transactions]

# Словарь для хранения времени выполнения
execution_times = {}

# Измерение времени выполнения алгоритма apyori
```

```
start_time = time.time()
rules_apyori = list(apriori(transactions, min_support=0.01,
min_confidence=0.2))
execution_times['apyori'] = time.time() - start_time

# Измерение времени выполнения алгоритма efficient_apriori
start_time = time.time()
itemsets, rules_efficient = efficient_apriori(transactions,
min_support=0.01, min_confidence=0.2)
execution_times['efficient_apriori'] = time.time() - start_time

# Измерение времени выполнения алгоритма apriori_python
start_time = time.time()
t3, rules_apriori = apriory_py(transactions, minSup=0.01, minConf=0.2)
execution_times['apriori_python'] = time.time() - start_time

# Измерение времени выполнения алгоритма FP-Growth
start_time = time.time()
freqItemSet, rules_fp = fpgrowth(transactions, minSupRatio=0.01,
minConf=0.2)
execution_times['fp_growth'] = time.time() - start_time
print(execution_times)
# Построение гистограммы времени выполнения
plt.figure(figsize=(10, 6))
plt.bar(execution_times.keys(), execution_times.values(),
color=['blue', 'green', 'red', 'orange'])
plt.title('Сравнение времени выполнения алгоритмов Apriori и FP-
Growth')
plt.xlabel('Алгоритмы')
plt.ylabel('Время выполнения (сек)')
plt.show()
```

