

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Пензенский государственный университет» (ПГУ)

С. В. Рындина

БИЗНЕС-АНАЛИТИКА
НА ОСНОВЕ БОЛЬШИХ ДАННЫХ:
ОБУЧЕНИЕ БЕЗ УЧИТЕЛЯ
НА ЯЗЫКАХ PYTHON И R

Учебно-методическое пособие

Пенза
Издательство ПГУ
2020

УДК 004.6 (075)

P93

Р е ц е н з е н т

кандидат технических наук, доцент

А. А. Масленников

Рындина, С. В.

P93

Бизнес-аналитика на основе больших данных: обучение без учителя на языках Python и R : учеб.-метод. пособие / С. В. Рындина. – Пенза : Изд-во ПГУ, 2020. – 76 с.

Рассмотрены возможности языков программирования Python и R для реализации методов обучения без учителя: кластеризации на основе k-средних, метода главных компонент для снижения размерности и построения ассоциативных правил. Приведены задания для лабораторных работ по методам обучения без учителя. Материал соответствует программе дисциплины «Бизнес-аналитика на основе больших данных» и может быть использован также при изучении дисциплины «Бизнес-прогнозирование» и написании выпускной работы бакалавра.

Издание подготовлено на кафедре «Экономическая кибернетика» Института экономики и управления ПГУ и предназначено для обучающихся по направлению подготовки 38.03.05 «Бизнес-информатика».

УДК 004.6 (075)

© Пензенский государственный
университет, 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1. ОБУЧЕНИЕ БЕЗ УЧИТЕЛЯ	5
1.1. Метод k-средних	5
1.2. Метод главных компонент	9
1.3. Ассоциативные правила	12
2. ОБУЧЕНИЕ БЕЗ УЧИТЕЛЯ НА ЯЗЫКЕ R, RSTUDIO	16
2.1. Настройка окружения	16
2.2. Метод k-средних	22
2.2.1. Набор данных <i>ирисы</i>	23
2.2.2. Набор данных <i>медицинское страхование</i>	26
2.2.3. Набор данных <i>смс</i>	30
2.3. Метод главных компонент	36
2.4. Ассоциативные правила	40
3. ОБУЧЕНИЕ БЕЗ УЧИТЕЛЯ НА ЯЗЫКЕ PYTHON, ANACONDA, JUPYTER NOTEBOOK, GOOGLE COLAB	44
3.1. Настройка окружения	44
3.2. Метод k-средних	50
3.2.1. Набор данных <i>ирисы</i>	50
3.2.2. Набор данных <i>медицинское страхование</i>	54
3.2.3. Набор данных <i>смс</i>	58
3.2.4. Машинное обучение на основе метода k-средних	62
3.3. Метод главных компонент	64
3.4. Ассоциативные правила	66
4. ЛАБОРАТОРНЫЕ РАБОТЫ	70
4.1. Лабораторная работа «Кластерный анализ методом k-средних»	70
4.2. Лабораторная работа «Понижение размерности данных»	71
4.3. Лабораторная работа «Ассоциативные правила»	72
БИБЛИОГРАФИЧЕСКИЙ СПИСОК	73
ПРИЛОЖЕНИЕ 1	75

ВВЕДЕНИЕ

В соответствии с учебным планом обучающиеся третьего курса направления подготовки 38.03.05 «Бизнес-информатика» изучают дисциплину «Бизнес-аналитика на основе больших данных».

Обучение без учителя – методы выявления в неразмеченных данных важных зависимостей, которые можно использовать в продуктах, услугах, информационных системах, бизнес-процессах для повышения прибыльности, экономии ресурсов и т.п.

Целью учебно-методического пособия является рассмотрение методов обучения без учителя и их реализация на двух языках: R и Python. Языки программирования R и Python – основные инструменты для Data science (науки о данных). Программирование на этих языках для реализации алгоритмов анализа данных – фундамент для аналитиков данных, специалистов в области Data science, специалистов в области технологий Big Data.

В пособии использованы материалы:

Открытый курс по машинному обучению от Open Data Science [1].

Книга «Введение в статистическое обучение» – авторы: Е. Джеймс, Д. Уиттон, Т. Хасты, Р. В. Тибширани [2].

Книга «Python и машинное обучение» – автор С. Рашка [3].

Книга «Построение систем машинного обучения» – автор Л. П. Коэлью [4].

Книга «Классификация, регрессия и другие алгоритмы Data Mining» – авторы В. К. Шитиков, С. Э. Мاستицкий [5].

1. ОБУЧЕНИЕ БЕЗ УЧИТЕЛЯ

Обучение без учителя (Unsupervised Learning) – группа методов анализа данных для выявления закономерностей в неразмеченных данных. Поиск скрытых закономерностей доверяется самим алгоритмам: группировка потребителей по сходным профилям покупок, нахождение ассоциативных правил в последовательности действий: вместе с этим товаром приобретали и эти (платформы для покупок товаров), вместе с этим материалом просматривали эти (информационные порталы, интернет СМИ и т.п.).

1.1. Метод k-средних

Материалы: [1; 2, с. 415–418; 3, с. 289–298; 4, с. 82–85; 5, глава 10].

Кластеризация позволяет структурировать информацию в содержательные подгруппы (кластеры), не имея предварительных сведений о количестве таких групп и о значениях характеристик представителей кластеров (содержательных связях между данными кластера). Кластеризацию относят к методам разведочного анализа данных: неразмеченные данные разносятся по различным кластерам на основе подобия характеризующих их признаков.

Один из самых популярных алгоритмов кластеризации – метод k-средних. Алгоритм k-средних – кластеризация на основе прототипов. Это означает, что каждый кластер представлен прототипом, который может быть либо центроидом (средним) подобных точек с непрерывными признаками, либо медоидом (наиболее представительной или наиболее часто встречающейся точкой) в случае категориальных признаков (или иначе факторов, например, пол, наличие вредной привычки и т.п.). Метод k-средних хорошо справляется с идентификацией кластеров сферической формы. Однако этот метод требует изначально задать количество кластеров. При неудачном выборе числа кластеров или неподходящем задании начального положения центроидов кластеров результат кластеризации может быть сомнительным. Кроме того, этот метод чувствителен к качеству данных: выбросы, аномальные наблюдения, шум в данных могут также ухудшить итоговый результат.

С помощью ресурса, визуализирующего метод k-средних [6], можно проиллюстрировать первую проблему с неудачным начальным выбором кластеров.

На рис. 1.1 и 1.2 визуально определяются три кластера. Но в первом случае один центроид оказался в левом нижнем кластере, а два других – в правом нижнем кластере, и за три шага было получено корректное разбиение на три кластера. А во втором случае центроид случайно оказался между двумя кластерами: левым нижним и верхним, а два других – в правом нижнем кластере. И окончательное разбиение сохранило эту ошибочную кластеризацию: точки, очевидно расположенные в одном кластере, разнесены в два разных, а два непохожих скопления точек принадлежат одному кластеру.

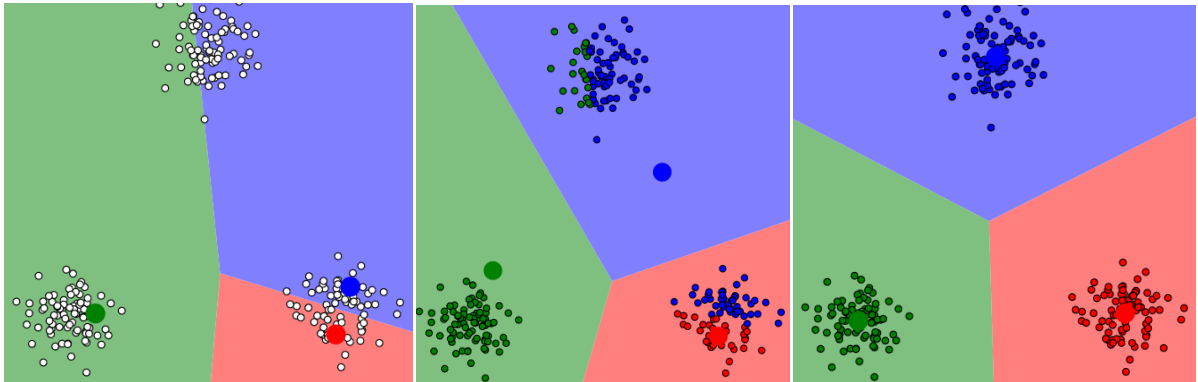


Рис. 1.1. Визуализация метода k-средних

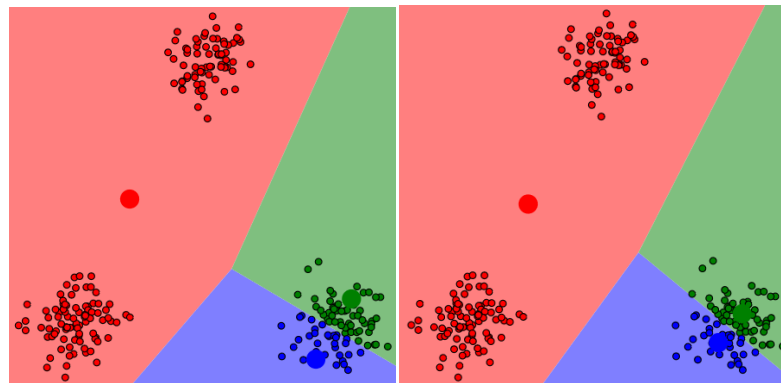


Рис. 1.2. Визуализация метода k-средних

В алгоритме k-средних последовательно реализуются следующие шаги:

- 1 шаг: выбрать количество кластеров (произвольно или опираясь на какие-то априорные предположения);
- 2 шаг: в пространстве данных расположить произвольным образом центроиды (центры кластеров);
- 3 шаг: определить для каждой точки набора данных, к какому центроиду она ближе (используя некоторую метрику для вычисления расстояния, например евклидову);

– 4 шаг: для получившихся кластеров найти новое положение центроида (как точки, суммарное расстояние от которой до всех точек кластера – наименьшее).

Шаги 3 и 4 повторяются либо фиксированное количество раз, либо до момента, когда смещение новых центров кластеров относительно центров кластеров на предыдущем шаге будет меньше некоторого значения (т.е. центры кластеров стабилизируются).

Для кластеризации объектов с непрерывными признаками обычно используют евклидово расстояние, т.е. алгоритм минимизирует суммарное квадратичное отклонение точек кластеров от центров этих кластеров:

$$J = \sum_{j=1}^k \sum_{i=1}^n (x_i^{(j)} - c_j)^2, \quad (1.1)$$

где $x_i^{(j)}$ – наблюдение с номером i , которое отнесено к кластеру j , а c_j – центроид кластера j , k – число кластеров, n – число наблюдений.

В классическом алгоритме k -средних для размещения начальных центроидов используется случайное число (random seed), что иногда дает плохие результаты кластеризации, а в некоторых случаях – замедление сходимости. Есть модификация метода k -средних, которая обязывает выбирать в качестве центроида не произвольную точку, а одно из наблюдений данных.

Более достоверные результаты кластеризации дает реализация метода k -средних k -means++, когда в качестве центроидов выбираются точки наблюдений, далеко отстоящие друг от друга.

Так как метод k -средних находит локальный, а не глобальный минимум для внутрикластерных расстояний (1.1), то результаты будут зависеть от начального разбиения кластеров. Важно запускать алгоритм несколько раз, используя разные конфигурации его реализации, и выбрать наилучшую.

Также важным параметром качества кластеризации является интерпретируемость конечного результата. Если кластеризация проводится в данных бизнес-кейса, то необходимо, чтобы результаты кластеризации помогали достижению бизнес-целей: выявляли сходство объектов (общие шаблоны поведения) по тем параметрам, которые позволят управлять поведением объектов в выгодном для бизнеса направлении.

Так как кластеризация дает неустойчивые решения, то, если объем данных позволяет, можно выполнять кластеризацию на разных частях данных, чтобы получить представление об устойчивости получаемых кластеров.

Так как данные не размечены, то количество кластеров, на которые должен быть разделен исходный набор, изначально неизвестно.

При кластеризации методом k -средних количество кластеров чаще всего оценивают с помощью «метода локтя» (Elbow method). Это интуитивная, довольно грубая эвристика. На графике откладывается некоторая величина, характеризующая качество кластеризации, например, внутрикластерная сумма расстояний J (1.1) для разного количества кластеров (в статистических пакетах для реализации метода k -средних нужна величина – это искажение или инерция, например в методе KMeans на языке Python – это значение атрибута `inertia_`). Оптимальное количество кластеров соответствует значению k , после которого величина J перестает резко падать. Понятно, что лучший вариант $J = 0$ достигается при количестве кластеров, совпадающих с числом наблюдений. Но в бизнесе обычно нужны стабильные кластеры с большим количеством наблюдений, для которых можно выявить полезные закономерности в поведении, расходах и т.п.

Если в данных есть разметка (разбиение на классы), то на основе метода k -средних можно построить простую реализацию машинного обучения с метриками качества обучения. На тренировочной части данных происходит разбиение на кластеры, а на тестовых данных определяется, насколько хорошо машина распознает в новых данных (тестовая часть, которая не участвовала в обучении) выявленные при обучении кластеры.

Важно заметить, что такое машинное обучение – это уже обучение с учителем. Хотя мы не предъявляем разметку данных (классы или категории данных) при обучении, но именно на основе разметки происходит оценка качества обучения.

Машинное обучение на неразмеченных данных с использованием метода k -средних также реализуемо, но в этом случае сложнее оценить качество.

Для описания метрик качества машинного обучения построим табл. 1.1.

Таблица 1.1

Анализ соответствия кластеров и классов размеченных данных

		Класс	
		1	0
Кластер	1	True Positive (TP)	False Positive (FP)
	0	False Negative (FN)	True Negative (TN)

Очевидно, что при кластеризации в случае двух классов возможны ошибки двух видов: False Positive (FP) – неверно распознан объект класса с меткой 0 и False Negative (FN) – неверно распознан объект класса с меткой 1.

Метрики, оценивающие качество проведенной кластеризации, следующие [7, 8].

Accuracy – это метрика, которая показывает долю правильных ответов модели. Ее значение равно отношению числа правильных ответов, которые дала модель, к числу всех объектов. Эта метрика зависит от соотношения классов объектов в выборке и используется нечасто:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \quad (1.2)$$

Для оценки качества распознавания каждого класса вводятся метрики precision (точность) и recall (полнота).

Precision (точность) показывает долю объектов, названных классификатором положительными и при этом действительно являющимися положительными:

$$precision = \frac{TP}{TP + FP}. \quad (1.3)$$

Recall (полнота) показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм:

$$recall = \frac{TP}{TP + FN}. \quad (1.4)$$

F-мера – среднее гармоническое precision и recall. F-мера достигает максимума при полноте и точности, равными единице, и близка к нулю, если один из аргументов близок к нулю:

$$F = \frac{precision \cdot recall}{precision + recall}. \quad (1.5)$$

1.2. Метод главных компонент

Материалы: [1; 2, с. 403–409; 3, с. 133–140; 4, с. 260–263].

Метод главных компонент, или PCA (Principal Component Analysis), – один из самых интуитивно простых и часто используемых методов для снижения размерности данных и проекции их на ортогональное подпространство признаков.

Это обучение без учителя, которое позволяет перейти к новым, искусственно сконструированным на основе исходных данных переменным, число которых будет существенно меньше. Такое снижение размерности без существенной потери исходной информации делает многие вычислимые процедуры более быстрыми и менее ресурсоемкими без потери качества результата.

Методы понижения размерности можно разбить на две группы: отбор признаков и выделение признаков. Метод главных компонент относится ко второй группе. К нему прибегают, когда нужно сократить количество признаков, но неизвестно, какой метод выделения признаков подойдет лучше. Ограничением PCA является его линейность.

Рассмотрим признаки (набор переменных) $X_1, X_2, \dots, X_p$, которыми характеризуют исходные наблюдения. Максимально интересны для анализа те признаки, по которым в наблюдениях есть наибольшая вариация (разброс), т.е. признаки, которые дают наибольший вклад в разброс точек наблюдений в пространстве (в направлении этих признаков облако точек наблюдений максимально вытянуто) и именно эти признаки дают наибольший вклад в дисперсию.

Первая главная компонента набора переменных $X_1, X_2, \dots, X_p$ представляет собой нормализованную линейную комбинацию этих переменных:

$$Z_1 = \gamma_{11}X_1 + \gamma_{21}X_2 + \dots + \gamma_{p1}X_p. \quad (1.6)$$

Вектор $\gamma_1 = (\gamma_{11}, \gamma_{21}, \dots, \gamma_{p1})^T$ определяет нагрузки первой главной компоненты, а нормализация означает, что

$$\sum_{j=1}^p \gamma_{j1}^2 = 1. \quad (1.7)$$

Условие (1.7) необходимо для контроля над величиной дисперсии.

Для применения метода главных компонент переменные $X_1, X_2, \dots, X_p$ предварительно центрируются (среднее значение по каждой центрированной переменной в наборе данных будет равно нулю). Компоненты вектора γ_1 находятся из условия, что дисперсия по новой сконструированной переменной Z_1 будет максимальной.

После определения первой главной компоненты Z_1 находят вторую Z_2 , которая также является линейной комбинацией переменных $X_1, X_2, \dots, X_p$ и обладает максимальной дисперсией, при этом не коррелирует с компонентой Z_1 (т.е. не зависит от нее).

$$Z_2 = \gamma_{12}X_1 + \gamma_{22}X_2 + \dots + \gamma_{p2}X_p. \quad (1.8)$$

Отсутствие корреляции между Z_1 и Z_2 эквивалентно условию, что векторы нагрузок компонент γ_1 и γ_2 ортогональны (перпендикулярны).

Каждый вектор нагрузок главных компонент является уникальным (по абсолютным значениям его элементов), т.е. два разных статистических пакета дадут одинаковые векторы нагрузок главных компонент (различие может быть только в знаках). Векторы значений главных компонент также уникальны. Если знаки нагрузок в разных пакетах будут отличаться, то и знаки главных компонент поменяются на противоположные.

Для определения числа главных компонент используют график «каменистой осыпи», который показывает, как много информации из исходного набора утрачивается при использовании главных компонент.

Для построения графика и определения достаточного числа главных компонент используется доля дисперсии, объясненная каждой компонентой (proportion of variance explained – PVE).

Для центрированных переменных общая доля дисперсии находится по формуле

$$\sum_{j=1}^p \text{Var}(X_j) = \frac{1}{n} \sum_{j=1}^p \sum_{i=1}^n x_{ij}^2. \quad (1.9)$$

Дисперсия, объясненная компонентой с номером m :

$$\frac{1}{n} \sum_{i=1}^n z_{im}^2 = \frac{1}{n} \sum_{i=1}^n \left(\sum_{j=1}^p \gamma_{jm} x_{ij} \right)^2. \quad (1.10)$$

PVE главной компоненты с номером m находится по формуле

$$\text{PVE}(Z_m) = \frac{\sum_{i=1}^n (\sum_{j=1}^p \gamma_{jm} x_{ij})^2}{\sum_{j=1}^p \sum_{i=1}^n x_{ij}^2}. \quad (1.11)$$

На графике «каменистой осыпи» (другое название график Кеттела) по оси абсцисс откладывается номер компоненты, по оси ординат – доля объясненной дисперсии PVE соответствующей компоненты. Вклад каждой последующей компоненты, оставаясь положительным, убывает.

Подобный визуальный анализ, как и в случае определения числа кластеров «методом локтя», является субъективным. Но он позволяет использовать интуитивные эвристики там, где общепринятого объективного подхода не существует.

Метод главных компонент, относясь к методам обучения без учителя, может быть в дальнейшем применен в обучении с учителем. В этом случае он рассматривается как метод подготовки данных для обучения с учителем.

1.3. Ассоциативные правила

Материалы: [4, с. 199–207; 5, глава 5; 9].

Ассоциативные правила представляют собой механизм нахождения логических закономерностей между связанными элементами (событиями или объектами). Пусть имеется $A = \{A_1, A_2, \dots, A_n\}$ – конечное множество уникальных элементов. Из этих компонентов может быть составлено множество наборов T .

Ассоциативные правила $A \rightarrow T$ имеют следующий вид:

$$\text{если } \langle \text{условие} \rangle \text{ то } \langle \text{результат} \rangle, \quad (1.12)$$

где $\langle \text{условие} \rangle$ – не логическое выражение, а набор объектов из множества A , с которыми связаны (ассоциированы) объекты того же множества, включенные в $\langle \text{результат} \rangle$ данного правила.

Например, ассоциативное правило «если... (горошек, докторская колбаса), то (майонез)» означает, что если транзакция оплаты покупки в продуктовом магазине содержит два продукта для приготовления популярного салата оливье, то с большой вероятностью там будет и третий продукт из этого рецепта.

Однако если примерно в 50 % транзакций встречается хлеб, то окажется, что в таком же проценте транзакций любой товар соседствует с хлебом: (стиральный порошок, хлеб), (яблоки, хлеб), (кетчуп, хлеб) и т.д., т.е. получаем ассоциативное правило: если $\langle \text{что угодно} \rangle$ то $\langle \text{хлеб} \rangle$, что не несет практической пользы бизнесу. В действительности интересны ассоциативные правила, которые вы-

являют, что покупатели, купившие товар X, купили и товар Y со статистически большей вероятностью, чем покупатели, не купившие X.

Поэтому на самом деле ассоциативное правило (1.12) звучит иначе:

если < условие > то с вероятностью, превышающей базовую,
< результат > ,

но в стандартной формулировке часть про вероятность для краткости опускают, хотя она и подразумевается.

