ДИСЦИПЛИНА	**Интеллектуальные системы и технологии**
	(полное наименование дисциплины без сокращений)
ИНСТИТУТ	**информационных технологий**
КАФЕДРА	**корпоративных информационных систем**
	полное наименование кафедры
ВИД УЧЕБНОГО МАТЕРИАЛА	**Учебно-методическое пособие**
	(в соответствии с пп.1-11)
ПРЕПОДАВАТЕЛЬ	**Демидова Лилия Анатольевна**
	(фамилия, имя, отчество)
СЕМЕСТР	**1 семестр (осенний), 2024 – 2025 учебный год**
	(семестр обучения, учебный год)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
МИРЭА – РОССИЙСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

РАЗВЕДОЧНЫЙ АНАЛИЗ ДАННЫХ. PYTHON ЧАСТЬ 2

Москва – 2023

УДК 004
ББК 32.97
Д 30
Демидова Л.А. Разведочный анализ данных. Python. Часть 2 [Электронный ресурс]: Учебное-методическое пособие / Демидова Л.А. — М.: МИРЭА – Российский технологический университет, 2023. — 1 электрон. опт. диск (CD-ROM)

Аннотация учебного-методического пособия.

В учебно-методическом пособии рассматриваются аспекты выполнения разведочного анализа категориальных данных средствами языка Python. Предлагается перечень заданий, при выполнении которых применяются различные инструменты разведочного анализа данных. Предназначено для магистрантов, изучающих дисциплину «Интеллектуальные системы и технологии» по направлению 09.04.04 Программная инженерия.

Учебно-методическое пособие издается в авторской редакции.

Автор: Демидова Лилия Анатольевна

Рецензенты:

Головин С.А., д.т.н., профессор, заведующий кафедрой МОСИТ ИИТ РТУ МИРЭА.

Андреева О.Н., д.т.н., доцент, начальник отдела научной работы АО «Концерн «Моринсис-Агат».

Минимальные системные требования:
Наличие операционной системы Windows, поддерживаемой производителем.
Наличие свободного места в оперативной памяти не менее 128 Мб.
Наличие свободного места в памяти хранения (на жестком диске) не менее 30 Мб.
Наличие интерфейса ввода информации.
Дополнительные программные средства: программа для чтения pdf-файлов (Adobe Reader).
Подписано к использованию по решению Редакционно-издательского совета
МИРЭА – Российского технологического университета от ___ _____ 2023 г.
Объем ___ Мб
Тираж 10

Оглавление

ВВЕДЕНИЕ

Учебно-методическое пособие излагает принципы разведочного анализа данных, уделяя особое внимание анализу категориальных данных.

Рассматриваются:

– метрики, используемые для анализа категориальных данных;

– аспекты анализа распределения категориальных данных;

– инструментарий, используемый для извлечения сводной информации из категориальных данных;

– инструментарий, используемый для кодирования категориальных данных.

Особое внимание уделяется применению инструментов визуализации, позволяющих выявить скрытую в категориальных данных информацию.

Приводятся примеры, демонстрирующие принципы применения инструментария разведочного анализа к категориальным данным.

Читателю предлагается перечень заданий: каждое из них содержит вариант набора данных из репозитория данных для машинного обучения, для которого необходимо осуществить разведочный анализ с использованием предлагаемых инструментов разведочного анализа, реализованных в программных библиотеках языка Python.

Читатель может получить дополнительные сведения по теоретическим вопросам, а также по аспектам программной реализации для рассматриваемых инструментов разведочного анализа данных в источниках, указанных в списке литературы.

РАЗВЕДОЧНЫЙ АНАЛИЗ ДАННЫХ

1. Основные понятия

Разведочный анализ данных (исследовательский анализ данных, exploratory data analysis, EDA) – подход к анализу данных для обобщения их основных характеристик, зачастую – с использованием статистической графики и других методов визуализации данных[1, 2].

Основоположником разведочного анализа данных принято считать Джона Уайлдера Тьюки (John Wilder Tukey), который в 70-е годы XX-го века предложил подразделить статистический анализ на два этапа: разведочный (exploratory data analysis) и подтверждающий (confirmatory data analysis). Разведочный этап должен был охватывать преобразование наборов данных и способы их наглядного представления, позволяющие выявить внутренние закономерности, проявляющиеся в данных, а подтверждающий – традиционные статистические методы оценки параметров и проверки гипотез в контексте анализируемого набора данных. При этом цель разведочного анализа заключалась в побуждении статистиков к исследованию данных и, возможно, к формулировке гипотез, которые могут привести к сбору новых данных и экспериментам.

Следует отметить, что многие идеи, предложенные Джоном Тьюки, позволили сформировать основы науки о данных, представляющей собой сплав многих дисциплин, включая статистику, информатику, информационные технологии с конкретикой предметных областей.

Разведочный анализ данных отличается от традиционного анализа исходных данных (initial data analysis, IDA), который более узко фокусируется на проверке предположений, необходимых для подбора модели и проверки гипотез, а также на обработке пропущенных значений и выполнении преобразований переменных по мере необходимости. EDA включает в себя IDA.

В 1961 году Джон Тьюки определил анализ данных как «процедуры анализа данных, методы интерпретации результатов таких процедур, способы планирования сбора данных, чтобы сделать их анализ более простым и точным, а также все механизмы и результаты математической статистики, которые применяются для анализа данных» [3].

В 1977 году Джон Тьюки написал книгу, посвященную аспектам разведочного анализа данных [4], отметив, что в статистике много внимания уделяется только проверке статистических гипотез, то есть подтверждающему анализу данных, и указав, что необходимо уделять больше внимания вопросам использования данных для выдвижения гипотез для проверки.

Целесообразно указать следующие цели разведочного анализа данных:

– максимальное «проникновение» в данные, подразумевающее поиск и извлечение знаний, скрытых в данных;

– выдвижение гипотез о причинах наблюдаемых явлений;

– оценка допущений, на которых будут основаны статистические выводы;

– поддержка выбора подходящих статистических инструментов и методов;

– обоснование плана для дальнейшего сбора данных с помощью опросов или экспериментов [5].

Разведочный анализ данных подразумевает выполнение анализа основных свойств данных, выявление общих закономерностей, основных структур, наиболее важных переменных, распределений, отклонений и аномалий в данных. Кроме того, при проведении разведочного анализа могут осуществляться проверка основных гипотез и построение начальных моделей на основе имеющихся данных.

Разведочный анализ данных предполагает, что будут осуществляться:

– систематический сбор данных, в ходе которого выполнять проверка гипотез и оценка результатов;

– очистка данных, в ходе которой осуществляется проверка правильности и достоверности данных, в частности, выявление ошибок в данных или отсутствие данных, а также исправление, удаление ошибок и т.п.;

– предварительная обработка данных, в ходе которой осуществляется преобразование необработанных данных в некоторый формат с применением инструментов масштабирования, извлечения, выбора признаков и т.п. с последующим получением итогового набора данных, который может быть использован для решения различных прикладных задач;

– визуализация данных, в ходе которой реализуется графическое представление информации и данных с использованием статистических информационных графиков, а также других инструментов, обеспечивающих четкую и эффективную передачу информации.

При проведении разведочного анализа принято осуществлять изучение вероятностных распределений переменных, выполнять построение и анализ корреляционных матриц, а также – применять многомерное шкалирование, дискриминантный анализ и факторный анализ.

Типичными графическими инструментами, используемыми в разведочном анализе, являются [6]:

– диаграмма размаха (box plot, box-and-whiskers diagram, «ящик с усами», «ящичковая диаграмма»), разработанная Джоном Тьюки в 1970-х годах;

– гистограмма (histogram, bar chart);

– многовариантная диаграмма (multi-vari chart);

– диаграмма Парето (Pareto chart);

– диаграмма рассеяния в 2D/3D (scatter plot);

– диаграмма «стебель-листья» (stem-and-leaf plot);

– параллельные координаты (parallel coordinates);

– отношение шансов (odds ratio);

– целенаправленное преследование проекций (targeted projection pursuit);

– тепловая карта (heat map);

– столбчатая диаграмма (bar chart);

– график горизонта (horizon graph);

– визуализация на основе глифов (glyph-based visualization methods), например, с использованием PhenoPlot [7] и лиц Чернова;

– проецирование, такое как обзорная экскурсия (grand tour), экскурсия с гидом (guided tour) и экскурсия вручную (manual tour).

Кроме того, могут применяться интерактивные версии графических инструментов.

Типичными инструментами снижения размерности, используемыми в разведочном анализе, являются:

– многомерное шкалирование (multidimensional scaling), представляющее собой метод анализа и визуализации данных с помощью расположения точек, соответствующих изучаемым (шкалируемым) объектам, в пространстве меньшей размерности, чем пространство признаков объектов;

– методы и алгоритмы линейного снижения размерности (linear dimensionality reduction);

– методы и алгоритмы нелинейного снижения размерности (nonlinear dimensionality reduction);

– иконография корреляций (iconography of correlations), заключающаяся в замене корреляционной матрицы диаграммой, где «замечательные» (remarkable) корреляции представлены сплошной линией в случае положительной корреляции и пунктирной линией в случае отрицательной корреляции.

В качестве наиболее известного инструмента линейного снижения размерности можно назвать метод главных компонент (Principal Component Analysis, PCA).

В качестве наиболее популярных в последние годы инструментов нелинейного снижения размерности можно назвать t-SNE-алгоритм (t-distributed

stochastic neighbor embedding, t-распределенное стохастическое вложение соседей) и UMAP-алгоритм (Uniform Manifold Approximation and Projection, равномерная аппроксимация и проекция топологического многообразия).

Эффективными количественными инструментами, применяемыми в разведочном анализе данных, являются:

– медианная полировка (median polish), предложенная Джоном Тьюки и позволяющая итеративно оценить влияние строк и столбцов на данные с применением медиан, вычисленных на основе строк и столбцов набора данных;

– тримин-мера (trimean), предложенная Джоном Тьюки и позволяющая оценить расположение распределения вероятностей на основе средневзвешенного для медианы распределения и его двух квартилей;

– ординационный или градиентный анализ (ordination or gradient analysis), дополняющий кластеризацию данных и позволяющий упорядочивать многомерные объекты, которые характеризуются значениями нескольких переменных таким образом, что похожие объекты находятся рядом друг с другом, а непохожие объекты находятся дальше друг от друга.

В настоящее время многие методы и алгоритмы разведочного анализа данных адаптированы для интеллектуального анализа данных.

1.1. Типы данных

Данные, к которым применяется разведочный анализ, поступают из различных источников. В качестве данных могут выступать показания датчиков, изображения, текстовая, графическая, аудио- и видеоинформация и т.п. Значительная часть этих данных не структурирована. Такие данные принято называть «сырыми».

Очевидно, что «сырые» данные должны быть переработаны в данные, полезные на практике, посредством обработки и представления их в структурированном виде (например, как в реляционных базах данных) или в виде, приемлемом для выполнения статистического исследования.

Основными типами структурированных данных являются *числовой* (*количественный*) и *категориальный* (*качественный*) типы.

Числовые (*количественные*) *данные* – данные, измеряемые с помощью чисел, имеющих содержательный смысл.

Числовые (*количественные*) *данные* могут *непрерывными* (continuous data) и *дискретными* (discrete data).

Числовые (*количественные*) *данные* являются *непрерывными*, если множество их возможных значений представляет собой некоторый конечный или бес-

конечный промежуток числовой оси, т.е. такие данные принимают свои значения на непрерывной шкале значений.

Числовые (*количественные*) *данные* являются *дискретными*, если множество их возможных значений конечно или счётно, т.е. такие данные принимают свои значения из ограниченного набора значений, обычно представленных целыми числами.

К *непрерывным числовым данным*, например, относятся температура тела, показатель гемоглобина в крови, рост и вес человека, а к *дискретным числовым данным* – число детей в семье, число рабочих дней в месяце и число дней в больничном листе.

Над *непрерывными данными* можно производить арифметические операции сложения, вычитания, умножения, деления, и они имеют смысл. По отношению к *дискретным данным* применение арифметических операций также возможно, но необходимо внимательно следить за сохранением типа данных и смыслом выполняемых операций.

Непрерывные данные могут быть преобразованы в дискретные с помощью операции квантования, т.е. посредством замены значений непрерывных данных отрезками, каждый из которых представляет некоторый диапазон.

Категориальные (*качественные*) *данные* – данные, которые могут принимать значения только из некоторого фиксированного набора значений, т.е. это данные с ограниченным числом уникальных значений или категорий.

Категориальные (*качественные*) *данные* – данные, описывающие качество и не имеющие количественного выражения. В *категориальных* (*качественных*) *данных* каждая единица наблюдения назначается определенной группе или номинальной категории на основе некоторого качественного свойства.

Категориальные (*качественные*) данные могут быть *номинальными* (nominal data) и *порядковыми* (ordered data), которые отражают соответственно условные коды неизмеримых категорий или условную степень выраженности признака.

К *качественным номинальным данным*, например, относится название применяемого для лечения препарата, название заболевания пациента.

Значения *категориальных* (*качественных*) *порядковых* данных могут быть ранжированы по какому-либо принципу, но интервал между значениями таких данных не может быть выражен количественно. Обычно *категориальные* (*качественные*) *порядковые данные* качественно отражают условную степень выраженности какого-либо признака, например, тяжесть состояния больного при

поступлении в стационар (тяжёлое, средней тяжести и т.д.), степень ожога (1, 2, 3 или 4), группа инвалидности (первая, вторая или третья).

При работе с *категориальными* (*качественными*) *данными* применяются только операции сравнения (такие, как «равно» и «не равно») и производится упорядочение данных, например, по алфавиту. Применение арифметических операций к категориальным данным некорректно, даже если они представлены числами.

Особый случай категориальных данных – *бинарные данные*, которые принимают только одно из двух допустимых альтернативных значений: 0 или 1 («да» или «нет»; «истина» или «ложь»; «здоров» или «болен»).

В зависимости от того, каков тип данных – числовой или категориальный – определяется тип применяемого в дальнейшем инструмента разведочного анализа: тип визуального отображения, тип анализа данных, тип статистической модели и т.п.

Знание о том, каков тип данных, используется как для улучшения вычислительной производительности в процессе анализа данных, так и для определения того, какие операции (вычисления) допустимы для анализируемых данных.

Зачастую возникает вопрос: зачем нужны такие понятия, как «категориальные» и «порядковые» данные, если, по сути, категории являются набором текстовых (либо числовых) значений, и при работе на компьютере автоматически используется их внутреннее представление? Оказывается, что четкая идентификация категориальных данных (в отличие от текстовых) даёт ряд преимуществ. Во-первых, информация о том, что данные категориальные, может учитываться при работе со статистическими инструментами (например, при построении графика или подгонке модели). В частности, порядковые данные могут быть представлены как порядковый фактор с сохранением определенной пользователем упорядоченности в моделях, таблицах и графиках. Во-вторых, информация о том, что данные категориальные, может быть учтена в процедурах оптимизации для организации хранения и индексации данных. В-третьих, значения, которые принимает конкретная категориальная переменная, могут реализовываться в программном обеспечении (например, перечисление ENUM).

1.2. Представление данных

При выполнении разведочного анализа данных *обычно* принято работать с так называемыми «прямоугольными» данными, т.е. данными, которые можно представить в виде таблицы, например, в виде электронной таблицы в MS Excel или таблицы базы данных. Очень часто для представления табличных данных

используют текстовый формат *CSV* (*Comma-Separated Values* – значения, разделённые запятыми). Файлы с расширением csv (csv-файлы) удобно использовать для представления как исходных, так и итоговых наборов данных при выполнении разведочного анализа данных на языке Python.

«Прямоугольные» данные, по сути, представляют собой двумерную матрицу. Строки матрицы являются записями (признаковыми описаниями объектов, т.е. векторами, которые составлены из значений, соответствующих некоторому набору признаков, описывающих объект). Столбцы матрицы соответствуют признакам (при этом значения признаков могут быть разного типа). В этом случае говорят, что «прямоугольные» данные – это матрица «объекты × признаки». Такое представление данных является стандартным и наиболее распространённым в задачах разработки статистических моделей и моделей машинного обучения (например, в задачах кластеризации, классификации и регрессионного анализа).

Любые неструктурированные данные (например, текст) необходимо обработать так, чтобы их можно было представить как набор признаков в матрице «объекты × признаки».

При выполнении разведочного анализа данных на языке Python целесообразно использовать программную библиотеку `pandas` (https://pandas.pydata.org/), в которой основной прямоугольной структурой данных является объект DataFrame, содержащий таблицу данных. По умолчанию для DataFrame создается автоматический целочисленный индекс, который основывается на порядке следования строк. Для повышения эффективности некоторых операций в программной библиотеке `pandas` можно задавать многоуровневые/иерархические индексы.

В процессе статистического исследования рассматривают набор данных, полученных в результате измерения одного или нескольких признаков у тех или иных объектов.

Реально наблюдаемая совокупность объектов, статистически представленная рядом наблюдений $x_1, x_2, ..., x_n$ случайной величины X, является выборкой, а гипотетически существующая (домысливаемая) совокупность объектов – генеральной совокупностью.

Генеральная совокупность может быть конечной или бесконечной, а выборка из генеральной совокупности всегда является результатом ограниченного ряда n наблюдений.

Следует отметить существование некоторых различий в терминологии, имеющих место в статистике и науке о данных (data science), при описании од-

них и тех же сущностей. Так, в статистике принято говорить о предикторных переменных (предикторах), которые используются для предсказания зависимой переменной (отклика), а в науке о данных принято говорить о *признаках*, которые используются для предсказания *целевой* переменной. В статистике для значений, вычисляемых на основе имеющихся данных, обычно применяется термин «оценка». Использование этого термина позволяет отличать вычисленные значения от теоретически истинных и учитывать возможную неопределенность данных. В науке о данных, интеллектуальном анализе данных и бизнес-аналитике вместо термина «оценка» обычно используют термин «метрика» («метрический показатель»), т.к. в центре внимания находятся конкретные данные, анализ которых выполняется с конкретными деловыми или организационными целями. Таким образом, в статистике принято *оценивать*, а в аналитике – *измерять*.

2. Обработка пропусков данных

Зачастую наборы данных содержат пропуски, т.е. для некоторых объектов, соответствующих строкам набора данных, имеет место отсутствие значений по некоторым признакам, соответствующим столбцам набора данных.

Очевидно, что такие наборы данных должны быть каким-либо образом преобразованы, для того чтобы к ним можно было применить те или иные средства разведочного анализа данных, а также – чтобы преобразованные наборы данных можно было бы использовать при решении задач кластеризации, классификации, прогнозирования и т.п.

Самый простой подход к решению проблемы пропусков данных – это удаление из набора данных информации об объектах с пропусками значений по некоторым признакам, т.е. удаление строк набора данных, соответствующих этим объектам. Однако применение этого подхода может привести к тому, что набор данных станет не репрезентативным, поскольку будет удалено очень много строк.

Альтернативой подходу, реализующему удаление из набора данных информации об объектах с пропусками значений по некоторым признакам, является подход, предполагающий заполнение пропусков данных некоторыми значениями. Этот подход применим как в случае работы с количественными признаками, так и с качественными. Однако при применении этого подхода целесообразно внимательно изучить саму природу набора данных, а также – особенности (специфику) признаков, для которых будет осуществляться заполнение пропусков значений.

Заполнение пропусков в случае количественных признаков может быть осуществлено, например, на основе метрик центрального значения. Так, могут быть использованы такие метрики, как *среднее* и *медиана*. Заполнение пропусков в случае качественных признаков может быть осуществлено, например, на основе наиболее часто встречающегося значения признака.

Возможно, что при заполнении пропусков значений одного признака следует предварительно обратить внимание на значения других признаков. Например, в случае пропусков в наборе данных значений по признаку *возраст*, следует сначала проанализировать значения по признаку *пол* (если он есть в наборе данных), и осуществить заполнение пропусков по признаку *возраст* с учетом гендерной принадлежности, т.е. выполнить вычисление *среднего* или *медианы* для женщин и мужчин по отдельности с целью заполнения пропусков значений с учетом пола.

Отметим, что заполнение пропусков данных обычно кажется более предпочтительным решением. Однако это не всегда так. Неудачный выбор подхода к заполнению пропусков данных может не только не улучшить, но и сильно ухудшить результаты (например, при использовании таких наборов данных при решении задач кластеризации, классификации, прогнозирования и т.п.).

Естественно, что после применения различных подходов (например, таких метрик, как *среднее* и *медиана*) к заполнению пропусков данных итоговые значения метрик будут отличаться друг от друга.

3. Метрики для категориальных признаков

Категориальный признак – признак, значения которого определяют принадлежность объекта, описываемого признаком, к некоторой категории.

Категориальные признаки могут принимать большое число различных значений из числа допустимых. В качестве примеров категориальных признаков можно назвать, например, гражданство, национальность, уровень образования, должность, специальность, издательство, тарифный план, область науки, материал некоторой конструкции и т.п.

Бинарные признаки являются частным случаем категориальных. Бинарные признаки могут принимать одно из двух допустимых значений. В качестве примеров бинарных признаков можно назвать, например, пол человека (значения: мужской, женский), истинность высказывания (значения: ложь, истина), состояние здоровья человека (значения: болен, здоров).

При выполнении обобщения признака необходимо получить его «типичное значение», т.е. понять, где расположено большинство его значений (какова их центральная тенденция).

Представление о бинарных и категориальных с несколькими категориями признаках можно получить, вычислив доли или процентные соотношения для их значений.

Например, для бинарного признака, принимающего значение 0 или 1, можно вычислить долю нулей и долю единиц.

В общем случае можно вычислить доли для всех значений категориального признака (для всех категорий).

Зная доли, можно вычислить соответствующие им процентные соотношения.

Категории могут представлять отличающиеся объекты (например, бакалавры и магистранты; мужчины и женщины), уровни значений признака (лёгкий, средний, тяжёлый), числовые данные, которые были разбиты на частотные интервалы.

В табл. 1 приведен пример процентных соотношений для национального состава населения по данным Всероссийской переписи 2021 года (с указанием процентных соотношений для 10 наиболее крупных народов и всех остальных) с учетом информации о лицах, указавших национальность.

Таблица 1. Национальный состав населения по данным Всероссийской переписи 2021 года (в процентах)

Русские	Татары	Чеченцы	Башкиры	Чуваши	Аварцы	Армяне	Украинцы	Даргинцы	Казахи	Остальные
80,85%	3,61%	1,28%	1,20%	0,82%	0,78%	0,72%	0,68%	0,48%	0,45%	9,13%

Для визуализации распределения значений категориального признака используют *столбчатые диаграммы* (*bar charts*): категории отображают по оси *x*, а частоты (доли, проценты) – по оси *y*.

Столбчатая диаграмма отображает частоту (долю, проценты) каждой категории в виде прямоугольника соответствующей высоты.