```
{'apyori': 0.04863429069519043, 'efficient_apriori': 0.03895401954650879, 'apriori_python': 5.692606210708618, 'fp_growth': 1.9
200215339660645}
```



6. Загрузить данные data.csv.

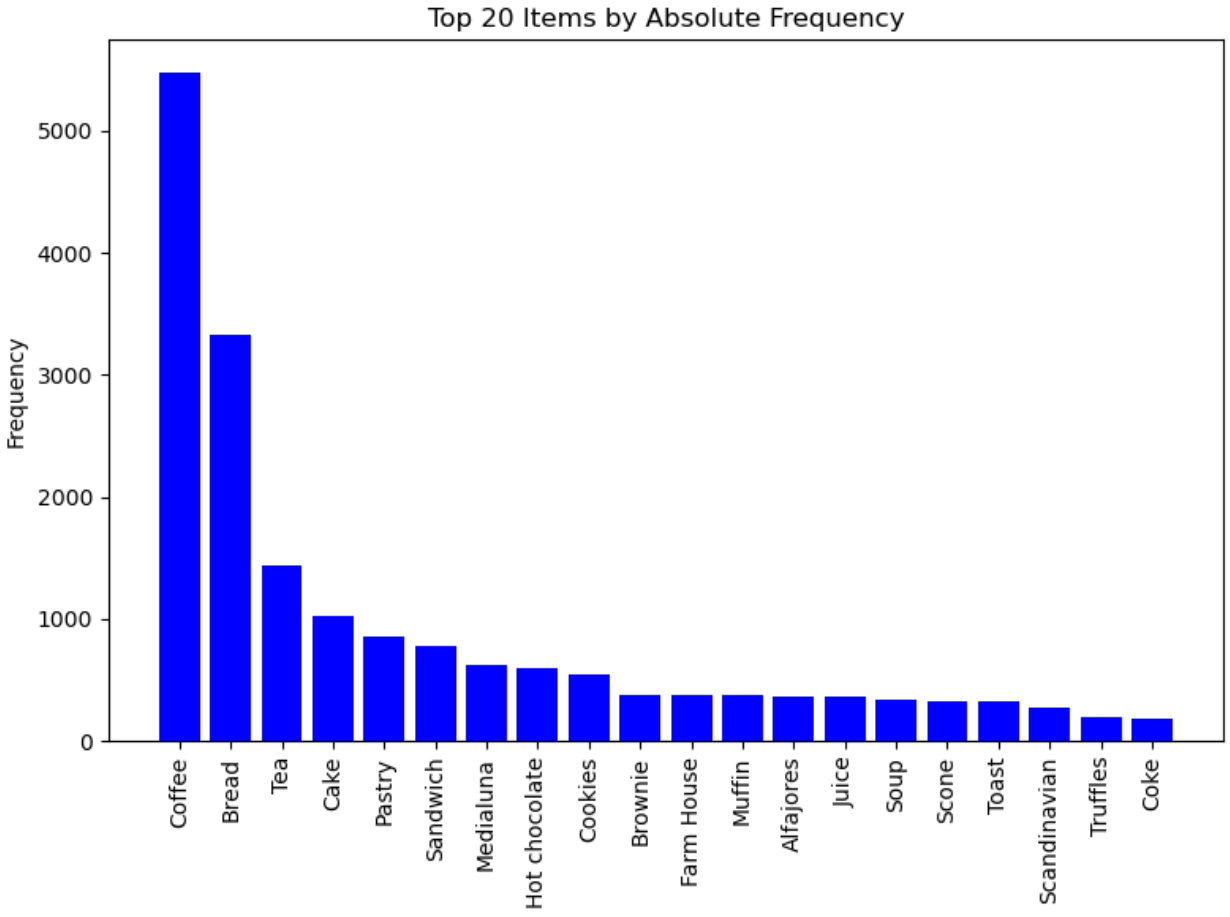
Листинг 6:

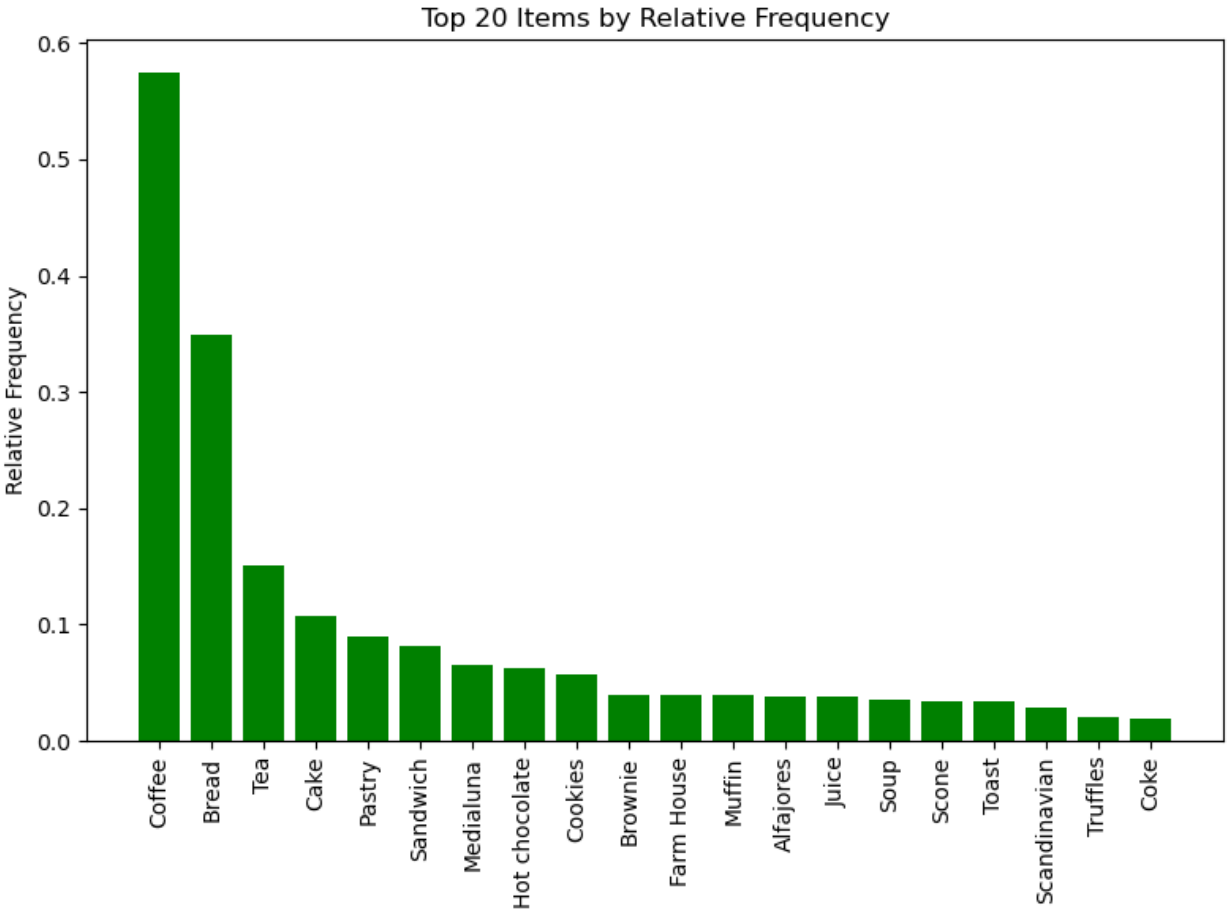
```
# Загружаем CSV файл
data = pd.read_csv('data.csv')