Выделяют три вида правил:

- полезные правила, содержащие действительную информацию, которая ранее была не известна, но имеет логическое объяснение;
- тривиальные правила, содержащие действительную и легко объяснимую информацию, отражающую известные законы в исследуемой области, и поэтому не приносящие какой-либо пользы;
- непонятные правила, содержащие информацию, которая не может быть объяснена (такие правила или получают на основе аномальных исходных данных, или они содержат глубоко скрытые закономерности, и поэтому для интерпретации непонятных правил нужен дополнительный анализ).

Полезное правило «пиво, подгузники» – это ассоциативное правило для покупок, совершаемых отцами малышей: подгузники они покупают детям, пиво – себе. Осталось разработать стратегию, как напомнить покупателям из этого сегмента, не забывать о себе (т.е. не забывать купить пиво, хотя пришли за подгузниками).

Тривиальное правило «хлеб, молоко» – это очень часто встречающийся в транзакции набор из двух товаров. Однако, чтобы понять, что очень многие покупают в продуктовом магазине эти два товара, не нужно проводить исследование, достаточно сравнить число транзакций и объем продаж этих категорий товаров. Рекомендации по тривиальным правилам не персонифицированы. Для таких рекомендаций нет смысла строить ассоциативные правила, достаточно определить по объему продаж бестселлеры и рекомендовать их всем.

Поиск ассоциативных правил обычно выполняют в два этапа:

- в пуле имеющихся признаков A находят наиболее часто встречающиеся комбинации элементов T;
- из этих найденных наиболее часто встречающихся наборов формируют ассоциативные правила.

Для оценки полезности и продуктивности перебираемых правил используются различные частотные критерии, анализирующие встречаемость кандидата в массиве экспериментальных данных.

Важнейшими из них являются поддержка (support) и достоверность (confidence).

Support: правило $A \rightarrow T$ имеет поддержку s , если оно справедливо для s % взятых в анализ случаев:

$$support(A \rightarrow T) = P(A \cup T). \quad (1.13)$$

Например, если нас интересует поддержка (встречаемость) набора из двух элементов, то мы делим число транзакций, в которых этот набор присутствует на общее число транзакций.

Confidence: какова вероятность того, что из наличия в рассматриваемом случае условной части правила следует наличие заключительной его части (т.е. из A следует T):

$$confidence(A \rightarrow T) = P(T|A) = \frac{P(A \cup T)}{P(A)} = \frac{support(A \rightarrow T)}{support(A)}. \quad (1.14)$$

Достоверность показывает, как часто правило срабатывает для всего набора данных (всех транзакций).

Алгоритмы поиска ассоциативных правил отбирают тех кандидатов, у которых поддержка и достоверность выше некоторых порогов. Если поддержка имеет большое значение, то алгоритмы будут находить тривиальные правила. С другой стороны, низкое значение поддержки ведет к генерации огромного количества вариантов, что требует существенных вычислительных ресурсов или ведет к генерации статистически необоснованных правил.

Conviction (убедительность) – это «частотность ошибок» правила. Например, как часто покупали товар x_1 без товара x_2 и наоборот.

$$\begin{aligned} conviction(A \rightarrow T) &= \frac{1 - P(T)}{1 - \frac{P(A \cup T)}{P(A)}} = \frac{1 - P(T)}{1 - P(T|A)} = \\ &= \frac{1 - support(T)}{1 - confidence(A \rightarrow T)} \end{aligned} \quad (1.15)$$

Чем больше значение conviction превышает 1, тем лучше. Например, значение 1,5 этого показателя для совместной покупки товаров x_1 и x_2 означает, что найденное правило «если x_1 , то x_2 »

в 1,5 раза более верное по сравнению с альтернативным вариантом, что попадание этих товаров в одну транзакцию было случайным.

Еще один показатель для качества ассоциативных правил – подъемная сила, или лифт (*lift*), которая показывает, насколько повышается вероятность нахождения T в анализируемом случае, если в нем уже имеется A :

$$lift(A \rightarrow T) = \frac{confidence(A \rightarrow T)}{support(T)}. \quad (1.16)$$

Если значение *lift* для двух товаров x_1 и x_2 равно 1,5, то это означает, что правило «вместе с товаром x_1 покупают и товар x_2 » на 50 % мощнее правила, что товар x_2 просто покупают. На практике желательно, чтобы подъем был как можно больше (например, 10 и выше).

Чаще всего ассоциативные правила используют для анализа корзины. Но этим варианты использования не исчерпываются. Можно выявлять устойчивые шаблоны поведения пользователей на сайте, в цифровом продукте и т.п. И улучшать поддержку за счет рекомендаций и подсказок. Например, почтовые сервисы рекомендуют дополнительных адресатов пользователю, который пишет письмо.

2. ОБУЧЕНИЕ БЕЗ УЧИТЕЛЯ НА ЯЗЫКЕ R, RSTUDIO

2.1. Настройка окружения

Для выполнения анализа данных с помощью языка R необходимо настроить и установить R и RStudio.

Установочные файлы для языка R под различные операционные системы (Windows, Linux, MacOS) можно найти на официальном сайте проекта R: <https://cran.r-project.org/>. На момент написания пособия актуальной для Windows является версия R-3.6.1

Среда разработки (integrated development environment IDE) для языка R – RStudio. Существует две версии: RStudio Desktop и RStudio Server. Обе версии существуют в варианте open source edition и commercial license. В пособии используется RStudio Desktop open source edition. Установочные файлы можно найти на официальном сайте продукта: <https://rstudio.com/products/rstudio/>.

Краткий обзор возможностей RStudio [10].

Начальное окно RStudio представлено на рис. 2.1:

- Source editor and data view (редактор кода и обзор данных);
- R console (консоль);
- Служебная панель 1: Environment, history and connections окружение (данные, функции, история команд и доступные внешние ресурсы);
- Служебная панель 2: Files, plots, packages, help and viewer (файлы, графики, пакеты, помощь и просмотрщик).

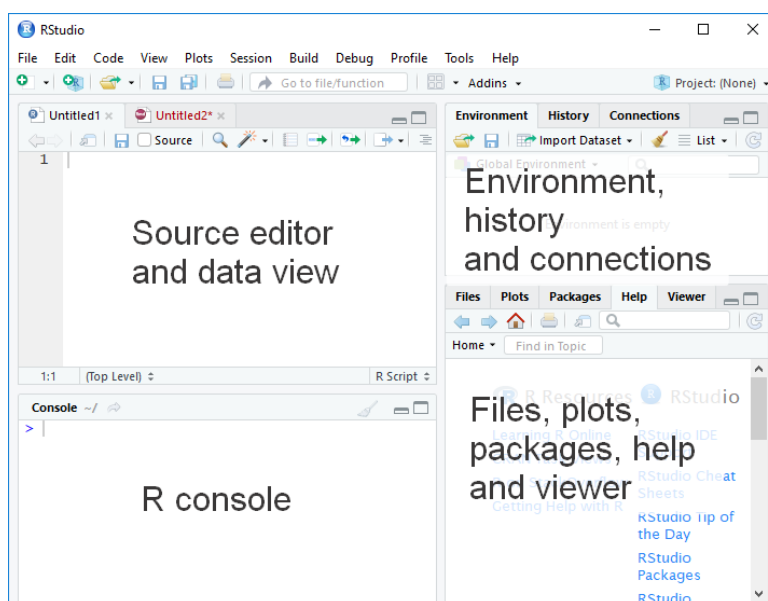
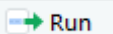


Рис. 2.1. IDE RStudio

Код предпочтительнее набирать в панели Source, открыв новый файл с помощью меню File > New file > R Script. Скрипт имеет расширение .R. На выполнение код посылается комбинацией клавиш CTRL+ENTER, или кнопкой в правом верхнем углу панели  Run.

Код RStudio поддерживает автоматическое завершение кода в окне Source при помощи клавиши TAB (рис. 2.2). Такие подсказки можно получить и при наборе аргументов, если используются ранее определенные переменные.

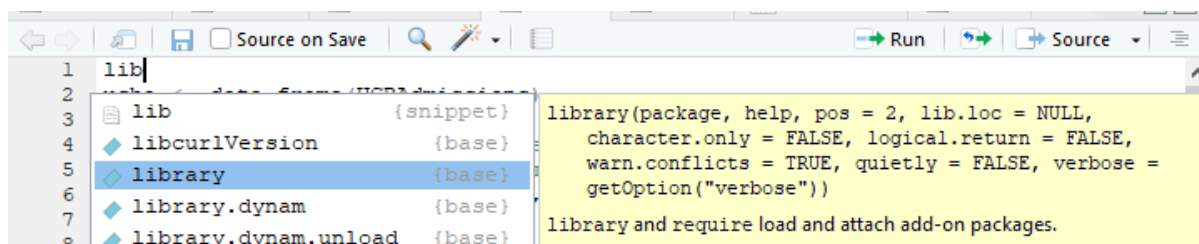


Рис. 2.2. Завершение функции library() с помощью клавиши TAB

Открыть существующий файл можно с помощью File > Open или кнопки . Открыть файл, с которым недавно работали, можно с помощью File > Recent Files или кнопки раскрыть список на панели инструментов (рисунок 2.3).

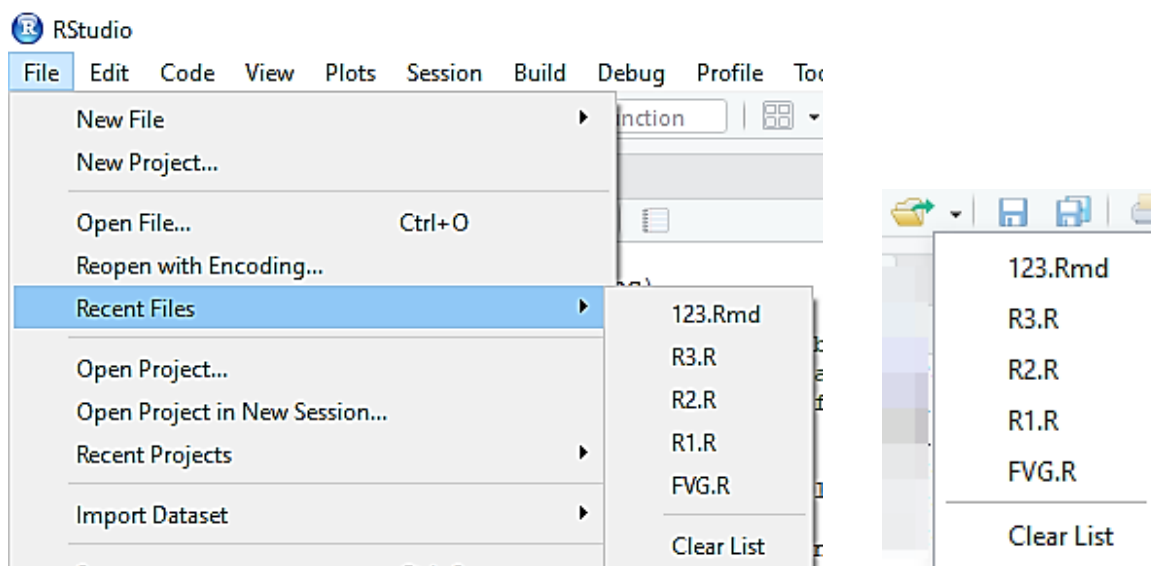


Рис. 2.3. Список недавно использовавшихся файлов

При закрытии RStudio программа предлагает сохранить код в открытых файлах текущей сессии и рабочее пространство (Workspace), которое включает в себя созданные переменные, историю команд (рис. 2.4).

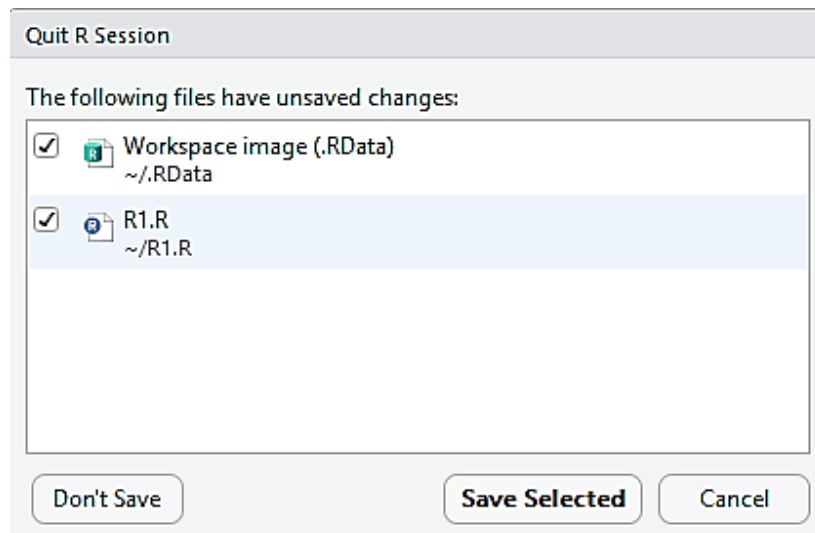
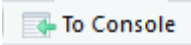
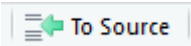


Рис. 2.4. Завершение работы RStudio

При открытии RStudio загружается сохраненное рабочее пространство со всеми незакрытыми в прошлой сохраненной сессии файлами, созданными переменными, функциями и историей команд.

Вкладка History (рис. 2.5) содержит журнал всех ранее выполненных в рабочем пространстве команд (в окне Console). Они сохраняются и после очистки окна Console. Эти команды можно передавать на выполнение в Console кнопкой , команды выполняются после нажатия ENTER. Или скопировать в открытый файл в окне Source (если открытого файла нет, то создается новый) кнопкой . Для работы с несколькими командами их нужно предварительно выделить.

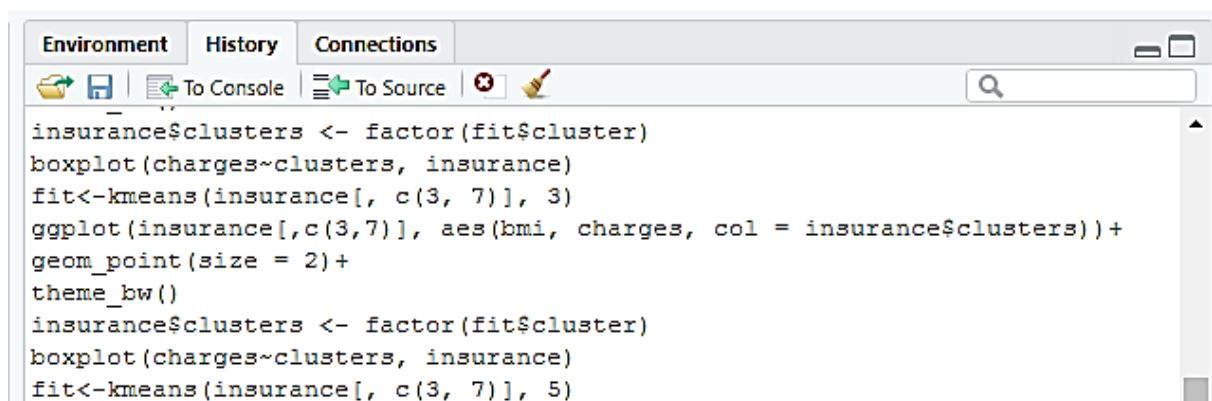


Рис. 2.5. История команд

Вкладка Environment (рис. 2.6) содержит описание всех объектов (это могут быть пользовательские функции, переменные), которые были созданы ранее в рабочем пространстве.

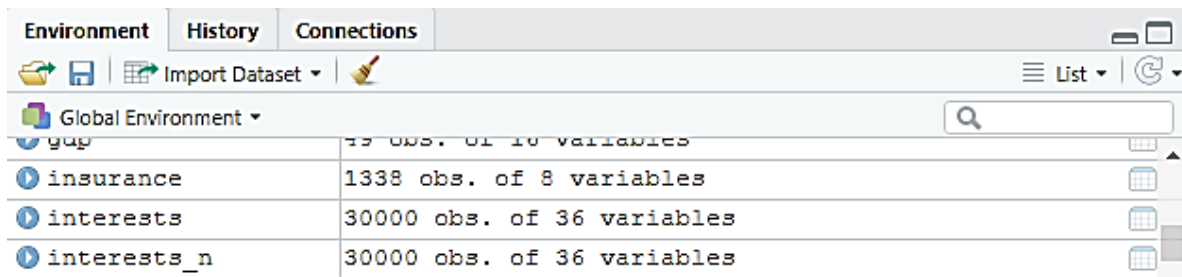


Рис. 2.6. Окружение

Текущую рабочую директорию (папку) можно узнать с помощью функции `getwd()`

Файлы текущей рабочей директории отображаются в окне вкладки Files (рис. 2.7).

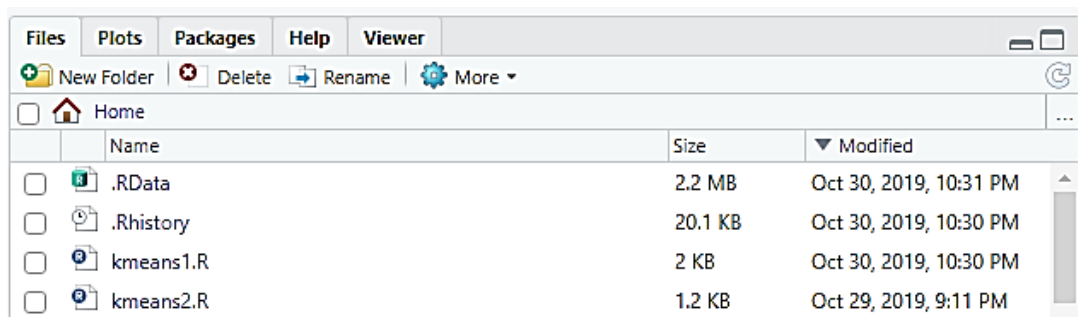


Рис. 2.7. Файлы текущей рабочей директории

Текущую рабочую папку можно поменять, например, с помощью кнопки на вкладке Files (рис. 2.8) или с помощью меню Tools > Change Working Dir.

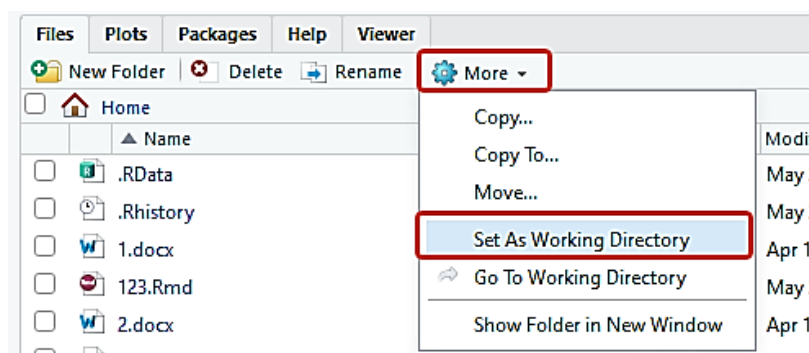


Рис. 2.8. Смена рабочей папки

Возможные последствия изменения рабочей папки:

– имеющиеся в коде относительные ссылки на файлы станут недействительными;

– при завершении работы файл `.RData` будет сохранен в новую папку.

Установка новых пакетов, необходимых в данной рабочей сессии, возможна на вкладке `Packages` с помощью кнопки `Install` (рис. 2.9). При наборе имени пакета подключается автозаполнение.

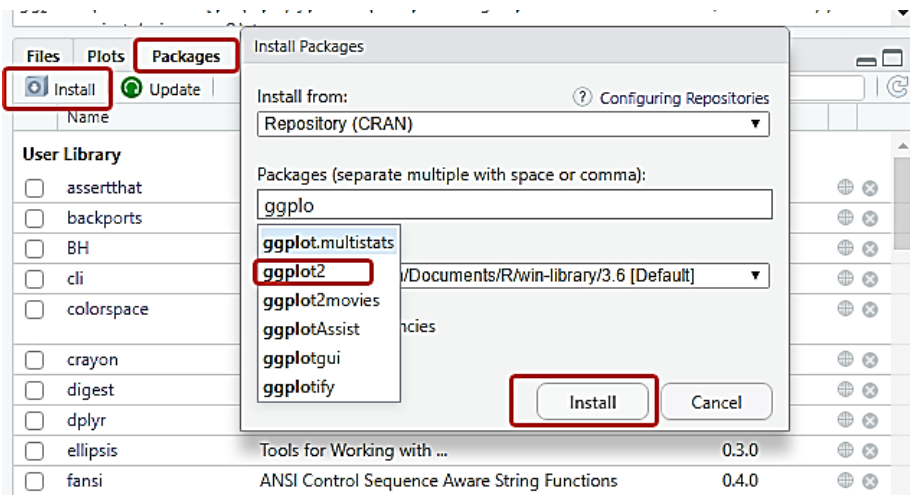


Рис. 2.9. Установка пакетов

Флаг на опции `Install dependencies` означает, что установятся и все связанные пакеты, на функции которых опирается нужный пакет (рис. 2.10).

