ВЫСШЕЕ ОБРАЗОВАНИЕ
серия основана в 1996 г.

Министерство науки и высшего образования Российской Федерации
Сибирский федеральный университет

Е.А. СОПОВ

ТЕХНОЛОГИИ ОБРАБОТКИ БОЛЬШИХ ДАННЫХ НА PYTHON

УЧЕБНИК

Москва
ИНФРА-М
2023

УДК 519.682(075.8)
ББК 32.973.2я73
С64

Работа выполнена при поддержке Министерства науки и высшего образования Российской Федерации (грант № 075-15-2022-1121)

Рецензенты:

Семёнкин Е.С., доктор технических наук, профессор кафедры системного анализа и исследования операций Сибирского государственного университета науки и технологий имени академика М.Ф. Решетнева;

Хамисов О.В., доктор физико-математических наук, профессор Института систем энергетики имени Л.А. Мелентьева Сибирского отделения Российской академии наук

Сопов Е.А.

С64 Технологии обработки больших данных на Python : учебник / Е.А. Сопов. — Москва : ИНФРА-М, 2023. — 157 с. : ил. — (Высшее образование).

ISBN 978-5-16-019627-5

В учебнике проблема обработки больших данных рассмотрена с позиции возможности анализировать доступные массивы данных и извлекать из этого анализа практическую пользу в терминах предметной области. Рассмотрены задачи и проблемы импорта и представления данных в виде кадра данных DataFrame и основные этапы исследовательского анализа данных EDA. Основной упор делается на обработку, визуализацию, предварительный анализ и интерпретацию данных для дальнейшего их использования в моделях машинного обучения и искусственного интеллекта. Инструментарий анализа данных показан с применением языка программирования Python и наиболее популярных в данной области библиотек Numpy, Pandas, Matplotlib, Seaborn и Sklearn. Процедура разведочного анализа данных подробно рассмотрена на практических примерах обработки данных для решения задач регрессии и классификации. Все примеры сопровождаются кодом на Python с демонстрацией результатов.

Предназначен для студентов бакалавриата, обучающихся по направлениям подготовки «Системный анализ и управление», «Прикладная информатика», «Информационные системы и технологии», а также для магистрантов направлений подготовки «Инженерия искусственного интеллекта» и «Прикладной искусственный интеллект».

УДК 519.682(075.8)
ББК 32.973.2я73

ISBN 978-5-16-019627-5

Оглавление

Введение

Большие данные или Big Data – одна из важных составляющих технологий современных информационных систем. Про большие данные говорят все: от ИТ-специалистов до менеджеров крупного бизнеса и чиновников правительства [1]. Однако, в этот термин вкладывают разный смысл.

С точки зрения технологических и технических задач, под большими данными имеются в виду такие задачи получения, передачи, ранения и обработки информации, которые не решаются эффективно традиционными способами. Например, с помощью обычных реляционных СУБД. Для описания этих технических особенностей применяют подход «VVV» (три V), суть которого связана с оценкой трех характеристик: объем (Volume), скорость (Velocity), многообразие (Variety) [2]. По мере развития технических средств и появления новых алгоритмических и программных решений, многие задачи, которые несколько лет назад относились к Big Data перестают быть «большими», а инструменты работы с ними превращаются в традиционную информационную технологию. Более того, в последние годы больше не наблюдается такой стремительный рост как в начале 2000-х. Например, американская исследовательская и консалтинговая компания Gartner, специализирующаяся на рынках информационных технологий, в 2016 исключила Big Data из числа прорывных технологий (Emerging Technologies). В исследовании «The Demise of Big Data, Its Lessons and the State of Things to Come» («Смерть Больших Данных, извлеченные уроки и ситуация в будущем») говорится о том, что прошла эпоха спекуляция на тему проблемы больших данных, и появилась новая прикладная ИТ-технология [3].

Однако в широком смысле, говоря о больших данных, имеют в виду то, что связанно с решением аналитических задач, которые решаются в областях прикладной статистики, машинного обучения и искусственного интеллекта, объединенных вместе в направление Data Science (наука о данных) [4]. Речь идет о новом социально-экономическом феномене, связанном с появлением технологических возможностей анализировать практически неограниченные массивы данных (в предельном случае, весь мировой объем данных) и извлекать из этого анализа практическою пользу в терминах предметной области или бизнеса, а не с точки зрения математических или технических критериев. Можно дать аналогичное определение Big Data в контексте аналитических задач: большие данные – это такие данные, с которыми аналитик не может справиться, используя традиционные инструменты анализа. Поэтому Data Science делает упор на применение методов машинного обучения и искусственного интеллекта, оставляя за аналитиком знания предметной области для постановки высокоуровневых целей и интерпретации результатов.

Поскольку большие данные обычно не удается обработать обычными электронными таблицами и стандартными методами проверки статистических гипотез о свойствах данных, были предложены разные инструменты для решения задач анализа данных. На сегодня общепризнанным способом обработки данных стало представление их в виде так называемого «кадра данных» (DataFrame), для которого строятся визуализации, проводится исследовательский (разведочный)

анализ (Exploratory Data Analysis, EDA) и строятся модели машинного обучения. Структуры данных DataFrame используются в языках программирования R и Python. А благодаря популярности и распространенности языка Python, используется и во многих технических фрейворках для работы с большими данными (например, Apache Spark).

В данном учебном пособии даны основные представления о решении задач анализа данных с использованием языка программирования Python. Основной упор сделал на решение задач исследовательского анализа данных, вовлекающего различные инструменты представления данных: вектора Numpy – для числовых данных, DataFrame пакета Pandas – для представления «кадра данных» из предметного набора данных, а также инструменты визуализации на основе фреймворка Matplotlib и пакета Seaborn. Раскрываются основные способы сбора и анализа информации, получаемой в процессе построения моделей машинного обучения с целью оценки эффективности и сравнения различных подходов.

Учебное пособие построено следующим образом. В первой главе рассматриваются основные понятия и термины в области анализа данных. Вторая глава посвящена описанию языка Python, применяемых средах разработки IDE, пакетах и фрейворках, необходимых для решения задач анализа данных. В главе 3 изучаются основы компьютерной визуализации и анализа данных на Python, рассматриваются способы и инструменты представления данных в табличном и графическом виде. Глава посвящена исследовательскому (разведочному) анализу данных EDA, который является основой и важнейшим этапом построения моделей машинного обучения и искусственного интеллекта. Каждая глава завершается контрольными вопросами по рассмотренной теме для оценки закрепления пройденного.

Глава 1. Проблема анализа данных, интеллектуальный анализ данных

1.1. Задача анализа данных

Анализ данных – область математики и информатики, занимающаяся построением и исследованием наиболее общих математических методов и вычислительных алгоритмов извлечения знаний из экспериментальных (в широком смысле) данных. Анализ данных – это процесс исследования, фильтрации, преобразования и моделирования данных с целью извлечения полезной информации и принятия решений [5].

В статистическом смысле разделяют анализ данных на описательную статистику, исследовательский анализ данных и проверку статистических гипотез:

– Исследовательский анализ данных занимается открытием новых характеристик данных.

– Прогнозный анализ фокусируется на применении статистических или структурных моделей для предсказания или классификации новых данных.

– Проверка статистических гипотез направлена на подтверждении или опровержении существующих гипотез о свойствах данных.

Отдельно выделяют интеллектуальный анализ данных (data mining, knowledge discovery in databases).

Интеллектуальный анализ данных – это особый метод анализа данных, который фокусируется на моделировании и открытии данных, а не на их описании. Первоначально задача ставится следующим образом: имеется достаточно крупная база данных и предполагается, что в базе данных находятся некие «скрытые знания», которые обладают следующими свойствами [6]:

– ранее неизвестные – то есть такие знания, которые должны быть новыми (а не подтверждающими какие-то ранее полученные сведения);

– нетривиальные – то есть такие, которые нельзя просто обнаружить (например, при непосредственном визуальном анализе данных или при вычислении простых статистических характеристик);

– практически полезные – то есть такие знания, которые представляют ценность для исследователя или потребителя;

– доступные для интерпретации – то есть такие знания, которые легко представить в наглядной для пользователя форме и легко объяснить в терминах предметной области.

В свою очередь, задачи, решаемые методами интеллектуального анализа данных, принято разделять на описательные (descriptive) и предсказательные (predictive).

К описательным задачам относятся такие задачи как:

– поиск ассоциативных правил или паттернов (образов);

– группировка объектов, кластерный анализ;

– построение регрессионной модели.

К предсказательным задачам относятся:

– классификация объектов (для заранее заданных классов);
– регрессионный анализ, анализ временных рядов.

Инструменты интеллектуального анализа данных базируются на следующих методах и используют из в различных комбинациях:

– алгебра, математический анализ;
– теория вероятностей и математическая статистика;
– методы и модели машинного обучения и искусственного интеллекта;
– разведочный анализ данных (как самостоятельнее направление, которое обычно применяется на ранних этапах анализа данных или независимо).

В рамках данного учебного пособия мы сделаем основной упор на разведочный анализ данных.

Понятие разведочного анализа данных (exploratory data analysis, EDA) введено математиком Джоном Тьюки, который сформулировал цели анализа следующим образом [7]:

– максимальное «проникновение» в данные,
– выявление основных структур,
– выбор наиболее важных переменных,
– обнаружение отклонений и аномалий,
– проверка основных гипотез,
– разработка начальных моделей.

Визуализация данных является одним из важнейших инструментов анализа данных, обеспечивающим следующие преимущества [8]:

– Больше привлеченной аудитории, так как большинство людей лучше воспринимает и запоминает зрительную информацию.
– Высокая вовлеченность читателя, так как красивый яркий график с понятным посылом привлечет к себе внимание.
– Скорость принятия решений. На практике проще и быстрее сделать вывод, глядя на график, где один из столбцов или одна из точек находится намного выше всех остальных, чем пролистать несколько страниц статистики, представленной таблицами.
– Лучшее понимание данных. В организациях, «грамотные» отчеты понятны не только технических специалистам и аналитикам и Data, но и позволяют каждому сотруднику принимать решения в своей зоне ответственности.

Необходимость использования визуализации в анализе данных связанна с физиологией восприятия информации человеком. Например, на рис. 1 показаны два варианта представления одной новости – какой способ позволят быстрее и проще понять, о чем идет речь?

Рис. 1. Проблема восприятия информации

Или попробуем посчитать количество цифр 5 в следующих двух строках цифр (рис. 2).

Строка 1:

74532727462374523457342456209872398283475623587235823537457563 24

Строка 2:

74**5**32727462374**5**234**5**73424**5**6209872398283475**5**623**5**8723**5**823**5**374**5**7**5**6324

Рис. 2. Акцентирование внимания на данных

Физиология человека такова, что он может легко различать длину линии, форму, ориентацию, расстояние и цвет (оттенок) без значительных усилий по обработке – такие свойства называются «атрибутами предварительного внимания». Визуальная обработка человека эффективна при обнаружении изменений и сравнении величин, размеров, форм и вариаций яркости. Когда свойства символьных данных сопоставляются с визуальными свойствами, люди могут эффективно просматривать большие объемы данных.

Так как все визуализации данных создаются для использования человеком. Знание человеческого восприятия и познания необходимо при разработке интуитивно понятных визуализаций и, следовательно, визуализация может стать средством исследования данных.

Визуализация данных является междисциплинарной областью, которая имеет дело с графическим представлением данных. Визуализация данных – это представление данных в виде, который обеспечивает наиболее эффективную работу человека по их изучению. Визуализация данных находит широкое

применение в научных и статистических исследованиях. Это особенно эффективный способ передачи данных, когда данных много [9].

Визуализация данных связана с визуализацией информации, инфографикой, визуализацией научных данных, разведочным анализом данных и статистической графикой. С академической точки зрения это представление можно рассматривать как отображение между исходными данными (обычно числовыми) и графическими элементами (например, линиями или точками на диаграмме). Отображение определяет, как атрибуты этих элементов изменяются в зависимости от данных. Визуализация данных берет свое начало в области статистики и поэтому обычно считается ветвью описательной статистики.

Для четкой и эффективной передачи информации при визуализации данных используются статистические графики, математические графики, информационные графики и другие инструменты. Числовые данные могут быть закодированы с использованием точек, линий или более сложных объектов для визуальной передачи количественного сообщения. Эффективная визуализация помогает пользователям анализировать и рассуждать о данных и доказательствах. Это делает сложные данные более доступными, понятными и удобными в использовании, а также более простыми

Поскольку для эффективной визуализации требуются как навыки дизайнера (художника), так и статистические и вычислительные навыки, некоторые авторы утверждают, что визуализация данных – это и искусство, и наука.

Развитие методов визуализации связано с работами, в частности, Фернанда Бертини Вьегаса, бразильского ученого и дизайнера, чья работа сосредоточена на социальных и художественных аспектах визуализации информации, и Мартина Ваттенберга, профессором компьютерных наук в Школе инженерии и прикладных наук Гарвардского университета, а также художником, известным своей работой с визуализацией данных. Виегас и Ваттенберг предположили, что идеальная визуализация должна не только четко передавать информацию, но и стимулировать участие и внимание зрителя.

Эдвард Рольф Тафте – американский статистик и почетный профессор политологии, статистики и информатики в Йельском университете, известный своими работами по информационному дизайну и как пионер в области визуализации данных в своей книге 1983 года «Визуальное отображение количественной информации» определяет «графическое отображение» и принципы эффективного графического отображения в следующем отрывке: «Превосходство в статистической графике состоит из сложных идей, переданных с ясностью, точностью и эффективностью» [10]. Согласно идеям Тафте, графическое отображение должно:

– показать данные,
– заставить зрителя задуматься о содержании, а не о методологии, графическом дизайне, технологии графического производства или о чем-то другом,
– избегать искажения того, о чем должны говорить данные,
– представить много чисел в маленьком пространстве,
– сделать большие наборы данных согласованными, связанными,

– побуждать зрительно сравнивать разные фрагменты данных,

– раскрывать данные на нескольких уровнях детализации, от общего обзора до детализированной структуры,

– служить достаточно ясной цели: описание, исследование, составление таблиц или украшение,

– быть тесно интегрировано со статистическими и словесными описаниями набора данных.

Наибольшее развитие визуализация получила в конце XX века, благодаря появлению работы Джона Тьюки, посвященной разведочному анализу данных, и с развитием компьютерных технологий визуализации и анализа данных [7].

1.2. Машинное обучение и искусственный интеллект в анализе данных

В последние годы машинное обучение (ML, machine learning) превратилось в большой бизнес – фирмы используют его, чтобы заработать денег, прикладные исследования бурно развиваются как в индустриальной, так и в академической среде. Прикладное машинное обучение совмещает в себе равные доли математических принципов и полученных эмпирическим путем приемов. Машинное обучение – это одновременно и наука, и искусство. Современное программное обеспечение тесно интегрировано с машинным обучением, а архитектура современных ЭВМ создается учетом применения машинного обучения в прикладном программном обеспечении [11].

1.2.1. Понятие интеллектуального анализа данных

В 1959 г. специалист по вычислительной технике из компании IBM Артур Самуэль написал компьютерную программу для игры в шашки. Каждому положению на доске присваивался некий вес, базирующийся на вероятности выигрыша. Вероятность определялась по формуле, в которой учитывались такие факторы, как количество шашек на каждой стороне и количество дамок. Сыграв с программой тысячу партий, он использовал их результаты для уточнения позиционных весов. К середине 1970-х программа достигла уровня хорошо подготовленного непрофессионального игрока. Самуэль написал компьютерную программу, которая могла по мере накопления опыта улучшать собственные результаты. Программа «училась» – так зародилось машинное обучение.

Первое приложение машинного обучения, которое действительно получило широкое распространение, увидело свет в 1990-х годах, это был фильтр спама. Машинное обучение десятилетиями используется в ряде специализированных приложений, таких как программы для оптического распознавания знаков (Optical Character Recognition, OCR). В 2006 году Джеффри Хинтон с соавторами опубликовали статью, в которой было показано, как обучать глубокую нейронную сеть, способную распознавать рукописные цифры с передовой точностью (более 98%). Они назвали такой прием «глубоким обучением» («Deep Learning»).

Обратимся к терминологии. Термин «обучение» ставит перед нами вопросы: где машинное обучение начинается и где заканчивается и что в точности означает для машины изучить что-то?

Например, у людей мы различаем механическое заучивание и интеллектуальное осмысление. Главная цель обучения – обобщить опыт на новые ситуации, которые не встречались в процессе обучения.

Процесс получения знаний человеком превосходит своей сложностью самые совершенные алгоритмы машинного обучения, но у компьютера есть преимущество в виде большей емкости для запоминания, извлечения и обработки данных, а также в виде скорости обработки. Аналогия между человеческим и машинным обучением закономерно заставляет вспомнить такое явление, как искусственный интеллект (artificial intelligence, AI). Считается, что машинное обучение – это одна из форм искусственного интеллекта, так как искусственный интеллект представляет собой более широкую область.

В свою очередь, терминология искусственного интеллекта также подвергается критике. Чтобы создать что-то искусственное, надо понимать исходное естественное и выделить часть естественного объекта, которую будем искусственно воссоздавать. Для этого необходимо ответить на следующие вопросы:

– Что такое естественный интеллект?
– Как работает естественный интеллект человека?
– Какую часть естественного интеллекта надо воссоздать

К сожалению, ответы на эти вопросы до сих пор не получены в полном объеме и, вероятно, не будут получены, так как естественный интеллект мы наблюдаем как «черный ящик», т.е. мы можем наблюдать результаты деятельности, которые воспринимаем как интеллектуальный продукт, но не можем наблюдать процесс их получения (например, концепция «философского зомби» и «китайской комнаты» в философии).

Поэтому более корректно говорить об реализуемых на компьютерах вычислительных интеллектуальных информационных технологиях.

Приведем несколько строгих определений машинного обучения.

Машинное обучение представляет собой науку (и искусство) программирования компьютеров для того, чтобы они могли обучаться на основе данных (более широко – на основе прецендентов).

Артур Самуэль (1959 год): «Машинное обучение – это научная дисциплина, которая наделяет компьютеры способностью учиться, не будучи явно запрограммированными».

Том Митчелл (1997 год): «Говорят, что компьютерная программа обучается на основе опыта E по отношению к некоторой задаче T и некоторой оценке производительности P, если ее производительность на T, измеренная посредством P, улучшается с опытом E».

Машинное обучение тесно связано с прикладной математической статистикой. Однако статистика основывается на выдвижении и проверки гипотез, тогда как машинное обучение обычно пытается найти новые, неочевидные свойства

данных. Применение методов машинного обучения для исследования больших объемов данных может помочь в обнаружении паттернов, которые не были замечены сразу. Это и называется интеллектуальным или глубинным анализом данных, т.е. поиском в сырых данных (raw data) раннее неизвестных, нетривиальных, практически полезных, доступных интерпретации знаний (закономерностей), необходимых для принятия решений.

1.2.2. Типы систем машинного обучения

Существует так много разных типов систем машинного обучения, что их удобно сгруппировать в категории на основе следующих признаков [12]:

– обучаются ли они с человеческим контролем (обучение с учителем, обучение без учителя, частичное обучение (semisupervised learning) и обучение с подкреплением (reinforcement Learning));

– могут ли они обучаться постепенно «на лету» (динамическое или пакетное обучение);

– работают ли они, сравнивая новые точки данных с известными точками данных, или взамен обнаруживают паттерны в обучающих данных и строят прогнозирующую модель (обучение на основе образцов или на основе моделей).

При обучении с учителем обучающие данные, поставляемые алгоритму, включают желательные решения, называемые метками (label). Типичной задачей обучения с учителем является классификация. Другая задача – прогнозирование целевого числового значения, располагая набором характеристик или признаков, которые называются предикторами (predictor), такую задачу именуют регрессией.

При обучении без учителя обучающие данные не помечены, т.е. нельзя сравнить совпал ли ответ системы с желаемым значением. Наиболее важные задачи (модели) обучения без учителя: кластеризация, визуализация и понижение размерности, обучение ассоциативным правилам, обнаружение аномалий.

Некоторые алгоритмы способны работать с частично помеченными обучающими данными, в состав которых обычно входит много непомеченных данных и немного помеченных. Процесс называется частичным обучением. Большинство алгоритмов частичного обучения являются комбинациями алгоритмов обучения без учителя и с учителем.

В некоторых случаях, обучающаяся система, которая в данном контексте называется агентом, может наблюдать за средой, выбирать и выполнять действия, получая в ответ награды (или штрафы в форме отрицательных наград). Система должна самостоятельно узнать, в чем заключается наилучшая стратегия, называемая политикой, чтобы со временем получать наибольшую награду. Политика определяет, какое действие агент обязан выбирать, когда он находится в текущей ситуации (примеры, автономные роботы, программа AlphaGo от DeepMind, Google переводчик и др.). Подобные задачи называются обучением с подкреплением.

При работе с быстро меняющимися данными или большими наборами данными часто обучение проводится не со всем набором данных, а порциями данных, так различают пакетное и динамическое обучение.

При пакетном обучении система должна учиться с применением всех доступных данных. В общем случае процесс будет требовать много времени и вычислительных ресурсов, поэтому обычно он проходит автономно. Сначала система обучается, а затем функционирует без дальнейшего обучения. Также называется автономным обучением (offline learning). Если необходимо, чтобы система пакетного обучения узнала о новых данных, тогда понадобится обучить новую версию системы с нуля на полном наборе данных, включая новые и старые.

При динамическом обучении система обучается постепенно за счет последовательного предоставления ей образцов данных либо по отдельности, либо небольшими группами, называемыми мини-пакетами. Каждый шаг обучения является быстрым и недорогим, так что система может узнавать о новых данных на лету по мере их поступления. Динамическое обучение подходит для систем, которые получают данные в виде непрерывного потока и должны адаптироваться к изменениям быстро или самостоятельно. Динамическое обучение будет хорошим вариантом в случае ограниченных вычислительных ресурсов. Например, алгоритмы динамического обучения также могут применяться для обучения систем на гигантских наборах данных, которые не умещаются в основную память одной машины.

Различают обучение на основе образцов и на основе моделей. Наиболее тривиальной формой обучения является просто заучивание на память. Это называется обучением на основе образцов: система учит примеры на память и затем обобщает их на новые примеры с применением измерения сходства (так работает, например, метод k ближайших соседей). Другой метод обобщения набора примеров предполагает построение модели этих примеров и ее использование для выработки прогнозов. Это называется обучением на основе моделей. Модели способны к обобщению, т.е. описанию зависимостей, определяющих данные, которые можно переносить на новые примеры (прогнозирование появления новых данных).

В общем случае, к методам машинного обучения стоит прибегать в следующих случаях:

– для задач, существующие решения которых требуют большого объема ручной настройки или длинных списков правил. В таких задачах один алгоритм машинного обучения часто способен упростить код и выполняться лучше;

– для сложных задач, для которых традиционный подход вообще не предлагает хороших решений, а машинное обучение может найти решение;

– для изменяющихся сред, где система машинного обучения способна адаптироваться к новым данным;

– для получения сведений о сложных задачах и крупных объемах данных и т.д.

При этом системы машинного обучения обычно сталкиваются со следующими проблемами:

– Недостаточный размер обучающих данных.
– Нерепрезентативные обучающие данные.
– Данные плохого качества (ошибки, выбросы и шумы).
– Несущественные признаки.
– Переобучение на обучающих данных.
– Недообучение на обучающих данных.
– Нет модели, которая априори гарантировала бы лучшую работу. Единственный способ достоверно знать, какая модель лучше, предусматривает оценку всех моделей.

1.2.3. Этапы анализа данных в машинном обучении

Процедуру машинного обучения можно разбить на пять стадий:

1. подготовка данных,
2. построение модели,
3. оценка качества модели,
4. оптимизация модели
5. и прогноз по модели на новых данных.

Этапы следуют друг за другом в определенном порядке, но в реальных приложениях требуется многократное повторение каждого этапа [6, 13].

Машинное обучение обычно использует представление данных в виде таблицы, даже если изначально они имеют другую форму. Данные распределены по строкам и столбцам, причем каждая строка соответствует изучаемому экземпляру (instance), а столбец – значению этого экземпляра.

Реальные данные могут иметь ряд проблем:

– На этапе сбора данных измерить какое-то значение не представляется возможным и нельзя и вернутся назад, чтобы отыскать недостающий фрагмент информации. В подобных случаях некоторые ячейки таблицы останутся незаполненными, что усложнит как построение модели, так и последующее прогнозирование. «Пропуски» в данных необходимо помечать особым типом (аналог NULL в БД). Экземпляр с пропусками или игнорируется, или пропуск нужно заполнить каким-то прогнозным значением (требуется иметь модель).

– Иногда сбор данных осуществляется вручную с ошибками. Технические средства сбора также имеют погрешности или могут давать сбои. В результате часть сведений оказывается некорректной (искаженной, зашумленной). Многие алгоритмы МО чувствительны к подобным шумам в данных, поэтому необходимо уметь обнаруживать наличие шумов и искажений и устранять их.

– Для построения работающей модели можно и нужно использовать эффективное множество признаков. Некоторые алгоритмы в достаточной степени невосприимчивы к неинформативным признакам, другие дают более точные предсказания, когда неинформативные признаки убираются из рассмотрения.

Иногда ценные сведения могут быть извлечены и из неинформативных признаков, путем их преобразования. Такой способ совершенствования данных называется извлечением признаков (feature extraction). Задача подготовки признаков к машинному обучению в целом называется инжиниринг данных (feature engineering)

Системы машинного обучения не используются, пока не будет проверена их производительность. Для этого часть набора данных убирается из обучающего набора и рассматривается как тестовый набор, т.е. целевая переменная тестового набора неизвестна для модели машинного обучения. Чтобы не «подстроиться» под тестовые данные, часто выделяют дополнительный валидационный набор, на котором проверяется итоговая система машинного обучения после обучения и тестирования.

Если что-то в процессе машинного обучения можно варьировать, то возникает естественное желание найти такой вариант, при котором полученная модель машинного обучения даст наилучший результат (в смысле некоторого критерия качества). Увеличение точности модели достигается разными способами: редактирование параметров модели, выбор подмножества признаков, предварительная обработка данных и др.:

– Редактирование параметров модели. Каждый алгоритм машинного обучения обладает набором параметров, оптимальные значения которых зависят от типа и структуры данных. Поиск оптимальных параметров называют «настройка гиперпараметров» модели. Данная задача рассматривается как самостоятельная задача оптимизации или как часть процесса машинного обучения (самонастраивающиеся, адаптивные модели). Простейший подход – это поиск «по решетке», т.е. перебор некоторых вариантов настроек гиперпараметров.

– Выбор подмножества признаков. Зачастую задачи машинного обучения включают множество признаков, и вносимые ими помехи порой мешают алгоритму обнаружить верную закономерность, даже если сами признаки являются информативными. Снижение размерности также необходимо из экономических соображений (требования к программному обеспечению и оборудованию). Задача выделения признаков может быть рассмотрена как самостоятельная задача машинного обучения (подходы «фильтр», когда работа с признаками ведется без учета применяемого алгоритма, и «обертка», когда признаки выбираются для конкретного алгоритма машинного обучения).

– Предварительная обработка данных. Многие наборы данных без проблем «скармливаются» многим алгоритмам машинного обучения. Однако реальные данные часто нуждаются в очистке и обработке. Этот процесс называют выпасом данных (data munging или data wrangling). Например, разные имена или написания могут означать один объект. Разные числа могут быть одним значением, но в разных шкалах или форматах. Подобная обработка почти всегда требует «ручной» работы, понимания и переосмысления данных.

Традиционные модели машинного обучения статические и перестраиваются редко. Но во многих случаях крайне желательно, чтобы модель постепенно

совершенствовалась и адаптировалась к изменениям в данных. Подобные алгоритмы называют алгоритмы с динамическим обучением (online learning).

Еще одна особенность современных наборов данных – это постоянно увеличивающиеся в размерах данные. Не все методы машинного обучения могут масштабироваться по мере роста данных. Многие масштабируемые методы также имеют свой предел (проблема «больших» данных (big data)).

Учет вышеобозначенных особенностей построения и применения методов машинного обучения привели к развитию методологии MLOps [14].

В программной инженерии, DevOps (development operations) - это методология автоматизации технологических процессов сборки, настройки и развертывания программного обеспечения. Методология предполагает активное взаимодействие специалистов по разработке со специалистами по информационно-технологическому обслуживанию и взаимную интеграцию их технологических процессов друг в друга. Методология предназначена для эффективной организации создания и обновления программных продуктов и услуг. Чтобы эффективно использовать DevOps, прикладные программы должны соответствовать набору архитектурно значимых требований, таких как: возможность развертывания, изменяемость, тестируемость и возможности мониторинга.

MLOps – это набор методов, подобно DevOps, направленных на надежное и эффективное внедрение и поддержку моделей машинного обучения в производстве. Слово представляет собой соединение «машинного обучения» и практики непрерывной разработки DevOps в области программного обеспечения. Модели машинного обучения тестируются и разрабатываются в изолированных экспериментальных системах. Когда алгоритм готов к запуску, специалисты переносят алгоритм в производственные системы.

В MLOps выделяют следующие этапы жизненного цикла системы машинного обучения:

1. сбор данных,
2. обработка данных,
3. разработка функций,
4. маркировка данных,
5. проектирование моделей,
6. обучение и оптимизация моделей,
7. развертывание конечных точек и
8. мониторинг конечных точек.

Каждый шаг в жизненном цикле машинного обучения встроен в свою собственную систему, но требует взаимосвязи в проекте в целом.

1.2.4. Некорректно поставленные задачи анализа данных, кросс-валидация

Поскольку машинное обучение строит прогнозные модели на основе данных, чтобы выяснить общую закономерность получения целевых значений, естественный способ оценки качества модели – это сравнение прогноза, полученного

моделью и истинного значения, которое содержится в обучающем наборе в базе данных [15].