# Просматриваем первые несколько строк данных для проверки
data.head()
```

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

Листинг 7:

```
# Преобразование данных в список транзакций (убираем NaN)
transactions = data.values.tolist()
transactions = [[str(item) for item in transaction if isinstance(item,
str)] for transaction in transactions]

# Подсчёт частоты товаров
item_counts = Counter([item for transaction in transactions for item
in transaction])

# Преобразуем в DataFrame для удобства работы
item_freq = pd.DataFrame(item_counts.items(), columns=['Item',
'Frequency'])

# Сортировка по убыванию частоты и выбор топ 20 товаров
top_20_items = item_freq.sort_values(by='Frequency',
ascending=False).head(20)

# Рассчитываем относительную частоту
total_transactions = len(transactions)
top_20_items['Relative Frequency'] = top_20_items['Frequency'] /
total_transactions

# Построение первой гистограммы - фактическая частота
plt.figure(figsize=(8, 6))
plt.bar(top_20_items['Item'], top_20_items['Frequency'], color='blue')
plt.xticks(rotation=90)
plt.title('Top 20 Items by Absolute Frequency')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()

# Построение второй гистограммы - относительная частота
plt.figure(figsize=(8, 6))
plt.bar(top_20_items['Item'], top_20_items['Relative Frequency'],
color='green')
plt.xticks(rotation=90)
plt.title('Top 20 Items by Relative Frequency')
plt.ylabel('Relative Frequency')
plt.tight_layout()
plt.show()
```


Top 20 Items by Absolute Frequency
Frequency
0
1000
2000
3000
4000
5000
Coffee
Bread
Tea
Cake
Pastry
Sandwich
Medialuna
Hot chocolate
Cookies
Brownie
Farm House
Muffin
Alfajores
Juice
Soup
Scone
Toast
Scandinavian
Truffles
Coke


Top 20 Items by Relative Frequency
Relative Frequency
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Coffee
Bread
Tea
Cake
Pastry
Sandwich
Medialuna
Hot chocolate
Cookies
Brownie
Farm House
Muffin
Alfajores
Juice
Soup
Scone
Toast
Scandinavian
Truffles
Coke

8. Применить алгоритм Apriori, используя 3 разные библиотеки (apriori_python, apyori, efficient_apriori). Подобрать гиперпараметры для алгоритмов так, чтобы выводилось порядка 10 наилучших правил.

Листинг 8:

```
# Преобразование данных в список транзакций без NaN
transactions = data.values.tolist()

# Убираем значения NaN, None, и float (оставляем только строки)
transactions = [[str(item) for item in transaction if isinstance(item,
str)] for transaction in transactions]

# Применение алгоритма Apriori из библиотеки apyori
rules_apyori = list(apriori(transactions, min_support=0.01,
min_confidence=0.2))
print("Применение алгоритма Apriori из библиотеки apyori")
# Вывод 10 наилучших правил
results_apyori = sorted(rules_apyori, key=lambda x: x[2],
reverse=True)[:10]
# for rule in results_apyori:
#     print(rule, '\n')
for result in results_apyori:
    for subset in result[2]:
        print(subset[0], subset[1])
        print("Support: {0}; Confidence: {1}; Lift:
{2};".format(result[1], subset[2],subset[3]))
        print()
# Применение алгоритма Apriori из библиотеки efficient_apriori
itemsets, rules_efficient = efficient_apriori(transactions,
min_support=0.01, min_confidence=0.2)

# Вывод 10 наилучших правил, отсортированных по уверенности
(confidence)
rules_sorted = sorted(rules_efficient, key=lambda rule:
rule.confidence, reverse=True)[:10]
print("Применение алгоритма Apriori из библиотеки efficient_apriori")
# Печать правил
if rules_sorted:
    for rule in rules_sorted:
        print(rule)
else:
    print("Правила не найдены.")