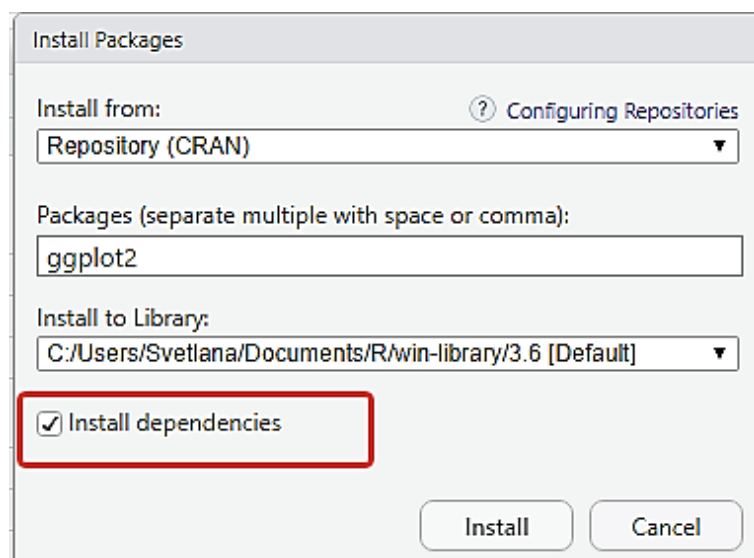


Рис. 2.10. Установка всех связанных пакетов

Установка новых пакетов возможна и с помощью команды, аргументы которой – это имя пакета и значение `true` для переменной `dependencies`, чтобы установились связанные пакеты

```
install.packages('ggplot2', dependencies = T)
```

Установленные пакеты нужно подключить в текущей рабочей сессии, чтобы можно было использовать определенные в них функции и имеющиеся в пакетах наборы данных. Это можно сделать на вкладке Packages, установив флаг на нужном пакете (рис. 2.11).

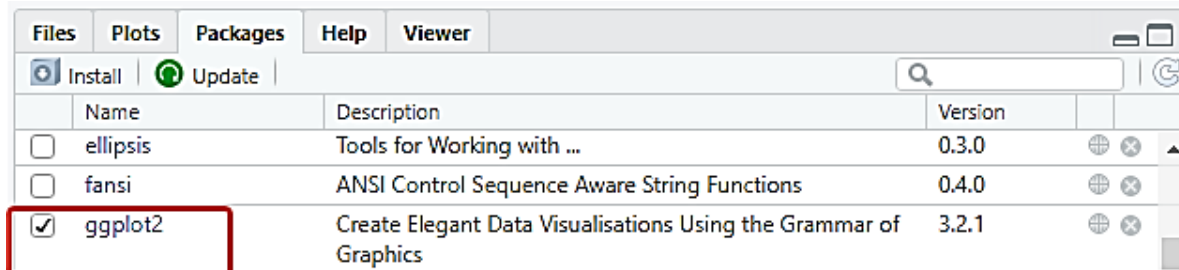


Рис. 2.11. Подключение пакета в текущей сессии

Или с помощью команды

```
library(ggplot2)
```

Но подключение пакета с помощью команды не гарантирует, что такой пакет уже был установлен (и подключение не выполнится), тогда как в первом варианте недоступные для подключения пакеты просто не отображаются.

На вкладке Help можно получить справку по пакетам, функциям, наборам данных и т.п. Помощь доступна и с помощью знака вопроса перед объектом, о котором нужна справка

```
?library()
```

На вкладке Plots отображаются построенные в текущей сессии графики: 1 – между графиками можно перемещаться, 2 – увеличивать размер графика, 3 – сохранять график в доступном формате (рис. 2.12).

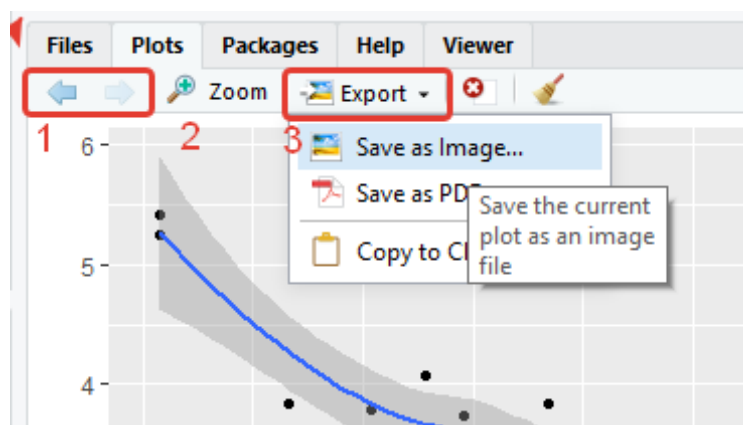


Рис. 2.12. Операции с графиками

2.2. Метод k-средних

В R для реализации метода k-средних необходимо импортировать пакет `cluster`, подключить его в текущей сессии и воспользоваться функцией `kmeans()` (аргументы функции в табл. 2.1, результат в табл. 2.2).

```
install.packages('cluster')
library(cluster)
fit <- kmeans(X, 4, nstart=20)
```

Таблица 2.1

Аргументы `kmeans()`

<code>x</code>	Числовая матрица данных или объект, который может быть принужден к такой матрице (например, числовой вектор или фрейм данных со всеми числовыми столбцами)
<code>centers</code>	Либо число кластеров, либо начальные значения для центроидов кластеров. Если задается число кластеров, то в качестве начальных центров выбирается случайный набор (различных) строк в <code>X</code>
<code>iter.max</code>	Максимальное количество итераций
<code>nstart</code>	Используется, чтобы запустить алгоритм функции <code>kmeans()</code> несколько раз с разными начальными метками центроидов, а итог – наилучший результат нескольких реализаций (значение больше 20 помогает избежать получения локального минимума целевой функции <code>tot.withinss</code>)

Таблица 2.2

Результат `kmeans()`

<code>cluster</code>	Вектор целых чисел, указывающий кластер, в который попала каждое наблюдение каждая точка
<code>centers</code>	Матрица кластерных центров
<code>withinss</code>	Вектор внутрикластерной суммы квадратов расстояний от центроидов, по одному компоненту на кластер
<code>tot.withinss</code>	Общая внутрикластерная сумма квадратов, т.е. сумма (<code>withinss</code>), которую в методе k-средних минимизируют
<code>size</code>	Количество точек в каждом кластере

```
# Выведем значения всех переменных результата функции kmeans()
fit
# Выведем значения отдельных переменных результата функции kmeans()
fit$cluster
fit$centers
fit$size
```

Рассмотрим реализацию метода на нескольких примерах.

2.2.1. Набор данных *ирисы*

Первый набор данных – классика всех обучающих курсов наряду с данными о пассажирах Титаника. Это набор данных *ирисы Фишера* (входит в пакет `base` имя `dataset – iris`) содержит 150 наблюдений.

Описание переменных набора:

- длина чашелистика (`sepal length`);
- ширина чашелистика (`sepal width`);
- длина лепестка (`petal length`);
- ширина лепестка (`petal width`);
- класс (`Species`), соответствующий одному из трех видов: *Iris Setosa*, *Iris Versicolor* или *Iris Virginica*.

```
# Скопируем набор iris в новую переменную
df_iris <- iris
# Инсталлируем пакет ggplot2
install.packages("ggplot2", dependencies = T)
# Подключим пакет ggplot2 в текущий сеанс работы
library(ggplot2)
# Применим метод кластерного анализа с числом кластеров 3
# к первому и четвертому столбцу фрейма данных df_iris
fit <- kmeans(df_iris[, c(1,4)], 3)
# Добавим в исходный набор новый столбец с номером кластера наблюдения
df_iris$clusters <- factor(fit$cluster)
# Добавим в исходный набор новый столбец,
# преобразовав название класса в его номер
df_iris$spec <- factor(iris$Species)
# Построим график
# (вид маркера pch определяется номером кластера, цвет номером класса)
ggplot(df_iris[,c(1,4)], aes(Sepal.Length, Petal.Width, col = df_iris$spec))+
  geom_point(pch = df_iris$clusters, size = 2)+
  theme_bw()
# Проверим, насколько полученные кластеры совпадают с классами цветков
table(df_iris$clusters, df_iris$Species)
```

График показывает, насколько верно определились классы по длине чашелистика и ширине листка (рис. 2.13). В силу случайного выбора центроидов на первом шаге при выполнении кода может получиться несколько иная картинка, и значения в таблице сопряженности двух переменных `clusters` и `Species` также могут быть иными (рис. 2.14).

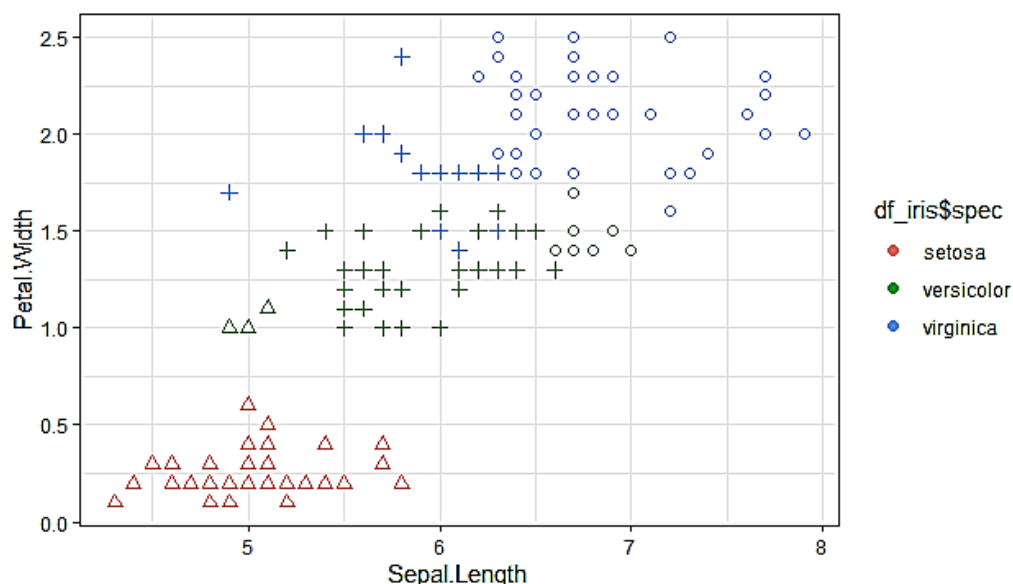


Рис. 2.13. Распределение ирисов по классам и кластерам

```
> table(df_iris$clusters, df_iris$Species)

      setosa versicolor virginica
1          0           7         35
2         50           4          0
3          0          39         15
```

Рис. 2.14. Анализ качества кластеризации

Как видим, растения класса `setosa` оказались во втором кластере, но в него же ошибочно попали и четыре растения класса `versicolor`, первый кластер преимущественно состоит из растений класса `virginica`, а третий – из растений класса `versicolor`.

Попробуем использовать для кластеризации всю доступную информацию, но график, как и в первом случае, построим для двух переменных `Sepal.Length`, `Petal.Width`.

```
# Скопируем первые четыре столбца исходного набора в новую переменную
iris_k <- iris[, 1:4]

# Применим метод кластерного анализа с числом кластеров 3 к iris_k
fit <- kmeans(iris_k, 3)
```

```
# Изменим значения в столбце с кластером наблюдения на новые
df_iris$clusters <- factor(fit$cluster)
# Построим график
ggplot(df_iris[, c(1,4)], aes(Sepal.Length, Petal.Width, col = df_iris$spec))+
  geom_point(pch = df_iris$clusters, size = 2)+
  theme_bw()
# Проверим насколько полученные кластеры совпадают с классами цветков
table(df_iris$clusters, df_iris$Species)
```

В результате кластеризации методом k-средних на всем объеме доступных данных (кроме метки класса) растения класса *setosa* оказались во втором кластере, и в нем нет представителей другого класса, первый кластер преимущественно состоит из растений *virginica*, а третий – из растений класса *versicolor* (рис. 2.15–2.16). Использование всей доступной информации позволило улучшить качество кластеризации.

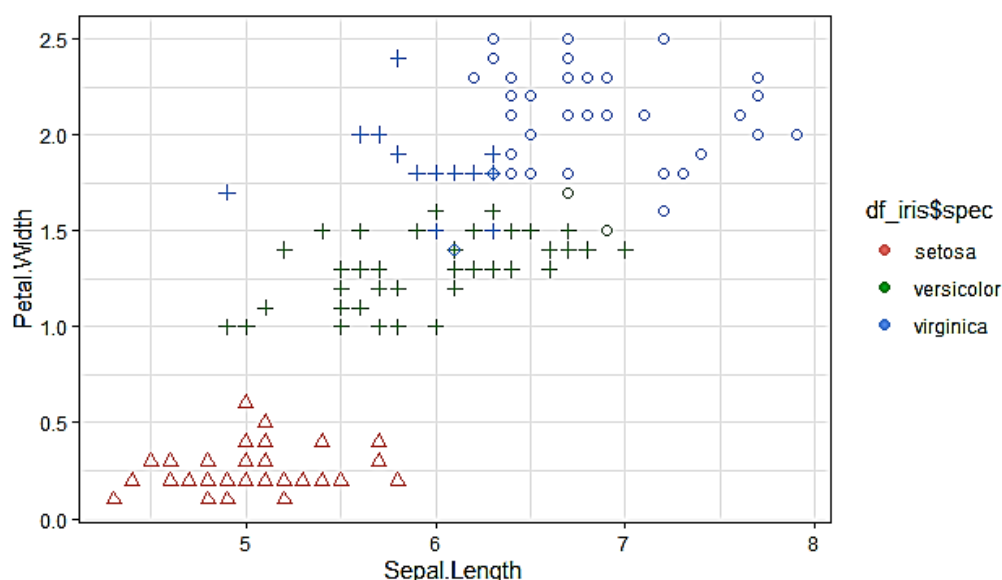


Рис. 2.15. Распределение ирисов по классам и кластерам

```
> table(df_iris$clusters, df_iris$Species)

      setosa versicolor virginica
1          0           2         36
2         50           0           0
3          0          48          14
```

Рис. 2.16. Таблица сопряженности по переменным *clusters* и *Species* для анализа качества кластеризации

Однако использование этого набора данных для кластеризации не в рамках классического подхода, так как нам изначально известно, какой класс у растений и их количество, т.е. данные являются размеченными.

2.2.2. Набор данных *медицинское страхование*

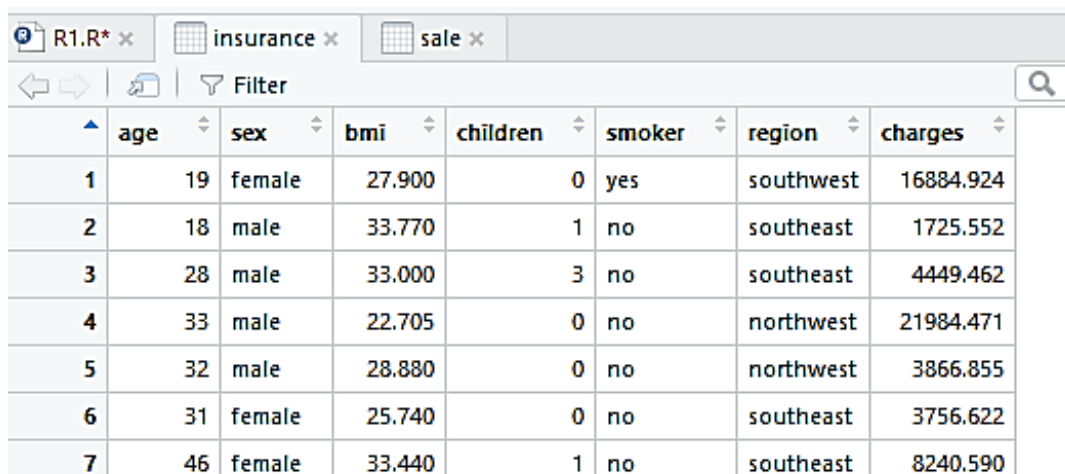
Рассмотрим теперь более корректный случай, когда изначально разметки у данных по классам нет, и мы не знаем, какое количество кластеров нужно выбрать.

Данные, которые мы используем, – расходы на медицинское обслуживание тех, кто имеет медицинскую страховку: <https://www.kaggle.com/mirichoi0218/insurance>.

Описание переменных набора:

- age: возраст основного бенефициара;
- sex: пол застрахованного;
- bmi: индекс массы тела;
- children: число детей, охваченных медицинским страхованием / число иждивенцев;
- smoker: курит ли застрахованный;
- region: жилой район получателя в США, Северо-Восток, Юго-Восток, Юго-Запад, Северо-Запад.
- charges: индивидуальные медицинские расходы, оплачиваемые страховкой.

Первые строки набора представлены на рис. 2.17.



	age	sex	bmi	children	smoker	region	charges
1	19	female	27.900	0	yes	southwest	16884.924
2	18	male	33.770	1	no	southeast	1725.552
3	28	male	33.000	3	no	southeast	4449.462
4	33	male	22.705	0	no	northwest	21984.471
5	32	male	28.880	0	no	northwest	3866.855
6	31	female	25.740	0	no	southeast	3756.622
7	46	female	33.440	1	no	southeast	8240.590

Рис. 2.17. Набор данных insurance

Не забываем подключить в новом сеансе работы все используемые пакеты.

```

library(ggplot2)
# Загружаем файл с расходами на медицинское обслуживание
insurance <- read.table("insurance.csv", header=T, sep=",")
# Копируем набор данных в новую переменную для преобразования
insurance_num <- insurance
# Преобразуем все факторы в числовые переменные
insurance_num$sex <- as.numeric(insurance$sex)
insurance_num$smoker <- as.numeric(insurance$smoker)
insurance_num$region <- as.numeric(insurance$region)
# Применим метод кластерного анализа с числом кластеров 3 ко всему набору
fit <- kmeans(insurance_num, 3)
# Добавим в исходный набор новый столбец с номером кластера
insurance$clusters <- factor(fit$cluster)
# Построим график "ящик с усами"
# для расходов на медицинское обслуживание в трех разных группах
boxplot(charges~clusters, insurance)
# Оценим, как распределились застрахованные по трем кластерам
# по переменной smoker
table(insurance$smoker, insurance$clusters)

```

Как видим, расходы на медицинское обслуживание в трех кластерах существенно различаются: застрахованные во втором кластере имеют самые небольшие расходы, в третьем – самые значительные (рис. 2.18).

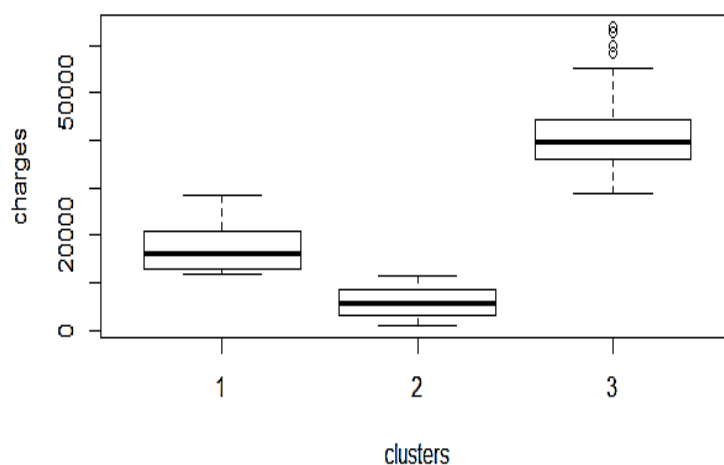


Рис. 2.18. Распределение застрахованных по кластерам в зависимости от расходов на медицинское обслуживание (3 кластера)

Распределение курящих и некурящих застрахованных по кластерам также обладает определенной закономерностью: курящие оказались в группах с большими расходами (1 и 3), причем в кластере с самыми существенными медицинскими расходами их – подавляющее большинство (рис. 2.19). Во втором кластере нет ни одного курящего.

```
> table(insurance$smoker, insurance$clusters)

      1    2    3
no   235 817  12
yes  117   0 157
> |
```

Рис. 2.19. Распределение застрахованных по кластерам в зависимости от наличия вредной привычки (3 кластера)

Количество кластеров было выбрано равным трем, как в предыдущем примере. Но, возможно, что оно выбрано ошибочно. Хотя на первый взгляд смысловое содержание в таком разбиении определено есть.