Рассмотрим пример из задачи прогнозирования цены на недвижимость (рис. 3). Модель слева описывает имеющуюся информацию о квартирах с 2, 3, 4 и 6 спальнями, но может спрогнозировать цену для квартиры с 5 спальнями. Модель справа идеально описывает цены известных квартир (ошибка равна нулю), но н способна спрогнозировать цену на квартиру, которой нет в базе данных.

Рис. 3. Обучение или обобщение?

Попытка повторить обучающие данные является некорректной постановкой задачи анализа данных и машинного обучения. Так как цель построения модели – прогноз на новых данных, которые не встречались при обучении. Такую способность называют обобщением (generalization) в противовес простому обучению (learning). Простое запоминание обучающих примеров часто называют «переобучением».

Один из способов решения данной проблемы – это оценка качества обучения на тестовых данных, которые не были использованы в процессе построения модели. Подход называется – кросс-валидация.

Кросс-валидация – это процедура эмпирического оценивания обобщающей способности алгоритмов. С помощью кросс-валидации эмулируется наличие тестовой выборки, которая не участвует в обучении, но для которой известны правильные ответы.

Есть несколько способов выполнить кросс-валидацию.

Валидация на отложенных данных, Hold-Out Validation (рис. 4). Метод Hold-out применяется в случаях больших наборов данных, так как требует меньше вычислительных мощностей по сравнению с другими методами кросс-валидации. Недостатком метода является то, что оценка существенно зависит от разбиения, тогда как желательно, чтобы она характеризовала только алгоритм обучения.

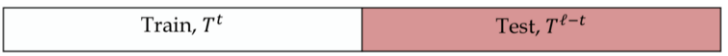


Рис. 4. Валидация на отложенных данных

Полная кросс-валидация, Complete cross-validation. Выборка разбивается всеми возможными способами на две части: на обучающую и тестовую. Самый затратный метод на практике, но позволяет получить полную оценку качества модели машинного обучения.

k-кратная (k-fold) кросс-валидация (рис. 5). Данные развиваются на k частей, обучение и тестирование проводится k разными способами, вычисляются средние оценки обучения и обобщения.

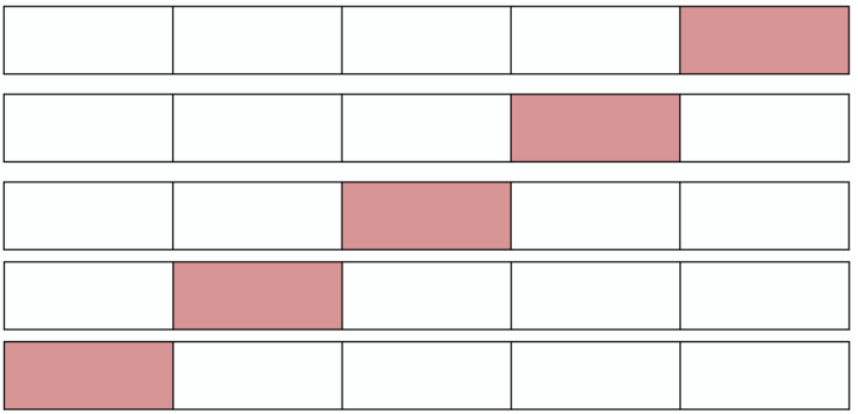


Рис. 5. k-кратная кросс-валидация

Кросс-валидация по отдельным объектам, Leave-One-Out (рис. 6). В русскоязычной литературе часто называется «скользящий экзамен». Выборка разбивается на $l-1$ и 1 объект l раз.

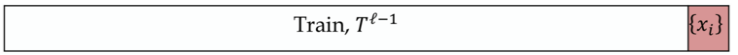


Рис. 6. Скользящий экзамен

Случайные разбиения, Random subsampling (рис. 7). Выборка разбивается в случайной пропорции. Процедура повторяется несколько раз.

Рис. 7. Случайные разбиения

Другие способы решения проблемы некорректной постановки задачи:

– Регуляризация для задачи регрессии – это метод добавления некоторых дополнительных ограничений к условию с целью решить некорректно поставленную задачу или предотвратить переобучение. Эта информация часто имеет

вид штрафа за сложность модели в предположении, что более простые модели не склонны к переобучению.

– Ранняя остановка обучения. В этом подходе на каждом шаге процедуры обучения модели осуществляется контроль на тестовой выборке. На первых шагах обе ошибки обучения и тестирования будут уменьшаться. Далее определяется шаг, на котором ошибка обучения продолжит снижаться, а ошибка тестирования начнет расти. Модель этого шага используется как итоговая.

– Разработка методов, которые не склонны к переобучению.

1.2.5. Методы машинного обучения и искусственного интеллекта для анализа данных

Существует множество подходов для решения задач машинного обучения и построения систем искусственного интеллекта, основанного на анализе данных. Более того, современные подходы все чаще проектируют новые методы непосредственно под имеющийся набор данных (подход называется гиперэвристики [16]).

Перечислим некоторые известные подходы (без детального их рассмотрения).

Для задачи кластеризации наиболее часто применяют:

– Методы визуального анализа.
– Методы на основе k-средних.
– Агломеративные методы.
– Иерархическая кластеризация.
– Дивизимные методы.
– EM-алгоритм (expectation maximization).
– Искусственные нейронные сети обучения без учителя.

Для задачи классификации наиболее часто применяют:

– Классификационные правила и деревья решений.
– Разделяющие поверхности, включая:
    - Линейный разделитель.
    - Нелинейный разделитель.
    - Метод опорных векторов.
    - Нейронные сети.
– K ближайших соседей.
– Вероятностные модели, включая:
    - Наивный Байесовский классификатор.
    - Нейронные сети с функцией softmax на выходном слое.
– Нечеткие модели.

Для задачи регрессии наиболее часто применяют:

– Линейная регрессия.
– Непараметрическая регрессия (ядерные оценки).
– Нейронные сети.

– Метод группового учета аргументов, МГУА.
– Генетическое программирование.
– Регрессия опорных векторов.
– Регрессия случайным лесом.

Отдельно нужно отметить методы так называемой «большой тройки искусственного интеллекта»: искусственные нейронные сети, методы на основе нечеткой логики и методы эволюционных вычислений [17, 18, 19].

Явление естественного интеллекта очевидным образом связывают с работой нервной системы человека, точнее – головного мозга, поэтому одно из наиболее ранних и наиболее изученных и разработанных направлений сегодня – это методы на основе искусственных нейронных сетей (или просто нейронных сетей). Идея подхода заключается в моделировании функционирования клеток головного мозга, нейронов. Полученные на основе подобных моделей математические структуры являются универсальными классификаторами и аппроксиматорами.

Структура биологического нейрона показана на рис. 8, а его математическая я модель на рис. 9.

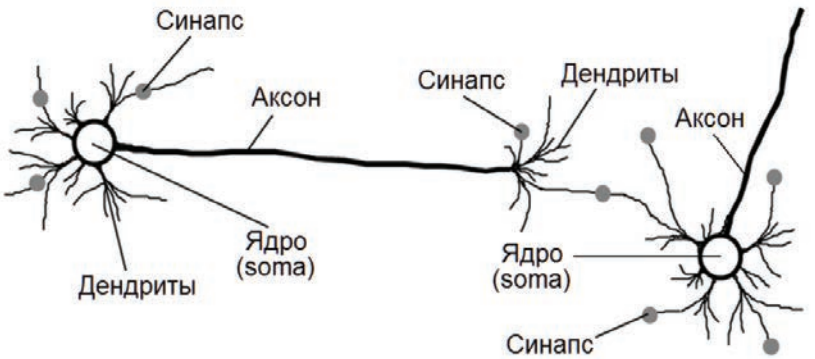


Рис. 8. Биологический нейрон

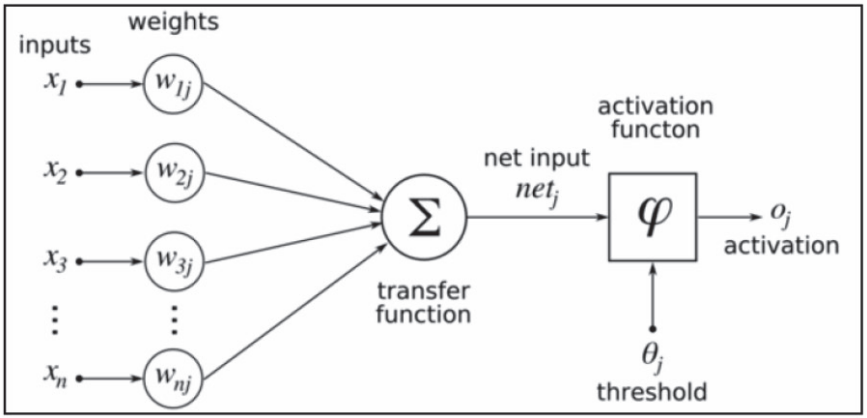


Рис. 9. Искусственный нейрон

Нервная клетка – это черный ящик, у которого есть дендриты, входы, по которым поступают сигналы (отрицательные ионы) в клетку или ее ядро. Если внутри накапливается большой отрицательный заряд, клетка возбуждается и генерирует импульс, который по аксону к дендритам следующих клеток. Место соединения аксона нейрона с дендритом называется синапсом. Заряды аддитивны, т.е. они накапливаются в клетках. Отсюда клетка – это элементарный классификатор, принимающий решение, возбудиться или нет.

Объединение нейронов с более сложные комбинации (сеть) позволяет сложным образом обрабатывать сложную информацию, поступающую на вход нейронной сети.

Выбор числа нейронов и способов их объединения в сеть называют настройкой архитектуры сети, а подбор весовых коэффициентов для корректной реакции на входные данные – машинным обучением или просто обучением нейронной сети.

Особенностью нейронных сетей является то, что они могут воспроизвести зависимость практически любой сложности при наличии достаточно большого числа нейронов. Поэтому нейронные сети весьма эффективны для решения многих задач анализа данных и машинного обучения.

Для многих специальных задач таких как компьютерное зрение, обработка естественного языка, построение больших языковых моделей были предложены так называемые глубокие (глубинные) сети (рис. 10), состоящие из большого числа слоев нейронов, часть которых отвечает за извлечение информативных признаков из обучающих данных, другая – за традиционное машинное обучение (например, классификацию). Несмотря на размеры, в такой нейронной сети число настраиваемых параметров существенно меньше, чем у обычных нейронных сетей, если бы их применяли для подобных задач.

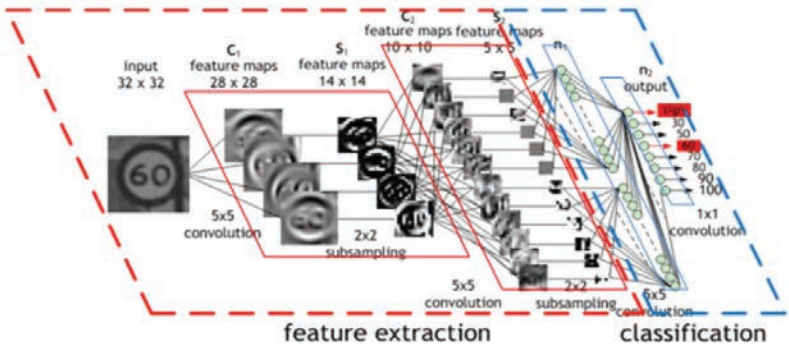


Рис. 10. Глубокая нейронная сеть

В то время как нейронные сети моделируют физиологические особенности мозга, методы на основе нечеткой логики пытаются смоделировать то, как человек оперирует понятиями.

Профессор университета Беркли Лотфи Заде предположил: «По мере того, как сложность возрастает, точные утверждения теряют значимость, а значимые утверждения теряют точность».

Знания человека можно представить в виде продукций типа «Если условие, то вывод». При этом люди описывают условия и вывод с помощью прилагательных: «Если Погода=Холодно и Влажность=Высокая, то Одежда=Теплая». При этом таких правил может быть много, т.е. имеется база правил. В базе правил можно сохранить знания, умения, опыт человека. Как машине ей пользоваться (рис. 11)?

Рис. 11. Работа системы на нечеткой логике

Заде предложил математический аппарат для работы с понятиями, которые мы выражаем расплывчатыми понятиями. Нечеткая логика (fuzzy logic) – это раздел математики, являющийся обобщением классической логики и теории множеств, базирующийся на понятии нечеткого множества, впервые введенного Лютфи Заде в 1965 году как объекта с функцией принадлежности элемента к множеству, принимающей любые значения в интервале [0, 1], а не только 0 или 1 как для обычных множеств. Нечеткая логика применяется в следующих областях: «нечеткая математика» (множества, графы, логика, отношения и т.д.), нечеткое принятие решений, нечеткое управление, применение в системах искусственного интеллекта.

Изучая природные явления, исследователи заметили, что практически все существующие природные явления от физических процессов, до биологических и социальных являются устойчивыми и способны наилучшим образом адаптироваться к изменяющейся внешней среде. На основе этих наблюдений были предложены первые эволюционные алгоритмы, имитирующие процессы биологической эволюции, в которой каждый биологический вид целенаправленно развивается и изменяется для того, чтобы наилучшим образом приспособиться к окружающей среде. Эволюционный алгоритм использует следующие метафоры:

– Эволюция – процесс решения задачи.

– Индивид – потенциальное решение задачи (популяция – множество индивидов).

– Пригодность индивида (приспособленность к внешней среде) – оценка качества решения задачи.

– Окружающая среда – решаемая задача.

Эволюционные алгоритмы реализуют несколько важных идей:

– Популяционный алгоритм – множество параллельно проектируемых решений повышают надежность.

– Прямой поиск – мы лишь сравниваем решения между собой, не привлекая дополнительную информацию о свойствах задачи (если она не доступна).

– Селективное давление – выживают и оставляют потомство наиболее приспособленные (модель эволюционный процесс).

– Случайные мутации – модель изменчивости для поддержки разнообразия решений и предотвращения стагнации.

В настоящее время предложено множество алгоритмов, которые имитируют различные природные процессы, направление в целом принято называть эволюционными вычислениями. Все эволюционные алгоритмы являются методами машинного обучения, а точнее, методами обучения с подкреплением. Основная область применения – сложные задачи оптимизации, поэтому эволюционные вычисления часто используются для проектирования и настройки других алгоритмов машинного обучения.

В последние годы наблюдается разработка и применение эффективных гибридных подходов, в которых используется более одной технологии

искусственного интеллекта. Наилучшие результаты достигнуты в гибридах типа нейро-эволюция, нейро-нечеткие системы, эволюционные алгоритмы для проектирования систем на нечеткой логике.

1.3. Data Science и Big Data

Наука о данных (data science) – раздел информатики, изучающий проблемы анализа, обработки и представления данных в цифровой форме. Объединяет методы по обработке данных в условиях больших объемов и высокого уровня параллелизма, статистические методы, методы интеллектуального анализа данных и приложения искусственного интеллекта для работы с данными, а также методы проектирования и разработки баз данных [20].

Основная практическая цель профессиональной деятельности в науке о данных – это обнаружение закономерностей в данных, извлечение знаний из данных в обобщенной форме. Для объяснения навыков, необходимых для деятельности в этой области, часто используется диаграмма Венна (рис. 12), на которой навыки, требуемые специалисту, отражены на пересечении сфер общепредметного опыта (substantive expertise), практического опыта в информационных технологиях (hacking skills) и знания математической статистики.

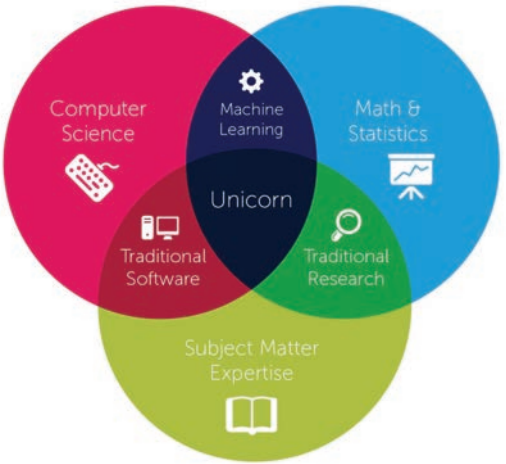


Рис. 12. Навыки в Data Science

Как видно из диаграммы выше, наука о данных отличается от традиционной математической статистики или машинного обучения тем, что активно вовлекает практические знания предметной области. Развитие науки о данных как самостоятельной дисциплины произошло во многом благодаря популяризации концепции «больших данных».

Большие данные (big data) – обозначение структурированных и неструктурированных данных огромных объемов и значительного многообразия, эффективно обрабатываемых горизонтально масштабируемыми программными инструментами, появившимися в конце 2000-х годов и альтернативных традиционным системам управления базами данных и решениям класса Business Intelligence [21].

С точки зрения информационных технологий, в качестве базового принципа обработки больших данных отмечают горизонтальную масштабируемость, обеспечивающую обработку данных, распределенных на сотни и тысячи вычислительных узлов, без деградации производительности. Для это го используют такие инструменты как MapReduce и Hadoop [22, 23].

MapReduce – это модель распределенных вычислений, представленная компанией Google, используемая для параллельных вычислений над очень большими наборами данных в компьютерных кластерах, а также фреймворк для вычисления некоторых наборов распределенных задач.

Hadoop является проектом фонда Apache Software Foundation, и представляет собой свободно распространяемый набор утилит, библиотек и фреймворк для разработки и выполнения распределенных программ, работающих на кластерах из сотен и тысяч узлов. Для работы с Hadoop аналитики данных часто применяют Apache Spark, фреймворк с открытым исходным кодом для реализации распределенной обработки неструктурированных и слабоструктурированных данных. В частности, Spark имеет API на языке Python – pyspark, что делает порог вхождения в эту область проще, так как в настоящее время язык Python является наиболее популярным для решения задач анализа данных.

Контрольные вопросы

1. Назовите отличия исследовательского анализа данных от прогнозного.
2. Сравните интеллектуальный анализ данных и метод проверки статистических гипотез. В чем принципиальные отличия?
3. Назовите описательные задачи анализа данных.
4. Назовите описательные задачи анализа данных.
5. Какие преимущества дает визуализация для задач анализа данных?
6. Что такое «атрибуты предварительного внимания»?
7. Почему машинное обучение является одновременно и наукой, и искусством?
8. Какая связь между машинным обучением и искусственным интеллектом?
9. Дайте определение машинного обучения по Самуэлю.
10. Опишите суть обучения с подкреплением.
11. В чем отличие обучения с учителем и без учителя?
12. Сравните 2 типа машинного обучения: пакетное и динамическое.
13. Назовите типичные проблемы данных в машинном обучении.
14. В чем суть задачи извлечения признаков?
15. Каким образом формируется валидационный набор данных?

16. Какие параметры модели называются гиперпараметрами?
17. Назовите этапы MLOps.
18. Перечислите подходы реализации кросс-валидации.
19. Назовите 2 способа применения нейронных сетей для задачи классификации.
20. Какие методы искусственного интеллекта образуют «большую тройку»?
21. В чем преимущества гибридных подходов?
22. Чем наука о данных (data science) отличается от традиционного анализа данных?
23. Перечислите навыки, которые необходимы специалисту data science.
24. Какие данные называют большими?
25. Какой эффект дает использование модели MapReduce?

Глава 2. Язык программирования Python, основные инструменты для анализа данных на Python

2.1. Кратко о языке программирования Python

Python — это язык программирования, который широко используется в интернет-приложениях, разработке программного обеспечения, а в последние годы при решении задач анализа данных и машинного обучения. Для Python предложено огромное число (порядка 130000) дополнительных пакетов (библиотек классов и функций), которые распространяются через онлайн репозитории.

Язык Python приобрел свою популярность благодаря тому, что он эффективен, прост в изучении и работает на разных платформах. Уже несколько лет Python лидирует в различных рейтингах языков программирования, таких как TIOBE (www.tiobe.com) и IEEE Spectrum (spectrum.ieee.org).

Python – это высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нем программ [24]. Язык является полностью объектно-ориентированным, в Python все является объектами-классами. Язык является интерпретируемым и используется в том числе для написания скриптов. В свою очередь, к недостаткам языка относят более низкая скорость работы и более высокое потребление памяти по сравнению с компилируемыми языками, такими как C или C++.

Важной особенностью Python является интерактивный режим работы, при котором введенные с клавиатуры операторы сразу же выполняются, а результат выводится на экран (REPL, Read-Eval-Print Loop, цикл чтение-вычисление-вывод). Этот режим удобен для быстрого тестирования отдельных фрагментов кода, так как обеспечивает немедленную обратную связь. В анализе данных REPL играет важнейшую роль, так как любая задач анализа данных – своего рода исследование, и аналитику сложно заранее предусмотреть все действия с данными и написать завершенный код программы от первого до последнего действия с данными.

Задумка по реализации языка появилась в конце 1980-х годов, а разработка его реализации началась в 1989 году Гвидо ван Россумом. Гвидо ван Россум назвал язык в честь популярного британского комедийного телешоу 1970-х «Летающий цирк Монти Пайтона», и несмотря на то, что на логотипах, связанных с языком Python часть изображают змею, название язык не имеет отношения к питону. Первая версия языка вышла в 1994 году (Python 1.0), с 2000 года используется версия Python 2.0, с 2008 – версия Python 3.0. В 2023 вышла версия Python 3.12.0.

Python портирован и работает почти на всех известных платформах и операционных системах - от КПК до мейнфреймов. Для задач анализа данных,

машинного обучения и искусственного интеллекта реализованы облачные сервисы такие как Google Colab, Kaggle, Microsoft Azure и др.

Язык Python обладает четким и последовательным синтаксисом, продуманной модульностью и масштабируемостью, благодаря чему исходный код написанных на Python программ легко читаем. Разработчики языка Python придерживаются определенной философии программирования, называемой «The Zen of Python» [25]. Python стремится к более простому, менее громоздкому синтаксису и грамматике, предоставляя разработчикам выбор в их методологии кодирования.

Одной из интересных синтаксических особенностей языка является выделение блоков кода с помощью отступов (пробелов или табуляций), поэтому в Python отсутствуют операторы `begin/end` или фигурные скобки, как в C и C++. Такой подход сокращает количество строк и символов в программе. С другой стороны, поведение и даже корректность программы может зависеть от начальных пробелов в тексте.

Python поддерживает динамическую типизацию, то есть тип переменной определяется во время исполнения, хотя и существуют явные способы задания типа переменной. К базовым типам в Python относятся булевый, целое число произвольной длины, число с плавающей запятой и комплексное число. Контейнерные типы: строка, список, кортеж, словарь и множество. Для задач анализа данных важнейшими типами являются вектора и матрицы Numpy и DataFrame Pandas, однако эти типы не являются встроенным и подключаются через импорт соответствующих библиотек. Как было написано ранее, все переменные являются объектами, как и функции, методы, модули, классы [26].

Разработчик может добавить новый тип переменной либо написав класс, либо определив новый тип в модуле расширения. Поскольку классы в объектно-ориентированном программировании поддерживают наследование, возможно наследование от большинства встроенных типов и типов расширений.

Для списков и других массивов Python предлагает набор операций над срезами (slice). Индексы элементов списка, ка и в большинстве других языков программирования, начинаются с нуля. Запись среза `s[N:M]` для массива `s` означает, что в срез попадают все элементы от `N` до `M`, при этом элемент с индексом `N` включен, с индексом `M` не включен. Индекс можно не указывать, например, для `s[:M]` в срез попадают все элементы начиная с первого (нулевой индекс), а `s[N:]` выбирает все элементы от `N` до последнего в массиве, `s[:]` возвращает все элементы массива.

Имена объектов в Python могут начинаться с буквы любого регистра любого алфавита в Юникоде или подчеркивания, после чего в имени можно использовать и цифры. В качестве имени нельзя использовать ключевые слова, а имена, начинающиеся с символа подчеркивания, имеют специальное значение.

Существует руководство по написанию кода на Python PEP 8 (PEP 8: The Style Guide for Python Code) [25]. Этот документ, созданный на основе рекомендаций Гвидо ван Россума, описывает соглашение о том, как писать код для языка Python и включает: рекомендации о внешнем виде кода (отступы, табуляция, максимальная длинна строки, пустые строки, импорты), пробелы в выражениях и инструкциях, комментарии и документация, контроль версий, а также соглашения по именованию объектов.

Благодаря популярности и распространенности языка Python, существует множество учебников, тьюториалов, интерактивных онлайн ресурсов для обучения Python. В данном учебном пособии мы не предполагаем изучение базового синтаксиса и стандартной библиотеки.

2.2. Установка Python, выбор и настройка IDE

Для работы с Python потребуется установить сам интерпретатор в операционную систему, добавить необходимые пакеты, которых нет в стандартной библиотеке и выбрать интегрированную среду разработки (Integrated Development Environment, IDE). Использование IDE не является обязательным, но упрощает работу и повышает производительность при написании кода.

Для установки интерпретатора Python нужно скачать и установить дистрибутив с официального сайта https://www.python.org/downloads/. Обычно стоит использовать последнюю актуальную версию Python, однако иногда новые версии несовместимы с некоторыми используемыми библиотеками. Обычно данная проблема решается с помощью виртуальных окружений (virtualenv) – «песочниц», в рамках которых запускается приложение со своими версиями библиотек, обновление и изменение которых не затронет другие приложение, использующие те же библиотеки [27].

Для операционных систем семейства Windows дистрибутив Python распространяется либо в виде исполняемого файла (с расширением exe), либо в виде архивного файла (с расширением zip). При установке рекомендуется включить опцию «Add python to PATH» для того, чтобы появилась возможность запускать интерпретатор без указания полного пути до исполняемого файла при работе в командной строке Windows.

