|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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.

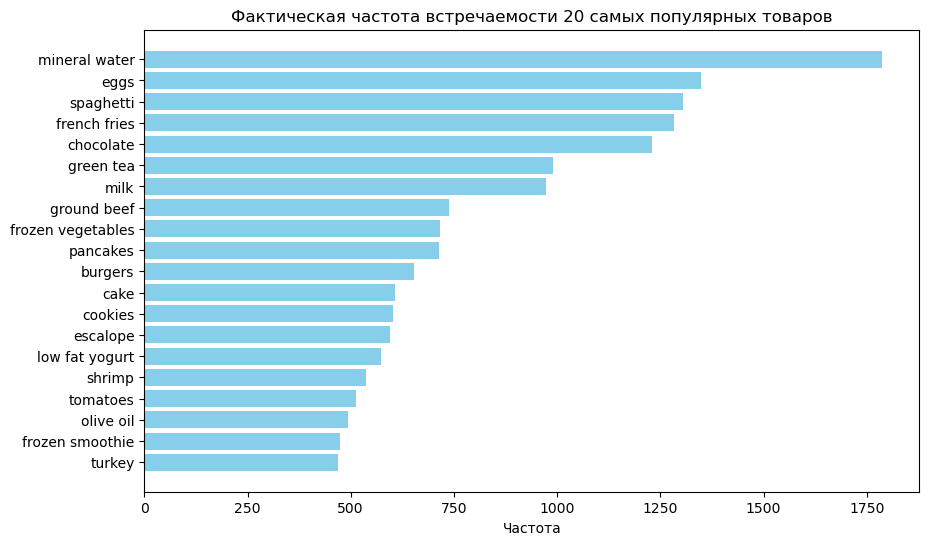
Листинг 1:

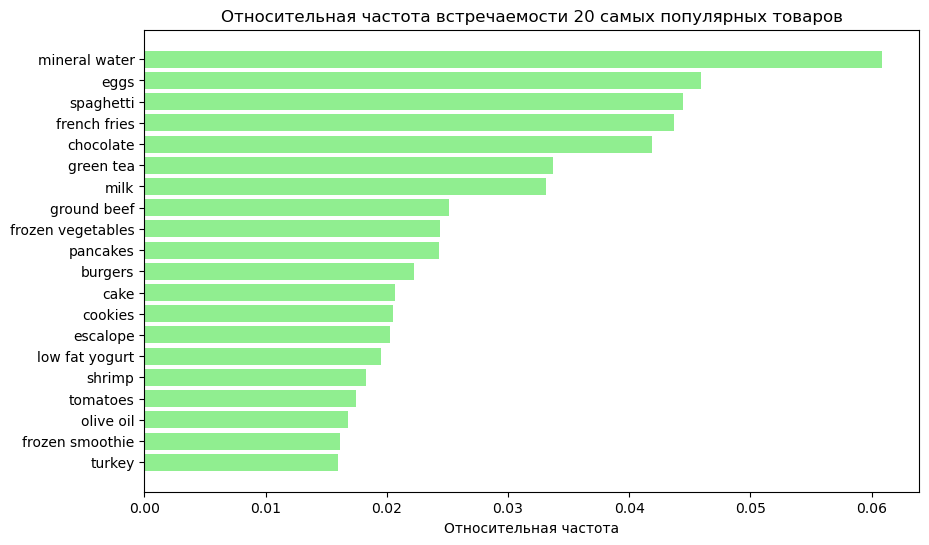
|  |
| --- |
| # Загрузка данных из файла Market\_Basket\_Optimisation.csv  file\_path = 'Market\_Basket\_Optimisation.csv'  data = pd.read\_csv(file\_path, header=None)  # Отображение первых строк данных  data.head() |