# Применение алгоритма Apriori из библиотеки apriori_python
t3, rules_apriori = apriory_py(transactions, minSup=0.01, minConf=0.2)
print("Применение алгоритма Apriori из библиотеки apriori_python")
# Выводим одно правило для проверки структуры
# Вывод 10 наилучших правил
for rule in rules_apriori[:10]:
    print(rule, "\n")
```

```
Применение алгоритма Apriori из библиотеки apyori
frozenset({'Alfajores'}) frozenset({'Bread'})
Support: 0.010283315844700944; Confidence: 0.28488372093023256; Lift: 0.876919205576588;

frozenset({'Alfajores'}) frozenset({'Coffee'})
Support: 0.01951731374606506; Confidence: 0.5406976744186047; Lift: 1.137996651327143;

frozenset({'Brownie'}) frozenset({'Bread'})
Support: 0.010703043022035676; Confidence: 0.26912928759894456; Lift: 0.82842445439856;

frozenset({'Cake'}) frozenset({'Bread'})
Support: 0.023189926547743968; Confidence: 0.22482197355035607; Lift: 0.692039214449255;

frozenset({'Bread'}) frozenset({'Coffee'})
Support: 0.089401888772298; Confidence: 0.2751937984496124; Lift: 0.5791954282740296;

frozenset({'Cookies'}) frozenset({'Bread'})
Support: 0.014375655823714585; Confidence: 0.26601941747572816; Lift: 0.8188517598655328;

frozenset({'Hot chocolate'}) frozenset({'Bread'})
Support: 0.013326337880377754; Confidence: 0.23007246376811594; Lift: 0.7082010916376438;

frozenset({'Medialuna'}) frozenset({'Bread'})
Support: 0.016789087093389297; Confidence: 0.27350427350427353; Lift: 0.8418913845270436;

frozenset({'Pastry'}) frozenset({'Bread'})
Support: 0.028961175236096537; Confidence: 0.33865030674846625; Lift: 1.0424216483568745;

frozenset({'Sandwich'}) frozenset({'Bread'})
Support: 0.01689401888772298; Confidence: 0.23676470588235293; Lift: 0.7288009195926433;

Применение алгоритма Apriori из библиотеки efficient_apriori
{Toast} -> {Coffee} (conf: 0.704, supp: 0.024, lift: 1.483, conv: 1.776)
{Spanish Brunch} -> {Coffee} (conf: 0.599, supp: 0.011, lift: 1.260, conv: 1.308)
{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)
{Alfajores} -> {Coffee} (conf: 0.541, supp: 0.020, lift: 1.138, conv: 1.143)
{Juice} -> {Coffee} (conf: 0.534, supp: 0.020, lift: 1.124, conv: 1.127)
{Sandwich} -> {Coffee} (conf: 0.532, supp: 0.038, lift: 1.120, conv: 1.122)
{Cake} -> {Coffee} (conf: 0.527, supp: 0.054, lift: 1.109, conv: 1.110)
{Scone} -> {Coffee} (conf: 0.523, supp: 0.018, lift: 1.101, conv: 1.100)
{Cookies} -> {Coffee} (conf: 0.518, supp: 0.028, lift: 1.091, conv: 1.090)

Применение алгоритма Apriori из библиотеки apriori_python
[{'Cake'}, {'Bread'}, 0.22482197355035605]

[{'Cake'}, {'Tea'}, 0.2288911495422177]

[{'Hot chocolate'}, {'Bread'}, 0.23007246376811594]

[{'Pastry', 'Coffee'}, {'Bread'}, 0.23555555555555555]