В дистрибутивах семейства Linux интерпретатор Python обычно включен в состав дистрибутива, тем не менее, Python может быть установлен из стандартного репозитория используя команду в терминале:

```
> sudo apt-get install python3
```

Существует несколько способов установки Python 3 на MacOS, включая скачивание с официального сайта Python, однако рекомендуется использовать

менеджер пакетов Homebrew. Для работы Homebrew необходим пакет Xcode Apple, для его установки выполните следующую команду:

```
$ xcode-select –install
```

Далее устанавливается Homebrew с помощью команды:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercon-
tent.com/Homebrew/install/master/install)"
```

И наконец, чтобы скачать и установить последнюю версию Python, выполняется команда:

```
$ brew install python3
```

Популярным альтернативным вариантом работы с Python является установка пакета Anaconda, который включает в себя интерпретатор языка Python, набор наиболее часто используемых библиотек, среду разработки и исполнения, запускаемую в браузере и IDE Spyder. Есть варианты установки под Windows, Linux и MacOS. Anaconda использует свои собственные менеджер пакетов и репозитории. Однако, для изучения Python рекомендуется использовать «чистый» Python, а не версию в Anaconda.

В общем случае работа с Python – это написание кода (скрипта) в обычном текстовом файле, который принято сохранять с расширением *.py (file_name.py). Однако для повышения производительности, программисты используют специальные среды для редактирования кода – IDE. Для задач анализа данных и машинного обучения, где аналитик должен концентрировать усилия на исследовании, интерпретации и проверке своих гипотез, выбор «хорошей» IDE становится критически важным.

Обычно IDE объединяет несколько инструментов, специально предназначенных для разработки: редактор, предназначенный для работы с кодом (например, подсветка синтаксиса и автодополнение), инструменты сборки, выполнения и отладки кода, и определенную форму системы управления версиями. В свою очередь по функционалу IDE делятся на универсальные IDE, предназначенные для профессиональной разработки на многих языках, и простые текстовые редакторы с подсветкой синтаксиса и возможностями форматирования кода.

Сегодня существует множество IDE для Python, кратко рассмотрим наиболее известные из них.

Простейший вариант, лишенный любых функций, повышающих удобство и производительность разработки – это написание кода в обычном текстовом редакторе (например, блокнот Windows) и запуск файла в командной строке (рис. 13).

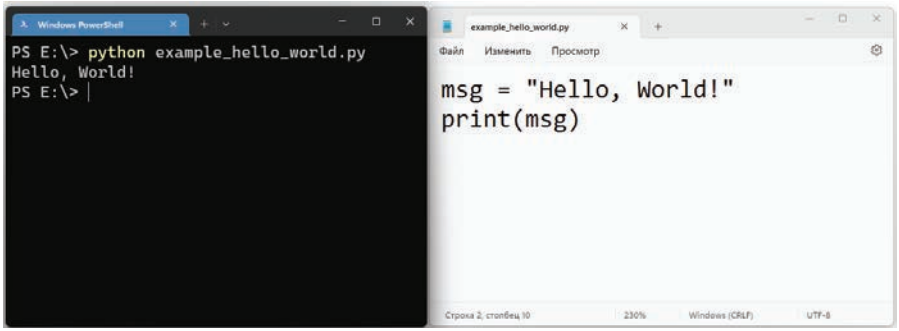


Рис. 13. Python в терминале

Более продвинутый вариант – замена блокнота на один из текстовых редакторов с подсветкой синтаксиса. Среди известных и популярных стоит отметить редакторы Vim, Atom, Sublime Text, Notepad++. Подобные редакторы обычно обеспечивают такой функционал, как подсветка синтаксиса, сворачивание блоков кода, автодополнение и автоматическое закрытие скобок и тэгов, регулярные выражения для поиска и замены и др. Пример отображения кода в Notepad++ показан на рис. 14.

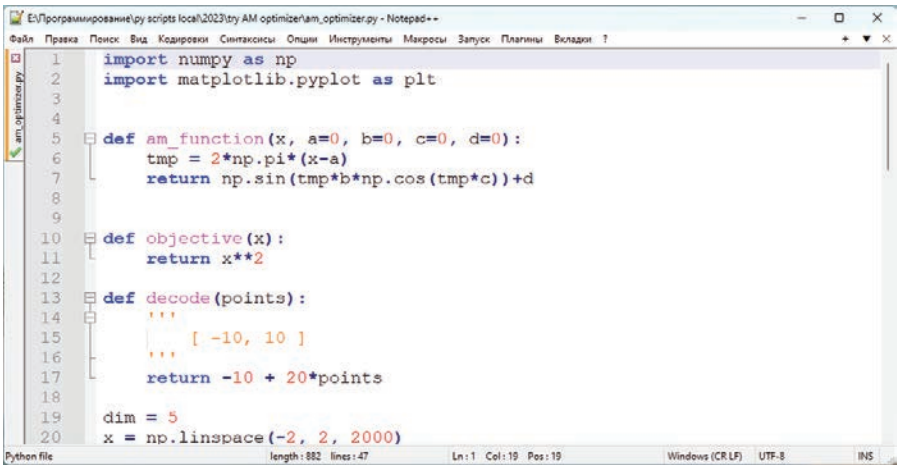


Рис. 14. Python в Notepad++

Python IDLE – это редактор, поставляемый вместе с Python в базовом дистрибутиве. Это самый базовый и максимально упрощенный режим программирования на Python (рис.15). Тем не менее, функционал IDLE включает: интерактивный интерпретатор, автозавершение кода, подсветка синтаксиса, подбор отступа и базовый встроенный отладчик.

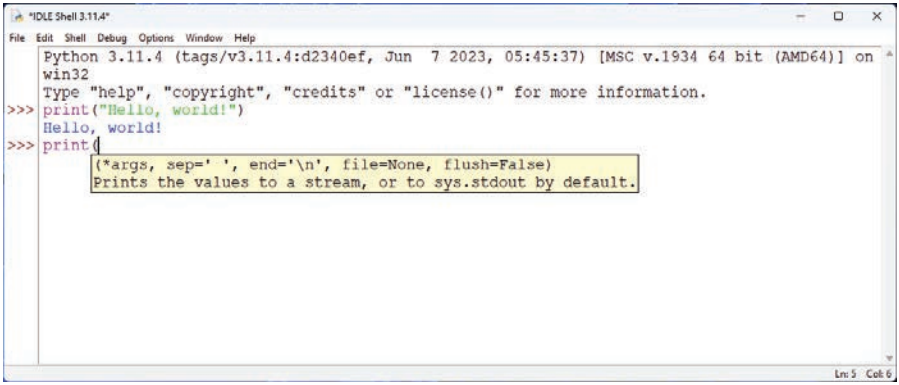


Рис. 15. Python IDLE

PyCharm — это одна из самых популярных интегрированных сред разработки (IDE), специально созданных для Python (рис. 16). Вот некоторые особенности PyCharm: интеллектуальное автозаполнение, встроенная поддержка PEP8, глубокая интеграция с большинством популярных библиотек и фреймворков Python, встроенная отладка и тестирование, интеграция с системами контроля версий. Большой плюс – управление виртуальными окружениями по умолчанию при создании проектов.

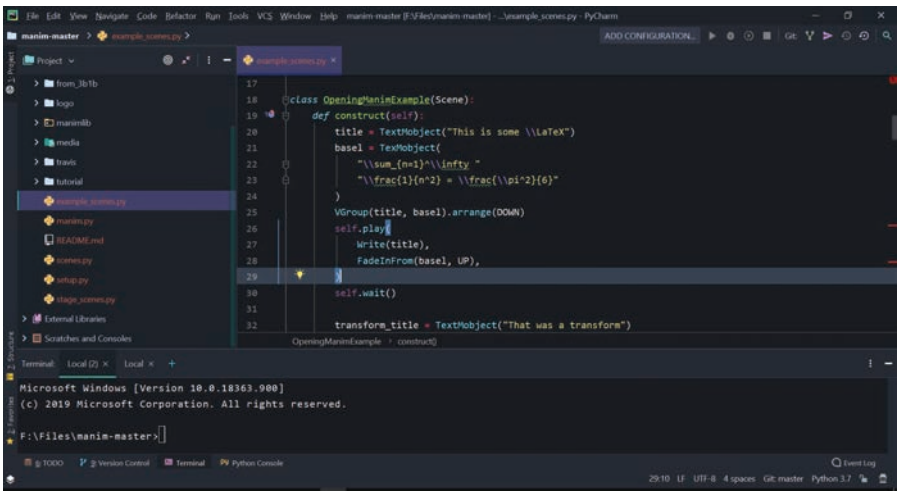


Рис. 16. PyCharm IDE

PyCharm доступен в двух версиях: бесплатная версия Community Edition, которая имеет основной набор функций, достаточный для большинства разработчиков, и платная версия Professional Edition, которая содержит

дополнительные функции, такие как поддержка веб-разработки, работы с базами данных и другие. IDE доступна на Microsoft Windows, Linux и MacOS.

Visual Studio Code (часто сокращается до VS Code) — это бесплатная открытая среда разработки от Microsoft (не путать с Visual Studio) для Windows, Linux и MacOS. Хотя она не является специализированной IDE исключительно для Python, VS Code может использоваться для разработки на Python благодаря своим расширениям и гибкости в настройке (рис. 17).

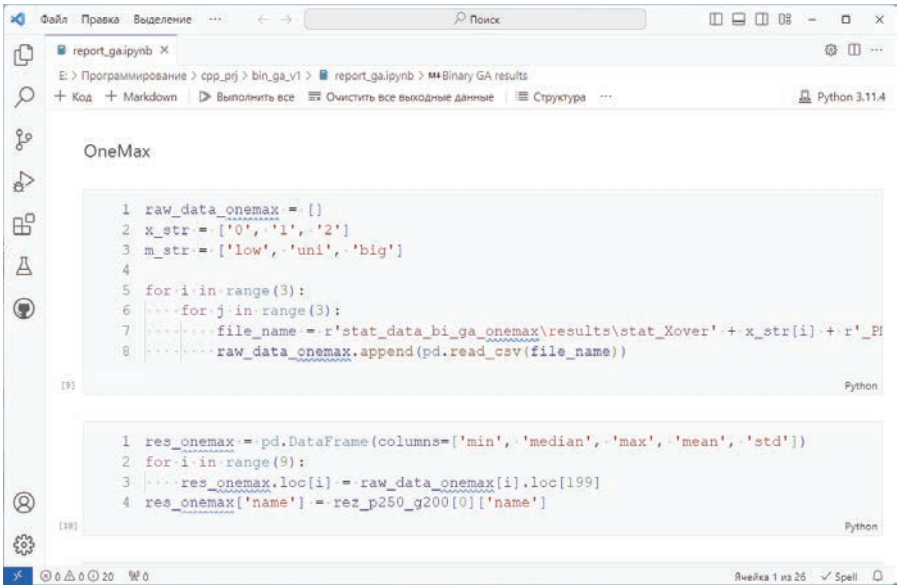


Рис. 17. Visual Studio Code

Некоторые особенности VS Code для Python:

– Расширение Python: Расширение Python для VS Code, разработанное Microsoft, предоставляет множество функций для разработки на Python, включая интеллектуальное автозаполнение, линтинг, форматирование кода, отладку и поддержку Jupyter Notebook. Более того, имеются инструменты, значительно упрощающие именно решение задач анализа данных.

– Встроенный отладчик: VS Code имеет встроенные инструменты для отладки кода, позволяющие устанавливать точки останова, просматривать переменные и контролировать выполнение программного кода.

– Встроенная поддержка Git: VS Code включает поддержку Git «из коробки», позволяющую удобно создавать коммиты, выполнять push и pull обновлений, просматривать разницу между версиями без использования командной строки.

– Поддержка расширений: Одной из сильных сторон VS Code является поддержка расширений, которая позволяет вам настраивать среду разработки под свои нужды. Существует большое количество расширений в собственном репозитории Marketplace для разных языков, библиотек, фреймворков, инструментов форматирования кода и других функций.

– Настройки и гибкость: VS Code очень настраиваема. Настраивается все, от темы и шрифтов до поведения редактора кода и интеграции с другими инструментами.

Редактор постоянно обновляется и поддерживается большим сообществом. VS Code — один из лучших редакторов не только для Python, но и для других языков программирования.

Отдельно стоит упомянуть Jupyter Notebook - интерактивный блокнот, первоначально являвшийся веб-реализацией и развитием фреймворка IPython, ставший самостоятельным проектом, ориентированным на работу со множеством сред выполнения не только Python, но и R, Julia, Scala и ряда других языков. Чаще всего его используют в Data Science для анализа и визуализации данных.

Jupyter Notebook может быть установлен как самостоятельное приложение через дополнительные библиотеки Python. Далее запускается приложение сервер, а работа с интерактивным блокнотом ведется в браузере на отдельно созданной странице. Однако, Jupyter Notebook также доступен через расширения в VS Code, PyCharm или в облачных средах типа Google Colab. Часто функционал Jupyter Notebook в профессиональных IDE выше, чем в обычной реализации. Облачные реализации предоставляют помимо самой IDE вычислительные мощности и облачное хранилище. Это особенно важно для задач машинного обучения, так как пользователь может использовать дорогостоящие GPU и TPU.

Поскольку Jupyter Notebook сегодня является фактически стандартом для аналитика данных и прикладных специалистов по машинному обучению и искусственному интеллекту, остановимся на возможностях Jupyter notebook подробнее и рассмотрим их на примере реализации VS Code и Google Colab.

Jupyter Notebook представляет собой оригинальную концепцию работы на Python с помощью интерактивного блокнота (Notebook), который представляет собой вэб-страницу с формами для редактирования кода Python и форматирования текста, которые называются ячейки (Cells). Число и порядок ячеек может быть любым, пользователь может добавлять, удалять и изменять их в процессе работы с блокнотом.

Сервер Jupyter – это программа, обеспечивающая интерфейс между пользователем, работающем с блокнотом и интерпретатором Python. Сам блокнот выступает в качестве front-end, ячейки генерируют запросы на сервер чтобы выполнить код Python и корректно отобразить форматирование Markdown в текстовых ячейках. Сервер Jupyter в связке с ядром Python – это back-end, отвечающий за выполнение кода Python и формирование содержимого блокнота (рис. 18).

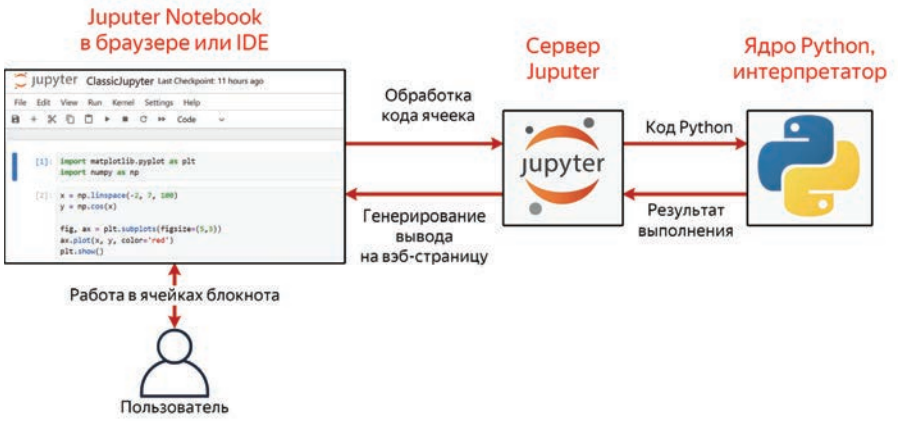


Рис. 18. Схема работы Jupyter Notebook

Текстовые ячейки не влияют на выполнение кода и используются для оформления структуры блокнота, текстовых описаний, комментариев и т.д. Более того, сервер Jupyter обеспечивает в текстовых ячейках форматирование Markdown [28], использование любых тегов HTML и форматирование Latex (например, для отображения сложных формул и математических выражений). На рис. 19 показан пример текстовой ячейки в блокноте, запущенном в Jupyter Notebook.

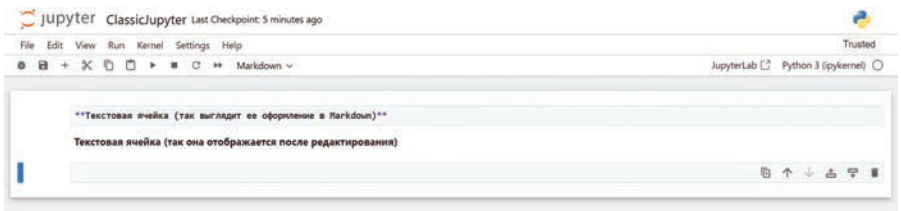


Рис. 19. Текстовая ячейка Jupyter Notebook

Ячейка кода – это форма для работы с кодом на Python, представляющая собой полноценный редактор кода Python с подсветкой синтаксиса и автодополнением. Ячейка может содержать как одну строку кода (одну команду), так и код произвольной длины. Пользователь выполняет код в ячейках в любом порядке, но должен самостоятельно следить за логикой работы, например, за тем, как переопределяются значения переменных. В общем случае, разбивка на ячейки позволяет выполнять код в некоторой логике, например, отражая ход исследования данных, когда следующие ячейки формируются только после того, как был сделан вывод по результатам выполнения предыдущих ячеек.

При запуске кода в ячейке, код отправляется на сервер и обрабатывается интерпретатором Python. Если результат работы кода предполагает какой-то вывод результата: текстовый вывод, вывод изображения или анимации и другие, то сервер осуществляет вывод результата на вэб-страницу ниже ячейки, которая была запущена. Поскольку Jupyter Notebook – это обычная вэб-страница, то для вывода доступны все инструменты HTML и JavaScript, что делает Jupyter Notebook отличным инструментом как для анализа и визуализации данных, так и построения полноценных отчетов, позволяющих одновременно показать и исходный код, и результаты работы в виде удобном для конечного пользователя, а не разработчика. На рис. 20 показан пример выполнения ячейки кода в блокноте, запущенном в Jupyter Notebook.

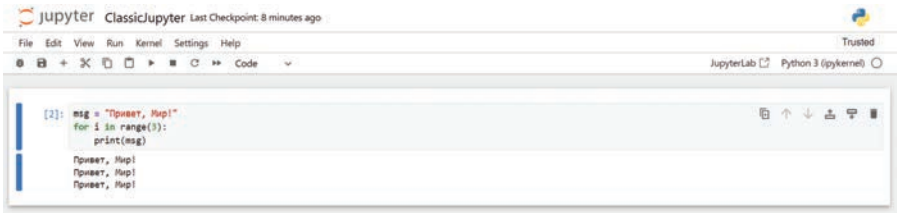


Рис. 20. Текстовая ячейка Jupyter Notebook

В качестве замечания стоит отметить, что после завершения работы с проектом Jupyter Notebook, все, что выводилось в качестве результата выполнения ячеек кода, сохраняется и будет отображено при повторном запуске блокнота. Но сама вычислительная среда обнуляется, т.е. после завершения работы с блокнотом из оперативной памяти удаляются все загруженные пакеты, объявленные функции и переменные. Для повторного выполнения кода необходимо заново инициировать эти объекты.

Разные IDE часто немного модифицируют внешний вид блокнота и могут добавлять новые инструменты работы с блокнотом, однако общая концепция всегда сохраняется. На рис. 21–24 показаны варианты реализации оригинального Jupyter Notebook, VS Code, Google Colab и Kaggle.

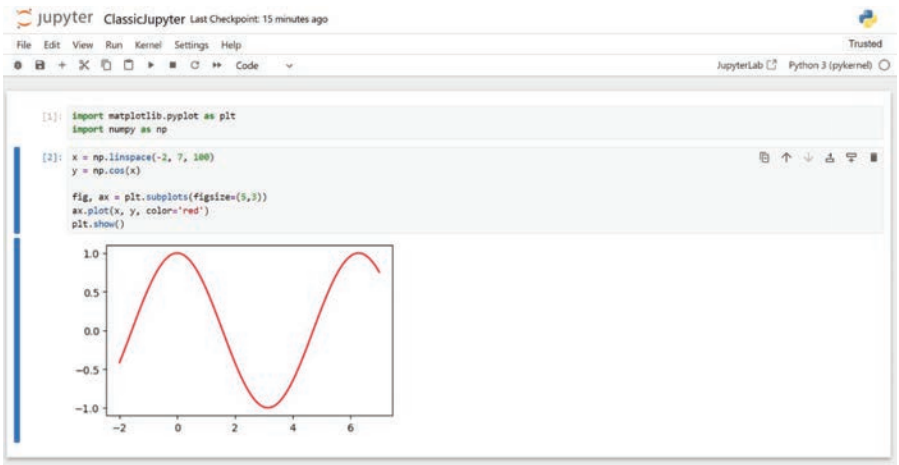


Рис. 21. Классический Jupyter Notebook

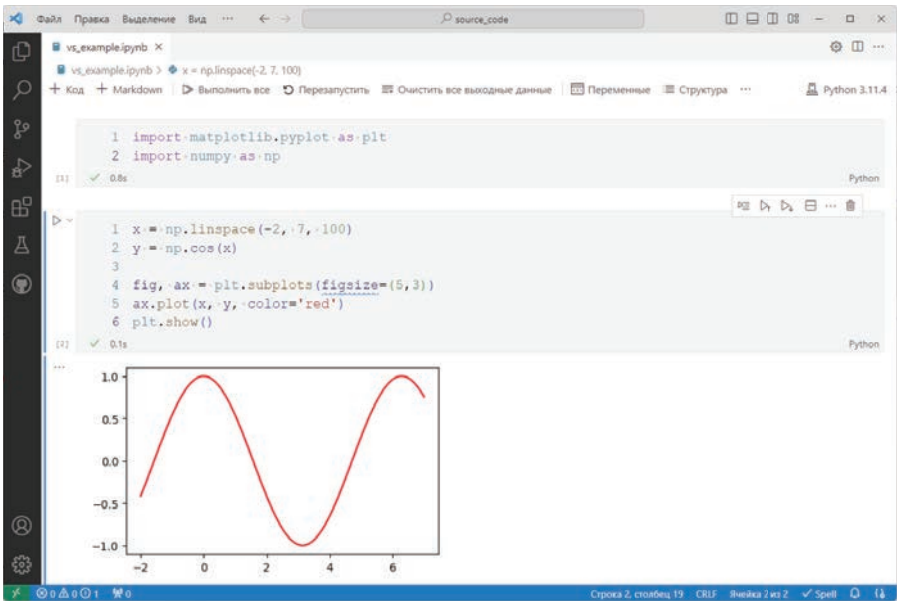


Рис. 22. Блокнот в VS Code

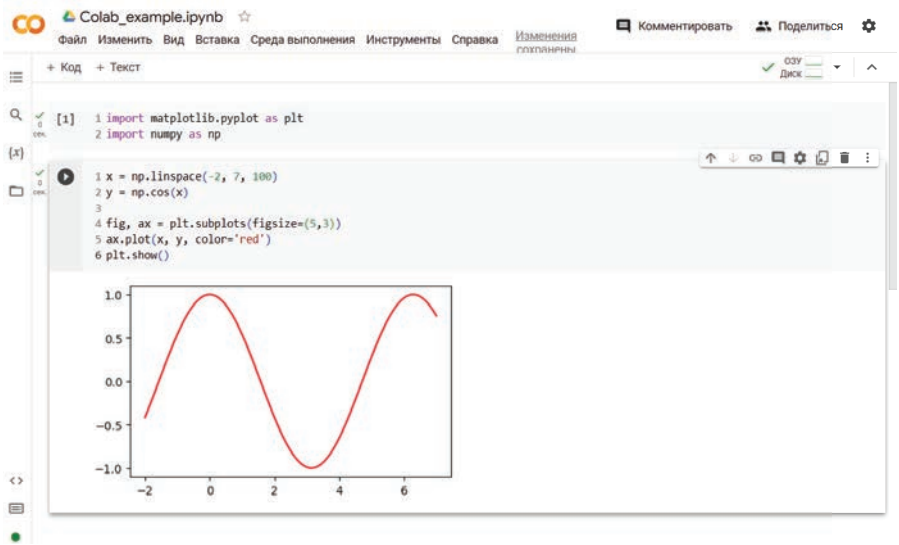


Рис. 23. Блокнот в Google Colab

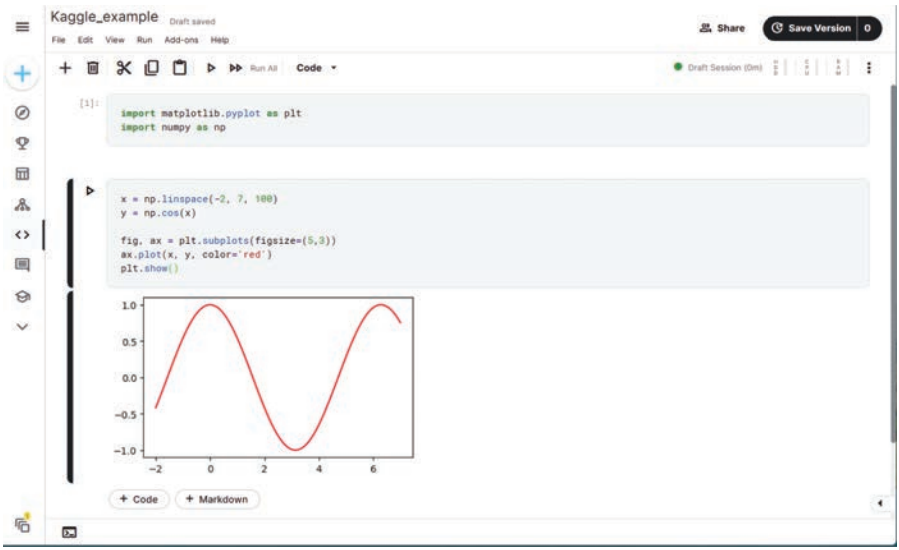


Рис. 24. Блокнот в Kaggle

Рассмотрим некоторые важные элементы форматирования блокнота, полезные для решения задач анализа данных.

В языке разметки Markdown заголовки оформляется символом # (хештэг). Один символ # определяет заголовок 1-го уровня, два символа ## – заголовок 2-го уровня и т.д. Заголовки определяют структуру блокнота, разделы отображаются в навигаторе (оглавлении), а сами разделы являются сворачиваемыми. Наличие структуры позволяет быстро переключаться между логическими разделами, в отличии от обычного текстового файла с кодом на Python (рис. 25).

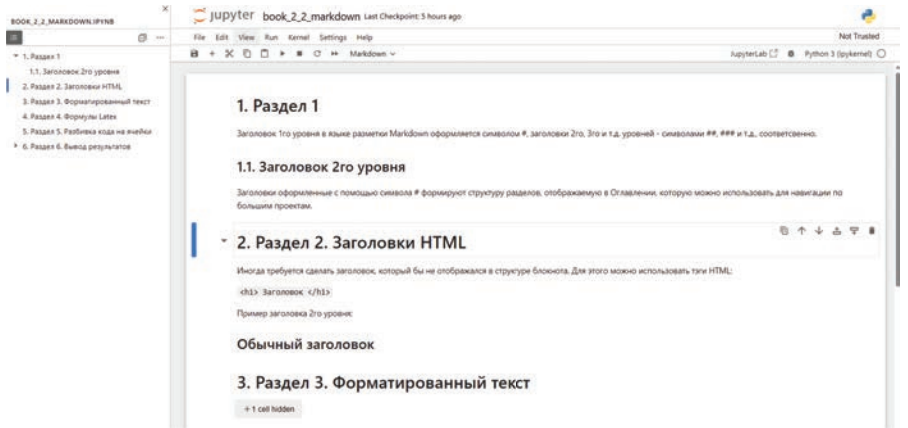


Рис. 25. Заголовки и структура блокнота

Текст в ячейке может быть форматированным: обычный, курсив, жирный, перечисления с маркером или нумерованные, текст с цитированием программного кода. Абзацы отделяются пустой строкой. Текст, оформленный курсивом, заключается между одиночными символами `*курсив*`, оформленный жирным – между двойными символами `**жирный**`. Вставка кода заключается между символами ```` ```code``` ````. Пример оформления текста в ячейке Markdown показан ниже и его отображение на рис. 26:

````
Текст может быть *курсивом*, **жирным**.

Перечисления:
* блок 1
* блок 2

или

1. Блок 1
2. Блок 2

Вставка кода:

```
print('Hello, World!")
```
````

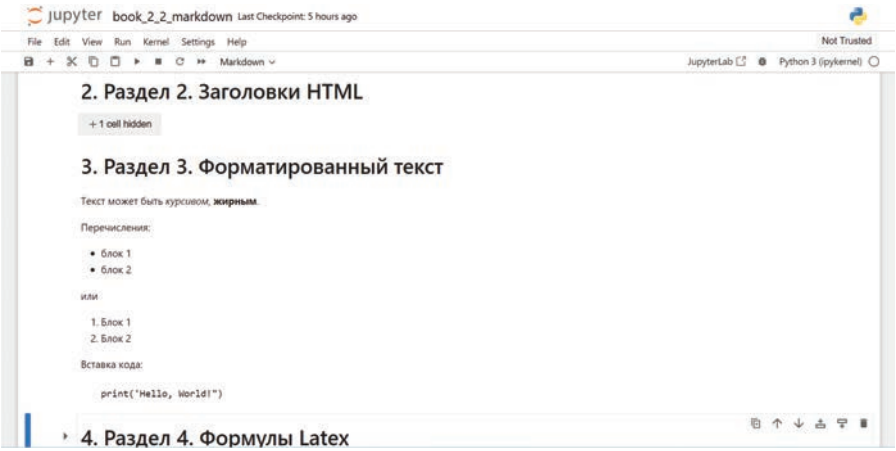


Рис. 26. Форматированный текст

Разметка Markdown поддерживает формулы Latex. Пример описания математического выражения показан ниже и его отображение на рис. 27:

```
Формулы могут быть в строке: $f(x) = \sum_{i=1}^{N+1} x_i^2$.

Или отдельной строкой:

$$f(x) = \sum_{i=1}^{N+1} x_i^2$$
```

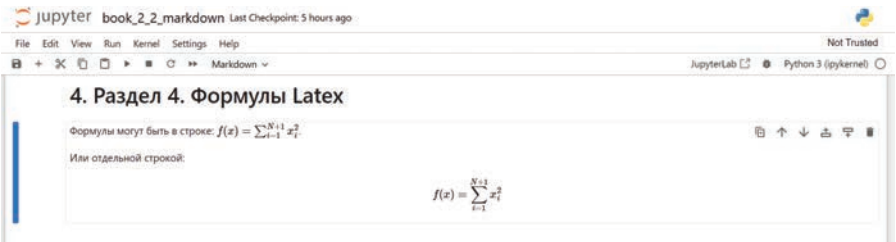


Рис. 27. Формулы Latex

Традиционное программирование на Python – это текстовый файл с кодом, который выполняется от первой до последней строки. Разбивка на ячейки кода позволяет осуществлять декомпозицию кода, для последующего выполнения необходимых фрагментов. В случае изменения кода (например, проверка новой гипотезы о данных или запуск функции с другими параметрами), нет необходимости выполнять весь код в тех частях, где изменения не требуются (рис. 28).

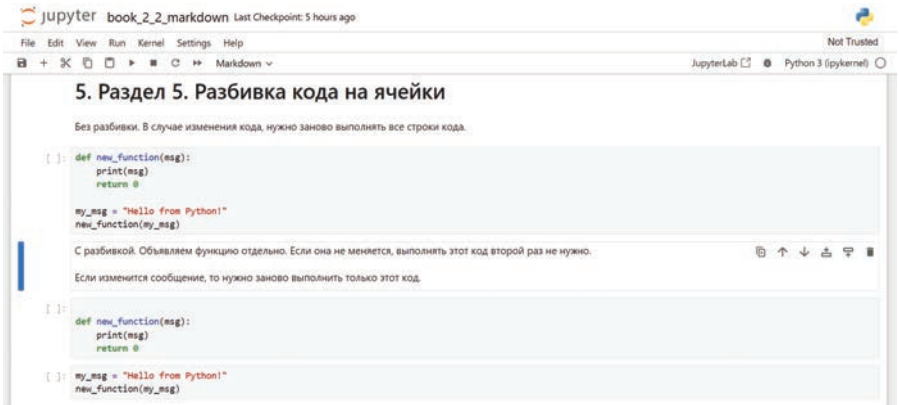


Рис. 28. Разбивка кода на фрагменты

Если ячейка кода подразумевает какой-то вывод результата, например, вывод текста в консоль с помощью функции `print`, результат отображается сразу под запущенной ячейкой. В Jupyter Notebook объект, указанный последним в ячейке будет отображен средствами Jupyter, если объект принадлежит одному из стандартных типов: текст, картинки, видео, таблицы и т.д. Если правила отображения объекта не заданы, то Jupyter отображает тип объекта.

В следующем примере первое сообщение выводится путем явного вызова функции печати текста `print(msg1)`, имя второго сообщения указано, но оно не последнее в ячейке и будет проигнорировано. Имя объекта `msg3` находится в последней строке, его содержимое будет отображено в выводе ячейки (рис. 29).

Рис. 29. Вывод последнего объекта

В библиотеке, реализующей основные компоненты Jupyter Notebook есть функции принудительного отображения содержимого объектов – функция `display` и функция отображения кода `HTML` (рис. 30).

Рис. 30. Функции `display` и `HTML`

Волшебные (магические, Magics) команды – это специальные команды Jupyter Notebook, которые предоставляют пользователям особые функциональные возможности, такие как явное изменение поведения ячейки кода, упрощение общих задач, синхронизация выполнения кода, профилирование и т.д. Магические команды не являются частью языка Python, но понимаются сервером Jupyter, если вызваны из кодовых ячеек. Этим командам предшествует символ % или %%.

Магические команды, которые используются для предоставления особой функциональности одной строке кода (inline) начинаются с одинарного символа %, за которой следует команда. Можно вывести в ячейке все доступные магические команды, запустив `%lsmagic` (на данный момент существует 97 inline магических команд). Команды Cell Magic – это специальные команды, которые позволяют пользователю явно изменять поведение всей ячейки кода. Функции Cell Magic имеют префикс %%, за которым следует название команды (согласно официальной документации, в настоящее время доступно 16 функций Cell Magic).

Некоторые полезные магические команды:

- `%run` : Запустить скрипт Python в текущем ядре.
- `%load` : Загрузить код из скрипта и запустить его в текущем ядре.
- `%who` : Перечислить все переменные.
- `%timeit` : Засечь время выполнения одной строки кода.
- `%debug` : Запустить отладчик в точке исключения.
- `%matplotlib inline` : Отображать графики в блокноте.
- `%load_ext` : Загрузить расширение, например, расширение IPython.
- `%pwd` : Вывести текущий рабочий каталог.
- `%ls` : Показать все файлы в текущем каталоге.

Пример выполнения магической команды `%timeit` показан на рис. 31. Функция `foo()` выполняется 10 раз, оценивается среднее время выполнения функции.