Выясним наилучшее количество кластеров с помощью «метода локтя».

```
# Инициализируем переменную, в которую будем записывать fit$tot.withinss
km <- c()

# В цикле проведем кластерный анализ для различного числа кластеров,
# присоединяя к вектору km соответствующие значения
# общей внутрикластерной суммы квадратов
for (k in 1:10) {
  fit <- kmeans(insurance_num, k)
  km <- c(km, fit$tot.withinss)}

# Построим график (тип - сплошная линия) по значениям вектора km
plot(km, type="l")

# Нанесем на график маркеры точек
points(km, pch=2)
```

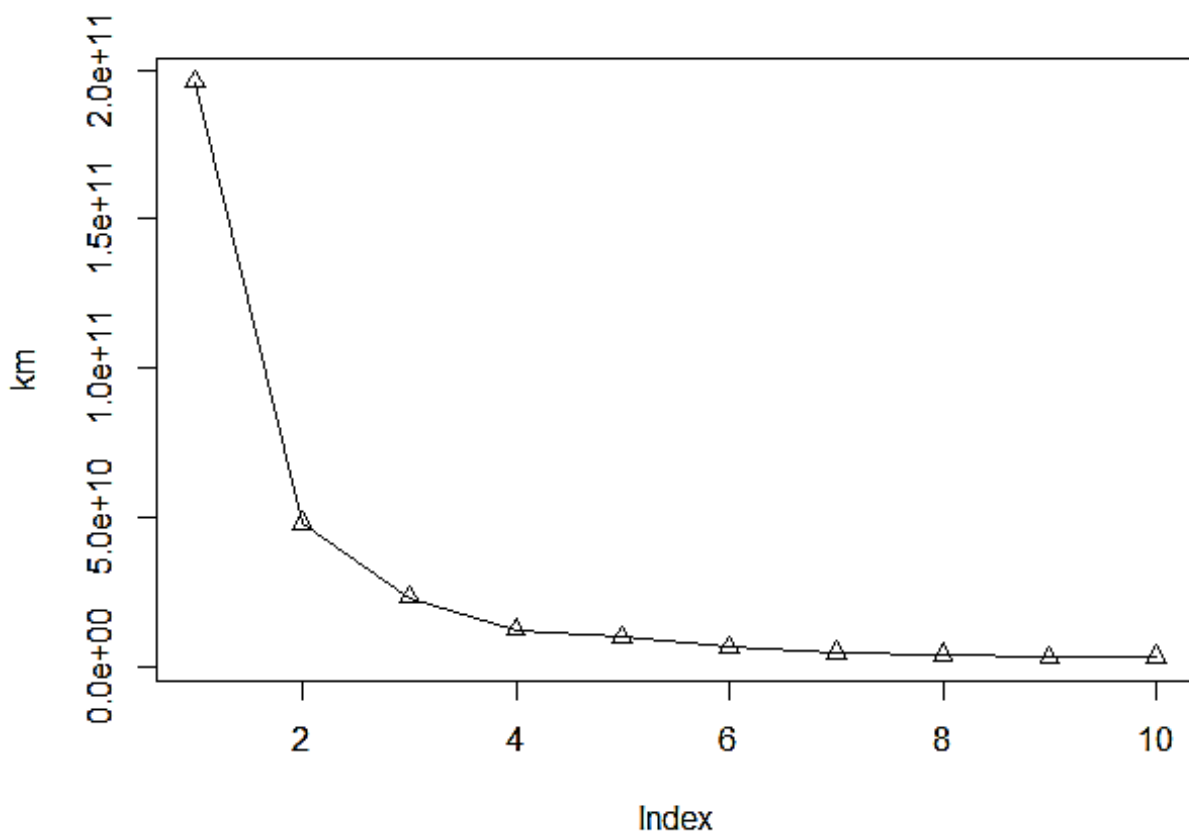


Рис. 2.20. График для выявления оптимального числа кластеров (insurance)

Судя по графику (рис. 2.20), можно попробовать разбиение на 4 кластера.

```
# Применим метод кластерного анализа с числом кластеров 4 к исх. набору
fit <- kmeans(insurance_num[, 1:7], 4)
# Изменим в наборе значения столбца с номером кластера
insurance$clusters <- factor(fit$cluster)
```

Построим график «ящик с усами» для четырех кластеров (рис. 2.21) и оценим распределение застрахованных по кластерам для переменной *smoker* (рис. 2.22).

```
# Построим график "ящик с усами"
# для расходов на медицинское обслуживание в четырех разных группах
boxplot(charges~clusters, insurance)
# Оценим, как распределились застрахованные по трем кластерам
# по переменной smoker
table(insurance$smoker, insurance$clusters)
```

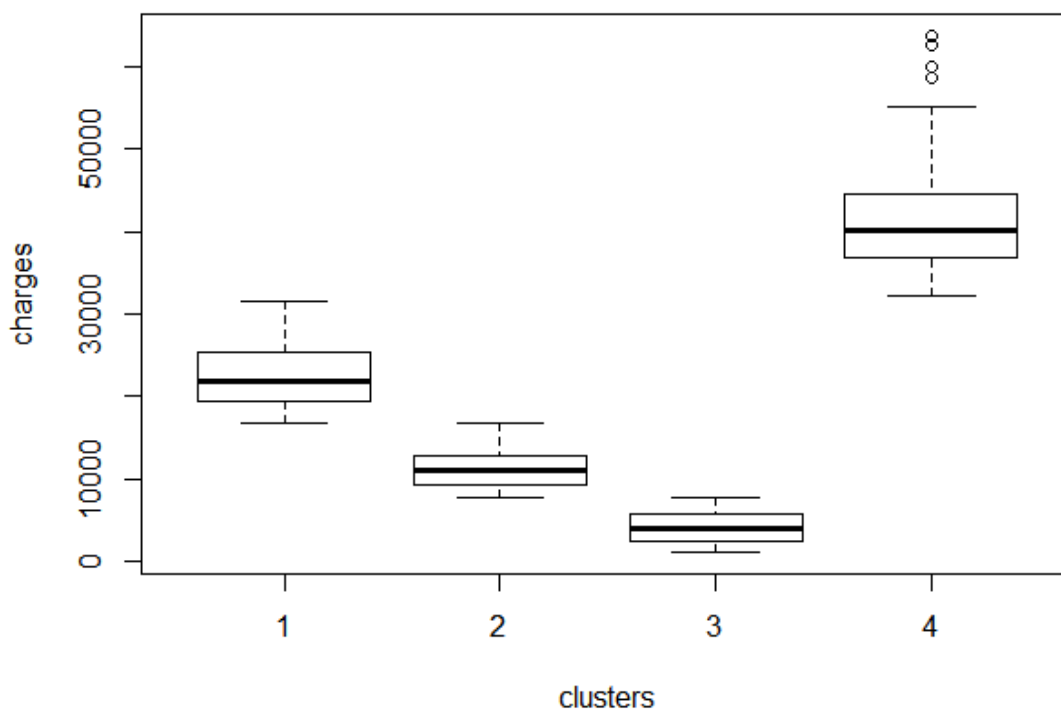


Рис. 2.21. Распределение застрахованных по кластерам в зависимости от расходов на медицинское обслуживание (4 кластера)

```
> table(insurance$smoker, insurance$clusters)
```

	1	2	3	4
no	74	429	556	5
yes	103	21	0	150

Рис. 2.22. Распределение застрахованных по кластерам в зависимости от наличия вредной привычки (4 кластера)

2.2.3. Набор данных *смс*

И в заключение рассмотрим задачу кластерного анализа методом *k*-средних для *смс*. Это данные, полученные на основе методов ОЕЯ – обработки естественного языка. Пример рассмотрен в [11, с. 279–290]. Подготовленные данные возьмем из открытого репозитория на github: <https://github.com/stedy/Machine-Learning-with-R-datasets> Набор данных *snsdata.csv*.

Описание переменных набора:

- *gradyear* – год выпуска;
- *gender* – пол;
- *age* – возраст;
- *friends* – число друзей в социальной сети;

– basketball, football, soccer, softball, volleyball, swimming, cheerleading, baseball, tennis, sports, cute, sex, sexy, hot, kissed, dance, band, marching, music, rock, god, church, jesus, bible, hair, dress, blonde, mall, shopping, clothes, hollister , abercrombie, death, drunk, drugs (переменные с 5 по 40) – частота встречаемости соответствующих слов в sms выпускника.

С точки зрения машинного обучения исходный текст сообщений бесполезен, его нужно преобразовать, чтобы использовать, например, алгоритмы кластеризации.

Для преобразования текста сообщений в форму, пригодную для машинного обучения, можно использовать вариант редакционного расстояния – набор слов. В этом подходе порядок слов полностью игнорируется, а в основу кладутся просто счетчики слов. Этот процесс называется векторизацией. Векторы получаются очень большими, так как в каждый вектор входит столько элементов, сколько есть слов во всем наборе данных.

Данные *смс* были предварительно обработаны: вначале были отобраны слова, которые встречались в сообщениях максимально часто, из них были выбраны слова, характеризующие пять групп интересов подростков: религия, романтические отношения, мода, внеклассные занятия (хобби) и асоциальные интересы. Таких слов-индикаторов интересов оказалось 36. Переменная, соответствующая конкретному слову, показывает, как часто оно встречается в сообщениях определенного выпускника.

Данные обезличенные, в них есть пропуски и выбросы.

На первом этапе займемся подготовкой данных:

```
# Загружаем файл со словами из смс
df_sms <- read.table("snsdata.csv", sep=";", header=T)
# Проводим разведочный анализ данных (находим описательные статистики)
summary(df_sms)
```

Как видим из результатов анализа (рис. 2.23), набор данных не сбалансирован по полу (gender): девушек примерно в четыре раза больше (среди тех, чей пол известен) и 9 % наблюдений имеют значение переменной NA – Not Available (недоступен), так обозначается пропущенное или отсутствующее значение. NA – это служебная аббревиатура и, как и в случае с логическими значениями для аргументов функций TRUE, FALSE, кавычки не нужны.

```
> summary(df_sms)
  gradyear  gender  age  friends  basketball
Min.   :2006   F   :22054 Min.   : 3.086 Min.   : 0.00 Min.   : 0.0000
1st Qu.:2007   M   : 5222 1st Qu.: 16.312 1st Qu.: 3.00 1st Qu.: 0.0000
Median :2008  NA's: 2724 Median : 17.287 Median : 20.00 Median : 0.0000
Mean   :2008                Mean   : 17.994 Mean   : 30.18 Mean   : 0.2673
3rd Qu.:2008                3rd Qu.: 18.259 3rd Qu.: 44.00 3rd Qu.: 0.0000
Max.   :2009                Max.   :106.927 Max.   :830.00 Max.   :24.0000
      NA's :5086

  football  soccer  softball  volleyball
Min.   : 0.0000 Min.   : 0.0000 Min.   : 0.0000 Min.   : 0.0000
1st Qu.: 0.0000 1st Qu.: 0.0000 1st Qu.: 0.0000 1st Qu.: 0.0000
Median : 0.0000 Median : 0.0000 Median : 0.0000 Median : 0.0000
Mean   : 0.2523 Mean   : 0.2228 Mean   : 0.1612 Mean   : 0.1431
3rd Qu.: 0.0000 3rd Qu.: 0.0000 3rd Qu.: 0.0000 3rd Qu.: 0.0000
Max.   :15.0000 Max.   :27.0000 Max.   :17.0000 Max.   :14.0000
```

Рис. 2.23. Описательные статистики для набора данных df_sms (начало)

Для переменной возраст (age) обнаружили явно аномальные значения: минимальное значение – 3 года, максимальное значение – 106 лет, у 5086 наблюдений значение соответствующей переменной – NA (см. рис. 2.23).

```
# Заменяем аномальные значения возраста выпускников на NA
df_sms$age <- ifelse(df_sms$age >= 13 & df_sms$age <= 20, df_sms$age, NA)
# Найдём для каждого наблюдения среднее значение возраста
# в каждом выпуске (gradyear)
# значение параметра функции mean() na.rm=TRUE даёт указание
# игнорировать NA (rm это remove missing – удалить отсутствующие)
ave_age <- ave(df_sms$age, df_sms$gradyear, FUN=function(x) mean(x,
na.rm=TRUE))
# Заменяем отсутствующие значения возраста выпускников
# на среднее значение возраста в данных сгруппированных по году выпуска
df_sms$age <- ifelse(is.na(df_sms$age), ave_age, df_sms$age)
```

Проведём кластеризацию с использованием метода k-средних, предварительно подготовив данные для анализа. Единообразие данных, которые используются для кластерного анализа, позволяет улучшить его результаты.

В стандартизованных данных среднее значение для всех переменных набора будет нулевым (рис. 2.24).

```
# Копируем столбцы набора данных, соответствующие словам, в новую переменную
interests <- df_sms[,5:40]
# Проводим стандартизацию данных с помощью функции scale(), приводящей
```

```
# данные к виду: mean 0, sd 1
# функция lapply() позволяет применить функцию,
# передаваемую в нее в качестве аргумента,
# к каждой переменной набора данных отдельно
interests_n <- as.data.frame(lapply(interests, scale))
# Найдем описательные статистики для стандартизованного набора данных
summary(interests_n)
```

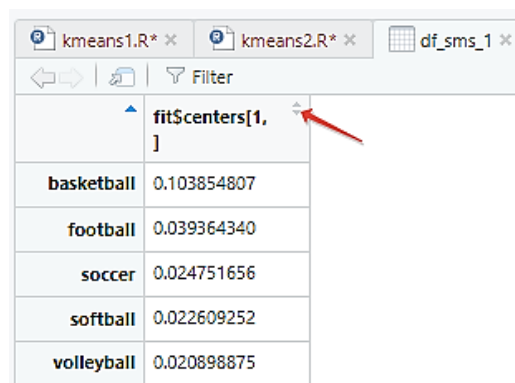
basketball	football	soccer	softball
Min. :-0.3322	Min. :-0.3577	Min. :-0.2429	Min. :-0.2179
1st Qu.:-0.3322	1st Qu.:-0.3577	1st Qu.:-0.2429	1st Qu.:-0.2179
Median :-0.3322	Median :-0.3577	Median :-0.2429	Median :-0.2179
Mean : 0.0000	Mean : 0.0000	Mean : 0.0000	Mean : 0.0000
3rd Qu.:-0.3322	3rd Qu.:-0.3577	3rd Qu.:-0.2429	3rd Qu.:-0.2179
Max. :29.4923	Max. :20.9081	Max. :29.1937	Max. :22.7642

Рис. 2.24. Описательные статистики для стандартизованного набора данных interests_n (начало)

Частоту использования в *смс* отдельных слов для кластера наиболее корректно характеризует центроид соответствующего кластера.

```
# Применим метод кластерного анализа с числом кластеров 5
fit <- kmeans(interests_n, 5)
# Выясним, какая частота слов для наблюдений, попавших в первый кластер
df_sms_1 <- as.data.frame(fit$centers[1, ])
```

Из окружения (Environment) можно вывести набор данных df_sms_1 на просмотр в панель (data view). По столбцу fit\$centers [1,] их можно упорядочить и получить список самых частотных слов для первого кластера (рис. 2.25).



	fit\$centers[1,]
basketball	0.103854807
football	0.039364340
soccer	0.024751656
softball	0.022609252
volleyball	0.020898875

Рис. 2.25. Частотность слов в первом кластере, упорядоченная по убыванию частоты (начало)

Выполним операции упорядочения данных для центроида первого кластера (рис. 2.26).

```
#Добавим столбец с названиями строк
df_sms_1$names <- row.names(df_sms_1)
#Переименуем столбцы
colnames(df_sms_1) <- c('frequency','word')
#Упорядочим данные по убыванию (decreasing = TRUE)
#значений столбца frequency
df_sms_1 <- df_sms_1[order(df_sms_1$frequency, decreasing = TRUE),]
#Выведем на экран первые 10 строк упорядоченных данных
head(df_sms_1, 10)
```

```
> head(df_sms_1, 10)
      frequency      word
bible    4.59710682    bible
jesus    2.56579405    jesus
god       2.47451053     god
church    1.90489482   church
death     0.25519217   death
die       0.20906155     die
music     0.19486694    music
basketball 0.10385481 basketball
rock      0.08986874     rock
band      0.08119947     band
~ |
```

Рис. 2.26. Частотность слов в первом кластере, упорядоченная по убыванию частоты (первые 10 строк)

Аналогичные действия выполним для 2–5 кластеров (рис. 2.27–2.28).

<pre>> head(df_sms_2, 10) frequency word marching 4.60095298 marching band 3.38634877 band music 0.39085683 music rock 0.17264425 rock dress 0.06819260 dress swimming 0.04999794 swimming dance 0.04994361 dance god 0.03739352 god football 0.03504416 football death 0.03424762 death</pre>	<pre>> head(df_sms_3, 10) frequency word blonde -0.02808224 blonde tennis -0.03684333 tennis jesus -0.07117564 jesus death -0.07265004 death soccer -0.07290284 soccer baseball -0.07668648 baseball drunk -0.08491889 drunk sexy -0.08532494 sexy die -0.09017304 die swimming -0.09275655 swimming</pre>
---	---

Рис. 2.27. Частотность слов во втором и третьем кластерах, упорядоченная по убыванию частоты (первые 10 строк)

```

> head(df_sms_4, 10)
      frequency      word
shopping 0.8881388 shopping
mall     0.6708061      mall
cute     0.6700092      cute
hollister 0.6140679 hollister
abercrombie 0.5750249 abercrombie
dance     0.5358169      dance
basketball 0.5197398 basketball
football  0.5126637 football
hot       0.5107753      hot
clothes   0.4945776      clothes

> head(df_sms_5, 10)
      frequency      word
kissed 2.9781312 kissed
drugs   2.6731162 drugs
hair    2.5892728 hair
sex     2.0346259 sex
drunk   1.7750559 drunk
die     1.6804750 die
clothes 1.3034030 clothes
rock    1.2196681 rock
music   1.2040770 music
death   0.9476039 death

```

Рис. 2.28. Частотность слов в четвертом и пятом кластерах, упорядоченная по убыванию частоты (первые 10 строк)

Так как данные стандартизованные, то положительные значения частоты встречаемости слов в *смс* выпускника говорят о том, что слова встречаются чаще, чем в среднем (частота равна нулю) в наборе данных, а меньше нуля, что встречаются реже, чем в среднем по набору данных.

Представители пятого кластера имеют асоциальную направленность интересов, выявить значимые интересы у представителей третьего кластера не удалось (слишком размыты или отсутствуют).

Проанализируем полученные кластеры по таким переменным, как возраст, количество друзей и пол.

```

# Добавим в исходный набор новый столбец с номером кластера
df_sms$cluster <- as.factor(fit$cluster)
# В сгруппированных по номеру кластера данных найдем средний возраст
aggregate(data = df_sms, age ~ clusters, mean)

```

Видимых различий в возрасте для представителей выделенных кластеров не наблюдается (рис. 2.29).

```

> aggregate(data = df_sms, age ~ clusters, mean)
  clusters      age
1         1 18.53289
2         2 18.06595
3         3 18.12605
4         4 17.47294
5         5 17.54657

```

Рис. 2.29. Средний возраст для выпускников в каждом кластере

```

# В сгруппированных по номеру кластера данных найдем среднее число друзей
aggregate(data = df_sms, friends ~ clusters, mean)

```

```
> aggregate(data = df_sms, friends ~ clusters, mean)
  clusters friends
1         1  34.90610
2         2  32.96398
3         3  27.85921
4         4  39.04484
5         5  30.52627
```

Рис. 2.30. Среднее число друзей для выпускников в каждом кластере

```
# Проанализируем распределение выпускников по полу в кластерах 1–5 table
(df_sms$gender, df_sms$cluster)
```

```
> table (df_sms$gender, df_sms$cluster)
```

	1	2	3	4	5
F	454	588	15641	4509	862
M	111	169	4427	370	145

Рис. 2.31. Разделение по полу выпускников в каждом кластере

В четвертом кластере преобладающие представители – девушки (рис. 2.31), и у них в среднем больше друзей (рис. 2.30). В третий кластер попало очень много выпускников. Это самый представительный кластер.

Общее число наблюдений в каждом кластере находится в переменной `fit$size` (рис. 2.32).

```
> fit$size
[1] 607 805 22303 5219 1066
```

Рис. 2.32. Число наблюдений в каждом кластере

Сумма по столбцам в кластерах, разбитых на части по полу, не совпадает с количеством наблюдений в кластере из-за наличия в переменной `df_sms$gender` значений NA.

2.3. Метод главных компонент

В R для реализации метода главных компонент можно воспользоваться функцией `prcomp()` (аргументы функции см. в табл. 2.3, результат – в табл. 2.4).

```
fit <- prcomp(x, scale=TRUE)
```

Таблица 2.3

Аргументы prcomp()

x	Числовая матрица данных или фрейм данных
scale	Логическое значение, указывающее, должны ли переменные масштабироваться, чтобы иметь единичную дисперсию до начала анализа. Значение по умолчанию – FALSE

Таблица 2.4

Результат prcomp()

sdev	Стандартные отклонения главных компонент (т.е. квадратные корни собственных значений из ковариационной / корреляционной матрицы)
rotation	Матрица нагрузок главных компонент (векторы нагрузок расположены в столбцах)
center	Средние значения, используемые для стандартизации переменных перед выполнением PCA
scale	Стандартные отклонения, используемые для стандартизации переменных перед выполнением PCA
x	Матрица значений главных компонент (столбец с номером k – это вектор значений компоненты с номером k)

```
# Выведем значение стандартного отклонения каждой главной компоненты
fit$sdev
# Рассчитаем значения дисперсий каждой главной компоненты
fit$sdev^2
# Определим долю дисперсии, объясненную каждой главной компонентой
pve = fit$sdev^2/sum(fit$sdev^2)
```

Данные, которые мы используем представляют собой медицинские данные о пациентах, связанные с диабетом: <https://www.kaggle.com/kumargh/pimaindiansdiabetescsv#pima-indians-diabetes.csv>

Описание переменных набора:

- Pregnancies – число беременностей (все пациенты из источника – женщины не моложе 21 года индийской народности пима);
- Glucose – концентрация глюкозы в плазме через 2 ч после введения в пероральном глюкозотолерантном тесте;
- BloodPressure – диастолическое артериальное давление (мм рт. ст.);
- SkinThickness – толщина кожной складки в районе трицепса (мм);

- Insulin – концентрация инсулина в сыворотке крови (мкЕд/мл);
- BMI – индекс массы тела (вес в кг/(рост в м)^2);
- DiabetesPedigreeFunction – функция, описывающая генетическую предрасположенность к диабету (diabetes pedigree);
- Age – возраст (годы);
- Outcome – целевой показатель, регистрирующий, наблюдался ли у пациента сахарный диабет или нет (соответственно, 1 или 0).

```
# Считываем данные в фрейм данных df_diabetes
df_diabetes <- read.table("diabetes.csv", sep=";", header=T)
# Находим для первых восьми столбцов значения дисперсии и среднего
apply(df_diabetes[, 1:8], 2, var)
apply(df_diabetes[, 1:8], 2, mean)
```

Функция `apply(X, MARGIN, FUN)` позволяет применить любую функцию `FUN`, которая передается в нее как аргумент, к каждой строке `MARGIN=1` или каждому столбцу `MARGIN=2` матрицы или фрейма данных `X`.