[{'Sandwich'}, {'Bread'}, 0.23676470588235293]

[{'Cookies'}, {'Bread'}, 0.26601941747572816]

[{'Brownie'}, {'Bread'}, 0.2691292875989446]

[{'Medialuna'}, {'Bread'}, 0.27350427350427353]

[{'Bread'}, {'Coffee'}, 0.2751937984496124]

[{'Alfajores'}, {'Bread'}, 0.28488372093023256]
```

9. Применить алгоритм FP-Growth из библиотеки fpgrowth_py. Подобрать гиперпараметры для алгоритма так, чтобы выводилось порядка 10 наилучших правил.

Листинг 9:

```
# Преобразование данных в список транзакций без NaN
transactions = data.values.tolist()

# Убираем значения NaN, None, и float (оставляем только строки)
transactions = [[str(item) for item in transaction if isinstance(item,
str)] for transaction in transactions]

# Применение алгоритма FP-Growth
min_support = 0.01  # Минимальная поддержка
min_confidence = 0.2  # Минимальная уверенность

# Генерация частых наборов и правил
freqItemSet, rules = fpgrowth(transactions, minSupRatio=min_support,
minConf=min_confidence)

# Вывод 10 наилучших правил
print("Применение алгоритма FP-Growth из библиотеки fpgrowth_py")
for rule in rules[:10]:
    print(rule)
```

```
Применение алгоритма FP-Growth из библиотеки fpgrowth_py
[{'Tiffin'}, {'Coffee'}, 0.547945205479452]
[{'Spanish Brunch'}, {'Coffee'}, 0.5988372093023255]
[{'Toast'}, {'Coffee'}, 0.7044025157232704]
[{'Scone'}, {'Bread'}, 0.26299694189602446]
[{'Scone'}, {'Coffee'}, 0.5229357798165137]
[{'Soup'}, {'Tea'}, 0.26380368098159507]
[{'Soup'}, {'Coffee'}, 0.4601226993865031]
[{'Juice'}, {'Coffee'}, 0.5342465753424658]
[{'Alfajores'}, {'Bread'}, 0.28488372093023256]
[{'Alfajores'}, {'Coffee'}, 0.5406976744186046]
```

10. Сравнить время выполнения всех алгоритмов и построить гистограмму.

Листинг 10:

```
# Преобразование данных в список транзакций без NaN
transactions = data.values.tolist()
transactions = [[str(item) for item in transaction if isinstance(item,
str)] for transaction in transactions]

# Словарь для хранения времени выполнения
execution_times = {}

# Измерение времени выполнения алгоритма apyori
start_time = time.time()
rules_apyori = list(apriori(transactions, min_support=0.01,
min_confidence=0.2))
execution_times['apyori'] = time.time() - start_time

# Измерение времени выполнения алгоритма efficient_apriori
start_time = time.time()
itemsets, rules_efficient = efficient_apriori(transactions,
min_support=0.01, min_confidence=0.2)
```

```
execution_times['efficient_apriori'] = time.time() - start_time

# Измерение времени выполнения алгоритма apriori_python
start_time = time.time()
t3, rules_apriori = apriory_py(transactions, minSup=0.01, minConf=0.2)
execution_times['apriori_python'] = time.time() - start_time

# Измерение времени выполнения алгоритма FP-Growth
start_time = time.time()
freqItemSet, rules_fp = fpgrowth(transactions, minSupRatio=0.01,
minConf=0.2)
execution_times['fp_growth'] = time.time() - start_time
print(execution_times)
# Построение гистограммы времени выполнения
plt.figure(figsize=(10, 6))
plt.bar(execution_times.keys(), execution_times.values(),
color=['blue', 'green', 'red', 'orange'])
plt.title('Сравнение времени выполнения алгоритмов Apriori и FP-
Growth')
plt.xlabel('Алгоритмы')
plt.ylabel('Время выполнения (сек)')
plt.show()
```

```
{'apyori': 0.012509346008300781, 'efficient_apriori': 0.008517980575561523, 'apriori_python': 0.8036558628082275, 'fp_growth':
0.4264848232269287}
```