```
[1]: def foo():
         for i in range(100000):
             pass
         return 0

[2]: %timeit -n 10 foo()

1.99 ms ± 244 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Рис. 31. Пример Jupyter Magics

Как отмечалось ранее, некоторые реализации Jupyter Notebook могут добавлять собственный функционал для повышения удобства работы. Рассмотрим для примера некоторые дополнения, которые предложены в Google Colab, https://colab.research.google.com/.

Google Colab — сервис, созданный Google, который предоставляет возможность работать с кодом на языке Python через Jupyter Notebook, не устанавливая на свой компьютер дополнительных программ. В Google Colab можно применять различные библиотеки на Python, загружать и запускать файлы, анализировать данные и получать результаты в браузере. Главная особенность Google Colab – бесплатные мощные графические процессоры GPU и TPU, благодаря которым можно заниматься не только базовой аналитикой данных, но и более сложными исследованиями в области машинного обучения.

Поскольку Jupyter Notebook в основном используется для задач анализа данных, вывод ячейки должен уметь отображать таблицы исходных данных. Обычные средства Jupyter могут представить таблицу или ее часть только как форматированный текст. Google Colab позволяет использовать интерактивный режим, в котором можно просматривать таблицу, ограничивать число строк и диапазон строк для отображения, искать и фильтровать значения (рис. 32).

```
1 data = pd.read_csv("/content/sample_data/california_housing_test.csv")
2 data.head(3)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.05	37.37	27.0	3885.0	661.0	1537.0	606.0	6.6085	344700.0
1	-118.30	34.26	43.0	1510.0	310.0	809.0	277.0	3.5990	176500.0
2	-117.81	33.78	27.0	3589.0	507.0	1484.0	495.0	5.7934	270500.0

```
[5] 1 data
```

1 to 10 of 3000 entries Filter

index	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.05	37.37	27.0	3885.0	661.0	1537.0	606.0	6.6085	344700.0
1	-118.3	34.26	43.0	1510.0	310.0	809.0	277.0	3.599	176500.0
2	-117.81	33.78	27.0	3589.0	507.0	1484.0	495.0	5.7934	270500.0
3	-118.36	33.82	28.0	67.0	15.0	49.0	11.0	6.1359	330000.0
4	-119.67	36.33	19.0	1241.0	244.0	850.0	237.0	2.9375	81700.0
5	-119.56	36.51	37.0	1018.0	213.0	663.0	204.0	1.6635	67000.0
6	-121.43	38.63	43.0	1009.0	225.0	604.0	218.0	1.0641	67000.0
7	-120.65	35.48	19.0	2310.0	471.0	1341.0	441.0	3.225	166900.0
8	-122.84	38.4	15.0	3080.0	617.0	1446.0	599.0	3.6696	194400.0
9	-118.02	34.08	31.0	2402.0	632.0	2830.0	603.0	2.3333	164200.0

Show 10 per page 1 2 10 100 200 250 300

Рис. 32. Таблицы в Google Colab, текстовый вывод (вверху) и интерактивный (внизу)

Еще одним удобным инструментом являются формы. Формы – это специальные команды, добавляющие интерактивные поля ввода значений для переменных. Команды пишутся как комментарии, после которых идет специальный символ и команда, например, `# @title` или `# @param {type:"integer"}`. Преимущества – контроль типа данных при вводе. По умолчанию в ячейке показаны и форма ввода, и исходный код. Пользователь может скрыть код и работать только с формой (рис. 33).

Рис. 33. Формы в Google Colab

2.3. Основные пакеты для анализа данных на Python

Важной особенностью языка Python является возможность подключения сторонних пакетов, содержащих различные объекты и функции, отсутствующие в базовой библиотеке. Технически, каждый пакет – это папки с файлами типа py и предкомпилированными функциями (например, на языке C++). Если используется установщик пакетов pip, то установленные пакеты хранятся в системных каталогах и при импорте интерпретатор знает, где их искать. Иначе, импортируемый пакет ищется в текущем каталоге проекта.

Для задач анализа данных, машинного обучения и искусственного интеллекта следующие пакеты стали стандартными инструментами аналитика [29]:

– Numpy – векторы и матрицы, векторные операции, математические и статистические векторные операции, генерация случайных чисел.

– Matplotlib – «конструктор» для построения графиков, диаграмм и работы с графикой.

– Seaborn – обертка поверх Matplotlib, предназначенная для визуализации данных, содержит ряд инструментов математической статистики, отсутствующие в Matplotlib.

– Pandas – создание и работа со структурой DataFrame, импорт/экспорт табличных данных в разный форматах, инструменты анализа и визуализации данных (примечание, Pandas использует инструменты Numpy и Matplotlib).

– Sklearn – наиболее известный на сегодня пакет для анализа данных и машинного обучения.

Данные пакеты не являются стандартными для Python, поэтому первоначально требуется их установка (в облачных системах эти пакеты присутствуют по умолчанию). Для установки нужно выполнить в терминале (командной строке) от имени администратора команду `pip install` и указать имя пакета или имена нескольких пакетов через пробел:

```
> pip install numpy matplotlib pandas seaborn sklearn
```

По умолчанию из репозитория будет загружена актуальная версия пакета. Если требуется установка особой версии, необходимо обратиться к документации и изучить возможность использования виртуальных окружений.

Далее в тексте будут показаны примеры выполнения некоторых команд на языке Python. Если кроме самого кода, будет показа пример вывода результата, то в листинге кода вывод и исходный код будут выделены разным цветом:

```
a, b = 2, 3
print("2 + 3 =", a + b)
```

```
Вывод:
2 + 3 = 5
```

2.3.1. Пакет Numpy

2.3.1.1. Вектора Numpy

Библиотека Numpy (сокращение от Numerical Python — «числовой Python») реализует специальный тип массивов `<class 'numpy.ndarray'>` похожий на встроенный тип данных языка Python `list`, но обеспечивающий гораздо более эффективное хранение и операции с данными. Практически во всех тестах на производительность, работа с данными Numpy оказывается многократно быстрее, чем со стандартными списками, и отрыв увеличивается с ростом объема данных.

Библиотека Numpy [30, 31] довольна большая и, более того, позволяет проектировать свои собственные векторные функции, поэтому изучить ее целиком в рамках данного учебного пособия затруднительно. Рассмотрим некоторые особенности и наиболее полезные для анализа данных операции.

После импорта любого пакета, обращение к элементам пакета происходит через имя пакета. Часто импортируется не весь пакет, а лишь его часть, или имя пакета весьма громоздкое, для этого в Python используется произвольные псевдонимы импортированных пакетов. Далее для импорта пакета Numpy будет использоваться стандартный псевдоним, который встречается в документации, книгах, на различных сайтах:

```
import numpy as np

# иногда удобнее импортировать часть пакета
import numpy.random as rnd

# импорт одной функции
from numpy.random import uniform as my_rand
```

Вначале рассмотрим простой пример, демонстрирующий что такое векторная обработка данных. Создадим список из целых чисел:

```
list_of_numbers = [2, 4, 6, 8]
```

Если мы захотим увеличить все значения на 2, то следующая операция выдаст ошибку:

```
list_of_numbers + 2
```

Дело в том, что обычный массив list может хранить любые элементы любых типов, и поэтому, требуется проверять типы и возможность выполнения данной операции. В списках это не реализовано и отдается на усмотрение разработчика, сам список – это лишь контейнер. Т.е. для того, чтобы увеличить все значения на 2, нужно выполнить, например, такой код:

```
for i in range(len(list_of_numbers)):
    list_of_numbers[i] = list_of_numbers[i] + 2
```

Реализуем аналогичный массив средствами Numpy. Массивы Numpy могут быть получены из списков, из явного указания значений и многими другими способами:

```
numpy_array = np.array( list_of_numbers )
numpy_array + 2
```

Теперь операция выполняется корректно и каждый элемент массива будет увеличен на 2. Функции Numpy являются векторными, т.е. каждый раз они применяются ко всем элементам массива (если не указаны дополнительные правила обработки массива), например:

```
np.sin(numpy_array)
```

Такой подход не типичен для традиционных языков программирования, но как только он освоен, код на Python становится лаконичным, а многие поточные обработки массивов не требуют низкоуровневой реализации (циклы, условия, перебор элементов массива) и выполняются эффективно средствами Numpy. Понимая суть векторов Numpy, можно создавать свои векторные функции, важно лишь следить за логикой обработки данных в массиве:

```
def my_vectorized_power2(x):
    return x**2
```

Рассмотрим некоторые базовые свойства и операции с массивами Numpy.

Массивы Numpy могут иметь любую размерность, которая называется форма (`shape`). Одномерные массивы представляют собой вектора, двухмерные – матрицы (тем не менее, в Numpy существует особый класс matrix для операций линейной алгебры). Каждое измерение в массиве называется осью (axis), которые нумеруются с нуля и используются для явного указания логики работы с массивом, например, сложить все элементы по строкам матрицы, т.е. по `axis=1` (рис. 34).

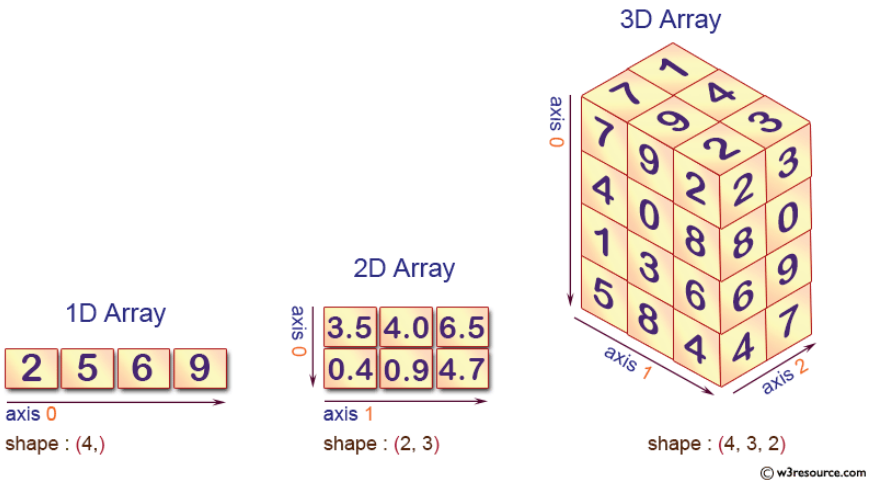


Рис. 34. Массивы Numpy (источник www.w3resource.com)

Узнать форму массива можно или обратившись к свойству массива `shape`, или вызвав функцию `shape()`:

```
array_1 = np.array( [[2, 1, 3, 5], [0, 1, 2, 3]])
print(array_1)
print(array_1.shape)
print(np.shape(array_1))
```

Вывод

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

Пример указания номера оси для управления логикой выполнения функции сложения:

```
# сумма всех элементов
print(np.sum(array_1))
# сумма по первой оси, по столбцам
print(np.sum(array_1, axis=0))
# сумма по второй оси, по строкам
print(np.sum(array_1, axis=1))
```

Вывод

```
17
[2 2 5 8]
[11  6]
```

Двумерные массивы могут быть транспонированы, при этом форма массива (число элементов по осям) изменится:

```
print(np.transpose(array_1))
print(np.shape(np.transpose(array_1)))
```

Вывод

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

Существует важная функция изменения размера массива `reshape()`. Массив может быть преобразован в массив любой формы и размерности, но такой, чтобы число элементов исходного и результирующего массивов совпадали:

```
array_2 = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
print(array_2)
print(array_2.reshape(2,6))
print(array_2.reshape(4,3))
```

Вывод

```
[ 1  2  3  4  5  6  7  8  9 10 11 12]
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Функция `reshape()` часто необходима в задачах машинного обучения в случае, когда используется одномерный массив с данными. В анализе данных одна строка – это одна запись БД, в которой несколько атрибутов. Алгоритму машинного обучения требуется различать обучающие примеры как отдельные строки:

```
array_3 = np.array([1, 2, 3, 4, 5])
print("Для БД массив - это одна запись с 5 атрибутами")
print(array_3)
print("Теперь это 5 записей, каждая с 1 атрибутом")
print(array_3.reshape(-1, 1))
```

Вывод

```
Для БД массив - это одна запись с 5 атрибутами
[1 2 3 4 5]
Теперь это 5 записей, каждая с 1 атрибутом
[[1]
 [2]
 [3]
 [4]
 [5]]
```

Доступ к элементам массивов и срезы (slice) массивов Numpy аналогичны тому, как это реализовано в обычных списках `list`. Массивы Numpy предоставляют удобную векторную обработку логических условий. Проверка условий осуществляется поэлементно, формируя тем самым маску – массив аналогичной формы, который содержит значения `True/False`. В свою очередь, маски могут быть использованы как выбора элементов массива без явного указания значений индексов – отбираются все те элементы, где в соответствующей позиции маски стоит `True`:

```
array_3 = np.array([10, 2, 11, 3, 12, 13, 4, 5])
mask = array_3 <10
print(array_3)
print(mask)
print(array_3[mask])
# без предварительного определения маски
print(array_3[ array_3 < 10 ])
```

Вывод

```
[10  2 11  3 12 13  4  5]
[False  True False  True False False  True  True]
[2 3 4 5]
[2 3 4 5]
```

Чтобы узнать индексы элементов, для которых выполняется логическое условие можно применить функцию `where()`, она же используется для изменения значений массива по логическому условию:

```
print( "индексы, где array_3 < 10:", np.where(array_3 < 10) )
array_3_1000 = np.array([1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000])
array_3_555 = np.array([555, 555, 555, 555, 555, 555, 555, 555])
print( np.where(array_3 < 10, array_3, array_3_555) )
print( np.where(array_3 < 10, array_3_1000, array_3) )
```

Вывод

```
индексы, где array_3 < 10: (array([1, 3, 6, 7], dtype=int64),)
[555   2 555   3 555 555   4   5]
[  10 1000   11 1000   12   13 1000 1000]
```

2.3.1.2. Функции создания заполненных массивов

В Numpy реализованы различные функции, используемые для создания определенным образом заполненных массивов произвольной формы. Рассмотрим некоторые важные функции.

Наиболее важными являются функции, которые генерируют значения равномерно в заданном интервале. Например, для создания узлов сетки при построении графиков или поиска параметров алгоритмов машинного обучения (поиск по решетке, grid search). Функция `linspace()` генерирует заданное число значений, автоматически определяя шаг. Функция `arange()` генерирует значения с заданным шагом, начиная от начала интервала (первое значение) до конца интервала (последнее значение меньше либо равно правой границе интервала, зависит от шага):

```
nodes_1 = np.linspace(-10, 10, num=13)
print(nodes_1)
nodes_2 = np.arange(-10, 10, 0.85)
print(nodes_2)
```

Вывод

```
[-10.          -8.33333333  -6.66666667  -5.          -3.33333333
  -1.66666667   0.           1.66666667   3.33333333   5.
   6.66666667   8.33333333  10.        ]
[-10.    -9.15  -8.3   -7.45  -6.6   -5.75  -4.9   -4.05  -3.2   -2.35
  -1.5   -0.65   0.2    1.05   1.9    2.75   3.6    4.45   5.3    6.15
   7.     7.85   8.7    9.55]
```

Функция `empty()` создаете «пустой» массив, значения которого неопределенны и заполнены «мусором» из оперативной памяти:

```
empty_array = np.empty(shape=(3,3))
print(empty_array)
```

Вывод

```
[[2.19872335e-316, 0.00000000e+000, 6.92152587e-310],
 [2.20820625e-316, 0.00000000e+000, 0.00000000e+000],
 [9.88131292e-324, 0.00000000e+000, 3.95252517e-322]]
```

Функции `zero()` и `ones()` создают массивы заполненные нулями и единицами, соответственно, а функция `full()` – заданным значением:

```
zero_array = np.zeros((3,5))
print(zero_array)
one_array = np.ones((2,7))
print(one_array)
full_array = np.full( (3,4), 15.7 )
print (full_array)
```

Вывод

```
[[0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0.]]
[[1., 1., 1., 1., 1., 1., 1.],
 [1., 1., 1., 1., 1., 1., 1.]]
[[15.7, 15.7, 15.7, 15.7],
 [15.7, 15.7, 15.7, 15.7],
 [15.7, 15.7, 15.7, 15.7]]
```

Для матричной алгебры часто требуется единичная матрица (элементы главной диагонали которой равны единице, а остальные равны нулю), для чего используется функция `eye()`:

```
print(np.eye(5))
```

Вывод

```
[[1., 0., 0., 0., 0.],
 [0., 1., 0., 0., 0.],
 [0., 0., 1., 0., 0.],
 [0., 0., 0., 1., 0.],
 [0., 0., 0., 0., 1.]]
```

2.3.1.3. Математические операции и функции

Рассмотрим некоторые особенности математических операций с массивами Numpy. Операции сложения, вычитания, умножения и деления выполняются поэлементно для двух массивов одинаковой формы. Для выполнения можно применять как перегруженные операторы `+`, `-`, `*`, `/`, так и аналогичные функции Numpy: `add()`, `subtract()`, `multiply()`, `divide()`.

Так же Numpy реализует векторные варианты всех элементарных функций, которые применяются поэлементно: тригонометрические, гиперболические, логарифмы и степенные, и многие другие. Пример, векторной функции `sin()`:

```
x = np.linspace(-10, 10, num=7)
y = np.sin(x)
print(x)
print(y)
```

Вывод

```
[-10.          -6.66666667  -3.33333333   0.           3.33333333
   6.66666667  10.        ]
[ 0.54402111 -0.37415123  0.19056796  0.         -0.19056796  0.37415123
 -0.54402111]
```

Для выполнения умножения матриц по правилам линейной алгебры используется функция `dot()`:

```
A = np.array([ [1, 2], [3, 4] ])
B = np.array([ [10, 20], [30, 40] ])
C = np.dot(A, B) # аналогичная запись  C = A.dot(B)
print(A)
print(B)
print(C)
```

Вывод

```
[[1 2]
 [3 4]]
[[10 20]
 [30 40]]
[[ 70 100]
 [150 220]]
```

2.3.1.4. Генерация случайных чисел

Случайные числа и различные функции математической статистики – основа многих методов анализа данных, машинного обучения и искусственного интеллекта. Умение пользоваться генераторами случайных числе очень важное.

Необходимые функции определены в разделе пакета `np.random`, которая может быть импортирована отдельно со своим псевдонимом. В Numpy по умолчанию используется достаточно эффективный генератор PCG64, который можно инициализировать «зерном» с помощью параметра `seed`. При инициализации «зерна» одним и тем же значением, последовательность случайных чисел будет каждый раз одинаковая, что можно использовать в численных экспериментах для повторяемости результатов:

```
np.random.seed(0)
print("Зададим 'зерно' равное 0.")
print(np.random.rand(8))
print(np.random.rand(8))
print("Зададим 'зерно' равное 0 заново.")
np.random.seed(0)
print(np.random.rand(8))
```

Вывод

```
Зададим 'зерно' равное 0.
[0.5488135  0.71518937 0.60276338 0.54488318 0.4236548  0.64589411
 0.43758721 0.891773  ]
[0.96366276 0.38344152 0.79172504 0.52889492 0.56804456 0.92559664
 0.07103606 0.0871293 ]
Зададим 'зерно' равное 0 заново.
[0.5488135  0.71518937 0.60276338 0.54488318 0.4236548  0.64589411
 0.43758721 0.891773  ]
```

Наиболее часто используемые законы распределения: равномерный и нормальный реализованы функциями `random.uniform()` и `random.normal()`:

```
print(np.random.uniform(-5, 5))
print(np.random.uniform(-5, 5, (5,)))
print(np.random.uniform(-5, 5, (2, 5)))
```

Вывод

```
-3.817255741310668
[ 1.39921021 -3.56646713  4.44668917  0.21848322 -0.8533806 ]
[[-2.35444388  2.74233689 -0.43849668  0.68433949 -4.812102  ]
 [ 1.17635497  1.12095723  1.16933997  4.43748079  1.81820299]]
```

```
# np.random.normal(мат.ожидание, стандартное отклонение)
print(np.random.normal(0, 1))
print(np.random.normal(0, 1, (5,)))
print(np.random.normal(0, 1, (2, 5)))
```

Вывод

```
-0.8877857476301128
[-1.98079647 -0.34791215  0.15634897  1.23029068  1.20237985]
[[-0.38732682 -0.30230275 -1.04855297 -1.42001794 -1.70627019]
 [ 1.9507754  -0.50965218 -0.4380743  -1.25279536  0.77749036]]
```

Для работы с произвольными массивами часто полезны функции случайной перестановки и случайного выбора заданного числа элементов с и без повторений `random.permutation()` и `random.choice()`:

```
some_array = [1, 2, 10, 20, 100, 200, 1000, 2000]
# случайные перестановки
print(np.random.permutation(some_array))
# случайные 5 элемента с повторением
print(np.random.choice(some_array, 5))
# случайные 5 элемента из 5 без повторения
print(np.random.choice(some_array, 5, replace=False))
```

Вывод

```
[   2   10  100   20 2000 1000    1  200]
[ 10   1 100  20 100]
[2000   10    2   20  100]
```

При решении различных прикладных задач и реализации стохастических алгоритмов анализа данных, машинного обучения и искусственного интеллекта необходимо моделировать случайные события с заданной вероятностью $P(\text{события A}) = p$. Общая схема разыгрывания вероятности p использует следующий алгоритм:

1. Генерируется равномерно распределенная случайная величина rnd в интервале $[0, 1]$.
2. Если $rnd \leq p$ (вероятность попасть в интервал $[0, p]$), то считаем что случайное событие A произошло, иначе – не произошло.

Для примера рассмотрим моделирование подбрасывания двух «неправильных» монет, у одной из которых чаще выпадает «орел» с вероятностью $P_1 = 0.75$, а у второй «орел» выпадает реже «решки» с вероятностью $P_2 = 0.25$:

```
P1, P2 = 0.75, 0.25
if(np.random.uniform(0,1) <= P1):
  print("У монеты 1 выпал орел")
else:
  print("У монеты 1 выпала решка")

if(np.random.uniform(0,1) <= P2):
  print("У монеты 2 выпал орел")
else:
  print("У монеты 2 выпала решка")
```

Вывод

```
У монеты 1 выпал орел
У монеты 2 выпала решка
```

2.3.1.5 Функции-агрегаторы

Агрегаторы – функции, вычисляющие одну характеристику для всего набора значений массива Numpy. Примером такого агрегатора является функция сложения всех элементов. Для задач анализа данных полезными бывают функции вычисления размаха значений, центральных моментов и вариации: `min()`, `max()`, `mean()`, `std()`, `median()`, используемые в описательной статистике:

```
# сгенерируем случайные данные, распределенные нормально
data = np.random.normal(3, 1.5, 100)
print( "Минимальное =", data.min() )
print( "Максимальное =", data.max() )
print( "Математическое ожидание  =", data.mean() )
print( "Среднеквадратическое отклонение =", data.std() )
print( "Медиана =", np.median(data) )
```

Вывод

```
Минимальное = -0.44120621819912653
Максимальное = 6.426500031705516
Математическое ожидание  = 3.0121800360686053
Среднеквадратическое отклонение = 1.588377996952718
Медиана = 3.1380487626581255
```

Для многомерных массивов агрегаторы могут применяться с указанием оси:

```
A = np.array([[3, 15, 75],[42, 11, 20]])
print(A)
print("max по всему массиву:", np.max(A))
print("max по столбцам:", np.max(A, axis=0))
print("max по строкам:", np.max(A, axis=1))
```

Вывод

```
[[ 3 15 75]
 [42 11 20]]
max по всему массиву: 75
max по столбцам: [42 15 75]
max по строкам: [75 42]
```

Для агрегаторов, которые выбирают конкретное значение из массива, а не вычисляют некоторую характеристику, часто нужно знать индекс выбранного элемента. Для этого применяются функции следующего вида: `argName`, где

`Name` – имя функции агрегатора. Например, для функции `max()` применяется `argmax()`:

```
data = np.array([3, 15, 75, 42, 11, 20])
print("наибольший элемент:", data.max())
print("индекс наибольшего элемента",  data.argmax())
```

Вывод

```
наибольший элемент: 75
индекс наибольшего элемента 2
```

Функции вида `argName` могут применяться и к другим функциям, которые манипулируют элементами. Например, для функции сортировки:

```
data = np.array([55, 22, 11, 33, 44])
print("до сортировки:", data)
print("после сортировки:", np.sort(data))
print("индексы сортировки (новый порядок элементов):", np.argsort(data))
```

Вывод

```
до сортировки: [55 22 11 33 44]
после сортировки: [11 22 33 44 55]
индексы сортировки (новый порядок элементов): [2 1 3 4 0]
```

Пакет Numpy содержит огромное число тематических разделов и различных функций. В этом разделе мы затронули лишь те, которые часто используются в анализе данных и будут далее применяться в этом учебнике. Дополнительная информация по пакету может быть получена из официальной документации или в любом специализированном учебнике по разработке на Python.

2.3.2. Пакет Pandas

В задачах анализа данных и в Data Science в целом обычно используются программные структуры – объекты с прямоугольными данными (наподобие электронной таблицы или таблицы базы данных) [15]. Прямоугольные данные – это общий термин для двумерного массива (матрицы), в котором строки обозначают некоторые записи (примеры, прецеденты, результаты наблюдений или экспериментов, экземпляры, паттерны и т.д.), а столбцы – признаки описания этих записей (переменные, атрибуты, характеристики, входы). Некоторые из столбцов могут быть целевыми (зависимая переменная, отклик, исход, выход), многие задачи анализа данных состоят в том, чтобы научиться понимать, как формируются значения целевых признаков на основе прочих.

В языке Python в пакете Pandas реализован специальный формат представления – кадр данных (DataFrame) [32, 33]. Название как бы подчеркивает, что это один «кадр», «мгновенная фотография» некоторого генерального набора данных. DataFrame содержит текущую актуальную информацию, отражающую некоторый контекст и представляет данные, которые будут подвергаться анализу с заданной целью. При смене цели, контекста или существенных изменениях информации необходимо формировать новый кадр актуальных данных.

Pandas – это программная библиотека на языке Python для обработки и анализа данных, которая строится поверх библиотеки Numpy и Matplotlib, являющихся инструментами более низкого уровня. Pandas предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временными рядами. Основная область применения – обеспечение решения на Python таких задач, как сбор и очистка данных, импорт и экспорт данных, анализ и моделирование данных, визуализация и другие. Библиотека оптимизирована для высокой производительности, так как написана на Cython и C++, и поэтому используется, в том числе и для анализа больших данных. Библиотека была создана в 2008 году компанией AQR Capital, а в 2009 году она стала проектом с открытым исходным кодом с поддержкой большого комьюнити.

Среди ключевых возможностей библиотеки Pandas стоит отметить:

– Класс DataFrame для работы с индексированными массивами двумерных данных.

– Инструменты для обмена данными между структурами в памяти и файлами различных форматов.

– Встроенные средства слияния данных и способы обработки пропусков в данных.

– Срез данных по значениям индекса, расширенные возможности индексирования, выборка из больших наборов данных.

– Возможности группировки, операции типа «разделение, изменение, объединение» (split-apply-combine).

– Работа с временными рядами: формирование временных периодов и изменение интервалов и так далее.

Исходные данные не всегда поступают в табличном виде или не всегда пригодны для использования в задачах анализа. Такие данные называют «сырыми» (raw data). Для формирования DataFrame «сырые» данные необходимо преобразовать в таблицы, подготовить, выделить существенные признаки и записи. Сегодня DataFrame – это стандарт, базовая структура данных для задач математической статистики, анализа данных и машинного обучения.

Традиционные массивы и таблицы базы данных имеют один или несколько дополнительных столбцов, не содержащих предметные данные, но используемые для доступа к элементам таблицы – индексы. По умолчанию для объектов DataFrame создаются автоматические целочисленный индексы строк и столбцов, существует возможность задавать многоуровневые/иерархические индексы с целью повышения эффективности некоторых операций. Поскольку данные отражают информацию о некоторых реальных объектах (процессах, системах) предметной области и должны быть интерпретируемыми, дополнительно к числовым индексам в DataFrame можно задавать индексы в виде имен столбцов и строк.

Библиотека Pandas довольна большая и постоянно дополняется. Рассмотрим некоторые особенности и наиболее полезные для анализа данных операции. Важно усвоить основную концепцию представления и манипулирования

данными с помощью DataFrame, а конкретные прикладные инструменты анализа данных будут выбираться исходя из целей и задач анализа.

2.3.2.1. Серии и таблицы Pandas

Для импорта пакета Pandas будем использоваться стандартный псевдоним, который встречается в документации, книгах, на различных сайтах:

```
import pandas as pd
```

Основная структура данных в Pandas – это серия (Series). Серия – это одномерный массив, который имеет индексы входящих в него объектов. Индексы могут быть как номерами (по умолчанию), так и любыми именами. Тип данных в серии может быть любой.