Самую большую дисперсию имеют переменные `Insulin`, `Glucose` и `BloodPressure` (рис. 2.33), у этих переменных и самые большие средние значения (рис. 2.34). Можно предположить, что их вклад в главные компоненты будет наибольшим.

```
> apply(df_diabetes[, 1:8], 2, var)
```

Pregnancies	Glucose	BloodPressure	SkinThickness
1.135406e+01	1.022248e+03	3.746473e+02	2.544732e+02
Insulin	BMI	DiabetesPedigreeFunction	Age
1.328118e+04	6.215998e+01	1.097786e-01	1.383030e+02

Рис. 2.33. Дисперсии для переменных набора данных `df_diabetes`

```
> apply(df_diabetes[, 1:8], 2, mean)
```

Pregnancies	Glucose	BloodPressure	SkinThickness
3.8450521	120.8945312	69.1054688	20.5364583
Insulin	BMI	DiabetesPedigreeFunction	Age
79.7994792	31.9925781	0.4718763	33.2408854

Рис. 2.34. Средние значения для переменных набора данных `df_diabetes`

```
# Применим метод главных компонент к исходному набору, исключив столбец 9
pca <- prcomp(df_diabetes[, 1:8])
# Выведем стандартные отклонения трех первых главных компонент
pca$sdev[1:3]
# Рассчитаем PVE для трех первых главных компонент
pve = pca$sdev[1:3]^2/sum(pca$sdev^2)
```

```
pve
# Выведем матрицу нагрузок трех первых компонент
pca$rotation[,1:3]
# Выведем матрицу значений трех главных компонент (первые 10 строк)
pca$x[1:10,1:3]
```

Для набора, в котором n наблюдений и p переменных, число информативных главных компонент определяется выражением $\min(n-1, p)$. Так как набор данных содержит 768 наблюдений и в рассмотрении оставили 8 переменных из 9, то будет построено 8 главных компонент. Но для простоты анализа будем выводить информацию о первых трех главных компонентах.

Стандартные отклонения главных компонент – убывающая последовательность, которая показывает, что каждая следующая компонента объясняет все меньшую долю дисперсии наблюдений PVE (формула 1.11) (рис. 2.35–2.36). По данным PVE далее построим график «каменистой осыпи».

```
> pca$sdev[1:3]
[1] 116.00247 30.54112 19.76304
```

Рис. 2.35. Средние значения для переменных набора данных df_diabetes

```
> pve
[1] 0.88854663 0.06159078 0.02579012
`|`
```

Рис. 2.36. Средние значения для переменных набора данных df_diabetes

Векторы нагрузок главных компонент показывают, как определяется каждая компонента через исходные переменные набора (рис. 2.37), а значения новых преобразованных переменных – главных компонент для всех наблюдений находятся в матрице x результата метода (первые 10 строк для тех главных компонент на рис. 2.38).

```
> pca$rotation[,1:3]
```

	PC1	PC2	PC3
Pregnancies	-0.0020217659	0.0226488861	-0.022464900
Glucose	0.0978115765	0.9722100405	0.143428710
BloodPressure	0.0160930503	0.1419093303	-0.922467192
SkinThickness	0.0607566861	-0.0578614699	-0.307013055
Insulin	0.9931108438	-0.0946266913	0.020977302
BMI	0.0140108085	0.0469729766	-0.132444542
DiabetesPedigreeFunction	0.0005371679	0.0008168046	-0.000639983
Age	-0.0035647443	0.1401681812	-0.125454310

```
`|`
```

Рис. 2.37. Векторы нагрузок первых трех главных компонент

```
> pca$x[1:10,1:3]
      PC1      PC2      PC3
[1,] -75.714655  35.950783 -7.26078895
[2,] -82.358268 -28.908213 -5.49667139
[3,] -74.630643  67.906496 19.46180812
[4,]  11.077423 -34.898486 -0.05301779
[5,]  89.743788  2.746937 25.21285861
[6,] -80.977922  3.946887  0.64139494
[7,]   4.350959 -46.933912  9.18341780
[8,] -82.137294 -7.097954 67.48899905
[9,] 468.866389 31.528258 10.05707153
[10,] -80.193850 18.048283 -18.04976259
> |
```

Рис. 2.38. Первые 10 строк значений для первых трех главных компонент

Построим график «каменистой осыпи»

```
pve = pca$sdev^2/sum(pca$sdev^2)
```

```
plot(pve, xlab="Principal Component", ylab="Proportion of Variance Explained",
type='b')
```

График «каменистой осыпи» показывает, что даже две главные компоненты дают достаточную долю объясненной дисперсии (рис. 2.39).

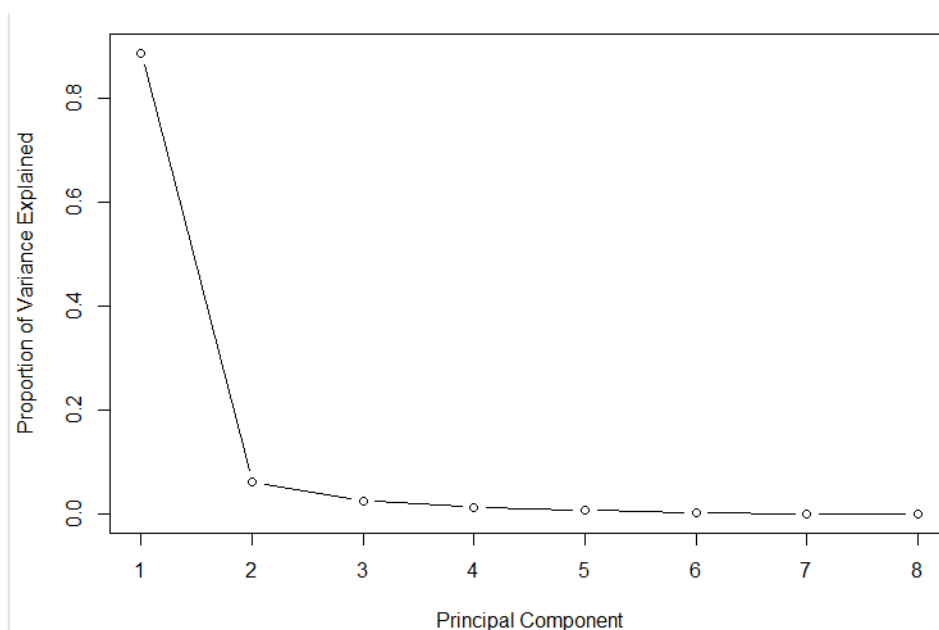


Рис. 2.39. График каменистой осыпи

2.4. Ассоциативные правила

В R для реализации поиска ассоциативных правил можно воспользоваться функцией `apriori()` из пакета `arules` (аргументы функции представлены в табл. 2.5).

```
install.packages('arules')
library(arules)
rules <- apriori(data)
```

Таблица 2.5

Аргументы `apriori()`

<code>data</code>	Данные представлены объектом класса транзакций или имеют другую структуру, которая может быть приведена к структуре транзакций (например, двоичная матрица или фрейм данных)
<code>parameter</code>	По умолчанию находятся правила с минимальной поддержкой (<code>support</code>) 0.1, минимальной уверенностью (<code>confidence</code>) 0.8, максимальной длиной 10 элементов (<code>maxlen</code>) и максимальным временем для проверки подмножества 5 с (<code>maxtime</code>). Эти значения можно изменить

Результат `apriori()` объект класса `rules` содержит выявленные правила (кортежи из элементов транзакций), для которых известны параметры качества правила: `support`, `confidence`, `lift`.

Пример рассмотрен в [11, с. 250–266]. Данные, которые мы используем, – данные транзакций покупки продуктов, в которых имеется 169 уникальных элементов товаров: <https://www.kaggle.com/irfanasrullah/groceries>

Описание переменных набора:

– каждая строка набора данных – перечисление уникальных продуктов покупки, например: `citrus fruit`, `semi-finished bread`, `margarine`, `ready soups`.

```
# Считываем данные как транзакции, удаляем при считывании дубликаты
store_data = read.transactions('groceries.csv', sep = ',', rm.duplicates = TRUE)
# Проводим разведочный анализ данных
summary(store_data)
```

Полученный набор данных – объект класса `transactions`, аргумент `rm.duplicates` по умолчанию имеет значение `FALSE` (`rm` – сокращение от `remove` – удалить), присвоенное значение `TRUE` означает, что считываются данные без дубликатов покупок.

Результат разведочного анализа данных для транзакций представлен на рис. 2.39. Перечислены уникальные продукты, упорядоченные по встречаемости в транзакциях. Например, цельное молоко (`whole milk`) присутствует в 2513 транзакциях. Также проанализирован размер транзакций: транзакция, включающая максимальное

число продуктов (32), в наборе данных одна, а транзакций, в которых покупался только один продукт – 2159.

```
> summary(store_data)
transactions as itemMatrix in sparse format with
 9835 rows (elements/itemsets/transactions) and
 169 columns (items) and a density of 0.02609146

most frequent items:
      whole milk other vegetables      rolls/buns      soda
      2513          1903          1809          1715
      yogurt          (Other)
      1372          34055

element (itemset/transaction) length distribution:
sizes
 1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
2159 1643 1299 1005  855  645  545  438  350  246  182  117  78   77   55   46
 17   18   19   20   21   22   23   24   26   27   28   29   32
 29   14   14    9   11    4    6    1    1    1    1    3    1

      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1.000  2.000   3.000   4.409  6.000  32.000
```

Рис. 2.40. Разведочный анализ для транзакций

```
# Построим на графике наиболее часто встречающиеся в транзакциях продукты
itemFrequencyPlot(store_data, topN = 10)
```

Цельное молоко встречается примерно в 25 % транзакций (2513/9835), овощи (не корнеплоды) other vegetables примерно в 20 % транзакций (1903/9835) (рис. 2.41).

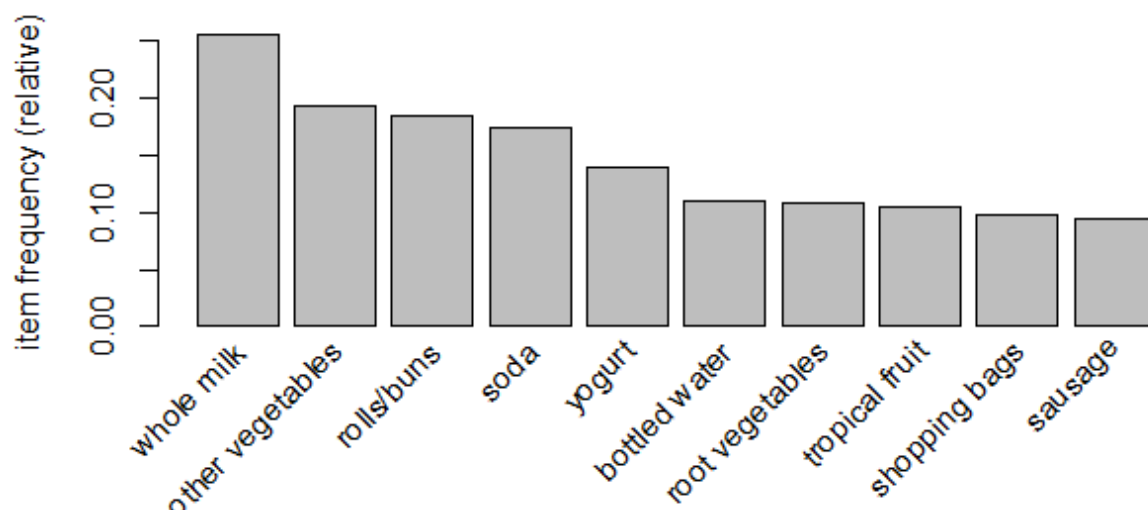


Рис. 2.41. Частота встречаемости продуктов в транзакциях

```
# Применим метод apriori к транзакциям набора store_data
rules = apriori(data = store_data, parameter = list(support = 0.001, confidence = 0.9))
rules
```

С такими значениями параметров будет найдено 129 правил. Первые три правила с характеристиками качества даны на рис. 2.42. Анализ найденных правил приведен на рис. 2.43.

```
# Выведем первые три правила, упорядоченные по убыванию параметра lift
inspect(sort(rules, by = 'lift')[1:3])
```

```
> inspect(sort(rules, by = 'lift')[1:3])
  lhs                                     rhs      support  confidence lift  count
[1] {liquor,red/blush wine}              => {bottled beer}  0.001931876  0.9047619  11.23527  19
[2] {citrus fruit,fruit/vegetable juice,other vegetables,soda} => {root vegetables} 0.001016777  0.9090909   8.34040  10
[3] {oil,other vegetables,tropical fruit,whole milk,yogurt}   => {root vegetables} 0.001016777  0.9090909   8.34040  10
```

Рис. 2.42. Частота встречаемости продуктов в транзакциях

```
# Проведем анализ правил
summary(rules)
```

```
> summary(rules)
set of 129 rules

rule length distribution (lhs + rhs):sizes
 3  4  5  6
10 57 56  6

      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      3.00   4.00   4.00   4.45   5.00   6.00

summary of quality measures:
      support      confidence      lift      count
Min.   :0.001017  Min.   :0.9000  Min.   : 3.522  Min.   :10.0
1st Qu.:0.001017  1st Qu.:0.9091  1st Qu.: 3.588  1st Qu.:10.0
Median :0.001017  Median :0.9167  Median : 3.669  Median :10.0
Mean   :0.001139  Mean   :0.9337  Mean   : 4.216  Mean   :11.2
3rd Qu.:0.001220  3rd Qu.:0.9333  3rd Qu.: 4.698  3rd Qu.:12.0
Max.   :0.001932  Max.   :1.0000  Max.   :11.235  Max.   :19.0

mining info:
      data ntransactions support confidence
store_data      9835      0.001      0.9
```

Рис. 2.43. Анализ найденных правил

Можно выбрать и проанализировать правила для конкретного продукта рис. 2.44.

```
# Выберем правила, в которые входит продукт rolls/buns
rollsrules <- subset(rules, items %in% "rolls/buns")
inspect(rollsrules)
```

```
> inspect(rollsrules)
  lhs                                     rhs      support  confidence lift  count
[1] {pork,rolls/buns,waffles}              => {whole milk}  0.001016777  0.9090909   3.557863  10
[2] {beef,rolls/buns,tropical fruit,yogurt} => {whole milk}  0.001321810  0.9285714   3.634103  13
[3] {brown bread,other vegetables,rolls/buns,root vegetables} => {whole milk}  0.001016777  0.9090909   3.557863  10
[4] {newspapers,rolls/buns,soda,whole milk} => {other vegetables} 0.001016777  1.0000000   5.168156  10
[5] {citrus fruit,pastry,rolls/buns,whipped/sour cream}      => {whole milk}  0.001016777  1.0000000   3.913649  10
[6] {rolls/buns,root vegetables,sausage,tropical fruit}      => {whole milk}  0.001016777  1.0000000   3.913649  10
[7] {bottled water,rolls/buns,root vegetables,tropical fruit} => {whole milk}  0.001118454  0.9166667   3.587512  11
```

Рис. 2.44. Правила для продукта rolls/buns

3. ОБУЧЕНИЕ БЕЗ УЧИТЕЛЯ НА ЯЗЫКЕ PYTHON, ANACONDA, JUPYTER NOTEBOOK, GOOGLE COLAB

3.1. Настройка окружения

Для выполнения анализа с помощью языка Python установим Anaconda с официального сайта: <https://www.anaconda.com/distribution/>. На момент написания пособия актуальна версия Anaconda для Python 3.7 [12].

Anaconda – это дистрибутив Python и R. Он включает:

- пакеты для анализа данных (предустановленные и легко подключаемые) на языках Python/R;
 - управление библиотеками, зависимостями и средами с помощью собственного менеджера Conda;
 - библиотеки для настройки моделей машинного обучения и глубокого обучения: scikit-learn, TensorFlow и Theano;
 - библиотеки для анализа данных, управления масштабируемостью и производительностью Dask, NumPy, pandas и Numba;
 - возможности визуализации данных с помощью библиотек Matplotlib, Bokeh, Datashader и Holoviews;
 - Spyder (IDE/редактор) и Jupyter.
- Экосистема Anaconda показана на рис. 3.1.

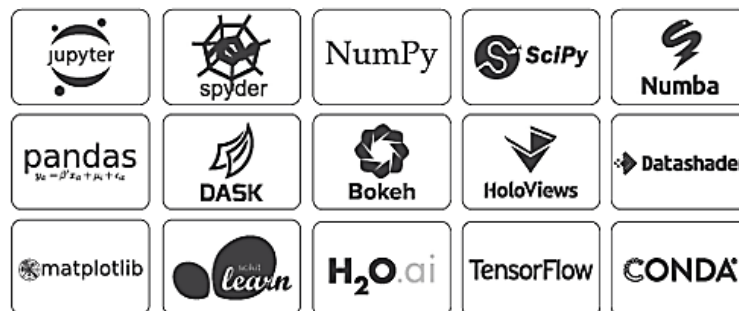


Рис. 3.1. Возможности Anaconda

При установке Anaconda необходимо обратить внимание на следующие параметры: если вы не устанавливаете программу для всех пользователей (в Windows для этого нужны привилегии Администратора), то выбирайте вариант установки «Только я» (Just Me). В качестве папки для установки Anaconda выберите путь, который не содержит имен папок с пробелами в названии (например, Program Files) и не английских символов юникода (например, русских букв).

В противном случае при подключении пакетов могут возникнуть проблемы интеграции.

Подумайте, нужно ли добавлять Anaconda в переменную окружения PATH, поскольку это может повлиять на работу других программ (рис. 3.2).

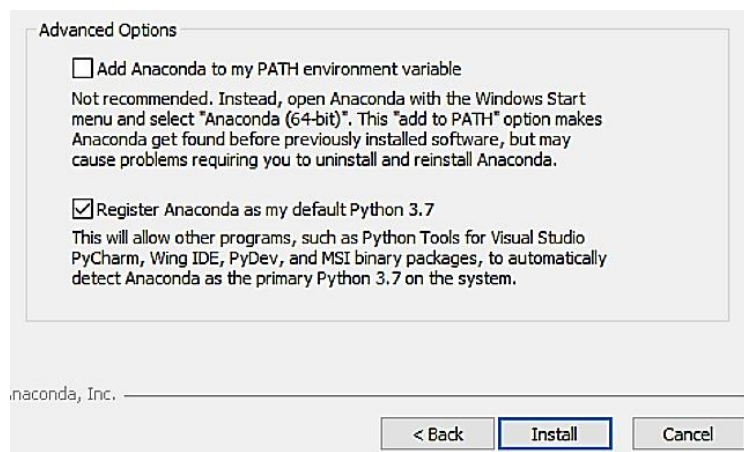


Рис. 3.2. Диалоговое окно инсталляции Anaconda

В Anaconda есть возможность работать с Jupyter Notebook. Notebook состоит из блоков кода или текста и имеет расширение .ipynb. Создать новый файл можно с домашней страницы (<http://localhost:8888/tree#>), которая открывается в браузере при запуске Jupyter Notebook (компьютер пользователя – локальный хост, выход в Интернет не требуется). При выборе доступного языка Python 3 (рис. 3.3) открывается новый notebook (рис. 3.4).

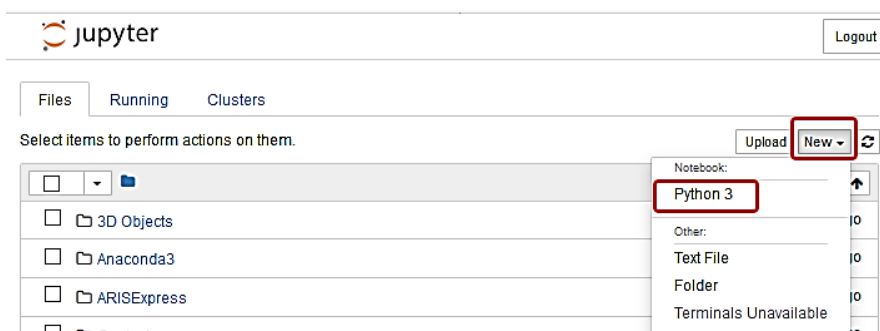


Рис. 3.3. Создание нового notebook

Новый блок добавляется с помощью кнопки + на панели инструментов. По умолчанию добавляется блок для ввода кода (рис. 3.4), но параметры блока можно изменить, выбрав в раскрывающемся списке Markdown (инструмент работы с текстом включает в себя массу возможностей работы с формулами и не только).

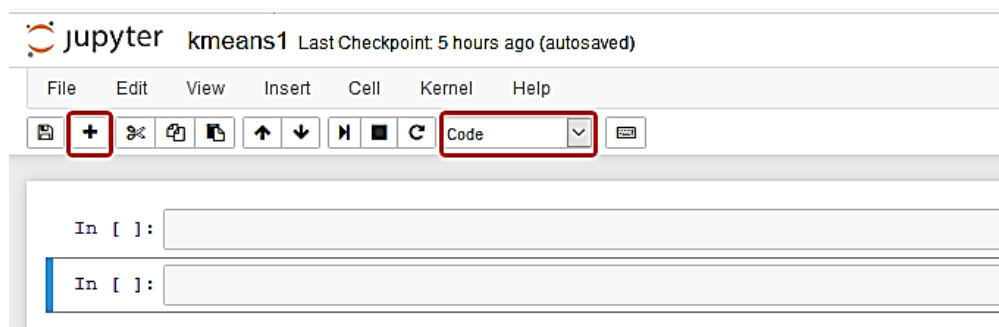


Рис. 3.4. Добавление новых блоков в notebook

Код в блоке с маркером In [] можно редактировать и посылать на выполнение любое количество раз с помощью кнопки . Сразу под блоком кода, отправленного на выполнение, если этот код предусматривал вывод, появится блок Out [] с результатом исполнения кода, в квадратных скобках будет отображаться, какой блок по счету был отправлен на исполнение в текущем notebook и, соответственно, результат исполнения кода будет иметь тот же порядковый номер, только с маркером Out (рис. 3.5).

```
In [1]: import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
dataset = pd.read_csv('insurance.csv')
dataset.head(10)
```

Out [1]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3886.85520
5	31	female	25.740	0	no	southeast	3756.62160
6	46	female	33.440	1	no	southeast	8240.58960
7	37	female	27.740	3	no	northwest	7281.50560
8	37	male	29.830	2	no	northeast	6406.41070
9	60	female	25.840	0	no	northwest	28923.13692

Рис. 3.5. Результат выполнения кода в notebook

В состав Anaconda входит редактор Spyder (аналог RStudio для языка R). В нем можно писать код, который сохраняется в скрипте, имеющем расширение .py. Выполнение кода происходит в консоли. Комментарии отделяются от команд открывающимися и закрывающимися звездочками *** (рис. 3.6). В пособии для работы с кодом используется Jupyter Notebook.