Столбчатая диаграмма напоминает гистограмму. В столбчатой диаграмме ось *x* представляет разные категории категориального признака, в то время как на гистограмме ось *x* представляет значения количественного признака на числовой шкале. На столбчатой диаграмме прямоугольники располагают отдельно друг от друга. На гистограмме прямоугольники обычно располагают впритык друг к другу, разрывы на гистограмме указывают на то, что значения в данных отсутствуют.

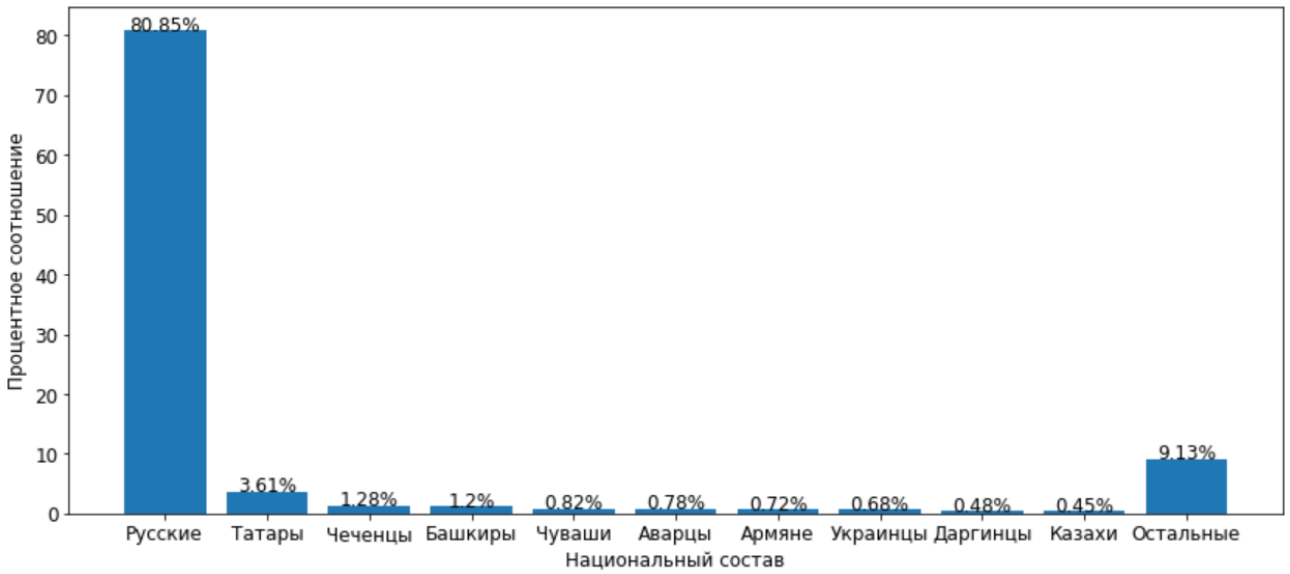

Рисунок 1. Столбчатая диаграмма

Круговые диаграммы (*pie charts*) являются альтернативой столбчатым диаграммам. Однако считается, что они менее визуально информативны.

Круговая диаграмма отображает частоту (долю, проценты) каждой категории в виде сектора круга соответствующего размера.

На рис. 1 приведена столбчатая диаграмма для данных табл. 1.

На рис. 2 приведена круговая диаграмма для данных табл. 1.

Рисунок 2. Круговая диаграмма

Следует отметить, что при работе с количественными данными могут быть построены частотные таблицы на основе разбивки данных на частотные интервалы. В результате количественные данные неявно преобразуются в порядковые. При этом можно сказать, что гистограммы и столбчатые диаграммы подобны, но категории на оси *x* в столбчатой диаграмме не упорядочены.

Преобразование количественных данных в категориальные позволяет уменьшить размер и сложность данных, что позволяет выявлять связи между признаками.

Мода (*mode*) – значение, которое появляется в данных чаще других.

Мода – наиболее часто встречающаяся категория или значение в наборе данных. Вообще говоря, мод может быть несколько.

Мода для данных из табл. 1 – категория «Русские».

Мода представляет собой статистику (сводную статистическую величину) для категориальных данных.

Особый тип категориальных данных – данные, в которых категории представлены дискретными значениями или могут быть сопоставлены с ними на одинаковой шкале измерения. Для таких данных можно вычислить *математическое ожидание* (*expected value*), которое будет представлять собой среднее взвешенное, в котором весами являются вероятности.

Такое *математическое ожидание* вычисляется в соответствии со следующей последовательностью шагов.

1. Умножить каждый исход на вероятность его наступления.
2. Просуммировать эти значения.

Таким образом, если категории связаны с числовыми значениями, математическое ожидание дает среднее значение на основе вероятности появления категории.

Вычисленное таким образом *математическое ожидание* оценивает будущие ожидания и базируется на весах (вероятностях), которые очень часто основаны на субъективных суждениях.

Математическое ожидание может быть использовано для оценки бизнеса и при составлении бюджета долгосрочных расходов. Например, используя математическое ожидание, можно оценить возможную пяти- или десятилетнюю прибыль от приобретения нового оборудования, ожидаемое сокращение затрат от новой программной системы учета клиентов в компании и т.п.

Пример.

Компания предлагает два варианта обслуживания (сопровождения) клиентов: один по цене 500 рублей в месяц, а другой – по цене 100 рублей в месяц. При этом предлагается, что 5% клиентов согласятся на обслуживание за 500 рублей в месяц, 15% согласятся на обслуживание за 100 рублей в месяц и 80% откажутся от обслуживания.

Математическое ожидание выплат со стороны клиента в таком случае составит: $0.05 \cdot 500 + 0.15 \cdot 100 + 0.80 \cdot 0 = 40$ рублей в месяц.

4. Таблица сопряженности

Для обобщения двух категориальных признаков удобно использовать *таблицу сопряженности* (*таблицу контингентности*, *факторную таблицу*), которая представляет собой таблицу количеств по категориям. *Таблица сопряженности* может содержать только количества по категориям либо также включать столбцовые и итоговые проценты.

Таблица сопряжённости – инструмент представления совместного распределения двух признаков, позволяющий исследовать связь между ними.

Таблица сопряжённости – универсальный инструмент изучения статистических связей между признаками, т.к. в ней могут быть представлены два признака с любым уровнем измерения.

Таблица сопряжённости применяется для проверки гипотезы о наличии связи между двумя признаками на основе критерия согласия Пирсона или точного теста Фишера (теста статистической значимости, используемого в анализе таблиц сопряжённости для выборок маленьких размеров).

Строки *таблицы сопряжённости* соответствуют значениям одного признака, столбцы – значениям другого признака, при этом количественные шкалы предварительно группируются в интервалы.

Например, *таблица сопряжённости* может быть использована для выявления зависимости предпочтений в сфере образования от места проживания.

На пересечении строки и столбца таблицы сопряженности указывается частота совместного появления f_{ij} соответствующих значений двух признаков x_i и y_j ($i=\overline{1,n}$; $j=\overline{1,m}$). Сумма частот по строке r_i ($r_i=\sum_{j=1}^{m} f_{ij}$) называется маргинальной частотой строки; сумма частот по столбцу c_j ($c_j=\sum_{i=1}^{n} f_{ij}$) – маргинальной частотой столбца.

Сумма маргинальных частот равна объёму выборки V. Распределение маргинальных частот представляет собой одномерное распределение переменной, образующей строки или столбцы таблицы. В таблице сопряжённости могут быть представлены как относительные частоты (в долях или процентах), так и абсолютные частоты.

Относительные частоты могут рассчитываться по отношению:

- к маргинальной частоте по строке;
- к маргинальной частоте по столбцу;
- к объёму выборки.

Табл. 2 демонстрирует пример таблицы сопряженности.

Таблица 2. Таблица сопряженности

	y_1	y_2	y_3	y_4	
x_1	f_{11}	f_{12}	f_{13}	f_{14}	r_1
x_2	f_{21}	f_{22}	f_{23}	f_4	r_2
	c_1	c_2	c_3	c_4	V

5. Категориальные и количественные данные

Коробчатая диаграмма (*ящик с усами*) (см. Демидова Л.А. Разведочный анализ данных. Python. Часть 1 [Электронный ресурс]: Учебное-методическое пособие / Демидова Л.А. – М.: МИРЭА – Российский технологический университет, 2022) представляет простой способ визуального сравнения распределения количественного признака, значения которого сгруппированы согласно категориальному признаку [8].

Коробчатая диаграмма позволяет в компактном виде изобразить одномерное распределение вероятностей: на диаграмме отображаются медиана (или, если нужно, среднее), нижний и верхний квартили, минимальное и максимальное значение выборки и выбросы. Расстояния между различными частями коробчатой диаграммы позволяют определить степень разброса и асимметрии данных. Отметим, что асимметрию данных можно выявить не только по расположению медианы, смещённой к какому-либо концу «коробки», но и по разной длине усов, выходящих из «коробки».

Скрипичный график [8] является дополнением к коробчатой диаграмме. Он используется для визуализации распределения данных и их плотности вероятности. Скрипичный график сочетает в себе диаграмму размаха и график плотности, развернутые и расположенные по обе стороны для отображения формы распределения данных.

Коробчатая диаграмма имеет ограничение в отображении данных, т.к. его визуальная простота скрывает ряд существенных деталей относительно того, каким образом распределяются значения данных. Например, с помощью коробчатой диаграммы нельзя увидеть, какое распределение имеют данные (является ли распределение, например, бимодальным или мультимодальным). *Скрипичный график* позволяет отобразить больше информации, но при этом он более «зашумлен», чем коробчатая диаграмма. В частности, *коробчатая диаграмма* более ясно показывает выбросы в данных.

В целом следует отметить, что коробчатые диаграммы и скрипичные графики позволяют представлять количественный признак в сопоставлении с категориальным.

6. Кодирование категориальных признаков

Многие модели машинного обучения работают только с количественными данными. В связи с этим требуется выполнение перекодировки категориальных признаков в количественные с применением тех или иных методов кодирования [2].

6.1. Label Encoder

`Label Encoder` – один из наиболее часто используемых методов кодирования, при котором устанавливается однозначное соответствие между уникальным значением категориального признака и некоторым числовым значением.

Выбранное по определенному правилу первое уникальное значение категориального признака кодируется нулем, второе – единицей и т.д., последнее уникальное значение категориального признака кодируется числом $N - 1$, где N – количество уникальных значений категориального признака.

Например, правило преобразования у кодировщика `LabelEncoder` из `sklearn.preprocessing` предполагает сортировку по алфавиту уникальных значений категориального признака с последующим присвоением им порядковых номеров.

Метод `Label Encoder` используют, когда категории имеют естественный порядок или взаимосвязь друг с другом, например, в случае порядковых признаков, таких как «холодный», «тёплый» и «горячий». Целочисленные значения, присвоенные категориям, должны отражать порядок категорий. Метод `Label Encoder` может быть полезным, когда количество категорий очень велико: в этом случае удается избежать увеличения размерности данных.

Главным недостатком кодирования по методу `Label Encoder` является создание избыточных зависимостей в данных. Например, после такого кодирования можно будет сказать, что одно значение категориального признака важнее (больше и т.д.), чем другое в некоторое количество раз (если вычислить отношение их кодов). Однако анализ исходных данных не позволяет сделать такие однозначные выводы (поскольку их, скорее всего, и нет).

6.2. One-Hot Encoder

`One-Hot Encoder` – метод кодирования, который основывается на создании бинарных признаков, которые показывают принадлежность к уникальному значению. При этом количество бинарных признаков равно количеству уникальных значений категориального признака. Значение конкретного бинарного

признака у модифицированного объекта равно 1, если у исходного объекта категориальный признак имеет значение, соответствующее названию рассматриваемого бинарного признака; в противном случае значение бинарного признака равно 0.

`One-Hot Encoder` используют, когда категории не имеют внутреннего порядка или отношений друг с другом. Это связано с тем, что однократное кодирование рассматривает каждую категорию как отдельный признак, не связанный с другими категориями.

Главным недостатком кодирования по методу `One-Hot Encoder` является значительное увеличение объема данных, т.к. признаки с большим числом уникальных значений кодируются большим числом бинарных признаков.

Метод `One-Hot Encoder` чаще используется в приложениях машинного обучения, поскольку он более гибкий и позволяет избежать проблем неоднозначности и произвольного порядка категорий в данных. Однако метод `Label Encoder` может быть полезен в определенных контекстах, когда категории имеют естественный порядок или при работе с очень большим количеством категорий.

6.3. Ordinal Encoder

`Ordinal Encoder` – метод кодирования, реализующий кодирование значений категориального признака с сохранением заданного для них порядка. В результате значения категориального признака преобразуются в порядковые целые числа, что приводит к одному столбцу целых чисел (со значениями от 0 до $N - 1$, где N – количество уникальных значений категориального признака).

Следует также отметить существование и таких методов кодирования, как `Binary Encoder`, `Contrast Encoder` (`Helmert Encoder` и `Backward-Difference Encoder`), `Target Encoder` (`Target Encoder`, `Leave-One-Out Encoder` и `James-Stein Encoder`).

7. Разведочный анализ для наборов данных с категориальными признаками на языке Python

Для демонстрации принципов работы библиотек и модулей, реализующих инструменты разведочного анализа данных, будем использовать `Jupyter Notebook`.

Для работы с метриками центральной тенденции для категориальных признаков можно воспользоваться различными программными библиотеками и

модулями. В частности, можно использовать программные инструменты, предложенные по ссылкам:

https://pypi.org/project/statistics/,
https://numpy.org/ (https://pypi.org/project/numpy/),
https://pandas.pydata.org/.

7.1. Разведочный анализ с использованием модуля statistics

Для установки программного модуля `statistics` [9] необходимо выполнить команду:

```
pip install statistics
```

Подробная документация по всем функциям, реализованным в программном модуле `statistics`, приведена по ссылке:

https://docs.python.org/3/library/statistics.html.

Модуль `statistics` предоставляет функции для вычисления математической статистики, в том числе – метрик центральной тенденции для категориальных признаков.

Ниже приведена информация по двум функциям, используемым для вычисления метрик центральной тенденции.

1. Функция `mode`.

Возвращает одиночную моду (одно наиболее частое значение из дискретных или номинальных данных). Мода (если она существует) является наиболее типичной величиной и служит мерой центральной тенденции. Если имеется несколько мод с одинаковой частотой, функция возвращает моду, первой обнаруженную в данных.

Синтаксис:

`statistics.mode(`*`data`*`)`

Подробная информация по функции приведена по ссылке:

https://docs.python.org/3/library/statistics.html#statistics.mode.

2. Функция `multimode`.

Возвращает список мод (наиболее часто встречающихся значений) в порядке их первого появления в данных. Возвращает более одного результата, если есть несколько мод.

Синтаксис:

`statistics.multimode(`*`data`*`)`

Подробная информация по функции приведена по ссылке:

https://docs.python.org/3/library/statistics.html#statistics.multimode.

7.2. Разведочный анализ с использованием библиотеки NumPy

Для установки программной библиотеки NumPy [10] необходимо выполнить команду:

```
pip install numpy
```

Модуль Statistics из программной библиотеки NumPy содержит, в частности, функцию average, позволяющую вычислить метрику центральной тенденции с учётом весовых коэффициентов.

Ниже приведена информация по функции average.

Функция numpy.average возвращает взвешенное среднее вдоль выбранной оси массива.

Синтаксис:

```
numpy.average(a, axis=None, weights=None, returned=False, *, keepdims=<no value>)
```

Подробная документация по функциям, предназначенным для решения задач в области статистики, приведена по ссылке:

https://numpy.org/doc/stable/reference/generated/numpy.average.html.

7.3. Разведочный анализ с использованием библиотеки pandas

Библиотека pandas [11] – библиотека на языке программирования Python для обработки и анализа данных. Работа библиотеки pandas с данными строится поверх библиотеки NumPy, являющейся инструментом более низкого уровня. Библиотека pandas предоставляет специальные структуры данных и операции для манипулирования таблицами данных и временными рядами.

Библиотека pandas доступна по ссылке:

https://pandas.pydata.org/.

7.3.1. Справочные методы

Библиотека pandas предоставляет возможности для получения справочной информации по анализируемому набору данных.

Ниже приведена информация по двум методам, наиболее часто используемым для получения справочной информации.

1. Метод pandas.DataFrame.info.

Метод выводит краткую сводку по датафрейму, включая типы столбцов (признаков), информацию по ненулевым значениям и использованию памяти.

Синтаксис:

```
DataFrame.info(verbose=None, buf=None, max_cols=None, memory_usage=None, show_counts=None)
```

Подробная информация по методу приведена по ссылке:
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.info.html.

2. Метод `pandas.DataFrame.describe`.

Метод выводит описательную статистику, которая включает те метрики, которые обобщают центральную тенденцию, дисперсию и форму распределения набора данных, за исключением значений NaN. Метод анализирует как числовые ряды, так и ряды объектов, а также наборы столбцов датафрейма со смешанными типами данных.

Для количественных данных описательная статистика будет включать количество `count`, среднее `mean`, стандартное отклонение `std`, минимальное `min` и `max` максимальное значения, а также нижний, 50-й и верхний процентили. По умолчанию нижний процентиль равен 25, а верхний процентиль равен 75. 50-й процентиль соответствует медиане.

Для объектных (категориальных) данных описательная статистика будет включать число `count`, уникальность `unique`, моду `top` и частоту `freq`. Мода `top` – наиболее распространенное значение. Частота `freq` –частота наиболее распространенного значения. Временные метки включают первый (`first`) и последний (`last`) элементы. Если несколько значений признака типа `object` имеют одинаковое наибольшее количество, то `count` и `top` будут произвольно выбраны среди тех, для которых зафиксировано наибольшее количество.

Для смешанных типов данных по умолчанию возвращается только описательная статистика количественных столбцов. Если датафрейм не содержит какие-либо количественные столбцы, по умолчанию возвращается описательная статистика как объектных, так и категориальных столбцов.

Если значение параметра `include` метода `describe` установлено как `include='all'`, описательная статистика будет включать объединение описаний признаков каждого типа.

Синтаксис:

`DataFrame.describe(`*`percentiles=None, include=None, exclude=None`*`)`

Подробная информация по методу приведена по ссылке:
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html.

7.3.2. Методы для построения сводной таблицы и таблицы сопряженности

Библиотека `pandas` предоставляет широкий спектр возможностей для извлечения сводной информации и построения таблиц сопряженности на основе анализируемого набора данных.

Ниже приведена информация по некоторым наиболее часто используемым методам.

1. Метод `pandas.pivot`.

Метод возвращает измененный датафрейм (`DataFrame`), организованный по заданным значениям индекса/столбца. Метод позволяет изменить форму данных (создать «сводную» таблицу) на основе значений столбцов. Она использует уникальные значения из указанного индекса/столбцов для формирования осей результирующего датафрейма. Метод не поддерживает агрегацию данных, множественные значения приведут к мультииндексу (`MultiIndex`) в столбцах.

Синтаксис:

`pandas.pivot(`*`data, *, columns, index=typing.Literal[<no_default>], values=typing.Literal[<no_default>]`*`)`

Подробная информация по методу приведена по ссылке:

https://pandas.pydata.org/docs/reference/api/pandas.pivot.html.

2. Метод `pandas.pivot_table`.

Метод создает сводную таблицу в стиле электронной таблицы как датафрейм (`DataFrame`). Уровни в сводной таблице будут храниться в объектах `MultiIndex` (иерархических индексах) в индексе и столбцах результирующего датафрейма.

Синтаксис:

`pandas.pivot_table(`*`data, values=None, index=None, columns=None, aggfunc='mean', fill_value=None, margins=False, dropna=True, margins_name='All', observed=False, sort=True`*`)`

Подробная информация по методу приведена по ссылке:

https://pandas.pydata.org/docs/reference/api/pandas.pivot_table.html.

Метод `pivot_table` – обобщение метода `pivot`, которая может обрабатывать повторяющиеся значения для одной сводной пары индекс/столбец.

Для метода `pivot_table` можно указать список функций агрегирования, используя ключевой параметр `aggfunc`. По умолчанию параметр `aggfunc` для метода `pivot_table` – `numpy.mean`. Кроме того, метод `pivot_table` поддерживает использование нескольких столбцов для индекса и столбца сводной таблицы. При этом будет автоматически сгенерирован иерархический индекс.

3. Метод `pandas.crosstab`.

Метод `crosstab` используется для вычисления таблицы сопряженности (перекрестной таблицы) двух (или более) признаков. По умолчанию `crosstab`

вычисляет таблицу частот признаков, если не переданы массив значений и функция агрегирования.

Синтаксис:

`pandas.crosstab(index, columns, values=None, rownames=None, colnames=None, aggfunc=None, margins=False, margins_name='All', dropna=True, normalize=False)`

Подробная информация по методу приведена по ссылке:

https://pandas.pydata.org/docs/reference/api/pandas.crosstab.html.

Следует отметить, что в библиотеке pandas имеется специальный класс `Categorical` для представления категориальных данных:

`pandas.Categorical`.

Этот класс представляет категориальную переменную в классическом стиле R/S-plus. Категориалы могут принимать только ограниченное и обычно фиксированное количество возможных значений (категорий). В отличие от статистических категориальных переменных, категориалы могут иметь порядок, но применение операций над числами (сложение, деление и т.д.) к ним невозможно. Все значения категориала находятся либо в категориях, либо в np.nan. Присвоение значений вне категорий вызывает ошибку ValueError. Порядок определяется порядком категорий, а не лексическим порядком значений.

Синтаксис:

`class pandas.Categorical(values, categories=None, ordered=None, dtype=None, fastpath=False, copy=True)`

Подробная информация по классу приведена по ссылке:

https://pandas.pydata.org/docs/reference/api/pandas.Categorical.html.

7.4. Методы визуализации результатов разведочного анализа данных

Для визуализации результатов разведочного анализа данных могут быть использованы возможности различных программных библиотек. Наиболее часто осуществляется работа с программными библиотеками `matplotlib`, `seaborn`, `pandas`.

Ниже приведено краткое описание некоторых функций, реализующих методы визуализации. Кроме того, для каждой функции приведена ссылка на Интернет-страницу с подробной документацией.

7.4.1. Визуализация с применением библиотеки matplotlib

Библиотека `matplotlib` – библиотека на языке программирования Python для визуализации данных с применением двумерной графики [12].

Эта библиотека доступна по ссылке:

https://matplotlib.org/.

Графические инструменты, применяемые в статистике и разведочном анализе данных, доступны по ссылке:

https://matplotlib.org/stable/plot_types/stats/index.html.

Информация по некоторым функциям, реализующим методы визуализации, приведена ниже.

1. Функция `matplotlib.pyplot.bar`.

Функция реализует построение столбчатой диаграммы. Прямоугольники располагаются по `x` с заданным выравниванием. Размеры прямоугольников указываются по высоте и ширине. Вертикальная базовая линия – нижняя (по умолчанию `0`). Многие параметры могут принимать либо одно значение, применимое ко всем столбцам, либо последовательность значений, по одному для каждого столбца.