«Кадр данных» DataFrame – это составная структура, двумерный массив или таблица, в котором каждый столбец является объектом Series (рис. 35). Названия столбцов – это тоже индексы, точно такие же, как и у строк. Поэтому при транспонировании DataFrame столбы становятся строками и наоборот, сохраняя все правила обращения к ним по индексам (рис. 36).

Рис. 35. «Кадр данных» из нескольких серий

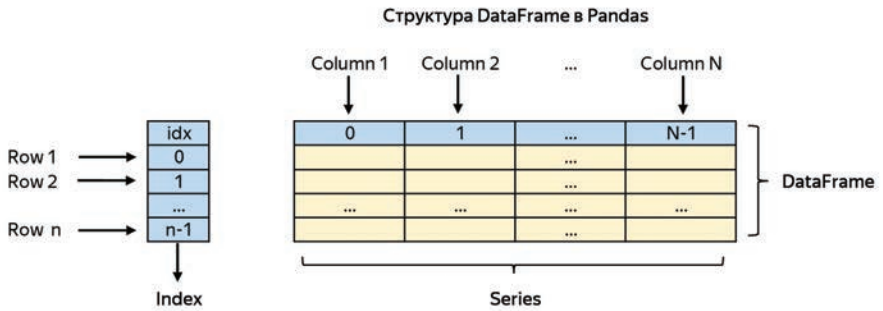


Рис. 36. Структура DataFrame

Рассмотрим некоторые операции с сериями.

Создание серии возможно, например, путем указания произвольного массива данных, при этом индексация будет только числовая:

```
some_array = np.array([11, 33, 72, 15, 100])
series_1 = pd.Series(some_array)
print(series_1)
```

Вывод

```
0     11
1     33
2     72
3     15
4    100
dtype: int32
```

При создании серии возможно явное указание индексов-наименований, при этом индексы-номера также используются.

```
some_array = np.array([11, 33, 72, 15, 100])
series_2 = pd.Series(some_array, index=["a1", "b1", "c1", "d1", "e1"])
print(series_2)
print("Пятый элемент:", series_2[4])
print("Элемент с именем 'e1':", series_2['e1'])
```

Вывод

```
a1     11
b1     33
c1     72
d1     15
e1    100
dtype: int32
Пятый элемент: 100
Элемент с именем 'e1': 100
```

Наиболее полный способ – это создание серии путем задания стандартного словаря Python. Индексы – это массив имен, можно задать дополнительное наименование этого массива. Также есть возможность задать имя для всей серии.

```
some_dictionary = {
      "Математика": "отл",
      "История": "хор",
      "Ин.яз.": "хор",
      "Физика": "отл" }
series_3 = pd.Series(some_dictionary)
series_3.name = "Оценки Иванова И.И."
series_3.index.name = "Предметы"
series_3
```

Вывод

```
Предметы
Математика    отл
История       хор
Ин.яз.        хор
Физика        отл
Name: Оценки Иванова И.И., dtype: object
```

Для описания данных серии используются две важные функции, которые применяются аналогично и для DataFrame: `info()` и `describe()`. Функция `info()` показывает общую информацию о структуре и элементах серии, показывает число непустых значений (пустое значение имеет специальный тип `np.nan`, аналог NULL в БД), тип значений серии и занимаемый объем памяти.

```
series_3.info()
```

Вывод

```
<class 'pandas.core.series.Series'>
Index: 4 entries, Математика to Физика
Series name: Оценки Иванова И.И.
Non-Null Count  Dtype
--------------  -----
4 non-null      object
dtypes: object(1)
memory usage: 236.0+ bytes
```

Функция `describe()` показывает описательную статистику. Для серий числовых данных: число непустых записей, оценки математического ожидания и стандартного отклонения, порядковую статистику, включая процентили 0% (минимальное значение), 25% (1й квартиль), 50% (медиана), 75% (3й квартиль), 100% (максимальное значение). Для серий категориальных данных: число непустых записей, число уникальных категорий, наиболее часто встречающуюся категорию и ее частоту. Результат выполнения этой функции – другая серия (или DataFrame).

```
series_2.describe()
```

Вывод

```
count      5.000000
mean      46.200000
std       38.557749
min       11.000000
25%       15.000000
50%       33.000000
75%       72.000000
max      100.000000
dtype: float64
```

```
series_3.describe()
```

Вывод

```
count       4
unique      2
top       отл
freq        2
Name: Оценки Иванова И.И., dtype: object
```

К сериям применимы различные перегруженные операции, такие как сложение, вычитание, умножение, деление и т.д. Операции осуществляются попарно для соответствующих элементов с одинаковыми индексами. Если уникальный индекс есть только у одной серии, операция будет не определена, в данной позиции будет значение `np.nan`. К серии также применяются векторные

функции Numpy. Дополнительная информация по сериям может быть найдена в официальной документации Pandas.

Dataframe – это двухмерная таблица (строки и столбы). DataFrame может быть создан из одной или нескольких серий, а также по аналогии с сериями – из произвольных массивов и словарей. По умолчанию индексы-наименования строк и столбцов не определены, используются числовые индексы.

```
rnd_numbers = np.random.uniform(0, 1, (2, 5))
df1 = pd.DataFrame(rnd_numbers)
print(df1)
```

Вывод

```
          0         1         2         3         4
0  0.103251  0.304433  0.87911  0.021683  0.600488
1  0.178537  0.038377  0.25801  0.203917  0.128468
```

Использование словарей более распространенный способ: ключи словаря определяют имена столбцов, значения — это массивы данных (серии).

```
dict_marks = {
    "ФИО": ["Иванов", "Петров", "Сидоров"],
    "Математика": [5, 3, 4],
    "Физика": [4, 5, 5],
    "История": [5, 4, 5]
    }
df_marks = pd.DataFrame(dict_marks)
print(df_marks)
```

Вывод

```
       ФИО  Математика  Физика  История
0   Иванов           5       4        5
1   Петров           3       5        4
2  Сидоров           4       5        5
```

Индексы являются массивами и могут иметь уникальное имя для удобства интерпретации имен индексов. Индексы строк хранятся в объекте `df.index`, а имя в `df.index.name`. Индексы столбцов – в `df.columns` и `df.columns.name`. После транспонирования, индексы строк и столбцов меняются местами.

Для доступа к элементам DataFrame применяются следующие операции:

- Два варианта доступа к строке:
  - `df_name.loc` - доступ по индексу-имени,
  - `df_name.iloc` - доступ по числовому индексу.
- Два варианта доступа к отдельному элементу:
  - `df_name.at` - указывается пара строка/столбец по индексу-имени,
  - `df_name.iat` - указывается пара строка/столбец по числовому индексу.

В задачах анализа данных часто требуется выделять не отдельные значения, а часть DataFrame, которая соответствует какому-то логическому запросу (фильтрация данных). Технически, логическое условие определяет маску, использование маски вместо индексов возвращает только те элементы, где маска содержит `True`. Так можно отбирать необходимые строки и столбцы. Если отбор выполняется поэлементно, то возвращается аналогичный по форме DataFrame, где в ячейках, не соответствующих условиям, стоит значение `np.nan`. Данный способ позволяет формировать выборки данных по условиям любой сложности.

```
df_marks[ df_marks['Физика']==5 ]
```

Вывод

	ФИО	Математика	Физика	История
1	Петров	3	5	4
2	Сидоров	4	5	5

Добавить новый столбец, т.е. новую серию данных, можно просто обратившись к пока несуществующему столбцу по имени и присвоив значения новой серии, число строк которой совпадает с числом строк DataFrame.

```
df_marks['Химия'] = [3, 3, 3]
df_marks
```

Вывод

	ФИО	Математика	Физика	История	Химия
0	Иванов	5	4	5	3
1	Петров	3	5	4	3
2	Сидоров	4	5	5	3

Существуют и другие способы добавления данных в DataFrame, например, `pd.concat()` - указать список из нескольких DataFrame (например, исходный и новый из одной строки или столбца), `pd.append()` - указать что добавить: словарь или DataFrame, `pd.DataFrame().loc` - обратиться по индексу следующему за последним индексом исходного массива. Также возможно формирование новых таблиц через перегруженные арифметические, математические или логические операции, например, сумма двух таблиц, максимум из данных двух таблиц и т.д. При применении векторных операций необходимо указывать вдоль каких осей их применять.

2.3.2.2. Импорт данных из файлов

В прикладных задачах анализа данных, аналитик получает данные из какого-либо внешнего источника в одном из популярных форматов. Пакет Pandas имеет встроенные инструменты импорта и экспорта в форматы csv, excel, sql, fwf (Fixed-Width Text File), json, html, xml, clipboard (локальный буфер обмена). Функции импорта начинаются со слова read: `read_csv()`, `read_excel()`, `read_sql()`, `read_fwf()`, `read_json()`, `read_html()`, `read_xml()`, `read_clipboard()` и другие. Для каждой команды чтения есть аналогичная для сохранения в файл заданного типа, начинающаяся со слова to_, например, `to_csv()`, `to_excel()`.

Файл может быть расположен в локальной папке (если не указан полный путь, то файл ищется в той же папке что и текущий проект на Python) или в сети интернет (указать адрес в сети).

Рассмотрим некоторые особенности импорта данных на примере набора данных «Students Performance in Exams» (оценки студентов), представленного на известной площадке для соревнований по анализу данных Kaggle (www.kaggle.com). Исходный набор данных был изменен, чтобы продемонстрировать некоторые важные ситуации, возникающие при импорте.

Данные могут быть импортированы из локального файла и по ссылке в сети интернет. Если файл содержит данные, представленные строго в соответствии с требованием формата, то он будет импортирован корректно, иначе необходимо указывать дополнительные параметры импорта.

Например, в задачах анализа данных наиболее популярен простой текстовый формат csv (Comma-Separated Values), в котором каждая строка файла соответствует строке данных в таблице, а данные разных столбцов разделяются запятой. Разделитель десятичных разрядов – точка. Первая строка содержит заголовки (имена) столбцов, индексы (или имена) строк отсутствуют.

После импорта стоит визуально оценить таблицу, выведя на экран несколько первых и последних строк. Общая структура данных начала и конца таблицы должна совпадать. Для этого используются функции `pd.head(X)` и `pd.tail(X)`, которые показывают `X` первых и последних строк таблицы. По умолчанию `X=5`.

```
# импорт данных из локального файла
data_csv_local = pd.read_csv("C:\my_data\db_students_performance_cor-
rect.csv")
data_csv_local.head(3)
```

Вывод

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93

```
# импорт по ссылке в сети интернет
data_csv_web = pd.read_csv("https://www.kaggle.com/datasets/spscien-
tist/students-performance-in-exams/db_students_performance_correct.csv")
data_csv_web.tail(3)
```

Вывод

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
997	female	group C	high school	free/reduced	completed	59	71	65
998	female	group D	some college	standard	completed	68	78	77
999	female	group D	some college	free/reduced	none	77	86	86

Функция `info()` рассказывает о типах данных и о количестве пропусков. После импорта желательно проверить типы данных и скорректировать, если типы интерпретированы неверно. Например, иногда категории нумеруют вместо того, чтобы использовать имена, такой тип будет импортирован как целочисленный вместо категориального. Часто вместо пропуска числового значения в csv ставят какой-то спец символ, тогда весь столбец будет импортирован как категориальный.

Для набора данных о оценках студентов, результат выполнения функции `info()` показывает, что в DataFrame 1000 записей с индексами от 0 до 999, 3 столбца имеют целочисленные значения, 5 столбцов – категории (тип `object` используется для текстовых значений), занимаемый размер в памяти – 62.6KB. Для каждого столбца указаны его имя из массива индексов-имен, число заполненных значений и тип данных.

```
data_csv_local.info()
```

Вывод

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   gender                       1000 non-null   object
 1   race/ethnicity               1000 non-null   object
 2   parental level of education  1000 non-null   object
 3   lunch                        1000 non-null   object
 4   test preparation course      1000 non-null   object
 5   math score                   1000 non-null   int64
 6   reading score                1000 non-null   int64
 7   writing score                1000 non-null   int64
dtypes: int64(3), object(5)
memory usage: 62.6+ KB
```

Если известны диапазоны значений и их центральные моменты, то стоит проверить корректность данных проанализировав описательную статистику. Функция `describe()` по умолчанию применяется только к числовым данным, для анализа категорий функция вызывается с дополнительным параметром `describe(include=[object])`.

```
data_csv_local.describe()
```

Вывод

```
       math score  reading score  writing score
count  1000.00000    1000.000000    1000.000000
mean     66.08900      69.169000      68.054000
std      15.16308      14.600192      15.195657
min       0.00000      17.000000      10.000000
25%      57.00000      59.000000      57.750000
50%      66.00000      70.000000      69.000000
75%      77.00000      79.000000      79.000000
max     100.00000     100.000000     100.000000
```

```
data_csv_local.describe(include=[object])
```

Вывод

	gender	race/ ethnicity	parental level of education	lunch	test preparation course
count	1000	1000	1000	1000	1000
unique	2	5	6	2	2
top	female	group C	some college	standard	none
freq	518	319	226	645	642

Следующий пример показывает пример импорта данных, в которых некоторые значения отсутствуют (говорят о наличии пропусков). Задача восстановления пропущенных значений – самостоятельная задача анализа данных, в данном разделе не обсуждается. В таблице данных пропуск отображается как значение особого типа `np.nan` (в примере ниже – пропуск в строке 4 в столбце math score), функция `info()` показывает, сколько значений заполнено в каждом столбце (например, в столбце lunch заполнено 944 значения из 1000).

```
data_csv_with_nan = pd.read_csv("db_students_performance_nan_cor-
rect.csv")
data_csv_with_nan.loc[3:5]
```

Вывод

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
3	male	group A	associate's degree	free/reduced	none	47.0	57.0	44.0
4	male	group C	some college	standard	none	NaN	78.0	75.0
5	female	group B	associate's degree	standard	none	71.0	83.0	78.0

```
data_csv_with_nan.info()
```

Вывод

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   gender                       946 non-null    object
 1   race/ethnicity               960 non-null    object
 2   parental level of education  952 non-null    object
 3   lunch                        944 non-null    object
 4   test preparation course      953 non-null    object
 5   math score                   965 non-null    float64
 6   reading score                959 non-null    float64
 7   writing score                953 non-null    float64
dtypes: float64(3), object(5)
memory usage: 62.6+ KB
```

Многие форматы представления данных не имеют внутренних инструментов контроля корректности, в частности csv. Если при формировании файла данных будет допущена ошибка: пропущен разделитель, переход на новую строку до окончания строки таблицы и т.д., то файл будет импортирован «как есть». Для этого после каждого импорта желательно внимательно просмотреть все строки таблицы. Однако для больших данных это затруднительно. Часто достаточно

сравнить структуру данных в первых и последних строках таблицы, в соответствующих столбцах должны быть похожие форматы и содержимое.

В следующем примере в последних строках значение из столбца race/ethnicity переместилось в столбец gender, остальные значения по столбцам тоже сместились. После просмотра строк таблицы (или путем фильтрации) можно обнаружить, что проблема в строке 991, в которой было не введено одно из значений оценок по предметам и разделитель.

```
data_csv_broken = pd.read_csv("db_students_performance_nan_broken.csv")
display(data_csv_broken.head(3))
display(data_csv_broken.tail(3))
```

Вывод

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72.0	72.0	74.0
1	female	group C	some college	standard	completed	69.0	90.0	88.0
2	female	group B	master's degree	standard	none	90.0	95.0	93.0

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
997	group C	high school	free/reduced	completed	59.0	71.0	65.0	female
998	group D	some college	standard	completed	68.0	78.0	77.0	female
999	group D	some college	free/reduced	none	77.0	86.0	86.0	NaN

```
data_csv_broken.loc[990:993]
```

Вывод

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
990	male	group E	high school	free/reduced	completed	NaN	81.0	75.0
991	female	group B	some high school	standard	completed	65.0	82.0	female
992	group D	associate's degree	NaN	none	55.0	76.0	76.0	female
993	group D	bachelor's degree	free/reduced	none	62.0	72.0	74.0	male

По умолчанию, формат csv подразумевает, что первая строка содержит названия столбцов. Иногда данные не содержат названия столбцов, первая строка содержит данные. Часто данные имеют уникальный индекс (id), записанный отдельным столбцом, однако при импорте записям будет присвоен обычный числовой индекс, а уникальный индекс импортируется как столбец данных. Если после импорта обнаружено, что данные из первой строки файла попали в заголовки, то необходимо использовать параметр `header=None`, столбцы получат числовой индекс. Чтобы указать какой из столбцов нужно использовать как индексы строк, используется параметр `index_col=[X]`, где `X` – номер нужного столбца. Лишние столбцы, например, дубли или нерелевантная информация, можно удалить, используя функцию `pd.drop()`.

```
data_csv_no_indexes = pd.read_csv("db_students_performance_complex_in-
dexes_no_columns.csv")
display(data_csv_no_indexes.head(3))

data_csv_with_indexes = pd.read_csv("db_students_performance_complex_in-
dexes_no_columns.csv",
                                    index_col=[1], header=None)
data_csv_with_indexes = data_csv_with_indexes.drop([0], axis=1)
data_csv_with_indexes.index.name = None
display(data_csv_with_indexes.head(3))
```

Вывод

	0	id1	female	group B	bachelor's degree	standard	none	72	72.1	74
0	1	id2	female	group C	some college	standard	completed	69	90	88
1	2	id3	female	group B	master's degree	standard	none	90	95	93
2	3	id4	male	group A	associate's degree	free/reduced	none	47	57	44

	2	3	4	5	6	7	8	9
id1	female	group B	bachelor's degree	standard	none	72	72	74
id2	female	group C	some college	standard	completed	69	90	88
id3	female	group B	master's degree	standard	none	90	95	93

Еще одна из проблем импорта связна с используемыми символами разделителями столбцов в csv и разделителями десятичных разрядов. Формат csv по умолчанию разделяет столбцы запятой, однако также используются точка с запятой, пробел, табуляция и другие. Некоторые приложения ОС MS Windows используют запятую как разделитель десятичных разрядов, однако в языках программирования запятая используется для перечислений (например, параметров функции), а точка – для чисел с плавающей точкой. Для явного указания разделителей в Pandas используются параметры `sep` и `decimal`.

```
data_csv_delim = pd.read_csv("db_students_performance_delim.csv")
display(data_csv_delim.head(3))
data_csv_delim = pd.read_csv("db_students_performance_delim.csv",
sep=";")
display(data_csv_delim.head(3))
```

Вывод

	gender;race/ethnicity;parental level of education;lunch;test preparation course;math score;reading score;writing score
female;group B;bachelor's degree;standard;none;69	59979869372896;72;74
female;group C;some college;standard;completed;59	569645063792336;90;88
female;group B;master's degree;standard;none;66	65791655439577;95;93

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	69,59979869372896	72	74
1	female	group C	some college	standard	completed	59,569645063792336	90	88
2	female	group B	master's degree	standard	none	66,65791655439577	95	93

В DataFrame выше можно заметить, что оценки по математике представлены числом с плавающей точкой, но в ячейке используется запятая как

разделитель знаков (например, в строке 0 – 69,59979869372896). Можно убедиться в некорректности импорта этого столбца, используя функцию `info()`. Столбец math score имеет тип object. После явного указания разделителя, тип данных будет корректный.

```
data_csv_delim.info()
```

Вывод

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   gender                       1000 non-null   object
 1   race/ethnicity               1000 non-null   object
 2   parental level of education  1000 non-null   object
 3   lunch                        1000 non-null   object
 4   test preparation course      1000 non-null   object
 5   math score                   1000 non-null   object
 6   reading score                1000 non-null   int64
 7   writing score                1000 non-null   int64
dtypes: int64(2), object(6)
memory usage: 62.6+ KB
```

```
data_csv_delim = pd.read_csv("db_students_performance_delim.csv",
sep=";", decimal=',')
data_csv_delim['math score'].dtype
```

Вывод

```
dtype('float64')
```

2.3.2.3. Фильтры и группировки

Рассмотрим фильтрацию данных в DataFrame на следующем примере, выведем всех девушек, которые набрали более 95 баллов по математике в порядке убывания баллов по этому предмету.

Логическое условие для столбца формирует маску для строк, где для строк, соответствующих логическому условию, стоит `True`. Маска по запросу: пол = женский:

```
data_csv_local['gender']=='female'
```

Вывод

```
0       True
1       True
2       True
3      False
4      False
       ...
995     True
996    False
997     True
998     True
999     True
Name: gender, Length: 1000, dtype: bool
```

Если подставить маску вместо индексов, то отобразятся только отобранные строки:

```
data_csv_local[data_csv_local['gender']=='female']
```

Вывод

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
5	female	group B	associate's degree	standard	none	71	83	78
6	female	group B	some college	standard	completed	88	95	92
...	...	...	...	...	...	...	...	...
993	female	group D	bachelor's degree	free/reduced	none	62	72	74
995	female	group E	master's degree	standard	completed	88	99	95
997	female	group C	high school	free/reduced	completed	59	71	65
998	female	group D	some college	standard	completed	68	78	77
999	female	group D	some college	free/reduced	none	77	86	86

К полученному DataFrame можно повторно применить новое условие – новую маску. Для рассматриваемого примера применим условие – оценки по математике больше 95:

```
data_csv_local[data_csv_local['gender']=='female'][data_csv_local['math score'] > 95]
```

Вывод

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
114	female	group E	bachelor's degree	standard	completed	99	100	100
165	female	group C	bachelor's degree	standard	completed	96	100	100
179	female	group D	some high school	standard	completed	97	100	100
263	female	group E	high school	standard	none	99	93	90
451	female	group E	some college	standard	none	100	92	97
458	female	group E	bachelor's degree	standard	none	100	100	100
712	female	group D	some college	standard	none	98	100	99
717	female	group C	associate's degree	standard	completed	96	96	99
855	female	group B	bachelor's degree	standard	none	97	97	96
962	female	group E	associate's degree	standard	none	100	100	100

Наконец, применим сортировку в порядке убывания. Итоговый код полного запроса выглядит так:

```
data_csv_local[data_csv_local['gender']=='female'][data_csv_local['math
score'] > 95].sort_values(["math score"], ascending=False)
```

Вывод

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
451	female	group E	some college	standard	none	100	92	97
458	female	group E	bachelor's degree	standard	none	100	100	100
962	female	group E	associate's degree	standard	none	100	100	100
114	female	group E	bachelor's degree	standard	completed	99	100	100
263	female	group E	high school	standard	none	99	93	90
712	female	group D	some college	standard	none	98	100	99
179	female	group D	some high school	standard	completed	97	100	100
855	female	group B	bachelor's degree	standard	none	97	97	96
165	female	group C	bachelor's degree	standard	completed	96	100	100
717	female	group C	associate's degree	standard	completed	96	96	99

Еще один вариант представления информации из DataFrame – это группировка данных по значениям с помощью функции `pd.groupby()`.

Например, сгруппируем по полу, базовому образованию и посчитаем, сколько студентов попало в каждую группу с помощью функции `count()`:

```
data_csv_local.groupby(["gender", "parental level of education"])['math
score'].count()
```

Вывод

```
gender  parental level of education
female  associate's degree             116
        bachelor's degree               63
        high school                     94
        master's degree                 36
        some college                   118
        some high school                91
male    associate's degree             106
        bachelor's degree               55
        high school                    102
        master's degree                 23
        some college                   108
        some high school                88
Name: math score, dtype: int64
```

Сводные данные можно также получить с помощью функций-агрегаторов (используются стандартные агрегаторы Numpy). Для примера посчитаем математическое ожидание и стандартное отклонение оценок по математике в разрезе пол/национальность:

```
agg_functions = {"math score": ["mean", "std"]}
data_csv_local.groupby(["gender", "race/ethnicity"]).agg(agg_functions)
```

Вывод

		math score	
		mean	std
gender	race/ethnicity		
female	group A	58.527778	14.157252
	group B	61.403846	16.256750
	group C	62.033333	15.007224
	group D	65.248062	14.174157
	group E	70.811594	16.269129
male	group A	63.735849	14.520742
	group B	65.930233	14.156928
	group C	67.611511	14.090037
	group D	69.413534	13.094139
	group E	76.746479	14.298570

Другие варианты фильтрации, отбора и группировки данных в DataFrame можно узнать из официальной документации Pandas.

2.3.3. Пакет Matplotlib

Matplotlib — библиотека на языке программирования Python для визуализации данных двумерной и трехмерной графикой. Получаемые изображения могут быть использованы в качестве иллюстраций в публикациях. Matplotlib является гибким, легко конфигурируемым пакетом, который вместе с NumPy, SciPy и IPython предоставляет возможности, подобные MATLAB. Генерируемые в различных форматах изображения могут быть использованы в интерактивной графике, в научных публикациях, графическом интерфейсе пользователя, веб-приложениях, где требуется построение диаграмм [34, 35].

Matplotlib поддерживает многие виды графиков и диаграмм, включая графики функций (line plot), диаграммы рассеяния (scatter plot), столбчатые диаграммы (bar chart), гистограммы (histogram), круговые диаграммы (pie chart), графики 3D поверхностей (surface plot), контурные графики (contour plot), поля градиентов (quiver) и многие другие. Практически все элементы графиков могут быть настроены пользователем, что позволяет помимо встроенных стилей, использовать произвольные авторские.

Графики, построенные в Matplotlib могут быть экспортированы в различные форматы: eps (Encapsulated Postscript), jpg (Joint Photographic Experts Group), jpeg (Joint Photographic Experts Group), pdf (Portable Document Format), pgf (PGF code for LaTeX), png (Portable Network Graphics), ps (Postscript), raw (Raw RGBA bitmap), rgba (Raw RGBA bitmap), svg (Scalable Vector Graphics), svgz (Scalable Vector Graphics), tif (Tagged Image File Format), tiff (Tagged Image File Format).

С помощью Matplotlib можно делать несложные анимации и экспортировать их в форматы mp4 (MPEG-4), gif (Graphics Interchange Format) и другие, при наличии кодеков кодирования.

Импорт пакета, вернее, его важной части для построения графиков `pyplot`, осуществляется командой (будем использовать стандартный псевдоним):

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

Для вывода изображений и графиков используется один из способов, который может реализовать Python в операционной системе. По умолчанию графика выводится Python'ом в отдельном окне (собственное приложение для отображения). При работе с Jupyter Notebook, если мы хотим видеть графику пакета Matplotlib в выводе ячейки, нужно явно указать этот способ с помощью магической команды `%matplotlib inline`. Google colab и реализация Jupyter Notebook в VS Code автоматически выводят результаты в вывод ячейки, "магическую" команду выполнять не обязательно.

2.3.3.1. Анатомия и стилизация графиков Matplotlib

Знание анатомии Matplotlib позволяет получить доступ ко всем элементам графика, добавить, удалить, изменить элементы или создать свои собственные из базовых примитивов. Основные элементы графика Matplotlib показаны на рис. 37 [36].

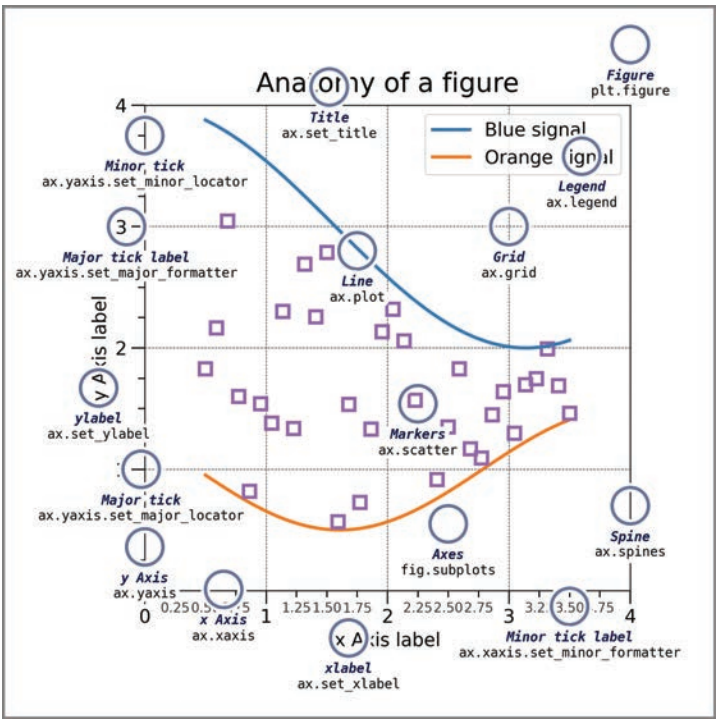


Рис. 37. Анатомия Matplotlib

Все изображение, на котором может быть изображено сразу несколько графиков, задается классом `plt.figure`. Внутри `figure` создается массив графиков `fig.subplots`. Поскольку массив графиков может быть многомерным в терминологии Python размерность массива называют осью и для обозначения элемента массива графиков используют обозначение ax от axis. В тоже время оси графика имеют схожее название `xaxis`, `yaxis`.

Внутри поля конкретного графика могут быть добавлены точки `ax.scatter`, линии `ax.plot`, легенда `ax.legend`, произвольный текст `ax.text`, вспомогательная сетка `ax.grid` и другие элементы. Все элементы имеют тонкую настройку.

Все изображение и каждый график внутри могут иметь заголовки, определяемые в `plt.title` и `ax.set_title`, которые имеют все настройки, относящиеся к текстовым элементам.

Для оформления графика используется рамка с осями, которые называются `ax.spines`. По умолчанию отображаются все четыре, но оформлены только нижняя `ax.xasis` и левая `ax.yaxis`. Оси `x` и `y` могут иметь подписи и настраиваемые деления `major_ticks` и `minor_ticks`.

Некоторые примеры обращения к элементам графика далее будут использоваться в примерах, другие особенности работы с Matplotlib можно узнать из официальной документации.

В большинстве случаев пользователю не нужно тонко настраивать элементы оформления графика, так как Matplotlib уже использует ряд предустановленных стилей оформления. Изменение стиля возможно поэлементно, используя команды `matplotlib.rcParams` или используя наборы стилей `matplotlib.style.use`.

Например, следующая команда изменит толщину и стиль линий на всех отображаемых графиках проекта:

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

mpl.rcParams['lines.linewidth'] = 2
mpl.rcParams['lines.linestyle'] = '--'
```

Следующий пример установит стиль оформления ggplot, используемый в популярном пакете языка R:

```
plt.style.use('ggplot')
```

Список всех предустановленных по умолчанию стилей можно узнать командой:

```
print(plt.style.available)
```

Вывод

```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-nogrid', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn-v0_8', 'seaborn-v0_8-bright', 'seaborn-v0_8-colorblind', 'seaborn-v0_8-dark', 'seaborn-v0_8-dark-palette', 'seaborn-v0_8-darkgrid', 'seaborn-v0_8-deep', 'seaborn-v0_8-muted', 'seaborn-v0_8-notebook', 'seaborn-v0_8-paper', 'seaborn-v0_8-pastel', 'seaborn-v0_8-poster', 'seaborn-v0_8-talk', 'seaborn-v0_8-ticks', 'seaborn-v0_8-white', 'seaborn-v0_8-whitegrid', 'tableau-colorblind10']
```

Пример некоторых стилей показан на рис. 38, слева направо показаны стили classic, ggplot и tableau-colorblind10:

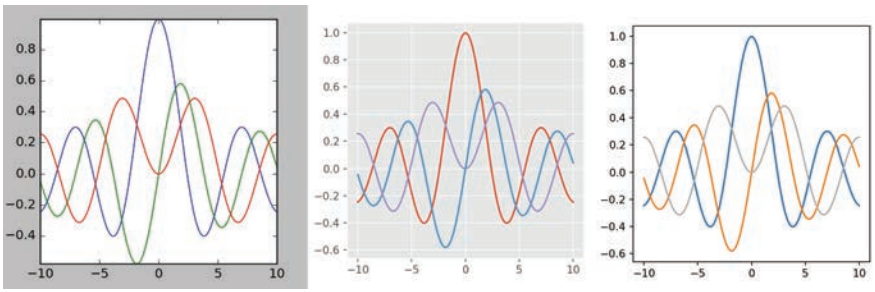


Рис. 38. Стили Matplotlib

Дополнительно в Matplotlib есть возможность использовать стиль XKCD – стиль вэб-комикса, предложенного Рэнделом Мандо в 2005 году для описания будничных эпизодов из жизни программистов, абстрактных понятий математики и информатики, теоретической физики и др. Пример данного стиля показан на рис. 39.