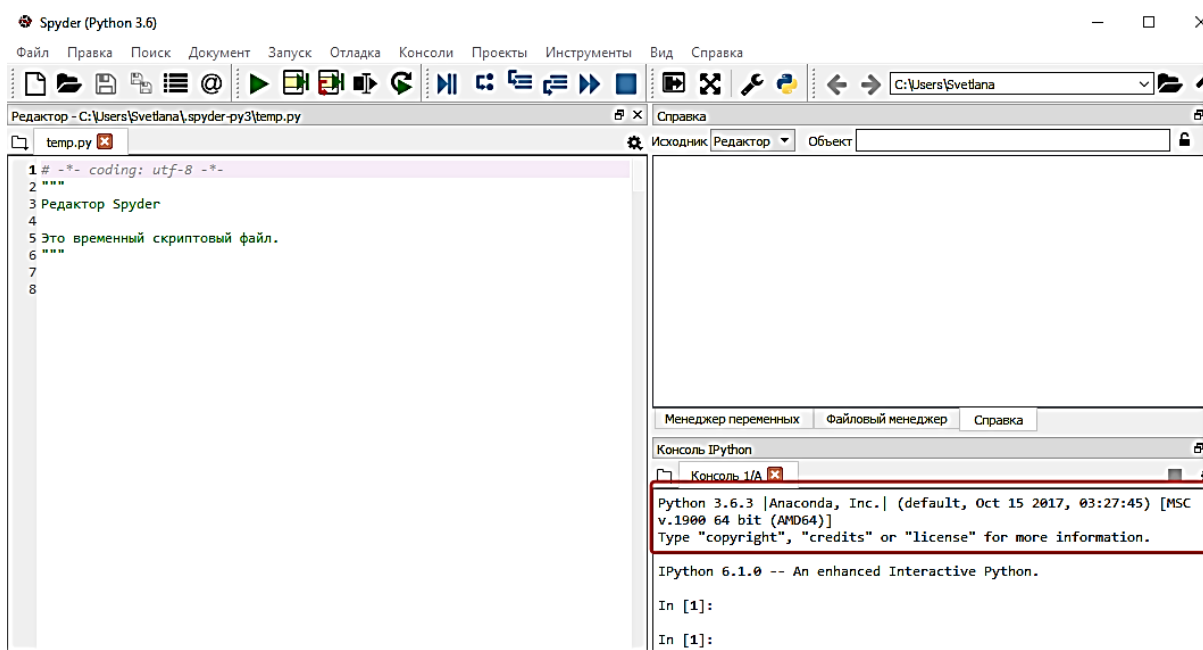


Рис. 3.6. Редактор Spyder

Для решения проблемы с малой мощностью компьютера для анализа данных есть браузерное решение (доступ через Интернет) от компании Google [13].

Colab Laboratory – это бесплатная среда для ноутбуков Jupyter, предоставляемая Google, где можно использовать бесплатные графические процессоры и TPU.

Для получения доступа к возможностям Colab Laboratory нужно создать учетную запись в Google-клиенте gmail.com, далее перейти по ссылке: <https://colab.research.google.com>, тогда откроется доступ в файл Welcome To Colaboratory, и появится возможность создать новый notebook с помощью Google Colab (рис. 3.7):

- Examples (примеры): содержит несколько Jupyter Notebook с примерами;
- Recent (последние): Jupyter Notebook, с которым недавно работали;
- Google Drive: сохраненные на вашем диске Google Jupyter Notebook;
- GitHub: доступ в репозиторий github для загрузки Jupyter Notebook в Colab (доступен при подключении к GitHub);
- Upload (загрузить): загрузка Jupyter Notebook из вашего локального каталога.

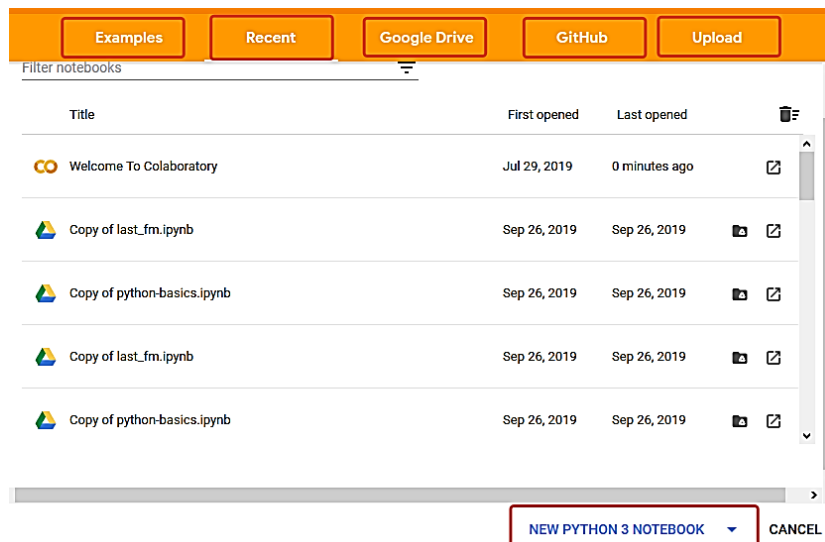


Рис. 3.7. Стартовое окно Google Colab

Новый файл `Untitled.ipynb` будет сохранен на подключенный к аккаунту Google Диск, как и в Jupyter файл `notebook` состоит из блоков текста и кода. В панели инструментов добавление блока – это кнопка `+` рядом с нужной опцией, но можно навести мышь на центр блока снизу, и те же кнопки будут доступны для выбора (рис. 3.8). Код можно сразу передавать на выполнение кнопкой  слева от блока кода.

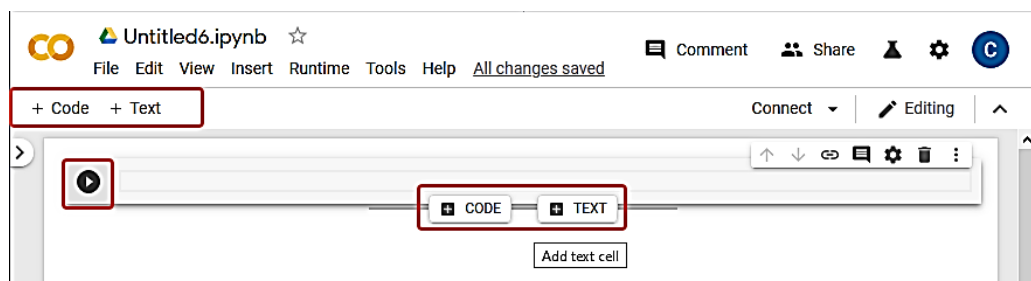


Рис. 3.8. Добавление блоков в notebook

Аналогично пакетам языка R в Python для выполнения анализа данных используются модули (библиотеки). Модуль – файл с расширением `.py`, содержащий определения и другие инструкции языка Python.

Наиболее часто используемые модули для анализа данных уже входят в Anaconda и Google Colab, нет необходимости их загружать.

В начальном блоке `notebook` все необходимые библиотеки подключаются с помощью инструкции `import`. Но можно это сделать и позже, главное – обращаться к функциям библиотеки после ее подключения.

```
# Импортируем библиотеку inspect
import inspect
```

После подключения, используя имя модуля, можно получить доступ к его функциям: <имя модуля>.<имя функции>():

```
#возвращаем аргументы функции KMeans() с помощью signature() из inspect
inspect.signature(KMeans)
```

Можно подключать библиотеку, сразу задавая сокращенное имя для обращения к ее функциям:

```
import pandas as pd
# Загружаем набор данных
iris_df = pd.read_csv('iris.csv')
```

Можно импортировать имена функций из модуля непосредственно в текущее пространство имен:

```
import matplotlib.pyplot as plt
```

или импортировать конкретные функции из модуля:

```
from sklearn import datasets
# Загружаем встроенный набор данных
iris_df = datasets.load_iris()
```

Для выяснения имен, определенных в модуле, функции, переменной можно использовать встроенную функцию `dir()`. Она возвращает отсортированный список строк (рис. 3.9).

```
In [125]: print(dir(KMeans))

['_class_', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setstate__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_check_fit_data', '_check_test_data', '_estimator_type', '_get_param_names', '_transform', '_fit', '_fit_predict', '_fit_transform', '_get_params', '_predict', '_score', '_set_params', '_transform']
```

Рис. 3.9. Имена, определенные в методе `KMeans()`

Для построения графиков в Python будем использовать библиотеку `matplotlib` [14] и `seaborn` [15], для работы с многомерными массивами – `NumPy` [16], для работы с данными (чтения файлов, работы с пропусками и т.п.) – библиотеку `pandas` [17], для реализации алгоритмов машинного обучения без учителя – библиотеку `scikit-learn` (`sclearn`) [18], основанную на других библиотеках Python: `NumPy`, `SciPy` и `matplotlib`.

3.2. Метод k-средних

В Python для реализации метода k-средних необходимо импортировать из класса `sklearn.cluster` (библиотека `scikit-learn`) класс `KMeans` (параметры класса приведены в табл. 3.1, аргументы класса – в табл. 3.2, методы класса – в табл. 3.3) [19].

```
from sklearn.cluster import KMeans
```

Таблица 3.1

Параметры класса `KMeans()`

<code>n_clusters</code>	Количество кластеров
<code>init</code>	Метод определения начального положения центроидов
<code>max_iter</code>	Максимальное количество итераций

Таблица 3.2

Атрибуты класса `KMeans()`

<code>cluster_centers_</code>	Массив кластерных центров
<code>inertia_</code>	Сумма квадратов расстояний выборок до их ближайших центров скоплений

Таблица 3.3

Методы класса `KMeans()`

<code>fit(X)</code>	Выполняет кластеризацию методом k-средних
<code>fit_predict(X)</code>	Находит значение кластера для каждого наблюдения данных X

Повторим реализацию метода k-средних на языке Python на тех же данных, что и на языке R.

3.2.1. Набор данных *ирисы*

Первый набор данных – набор данных *ирисы Фишера* (см. 2.2.1). Файл с данными `iris.csv` должен находиться в той же директории, что и папка `.anaconda` (этот путь указывался при установке Anaconda (см. 3.1)).

Код notebook для подключения библиотек и загрузки набора данных представлен на рис. 3.10.

```
In [126]: # Импортируем библиотеки
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd

# Загружаем набор данных
iris_df = pd.read_csv('iris.csv')
# Выводим на просмотр первые строки набора
iris_df.head()
```

Out[126]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa

Рис. 3.10. Загрузка библиотек и данных для анализа

Для загрузки файла с локального диска в облачный Google Colab воспользуемся инструкциями (рис. 3.11). Появится виджет с предложением выбрать файлы на компьютере (файлы закачиваются в ту же папку, где запущен ноутбук на Colab). Найдём нужный файл во время выполнения инструкции.

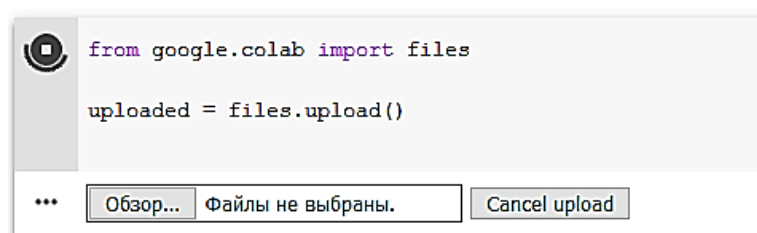


Рис. 3.11. Загрузка файла с локального диска в Google Colab

При успешном завершении загрузки будет выведена информация о файле (рис. 3.12). В остальном исполнение инструкций будет аналогичным, и далее все инструкции в Jupyter можно использовать в Google Colab.

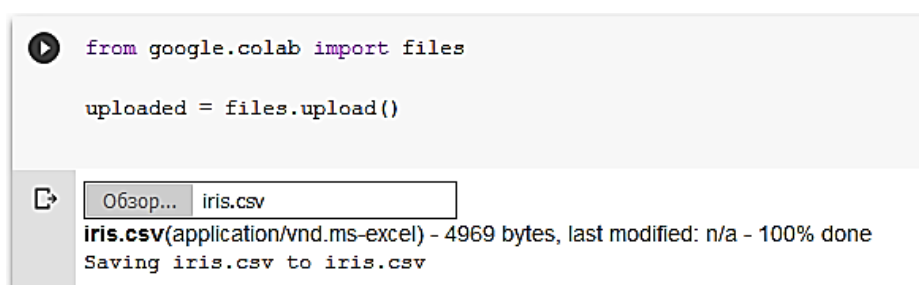


Рис. 3.12. Результат загрузки файла с локального диска в Google Colab

Создаем новый набор данных без переменной, определяющей вид ириса (рис. 3.13).

```
In [127]: # Копируем первые четыре столбца в новую переменную
iris_df1 = pd.DataFrame(iris_df[['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width']])
iris_df1.head()
```

Out[127]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
1	5.1	3.5	1.4	0.2
2	4.9	3.0	1.4	0.2
3	4.7	3.2	1.3	0.2
4	4.6	3.1	1.5	0.2
5	5.0	3.6	1.4	0.2

Рис. 3.13. Создание нового набора данных

Применим метод k-средних к данным набора *ирисы* (рис. 3.14). Добавим новые столбцы в набор данных: номер кластера и факторную переменную на основе данных о виде ириса (рис. 3.14–3.15). В Python нумерация кластеров и преобразование категориальной переменной в фактор начинаются с нуля.

```
In [129]: # Описываем модель
model = KMeans(n_clusters=3)

# Проводим моделирование
model.fit(iris_df1)

# Добавим в исходный набор новый столбец с номером кластера наблюдения
iris_df1['cluster'] = model.fit_predict(iris_df1)
iris_df1.head()
```

Out[129]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	cluster
1	5.1	3.5	1.4	0.2	0
2	4.9	3.0	1.4	0.2	0
3	4.7	3.2	1.3	0.2	0
4	4.6	3.1	1.5	0.2	0
5	5.0	3.6	1.4	0.2	0

Рис. 3.14. Результат кластеризации методом k-средних для данных ирисов

```
In [130]: # Добавим в исходный набор новый столбец, преобразовав категориальную переменную в фактор
iris_df1['species'] = pd.factorize(iris_df['Species'])[0]

iris_df1.head()
```

Out[130]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	cluster	species
1	5.1	3.5	1.4	0.2	0	0
2	4.9	3.0	1.4	0.2	0	0
3	4.7	3.2	1.3	0.2	0	0
4	4.6	3.1	1.5	0.2	0	0
5	5.0	3.6	1.4	0.2	0	0

Рис. 3.15. Добавление в набор данных переменной, определяющей класс ириса

Построим график по переменным Sepal.Length и Petal.Width (рис. 3.16). Переменная cluster задает цвет (color), а переменная species – размер (используется формула, так как нулевое значение не позволит отобразить маркер).

```
In [131]: #Строим график
plt.scatter(iris_df1['Sepal.Length'], iris_df1['Petal.Width'], c = iris_df1['cluster'],
            s = 30*(3*iris_df1['species']+1), marker = "v")
plt.title('Iris')
plt.xlabel('Sepal.Length')
plt.ylabel('Petal.Width')

Out[131]: Text(0,0.5, 'Petal.Width')
```

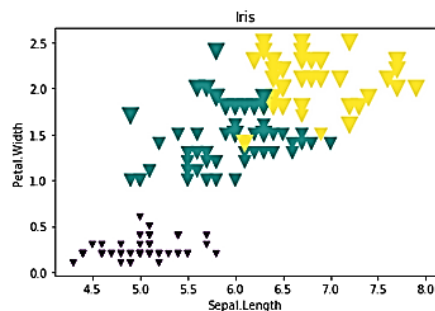


Рис. 3.16. Построение графика для ирисов

Для повышения читабельности графика настроим параметры отображения, добавив дополнительную строчку в начало фрагмента кода, строящего график.

```
plt.figure(figsize=(16, 10), dpi= 80, facecolor='w', edgecolor='k')
```

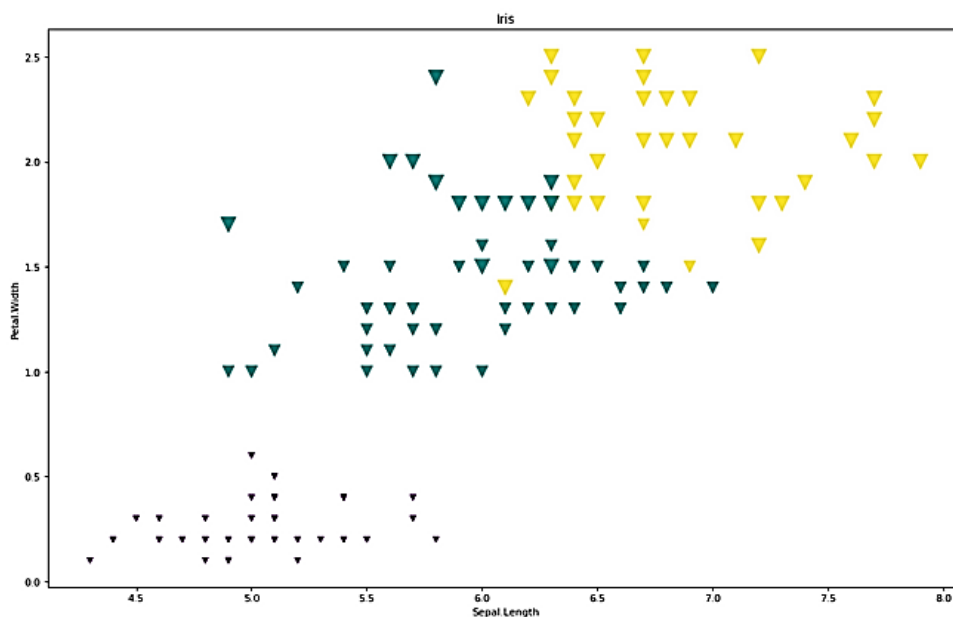


Рис. 3.17. Результат применения дополнительных настроек отображения графика

Составим таблицу сопряженности переменных `cluster` и `species` для анализа верности определения классов растений при разбиении на кластеры (рис. 3.18) с помощью кластерного анализа (значения NaN – это нулевые значения).

```
In [132]: #Составим по данным класса и кластера сводную таблицу
iris_df1.groupby(['cluster', 'species'])['cluster'].count().unstack()
```

Out[132]:

species	0	1	2
cluster			
0	50.0	NaN	NaN
1	NaN	48.0	14.0
2	NaN	2.0	36.0

Рис. 3.18. Распределение наблюдений по кластерам и классам для ирисов

3.2.2. Набор данных *медицинское страхование*

Второй набор данных – набор данных *медицинское страхование* (см. 2.2.2). Файл с данными `insurance.csv` должен находиться в той же директории, что и папка `.anaconda`.