1. 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() |

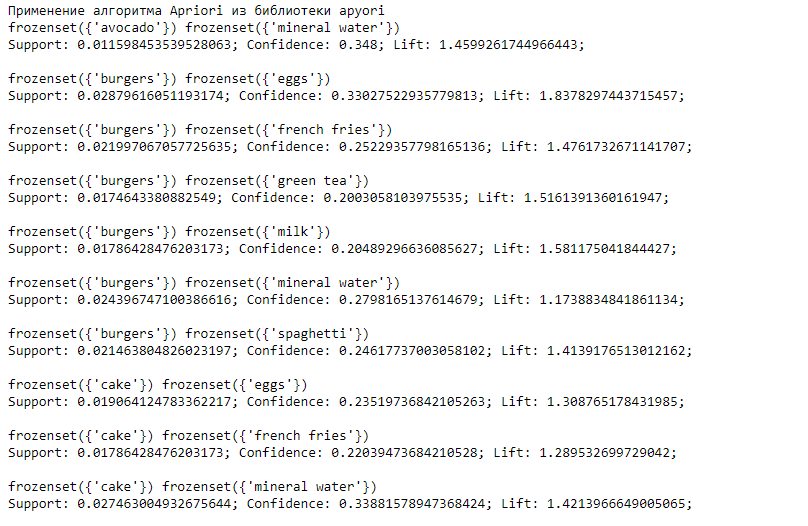


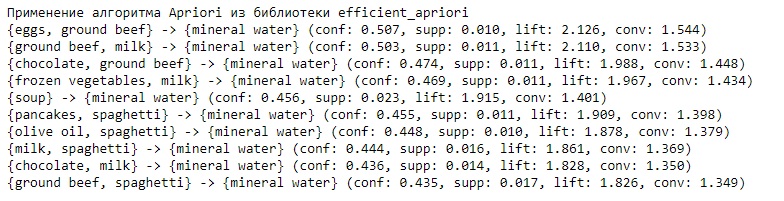


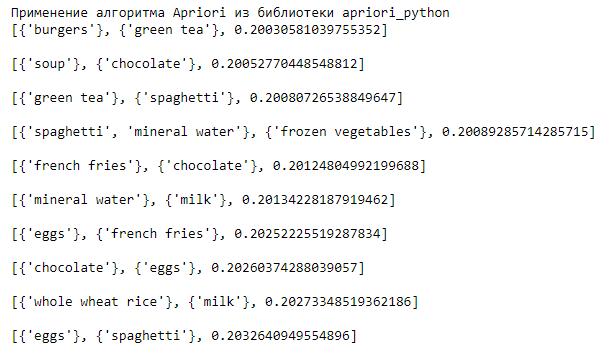
1. Применить алгоритм Apriori, используя 3 разные библиотеки (apriori\_python, apyori, efficient\_apriori). Подобрать гиперпараметры для алгоритмов так, чтобы выводилось порядка 10 наилучших правил.

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") |



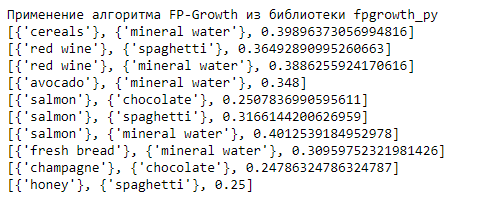




1. Применить алгоритм FP-Growth из библиотеки fpgrowth\_py. Подобрать гиперпараметры для алгоритма так, чтобы выводилось порядка 10 наилучших правил.

Листинг 4:

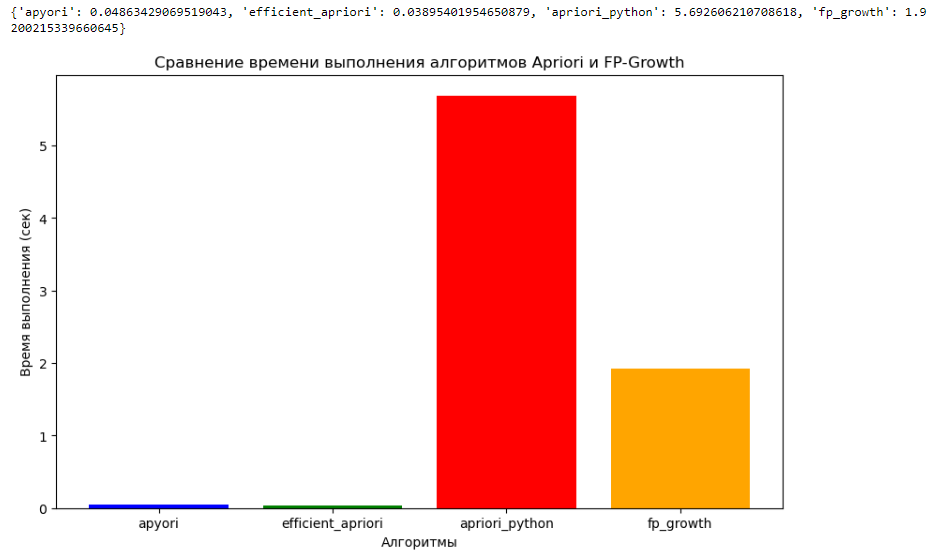
|  |
| --- |
| # Преобразование данных в список транзакций без NaN  transactions = data.values.tolist()  # Убираем значения NaN, None, и float (оставляем только строки)  transactions = [[str(item) for item in transaction if isinstance(item, str)] for transaction in transactions]  # Применение алгоритма 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) |



1. 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() |



1. Загрузить данные data.csv.