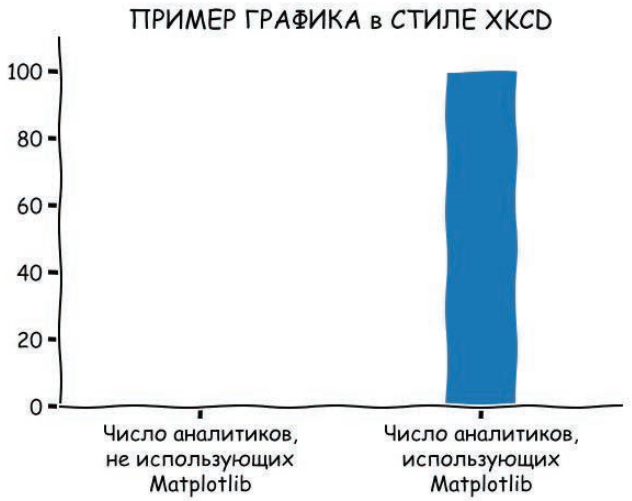


Рис. 39. Стиль XKCD

2.3.3.2. Основы построения графиков

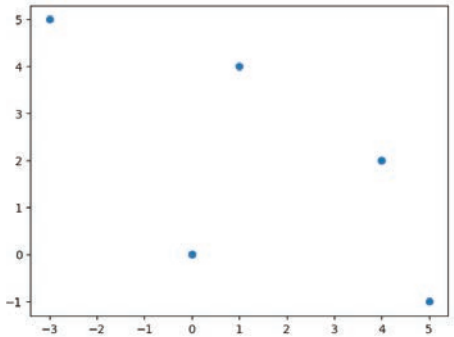
Для быстрого и простого построения графика можно использовать способ прямого высокоуровневого обращения в Matplotlib. Однако, такой способ не рекомендуется использовать, так как доступ к настройкам и элементам графика усложняется. Для создания графика необходимо прописать 2 блока кода: первый описывает необходимые команды для формирования графика, второй – способ его отображения.

```
# Блок 1. Описать элементы графика и все его настройки
plt.scatter([-3, 1, 0, 4, 5], [5, 4, 0, 2, -1])

# Блок 2. Дать команду отобразить изображение или сохранить его в файл
plt.show()
```

Более корректный способ – явно объявить все элементы графика и описывать их свойства путем обращения к соответствующим переменным. И первый, и второй способ дадут одинаковый результат, но второй способ позволяет более гибко настраивать необходимые элементы графика. Для создания графика необходимо прописать 3 блока кода: первый блок описывает переменные для доступа к всей фигуре и всем графикам изображения, размеры и плотность изображения, второй – описание всех элементов графика и их настройки, третий – команда на отображение.

```
# Блок 1. Инициализация figure и массива графиков внутри figure
fig, ax = plt.subplots()

# Блок 2. Описание всех элементов графика путем
# обращения к каждому элементу массива графиков
ax.scatter([-3, 1, 0, 4, 5], [5, 4, 0, 2, -1])

# Блок 3. Команда на отображение изображения
plt.show()
```

Вывод



Рассмотрим пример построения графика функции $y = sin(x)$. Обычно данные импортируются из внешних источников или генерируются алгоритмом в виде пар (x, y). Сгенерируем данные для функции $sin(x)$ по точкам x равномерно распределенным и случайно равномерно распределенным в заданном интервале используя функции Numpy.

Используем в блоке 2 функции scatter и plot для формирования одновременно и точек по парам (x, y), и графика, соединяющего эти точки. Опции set_xlim и set_ylim задают границы графика. По умолчанию границы будут установлены по минимальному и максимальному значению из массивов x и y, поэтому может отобразиться не вся интересующая область определения графика, а только фрагмент.

В блоке 1 при создании массива графиков укажем, что figure имеет 2 столбца (одномерный массив с двумя элементами), параметр `n_cols=2`. По умолчанию размер всего графика задан в дюймах и имеет размер (6.4, 4.8). Поскольку мы отображаем 2 графика на одной фигуре, увеличим ширину изображения с помощью параметра `figsize=(12.8, 4.8)`.

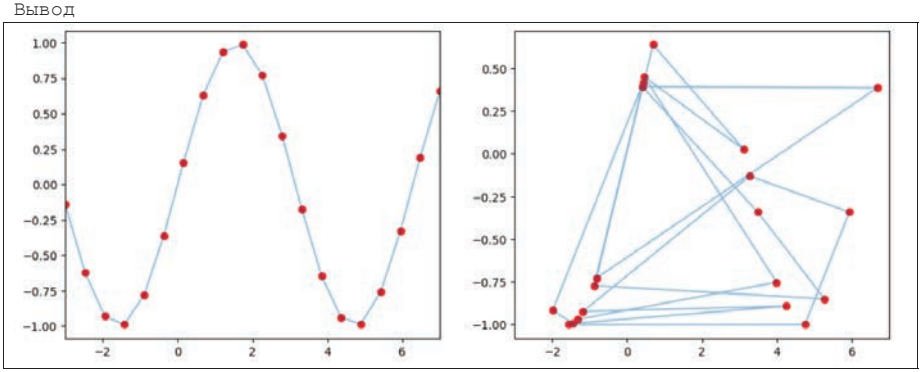
Создаваемая картинка отображается в пикселях монитора с учетом установленного параметра – точек на дюйм, DPI (dots per inch). По умолчанию dpi=100. Качественная картинка с множеством элементов должна иметь высокое разрешение от 300dpi и выше. Плотность 300dpi – минимальный стандарт для публикаций, многие печатные издания требуют плотность 600dpi.

Функция `plot` соединяет пары точек отрезками прямых линий в том порядке, в котором точки встречаются в массиве (x, y). В примере ниже, в первом наборе данных пары сгенерированы в порядке возрастания x, во второй –

значения x сгенерированы случайно, поэтому порядок нарушен, и точки соединяются соответствующим образом.

```
# генерируем массивы точек графика
x_min = -3
x_max = 7
sample_volume = 20
x1 = np.linspace(x_min, x_max, sample_volume)
y1 = np.sin(x1)
x2 = np.random.uniform(x_min, x_max, sample_volume)
y2 = np.sin(x2)

# создание графика
fig, ax = plt.subplots(ncols=2, figsize=(12.8, 4.8))
ax[0].scatter(x1, y1, color='red')
ax[0].plot(x1, y1)
ax[0].set_xlim(x_min, x_max)
ax[1].scatter(x2, y2, color='red')
ax[1].plot(x2, y2)
ax[1].set_xlim(x_min, x_max)
plt.show()
```

Вывод

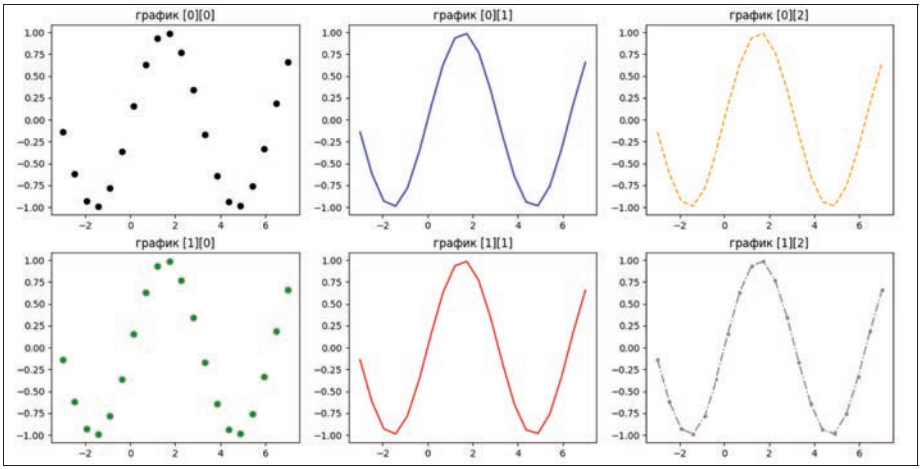


Есть несколько способов создания изображений с несколькими графиками. Рассмотрим наиболее простые и популярные.

Первый способ использует указание числа строк и столбцов (матрицы) графиков. Создание каждого графика – это обращение к переменной двухмерного массива:

```
fig, ax = plt.subplots(figsize=(16,8), ncols=3, nrows=2)

ax[0][0].scatter(x1, y1, color='black')
ax[0][0].set_title("график [0][0]")
ax[1][0].scatter(x1, y1, color='green')
ax[1][0].set_title("график [1][0]")

ax[0][1].plot(x1, y1, color='blue')
ax[0][1].set_title("график [0][1]")
ax[1][1].plot(x1, y1, color='red')
ax[1][1].set_title("график [1][1]")

ax[0][2].plot(x1, y1, '--', color='orange')
ax[0][2].set_title("график [0][2]")
ax[1][2].plot(x1, y1, '-..', color='gray')
ax[1][2].set_title("график [1][2]")

plt.show()
```

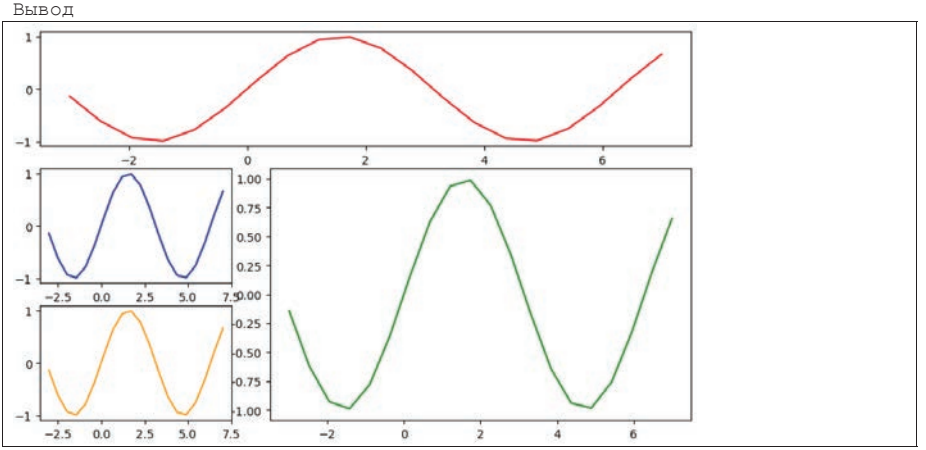
Вывод



Второй способ – создание сетки и объявление переменных графиков путем указания множества соседних ячеек сетки. В примере ниже создается сетка 3 на 3. Первый график – это объединение ячеек первой строки, второй график – объединение ячеек 2й и 3й строк 2го и 3го столбцов, третий и четвертый графики – это отдельные ячейки:

```
fig = plt.figure(figsize=(10, 6))
my_grid = fig.add_gridspec(3,3, figure=fig)
ax1 = fig.add_subplot(my_grid[0, :])
ax2 = fig.add_subplot(my_grid[1:3, 1:3])
ax3 = fig.add_subplot(my_grid[1, 0])
ax4 = fig.add_subplot(my_grid[2, 0])

axs = [ax1, ax2, ax3, ax4]
cs = ['red', 'green', 'blue', 'orange']
for i in range(4):
  axs[i].plot(x1, y1, color=cs[i])

plt.show()
```

Вывод



Рассмотрим пример построения графиков двух функций, на котором отобразим наиболее важные элементы графика и их настройки. При создании контейнера графиков задается размер, чтобы наилучшим образом соответствовать пропорциям графика, а также плотность 300dpi, так как график будет содержать много деталей.

Далее перечисляются 3 серии данных для отображения. Для первой функции будут показаны точки пар (x, y) и линия графика функции, для второй функции – линия. Каждая серия имеет свои настройки отображения. Для точечного графика задаются тип маркера, его цвет и размер, для линий – тип, толщина и цвет линии.

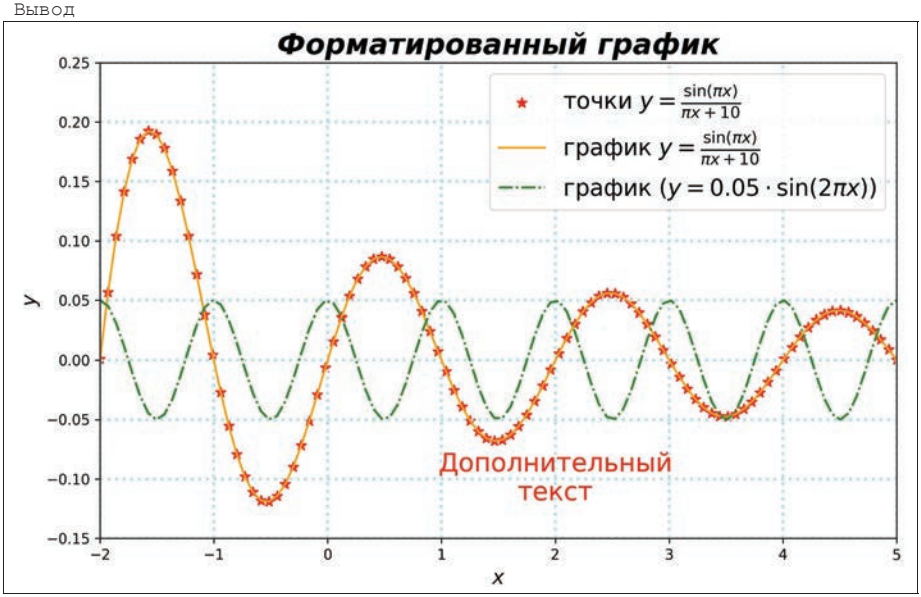
Если у серии задано поле `label`, то его содержимое позже отобразится в легенде. Поле `label` – это строка, символ `r` перед строкой говорит о том, что строка должна читаться «как есть», игнорируя спец символы. В данном примере это нужно, чтобы описать в строке формулу Latex, заключенную в символы `$...$`.

Для описания легенды, заголовка и подписей на осях указываются параметры текста: размер шрифта и стиль. Также показан пример произвольного текста, размещенного в указанной точке графика.

Наконец, добавлена вспомогательная сетка, указаны параметры сетки. Полный код и результат примера:

```
x = np.linspace(-2, 5, 100)
y1 = np.sin(3.14*x) / (3.14*x+10)
y2 = 0.05*np.cos(2*3.14*x)

fig, ax = plt.subplots(figsize=(10, 6), dpi=300)

ax.scatter(x, y1, s=50, color='red', marker='*',
           label=r'точки $y = \frac{\sin (\pi x)}{\pi x + 10}$')
ax.plot(x, y1, color='orange', label=r'график $y = \frac{\sin (\pi
x)}{\pi x + 10}$')
ax.plot(x, y2, linestyle='-.', color='green',
        label=r'график ($y =0.05 \cdot \sin (2 \pi x)$)')

ax.legend(fontsize=16)
ax.set_title('Форматированный график', fontsize=20, style='italic',
weight='bold')
ax.set_xlabel(r'$x$', fontsize=14)
ax.set_ylabel(r'$y$', fontsize=14)

ax.text(2, -0.1, 'Дополнительный\nтекст', fontsize = 18,
        horizontalalignment='center', verticalalignment='center',
        color="red")

ax.grid(color = 'lightblue', linestyle = 'dotted', linewidth = 2)
ax.set_xlim(-2, 5 )
ax.set_ylim(-0.15, 0.25)

plt.show()
```

Вывод



Кроме графиков функций одной переменой, Matplotlib позволяет строить 3D графики поверхностей с помощью специальной библиотеки `mplot3d`.

```
from mpl_toolkits import mplot3d
```

Рассмотрим построение поверхности на примере функции $f(x, y) = x^2$.

График поверхности строится, как и обычный график, по точкам, для чего необходимо задать набор троек (x, y, z). Значения каждой из трех переменных задаются матрицей в узлах сетки на плоскости xy. Сетку с равномерными отсчетами можно создать средствами Numpy, используя функцию `meshgrid`. Пример сетки 3 на 3 в точках 0, 1, 2:

```
x = np.array([0, 1, 2])
y = np.array([0, 1, 2])
X, Y = np.meshgrid(x, y)
print(X)
print(Y)
```

Вывод

```
[[0 1 2]
 [0 1 2]
 [0 1 2]]
[[0 0 0]
 [1 1 1]
 [2 2 2]]
```

Для вычисления функции в точках (x, y) необходимо сформировать пары координат из матриц X и Y (например, с помощью функции `np.vstack`), вычислить значения функций, а далее сформировать матрицу Z в соответствующих узлах сетки с помощью функции `np. reshape`.

Полный код построения поверхности:

```
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
pairs = np.vstack([X.ravel(), Y.ravel()])
z = np.zeros(100*100)
for i in range(pairs.shape[1]):
  z[i] = pairs[0, i]**2
Z = z.reshape( (100, 100) )

fig = plt.figure(figsize=(8,8))
ax = plt.axes(projection ='3d')
ax.plot_surface(X,Y,Z)
plt.show()
```

Вывод

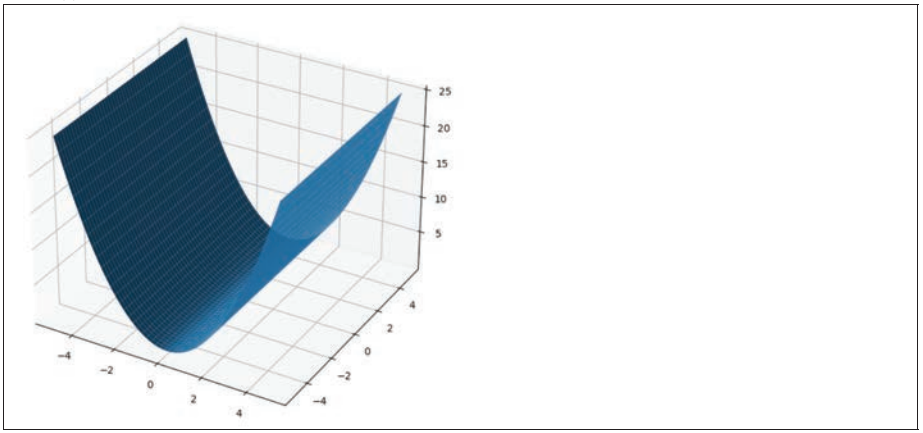


График поверхности – это трехмерная проекция, показанная с одной из точек, в которой установлена «камера». Положение камеры может быть изменено, задав углы azimuth, roll, elevation и расстояние до камеры (рис. 40).

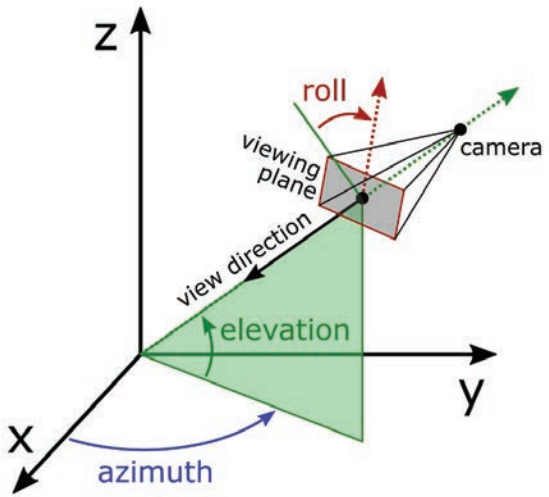
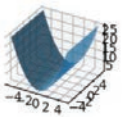


Рис. 40. 3D проекция Matplotlib

Значения параметров установки камеры по умолчанию: `ax.dist=10`, `ax.elev=30`, `ax.azim=-60`. Примеры измененных параметров:

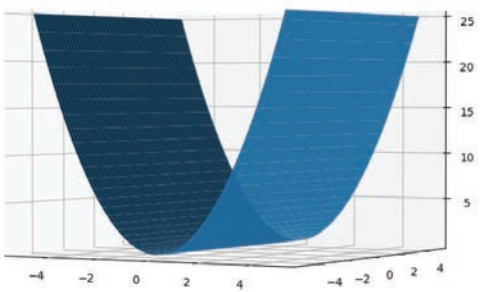
```
fig = plt.figure(figsize=(8,8))
ax = plt.axes(projection ='3d')
ax.dist = 50 # default 10
ax.plot_surface(X,Y,Z)
plt.show()
```

Вывод



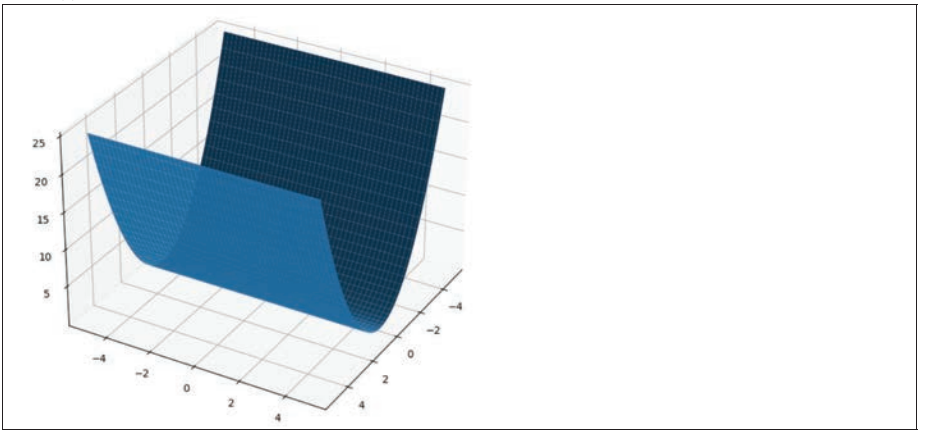
```
fig = plt.figure(figsize=(8,8))
ax = plt.axes(projection ='3d')
ax.elev = 1 # default 30
ax.plot_surface(X,Y,Z)
plt.show()
```

Вывод




```
fig = plt.figure(figsize=(8,8))
ax = plt.axes(projection ='3d')
ax.azim = -60+90 # default -60
ax.plot_surface(X,Y,Z)
plt.show()
```

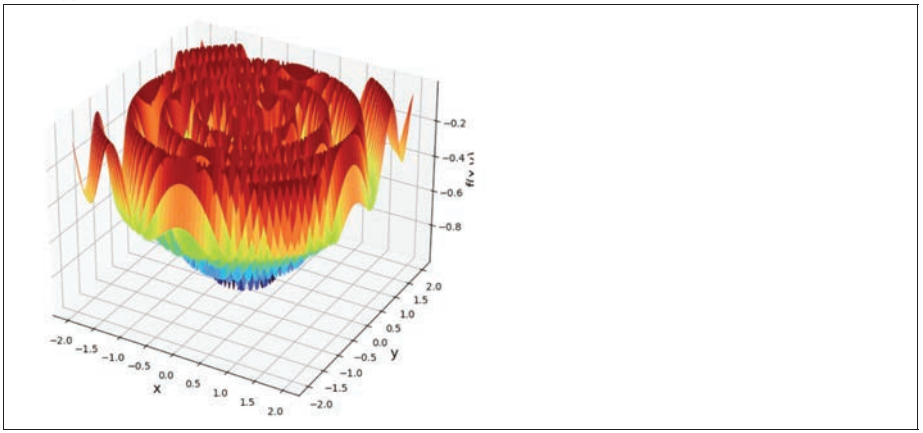
Вывод



Часто дополнительно к графику поверхности строится график линий уровня, которая позволяет получить представление о ландшафте поверхности без подбора положения камеры. Рассмотрим пример построения линий уровня одной из функций двух переменных. График поверхности функции:

```
s = 200
x = np.linspace(-2, 2, s)
y = np.linspace(-2, 2, s)
X, Y = np.meshgrid(x, y)
pairs = np.vstack([X.ravel(), Y.ravel()])
z = np.zeros(s*s)
for i in range(pairs.shape[1]):
  z[i] = -1*( 1 + np.cos(12*np.sqrt(pairs[0, i]**2 + pairs[1, i]**2)) ) /
(0.5*(pairs[0, i]**2 + pairs[1, i]**2)+2)
Z = z.reshape( (s, s) )

fig = plt.figure(figsize=(8,8))
ax = plt.axes(projection ='3d')
ax.plot_surface(X,Y,Z, cmap = plt.cm.turbo)
ax.set_xlabel('x', fontsize=14)
ax.set_ylabel('y', fontsize=14)
ax.set_zlabel('f(x,y)', fontsize=14)
plt.show()
```

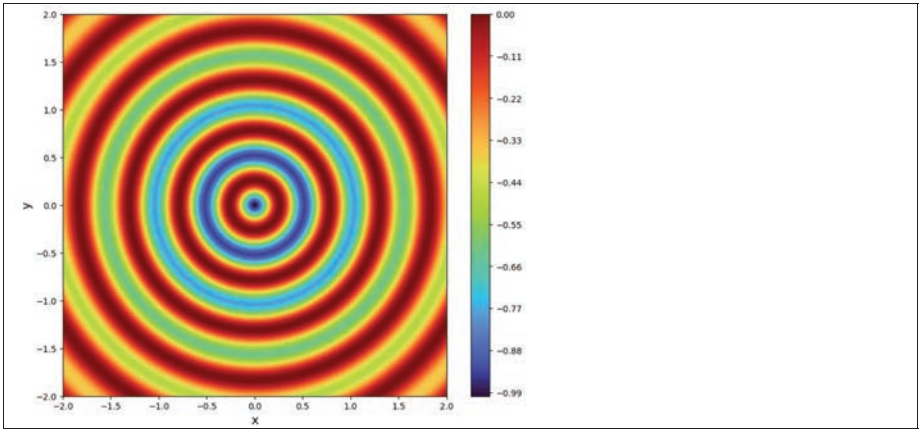
Вывод



Линии уровня:

```
fig = plt.figure(figsize=(10,8))
ax = plt.axes()
c_bar = ax.contourf(X, Y, Z, 100, cmap = plt.cm.turbo)
ax.set_xlabel('x', fontsize=14)
ax.set_ylabel('y', fontsize=14)
fig.colorbar(c_bar)
plt.show()
```

Вывод



2.3.3.3. Инструменты визуального анализа данных

Визуализация данных – это сопоставление данных с графическими элементами, что позволяет на простых графиках отображать большие объемы информации или сложные зависимости в данных. Некоторые инструменты визуализации традиционно используются в математической статистике (например,

гистограммы), другие появлялись позже с развитием компьютерной графики. Рассмотрим наиболее важные и популярные типы графиков и диаграмм.

Пакет Matplotlib позволяет строить любые типы графиков, вплоть до уникальных, созданных пользователем «с нуля». Однако задача предварительной подготовки и обработки данных для построения ложится на плечи пользователя. В тоже время, такие задачи подготовки данных являются типовыми. Пакет Seaborn является надстройкой поверх пакета Matplotlib и предназначен для решения задач визуализации и статистического анализа данных [37].

Seaborn практически полностью наследует синтаксис Matplotlib, за исключением того, что данные по умолчанию должны быть представлены в виде DataFrame. Знание анатомии Matplotlib позволяет получить доступ к любым глубоким настройкам графиков, хотя авторы Seaborn предполагают, что пакет имеет все необходимые предустановленные стили оформления. Далее многие примеры будут продемонстрированы на примере двух пакетов: Matplotlib и Seaborn.