Синтаксис:

`matplotlib.pyplot.bar(x, height, width=0.8, bottom=None, *, align='center', data=None, **kwargs)`

Подробная информация по функции приведена по ссылке:

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html

2. Функция `matplotlib.pyplot.pie`.

Функция реализует построение круговой диаграммы массива `x`. Площадь каждого клина определяется как `x/sum(x)`. Клинья строятся против часовой стрелки, по умолчанию начиная с оси `x`.

Синтаксис:

`matplotlib.pyplot.pie(x, explode=None, labels=None, colors=None, autopct=None, pctdistance=0.6, shadow=False, labeldistance=1.1, startangle=0, radius=1, counterclock=True, wedgeprops=None, textprops=None, center=(0, 0), frame=False, rotatelabels=False, *, normalize=True, hatch=None, data=None)`

Подробная информация по функции приведена по ссылке:

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.pie.html.

7.4.2. Визуализация с применением библиотеки seaborn

Библиотека `seaborn` – библиотека визуализации данных на языке программирования Python, основанная на библиотеке `matplotlib`. Эта библио-

тека предоставляет высокоуровневый интерфейс для построения привлекательных и информативных статистических графиков.

Библиотека `seaborn` [13] доступна по ссылке:

https://seaborn.pydata.org/.

Информация по некоторым функциям, реализующим методы визуализации, приведена ниже.

1. Функция `seaborn.stripplot`.

Функция реализует построение категориальной диаграммы рассеивания. Ленточная диаграмма (в виде вытянутых в ленту облаков данных) может быть использована самостоятельно. Также она является хорошим дополнением к коробчатой или скрипичной диаграмме в тех случаях, когда требуется показать все наблюдения вместе с некоторым представлением основного распределения.

Синтаксис:

`seaborn.stripplot(`*`data=None, *, x=None, y=None, hue=None, order=None, hue_order=None, jitter=True, dodge=False, orient=None, color=None, palette=None, size=5, edgecolor='gray', linewidth=0, hue_norm=None, native_scale=False, formatter=None, legend='auto', ax=None, **kwargs`*`)`

Подробная информация по функции приведена по ссылке:

https://seaborn.pydata.org/generated/seaborn.stripplot.html.

2. Функция `seaborn.swarmplot`.

Функция реализует построение категориальной диаграммы с точками данных, скорректированными (только вдоль категориальной оси) так, чтобы они не перекрывались. Эта диаграмма похожа на `seaborn.stripplot`, но дает лучшее представление о распределении значений, однако плохо масштабируется для большого числа точек данных. Такой стиль диаграммы иногда называют «пчелиным». Диаграмма может быть использована самостоятельно. Также она является хорошим дополнением к коробчатой или скрипичной диаграмме в тех случаях, когда требуется показать все наблюдения вместе с некоторым представлением основного распределения.

Синтаксис:

`seaborn.swarmplot(`*`data=None, *, x=None, y=None, hue=None, order=None, hue_order=None, dodge=False, orient=None, color=None, palette=None, size=5, edgecolor='gray', linewidth=0, hue_norm=None, native_scale=False, formatter=None, legend='auto', warn_thresh=0.05, ax=None, **kwargs`*`)`

Подробная информация по функции приведена по ссылке:

https://seaborn.pydata.org/generated/seaborn.swarmplot.html#seaborn.swarmplot.

3. Функция `seaborn.boxplot`.

Функция реализует построение коробчатой диаграммы (диаграммы «ящик с усами»). Коробчатая диаграмма показывает распределение количественных данных: в коробке показаны квартили набора данных, усы показывают остальную часть распределения, за исключением данных, которые определены как «выбросы» с использованием метода, который является функцией межквартильного диапазона.

Синтаксис:

`seaborn.boxplot(`*`data=None, *, x=None, y=None, hue=None, order=None, hue_order=None, orient=None, color=None, palette=None, saturation=0.75, width=0.8, dodge=True, fliersize=5, linewidth=None, whis=1.5, ax=None, **kwargs`*`)`

Подробная информация по функции приведена по ссылке:

https://seaborn.pydata.org/generated/seaborn.boxplot.html.

4. Функция `seaborn.violinplot`.

Функция реализует построение скрипичного диаграммы, представляющей собой комбинацию коробчатой диаграммы с оценкой плотности ядра. Скрипичная диаграмма, как и коробчатая диаграмма, показывает распределение количественных данных. В отличие от коробчатой диаграммы, в которой все компоненты диаграммы соответствуют фактическим точкам данных, в скрипичной диаграмме используется оценка плотности ядра базового распределения данных. Скрипичная диаграмма позволяет эффективно и привлекательно показать сразу несколько распределений данных одновременно. При этом следует иметь в виду, что на процедуру оценки влияет размер выборки, поэтому скрипичные диаграммы для относительно небольших выборок могут выглядеть обманчиво гладкими.

Синтаксис:

`seaborn.violinplot(`*`data=None, *, x=None, y=None, hue=None, order=None, hue_order=None, bw='scott', cut=2, scale='area', scale_hue=True, gridsize=100, width=0.8, inner='box', split=False, dodge=True, orient=None, linewidth=None, color=None, palette=None, saturation=0.75, ax=None, **kwargs`*`)`

Подробная информация по функции приведена по ссылке:

https://seaborn.pydata.org/generated/seaborn.violinplot.html.

5. Функция `seaborn.boxenplot`.

Функция реализует построение улучшенной коробчатой диаграммы для больших наборов данных. Эта диаграмма ранее называлась диаграммой «буквенного значения» («letter value»), потому что показывала большое количество

квантилей, которые определены как «буквенные значения». Она похожа на блочную диаграмму при построении непараметрического представления распределения, в котором все признаки соответствуют фактическим наблюдениям. Построение большего количества квантилей дает больше информации о форме распределения, особенно о хвостах.

Синтаксис:

`seaborn.boxenplot(data=None, *, x=None, y=None, hue=None, order=None, hue_order=None, orient=None, color=None, palette=None, saturation=0.75, width=0.8, dodge=True, k_depth='tukey', linewidth=None, scale='exponential', outlier_prop=0.007, trust_alpha=0.05, showfliers=True, ax=None, box_kws=None, flier_kws=None, line_kws=None)`

Подробная информация по функции приведена по ссылке:

https://seaborn.pydata.org/generated/seaborn.boxenplot.html.

6. Функция `seaborn.pointplot`.

Функция реализует отображение точечных оценок и ошибок, используя точечные метки. Точечный график представляет собой оценку центральной тенденции для количественного признака по положению точки и дает некоторое представление о неопределенности этой оценки с помощью планок погрешностей. Точечные графики полезны для сравнения различных уровней одной или нескольких категориальных переменных. Они особенно хорошо показывают взаимодействия: как отношения между уровнями одной категориальной переменной меняются по уровням второй категориальной переменной. Линии, соединяющие каждую точку с одного и того же уровня оттенка, позволяют судить о взаимодействиях по разнице в наклоне, что визуально проще для глаз, чем сравнение высот нескольких групп точек или полос. Точечный график показывает только среднее (или другое оценочное) значение, но во многих случаях распределение значений на каждом уровне категориальных переменных может быть более информативным. В этих случаях целесообразно использовать другие диаграммы, например, коробчатую или скрипичную.

Синтаксис:

`seaborn.pointplot(data=None, *, x=None, y=None, hue=None, order=None, hue_order=None, estimator='mean', errorbar=('ci', 95), n_boot=1000, units=None, seed=None, markers='o', linestyles='-', dodge=False, join=True, scale=1, orient=None, color=None, palette=None, errwidth=None, ci='deprecated', capsize=None, label=None, ax=None)`

Подробная информация по функции приведена по ссылке:

https://seaborn.pydata.org/generated/seaborn.pointplot.html.

7. Функция `seaborn.barplot`.

Функция реализует построение столбчатой диаграммы с отображением точечных оценок и ошибок в виде прямоугольников. Столбчатая диаграмма отображает сравнение нескольких дискретных категорий. Она представляет собой оценку центральной тенденции для анализируемого признака с высотами прямоугольников, пропорциональными величинам, которые они отображают. Столбчатая диаграмма дает некоторое представление о неопределенности вокруг оценки центральной тенденции с использованием планок погрешностей. Важно иметь в виду, что столбчатая диаграмма показывает только среднее (или другое оценочное) значение. Иногда более подходящими могут оказаться другие диаграммы, например, коробчатая или скрипичная.

Синтаксис:

`seaborn.barplot(data=None, *, x=None, y=None, hue=None, order=None, hue_order=None, estimator='mean', errorbar=('ci',95), n_boot=1000, units=None, seed=None, orient=None, color=None, palette=None, saturation=0.75, width=0.8, errcolor='.26', errwidth=None, capsize=None, dodge=True, ci='deprecated', ax=None, **kwargs)`

Подробная информация по функции приведена по ссылке:

https://seaborn.pydata.org/generated/seaborn.barplot.html.

8. Функция `seaborn.countplot`.

Функция реализует отображение числа наблюдений в каждом категориальном интервале с помощью столбцов. График подсчета можно рассматривать как гистограмму категориального, а не количественного признака.

Синтаксис:

`seaborn.countplot(data=None, *, x=None, y=None, hue=None, order=None, hue_order=None, orient=None, color=None, palette=None, saturation=0.75, width=0.8, dodge=True, ax=None, **kwargs)`

Подробная информация по функции приведена по ссылке:

https://seaborn.pydata.org/generated/seaborn.countplot.html.

9. Функция `seaborn.catplot`.

Функция является универсальным инструментом для визуализации категориальных данных с применением различных специализированных инструментов. Она предоставляет интерфейс для создания категориальных графиков и диаграмм на уровне рисунков, а не на уровне осей (как, например, функция `seaborn.barplot`). При этом функция `seaborn.catplot` обеспечивает

доступ к нескольким функциям уровня осей, которые показывают отношения между числовыми и одним или несколькими категориальными признаками, используя одно из нескольких визуальных представлений. В частности, в качестве функций осей могут выступать такие функции, как:

– функции категориальных диаграмм рассеяния (categorical scatterplots): `seaborn.stripplot`, `seaborn.swarmplot`;

– функции категориальных графиков распределения (categorical distribution plots): `seaborn.boxplot`, `seaborn.violinplot`, `seaborn.boxenplot`;

– функции категориальных оценочных графиков (categorical estimate plots): `seaborn.pointplot`, `seaborn.barplot`, `seaborn.countplot`.

При использовании функции `seaborn.catplot` открывается много возможностей для выполнения наглядной визуализации. Используя параметр `kind` этой функции, можно выбрать тип (функцию) визуализации.

Синтаксис:

`seaborn.catplot(`*`data=None, *, x=None, y=None, hue=None, row=None, col=None, col_wrap=None, estimator='mean', errorbar=('ci', 95), n_boot=1000, units=None, seed=None, order=None, hue_order=None, row_order=None, col_order=None, height=5, aspect=1, kind='strip', native_scale=False, formatter=None, orient=None, color=None, palette=None, hue_norm=None, legend='auto', legend_out=True, sharex=True, sharey=True, margin_titles=False, facet_kws=None, ci='deprecated', **kwargs`)

Подробная информация по функции приведена по ссылке:
https://seaborn.pydata.org/generated/seaborn.catplot.html.

7.4.3. Визуализация с применением библиотеки pandas

Инструменты визуализации библиотеки `pandas` доступны по ссылке:
https://pandas.pydata.org/docs/reference/plotting.html.

Информация по некоторым функциям, реализующим методы визуализации, приведена ниже.

1. Функция `pandas.DataFrame.plot.bar`.

Функция реализует построение столбчатой диаграммы.

Синтаксис:

`DataFrame.plot.bar(`*`x=None, y=None, **kwargs`)

Подробная информация по функции приведена по ссылке:
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.bar.html.

2. Функция `pandas.plotting.boxplot`.

Функция реализует построение коробчатой диаграммы (диаграммы «ящик с усами») из столбцов DataFrame при необходимости сгруппированных по некоторым другим столбцам. «Ящик» простирается от квартильных значений данных Q_1 до Q_3 с линией в медиане Q_2. Усы отходят от краев «ящика», показывая диапазон данных. По умолчанию усы простираются не более чем на 1,5**IQR* (*IQR*=Q_3–Q_1) от краев «ящика», заканчиваясь в самой дальней точке данных в этом интервале. Выбросы наносятся отдельными точками.

Синтаксис:

`pandas.plotting.boxplot(`*`data, column=None, by=None, ax=None, fontsize=None, rot=0, grid=True, figsize=None, layout=None, return_type=None, **kwargs`*`)`

Подробная информация по функции приведена по ссылке:
https://pandas.pydata.org/docs/reference/api/pandas.plotting.boxplot.html.

8. Примеры выполнения разведочного анализа на языке Python

Ниже приведены примеры выполнения разведочного анализа данных для категориальных данных.

8.1. Примеры выполнения разведочного анализа с применением круговых диаграмм

В табл. 3 представлена информация о среднем числе ясных, облачных и пасмурных дней в Москве по месяцам года и в целом за год (https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%BC%D0%B0%D1%82_%D0%9C%D0%BE%D1%81%D0%BA%D0%B2%D1%8B).

В среднем за год в Москве бывает 91 ясный, 172 облачных и 102 пасмурных дня.

Отобразим сводную информацию по категориям «Ясный», «Облачный» и «Пасмурный» для Москвы (табл. 3) с помощью круговой диаграммы, используя разные библиотеки.

8.1.1. Примеры визуализации данных с применением круговых диаграмм на основе библиотеки matplotlib

На рис. 3 представлен фрагмент программного кода, реализующего вывод круговой диаграммы, позволяющей выполнить анализ распределения категорий. При этом используются значения параметров функции `pie` библиотеки

matplotlib, заданные по умолчанию. Кроме того, определены размеры рисунка.

Таблица 3. Среднее количество ясных, облачных и пасмурных дней в Москве

Месяц		Январь	Февраль	Март	Апрель	Май	Июнь	Июль	Август	Сентябрь	Октябрь	Ноябрь	Декабрь	Год
Дни	Ясные	8	9	10	8	11	7	8	10	8	5	3	4	91
	Облачные	11	10	12	17	16	20	20	17	16	14	9	10	172
	Пасмурные	12	9	9	5	4	3	3	4	6	12	18	17	102

На рис. 4 представлена круговая диаграмма, отражающая распределение категорий.

```
1  # импорт библиотек
2  import matplotlib.pyplot as plt
3
4  # данные
5  days = [91, 172, 102]
6  category = ['Ясный', 'Облачный', 'Пасмурный']
7
8  # круговая диаграмма
9  fig = plt.figure(figsize =(10, 7)) # размер рисунка
10 plt.pie(days, labels = category)
11
12 # визуализация
13 plt.show()
```

Рисунок 3. Фрагмент программного кода, реализующего вывод круговой диаграммы со значениями параметров функции pie, заданными по умолчанию

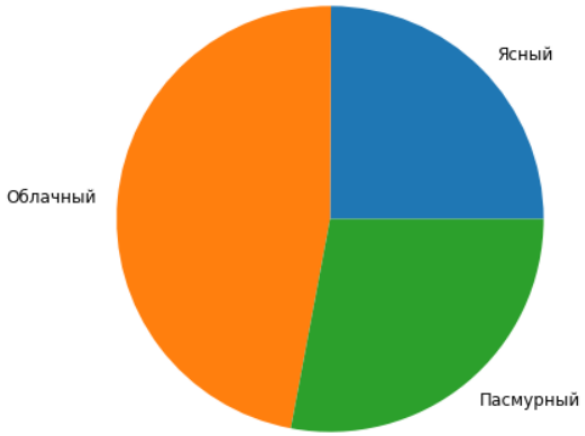

Рисунок 4. Круговая диаграмма
со значениями параметров функции pie, заданными по умолчанию

```python
# импорт библиотек
import matplotlib.pyplot as plt

# данные
days = [91, 172, 102]
category = ['Ясный', 'Облачный', 'Пасмурный']



# определение смещений секторов относительно центра
explode = (0.1, 0.0, 0.2)

# палитра
colors = ( "orange", "cyan", "indigo")

# свойства сектора круговой диаграммы
wp = { 'linewidth' : 1, 'edgecolor' : "green" }

# Функция для определения формата вывода текста на круговой диаграмме
def func(pct, allvalues):
    absolute = int(pct / 100.*np.sum(allvalues))
    return "{:.1f}%\n({:d} д.)".format(pct, absolute)

# круговая диаграмма
fig, ax = plt.subplots(figsize =(10, 7))
wedges, texts, autotexts = ax.pie(days,
                                  autopct = lambda pct: func(pct, days),
                                  explode = explode,
                                  labels = category,
                                  shadow = True,
                                  colors = colors,
                                  startangle = 90,
                                  wedgeprops = wp,
                                  textprops = dict(color ="red"))

# легенда
ax.legend(wedges, category,
          title ="Количество дней",
          loc ="center left",
          bbox_to_anchor =(1, 0, 0.5, 1))

plt.setp(autotexts, size = 8, weight ="bold")
ax.set_title("Круговая диаграмма")

# визуализация
plt.show();
```

Рисунок 5. Фрагмент программного кода, реализующего вывод модифицированной круговой диаграммы

Рисунок 6. Модифицированная круговая диаграмма

```python
import matplotlib.pyplot as plt

# данные
days = [91, 172, 102]
category = ['Ясный', 'Облачный', 'Пасмурный']
year=['Январь', 'Февраль', 'Март', 'Апрель', 'Май', 'Июнь', 'Июль', 'Август',
'Сентябрь', 'Октябрь', 'Ноябрь', 'Декабрь']
months_days=[8,  9, 10,  8, 11,  7,  8, 10,  8,  5,  3,  4,
             11, 10, 12, 17, 16, 20, 20, 17, 16, 14,  9, 10,
             12,  9,  9,  5,  4,  3,  3,  4,  6, 12, 18, 17]

months=year + year + year

# палитра
a, b, c=[plt.cm.Blues, plt.cm.Reds, plt.cm.Greens]

# внешнее кольцо
fig, ax = plt.subplots()
ax.axis('equal')
mypie, _ = ax.pie(days, radius=4, labels=category, colors=
                  [a(0.6), b(0.6), c(0.6)])
plt.setp(mypie, width=0.3, edgecolor='white')

# внутреннее кольцо
mypie2, _ = ax.pie(months_days, radius=4-0.3,
labels=months,
             labeldistance=0.8,
                    colors=[a(0.65), a(0.6),  a(0.55),
                            a(0.5),  a(0.45), a(0.4),
                            a(0.35), a(0.3),  a(0.25),
                            a(0.2),  a(0.15), a(0.1),
                            b(0.65), b(0.6),  b(0.55),
                            b(0.5),  b(0.45), b(0.4),
                            b(0.35), b(0.3),  b(0.25),
                            b(0.2),  b(0.15), b(0.1),
                            c(0.65), c(0.6),  c(0.55),
                            c(0.5),  c(0.45), c(0.4),
                            c(0.35), c(0.3),  c(0.25),
                            c(0.2),  c(0.15), c(0.1)],
                   rotatelabels = 270)
plt.setp( mypie2, width=2, edgecolor='white')
plt.margins(0,0)

handles, labels = ax.get_legend_handles_labels()

# визуализация
plt.show();
```

Рисунок 7. Фрагмент программного кода, реализующего вывод вложенной круговой диаграммы

На рис. 5 представлен фрагмент программного кода, реализующего вывод модифицированной круговой диаграммы. При этом выполнена настройка значений некоторых параметров функции `pie` библиотеки `matplotlib`, в частности, выбраны цветовая палитра секторов круга и цвет добавленного текста, определены значения параметров, отвечающие за смещение секторов круга относительно центра. Кроме того, к круговой диаграмме добавлена легенда. Также определены размеры рисунка.

На рис. 6 представлена модифицированная круговая диаграмма.

На рис. 7 представлен фрагмент программного кода, реализующего вывод вложенной круговой диаграммы с применением библиотеки `matplotlib`. При этом одна круговая диаграмма вложена в другую: внешняя диаграмма отражает укрупненно информацию по категориям «Ясный», «Облачный» и «Пасмурный»; внутренняя диаграмма для каждой из перечисленных категорий отображает подробную информацию по месяцам.

На рис. 8 представлена вложенная круговая диаграмма.

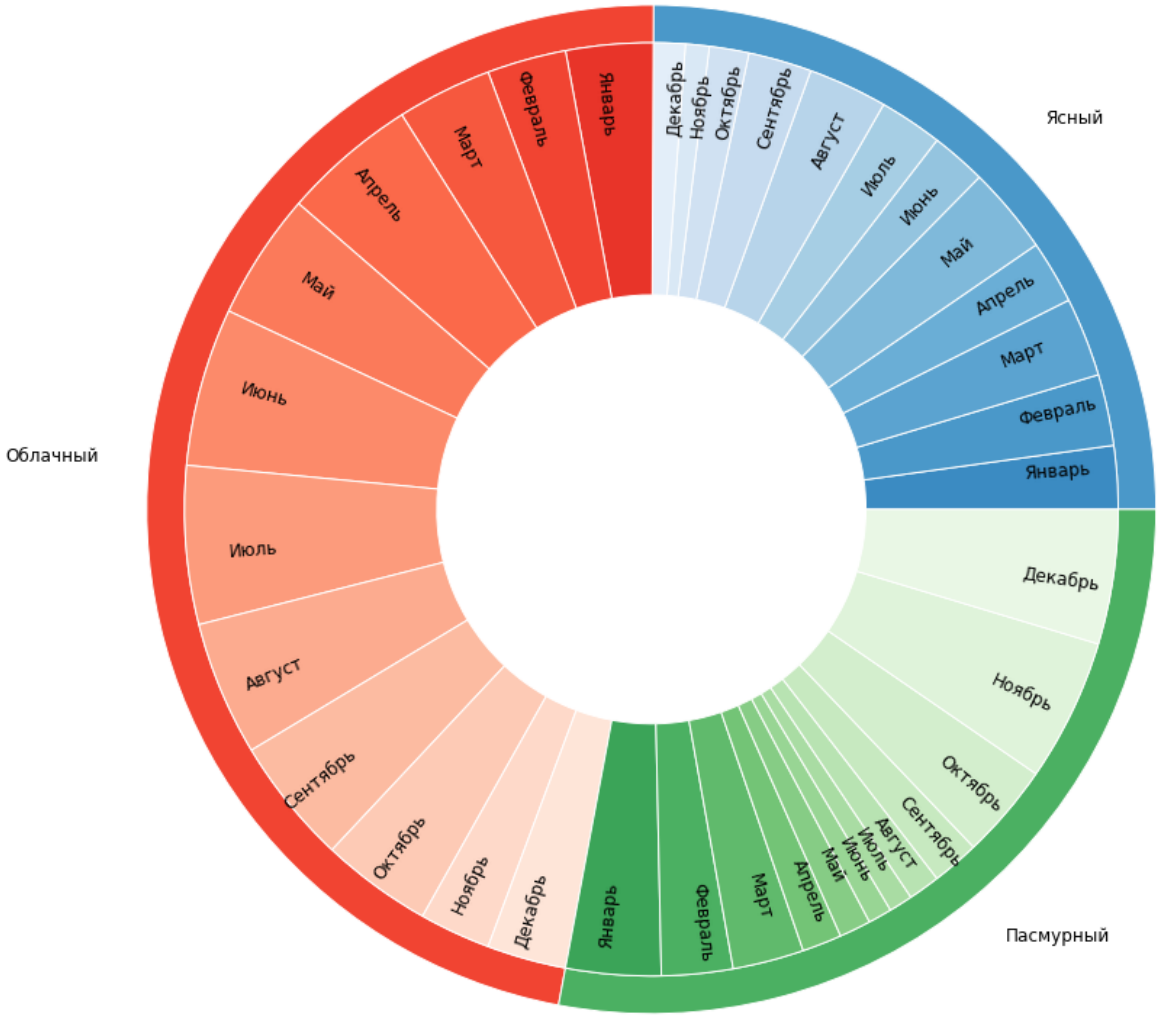

Рисунок 8. Вложенная круговая диаграмма

8.1.2. Примеры визуализации данных с применением круговых диаграмм на основе библиотеки seaborn

На рис. 9 представлен фрагмент программного кода, реализующего вывод круговой диаграммы с применением библиотеки `matplotlib`. При этом с помощью библиотеки `seaborn` определена цветовая палитра `'dark'`. Размеры рисунка определены по умолчанию.