Код `notebook` для подключения библиотек и загрузки набора данных (рис. 3.19).

```
In [56]: # Импортируем библиотеки
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import numpy as np
import seaborn as sns

# Загружаем набор данных
insurance_df = pd.read_csv('insurance.csv')
insurance_df.head()
```

Out[56]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

Рис. 3.19. Загрузка библиотек и данных для анализа

Код notebook для подготовки данных к анализу представлен на рис. 3.20. Преобразование в факторы необходимо, так как модель на основе класса KMeans() применяется только к числовым данным.

```
In [62]: insurance_nu = insurance_df
# Преобразуем категориальные переменные в факторы
insurance_nu['sex'] = pd.factorize(insurance_nu['sex'])[0]
insurance_nu['smoker'] = pd.factorize(insurance_nu['smoker'])[0]
insurance_nu['region'] = pd.factorize(insurance_nu['region'])[0]
# Создадим дубликат преобразованного набора
insurance_nu1 = insurance_nu
insurance_nu.head()
```

Out[62]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	0	0	16884.92400
1	18	1	33.770	1	1	1	1725.55230
2	28	1	33.000	3	1	1	4449.46200
3	33	1	22.705	0	1	2	21984.47061
4	32	1	28.880	0	1	2	3866.85520

Рис. 3.20. Преобразование категориальных переменных в факторы

Выполняем кластеризацию (рис. 3.21).

```
In [63]: # Описываем модель
model = KMeans(n_clusters=3)

# Проводим моделирование
model.fit(insurance_nu)

# Добавляем в исходный набор новый столбец с номером кластера наблюдения
insurance_nu['cluster'] = model.fit_predict(insurance_nu)
insurance_nu.head()
```

Out[63]:

	age	sex	bmi	children	smoker	region	charges	cluster
0	19	0	27.900	0	0	0	16884.92400	2
1	18	1	33.770	1	1	1	1725.55230	1
2	28	1	33.000	3	1	1	4449.46200	1
3	33	1	22.705	0	1	2	21984.47061	2
4	32	1	28.880	0	1	2	3866.85520	1

Рис. 3.21. Результат кластеризации, добавленный в набор данных

Построим график зависимости медицинских расходов от возраста для разных кластеров (рис. 3.22).

```
In [64]: #Строим график
plt.scatter(insurance_nu['age'], insurance_nu['charges'], c = insurance_nu['cluster'])
plt.title('Insurance')
plt.xlabel('age')
plt.ylabel('charges')
```

Out[64]: Text(0,0.5,'charges')

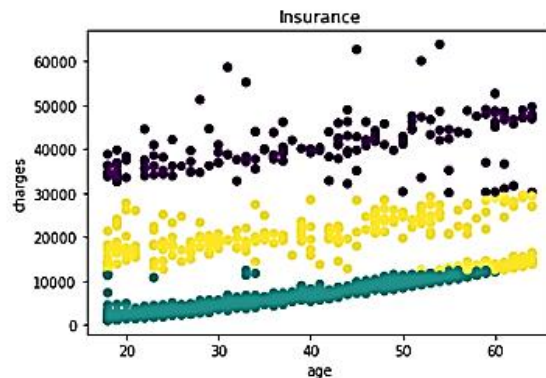


Рис. 3.22. График зависимости медицинских расходов от возраста

Выясним наилучшее количество кластеров с помощью «метода локтя» (рис. 3.23). Как видим, оптимальное число кластеров для набора данных – 3 или 4.

```
In [65]: # Инициализируем переменную, в которую будем записывать kmeans.inertia_
wcss = []
# В цикле проведем кластерный анализ для различного числа кластеров,
# присоединяя к массиву wcss соответствующие значения внутрикластерной суммы квадратов
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(insurance_nu1)
    wcss.append(kmeans.inertia_)
# Построим график по значениям wcss
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
```

Out[65]: Text(0,0.5,'WCSS')

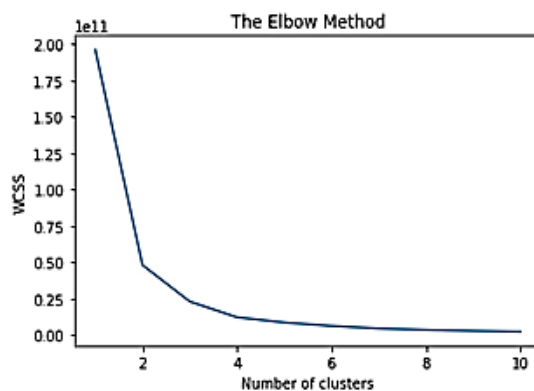


Рис. 3.23. Визуализация «метода локтя»

Визуализируем различие в медицинских расходах для разных кластеров с помощью графика «ящик с усами» (рис. 3.24).

```
In [72]: # Построим график "ящик с усами"
# для расходов на медицинское обслуживание в трех разных группах

sns.boxplot(x="cluster", y="charges", data=insurance_nu)
```

```
Out[72]: <matplotlib.axes._subplots.AxesSubplot at 0x189f6762cc0>
```

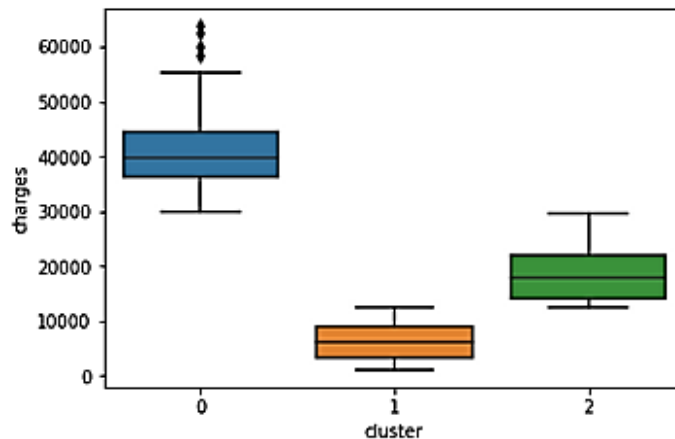


Рис. 3.24. Визуализация различий в медицинских расходах у представителей трех разных кластеров

Составим таблицу сопряженности для переменных `smoker` и `cluster` (анализ совместного распределения курящих и некурящих застрахованных по кластерам) (рис. 3.25). Значения NaN – это нулевые значения.

```
In [87]: #Составим по данным о наличии вредной привычки у застрахованного и кластера сводную таблицу
insurance_df['cluster'] = insurance_nu['cluster']
insurance_df.groupby(['cluster', 'smoker'])['cluster'].count().unstack()
```

```
Out[87]:
smoker no  yes
cluster
0      10.0 152.0
1      870.0  NaN
2      184.0 122.0
```

Рис. 3.25. Таблица сопряженности для переменных `smoker` и `cluster`

Визуализируем различие в медицинских расходах в зависимости от возраста для разных кластеров отдельно для курящих и некурящих застрахованных (рис. 3.26).

```
In [88]: # Построим визуализацию разброса расходов на медицинское обслуживание
# в зависимости от возраста отдельно для некурящих и для курящих smoker = 0
g = sns.FacetGrid(insurance_df, row="smoker", hue='cluster')
g = g.map(plt.scatter, "age", "charges").add_legend()
# Зададим размер отображения графиков
g.fig.set_size_inches(10,5)
```

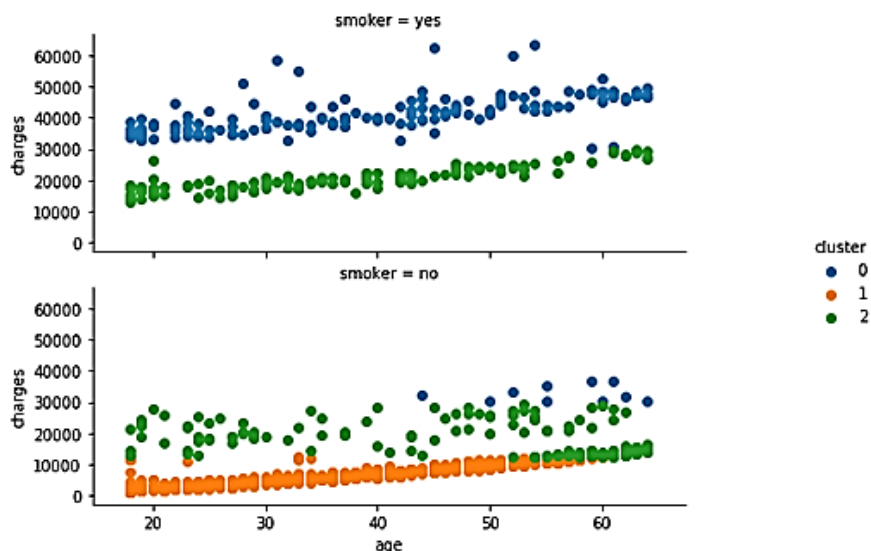


Рис. 3.26. Медицинские расходы в зависимости от возраста для разных кластеров отдельно для курящих и некурящих застрахованных

3.2.3. Набор данных *смс*

Третий набор данных – набор данных *смс* (см. 2.2.3). Файл с данными `snsdata.csv` должен находиться в той же директории, что и папка `.anaconda`.

Код `notebook` для подключения библиотек и загрузки набора данных представлен на рис. 3.27.

```
In [13]: # Импортируем библиотеки
from sklearn.cluster import KMeans
import pandas as pd

# Загружаем набор данных
sms_df = pd.read_csv('snsdata.csv')
# Копируем столбцы с частотой слов в отдельный набор
interests = pd.DataFrame(sms_df.iloc[:, 4:39])
interests.head()
```

Out[13]:

	basketball	football	soccer	softball	volleyball	swimming	cheerleading	baseball	tennis	sports	...	dress	blonde	mall	shopping	clothes	holl
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	...	4	0	1	0	0	0
2	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	2	0	0

5 rows × 35 columns

Рис. 3.27. Загрузка библиотек и данных для анализа

Предварительно подготовим данные для анализа. Единообразие данных, которые используются для кластерного анализа, позволяет улучшить его результаты. Проведем стандартизацию данных (в стандартизованных данных среднее значение равно нулю, стандартное отклонение – 1 (рис. 3.28).

```
In [51]: # Стандартизация данных
from sklearn.preprocessing import StandardScaler
colnames_int = list(interests) # массив из имен столбцов
scaler = StandardScaler()
interests_sc = pd.DataFrame(scaler.fit_transform(interests), columns= colnames_int)
interests_sc.head()
```

Out[51]:

	basketball	football	soccer	softball	volleyball	swimming	cheerleading	baseball	tennis	sports ...	dress	blonde	mall
0	-0.332217	-0.357697	-0.242874	-0.217928	-0.22367	-0.259971	-0.207327	-0.201131	-0.168939	-0.297123 ...	-0.246906	-0.050937	-0.369915
1	-0.332217	1.060049	-0.242874	-0.217928	-0.22367	-0.259971	-0.207327	-0.201131	-0.168939	-0.297123 ...	8.653277	-0.050937	1.067392
2	-0.332217	1.060049	-0.242874	-0.217928	-0.22367	-0.259971	-0.207327	-0.201131	-0.168939	-0.297123 ...	-0.246906	-0.050937	-0.369915
3	-0.332217	-0.357697	-0.242874	-0.217928	-0.22367	-0.259971	-0.207327	-0.201131	-0.168939	-0.297123 ...	-0.246906	-0.050937	-0.369915
4	-0.332217	-0.357697	-0.242874	-0.217928	-0.22367	-0.259971	-0.207327	-0.201131	-0.168939	-0.297123 ...	-0.246906	-0.050937	-0.369915

5 rows x 35 columns

Рис. 3.28. Подготовка данных

Проведем кластеризацию методом к-средних с числом кластеров равным 5 (рис. 3.29).

```
In [52]: # Описываем модель
model = KMeans(n_clusters=5)

# Проводим моделирование
model.fit(interests_sc)
# Добавляем в исходный набор новый столбец с номером кластера наблюдения
sms_df['cluster'] = model.fit_predict(interests_sc)
sms_df.head()
```

Out[52]:

	gradyear	gender	age	friends	basketball	football	soccer	softball	volleyball	swimming	...	mall	shopping	clothes
0	2006	M	18.982	7	0	0	0	0	0	0 ...	0	0	0	0
1	2006	F	18.801	0	0	1	0	0	0	0 ...	1	0	0	0
2	2006	M	18.335	69	0	1	0	0	0	0 ...	0	0	0	0
3	2006	F	18.875	0	0	0	0	0	0	0 ...	0	0	0	0
4	2006	NaN	18.995	10	0	0	0	0	0	0 ...	0	2	0	0

5 rows x 41 columns

Рис. 3.29. Кластеризация методом к-средних

Частоту использования в *смс* отдельных слов для кластера наиболее корректно характеризует центроид соответствующего кластера (рис. 3.30).

```

In [53]: # Выведем на просмотр информацию о центрах кластеров
cluster_centers = model.cluster_centers_

cluster_centers

Out[53]: array([[ 1.58325356e-01,  2.27893455e-01,  1.03914141e-01,
                  7.28719462e-02,  1.87558332e-01,  2.53483344e-01,
                  3.84416117e-01,  2.91421521e-02,  1.32068452e-01,
                  9.87792448e-02,  3.74741640e-01,  2.54442852e-02,
                  1.16086881e-01,  4.02018409e-01,  3.99933608e-02,
                  2.21838772e-01, -9.08674343e-02, -9.35957638e-02,
                  1.36251154e-01,  5.61099773e-02,  3.17964852e-02,
                  4.25601835e-03,  1.78080973e-02, -4.27455465e-02,
                  4.47766983e-01,  1.52787660e-01,  6.21670739e-02,
                  6.20083424e-01,  7.81033998e-01,  5.79090603e-01,
                  4.13730941e+00,  3.98350903e+00,  5.51403727e-02,
                  1.18296549e-01,  5.19369814e-02],

```

Рис. 3.30. Массив значений центроидов построенных кластеров
(первая строка массива)

```

[ -1.65956198e-01, -1.62716858e-01, -9.01478905e-02,
  -1.13313569e-01, -1.16142800e-01, -1.06937332e-01,
  -1.14323459e-01, -1.08483492e-01, -5.32974087e-02,
  -1.30724866e-01, -1.87726342e-01, -9.61579560e-02,
  -9.07775272e-02, -1.37458098e-01, -1.36726193e-01,
  -1.58812828e-01, -7.58207116e-02, -3.77010459e-02,
  -1.49894235e-01, -1.21683418e-01, -1.20993141e-01,
  -1.59284053e-01, -8.76107834e-02, -6.99319349e-02,
  -2.00636484e-01, -1.44024175e-01, -2.91261135e-02,
  -1.87449580e-01, -2.33185072e-01, -1.89063072e-01,
  -1.55846874e-01, -1.48613943e-01, -9.40689834e-02,
  -8.22795712e-02, -8.37584393e-02],
[ -3.32217264e-01,  2.47779503e+00, -2.42874104e-01,
  -2.17927819e-01, -2.23669630e-01,  1.67433426e+00,
  -2.07327097e-01, -2.01130618e-01, -1.68938902e-01,
  -2.97123387e-01,  8.43855726e-01,  1.93955434e+01,
  1.62589738e+00, -2.64224954e-01, -2.02619370e-01,
  4.94457044e-01,  6.26046293e-01, -1.41421093e-01,
  -5.89161280e-01,  1.05039699e+00,  7.09847711e+00,
  -2.97556823e-01,  1.52644659e+00, -1.04247248e-01,
  1.05446933e+01,  4.20318551e+00,  1.68307291e+02,
  -3.69914728e-01, -4.87314241e-01,  1.80161279e+00,
  -2.01476270e-01, -1.83031743e-01,  9.31280745e+00,
  1.34750994e+01,  1.48127445e+01],

```

Рис. 3.30. Массив значений центроидов построенных кластеров
(вторая и третья строки массива)

```
[ 4.92555591e-01, 4.73371846e-01, 2.76613032e-01,
 3.54871362e-01, 3.61177715e-01, 2.99562034e-01,
 3.19751283e-01, 3.25202945e-01, 1.49093592e-01,
 3.17616312e-01, 5.17744471e-01, -7.82746265e-03,
 2.08171122e-01, 3.72783590e-01, -5.26508948e-02,
 4.37532715e-01, 1.72925452e-01, 1.11867830e-01,
 3.04585247e-01, 2.08097134e-01, 3.53795846e-01,
 5.36652190e-01, 2.93113324e-01, 2.35499704e-01,
 2.48517114e-01, 3.93890227e-01, 3.34635736e-02,
 4.85913956e-01, 6.61812947e-01, 3.79825804e-01,
-5.98421807e-02, -7.80725912e-02, 6.54981948e-02,
 1.26918617e-01, 2.45162771e-02],
[ 4.46492601e-01, 4.30962316e-01, 1.64204362e-01,
 2.12225218e-01, 1.28099397e-01, 2.63380304e-01,
 1.79849353e-01, 3.40288086e-01, 1.28341311e-01,
 8.53883631e-01, 5.86250064e-01, 2.29025557e+00,
 6.30112806e-01, 3.52413779e-01, 3.56474849e+00,
 5.89411385e-01, 7.11174015e-01, 2.24247718e-01,
 1.36215601e+00, 1.42148237e+00, 4.32724270e-01,
 1.34557566e-01, 7.15369046e-02, 1.02004917e-01,
 2.64027516e+00, 5.95914438e-01, 2.23208102e-01,
 5.64956570e-01, 3.04527326e-01, 1.35967398e+00,
 1.91758990e-01, 2.89278267e-01, 1.72585296e+00,
 9.64954616e-01, 1.75635859e+00]])
```

Рис. 3.30. Массив значений центроидов построенных кластеров (четвертая и пятая строки массива)

Запишем во фрейм данных `interests_1` значения центроида первого кластера и соответствующие им слова, проведем упорядочение по частоте использования слов в *смс* первого кластера (рис. 3.31).

```
In [55]: # Создадим набор данных по первому кластеру
# Первый столбец - частота упоминания слов в первом кластере
interests_1 = pd.DataFrame (cluster_centers[0])
interests_1.rename(columns={0: 'freq'}, inplace=True)
# Второй столбец - слова
interests_1['word'] = colnames_int
# Проведем упорядочение набора по столбцу freq по убыванию ascending=False
interests_1.sort_values(['freq'], ascending=False).head(10)
```

Out [55]:

	freq	word
30	4.137309	hollister
31	3.983509	abercrombie
28	0.781034	shopping
27	0.620083	mall
29	0.579091	clothes
24	0.447767	hair
13	0.402018	hot
6	0.384416	cheerleading
10	0.374742	cute
5	0.253483	swimming

Рис. 3.31. Частота использования слов в *смс* первого кластера

Аналогично можно построить фреймы данных, анализирующих частоту встречаемости слов в *смс*, для остальных кластеров.

3.2.4. Машинное обучение на основе метода k-средних

В заключении раздела о методе k-средних на языке Python проведем обучение по этому методу в Google Colab на данных iris.csv.

Произведем загрузку файла в notebook Google Colab (рис. 3.32).



Рис. 3.32. Загрузка файла

Загрузим данные и подключим библиотеки для кластеризации методом k-средних (рис. 3.33). Преобразуем переменную Species в фактор (рис. 3.34).

```
[25] from sklearn.cluster import KMeans
import pandas as pd

# Загружаем набор данных
iris_df = pd.read_csv('iris.csv')
# Выводим на просмотр первые строки набора
iris_df.head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa

Рис. 3.33. Загрузка данных

```
[52] # Копируем первые четыре столбца в новую переменную
iris_dfl = pd.DataFrame(iris_df[['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width']])
# Добавляем переменную класс как фактор
iris_dfl['species'] = pd.factorize(iris_df['Species'])[0]
iris_dfl.head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	species
1	5.1	3.5	1.4	0.2	0
2	4.9	3.0	1.4	0.2	0
3	4.7	3.2	1.3	0.2	0
4	4.6	3.1	1.5	0.2	0
5	5.0	3.6	1.4	0.2	0

Рис. 3.34. Преобразование категориальной переменной в фактор

Проверим версию библиотеки `sklearn` (должна быть не ранее версии 0.21.1) (рис. 3.35).

```
[66] import sklearn
      sklearn.__version__

0.21.3
```

Рис. 3.35. Проверка версии подключенной библиотеки

Подключим из библиотеки `sklearn` необходимые методы (рис. 3.36). Исходные данные разбиваются на две части: тренировочную (в нашем случае 70 % набора) и тестовую (оставшиеся 30 %). Алгоритму машинного обучения предъявляются только тренировочные данные. На тестовых данных контролируется качество обучения. Механизм машинного обучения подразумевает, что данные размеченные, т.е. данные, на которых можно научиться предсказывать отклик, и для этих данных должны быть известны значения отклика. Для ирисов по первым четырем переменным, характеризующим цветки ириса, предсказывается класс ириса (`species`).

```
[101] from sklearn.model_selection import cross_validate
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import accuracy_score
      from sklearn.metrics import classification_report
      # Разбиваем исходное множество данных на тренировочную
      # и тестовую выборки (в соотношении 7/3) и соответствующие им верные значения классов
      train_data, test_data, train_labels, test_labels = train_test_split(
          iris_df[['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width']],
          iris_df[['species']], test_size = 0.3, random_state = 69)
      model = KMeans(n_clusters=3)
      # Проводим кластеризацию на тренировочной выборке
      model.fit(train_data)
      # Определяем кластеры по построенной модели на тестовых данных
      model_predictions = model.predict(test_data)
      # С помощью метрики accuracy сравниваем предсказанные значения
      # с исходной классификацией тестовой выборки
      accuracy_score(test_labels, model_predictions)

0.8444444444444444
```

Рис. 3.36. Модель машинного обучения с использованием метода К-средних на данных `iris`

Отчет по качеству построенной модели (рис. 3.37) можно проинтерпретировать с использованием метрик качества из п. 1.1 (формулы 1.2–1.4).

```
[102] print(classification_report(test_labels, model_predictions))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	16
1	0.67	0.83	0.74	12
2	0.86	0.71	0.77	17
accuracy			0.84	45
macro avg	0.84	0.85	0.84	45
weighted avg	0.86	0.84	0.85	45

Рис. 3.37. Оценка качества машинного обучения на данных iris

3.3. Метод главных компонент

В Python для реализации метода главных компонент необходимо импортировать из класса `sklearn.decomposition` (библиотека `scikit-learn`) класс `PCA` (параметры класса даны в табл. 3.4, аргументы класса – в табл. 3.5, методы класса – в табл. 3.6) [20].