Для установки пакета Seaborn необходимо выполнить:

```
pip install seaborn
```

Рассмотрим инструменты визуализации на примере набора данных задачи машинного обучения «Airfoil Self-Noise», цель которой – построить модель уровня шума, создаваемого крылом самолета [38]. Данные содержат 1503 примера замеров шума разных форм крыла в аэродинамической трубе. Данные содержат 5 независимых переменных (частота в герцах, угол атаки в градусах, длина хорды в метрах, скорость набегающего потока в метрах в секунду, толщина смещения стороны всасывания в метрах) и одну целевую (масштабированный уровень звукового давления в децибелах.).

Пусть данные уже импортированы средствами Pandas в DataFrame с именем raw, имена переменных сохранены в английском написании. Ниже показана описательная статистика набора данных.

```
numerical_names = ['Frequency', 'Angle', 'Chord', 'Free_stream',
'Suction', 'Sound_level']
categorical_names = []
raw.describe()
```

Вывод

	Frequency	Angle	Chord	Free_stream	Suction	Sound_level
count	1486.000000	1451.000000	1456.000000	1426.000000	1449.000000	1503.000000
mean	2878.287349	6.753894	0.136316	50.840042	0.011249	124.835943
std	3137.745026	5.884402	0.093747	15.541817	0.013251	6.898657
min	200.000000	0.000000	0.025400	31.700000	0.000401	103.380000
25%	800.000000	2.000000	0.050800	39.600000	0.002535	120.191000
50%	1600.000000	5.400000	0.101600	39.600000	0.004957	125.721000
75%	4000.000000	9.900000	0.228600	71.300000	0.016104	129.995500
max	20000.000000	22.200000	0.304800	71.300000	0.058411	140.987000

Первый способ визуализировать описательную статистику – использование графиков box-plot (другие названия: ящик с усами, диаграмма размаха, усиковая диаграмма, коробчатая диаграмма, блочная диаграмма с ограничителями выбросов, box-and-whiskers diagram, box-and-whiskers plot). График «ящик с усами», или «ящичковая диаграмма», был разработан Джоном Тьюки в 1970-х годах [39].

График строится на основе порядковой статистики, его связь с нормальным законом распределения показана на рис. 41.

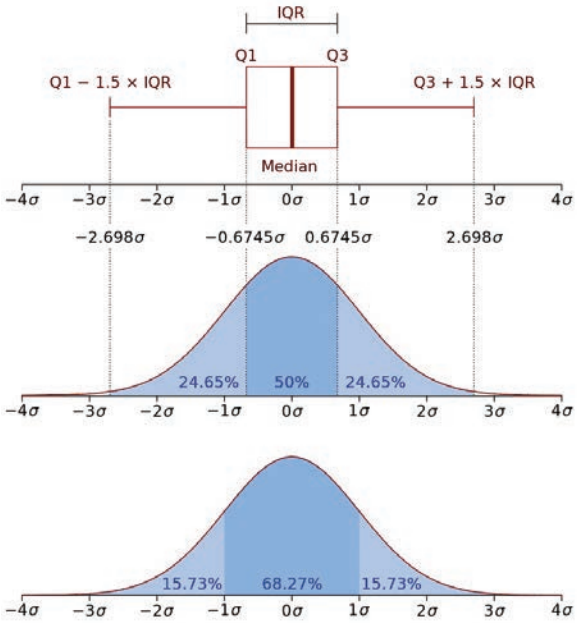
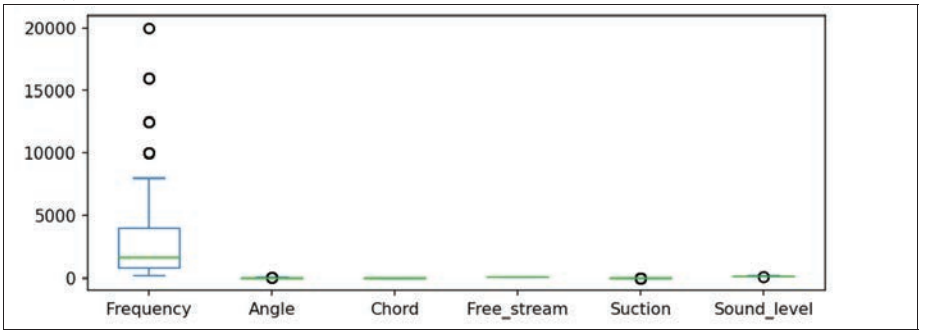


Рис. 41. Принципы построения графика Box-plot [40]

Воспользуемся возможностью доступа к функциям Matplotlib из пакета Pandas с помощью df.plot и отобразим диаграммы размаха для всех переменных задачи:

```
fig, ax = plt.subplots(figsize=(8,3), dpi=150)
raw.plot(kind='box', ax=ax)
plt.show()
```

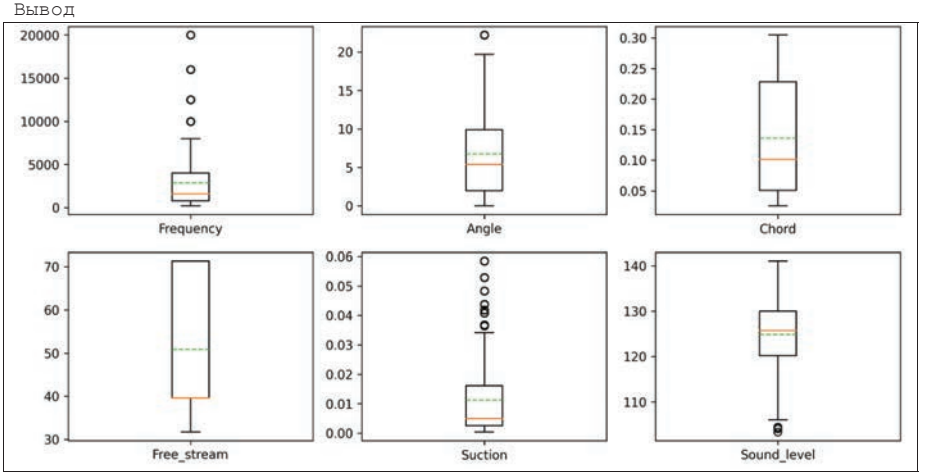
Вывод



Однако такой способ не всегда удобен, так как переменных может быть много, и они выражены в разных масштабах, из-за чего часть ящиков-с-усами будет сложно рассмотреть. Построим график по каждой переменной отдельно и отобразим их на одном изображении. Дополнительно покажем на графиках положение оценки математического ожидания (по умолчанию на графиках box-plot не показывается):

```
fig, ax = plt.subplots(figsize=(12,6), ncols=3, nrows=2)
for i in range(2):
  for j in range(3):
    ax[i][j].boxplot(raw[numerical_names[j + 3*i]][~np.isnan(raw[numeri-
cal_names[j + 3*i]])],
                     labels = [numerical_names[j + 3*i]],
                     showmeans=True, meanline=True)
plt.show()
```

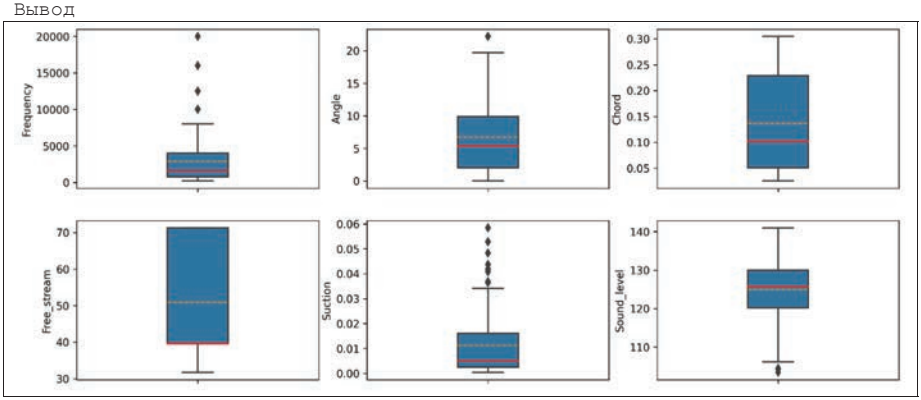
Вывод



Аналогичное построение средствами Seaborn.

```
fig, ax = plt.subplots(figsize=(12,6), ncols=3, nrows=2)
for i in range(2):
  for j in range(3):
    sns.boxplot(y=raw[numerical_names[j + 3*i]],
                showmeans=True, meanline=True,
                meanprops={'color': 'orange', 'ls': '--', 'lw': 1},
                medianprops={'color': 'red', 'ls': '-', 'lw': 1.5},
                ax=ax[i][j], width=0.25)
plt.show()
```

Вывод

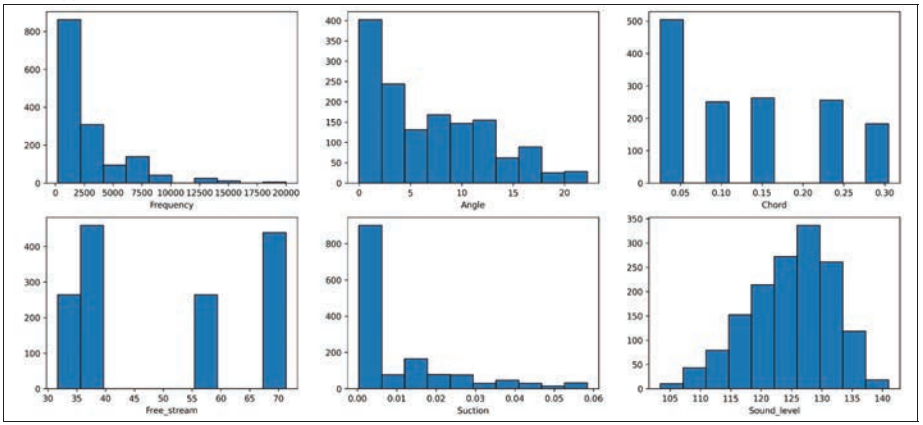


Другой важный инструмент анализа – это гистограммы [41]. Гистограммы позволяют оценить функцию плотности распределения данных. При построении пользователь должен задать число корзин (bins) или их ширину, т.е. диапазоны в области значений переменной, в которых ведется подсчет частоты попадания данных.

Гистограммы Matplotlib отображают по вертикальной оси число попаданий в интервал, число корзин по умолчанию равно 10:

```
fig, ax = plt.subplots(figsize=(18,8), ncols=3, nrows=2)
for i in range(2):
  for j in range(3):
    ax[i][j].hist(raw[numerical_names[j + 3*i]][~np.isnan(raw[numeri-
cal_names[j + 3*i]])],
                  edgecolor="black")
    ax[i][j].set_xlabel(numerical_names[j + 3*i])
plt.show()
```

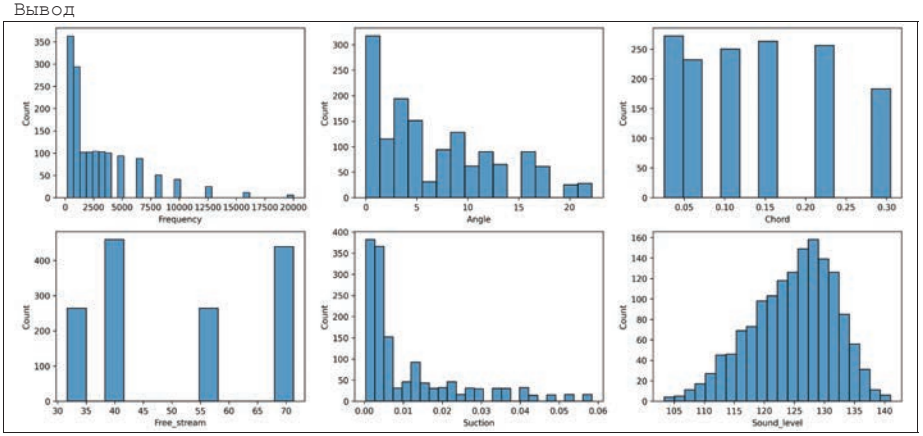
Вывод



Гистограммы Seaborn по умолчанию используют 20 корзин и тоже показывают число попаданий в интервал. Особенностью применения функций в Seaborn является указание источника данных (`data=raw`) и конкретного столбца данных для отображения:

```
fig, ax = plt.subplots(figsize=(18,8), ncols=3, nrows=2)
for i in range(2):
  for j in range(3):
    sns.histplot(data=raw, x=numerical_names[j + 3*i], ax = ax[i][j])
plt.show()
```

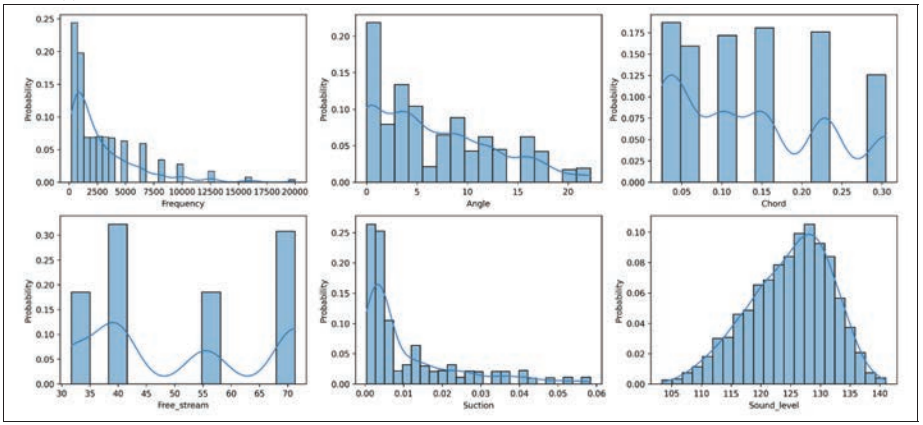
Вывод



В отличии от Matplotlib, Seaborn позволяет отобразить столбцы гистограммы как вероятности (значение от 0 до 1, сумма всех вероятностей на 1), а также построить график оценки плотности распределения вероятностей с помощью метода ядерных оценок (KDE):

```
fig, ax = plt.subplots(figsize=(18,8), ncols=3, nrows=2)
for i in range(2):
  for j in range(3):
    sns.histplot(data=raw, x=numerical_names[j + 3*i], ax = ax[i][j],
                 kde=True, stat='probability')
plt.show()
```

Вывод

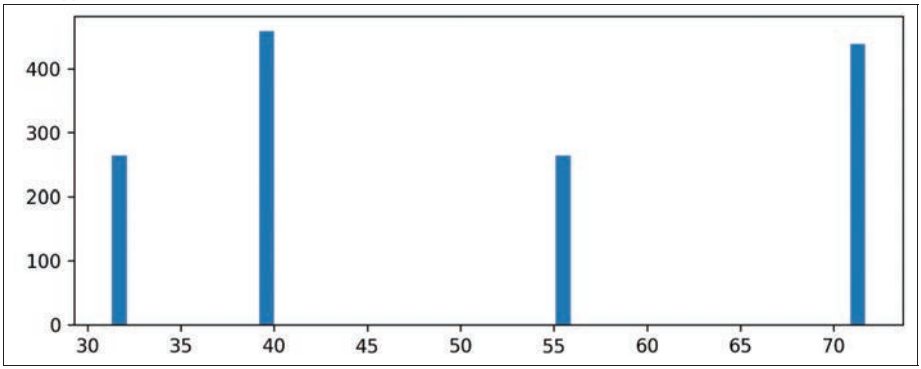


Для отображения категориальных данных можно использовать графики Bar-plot (столбчатая диаграмма). В нашем наборе данных переменная «Free_stream» имеет всего 4 уникальных значения, можно предположить, что эта переменная категориальная. На графике bar-plot можно визуально оценить, как часто встречаются категории.

Matplotlib имеют функцию построения столбиковой диаграммы, но подсчет высоты столбиков (частоты значений категорий) должен сделать пользователь:

```
raw = raw.astype({"Free_stream":'category'})

fig, ax = plt.subplots(figsize=(8, 3), dpi=300)
names = raw["Free_stream"].value_counts().index.values
values = raw["Free_stream"].value_counts().values
ax.bar(names, values)
plt.show()
```

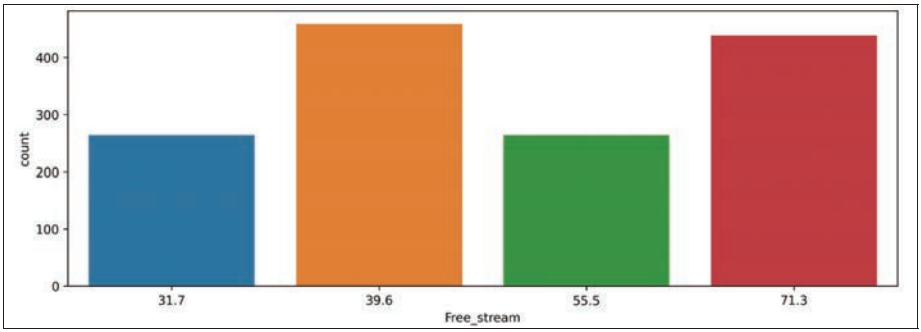
Вывод



Seaborn сам осуществляет подсчет частоты значений категорий и вычисляет их имена из набора данных:

```
fig, ax = plt.subplots(figsize=(12,4), dpi=300)
sns.countplot(data=raw, x="Free_stream",
                ax=ax)
plt.show()
```

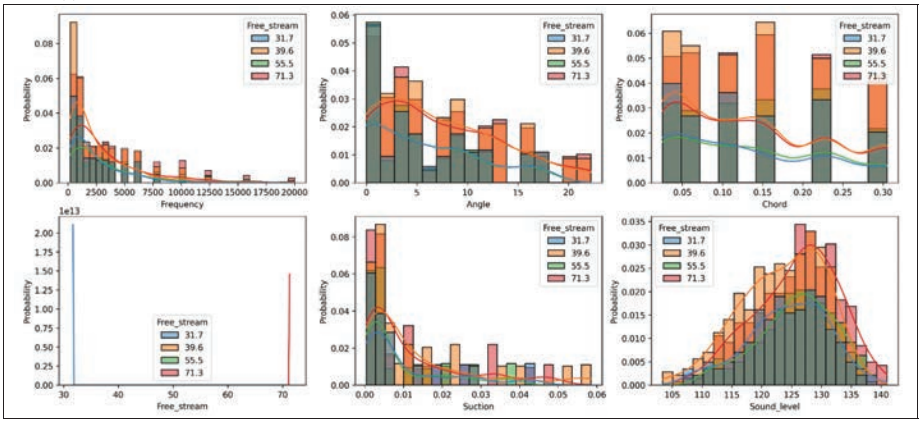
Вывод



При наличии категориальных данных, графики распределений можно построить для каждой категории чтобы сравнить типы распределений. Seaborn позволяет сделать это указав всего один параметр hue:

```
fig, ax = plt.subplots(figsize=(18,8), ncols=3, nrows=2)
for i in range(2):
  for j in range(3):
    sns.histplot(data=raw, x=numerical_names[j + 3*i], ax = ax[i][j],
                 kde=True, stat='probability',
                 hue='Free_stream')
plt.show()
```

Вывод



Другие инструменты визуализации Seaborn можно найти в официальной документации и в галерее на сайте [37].

Контрольные вопросы

1. Чем отличаются интерпретируемые языки программирования от компилируемых?
2. В чем суть режима работы REPL?
3. Перечислите облачные сервисы для работы с Python.
4. Каким образом в Python оформляются блоки кода?
5. В чем смысл динамической типизации в Python?
6. Для чего используются виртуальные окружения в Python?
7. Какие преимущества дает использование IDE?
8. Назовите наиболее популярные IDE для Python.
9. Что такое Jupyter Notebook и чем он отличается от обычных проектов Python?
10. Для чего используется языки markdown и latex?
11. Какие команды называются магическими (magics)?
12. Назовите три основные библиотеки, применяемые в анализа денных на Python.
13. В чем преимущество векторов numpy перед обычными списками list?
14. Что называют осями и формой в массивах numpy?
15. Назовите отличия в работе функций np.arange и np.linspace.
16. Приведите пример функции numpy, в которой указание оси меняет результат выполнения.
17. Какую структуру данных называют кадром данных?
18. На каких библиотеках базируется пакет Pandas?
19. Какие форматы импорта данных поддерживает Pandas?

20. Перечислите основные типы графиков, которые можно создавать в Matplolib.

21. В чем преимущество использования пакета Seaborn вместо Matplotlib?

22. Почему нельзя строить графики функций по неотсортированным данным?

23. Какие настройки камеры можно изменить при построении 3х мерных графиков?

24. Что общего между графиками box-plot и гистограммами?

25. Что дает параметр hue в функциях Seaborn?

Глава 3. Разведочный анализ данных EDA на Python

3.1. Основы разведочного анализа данных

Классическая статистика фокусировалась почти исключительно на статистическом выводе – иногда сложном наборе процедур для получения заключения о крупных популяциях, основываясь на малых выборках. В 1962 году Джон У. Тьюки призвал к реформированию статистики в своей концептуальной работе «Будущее анализа данных». Он предложил новую научную дисциплину под названием «анализ данных», которая включала статистический вывод как всего лишь один из компонентов [42].

Разведочный (исследовательский) анализ данных (Exploratory data analysis, EDA) как самостоятельная область сформировалась во многом благодаря книге Тьюки 1977 года «Разведывательный анализ данных» [43], в которой он показал, что простые графики и диаграммы совместно с описательной и сводной статистикой помогают лучше понимать большие наборы данных.

При решении прикладных задач интеллектуального анализа данных у исследователя могут возникнуть следующие вопросы:

– Как убедиться в том, что мы уже готовы использовать алгоритмы машинного обучения в задаче?

– Как выбрать наиболее эффективные алгоритмы машинного обучения для анализируемого набора данных?

– Как определить целевые переменные и информативные независимые признаки, которые можно использовать для машинного обучения?

Считается, что EDA помогает ответить на все эти и другие вопросы, что позволяет существенно повысить эффективность решения задач анализа данных и построения моделей машинного обучения. EDA позволяет обобщать и визуализировать данные, что обеспечивает глубокое вникание в анализируемые данные и близкое знакомство с их важными характеристиками.

EDA стоит использовать не только на предварительных этапах подготовки к анализу данных, но и на всех последующих:

1. Этап исследования данных: визуализация, поиск пропущенных значений, анализ корреляций.
2. Этап очистки данных: оценить эффективность повышения качества данных.
3. Этап построения моделей: визуализация результатов, анализ эффективности модели, анализ ошибок прогнозирования, анализ ROC и др.
4. Этап представления результатов: презентационная графика для объяснения моделей и результатов.

Исследовательский анализ данных EDA ценен для проектов по анализу данных, поскольку он повышает уверенность в том, что результаты будут адекватными постановке задачи, правильно интерпретированными и применимыми в прикладном смысле (например, в контексте бизнес-приложений). Высокий уровень достоверности может быть достигнут только после проверки «сырых»

необработанных данных и поиска аномалий в значениях признаков. Такой анализ гарантирует, что набор данных собран без ошибок и может быть использован при построении моделей. EDA также помогает находить новые гипотезы о данных, которые не были очевидны на момент постановки задачи анализа данных, но могут быть полезны с точки зрения практического приложения.

На практике, компании, которые только начинают использовать технологии Data Science, часто сталкиваются с ситуацией, когда они понимают, что у них много данных и нет представления о том, какую ценность эти данные могут принести для принятия их бизнес-решений, EDA помогает ответить и на этот вопрос.

В задачах с большим набором признаков, EDA применяется для определения и уточнения выбора информативных признаков, которые далее будут использоваться в моделях машинного обучения. Как правило, после того как специалисты по анализу данных глубоко знакомятся с набором данных, им часто приходится возвращаться к этапу проектирования признаков, так как исходные признаки могут оказаться несоответствующими сформулированным целям анализа.

Отказ от проведения EDA несет следующие риски:

– Создание неточных моделей.
– Создание точных моделей на неверных данных (решается не та задача).
– Выбор некорректных переменных для построения модели.
– Неэффективное использование ресурсов: лишние признаки, лишние данные, необоснованно сложные модели, многократное перепостроение моделей и т.д.

На практике полезно изучить каждый набор данных, используя несколько методов исследования, и сравнить результаты. Изучение одного и того же набора данных с разных точек зрения, акцентируясь на разных аспектах позволяет построить целостную картину данных и открыть нетривиальные свойства данных и зависимостей в данных. EDA выполняется с использованием следующих основных подходов:

– одномерная визуализация,
– двумерная визуализация,
– многомерная визуализация,
– снижение размерности.

Одномерная визуализация предоставляет сводную статистику для каждого поля в наборе необработанных данных, показывает свойства распределения значений каждого признака.

Двумерная визуализация выполняется, чтобы найти взаимосвязь между каждой переменной в наборе данных и интересующей целевой переменной, а также для поиска сильно коррелированных признаков.

Многомерная визуализация выполняется для понимания взаимодействий между различными полями в наборе данных.

Снижение размерности помогает понять поля данных, на которые приходится наибольшее расхождение между наблюдениями, и позволяет обрабатывать уменьшенный объем данных. В некоторых случаях удается построить

адекватную визуализацию многомерных данных, путем снижения размерности до 2 или 3 главных компонент, сохранив большой процент объясненной дисперсии.

В случае обнаружения пропусков в данных, проверяются распределения значений каждого признака и оцениваются возможные способы заполнения пропусков.

При визуальном анализе аналитик по данным проверяет гипотезы и выявляет паттерны в данных, которые позволят лучше понять задачу и выбрать модель, а также подтверждает, что данные обладают необходимыми качествами для построения моделей машинного обучения.

EDA реализует два типа анализа данных:

1. Подтверждающий анализ данных – проверка типовых гипотез о свойствах данных и гипотез, сформулированных на этапе постановки задачи до того, как данные были собраны.
2. Исследовательский анализ данных – глубокое проникновение в исходные данные и поиск новых свойств, нетривиальных гипотез.

В свою очередь, выделяют четыре основных задачи EDA:

1. обнаружение закономерностей,
2. обнаружение аномалий,
3. построение гипотез,
4. проверка предположений и гипотез.

Общая процедура EDA включает следующие этапы:

1. Идентификация переменных и типов данных.
2. Анализ основных метрик.
3. Неграфический одномерный анализ.
4. Графический одномерный анализ.
5. Двумерный анализ.
6. Преобразования переменных.
7. Обработка отсутствующих значений.
8. Обработка выбросов.
9. Корреляционный анализ.
10. Уменьшение размерности.

Поскольку EDA – это исследование, оно не гарантирует, что какие-то новые свойства данных будут выявлены, однако, как говорилось выше, риск пропустить такие свойства высок и может иметь серьезные последствия при построении моделей машинного обучения.

Некоторые этапы EDA могут варьироваться в зависимости от сформулированной задачи анализа: общий анализ, задача классификации, регрессии и т.д. Некоторые этапы могут быть пропущены, другие выполняться многократно с новыми гипотезами или преобразованными данными.

Применяемые инструменты EDA могут варьироваться в зависимости от типов данных. Существуют два базовых типа структурированных данных: числовой и категориальный:

– Числовые данные имеют две формы: непрерывную и дискретную.

– Категориальные данные принимают только фиксированный набор значений.

– Отдельно выделяют двоичные данные как особый тип категориальных данных.

– Также полезно отличать категориальные данные, в которых категории упорядочены.

Есть ряд причин анализировать и, в случае необходимости, преобразовывать типы данных. Для целей анализа данных и предсказательного моделирования тип данных играет важную роль, помогая определять тип визуального изображения, методы анализа данных и построения моделей. В вычислительных системах данные, как правило, классифицируются по типу (int, float, str, category), неверно выбранный тип приведет к некорректному использованию функций языка программирования.

Несмотря на то какие типы данных используются, традиционный способ их представления и хранения – это прямоугольные таблицы, в частности, рассмотренный выше тип DataFrame.

3.1.1. Немного об оценках и визуализации

Поскольку EDA имеет дело с большими объемами данных, а сами данные могут иметь множество значений или бесконечно много в случае непрерывных данных, то для исследования данных часто полезно знать «типичное», «обычное», «нормальное» для данного набора значение, а также вариацию значений данных, т.е. насколько сильно данные отличаются от их нормального значения. Для дискретных и категориальных данных число разных значений обычно невелико, поэтому обычно изучают какие значений встречаются чаще других. В общем случае, аналитику полезно иметь простые показатели, число которых невелико и которые характеризуют весь набор данных.

В математической статистике термин «оценка» часто используется для значений, вычисляемых из выборки данных, которые подвергаются анализу. Основная сложность заключается в том, что набор данных – это лишь часть всей совокупности данных, которые нам обычно недоступны. Часто выводы, полученные по части данных, могут быть неверными. Методы математической статистики использует специальные методы для построения выводов о всей совокупности данных по малой выборке. При этом, если по выборке удается сделать точные выводы, то выборка называется репрезентативной. Во многих случаях, репрезентативность выборки можно проконтролировать в момент сбора данных.

С практической точки зрения, часто полезно вычислять показатели, описывающие данные какими-либо метрическими показателями (так называемые метрики). Основная идея – сформулировать конкретные прикладные критерии

качества, которые могут не совпадать со строгими математическими оценками. Иногда неточные выводы в смысле математической статистики могут быть адекватными для решения прикладных задач, а математически точная модель, наоборот, может оказаться неадекватной на практике.

3.1.1.1. Среднее и нормальный закон

Для количественных данных в EDA пытаются определить «типичное» значение для каждого признака, т.е. оценки того, где расположено большинство данных, так называемая центральная тенденция. Самый простой вариант обобщения данных – взять среднее арифметическое данных.

Важно понимать, что данные должны обладать таким свойством, что какое-то значение встречается чаще других, и чем сильнее отклонение от этого типичного значения, тем реже такие данные встречаются в наборе. Поэтому оценка среднего не всегда является хорошей мерой центрального значения. В прикладной статистике используются разные альтернативные оценки среднего значения.