На рис. 10 представлена круговая диаграмма с цветовой палитрой `'dark'`.

```
# импорт библиотек
import matplotlib.pyplot as plt
import seaborn

# данные
days = [91, 172, 102]
category = ['Ясный', 'Облачный', 'Пасмурный']

# определение цветовой палитры seaborn
palette_color = seaborn.color_palette('dark')

# круговая диаграмма
plt.pie(days, labels=category, colors=palette_color,
        autopct='%.0f%%')

# визуализация
plt.show()
```

Рисунок 9. Фрагмент программного кода, реализующего вывод круговой диаграммы с цветовой палитрой 'dark'

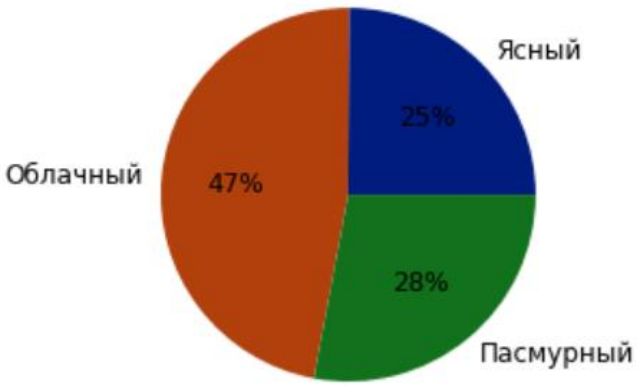

Рисунок 10. Круговая диаграмма с цветовой палитрой 'dark'

На рис. 11 представлен фрагмент программного кода, реализующего вывод круговой диаграммы с применением библиотеки `matplotlib`. При этом с по-

мощью библиотеки `seaborn` определена цветовая палитра `'bright'`. Кроме того, с помощью параметра `explode` определено смещение одного из секторов относительно центра круговой диаграммы. Размеры рисунка определены по умолчанию.

На рис. 12 представлена круговая диаграмма с цветовой палитрой `'bright'`.

```
# импорт библиотек
import matplotlib.pyplot as plt
import seaborn

# данные
days = [91, 172, 102]
category = ['Ясный', 'Облачный', 'Пасмурный']

# определение смещений секторов относительно центра
explode = [0.3, 0, 0]

# определение цветовой палитры seaborn
palette_color = seaborn.color_palette('bright')

# круговая диаграмма
plt.pie(days, labels=category, colors=palette_color,
        explode=explode, autopct='%.0f%%')

# визуализация
plt.show()
```

Рисунок 11. Фрагмент программного кода, реализующего вывод круговой диаграммы с цветовой палитрой 'bright' и смещением одного из секторов круга

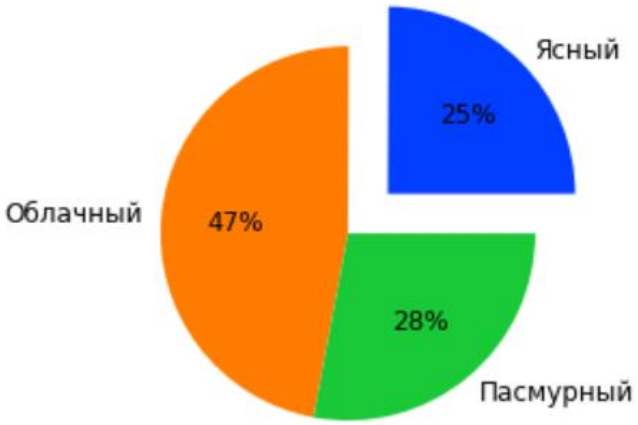

Рисунок 12. Круговая диаграмма с цветовой палитрой 'bright' и смещением одного из секторов круга

8.1.3. Примеры визуализации данных с применением круговых диаграмм на основе библиотеки pandas

На рис. 13 приведен пример формирования датафрейма на основе данных табл. 3.

На рис. 14 приведены программный код с выводом этого датафрейма и фрагмент этого датафрейма.

```
# импорт библиотек
import pandas as pd

# DataFrame для категорий в месяцах года и количества дней в каждой категории
dataframe = pd.DataFrame(
                  {'Месяц':
                          ['Январь', 'Февраль', 'Март',
                           'Апрель', 'Май', 'Июнь',
                           'Июль', 'Август', 'Сентябрь',
                           'Октябрь', 'Ноябрь', 'Декабрь',
                           'Январь', 'Февраль', 'Март',
                           'Апрель', 'Май', 'Июнь',
                           'Июль', 'Август', 'Сентябрь',
                           'Октябрь', 'Ноябрь', 'Декабрь',
                           'Январь', 'Февраль', 'Март',
                           'Апрель', 'Май', 'Июнь',
                           'Июль', 'Август', 'Сентябрь',
                           'Октябрь', 'Ноябрь', 'Декабрь'],
                   'День': ['Ясный', 'Ясный', 'Ясный',
                           'Ясный', 'Ясный', 'Ясный',
                           'Ясный', 'Ясный', 'Ясный',
                           'Ясный', 'Ясный', 'Ясный',
                           'Облачный', 'Облачный', 'Облачный',
                           'Облачный', 'Облачный', 'Облачный',
                           'Облачный', 'Облачный', 'Облачный',
                           'Облачный', 'Облачный', 'Облачный',
                           'Пасмурный', 'Пасмурный', 'Пасмурный',
                           'Пасмурный', 'Пасмурный', 'Пасмурный',
                           'Пасмурный', 'Пасмурный', 'Пасмурный',
                           'Пасмурный', 'Пасмурный', 'Пасмурный'],
                    'Количество дней':
                          [8,  9, 10,  8, 11,  7,  8, 10,  8,  5,  3,  4,
                           11, 10, 12, 17, 16, 20, 20, 17, 16, 14,  9, 10,
                           12,  9,  9,  5,  4,  3,  3,  4,  6, 12, 18, 17]})
```

Рисунок 13. Пример формирования датафрейма на основе данных табл. 3.

На рис. 15 представлен фрагмент программного кода, реализующего вывод круговой диаграммы с применением библиотеки `pandas`. Размеры рисунка определены по умолчанию.

На рис. 16 представлена круговая диаграмма с применением библиотеки `pandas`.

Отметим, что круговая диаграмма может быть построена и с применением библиотеки `plotly` [14], доступной по ссылке:

https://plotly.com/python/.

В частности, информация по работе с круговыми диаграммами доступна по ссылке:

https://plotly.com/python/pie-charts/.

```
1  dataframe.head(15)
```

	Месяц	День	Количество дней
0	Январь	Ясный	8
1	Февраль	Ясный	9
2	Март	Ясный	10
3	Апрель	Ясный	8
4	Май	Ясный	11
5	Июнь	Ясный	7
6	Июль	Ясный	8
7	Август	Ясный	10
8	Сентябрь	Ясный	8
9	Октябрь	Ясный	5
10	Ноябрь	Ясный	3
11	Декабрь	Ясный	4
12	Январь	Облачный	11
13	Февраль	Облачный	10
14	Март	Облачный	12

Рисунок 14. Программный код с выводом датафрейма на основе данных табл. 3 и фрагмент этого датафрейма.

```
1  # определение смещений секторов относительно центра
2  explode = (0.05, 0.05, 0.05)
3
4  # палитра
5  colors = ['pink', 'silver', 'steelblue']
6
7  # визуализация
8  dataframe.groupby(['День']).sum().plot(kind='pie',
9                    y='Количество дней', autopct='%1.0f%%',
10                   startangle=44,
11                   shadow=True,
12                   explode=explode);
```

Рисунок 15. Фрагмент программного кода, реализующего вывод круговой диаграммы с применением библиотеки pandas

Рисунок 16.Круговая диаграмма с применением библиотеки pandas

8.2. Примеры выполнения разведочного анализа с применением столбчатых диаграмм

Отобразим сводную информацию по категориям «Ясный», «Облачный» и «Пасмурный» для Москвы (табл. 3) с помощью столбчатой диаграммы, используя разные библиотеки.

Кроме того, рассмотрим возможность отображения информации по 2 городам – Москве и Сочи – на одной столбчатой диаграмме.

Таблица 4. Среднее количество ясных, облачных и пасмурных дней при учете нижней облачности в Сочи

Месяц		Январь	Февраль	Март	Апрель	Май	Июнь	Июль	Август	Сен-тябрь	Октябрь	Ноябрь	Декабрь	Год
Дни	Ясные	8	8	11	10	12	13	14	16	14	14	11	8	139
	Облачные	12	13	13	14	14	15	16	14	14	12	11	13	161
	Пасмурные	10	7	7	7	5	2	2	1	2	5	8	10	66

В табл. 4 представлена информация о среднем числе ясных, облачных и пасмурных дней в Сочи по месяцам года и в целом за год при учете нижней облачности (https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%BC%D0%B0%D1%82_%D0%A1%D0%BE%D1%87%D0%B8).

В среднем за год в Сочи бывает 139 ясный, 161 облачных и 66 пасмурных дней.

8.2.1. Примеры визуализации данных с применением столбчатых диаграмм на основе библиотеки matplotlib

На рис. 17 представлен фрагмент программного кода, реализующего вывод столбчатой диаграммы, позволяющей выполнить анализ распределения категорий. При построении диаграммы используется функция `bar` библиотеки `matplotlib`. При этом определены размеры рисунка.

На рис. 18 представлена столбчатая диаграмма, отражающая распределение категорий.

```
# импорт библиотек
import matplotlib.pyplot as plt

# данные
days = [91, 172, 102]
category = ['Ясный', 'Облачный', 'Пасмурный']

fig = plt.figure(figsize = (10, 5))

# столбчатая диаграмма
plt.bar(category, days, color ='maroon',
        width = 0.4)

plt.xlabel("Дни")
plt.ylabel("Количество дней")
plt.title("Столбчатая диаграмма")

# визуализация
plt.show()
```

Рисунок 17. Фрагмент программного кода, реализующего вывод столбчатой диаграммы с использованием функции bar

На рис. 19 представлен фрагмент программного кода, реализующего вывод столбчатой диаграммы с горизонтальным расположением столбцов. При этом используется функция `bar` библиотеки `matplotlib`. При выводе диаграммы около столбцов отображены соответствующие им количества дней в категориях. Кроме того, определены размеры рисунка.

На рис. 20 представлена столбчатая диаграмма с горизонтальным расположением столбцов.

Рисунок 18. Столбчатая диаграмма с использованием функции bar

```python
# импорт библиотек
from matplotlib import pyplot as plt

# данные
days = [91, 172, 102]
category = ['Ясный', 'Облачный', 'Пасмурный']

fig, ax = plt.subplots(figsize =(8, 4))

# горизонтальная столбчатая диаграмма
ax.barh(category, days)

# аннотация
for i in ax.patches:
    plt.text(i.get_width()+0.2, i.get_y()+0.5,
             str(round((i.get_width()), 2)),
             fontsize = 10, fontweight ='bold',
             color ='grey')

ax.set_title('Столбчатая диаграмма')
plt.ylabel("Дни")
plt.xlabel("Количество дней")

# визуализация
plt.show()
```

Рисунок 19. Фрагмент программного кода, реализующего вывод столбчатой диаграммы с горизонтальным расположением столбцов

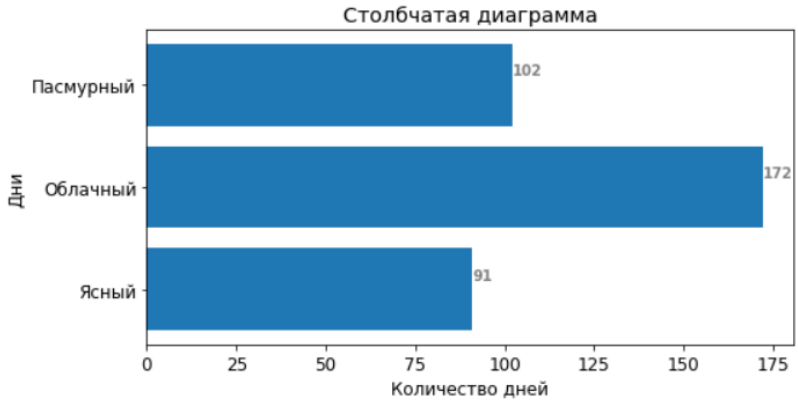

Рисунок 20. Столбчатая диаграмма с горизонтальным расположением столбцов

На рис. 21 представлен фрагмент программного кода, реализующего вывод столбчатой диаграммы с информацией по 2 городам (табл. 3 и табл. 4) в виде парных столбцов. Кроме того, определены размеры рисунка, заданы цветовая палитра, ширина и позиции столбцов.

На рис. 22 представлена столбчатая диаграмма с информацией по 2 городам (табл. 3 и табл. 4) в виде парных столбцов.

```
# импорт библиотек
import numpy as np
import matplotlib.pyplot as plt

# ширина столбца
barWidth = 0.25
fig = plt.subplots(figsize =(10, 5))

# данные
days_M = [91, 172, 102]  # Москва
days_S = [139, 161, 66] # Сочи
category = ['Ясный', 'Облачный', 'Пасмурный']

# позиция столбца на оси X
br1 = np.arange(len(category))
br2 = [x + barWidth for x in br1]

# столбчатая диаграмма
plt.bar(br1, days_M, color ='r', width = barWidth,
        edgecolor ='grey', label ='Москва')
plt.bar(br2, days_S, color ='b', width = barWidth,
        edgecolor ='grey', label ='Сочи')

plt.xlabel('Дни', fontsize = 12)
plt.ylabel('Количество дней', fontsize = 12)
plt.xticks([r + barWidth/2 for r in range(len(category))],
           category)
plt.legend()

# визуализация
plt.show()
```

Рисунок 21. Фрагмент программного кода, реализующего вывод столбчатой диаграммы с информацией по 2 городам (табл. 3 и табл. 4) в виде парных столбцов

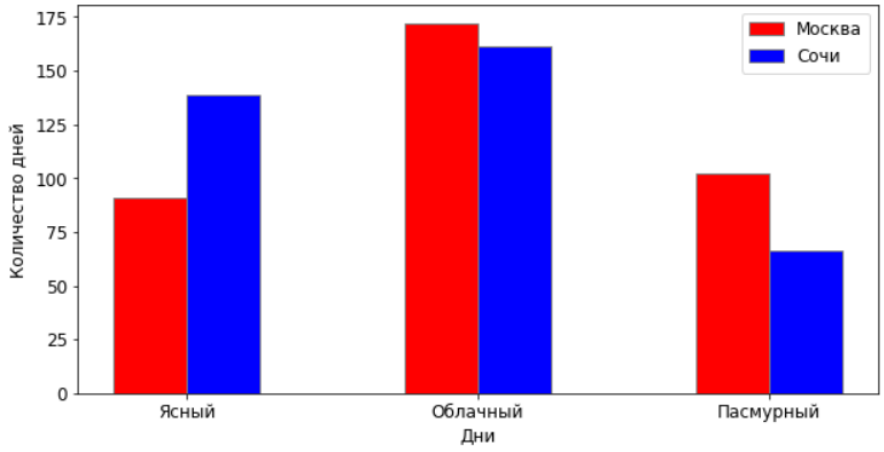

Рисунок 22. Столбчатая диаграмма с информацией по 2 городам (табл. 3 и табл. 4) в виде парных столбцов

На рис. 23 представлен фрагмент программного кода, реализующего вывод столбчатой диаграммы с информацией по 2 городам (табл. 3 и табл. 4) в виде составных столбцов. Кроме того, определены размеры рисунка, заданы ширина и позиции столбцов.

На рис. 24 представлена столбчатая диаграмма с информацией по 2 городам (табл. 3 и табл. 4) в виде составных столбцов.

```python
# импорт библиотек
import numpy as np
import matplotlib.pyplot as plt

# данные
days_M = [91, 172, 102]  # Москва
days_S = [139, 161, 66] # Сочи
category = ['Ясный', 'Облачный', 'Пасмурный']

ind = np.arange(3) # номера столбцов
width = 0.35 # ширина столбца

# столбчатая диаграмма
fig = plt.subplots(figsize =(10, 5))
p1 = plt.bar(ind, days_M, width)
p2 = plt.bar(ind, days_S, width, bottom = days_M)

plt.ylabel('Количество дней', fontsize = 12)
plt.xlabel('Дни', fontsize = 12)
plt.title('Столбчатая диаграмма')
plt.xticks(ind, tuple(category))
plt.legend((p1[0], p2[0]), ('Москва', 'Сочи'))

# визуализация
plt.show()
```

Рисунок 23. Фрагмент программного кода, реализующего вывод столбчатой диаграммы с информацией по 2 городам (табл. 3 и табл. 4) в виде составных столбцов

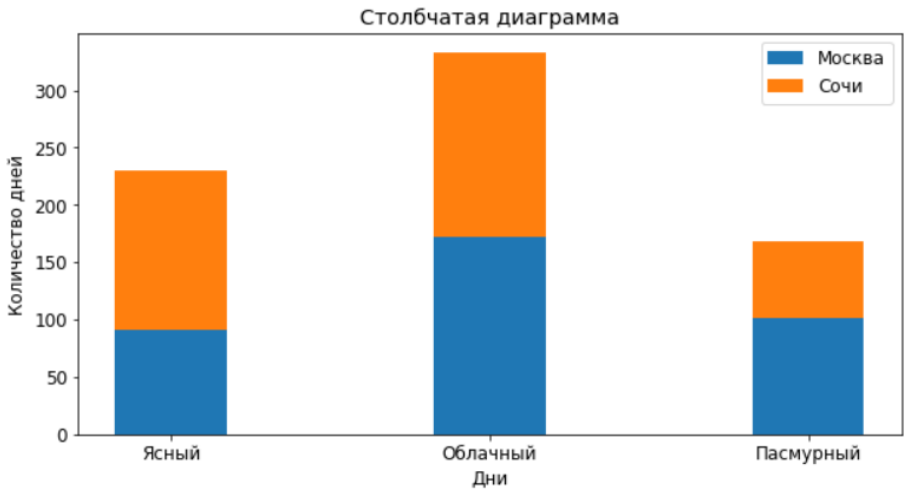

Рисунок 24. Столбчатая диаграмма с информацией по 2 городам (табл. 3 и табл. 4) в виде составных столбцов

8.2.2. Примеры визуализации данных с применением столбчатых диаграмм на основе библиотеки seaborn

На рис. 25 представлен фрагмент программного кода, реализующего вывод столбчатой диаграммы, позволяющей выполнить анализ распределения категорий. При построении диаграммы используется функция `barplot` библиотеки `seaborn`. При этом использован датафрейм, описанный в п. 8.1.3 (рис. 13).

На рис. 26 представлена столбчатая диаграмма, отражающая центральные тенденции и доверительные интервалы для каждой категории.

```
# импорт библиотек
import seaborn as sns

sns.barplot(data=dataframe, x="День", y="Количество дней")
```

Рисунок 25. Фрагмент программного кода, реализующего вывод столбчатой диаграммы с использованием функции barplot

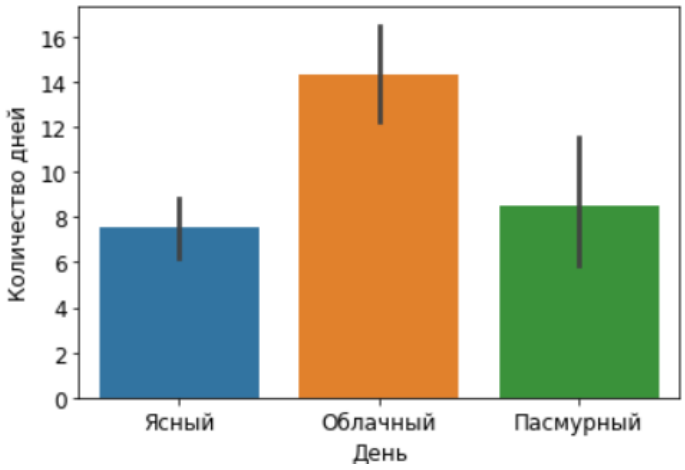

Рисунок 26. Столбчатая диаграмма с использованием функции barplot

8.2.3. Примеры визуализации данных с применением столбчатых диаграмм на основе библиотеки pandas

На рис. 27 представлен фрагмент программного кода, реализующего вывод столбчатой диаграммы с суммированием по категориям. При этом использован датафрейм, описанный в п. 8.1.3 (рис. 13). Для столбчатой диаграммы определена цветовая палитра и задан поворот подписей по горизонтальной оси на 45 градусов.

На рис. 28 представлена столбчатая диаграмма на основе датафрейма с суммированием по категориям.

```
# импорт библиотек
import pandas as pd

# визуализация
dataframe.groupby(['День']).sum().plot(kind='bar',
                  y='Количество дней',
                  color ='green', rot=45);
```

Рисунок 27. Фрагмент программного кода, реализующего вывод столбчатой диаграммы на основе датафрейма с суммированием по категориям

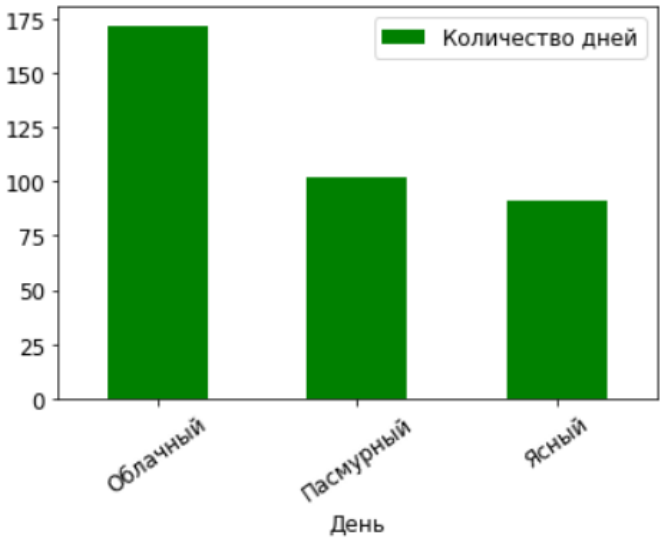

Рисунок 28. Столбчатая диаграмма на основе датафрейма с суммированием по категориям

На рис. 29 представлен фрагмент программного кода, реализующего вывод столбчатой диаграммы с суммированием по категориям. При этом использован датафрейм, описанный в п. 8.1.3 (рис. 13). Для столбчатой диаграммы задан поворот подписей по горизонтальной оси на 0 градусов. Сравнение фрагментов программных кодов на рис. 27 и 29 позволяет выявить различия в обращении к функции `bar`, однако результаты визуализации – идентичны (в смысле отображения столбчатой диаграммы).