```
from sklearn.decomposition import PCA
```

Таблица 3.4

Параметры класса `PCA()`

<code>n_components</code>	Количество главных компонент, о которых сохраняется информация. По умолчанию <code>n_components = min(n_samples, n_features)</code>
---------------------------	---

Таблица 3.5

Атрибуты класса `PCA()`

<code>explained_variance_</code>	Массив дисперсий главных компонент
<code>explained_variance_ratio_</code>	Доля объясненной дисперсии каждой главной компонентой
<code>components_</code>	Матрица нагрузок главных компонент

Таблица 3.6

Методы класса `PCA()`

<code>fit(X)</code>	Применение метода главных компонент к массиву данных <code>X</code>
<code>fit_transform(X)</code>	Получение матрицы значений для главных компонент

Повторим реализацию метода главных компонент на языке Python на тех же данных, что и на языке R (см. 2.3). Файл с данными `diabetes.csv` должен находиться в той же директории, что и папка `.anaconda`.

Код notebook для подключения библиотек и загрузки набора данных представлен на рис. 3.38.

In [6]:

```

# Импортируем библиотеки
import numpy
from pandas import read_csv
from sklearn.decomposition import PCA

# Загружаем набор данных
diabetes_df = read_csv('diabetes.csv')
diabetes_df.head()

```

Out [6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Рис. 3.38. Загрузка данных, подключение библиотек

Код notebook для подготовки данных к анализу и получения PVE для трех главных компонент приведен на рис. 3.39. Метод `PCA().fit()` применяется только к числовым данным.

In [17]:

```

# Исключаем из рассмотрения переменную отклика
X = diabetes_df.values[:,0:8]

# Описываем модель
pca = PCA(n_components = 3)
# Проводим моделирование
pca.fit(X)

features = pca.fit_transform(X)

# Выведем доли объясненной дисперсии (PVE)
pca.explained_variance_ratio_

```

Out[17]:

```
array([0.88854663, 0.06159078, 0.02579012])
```

Рис. 3.39. Подготовка данных и применение к ним метода главных компонент

Сравним матрицу значений трех главных компонент (первые 10 строк) (рис. 3.40) с аналогичным результатом использования функции `prcomp()` на языке R (см. рис. 2.37).

```

In [20]: # Выведем матрицу значений трех главных компонент (первые 10 строк)
         features[0:10,:]

Out[20]: array([[ -7.57146549e+01,  -3.59507826e+01,  -7.26078895e+00],
                [ -8.23582676e+01,   2.89082132e+01,  -5.49667139e+00],
                [ -7.46306434e+01,  -6.79064965e+01,   1.94618081e+01],
                [  1.10774227e+01,   3.48984859e+01,  -5.30177923e-02],
                [  8.97437881e+01,  -2.74693708e+00,   2.52128586e+01],
                [ -8.09779219e+01,  -3.94688717e+00,   6.41394937e-01],
                [  4.35095929e+00,   4.69339122e+01,   9.18341780e+00],
                [ -8.21372945e+01,   7.09795392e+00,   6.74889991e+01],
                [  4.68866389e+02,  -3.15282576e+01,   1.00570715e+01],
                [ -8.01938498e+01,  -1.80482829e+01,  -1.80497626e+01]])

```

Рис. 3.40. Матрица значений трех главных компонент
(первые 10 строк)

Также сравним матрицу нагрузок трех главных компонент (рис. 3.41) с аналогичным результатом использования функции `prcomp()` на языке R (рис. 2.36).

```

In [23]: # Выведем матрицу нагрузок трех первых компонент
         pca.components_

Out[23]: array([[ -2.02176587e-03,   9.78115765e-02,   1.60930503e-02,
                  6.07566861e-02,   9.93110844e-01,   1.40108085e-02,
                  5.37167919e-04,  -3.56474430e-03],
                [ -2.26488861e-02,  -9.72210040e-01,  -1.41909330e-01,
                  5.78614699e-02,   9.46266913e-02,  -4.69729766e-02,
                 -8.16804621e-04,  -1.40168181e-01],
                [ -2.24649003e-02,   1.43428710e-01,  -9.22467192e-01,
                 -3.07013055e-01,   2.09773019e-02,  -1.32444542e-01,
                 -6.39983017e-04,  -1.25454310e-01]])

```

Рис. 3.41. Матрица нагрузок трех главных компонент

График «каменистой осыпи» предлагается построить самостоятельно.

3.4. Ассоциативные правила

В Python для реализации метода `apriori` (для поиска ассоциативных правил) необходимо импортировать из библиотеки `apyori` функцию `apriori`.

```
from apyori import apriori
```

Аргументы функции `apriori()`

<code>transactions</code>	Массив транзакций
<code>min_support</code>	Минимальное значение поддержки
<code>min_confidence</code>	Минимальное значение достоверности
<code>min_lift</code>	Минимальное значение подъемной силы
<code>min_length</code>	Минимальная длина правила

Повторим реализацию поиска ассоциативных правил на языке Python на тех же данных, что и на языке R (см. 2.4). Будем работать в Google Colab. Файл с данными `gros.xlsx` должен содержать данные строк транзакций, разнесенные по ячейкам.

Код notebook для подключения библиотек и загрузки набора данных приведен на рис. 3.42.

```
[ ] import numpy as np
    import pandas as pd
```

Рис. 3.42. Подключение библиотек

Ранее все используемые библиотеки уже были предустановлены и доступны для подключения и в Jupyter, и в Google Colab. С помощью инструкции `pip` выполним установку библиотеки `apyori` (рис. 3.44) и подключим нужную функцию `apriori` (рис. 3.44), а также проведем загрузку данных (рис. 3.45).

```
[ ] pip install apyori

Collecting apyori
  Downloading https://files.pythonhosted.org/packages/25/fd/0561e2dd29eed544bad2d1991636e38700cdaef9530490b863741f35295/apyori-1.1.1.tar.gz
Building wheels for collected packages: apyori
  Building wheel for apyori (setup.py) ... done
  Created wheel for apyori: filename=apyori-1.1.1-cp36-none-any.whl size=5972 sha256=8e7bebc3085cc495adecb060d8d85fe8a4a84f91986238795f43a777ac0e6af8
  Stored in directory: /root/.cache/pip/wheels/7b/2a/35/c0c3749c1a36d4f454ea22d8396e1b854b86340d63cbbb7949
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.1
```

Рис. 3.43. Установка библиотеки `apyori`

```
[ ] from apyori import apriori
```

Рис. 3.44. Подключим из библиотеки `apyori` функцию `apriori`

```
[ ] from google.colab import files

uploaded = files.upload()

C➤ Обзор...  groc.xlsx
groc.xlsx(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 291283 bytes, last modified: n/a - 100% done
Saving groc.xlsx to groc.xlsx
```

Рис. 3.45. Загрузка данных

Считаем данные в переменную store_data (рис. 3.46).

```
[ ] store_data = pd.read_excel('groc.xlsx', header=None)
store_data.head()
```

	0	1	2	3	4	5	6	7
0	citrus fruit	semi-finished bread	margarine	ready soups	NaN	NaN	NaN	NaN
1	tropical fruit	yogurt	coffee	NaN	NaN	NaN	NaN	NaN
2	whole milk	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	pip fruit	yogurt	cream cheese	meat spreads	NaN	NaN	NaN	NaN
4	other vegetables	whole milk	condensed milk	long life bakery product	NaN	NaN	NaN	NaN

Рис. 3.46. Считаем данные в переменную store_data

Каждая транзакция приведена к виду максимальной по числу элементов транзакции (максимальная длина = 32). Избавимся от значений NaN, продублировав последний продукт в каждой транзакции (рис. 3.47).

```
[ ] store_data.fillna(method = 'ffill',axis = 1, inplace = True)
store_data.head()
```

	0	1	2	3	4	5	6	7
0	citrus fruit	semi-finished bread	margarine	ready soups	ready soups	ready soups	ready soups	ready soups
1	tropical fruit	yogurt	coffee	coffee	coffee	coffee	coffee	coffee
2	whole milk	whole milk	whole milk	whole milk	whole milk	whole milk	whole milk	whole milk
3	pip fruit	yogurt	cream cheese	meat spreads	meat spreads	meat spreads	meat spreads	meat spreads
4	other vegetables	whole milk	condensed milk	long life bakery product	long life bakery product	long life bakery product	long life bakery product	long life bakery product

Рис. 3.47. Избавление от NaN

Сформируем из данных набор транзакций для функции `apriori` (рис. 3.48) [9].

```
[ ] transactions = []
    for i in range(0, 983):
        transactions.append([str(store_data.values[i,j]) for j in range(0, 32)])
```

Рис. 3.48. Преобразование данных

Выполним поиск ассоциативных правил (рис. 3.49), определим количество найденных правил (рис. 3.50) и выведем их на экран (рис. 3.51). Полный список найденных правил приведен в прил. 1.

```
[ ] result = list(apriori(transactions, min support = 0.01, min confidence = 0.6,min length = 3))
```

Рис. 3.49. Выполнение поиска ассоциативных правил

```
[ ] len(result)
```

12

Рис. 3.50. Количество найденных правил

```
[ ] result
[ ] [RelationRecord(items=frozenset({'whole milk', 'baking powder'}), support=0.01017293997965412, ordered_statistics=[OrderedStatistic
RelationRecord(items=frozenset({'whole milk', 'ham'}), support=0.014242115971515769, ordered_statistics=[OrderedStatistic(items='ba
RelationRecord(items=frozenset({'whole milk', 'other vegetables', 'bottled beer'}), support=0.011190233977619531, ordered_statistic
RelationRecord(items=frozenset({'root vegetables', 'whole milk', 'butter'}), support=0.011190233977619531, ordered_statistics=[Order
RelationRecord(items=frozenset({'whole milk', 'other vegetables', 'domestic eggs'}), support=0.011190233977619531, ordered_statist
RelationRecord(items=frozenset({'whole milk', 'frozen vegetables', 'other vegetables'}), support=0.011190233977619531, ordered_stat
RelationRecord(items=frozenset({'whole milk', 'fruit/vegetable juice', 'yogurt'}), support=0.014242115971515769, ordered_statistics
RelationRecord(items=frozenset({'whole milk', 'margarine', 'rolls/buns'}), support=0.014242115971515769, ordered_statistics=[Order
RelationRecord(items=frozenset({'sugar', 'whole milk', 'other vegetables'}), support=0.01017293997965412, ordered_statistics=[Order
RelationRecord(items=frozenset({'soda', 'sausage', 'rolls/buns'}), support=0.012207527975584944, ordered_statistics=[OrderedStatist
RelationRecord(items=frozenset({'whole milk', 'tropical fruit', 'yogurt'}), support=0.014242115971515769, ordered_statistics=[Order
RelationRecord(items=frozenset({'whole milk', 'yogurt', 'whipped/sour cream'}), support=0.01017293997965412, ordered_statistics=[O
```

Рис. 3.51. Результат поиска ассоциативных правил

4. ЛАБОРАТОРНЫЕ РАБОТЫ

4.1. Лабораторная работа «Кластерный анализ методом k-средних»

Цель работы: научиться импортировать данные, проводить их предобработку, применять к ним метод кластерного анализа (k-средних) и интерпретировать полученные на основе выделения кластеров знания с точки зрения бизнес-контекста.

Формируемые знания, умения и навыки: изучить, как реализуются метод k-средних на языке R и на языке Python, получить навыки выделения бизнес-контекста из результатов кластерного анализа

Необходимо:

1. С сайта <https://www.kaggle.com/> импортировать один из наборов.
2. Используя возможности языка R и возможности языка Python, провести кластеризацию данных методом k-средних, выдвинуть гипотезы о бизнес-контексте результатов анализа.

Контрольные вопросы и задания

R

1. Как импортируются данные в формате .csv?
2. Какие аргументы у функции `kmeans()`? Какие значения могут принимать эти аргументы? В каком формате возвращается результат?
3. Какие преобразования исходных данных выполнялись и почему?
4. Какие функции использовались для визуализации полученных результатов?

Python

5. Как импортируются данные в формате .csv?
6. Какие параметры, аргументы и методы у класса `KMeans`? Как получить доступ к данным о кластерах наблюдений? Как получить доступ к данным о центроидах?
7. Какие преобразования исходных данных выполнялись и почему?
8. Какие функции использовались для визуализации полученных результатов?

Бизнес-контекст

Опишите выбранный набор данных: контекст данных, какие переменные есть в наборе, какое количество кластеров было выбрано для данных, в чем отличия полученных кластеров друг от друга, как можно использовать в бизнесе полученное разбиение на кластеры.

4.2. Лабораторная работа «Понижение размерности данных»

Цель работы: научиться импортировать данные, проводить их предобработку, применять к ним метод главных компонент для понижения размерности.

Формируемые знания, умения и навыки: изучить, как реализуются метод главных компонент на языке R и на языке Python.

Необходимо:

1. С сайта <https://www.kaggle.com/> импортировать один из наборов, включающий столбцы с числовыми значениями.
2. Используя возможности языка R и возможности языка Python, провести понижение размерности данных, используя метод главных компонент.

Контрольные вопросы и задания

R

1. Какие аргументы у функции `prcomp()`? Какие значения могут принимать эти аргументы? В каком формате возвращается результат?
2. Какие преобразования исходных данных выполнялись и почему?
3. Какие функции использовались для визуализации полученных результатов?

Python

4. Какие параметры, аргументы и методы у класса `PCA`? Как получить доступ к данным о векторах нагрузок главных компонент, а также к значениям главных компонент для наблюдений?
5. Какие преобразования исходных данных выполнялись и почему?
6. Какие функции можно использовать для визуализации полученных результатов?

4.3. Лабораторная работа «Ассоциативные правила»

Цель работы: научиться импортировать данные, проводить их предобработку, применять к ним метод поиска ассоциативных правил и интерпретировать полученные правила с точки зрения бизнес-контекста.

Формируемые знания, умения и навыки: изучить, как реализуется поиск ассоциативных правил на языках R и Python, получить навыки выделения бизнес-контекста из результатов построения ассоциативных правил.

Необходимо:

1. С сайта <https://www.kaggle.com/> импортировать один из наборов, содержащий транзакции.
2. Используя возможности языка R и возможности языка Python, провести в транзакциях поиск ассоциативных правил, выдвинуть гипотезы о бизнес-контексте результатов анализа.

Контрольные вопросы и задания

R

1. В каком формате должны быть данные для выявления в них правил?
2. Какие аргументы у функции `apriori()`? Какие значения могут принимать эти аргументы? В каком формате возвращается результат?
3. Какие функции использовались для визуализации полученных результатов?

Python

4. Как импортируются данные в формате `.csv`?
5. Какие аргументы у функции `apriori()`? Какие значения могут принимать эти аргументы? В каком формате возвращается результат?

Бизнес-контекст

Опишите выбранный набор данных: контекст данных, какие данные есть в наборе, как можно использовать в бизнесе полученные ассоциативные правила.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Открытый курс машинного обучения. Тема 7. Обучение без учителя: PCA и кластеризация / Open Data Science – URL: <https://habr.com/ru/company/ods/blog/325654/>
2. Тибширани, Р. Введение в статистическое обучение с примерами на языке R / Е. Джеймс, Д. Уиттон, Т. Хасты, Р. В. Тибширани ; пер. с англ. С. Э. Мастицкого. – 2-е изд., испр. – Москва : ДМК Пресс, 2017. – 456 с. – URL: <http://znanium.com/catalog/product/1027867>
3. Рашка, С. Python и машинное обучение: крайне необходимое пособие по новейшей предсказательной аналитике, обязательное для более глубокого понимания методологии машинного обучения / С. Рашка ; пер. с англ. А. В. Логунова. – Москва : ДМК Пресс, 2017. – 418 с. – URL: <http://znanium.com/catalog/product/1027758>
4. Коэльо, Л. П. Построение систем машинного обучения на языке Python / Л. П. Коэльо, В. Ричарт ; пер. с англ. А. А. Слинкина. – 2-е изд. – Москва : ДМК Пресс, 2016. – 302 с. – URL: <http://znanium.com/catalog/product/1027824>
5. Шитиков, В. К. Классификация, регрессия и другие алгоритмы Data Mining с использованием R / В. К. Шитиков, С. Э. Мастицкий. – URL: <https://ranalytics.github.io/data-mining/index.html>
6. Visualizing K-Means Clustering. – URL: <https://www.naftaliharris.com/blog/visualizing-k-means-clustering/>
7. Машинное обучение: от Ирисов до Телекома. – URL: <https://habr.com/ru/company/billing/blog/334738/>
8. Метрики в задачах машинного обучения – URL: <https://habr.com/ru/company/ods/blog/328372/>
9. Ассоциативные правила, или пиво с подгузниками / Open Data Science. – URL: <https://habr.com/ru/company/ods/blog/353502/>
10. Рындина, С. В. Бизнес-аналитика: визуализация данных / С. В. Рындина. – Пенза : Изд-во ПГУ, 2018. – 70 с.
11. Lantz, B. Machine Learning with R Packt Publishing, Birmingham / B. Lantz. – Mumbai, 2013. – P. 396. – URL: <https://books.google.ru/books?id=ZQu8AQAAQBAJ&printsec=frontcover&hl=ru#v=onepage&q&f=false>
12. Официальный сайт Anaconda. – URL: <https://www.anaconda.com/>
13. Официальный сайт Google Colaboratory. – URL: <https://colab.research.google.com>

14. Описание библиотеки matplotlib. – URL: <https://matplotlib.org/>
15. Описание библиотеки seaborn. – URL: <https://seaborn.pydata.org/>
16. Описание библиотеки NumPy. – URL: <https://numpy.org/>
17. Описание библиотеки pandas. – URL: <https://pandas.pydata.org/>
18. Описание библиотеки scikit-learn. – URL: <https://scikit-learn.org/stable/index.html>
19. Описание реализации метода k-средних на языке Python: sklearn.cluster.KMeans – URL: <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans>
20. Описание реализации метода главных компонент на языке Python: sklearn.decomposition.PCA – URL: <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>

ПРИЛОЖЕНИЕ 1

[RelationRecord(items=frozenset({'whole milk', 'baking powder'}), support=0.01017293997965412, ordered_statistics=[OrderedStatistic(items_base=frozenset({'baking powder'}), items_add=frozenset({'whole milk'}), confidence=0.7142857142857143, lift=2.659632034632035)]),
RelationRecord(items=frozenset({'whole milk', 'ham'}), support=0.014242115971515769, ordered_statistics=[OrderedStatistic(items_base=frozenset({'ham'}), items_add=frozenset({'whole milk'}), confidence=0.6666666666666667, lift=2.48232323232327)]),
RelationRecord(items=frozenset({'whole milk', 'other vegetables', 'bottled beer'}), support=0.011190233977619531, ordered_statistics=[OrderedStatistic(items_base=frozenset({'other vegetables', 'bottled beer'}), items_add=frozenset({'whole milk'}), confidence=0.6470588235294117, lift=2.4093137254901955)]),
RelationRecord(items=frozenset({'root vegetables', 'whole milk', 'butter'}), support=0.011190233977619531, ordered_statistics=[OrderedStatistic(items_base=frozenset({'root vegetables', 'butter'}), items_add=frozenset({'whole milk'}), confidence=0.6875, lift=2.559895833333333)]),
RelationRecord(items=frozenset({'whole milk', 'other vegetables', 'domestic eggs'}), support=0.011190233977619531, ordered_statistics=[OrderedStatistic(items_base=frozenset({'other vegetables', 'domestic eggs'}), items_add=frozenset({'whole milk'}), confidence=0.7333333333333333, lift=2.730555555555555)]),
RelationRecord(items=frozenset({'whole milk', 'frozen vegetables', 'other vegetables'}), support=0.011190233977619531, ordered_statistics=[OrderedStatistic(items_base=frozenset({'frozen vegetables', 'other vegetables'}), items_add=frozenset({'whole milk'}), confidence=0.6875, lift=2.559895833333333)]),
RelationRecord(items=frozenset({'whole milk', 'fruit/vegetable juice', 'yogurt'}), support=0.014242115971515769, ordered_statistics=[OrderedStatistic(items_base=frozenset({'fruit/vegetable juice', 'yogurt'}), items_add=frozenset({'whole milk'}), confidence=0.7777777777777778, lift=2.896043771043771)]),
RelationRecord(items=frozenset({'whole milk', 'margarine', 'rolls/buns'}), support=0.014242115971515769, ordered_statistics=[OrderedStatistic(items_base=frozenset({'margarine', 'rolls/buns'}), items_add=frozenset({'whole milk'}), confidence=0.7, lift=2.6064393939393935)]),
RelationRecord(items=frozenset({'sugar', 'whole milk', 'other vegetables'}), support=0.01017293997965412, ordered_statistics=[OrderedStatistic(items_base=frozenset({'sugar', 'other vegetables'}), items_add=frozenset({'whole milk'}), confidence=0.7142857142857143, lift=2.659632034632035)]),
RelationRecord(items=frozenset({'soda', 'sausage', 'rolls/buns'}), support=0.012207527975584944, ordered_statistics=[OrderedStatistic(items_base=frozenset({'soda', 'sausage'}), items_add=frozenset({'rolls/buns'}), confidence=0.6, lift=2.6687782805429863)]),
RelationRecord(items=frozenset({'whole milk', 'tropical fruit', 'yogurt'}), support=0.014242115971515769, ordered_statistics=[OrderedStatistic(items_base=frozenset({'tropical fruit', 'yogurt'}), items_add=frozenset({'whole milk'}), confidence=0.6086956521739131, lift=2.266469038208169)]),
RelationRecord(items=frozenset({'whole milk', 'yogurt', 'whipped/sour cream'}), support=0.01017293997965412, ordered_statistics=[OrderedStatistic(items_base=frozenset({'yogurt', 'whipped/sour cream'}), items_add=frozenset({'whole milk'}), confidence=0.7692307692307693, lift=2.8642191142191145)])]

Учебное издание

Рындина Светлана Валентиновна

**БИЗНЕС-АНАЛИТИКА НА ОСНОВЕ БОЛЬШИХ ДАННЫХ:
ОБУЧЕНИЕ БЕЗ УЧИТЕЛЯ НА ЯЗЫКАХ PYTHON И R**

Редактор *Е. Г. Акимова*

Технический редактор *Ю. В. Анурова*

Компьютерная верстка *Ю. В. Ануровой*

Подписано в печать 29.01.2020.

Формат 60×84¹/₁₆. Усл. печ. л. 4,42.

Заказ № 7. Тираж 3.

Издательство ПГУ

440026, Пенза, Красная, 40

Тел./факс: (8412) 56-47-33; e-mail: iic@pnzgu.ru