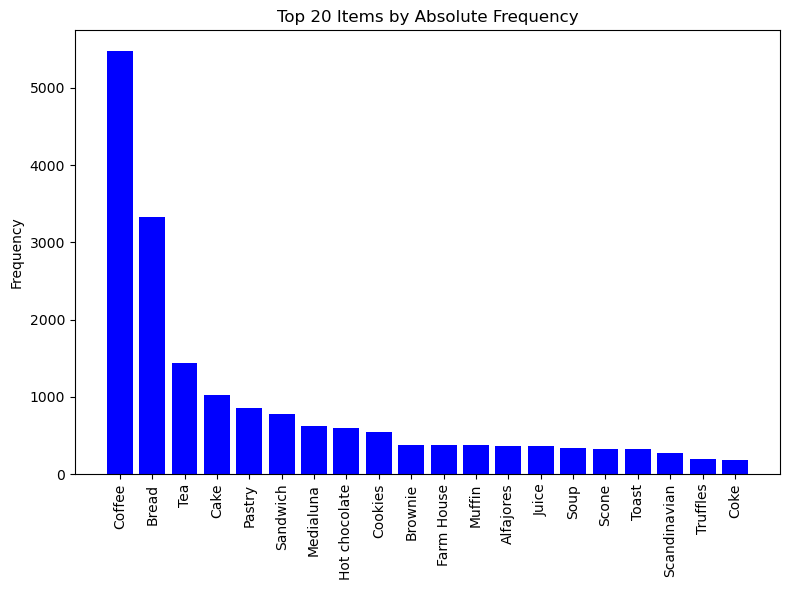
Листинг 6:

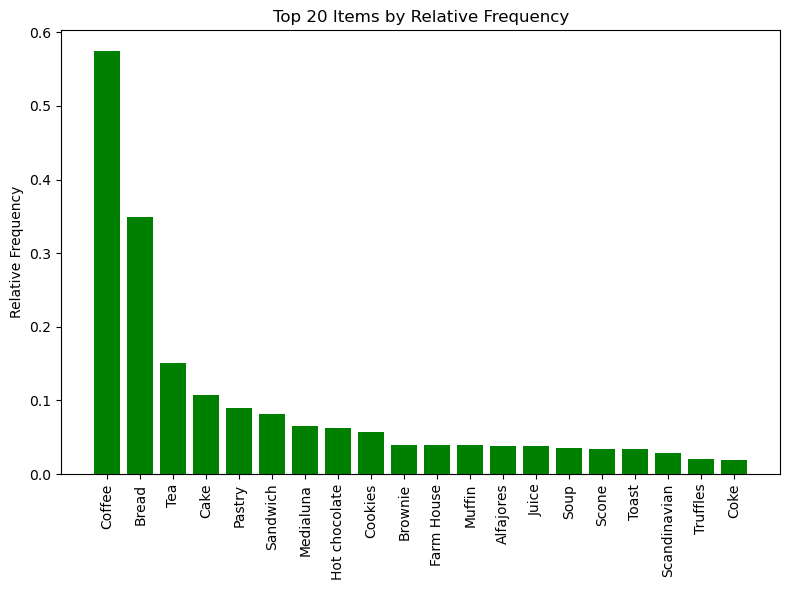
|  |
| --- |
| # Загружаем CSV файл  data = pd.read\_csv('data.csv')  # Просматриваем первые несколько строк данных для проверки  data.head() |

1. 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() |

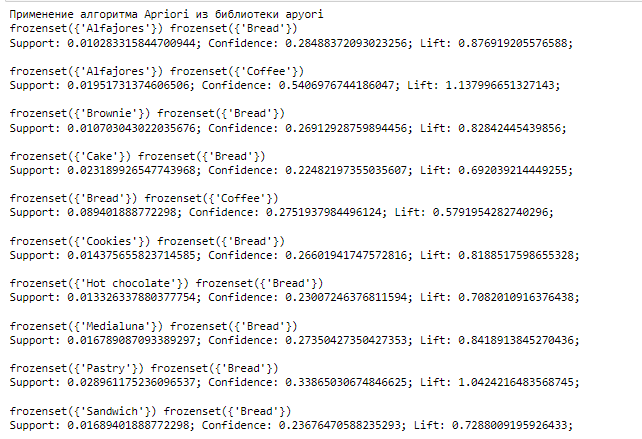


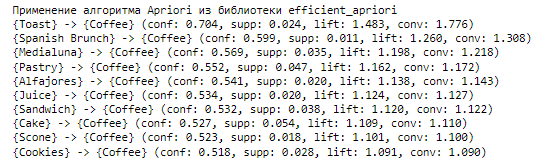


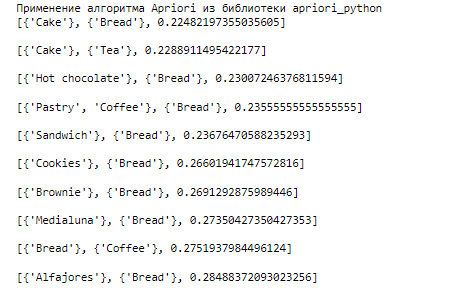
1. Применить алгоритм Apriori, используя 3 разные библиотеки (apriori\_python, apyori, efficient\_apriori). Подобрать гиперпараметры для алгоритмов так, чтобы выводилось порядка 10 наилучших правил.

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") |



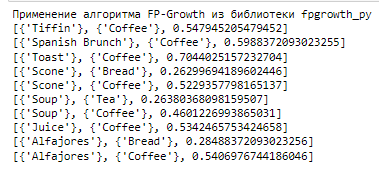




1. Применить алгоритм FP-Growth из библиотеки fpgrowth\_py. Подобрать гиперпараметры для алгоритма так, чтобы выводилось порядка 10 наилучших правил.

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) |



1. 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() |