На рис. 30 представлена столбчатая диаграмма на основе датафрейма с суммированием по категориям.

```
# импорт библиотек
import pandas as pd

# визуализация
dataframe.groupby(['День']).sum().plot.bar(
    y='Количество дней', rot=0);
```

Рисунок 29. Фрагмент программного кода, реализующего вывод столбчатой диаграммы на основе датафрейма с суммированием по категориям

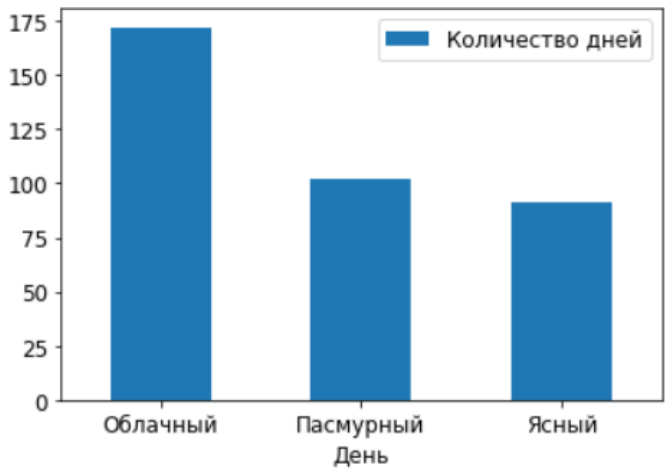

Рисунок 30. Столбчатая диаграмма
на основе датафрейма с суммированием по категориям

Отметим, что столбчатая диаграмма может быть построена и с применением библиотеки `plotly`.

В частности, информация по работе со столбчатыми диаграммами доступна по ссылке:

https://plotly.com/python/bar-charts/.

8.3. Примеры выполнения разведочного анализа данных с применением функции catplot из библиотеки seaborn

На рис. 31 представлен фрагмент программного кода, реализующего анализ данных с применением функции `catplot`. При этом использован датафрейм, описанный в п. 8.1.3 (рис. 13). По умолчанию для визуализации данных применена функция `stripplot`, реализующая построение категориальной диаграммы рассеяния.

На рис. 32 представлена категориальная диаграмма рассеяния.

```
# импорт библиотек
import seaborn as sns

# визуализация
sns.catplot(data=dataframe, x="Количество дней", y="День");
```

Рисунок 31. Фрагмент программного кода, реализующего анализ данных с применением функции catplot (с функцией stripplot по умолчанию)

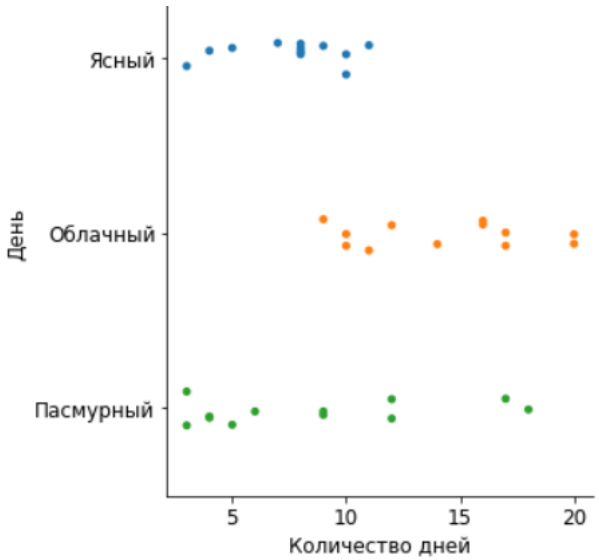

Рисунок 32. Категориальная диаграмма рассеяния

На рис. 33 представлен фрагмент программного кода, реализующего анализ данных с применением функции `catplot`. При этом использован датафрейм, описанный в п. 8.1.3 (рис. 13). Для визуализации данных применена функция `boxplot`, реализующая построение коробчатой диаграммы.

На рис. 34 представлена коробчатая диаграмма.

```
# импорт библиотек
import seaborn as sns

# визуализация
sns.catplot(data=dataframe, x="Количество дней", y="День", kind="box");
```

Рисунок 33. Фрагмент программного кода, реализующего анализ данных с применением функции catplot (с функцией boxplot для построения коробчатой диаграммы)

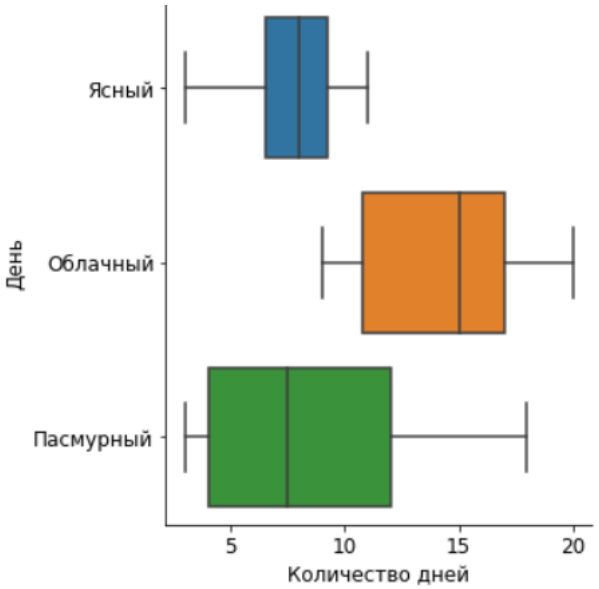

Рисунок 34. Коробчатая диаграмма

На рис. 35 представлен фрагмент программного кода, реализующего анализ данных с применением функции `catplot`. При этом использован датафрейм, описанный в п. 8.1.3 (рис. 13). Для визуализации данных применены функция `violinplot`, реализующая построение скрипичной диаграммы, и функция `swarmplot`, реализующая построение категориальной диаграммы рассеяния (без наложения точек данных друг на друга).

На рис. 36 представлена гибридная диаграмма на основе функций `violinplot` и `swarmplot`.

```
# импорт библиотек
import seaborn as sns

# визуализация
sns.catplot(data=dataframe, x="Количество дней", y="День", kind="violin", color=".9", inner=None)
sns.swarmplot(data=dataframe, x="Количество дней", y="День", size=3);
```

Рисунок 35. Фрагмент программного кода, реализующего анализ данных с применением функции catplot (с функцией violinplot для построения скрипичной диаграммы и функции swarmplot для построения категориальной диаграммы рассеяния (без наложения точек данных друг на друга))

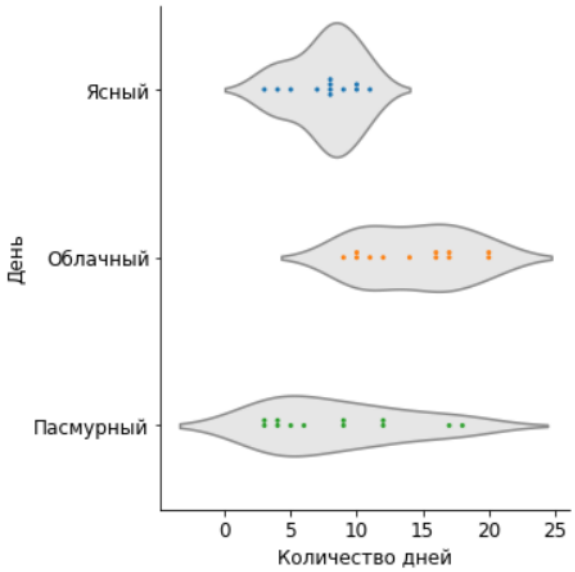

Рисунок 36. Гибридная диаграмма

8.4. Примеры выполнения разведочного анализа с применением сводной таблицы и таблицы сопряженности

На рис. 37 представлен фрагмент программного кода, реализующего построение сводной таблицы с применением функции `pivot_table`. При этом

использован датафрейм, описанный в п. 8.1.3 (рис. 13). Для сохранения упорядоченности столбцов датафрейма по месяцам года значение параметра `sort` установлено равным значению `False` (если значение этого параметра будет равно `True`, то столбцы будут упорядочены в алфавитном порядке).

На рис. 38 представлена сводная таблица на основе функции `pivot_table`.

```
pivoted = dataframe.pivot_table(index="День", columns="Месяц",
                                values="Количество дней", sort = False)
pivoted
```

Рисунок 37. Фрагмент программного кода, реализующего построение сводной таблицы с применением функции pivot_table

Месяц	Январь	Февраль	Март	Апрель	Май	Июнь	Июль	Август	Сентябрь	Октябрь	Ноябрь	Декабрь
День												
Ясный	8	9	10	8	11	7	8	10	8	5	3	4
Облачный	11	10	12	17	16	20	20	17	16	14	9	10
Пасмурный	12	9	9	5	4	3	3	4	6	12	18	17

Рисунок 38. Сводная таблица на основе функции pivot_table

Аналогичные результаты можно получить, используя функцию `pivot`.

Однако функция `pivot_table` предоставляет более широкий спектр возможностей, некоторые из которых будут рассмотрены ниже.

```
df = pd.DataFrame({'Группа': ['взрослый', 'взрослый', 'взрослый', 'взрослый', 'взрослый',
                              'ребенок', 'ребенок', 'ребенок', 'ребенок'],
                   'Пол': ['м', 'м', 'м', 'ж', 'ж',
                           'м', 'м', 'ж', 'ж'],
                   'Статус': ['любитель', 'профессионал', 'профессионал', 'любитель',
                              'любитель', 'профессионал', 'любитель', 'любитель',
                              'профессионал'],
                   'Дистанция, м': [1000, 5000, 3000, 10000, 500, 30, 60, 30, 100],
                   'Высота, м': [180, 190, 170, 165, 185, 115, 125, 110, 130]})
df
```

Рисунок 39. Пример датафрейма, описывающего спортсменов

	Группа	Пол	Статус	Дистанция, м	Высота, м
0	взрослый	м	любитель	1000	180
1	взрослый	м	профессионал	5000	190
2	взрослый	м	профессионал	3000	170
3	взрослый	ж	любитель	10000	165
4	взрослый	ж	любитель	500	185
5	ребенок	м	профессионал	30	115
6	ребенок	м	любитель	60	125
7	ребенок	ж	любитель	30	110
8	ребенок	ж	профессионал	100	130

Рисунок 40. Визуализация датафрейма, описывающего спортсменов

На рис. 39 приведен пример датафрейма, описывающего спортсменов, для которых определены категории по группе, полу, статусу.

На рис. 40 приведена визуализация этого датафрейма в виде таблицы.

На рис. 41 приведен пример построения сводной таблицы, значения в которой агрегируются посредством взятия суммы по признаку «`Дистанция, м`».

На рис. 42 приведена визуализация этой сводной таблицы. Можно увидеть, что одно из значений в сводной таблице равно `NaN`.

На рис. 43 приведен пример построения сводной таблицы, значения в которой агрегируются посредством взятия суммы по признаку «`Дистанция, м`». При этом отсутствующее значение `NaN` заменено на 1000 (`fill_value=1000`).

На рис. 44 приведена визуализация этой сводной таблицы.

```
table = pd.pivot_table(df, values='Дистанция, м', index=['Группа', 'Пол'],
                       columns=['Статус'], aggfunc=np.sum)
table
```

Рисунок 41. Пример построения сводной таблицы, значения в которой агрегируются посредством взятия суммы по признаку «Дистанция, м»

	Статус	любитель	профессионал
Группа	Пол		
взрослый	ж	10500.0	NaN
	м	1000.0	8000.0
ребенок	ж	30.0	100.0
	м	60.0	30.0

Рисунок 42. Визуализация сводной таблицы, значения в которой агрегируются посредством взятия суммы по признаку «Дистанция, м»

```
table = pd.pivot_table(df, values='Дистанция, м', index=['Группа', 'Пол'],
                       columns=['Статус'], aggfunc=np.sum, fill_value=1000)
table
```

Рисунок 43. Пример построения сводной таблицы, значения в которой агрегируются посредством взятия суммы по признаку «Дистанция, м» (отсутствующее значение NaN заменено на 1000)

	Статус	любитель	профессионал
Группа	Пол		
взрослый	ж	10500	1000
	м	1000	8000
ребенок	ж	30	100
	м	60	30

Рисунок 44. Визуализация сводной таблицы, значения в которой агрегируются посредством взятия суммы по признаку «Дистанция, м» (отсутствующее значение NaN заменено на 1000)

На рис. 45 приведен пример построения сводной таблицы, значения в которой агрегируются посредством получения среднего значения по признакам «Дистанция, м» и «Высота, м».

На рис. 46 приведена визуализация этой сводной таблицы.

```
table = pd.pivot_table(df, values=['Дистанция, м', 'Высота, м'],
                       index=['Группа', 'Статус'],
                       aggfunc={'Дистанция, м': np.mean,
                                'Высота, м': np.mean})
table
```

Рисунок 45. Пример построения сводной таблицы, значения в которой агрегируются посредством получения среднего значения по признакам «Дистанция, м» и «Высота, м»

		Высота, м	Дистанция, м
Группа	**Статус**		
взрослый	**любитель**	176.666667	3833.333333
	профессионал	180.000000	4000.000000
ребенок	**любитель**	117.500000	45.000000
	профессионал	122.500000	65.000000

Рисунок 46. Визуализация сводной таблицы, значения в которой агрегируются посредством получения среднего значения по признакам «Дистанция, м» и «Высота, м»

На рис. 47 приведен пример сортировки сводной таблицы `table` по убыванию значений признака «Высота, м».

На рис. 48 приведена визуализация отсортированной сводной таблицы.

```
sorted_table = table.sort_values(by=['Высота, м'], ascending=False)
sorted_table
```

Рисунок 47. Пример сортировки сводной таблицы table по убыванию значений признака «Высота, м»

		Высота, м	Дистанция, м
Группа	**Статус**		
взрослый	**профессионал**	180.000000	4000.000000
	любитель	176.666667	3833.333333
ребенок	**профессионал**	122.500000	65.000000
	любитель	117.500000	45.000000

Рисунок 48. Визуализация отсортированной сводной таблицы

На рис. 49 приведен пример построения сводной таблицы, значения в которой для каждого заданного признака («Дистанция, м» и «Высота, м») агрегируются несколькими способами (np.mean и max, np.mean, min соответственно).

На рис. 50 приведена визуализация этой сводной таблицы.

На рис. 51 приведен пример описания состояния погоды, для которого определены категории по облачности (cloudiness), ветрености (wind), температуре воздуха (temperature).

На рис. 52 приведен пример построения таблицы сопряженности с выбором строк и столбцов.

На рис. 53 приведена визуализация этой таблицы сопряженности.

```
table = pd.pivot_table(df, values=['Дистанция, м', 'Высота, м'],
                       index=['Группа', 'Статус'],
                       aggfunc={'Дистанция, м': np.mean,
                                'Высота, м': [max, np.mean, min]})
table
```

Рисунок 49. Пример построения сводной таблицы, значения в которой агрегируются по-разному по признакам «Дистанция, м» и «Высота, м»

		Высота, м			Дистанция, м
		max	mean	min	mean
Группа	Статус				
взрослый	любитель	185.0	176.666667	165.0	3833.333333
	профессионал	190.0	180.000000	170.0	4000.000000
ребенок	любитель	125.0	117.500000	110.0	45.000000
	профессионал	130.0	122.500000	115.0	65.000000

Рисунок 50. Визуализация сводной таблицы, значения в которой агрегируются по-разному по признакам «Дистанция, м» и «Высота, м»»

```
cloudiness = np.array(['ясный', 'ясный', 'ясный',
                       'ясный', 'пасмурный', 'пасмурный',
                       'пасмурный', 'пасмурный', 'ясный',
                       'ясный', 'ясный'],
                       dtype=object)

wind = np.array(['штиль', 'штиль', 'штиль',
                 'ветер', 'штиль', 'штиль',
                 'штиль', 'ветер', 'ветер',
                 'ветер', 'штиль'],
                 dtype=object)

temperature = np.array(['теплый', 'теплый', 'морозный',
                        'теплый', 'теплый', 'морозный',
                        'морозный', 'теплый', 'морозный',
                        'морозный', 'морозный'],
                        dtype=object)
```

Рисунок 51. Пример описания состояния погоды, для которого определены категории по облачности (cloudiness), ветрености (wind), температуре воздуха (temperature)

```
pd.crosstab(cloudiness, [wind, temperature],
            rownames=['temperature'],
            colnames=['wind', 'temperature'])
```

Рисунок 52. Пример построения таблицы сопряженности

wind	ветер		штиль	
temperature	морозный	теплый	морозный	теплый
temperature				
пасмурный	0	1	2	1
ясный	2	1	2	2

Рисунок 53. Пример таблицы сопряженности

8.5. Примеры выполнения разведочного анализа с вычислением значений метрик

На рис. 54 приведены примеры работы функций `multimode` и `mode` из библиотеки `statistics`, позволяющих вычислить соответственно все моды и одну (первую) моду в списке категориальных значений.

```
import statistics as stat
stat.multimode(['Ясный', 'Ясный', 'Ясный',
                'Ясный', 'Ясный', 'Ясный',
                'Ясный', 'Ясный', 'Ясный',
                'Ясный', 'Ясный', 'Ясный',
                'Облачный', 'Облачный', 'Облачный',
                'Облачный', 'Облачный', 'Облачный',
                'Облачный', 'Облачный', 'Облачный',
                'Облачный', 'Облачный', 'Облачный',
                'Пасмурный', 'Пасмурный', 'Пасмурный',
                'Пасмурный', 'Пасмурный', 'Пасмурный',
                'Пасмурный', 'Пасмурный', 'Пасмурный',
                'Пасмурный', 'Пасмурный', 'Пасмурный'])
```

```
['Ясный', 'Облачный', 'Пасмурный']
```

```
stat.mode(['Ясный', 'Ясный', 'Ясный',
           'Ясный', 'Ясный', 'Ясный',
           'Ясный', 'Ясный', 'Ясный',
           'Ясный', 'Ясный', 'Ясный',
           'Облачный', 'Облачный', 'Облачный',
           'Облачный', 'Облачный', 'Облачный',
           'Облачный', 'Облачный', 'Облачный',
           'Облачный', 'Облачный', 'Облачный',
           'Пасмурный', 'Пасмурный', 'Пасмурный',
           'Пасмурный', 'Пасмурный', 'Пасмурный',
           'Пасмурный', 'Пасмурный', 'Пасмурный',
           'Пасмурный', 'Пасмурный', 'Пасмурный'])
```

```
'Ясный'
```

Рисунок 54. Примеры работы функций multimode и mode из библиотеки statistics, позволяющих вычислить соответственно все моды и одну (первую) моду в списке категориальных значений

На рис. 55 приведен пример вычисления центральной тенденции на примере датафрейма, информация по которому приведена на рис. 39 и рис. 40, по признаку «Дистанция, м» с применением функции mean для категорий «взрослый» и «ребенок» признака «Группа».

На рис. 56 приведены результаты вычисления центральной тенденции.

На рис. 57 приведен пример вычисления максимального значения по признаку «Дистанция, м» для датафрейма table1 (рис. 52).

На рис. 58 приведен пример датафрейма, в котором определены весовые коэффициенты для различных значений скорости.

На рис. 59 приведена визуализация этого датафрейма.

```
table1 = df.groupby('Группа')['Дистанция, м'].mean().reset_index()
table1
```

Рис. 55. Пример вычисления центральной тенденции на примере датафрейма, информация по которому приведена на рис. 39 и рис. 40

	Группа	Дистанция, м
0	взрослый	3900.0
1	ребенок	55.0

Рисунок 56. Результаты вычисления центральной тенденции

```
table1['Дистанция, м'].max()
```

```
3900.0
```

Рисунок 57. Пример вычисления максимального значения по признаку «Дистанция, м» для датафрейма table1

```
# импорт библиотек
import pandas as pd

df = pd.DataFrame.from_dict({
    'W': [10, 7, 4, 9, 2], # весовые коэффициенты
    'Скорость': [100, 85, 95, 85, 60]
})

df
```

Рисунок 58. Пример датафрейма, в котором определены весовые коэффициенты для различных значений скорости

	W	Скорость
0	10	100
1	7	85
2	4	95
3	9	85
4	2	60

Рисунок 59. Визуализация датафрейма, в котором определены весовые коэффициенты для различных значений скорости

На рис. 60 приведен пример вычисления взвешенного среднего для датафрейма `df` (рис. 59) с использованием пользовательской функции.

На рис. 61 приведен пример вычисления взвешенного среднего для датафрейма `df` (рис. 59), расширенного с помощью признака «`Год`» с использованием пользовательской функции.

На рис. 62 приведен пример вычисления взвешенного среднего для датафрейма `df` (рис. 59) с использованием библиотеки `numpy`.

```python
# вычисление взвешенного среднего с помощью пользовательской функции
def weighted_average(df, values, weights):
    return sum(df[weights] * df[values]) / df[weights].sum()

print(weighted_average(df, 'Скорость', 'W'))
```

```
89.375
```

Рисунок 60. Пример вычисления взвешенного среднего для датафрейма df, с использованием пользовательской функции

```python
# импорт библиотек
import pandas as pd

df = pd.DataFrame.from_dict({
    'Год': ['2023', '2023', '2022', '2022', '2022'],
    'W': [10, 7, 4, 9, 2], # весовые коэффициенты
    'Скорость': [100, 85, 95, 85, 60]
})

def weighted_average(df, values, weights):
    return sum(df[weights] * df[values]) / df[weights].sum()

df.groupby('Год').apply(weighted_average, 'Скорость', 'W').reset_index()
```

	Год	0
0	2022	84.333333
1	2023	93.823529

Рисунок 61. Пример вычисления взвешенного среднего для датафрейма df, расширенного с помощью признака «Год» с использованием пользовательской функции

```python
# импорт библиотек
import pandas as pd
import numpy as np

df = pd.DataFrame.from_dict({
    'W': [10, 7, 4, 9, 2], # весовые коэффициенты
    'Скорость': [100, 85, 95, 85, 60]
})

# вычисление взвешенного среднего в датафреме с помощью Numpy
weighted_average = np.average(a=df['Скорость'], weights=df['W'])
weighted_average
```

```
89.375
```

Рисунок 62. Пример вычисления взвешенного среднего для датафрейма df с использованием библиотеки numpy

На рис. 63 приведен пример вычисления взвешенного среднего на основе двух списков с использованием пользовательской функции.

```
# вычисление взвешенного среднего двух списков
W = [10, 7, 4, 9, 2] # весовые коэффициенты
Speed = [100, 85, 95, 85, 60]

def weighted_average(values, weights):
    weighted_sum = []
    for value, weight in zip(values, weights):
        weighted_sum.append(value * weight)

    return sum(weighted_sum) / sum(weights)

weighted_average(Speed, W)
```

```
89.375
```

Рисунок 63. Пример вычисления взвешенного среднего на основе двух списков с использованием пользовательской функции

9. Визуализация категориальных данных с применением библиотеки bokeh

Обработка категориальных данных – сложная задача, т.к. их нельзя обрабатывать так же, как числовые данные.

Одним из способов визуализации и анализа категориальных данных является использование библиотеки `bokeh` [15] – мощной библиотеки для создания интерактивных визуализаций.

Основные концепции категориальных данных библиотеки `bokeh` (https://pypi.org/project/bokeh/) приведены ниже.

1. Категориальные данные – тип данных, которые можно разделить на отдельные категории или группы. Этот тип может быть номинальным (без внутреннего порядка) или порядковым (имеет внутренний порядок).

2. Категориальная ось используется для представления категориальных данных на графике.

3. Категориальное сопоставление цветов – способ назначения разных цветов разным категориям на графике. Это может быть полезно для визуального различения разных категорий.

На рис. 64 представлен фрагмент программного кода, реализующего анализ данных с построением столбчатой диаграммы и сохранением её в файле с применением библиотеки `bokeh`. При этом используется медицинский набор данных `load_breast_cancer` из `sklearn.datasets`.

На рис. 65 представлена столбчатая диаграмма.

```python
# импорт библиотек
from bokeh.io import output_file, show
from bokeh.plotting import figure
from sklearn.datasets import load_breast_cancer

# загрузка набора данных
data = load_breast_cancer(as_frame=True)


# файл для сохранения результатов
output_file("Breast Cancer.html")

# цветовая палитра
color = ["orange","green"]


p = figure(x_range=data.target_names, height=350,
           title="breast cancer",toolbar_location=None, tools="")

p.vbar(x=data.target_names, top=data['target'].value_counts(),
       color=color, width=0.9)

show(p)
```

Рисунок 64. Фрагмент программного кода, реализующего анализ данных с построением столбчатой диаграммы и сохранением её в файле с применением библиотеки bokeh

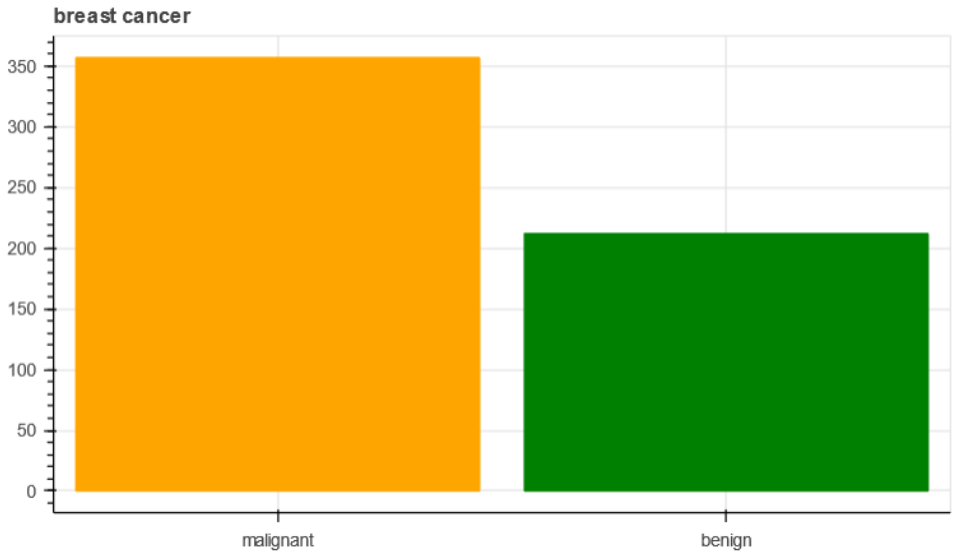

Рисунок 65. Столбчатая диаграмма

На рис. 66 представлен фрагмент программного кода, реализующего анализ данных с построением столбчатой диаграммы и сохранением её в файле с применением библиотеки `bokeh`. При этом используется набор данных о цветках ириса `load_iris` из `sklearn.datasets`.

На рис. 67 представлена вложенная столбчатая диаграмма. На рисунке можно увидеть панель инструментов, расположенную справа.

```python
# импорт библиотек
from bokeh.io import output_file, show
from bokeh.models import ColumnDataSource, FactorRange
from bokeh.plotting import figure
from bokeh.transform import factor_cmap
from sklearn.datasets import load_iris

# загрузка набора данных
data = load_iris(as_frame=True)
data= data.frame

# замена 0, 1, 2 на названия категорий для цветов ириса
data.target.replace({0: load_iris().target_names[0],
                     1:load_iris().target_names[1],
                     2:load_iris().target_names[2]},
                     inplace = True)

#  группирование цветов ириса по среднему значению
df = data.groupby('target').agg('mean')

# создание списка с названием цветка ириса
# и соответствующей ему длиной и шириной чашелистика и лепестка
x = [ (cls, col) for cls in data.target.unique() for col in data.columns[:-1] ]

# агрегирование по признакам x
mean = sum(zip(df['sepal length (cm)'],
               df['sepal width (cm)'],
               df['petal length (cm)'],
               df['petal width (cm)']), ())

# создание словаря
source= ColumnDataSource(data=dict(x=x, Average = mean))

# файл для сохранения результатов
output_file("Bokeh Nested Bar chart.html")

# создание рисунка
p = figure(x_range=FactorRange(*x),
           height=350,
           title="Iris Flower Average length",
           )

# цветовая палитра
color = ['orange','#FF0000','green','#00FF00',]

# вертикальная столбчатая диаграмма
p.vbar(x='x',
       top='Average',
       width=0.9,
       source=source,
       line_color="white",
       fill_color=factor_cmap('x', palette=color,
                              factors=data.columns[:-1], start=1, end=2)
       )

# Ориентация текста
p.xaxis.major_label_orientation = 1
p.xgrid.grid_line_color = None
show(p)
```

Рисунок 66. Фрагмент программного кода, реализующего анализ данных с построением вложенной столбчатой диаграммы и сохранением её в файле с применением библиотеки bokeh

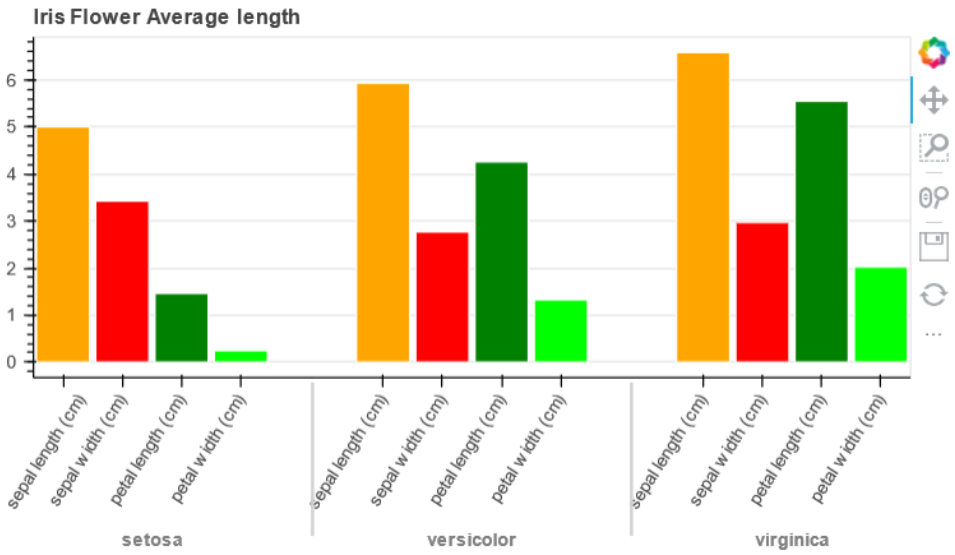

Рисунок 67. Вложенная столбчатая диаграмма

10. Пример обработки набора данных с категориальными признаками

Известно, что категориальные данные могут принимать значения только из некоторого конечного перечня. Однако по разным причинам в наборе данных могут присутствовать фиктивные (неверные) значения категориальных признаков (например, это может произойти из-за ошибки ввода данных).

Рассмотрим аспекты работы с набором данных `demographics` (https://drive.google.com/file/d/1MS0pMSn-HIOB3ywltnqAEPVlc9QUxZqM/view), содержащим категориальные признаки, и продемонстрируем примеры, связанные с выявлением фиктивных значений категориальных признаков, а также – с перекодировкой значений категориальных признаков.

На рис. 68 приведен фрагмент программного кода с загрузкой набора данных `demographics` в датафрейм.

На рис. 69 приведен фрагмент этого датафрейма.

```
# импорт библиотек
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.preprocessing import LabelEncoder

# загрузка набора данных
main_data = pd.read_csv('demographics.csv')
main_data.head()
```

Рис. 68. Фрагмент программного кода с загрузкой набора данных demographics в датафрейм

	first_name	last_name	blood_type	marriage_status	income	device
0	Abdul	Colon	A+	married	145000	AndroidOS
1	Abdul	Pierce	B+	married	85000	MacOS
2	Desirae	Pierce	B+	MARRIED	130000	iOS
3	Shannon	Gibson	A+	married	175000	MacOS
4	Desirae	Little	B+	unmarried	130000	MacOS

Рисунок 69. Фрагмент датафрейма

На рисунке 70 приведены фрагмент программного кода, позволяющего с применением метода `info` получить информацию по набору данных, и непосредственно сама информация.

```
1  main_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   first_name       10000 non-null  object
 1   last_name        10000 non-null  object
 2   blood_type       10000 non-null  object
 3   marriage_status  10000 non-null  object
 4   income           10000 non-null  int64 
 5   device           10000 non-null  object
dtypes: int64(1), object(5)
memory usage: 468.9+ KB
```

Рисунок 70. Фрагмент программного кода, позволяющего получить информацию по набору данных, и сама информация

Из рис. 70 видно, что только один признак, как и ожидалось, является количественным. При этом пропуски в наборе данных отсутствуют.

Для получения сводной информации в виде основных статистических характеристик по каждому признаку в наборе данных можно использовать метод `describe`.

Однако при использовании метода `describe` в случае, когда значения его параметров заданы по умолчанию, метод выходит сводную информацию только по количественным признакам. В рассматриваемом примере количественный признак только один – «доход» (`income`).

На рис. 71 приведен пример с выводом сводной информации по признаку «доход» (`income`).

```
1 main_data.describe()
```

	income
count	10000.000000
mean	115025.500000
std	47039.006331
min	40000.000000
25%	70000.000000
50%	115000.000000
75%	160000.000000
max	190000.000000

Рисунок 71. Пример с выводом сводной информации по количественному признаку «доход» (income)

Для вывода сводной информации по всем признакам можно задать значение параметра `include` метода `describe` как `include='all'` (рис. 72), однако в этом случае будет выполнена попытка расчета всех возможных показателей, некоторые из которых, вообще говоря, могут быть не применимы к количественным или категориальным данным (в таком случае будут выведены значения `NaN`).

```
1 main_data.describe(include='all')
```

	first_name	last_name	blood_type	marriage_status	income	device
count	10000	10000	10000	10000	10000.000000	10000
unique	8	7	10	8	NaN	5
top	Desirae	Gibson	O-	married	NaN	MacOS
freq	1286	1488	1346	1333	NaN	2047
mean	NaN	NaN	NaN	NaN	115025.500000	NaN
std	NaN	NaN	NaN	NaN	47039.006331	NaN
min	NaN	NaN	NaN	NaN	40000.000000	NaN
25%	NaN	NaN	NaN	NaN	70000.000000	NaN
50%	NaN	NaN	NaN	NaN	115000.000000	NaN
75%	NaN	NaN	NaN	NaN	160000.000000	NaN
max	NaN	NaN	NaN	NaN	190000.000000	NaN

Рисунок 72. Пример с выводом сводной информации по всем признакам

Для вывода сводной информации по конкретному типу признака следует использовать метод `include` с указанием этого типа.

В рассматриваемом примере целесообразно установить значение параметра `include` как `include='object'` (рис. 73), т.к. именно тип `'object'` определен для большинства признаков (рис. 70).

```
1 main_data.describe(include=['object'])
```

	first_name	last_name	blood_type	marriage_status	device
count	10000	10000	10000	10000	10000
unique	8	7	10	8	5
top	Desirae	Gibson	O-	married	MacOS
freq	1286	1488	1346	1333	2047

Рисунок 73. Пример с выводом сводной информации по признакам типа 'object'

Для категориальных признаков (т.е. признаков с типом `object`) и булевых признаков (т.е. признаков с типом `bool`) можно использовать метод `value_counts`, чтобы оценить распределение значений по категориям.

На рис. 74 приведен пример с вычислением распределения значений по категориям для признака `device` в виде абсолютных частот.

```
1 main_data['device'].value_counts()
```

```
MacOS      2047
Windows    2026
Linux      1981
iOS        1979
AndroidOS  1967
Name: device, dtype: int64
```

Рисунок 74. Пример с вычислением распределения значений по категориям для признака device в виде абсолютных частот

На рис. 75 приведен пример с вычислением распределения значений по категориям для признака `device` в виде относительных частот при установке значения параметра `normalize` как `normalize=True`.

```
1 main_data['device'].value_counts(normalize=True)
```

```
MacOS      0.2047
Windows    0.2026
Linux      0.1981
iOS        0.1979
AndroidOS  0.1967
Name: device, dtype: float64
```

Рисунок 75. Пример с вычислением распределения значений по категориям для признака device в виде относительных частот

Ниже рассмотрен пример, в котором продемонстрировано выявление фиктивных значений для категориального признака `blood_type`, описывающего группу крови.

На рис. 76 приведен фрагмент программного кода с созданием нового датафрейма с возможными значениями группы крови.

На рис. 77 приведен этот датафрейм.

```
# создание нового датафрейма с возможными значениями
# для группы крови (blood type)
blood_type_categories = pd.DataFrame({
    'blood_type': ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-']
})
blood_type_categories
```

Рисунок 76. Фрагмент программного кода с созданием нового датафрейма с возможными значениями для группы крови

	blood_type
0	A+
1	A-
2	B+
3	B-
4	AB+
5	AB-
6	O+
7	O-

Рисунок 77. Датафрейм с возможными значениями группы крови

На рис. 78 приведен фрагмент программного кода для поиска фиктивных значений группы крови на основе датафрейма с возможными значениями группы крови.

На рис. 79 приведены результаты поиска фиктивных значений группы крови.

```
# поиск фиктивных значений для группы крови
unique_blood_types_main = set(main_data['blood_type'])
bogus_blood_types = unique_blood_types_main.difference(
    blood_type_categories['blood_type']
)
bogus_blood_types
```

```
{'C+', 'D-'}
```

Рисунок 78. Фрагмент программного кода для поиска фиктивных значений группы крови

```
{'C+', 'D-'}
```

Рисунок 79. Результаты поиска фиктивных значений группы крови

На рис. 80 приведен фрагмент программного кода для извлечения и удаления записей с фиктивными группами крови.

На рис. 81 приведены значения группы крови в исходном датафрейме, полученные после удаления фиктивных значений группы крови.

На рис. 82 приведен фрагмент программного кода, реализующего построение столбчатой диаграммы на основе категориального признака `blood_type`.

На рис. 83 приведена столбчатая диаграмма на основе категориального признака `blood_type`.

```
# извлечение записей с фиктивными группами крови
bogus_records_index = main_data['blood_type'].isin(bogus_blood_types)

#  удаление записей с фиктивными группами крови
without_bogus_records = main_data[~bogus_records_index]
without_bogus_records['blood_type'].unique()
```

Рисунок 80. Фрагмент программного кода для извлечения и удаления записей с фиктивными группами крови

```
array(['A+', 'B+', 'A-', 'AB-', 'AB+', 'B-', 'O-', 'O+'], dtype=object)
```

Рисунок 81. Значения группы крови в исходном датафрейме, полученные после удаления фиктивных значений группы крови

```
sns.countplot(x='blood_type',
              data=without_bogus_records);
```

Рисунок 82. Фрагмент программного кода, реализующего построение столбчатой диаграммы на основе категориального признака blood_type

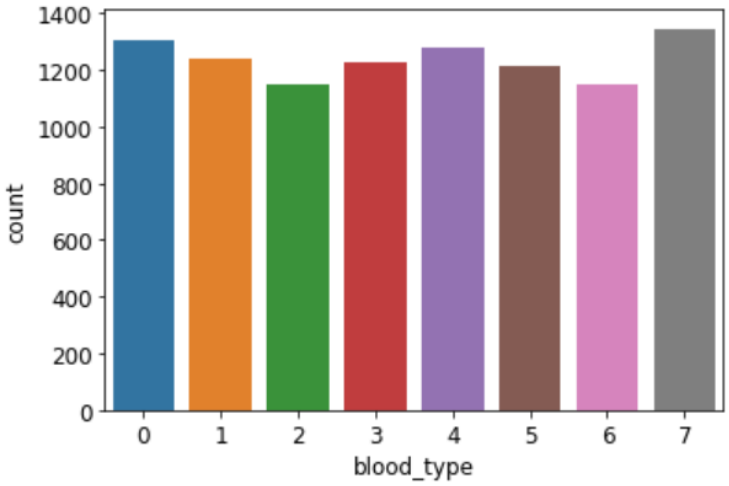

Рисунок 83. Столбчатая диаграмма на основе категориального признака blood_type

В категориальных данных довольно часто могут возникать несоответствия (опечатки). Ниже рассмотрен пример, в котором продемонстрировано выявление несоответствий для категориального признака, описывающего семейное положение.

Определим, какие уникальные значения имеются в исходном датафрейме для категориального признака, описывающего семейное положение.

На рис. 84 приведен фрагмент программного кода для изучения несоответствий для категориального признака, описывающего семейное положение.

На рис. 85 приведены результаты изучения несоответствий для категориального признака, описывающего семейное положение.

```
# изучение несоответствий для категориального признака,
# описывающего семейное положение (marriage status)
main_data['marriage_status'].unique()
```

Рисунок 84. Фрагмент программного кода для изучения несоответствий для категориального признака, описывающего семейное положение

```
array(['married', 'MARRIED', ' married', 'unmarried ', 'divorced',
       'unmarried', 'UNMARRIED', 'separated'], dtype=object)
```

Рисунок 85. Результаты изучения несоответствий для категориального признака, описывающего семейное положение

Очевидно, что имеются избыточные значения для этого категориального признака из-за заглавных букв, а также – из-за пробелов в начале и конце слов.

Сначала решим проблему с заглавными буквами.

На рис. 86 приведен фрагмент программного кода с удалением значений с заглавными буквами для категориального признака, описывающего семейное положение.

На рис. 87 приведены записи после удаления значений с заглавными буквами для категориального признака, описывающего семейное положение.

```
# удаление значений с заглавными буквами
# для категориального признака,
# описывающего семейное положение (marriage status)
inconsistent_data = main_data.copy()
inconsistent_data['marriage_status'] = inconsistent_data['marriage_status']\
    .str.lower()
inconsistent_data['marriage_status'].unique()
```

Рисунок 86. Фрагмент программного кода с удалением значений с заглавными буквами для категориального признака, описывающего семейное положение

```
array(['married', ' married', 'unmarried ', 'divorced', 'unmarried',
       'separated'], dtype=object)
```

Рисунок 87. Записи после удаления значений с заглавными буквами для категориального признака, описывающего семейное положение

Затем решим проблему с пробелами в начале и конце слов.

На рис. 88 приведен фрагмент программного кода с удалением пробелов из-за пробелов в начале и конце слов для категориального признака, описывающего семейное положение.

На рис. 89 приведены записи после удаления пробелов в начале и конце слов для категориального признака, описывающего семейное положение.

```
# удаление пробелов в начале и конце слов
inconsistent_data['marriage_status'] = inconsistent_data['marriage_status']\
    .str.strip()
inconsistent_data['marriage_status'].unique()
```

Рисунок 88. Фрагмент программного кода с удалением пробелов в начале и конце слов для категориального признака, описывающего семейное положение

```
array(['married', 'unmarried', 'divorced', 'separated'], dtype=object)
```

Рисунок 89. Записи после удаления пробелов в начале и конце слов для категориального признака, описывающего семейное положение

На рис. 90 приведен фрагмент программного кода с построением таблицы смежности. С помощью параметра `margins`, значение которого установлено как `margins=True`, установлен вывод агрегирующих значений по строкам и столбцам в соответствии со значением параметра `aggfunc`, установленным как `aggfunc=np.sum`. При этом при значении параметра `normalize`, установленном как `normalize=True`, осуществлен вывод значений в относительных частотах.

На рис. 91 приведена построенная таким образом таблица смежности.

```
pd.crosstab(
    inconsistent_data['marriage_status'],
    inconsistent_data['device'],
    values = inconsistent_data['income'],
    aggfunc=np.sum, normalize=True, margins=True
)
```

Рисунок 90. Фрагмент программного кода с построением таблицы смежности

Числовые данные, например, признак «`доход`» (`income`) в рассматриваемом примере, могут быть сопоставлены с различными группами. Это помогает получить больше информации об исследуемом наборе данных.

device	AndroidOS	Linux	MacOS	Windows	iOS	All
marriage_status						
divorced	0.023130	0.023538	0.027294	0.024551	0.024912	0.123425
married	0.077279	0.079526	0.074883	0.075944	0.072167	0.379798
separated	0.022665	0.022286	0.026907	0.025699	0.027120	0.124677
unmarried	0.074979	0.071323	0.072414	0.077300	0.076083	0.372100
All	0.198052	0.196674	0.201499	0.203494	0.200282	1.000000

Рисунок 91. Таблица смежности

На рис. 92 приведен фрагмент программного кода с выводом диапазона для признака «доход» (income).

На рис. 93 приведены максимальное и минимальное значения для признака «доход» (income).

```
# вывод диапазона для признака, описывающего доход (income) в наборе данных
print(f"Max income - {max(main_data['income'])},\
      Min income - {min(main_data['income'])}")
```

Рисунок 92. Фрагмент программного кода с выводом диапазона для признака «доход» (income)

```
Max income - 190000,      Min income - 40000
```

Рисунок 93. Максимальное и минимальное значения для признака «доход» (income)

Создадим поддиапазоны и соответствующие им метки для признака «доход» (income), используя метод cut библиотеки pandas.

На рис. 94 приведен фрагмент программного кода с добавлением нового категориального признака income_groups, определяющего группирование по поддиапазонам для признака «доход» (income).

```
# создание групп по доходам
range = [40000, 75000, 100000, 125000, 150000, np.inf]
labels = ['40k-75k', '75k-100k', '100k-125k', '125k-150k', '150k+']

remapping_data = main_data.copy()
remapping_data['income_groups'] = pd.cut(remapping_data['income'],
                                         bins=range,
                                         labels=labels)

remapping_data.head()
```

Рисунок 94. Фрагмент программного кода с добавлением нового категориального признака income_groups, определяющего группирование по поддиапазонам для признака «доход» (income)

На рис. 95 приведен фрагмент датафрема с добавленным новым категориальным признаком `income_groups`.

	first_name	last_name	blood_type	marriage_status	income	device	income_groups
0	Abdul	Colon	A+	married	145000	AndroidOS	125k-150k
1	Abdul	Pierce	B+	married	85000	MacOS	75k-100k
2	Desirae	Pierce	B+	MARRIED	130000	iOS	125k-150k
3	Shannon	Gibson	A+	married	175000	MacOS	150k+
4	Desirae	Little	B+	unmarried	130000	MacOS	125k-150k

Рисунок 95. Фрагмент датафрема с добавленным новым категориальным признаком income_groups

Выполним визуализацию данных с учетом нового категориального признака `income_groups`.

На рис. 96 приведен фрагмент программного кода, реализующего построение столбчатой диаграммы на основе нового категориального признака `income_groups`, определяющего группирование по поддиапазонам для признака «доход» (`income`).

```
# визуализация
remapping_data['income_groups'].value_counts().plot.bar();
```

Рисунок 96. Фрагмент программного кода, реализующего построение столбчатой диаграммы на основе нового категориального признака income_groups, определяющего группирование по поддиапазонам для признака «доход» (income)

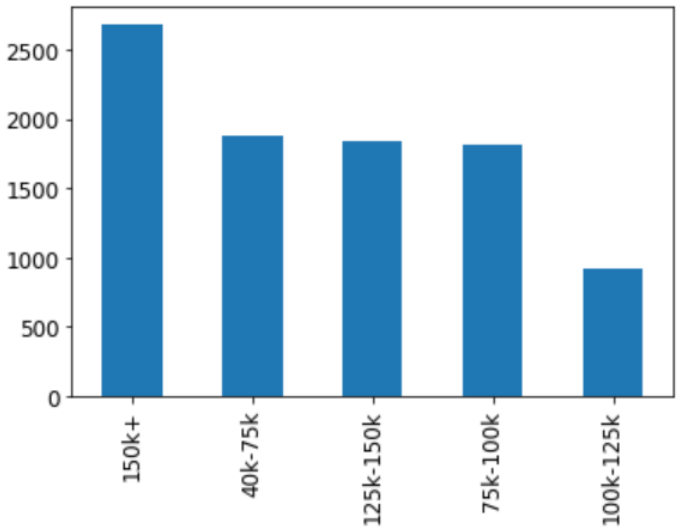

Рисунок 97. Столбчатая диаграмма на основе нового категориального признака income_groups, определяющего группирование по поддиапазонам для признака «доход» (income)

На рис. 97 приведена столбчатая диаграмма на основе нового категориального признака `income_groups`, определяющего группирование по поддиапазонам для признака «`доход`» (`income`).

Многие алгоритмы машинного обучения работают только с числовыми (количественными) значениями. Следовательно, категориальные данные должны быть преобразованы в числа. Ниже рассмотрено применение возможных подходов к проблеме перекодировки категориальных данных в числовые.

Можно осуществить кодирование с использованием метода `label encoder` [16] (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html). В этом случае будет осуществлена нумерация категорий категориального признака числами от 0 до `num_categories` – 1 (число категорий минус 1).

Выполним такую кодировку для категориального признака `blood_type`, описывающего группу крови.

На рис. 98 приведен фрагмент программного кода, реализующего кодировку для категориального признака `blood_type`, описывающего группу крови.

На рис. 99 приведены результаты кодировки для категориального признака `blood_type`, описывающего группу крови.

На рис. 100 приведены упорядоченные по алфавиту значения категориального признака `blood_type`, коды которых приведены на рис. 99.

```
1 le = LabelEncoder()
2 without_bogus_records['blood_type'] = le.fit_transform(
3     without_bogus_records['blood_type'])
4 without_bogus_records['blood_type'].unique()
```

Рисунок 98. Фрагмент программного кода, реализующего кодировку для категориального признака blood_type, описывающего группу крови

```
array([0, 1, 4, 5, 2, 3, 6, 7])
```

Рисунок 99. Результаты кодировки для категориального признака blood_type, описывающего группу крови

```
1 le.classes_
```

```
array(['A+', 'A-', 'AB+', 'AB-', 'B+', 'B-', 'O+', 'O-'], dtype=object)
```

Рисунок 100. Упорядоченные по алфавиту значения категориального признака blood_type, коды которых приведены на рис. 88.

Можно осуществить кодирование с использованием инструмента `one-hot encoder` [17] (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html).

В этом случае будут созданы новые бинарные признаки, принимающие значения 0 или 1. При этом число новых признаков будет равно числу категорий категориального признака, к которому применяется метод `one-hot encoder`.

Выполним такую кодировку для категориального признака `marriage_status`, описывающего семейное положение.

На рис. 101 приведен фрагмент программного кода, реализующего кодировку для категориального признака `marriage_status`, описывающего семейное положение.

На рис. 102 приведены результаты кодировки для категориального признака `marriage_status`, описывающего семейное положение.

```
inconsistent_data = pd.get_dummies(inconsistent_data,
                                   columns=['marriage_status'])
inconsistent_data.head()
```

Рисунок 101. Фрагмент программного кода, реализующего кодировку для категориального признака marriage_status, описывающего семейное положение

first_name	last_name	blood_type	income	device	marriage_status_divorced	marriage_status_married	marriage_status_separated	marriage_status_unmarried
Abdul	Colon	A+	145000	AndroidOS	0	1	0	0
Abdul	Pierce	B+	85000	MacOS	0	1	0	0
Desirae	Pierce	B+	130000	iOS	0	1	0	0
Shannon	Gibson	A+	175000	MacOS	0	1	0	0
Desirae	Little	B+	130000	MacOS	0	0	0	1

Рисунок 102. Результаты кодировки для категориального признака marriage_status, описывающего семейное положение

Категориальные данные могут быть порядковыми. В этом случае важен порядок кодировки и целесообразно использовать метод `ordinal encoder`, реализующий порядковое кодирование [18] (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html#sklearn.preprocessing.OrdinalEncoder).

Выполним порядковое кодирование по новому категориальному признаку (`income_groups`), сохранив порядок:

40K-75K < 75K-100K < 100K-125K < 125K-150K < 150K+.

На рис. 103 приведен фрагмент программного кода, реализующего кодировку для категориального признака `income_groups`.

На рис. 104 приведены результаты кодировки для категориального признака `income_groups`.

```
custom_map = {'40k-75k': 1, '75k-100k': 2, '100k-125k': 3,
              '125k-150k': 4, '150k+': 5}
remapping_data['income_groups'] = remapping_data['income_groups']\
    .map(custom_map)
remapping_data.head()
```

Рисунок 103. Фрагмент программного кода, реализующего кодировку для категориального признака income_groups

	first_name	last_name	blood_type	marriage_status	income	device	income_groups
0	Abdul	Colon	A+	married	145000	AndroidOS	4
1	Abdul	Pierce	B+	married	85000	MacOS	2
2	Desirae	Pierce	B+	MARRIED	130000	iOS	4
3	Shannon	Gibson	A+	married	175000	MacOS	5
4	Desirae	Little	B+	unmarried	130000	MacOS	4

Рисунок 104. Результаты кодировки для категориального признака income_groups

```
# импорт библиотек
import random
import pandas as pd

phone_numbers = []

for i in range(10):
# номер телефона имеет длину 6 или 7
    number = random.randint(100000, 9999999)
# +495 код вставляется в некоторых случаях
    if(i % 2 == 0):
        phone_numbers.append('+495 ' + str(number))
    else:
        phone_numbers.append(str(number))
phone_numbers_data = pd.DataFrame({
    'phone_numbers': phone_numbers
})

phone_numbers_data.head()
```

Рисунок 105. Фрагмент программного кода, реализующего создание датафрейма с номерами телефонов

	phone_numbers
0	+495 5590489
1	9878768
2	+495 120439
3	1781984
4	+495 4364115

Рисунок 106. Датафрейм с номерами телефонов

Следует отметить, что можно использовать различные варианты кодирования в зависимости от задач, которые должны быть решены (https://www.geeksforgeeks.org/feature-encoding-techniques-machine-learning/).

Существенный интерес представляет проблема очистки категориальных данных. Появление этой проблемы может быть вызвано, например, ошибками ввода данных или применением разных подходов к формированию одних и тех же по смыслу значений категориальных признаков.

Например, номер телефона может быть введен с кодом города или без него. В связи с этим должна быть выполнена очистка значений этого категориального признака.

Создадим датафрейм с номерами телефонов.

На рис. 105 приведен фрагмент программного кода, реализующего создание датафрейма с номерами телефонов.

На рис. 106 приведен фрагмент этого датафрейма.

Очевидно, что для единообразия записи номера телефона необходимо удалить код города из номера телефона. Кроме того, оставим только те номера телефонов, которые содержат 7 цифр (в частности, необходимо удалить строку с индексом 2, т.к. номер телефона содержит только 6 цифр).

На рис. 107 приведен фрагмент программного кода, реализующего очистку датафрейма с номерами телефонов.

На рис. 108 приведен фрагмент очищенного датафрейма с номерами телефонов.

```
phone_numbers_data['phone_numbers'] = phone_numbers_data['phone_numbers']\
    .str.replace('\+495 ', '')

num_digits = phone_numbers_data['phone_numbers'].str.len()
invalid_numbers_index = phone_numbers_data[num_digits < 7].index
phone_numbers_data['phone_numbers'] = phone_numbers_data.drop(
    invalid_numbers_index)
phone_numbers_data = phone_numbers_data.dropna()

phone_numbers_data.head()
```

Рисунок 107. Фрагмент программного кода, реализующего очистку датафрейма с номерами телефонов

	phone_numbers
0	5590489
1	9878768
3	1781984
4	4364115
5	6581364

Рисунок 108. Фрагмент очищенного датафрейма с номерами телефонов

11. Пример визуализации набора данных с категориальными признаками

Рассмотрим на примере набора данных `penguins`, предоставленным библиотекой `seaborn`, аспекты визуализации с использованием `catplot`.

На рис. 109 приведен фрагмент программного кода с загрузкой набора данных `penguins` в датафрейм.

На рис. 110 приведен фрагмент этого датафрейма.

```
# импорт библиотек
import seaborn as sns

df = sns.load_dataset('penguins') # загрузка датасета

df.head()
```

Рисунок 109. Фрагмент программного кода с загрузкой набора данных penguins в датафрейм

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female

Рисунок 110. Фрагмент набора данных penguins

На рис. 111 приведен фрагмент программного кода реализующего визуализацию набора данных `penguins` по категориальному признаку `island` и количественному признаку `bill_length_mm`. По умолчанию при визуализации используется `stripplot` (полосовая диаграмма).

На рис. 112 приведены результаты визуализации, позволяющие исследовать взаимосвязь между двумя признаками.

```
# импорт библиотек
import seaborn as sns
import matplotlib.pyplot as plt

df = sns.load_dataset('penguins')  # загрузка датасета

sns.catplot(data=df, x='island', y='bill_length_mm') # по умолчанию будет использован strip plot
plt.show()
```

Рисунок 111. Фрагмент программного кода реализующего визуализацию набора данных penguins по категориальному признаку island и количественному признаку bill_length_mm с использованием stripplot

Можно изменить тип диаграммы, задав то или иное допустимое значение параметру `kind`. Так, если `kind='bar'`, то будет построена столбчатая диаграмма, которая позволит объединить (агрегировать) данные с помощью некоторой метрики, например, посредством расчета среднего значения. Кроме того, будут отображены планки ошибок (error bars). По умолчанию будет использован процесс, называемый бутстрэпом (bootstrapping), чтобы вернуть 95-процентный доверительный интервал, определяющий, что новые данные попадут в диапазон ошибок. При этом будет выполнена многократная выборка данных с заменой для расчета среднего значения (по умолчанию выборка выполняется 1000 раз для каждого значения по оси `x`). Это позволит определить доверительный интервал, в который новые значения будут попадать с вероятностью 95%. При необходимости можно изменить как доверительный интервал, так и количество итераций бутстрэпа. Также можно изменить проценты, используемые в доверительном интервале, передав в функцию `catplot` в качестве значения параметра `errorbar` кортеж, который содержит (`'ci', n`), где `n` – проценты, которые следует использовать. Отметим, что параметр `errorbar` является частью определения функции `barplot`, а не функции `catplot`. При этом можно использовать различные варианты расчета ошибок: `'ci'` (доверительный интервал – вариант, отвечающий за непараметрическую неопределенность, `'pi'` (процентильный интервал – вариант, отвечающий за непараметрический спред), `'se'` (стандартная ошибка – вариант, отвечающий за параметрическую неопределенность), `'sd'` (стандартное отклонение – вариант, отвечающий за параметрический разброс).

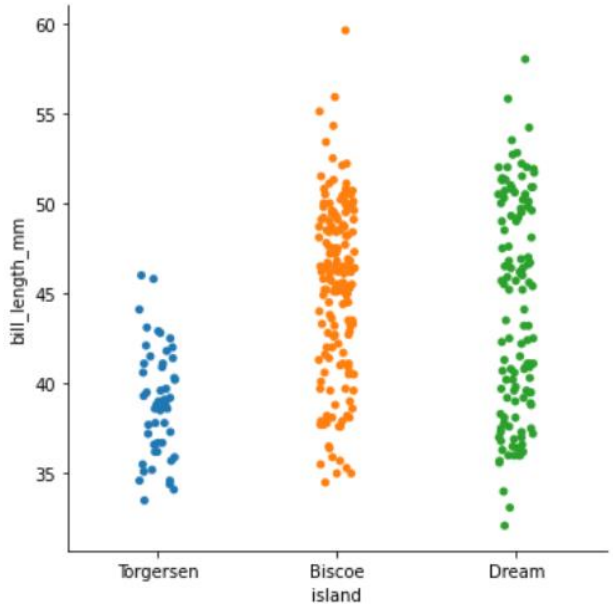

Рисунок 112. Результаты визуализации набора данных penguins по категориальному признаку island и количественному признаку bill_length_mm с использованием stripplot

На рис. 113 приведен фрагмент программного кода, реализующего визуализацию набора данных `penguins` по категориальному признаку `island` и количественному признаку `bill_length_mm`. При этом при визуализации используется `barplot` (столбчатая диаграмма) и значение параметра `errorbar` определено как `errorbar=('ci', 99)`.

На рис. 114 приведены результаты визуализации, позволяющие исследовать взаимосвязь между двумя признаками с использованием `barplot`.

```
# импорт библиотек
import seaborn as sns
import matplotlib.pyplot as plt

df = sns.load_dataset('penguins')  # загрузка датасета

sns.catplot(data=df, x='island', y='bill_length_mm', kind='bar',
            errorbar=('ci', 99))
plt.show()
```

Рисунок 113. Фрагмент программного кода, реализующего визуализацию набора данных penguins по категориальному признаку island и количественному признаку bill_length_mm с использованием barplot

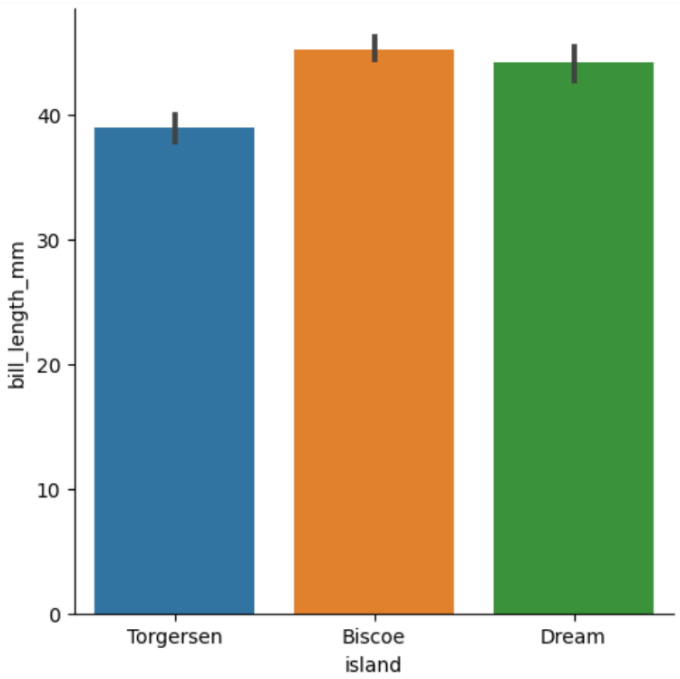

Рисунок 114. Результаты визуализации набора данных penguins по категориальному признаку island и количественному признаку bill_length_mm с использованием barplot

На рис. 115 приведены результаты визуализации набора данных `penguins` по категориальному признаку `island` и количественному признаку

bill_length_mm с использованием violinplot. Эти результаты получены в результате замены kind='bar' на kind='violin' в программном коде, приведенном на рис. 113.

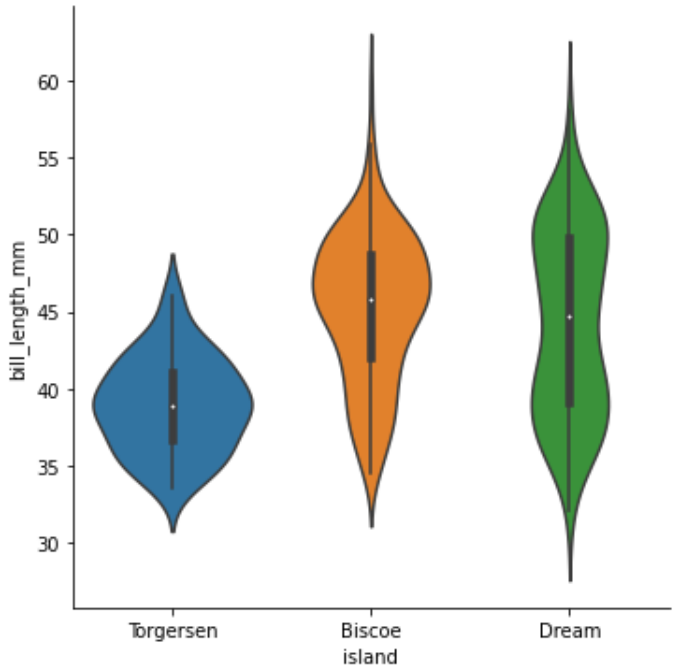

Рисунок 115. Результаты визуализации набора данных penguins по категориальному признаку island и количественному признаку bill_length_mm с использованием violinplot

Пусть необходимо разделить пингвинов на каждом острове по полу. Это можно сделать, добавив дополнительные уровни детализации (дополнительные измерения), используя цвет.

В catplot можно отобразить дополнительные измерения на диаграмме, оставив её при этом двухмерной. С этой целью можно использовать параметр hue: присвоив ему название некоторого признака, для которого на диаграмме будет выполнено разделение значений по цвету. В результате для каждого уникального значения признака будет установлен собственный цвет.

Добавим дополнительную переменную (дополнительный признак) в catplot, определив значение параметра hue как hue='sex' для использования признака sex (т.е. дополнительного столбца из датафрейма) для разделения данных по категориям с применением цвета.

На рис. 116 приведен фрагмент программного кода, реализующего визуализацию набора данных penguins по категориальному признаку island и количественному признаку bill_length_mm. При этом при визуализации

используется violinplot (скрипичная диаграмма) и значение параметра hue определено как hue='sex'.

На рис. 117 приведены результаты визуализации набора данных penguins по категориальному признаку island и количественному признаку bill_length_mm с использованием violinplot и значением параметра hue, определенным как hue='sex'.

```
# импорт библиотек
import seaborn as sns
import matplotlib.pyplot as plt

df = sns.load_dataset('penguins') # загрузка датасета

sns.catplot(data=df, x='island', y='bill_length_mm', kind='violin', hue='sex')
plt.show()
```

Рисунок 116. Фрагмент программного кода, реализующего визуализацию набора данных penguins по категориальному признаку island и количественному признаку bill_length_mm с использованием violinplot и разделением по признаку sex

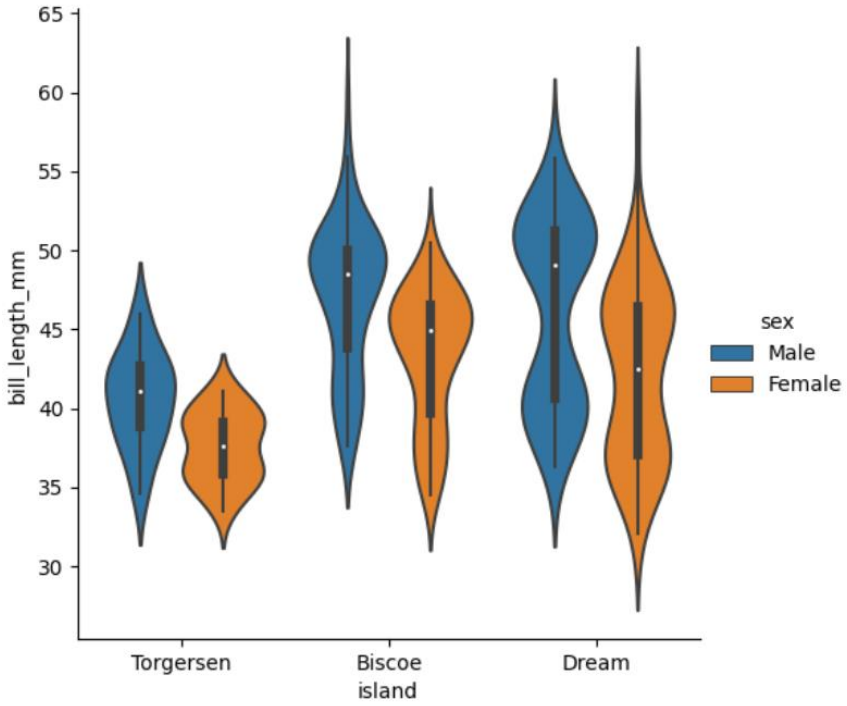

Рисунок 117. Результаты визуализации набора данных penguins по категориальному признаку island и количественному признаку bill_length_mm с использованием violinplot и разделением по признаку sex

Чтобы создать столбцы подграфиков, можно использовать параметр col, определив его через метку столбца датафрейма или массив данных. Тогда число столбцов с рисунками будет равно числу уникальных значений в столбце дата-

фрейма (массива). Если уникальных значений много, то имеет смысл использовать дополнительный параметр `col_wrap`, с помощью которого можно указать, сколько столбцов следует расположить в одной строке до перехода к новой строке.

```
# импорт библиотек
import seaborn as sns
import matplotlib.pyplot as plt

df = sns.load_dataset('penguins') # загрузка датасета

sns.catplot(data=df, x='island', y='bill_length_mm', kind='violin', col='sex')
plt.show()
```

Рисунок 118. Фрагмент программного кода, реализующего визуализацию набора данных penguins по категориальному признаку island и количественному признаку bill_length_mm с использованием violinplot, разделением по признаку sex и выводом на разных рисунках (в разных столбцах)

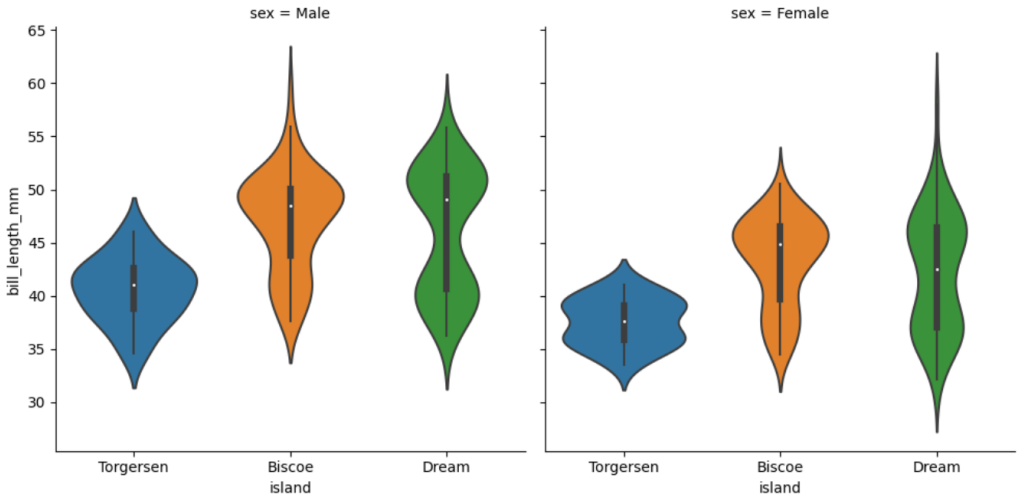

Рисунок 119. Результаты визуализации набора данных penguins по категориальному признаку island и количественному признаку bill_length_mm с использованием violinplot, разделением по признаку sex и выводом на разных рисунках (в разных столбцах)

На рис. 118 приведен фрагмент программного кода реализующего визуализацию набора данных `penguins` по категориальному признаку `island` и количественному признаку `bill_length_mm`. При этом при визуализации используется `violinplot` (скрипичная диаграмма) и значение параметра `col` определено как `col='sex'`.

На рис. 119 приведены результаты визуализации набора данных `penguins` по категориальному признаку `island` и количественному признаку

bill_length_mm с использованием violinplot и значением параметра col, определенным как col='sex'. Т.к. признак sex имеет всего 2 уникальных значения, то в сетке фасетов (FacetGrid) располагаются всего 2 рисунка, расположенные в пределах видимости (нет необходимости использовать полосу прокрутки).

```
# импорт библиотек
import seaborn as sns
import matplotlib.pyplot as plt

df = sns.load_dataset('penguins') # загрузка датасета

sns.catplot(data=df, x='sex', y='bill_length_mm', kind='violin', col='island', col_wrap=2)
plt.show()
```

Рисунок 120. Фрагмент программного кода, реализующего визуализацию набора данных penguins по категориальному признаку sex и количественному признаку bill_length_mm с использованием violinplot, разделением по признаку island и выводом на разных рисунках (в разных столбцах) по 2 рисунка в ряду

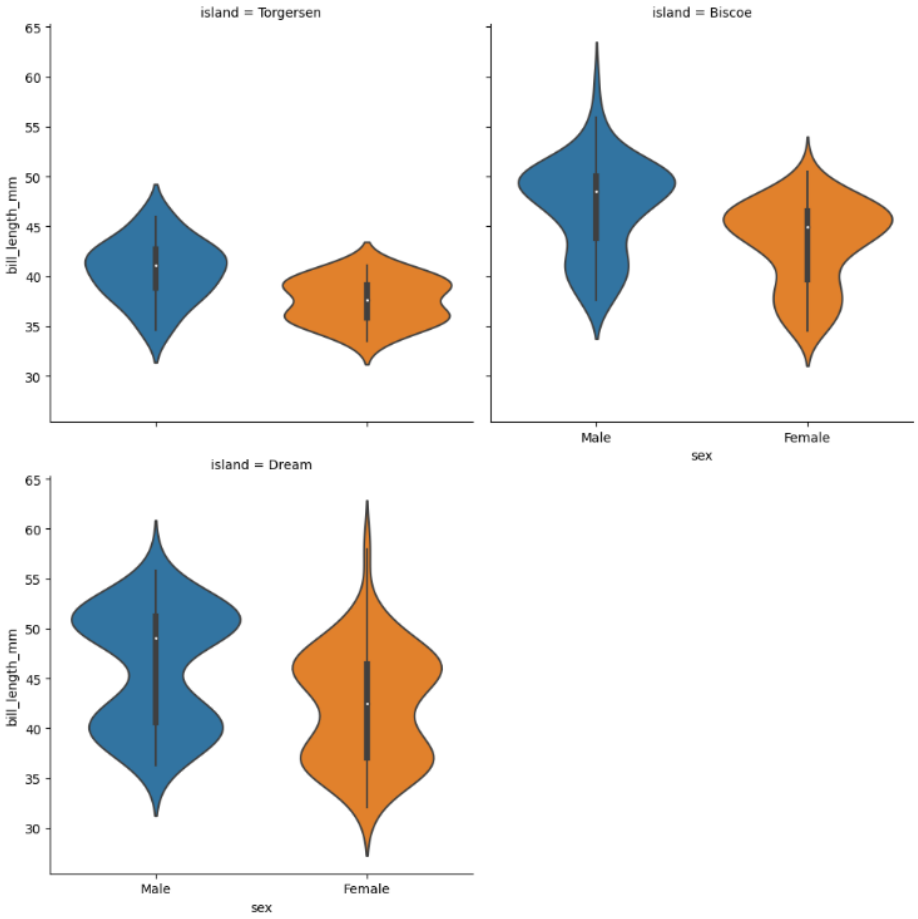

Рисунок 121. Результаты визуализации набора данных penguins по категориальному признаку sex и количественному признаку bill_length_mm с использованием violinplot, разделением по признаку island на разных рисунках (в разных столбцах) по 2 рисунка в ряду

На рис. 120 приведен фрагмент программного кода реализующего визуализацию набора данных `penguins` по категориальному признаку `sex` и количественному признаку `bill_length_mm`. При этом при визуализации используется `violinplot` (скрипичная диаграмма), значение параметра `col` определено как `col='island'`, а значение параметра `col_wrap` определено как `col_wrap=2`.

На рис. 121 приведены результаты визуализации набора данных `penguins` по категориальному признаку `sex` и количественному признаку `bill_length_mm` с использованием `violinplot`, значением параметра `col`, определенным как `col='island'`, и значением параметра `col_wrap`, определенным как `col_wrap=2`. Т.к. признак `island` имеет 3 уникальных значения, то в сетке фасетов (FacetGrid) располагаются 2 рисунка в ряду, находящиеся в пределах видимости, третий рисунок расположен в следующем ряду.

Можно комбинировать параметры `rows` и `col` для указания категориальных признаков, используемых в качестве дополнительных, при визуализации соответственно по строкам и столбцам в сетке фасетов (FacetGrid).

На рис. 122 приведен фрагмент программного кода реализующего визуализацию набора данных `penguins` по категориальному признаку `island` и количественному признаку `bill_length_mm`. При этом при визуализации используется `violinplot` (скрипичная диаграмма), значение параметра `row` определено как `row='sex'`, а значение параметра `col` определено как `col='species'`.

```
# импорт библиотек
import seaborn as sns
import matplotlib.pyplot as plt

df = sns.load_dataset('penguins') # загрузка датасета

cat = sns.catplot(data=df, x='island', y='bill_length_mm', kind='violin', row='sex', col='species')
cat.fig.subplots_adjust(top=0.92)
cat.fig.suptitle('Сравнение пингвинов (comparing penguins)')
cat.set_titles(row_template='Penguin Sex: {row_name}', col_template='Penguin Species: {col_name}')

cat.set_xlabels('Bill Length (mm)')
cat.set_ylabels('Island Name')

plt.show()
```

Рисунок 122. Фрагмент программного кода реализующего визуализацию набора данных penguins по категориальному признаку island и количественному признаку bill_length_mm с использованием violinplot, разделением по признакам sex(по строкам) и species (по столбцам) на разных рисунках

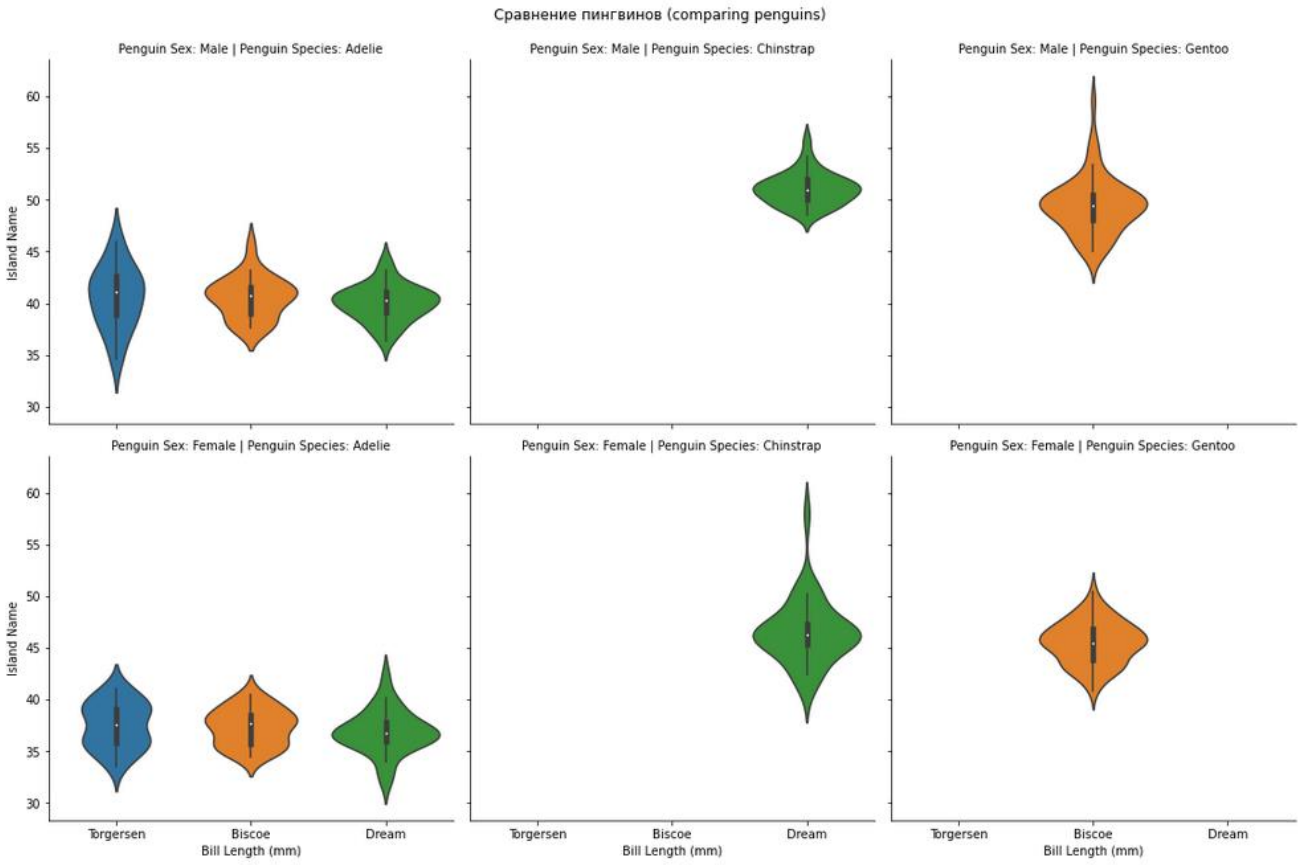

Рис. 123. Результаты визуализации
набора данных penguins
по категориальному признаку island и количественному признаку bill_length_mm
с использованием violinplot, разделением по признакам sex(по строкам)
и species (по столбцам)
на разных рисунках

На рис. 123 приведены результаты визуализации набора данных `penguins` по категориальному признаку `island` и количественному признаку `bill_length_mm` с использованием `violinplot`, значением параметра `row`, определенным как `row='sex'`, и значением параметра `col`, определенным как `col='species'`.

На рис. 124 показано, как выглядят результаты визуализации с применением функций `stripplot`, `swarmplot`, `boxplot`, `violinplot`, `boxenplot`, `pointplot`, `barplot countplot`, вызываемых функцией `catplot` для одной и той же задачи, связанной с анализом распределения длины клюва (по количественному признаку `bill_length_mm`) пингвина на островах (по категориальному признаку `island`).

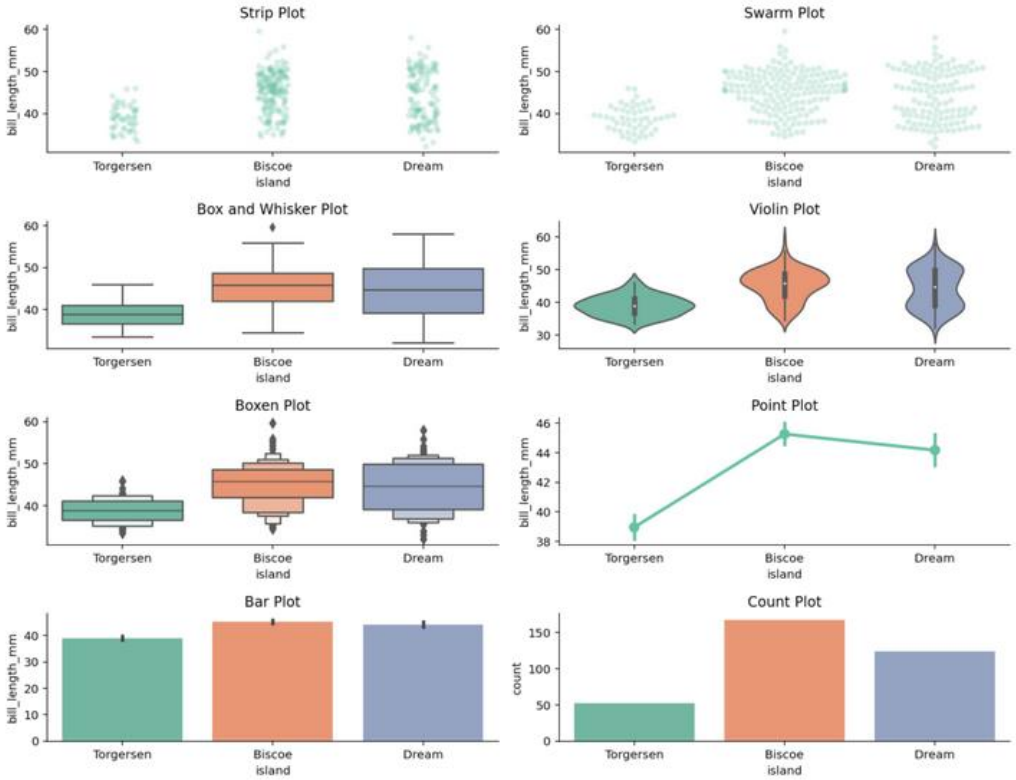

Рисунок 124. Результаты визуализации с применением различных инструментов для catplot *для одной и той же задачи, связанной с анализом распределения длины клюва пингвина на островах*

ВАРИАНТЫ И ЗАДАНИЯ

Необходимо выполнить разведочный анализ предлагаемого набора данных в соответствии с предложенным вариантом (табл. 5), используя программные реализации инструментов разведочного анализа.

Выделить в предлагаемом наборе данных все категориальные признаки и получить по ним сводную информацию.

Выполнить расчет метрик центральной тенденции.

Построить столбчатые и круговые диаграммы, используя инструменты визуализации, предлагаемые библиотеками `matplotlib`, `seaborn`, `pyplot` и `pandas`. Реализовать применение различных инструментов визуализации, предлагаемых функцией `catplot` библиотеки `seaborn`. Реализовать применение различных инструментов визуализации, предлагаемых библиотекой `Bokeh`.

Построить все возможные варианты сводных таблиц с применением метода `pivot_table` и таблиц сопряженности с применением метода `crosstab`.

Изучить методы кодирования категориальных признаков и применить их на практике.

Решить задачу кластеризации предлагаемого набора данных [19].

Информацию по работе с языком Python можно получить в [20, 21].

Варианты

Таблица 5. Варианты наборов данных

Вариант	Источник набора данных
1	https://www.kaggle.com/code/josh1337/bankchurners
2	https://www.kaggle.com/datasets/lepchenkov/usedcarscatalog
3	https://www.kaggle.com/datasets/heeraldedhia/bike-buyers
4	https://www.kaggle.com/datasets/shubhambathwal/flight-price-prediction
5	https://www.kaggle.com/datasets/volodymyrgavrysh/bank-marketing-campaigns-dataset
6	https://www.kaggle.com/datasets/gauravduttakiit/categorical-country-geotags
7	https://www.kaggle.com/datasets/prakharrathi25/banking-dataset-marketing-targets
8	https://www.kaggle.com/datasets/thedevastator/employee-attrition-and-factors
9	https://www.kaggle.com/datasets/ruthgn/bank-marketing-data-set
10	https://www.kaggle.com/datasets/rkiattisak/salaly-prediction-for-beginer
11	https://www.kaggle.com/datasets/manishkc06/startup-success-prediction
12	https://www.kaggle.com/datasets/miguelcorraljr/brilliant-diamonds
13	https://www.kaggle.com/datasets/brsdincer/alzheimer-features
14	https://www.kaggle.com/datasets/gabrielramos87/an-online-shop-business
15	https://www.kaggle.com/datasets/dev523/ml-marathon-dataset-by-azure-developer-community
16	https://www.kaggle.com/datasets/arashnic/hr-analytics-job-change-of-data-scientists
17	https://www.kaggle.com/datasets/jcraggy/marvel-vs-dc-imdb-rotten-tomatoes
18	https://www.kaggle.com/datasets/xwolf12/malicious-and-benign-websites
19	https://www.kaggle.com/datasets/aakashverma8900/portuguese-bank-marketing
20	https://www.kaggle.com/datasets/gabrielramos87/bike-trips
21	https://www.kaggle.com/datasets/thedevastator/bigmart-product-sales-factors
22	https://www.kaggle.com/datasets/devsubhash/fitness-trackers-products-ecommerce
23	https://www.kaggle.com/datasets/vysakhvms/cars-india-dataset
24	https://www.kaggle.com/datasets/chancev/carsforsale
25	https://www.kaggle.com/datasets/thedevastator/nfl-team-stats-and-outcomes
26	https://www.kaggle.com/datasets/ektanegi/spotifydata-19212020
27	https://www.kaggle.com/datasets/ranja7/vehicle-insurance-customer-data
28	https://www.kaggle.com/datasets/floser/french-motor-claims-datasets-fremtpl2freq
29	https://www.kaggle.com/datasets/devsubhash/television-brands-ecommerce-dataset
30	https://www.kaggle.com/datasets/fydrose/macros-of-popular-high-protein-foods

СПИСОК ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Тьюки Дж. Анализ результатов наблюдений. Разведочный анализ. – М.: Мир, 1981. – 696 с.
2. Брюс П., Брюс Э. Практическая статистика для специалистов Data Science. СПб.: БХВ-Петербург. 2018. С. 19–58. 304 с.
3. John W. Tukey. The Future of Data Analysis // The Annals of Mathematical Statistics, Vol. 33. No. 1. 1962. pp. 1-67.
4. Tukey J.W. Exploratory data analysis. Reading, PA: Addison-Wesley. 1977. 711 p.
5. Behrens J.T. Principles and Procedures of Exploratory Data Analysis // Psychological Methods Copyright 1997 by the American Psychological Association, Inc. 1997. Vol. 2. No. 2. pp. 131-160.
6. Exploratory data analysis [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Exploratory_data_analysis, свободный (дата обращения 04.05.2023).
7. Sailem H.Z., Sero J.E., Bakal C. Visualizing cellular imaging data using Pheno Plot // Nature Communication. 2015. Jan 8; 6: 5825. doi: 10.1038/ncomms682.
8. Демидова Л.А. Разведочный анализ данных. Python. Часть 1 [Электронный ресурс]: Учебное-методическое пособие / Демидова Л.А. – М.: МИРЭА – Российский технологический университет, 2022. 102 с.
9. statistics 1.0.3.5 [Электронный ресурс]. – Режим доступа: https://pypi.org/project/statistics/, свободный (дата обращения 04.05.2023).
10. NumPy [Электронный ресурс]. – Режим доступа: https://numpy.org/, свободный (дата обращения 04.05.2023).
11. pandas [Электронный ресурс]. – Режим доступа: https://pandas.pydata.org/, свободный (дата обращения 04.05.2023).
12. matplotlib: Visualization with Python [Электронный ресурс]. – Режим доступа: https://matplotlib.org/, свободный (дата обращения 04.05.2023).
13. seaborn: statistical data visualization [Электронный ресурс]. – Режим доступа: https://seaborn.pydata.org/, свободный (дата обращения 04.05.2023).
14. Plotly Python Graphing Library [Электронный ресурс]. – Режим доступа: https://plotly.com/python/, свободный (дата обращения 04.05.2023).
15. Bokeh [Электронный ресурс]. – Режим доступа: https://bokeh.org/, свободный (дата обращения 04.05.2023).

16. sklearn.preprocessing.LabelEncoder [Электронный ресурс]. – Режим доступа: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html#sklearn-preprocessing, свободный (дата обращения 04.05.2023).

17. sklearn.preprocessing.OneHotEncoder [Электронный ресурс]. – Режим доступа: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html#sklearn.preprocessing.OneHotEncoder, свободный (дата обращения 04.05.2023).

18. sklearn.preprocessing.OrdinalEncoder [Электронный ресурс]. – Режим доступа: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html#sklearn.preprocessing.OrdinalEncoder, свободный (дата обращения 04.05.2023).

19. Демидова Л.А. Кластерный анализ. Python [Электронный ресурс]: Учебное пособие / Демидова Л.А. — М.: МИРЭА – Российский технологический университет, 2022. — 1 электрон. опт. диск (CD-ROM). 103 с.

20. Лутц М. Изучаем Python. М.: Диалектика. 2020. Том. 1. 832 с.; Том. 2. 720 с.

21. Доусон М. Программируем на Python. СПб.: Питер. 2020. 416 с.

Сведения об авторах

Демидова Лилия Анатольевна, доктор технических наук, профессор, профессор кафедры корпоративных информационных систем Института информационных технологий РТУ МИРЭА.