Оценка среднего должна определяться в предположении, что данные распределены нормально или близко к нормальному закону распределения. Плотность нормального (гауссова) распределения задается как:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

где μ и σ – параметры распределения, а именно: математическое ожидание и среднеквадратичное отклонение. При этом стандартным нормальным распределением называется нормальное распределение с математическим ожиданием $\mu = 0$ и стандартным отклонением $\sigma = 1$.

Важнейшим свойством нормального закона в EDA является «правило трех сигм», которое гласит: практически все значения нормально распределенной случайной величины лежат в интервале: $(\mu - 3\sigma, \mu + 3\sigma)$ или приблизительно с вероятностью 0.9973 значение нормально распределенной случайной величины лежит в указанном интервале (рис. 42).

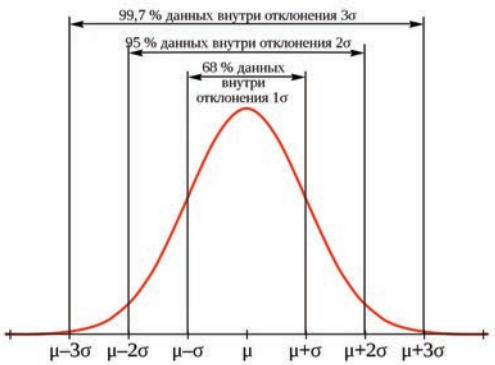


Рис. 42. Правило трех сигм в нормальном законе

Для нормально распределенных данных среднее позволяет оценить математическое ожидание, для других распределений среднее может не нести никакой полезной информации и искажать выводы.

Оценка среднего вычисляется как:

$$mean(x) = \overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

где n обозначает общее число записей или наблюдений в наборе. В статистике это обозначение используется с заглавной буквы N если оно обозначает популяцию, и строчной n, если оно обозначает выборку из популяции.

Среднее значение часто может быть сильно искажено, если в наборе есть несколько (иногда мало) резко отличающихся значений, которые называются выброси или аномалии (outliers).

Выброс – это любое значение, которое сильно дистанцировано от других значений в наборе данных. Выброс как таковой не делает значение данных недопустимым или ошибочным. Выбросы должны быть выявлены и обычно заслуживают дальнейшего расследования. В задаче обнаружении аномалий целевыми объектами являются именно выбросы, и значительный массив данных преимущественно служит для определения «нормы», с которой соразмеряются аномалии.

Для устранения влияния потенциальных выбросов используют робастные (устойчивые) оценки. Например, усеченное среднее (trimmed mean), которое вычисляется путем отбрасывания фиксированного числа сортированных значений с каждого конца последовательности и затем взятия среднего арифметического оставшихся значений. Если отсортировать значения от меньшего к большему $x_{(1)}, x_{(2)}, \dots, x_{(n)}$, то

$$trimmed\_mean(x) = \frac{1}{n-2p}\sum_{i=p+1}^{n-p} x_{(i)}$$

Усеченное среднее устраняет влияние предельных значений, оно получило широкое распространение и во многих случаях является более предпочтительным, чем обычное среднее. Например, в международных спортивных соревнованиях верхние и нижние баллы судей отбрасываются, и итоговым баллом считается среднеарифметический балл оставшихся судей.

Если некоторые значения более переменчивы, чем другие, и сильно переменчивым наблюдениям придается меньшее внимание при анализе, или собранные данные неодинаково представляют разные изучаемые группы, то часто вычисляют средневзвешенное значение (weighted mean):

$$weighted\_mean(x) = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

где w_i – вес каждого значения x_i из набора.

Другая и более популярная в EDA оценка центрального положения – это медиана. Медиана — это число, расположенное в сортированном списке данных посередине списка. По сравнению со средним, в котором используются абсолютно все наблюдения, медиана зависит только от значений в центре сортированных данных. Медиана называется робастной оценкой центрального положения, поскольку она не находится под влиянием выбросов (предельных случаев), которые могут исказить результаты.

Аналогично среднему взвешенному, можно вычислить и взвешенную медиану. Как и обычная медиана, взвешенная медиана робастна к выбросам.

3.1.1.2. Оценки вариабельности и порядковая статистика

Центральное положение – это всего лишь одна из размерностей в обобщении исследуемого признака. Вторая размерность – это вариабельность, именуемая также дисперсностью, которая показывает, сгруппированы ли значения данных плотно, или же они разбросаны относительно центрального значения.

В математической статистике вариабельность изучает дисперсионный анализ, ANOVA (ANalysis Of VAriance), цель которого измерение, уменьшение, различение случайной вариабельности от реальной, идентификация различных источников реальной вариабельности и принятие решений в условиях ее присутствия.

Наиболее широко используемые оценки вариабельности основаны на разницах, или отклонениях, между оценкой центрального положения и наблюдаемыми данными. Эти отклонения говорят о том, насколько данные разбросаны вокруг центрального значения.

Среднее абсолютное отклонение (mean absolute deviation):

$$mean\_absolute\_deviation(x) = \frac{1}{n}\sum_{i=1}^{n}|\overline{x} - x_i|$$

Самыми известными оценками вариабельности являются дисперсия и стандартное отклонение, которые основаны на квадратических отклонениях. Дисперсия – это среднее квадратических отклонений, стандартное отклонение – квадратный корень из дисперсии.

$$variance = s^2 = \frac{1}{n}\sum_{i=1}^{n}(\overline{x} - x_i)^2$$

Стандартное отклонение интерпретируется намного проще, чем дисперсия, поскольку оно находится на той же шкале измерения, что и исходные данные.

$$std = s = \sqrt{variance}$$

Ни дисперсия и стандартное отклонение, ни среднее абсолютное отклонение не устойчивы к выбросам и предельным значениям. Дисперсия и стандартное отклонение чувствительны к выбросам больше всего, поскольку они основаны на квадратических отклонениях. Робастная оценка может быть получена по аналогии с медианой:

$$MAD = median(|x_1 = m|, |x_2 - m|, \dots, |x_n - m|)$$

где $m = median(x)$.

Другой подход к оцениванию дисперсии основан на рассмотрении разброса сортированных данных или их спреда. Статистические показатели на основе сортированных (ранжированных) данных называются порядковыми статистиками (order statistics).

Элементарная мера — это размах (range): разница между самым крупным и самым малым числом. Минимальные (min) и максимальные (max) значения как таковые полезно знать, поскольку они помогают выявлять выбросы, но размах чрезвычайно чувствителен к выбросам и не очень полезен в качестве общей меры дисперсности в данных. Во избежание чувствительности к выбросам мы можем обратиться к размаху данных после отбрасывания значений с каждого конца. Эти типы оценок формально основываются на разницах между процентилями. В наборе данных P-й процентиль является таким значением, что, по крайней мере, P процентов значений принимает это значение или меньшее, и по крайней мере (100 − P) процентов значений принимает это значение или большее. Общепринятой мерой вариабельности является разница между 25-м и 75-м процентилями, которая называется межквартильным размахом (interquartile range IQR).

Визуализацией порядковых статистик является график box-plot, рассмотренный ранее, а сама статистика может быть легко получено с помощью функции `pd.describe()` в Pandas.

Соответствие распределения данных нормальному, оценка центрального положения и вариабельности может быть получена визуально с помощью гистограмм.

3.1.1.3. Разведывание категориальных данных

Категории могут представлять: четко различимые объекты (мужчина и женщина, больной и здоровый, истина и ложь), уровни факторной переменной (низкий, средний и высокий) или числовые данные, которые были разбиты на корзины или частотные интервалы. В случае категориальных данных представление о них дают простые доли или процентные соотношения.

Мода – это значение (или значения в случае, если они являются одинаковыми), которое появляется в данных чаще других. Мода представляет собой простую сводную статистику для категориальных данных, а для числовых данных обычно не используется.

Столбиковые диаграммы – это общепринятый визуальный инструмент для отображения одной-единственной категориальной переменной. Круговые диаграммы являются альтернативой столбиковым диаграммам, но считаются менее информативными.

Рассмотрим некоторые примеры визуализации категориальных данных (рис. 43–52).

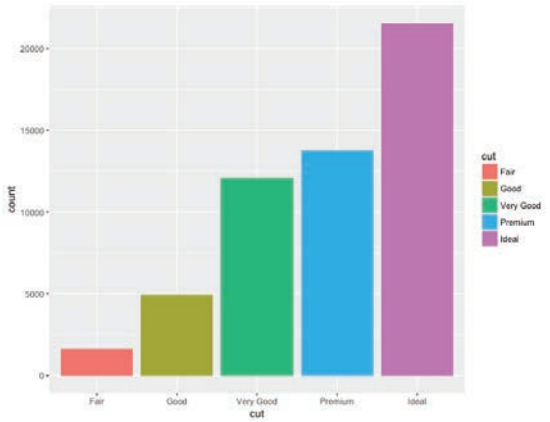


Рис. 43. Традиционный график box-plot

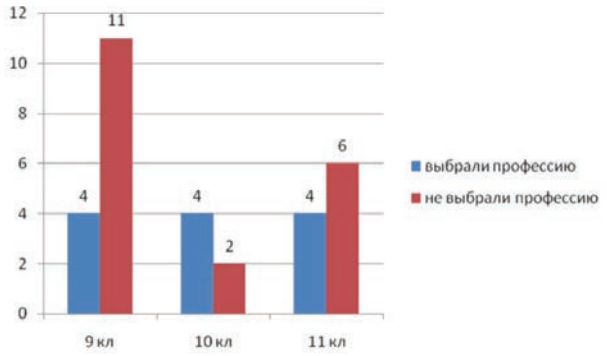


Рис. 44. Сравнение нескольких категориальных переменных

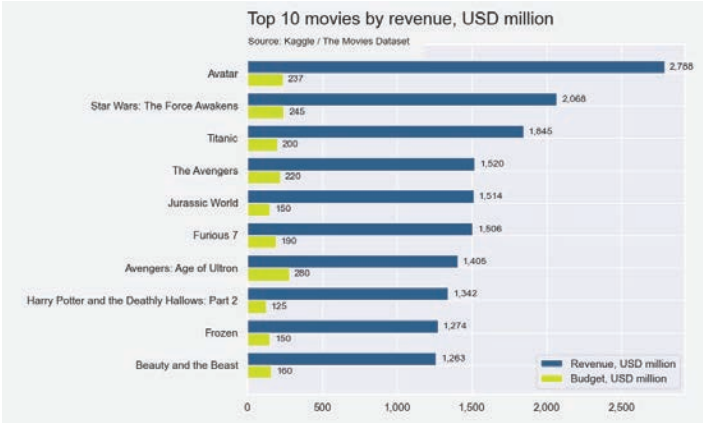


Рис. 45. Сравнение в горизонтальном представлении

Считается, что расположение столбиковой диаграммы вертикально или горизонтально воспринимается по-разному. Часто различия по вертикали воспринимают как рост показателя, по горизонтали – как прогресс.

Рис. 46. Восприятие – рост vs прогресс

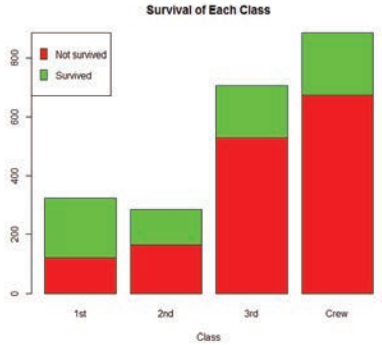


Рис. 47. Дополнительная разбивка на подгруппы

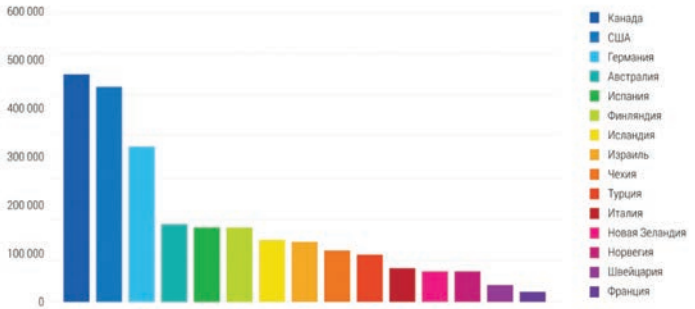


Рис. 48. Сортировка и поиск похожих групп

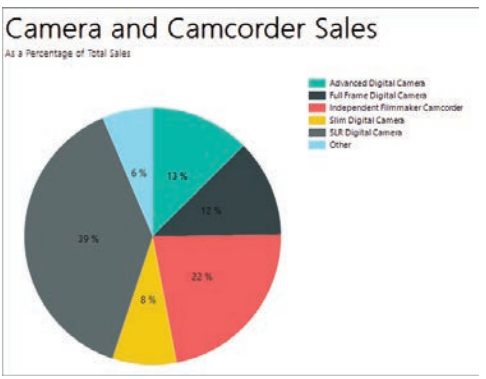


Рис. 49. Обычная круговая диаграмма

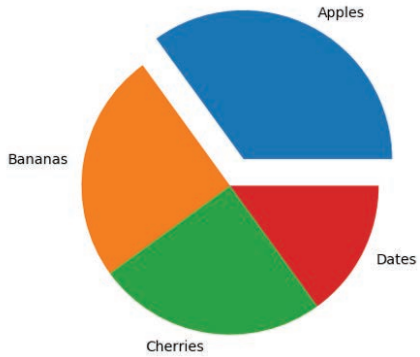


Рис. 50. Обычная круговая диаграмма с акцентом

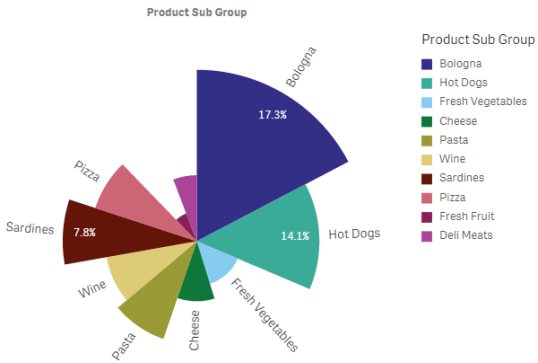


Рис. 51. Размер сектора пропорционален доли

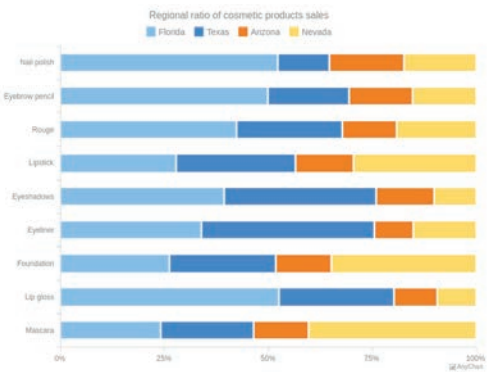


Рис. 52. «Выпрямление» круговых диаграмм для сравнения

3.1.1.4. Корреляция и диаграммы рассеяния

Корреляция или корреляционная зависимость – это статистическая взаимосвязь двух или более случайных величин. При этом изменения значений одной или нескольких из этих величин сопутствуют систематическому изменению значений другой или других величин [44].

Математической мерой корреляции двух случайных величин служит коэффициент корреляции. В случае если изменение одной случайной величины не ведет к закономерному изменению другой случайной величины, но приводит к изменению другой статистической характеристики данной случайной величины, то подобная связь не считается корреляционной, хотя и является статистической.

Значительная корреляция между двумя случайными величинами всегда является свидетельством существования некоторой статистической связи в данной выборке, но эта связь не обязательно должна наблюдаться для другой выборки и иметь причинно-следственный характер.

Часто исследователи делают ложные интуитивные выводы о наличии причинно-следственной связи между парами признаков, в то время как коэффициенты корреляции устанавливают лишь статистические взаимосвязи. В то же время, отсутствие корреляции между двумя величинами еще не значит, что между ними нет никакой связи. Например, зависимость может иметь сложный нелинейный характер, который корреляция не выявляет.

Метод вычисления коэффициента корреляции зависит от вида шкалы, к которой относятся переменные. Для измерения переменных с интервальной и количественной шкалами необходимо использовать коэффициент корреляции Пирсона. Если по меньшей мере одна из двух переменных имеет порядковую шкалу, либо не является нормально распределенной, необходимо использовать ранговую корреляцию Спирмена или Кендалла. Есть и другие способы [45].

Важной характеристикой совместного распределения двух случайных величин является ковариация (или корреляционный момент). Ковариация является совместным центральным моментом второго порядка. Ковариация определяется как математическое ожидание произведения отклонений случайных величин.

$$cov_{XY} = M\big[\big(X - M(X)\big)\big(Y - M(Y)\big)\big] = M(XY) - M(X)M(Y)$$

где $M(*)$ – математическое ожидание случайной величины.

Ковариация двух независимых случайных величин X и Y равна нулю. Ковариация имеет размерность, равную произведению размерности случайных величин, то есть величина ковариации зависит от единиц измерения независимых величин. Данная особенность ковариации затрудняет ее использование при анализе данных.

Для устранения недостатка ковариации был введен линейный коэффициент корреляции (или коэффициент корреляции Пирсона):

$$r_{XY} = \frac{cov_{XY}}{\sigma_X \sigma_Y}$$

где σ – стандартное отклонение случайной величины. Коэффициент корреляции изменяется в пределах от минус единицы до плюс единицы, $r_{XY} \in [-1,1]$.

В прикладной математической статистике корреляционный анализ – это метод обработки статистических данных, с помощью которого измеряется теснота связи между двумя или более переменными. С его помощью, в том числе, в регрессионном анализе определяют необходимость включения тех или иных факторов в уравнение множественной регрессии, а также оценивают полученное уравнение регрессии на соответствие выявленным связям.

При проведении корреляционного анализа необходимо помнить следующее:

– Применение возможно при наличии достаточного количества наблюдений для изучения. На практике считается, что число наблюдений должно не менее чем в 5-6 раз превышать число факторов.

– Необходимо, чтобы совокупность значений всех факторных и результативного признаков подчинялась многомерному нормальному распределению.

– Исходная совокупность значений должна быть качественно однородной.

– Сам по себе факт корреляционной зависимости не дает основания утверждать, что одна из переменных предшествует или является причиной изменений, или то, что переменные вообще причинно связаны между собой, а не наблюдается действие третьего фактора.

Пример распределения данных и соответствующие значения коэффициентов корреляции показаны на рис. 53.

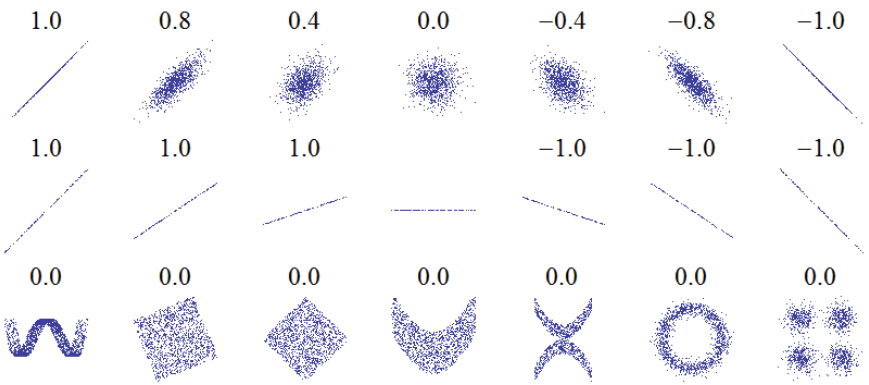


Рис. 53. Примеры корреляций [46]

Поскольку признаков в задаче анализа может быть много, необходимо проверить корреляцию для всех возможных пар признаков. Как было сказано выше, связь между независимой и целевой переменной может быть обоснованием включения независимой переменной в регрессионную модель. В свою очередь,

связь между независимыми переменными может означать потерю информации – вектора являются коллинеарными, один из них можно исключить из модели.

Поскольку всех возможных пар переменных может оказаться много, на практике удобно использовать следующую визуализацию корреляционной матрицы. Используются так называемые тепловые карты (heatmap). По горизонтальной и вертикальной осям откладываются имена переменных, а на пересечении двух имен – значения коэффициента корреляции. При этом значения окрашиваю в цвет в соответствии со значением. Традиционно в тепловых картах используют палитру от темного синего у самых малых значений (аналог холодной температуры, например, в тепловизоре) до красного у самых высоких значений (аналог горячей температуры). Для корреляционной матрицы лучше выбирать палитру, которая имеет яркие цвета вначале и конце палитры и бледный (или белый) цвет в середине. Таким образом низкая и отсутствующая корреляция будет обесцвечена на рисунке, а пары с наибольшей связью будут иметь яркие, резко отличающиеся оттенки. Эти яркие пары и требуют более детального изучения.

Примеры тепловых карт корреляционных матриц показаны для разного числа признаков на рис. 54 и 55.

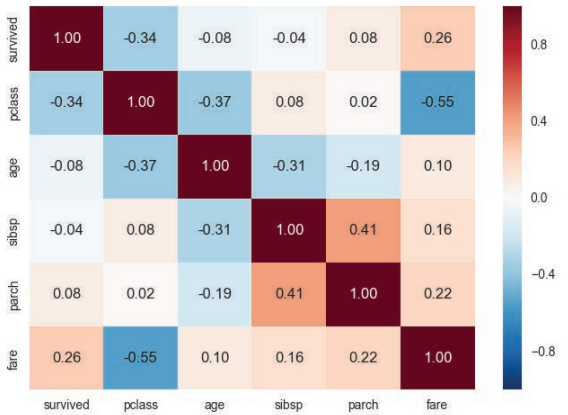


Рис. 54. Пример тепловой карты (мало признаков)

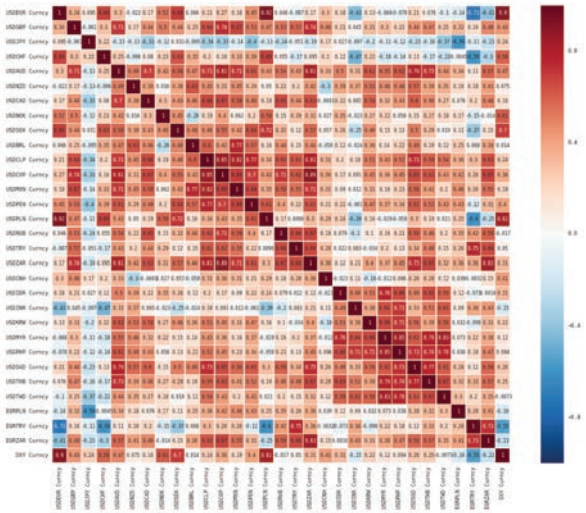


Рис. 55. Пример тепловой карты (много признаков)

Стандартным средством визуализации связи между двумя переменными с измеряемыми данными является диаграмма рассеяния, чья ось x представляет одну переменную, ось y – другую, а каждая точка на графике является записью.

Обычный график рассеяния показан на рис. 56. В свою очередь, если данные рассматриваемой пары признаков могут быть сгруппированы по третьему признаку (например, по классам целевой переменной или по уровням категориальной независимой переменной), то данные разных групп представляют на диаграмме рассеяния разным цветом и отражают группы в легенде как на рис. 57.

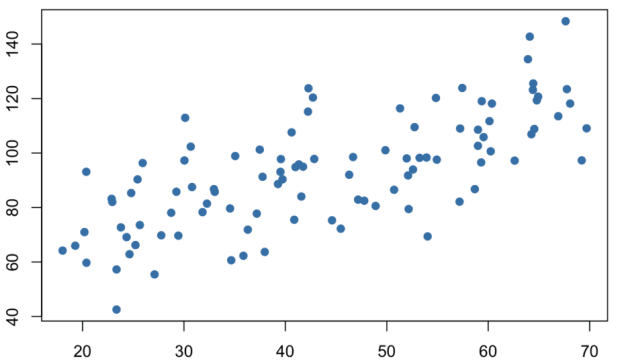


Рис. 56. Обычный график scatter plot

Рис. 57. График scatter plot с разбивкой на группы

Средства пакета Seaborn могут отобразить на одном рисунке диаграммы рассеяния всех пар переменных, что бывает полезно на ранних стадиях анализа. На главной диагонали обычно отображают гистограммы признака (рис. 58). Поскольку одна пара может быть представлена как зависимость y от x, так и x от y, то часто один из графиков рассеяния заменяют на график оценки совместной плотности распределения этих признаков (рис. 59).

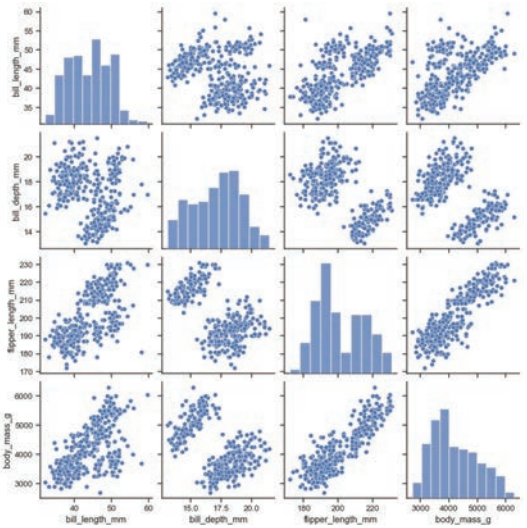


Рис. 58. Парные графики в пакете Seaborn

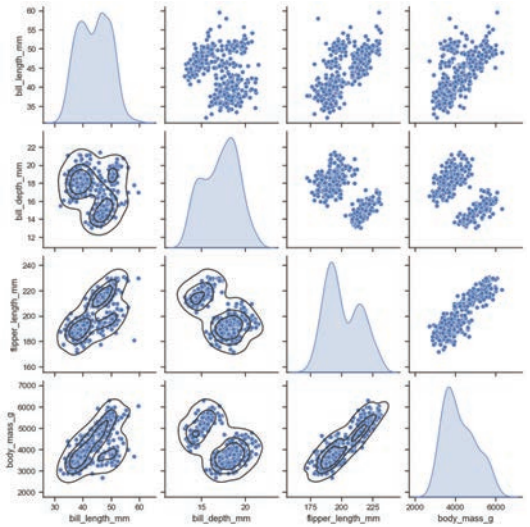


Рис. 59. Парные графики с оценкой KDE в пакете Seaborn

3.2. EDA в задаче кластеризации данных

В мире нет абсолютно одинаковых объектов, поэтому возникает естественный вопрос: как же описывать бесконечно разнообразный мир конечными фразами языка? Одно из решений – делать это огрублено, приблизительно, упрощенно. Первый шаг упрощения основан на том, что все объекты различны, но одни отличаются друг от друга «слабо», «мало», «незначительно», другие отличаются «сильно», «существенно». Идея состоит в том, чтобы объединить все мало различающиеся объекты в одну группу, оставив вне ее все сильно различающиеся. Второй шаг упрощения состоит в том, чтобы отказаться от учета различий внутри группы, пренебречь малыми отличиями, считать членов группы одинаковыми. Такую группу принято называть кластером или классом. Для выражения различий между классами им присваиваются различные имена. В дальнейшем, распознавание, идентификация объекта состоит в выяснении того, к какому классу он принадлежит [47].

Кластеризация (cluster analysis) — задача группировки множества объектов на подмножества (кластеры) таким образом, чтобы объекты из одного кластера были более похожи друг на друга, чем на объекты из других кластеров по какому-либо критерию. Задача кластеризации относится к классу задач обучения без учителя [48].

Кластеризация часто используется для решения следующих задач:

– Классификация объектов. Попытка понять зависимости между объектами путем выявления их кластерной структуры. В данном случае стремятся уменьшить число кластеров для выявления наиболее общих закономерностей.

– Сжатие данных. Можно сократить размер исходной выборки, взяв один или несколько наиболее типичных представителей каждого кластера.

– Обнаружение новизны (обнаружение шума). Выделение объектов, которые не подходят по критериям ни в один кластер.

Задаче кластерного анализа посвящено большое число учебников и монографий, в данном пособии мы не будем описывать алгоритмы кластеризации, но сделаем следующие замечания, полезные для практики анализа данных:

– Описание объектов может быть задано как признаками, по которым далее вычисляется расстояние между объектами, а может быть задано матрицей расстояний.

– Число кластеров выбирается произвольно, субъективно.

– Выбор меры расстояния определяет результат кластеризации, для одних и тех же данных возможно применение различных мер расстояний.

– Существует множество мер качества решения задачи кластеризации, не существует лучшего метода оценки качества кластеризации, выбор метода также субъективен и влияет на результат кластеризации.

– Во многих методах для оценки качества кластеризации задачу переформулируют в задачу дискретной оптимизации. Часто стремятся достичь: минимума среднего внутрикластерного расстояния или максимума среднего межкластерного расстояния.

При выполнении разведочного анализа данных, в случае малых размерностей или сокращения размерностей, результаты кластеризации можно визуализировать. Например, показать способ разбиения пространства поиска на подобласти, принадлежащие разным кластерам: линейное разделение (рис. 60) и выпуклые области (рис. 61). В алгоритмах, где решение представляется в виде набора точек центров кластеров, отображаются точки центров, а данные раскрашиваются по принадлежности к кластеру (рис. 62).

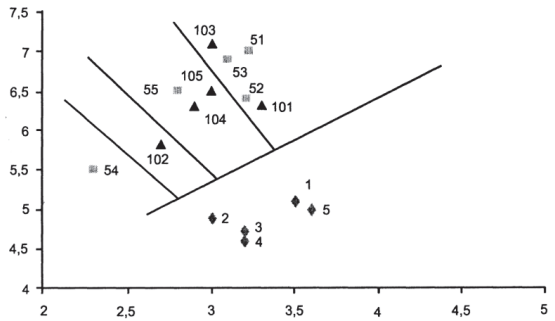


Рис. 60. Пример визуализации линейного разделения на группы

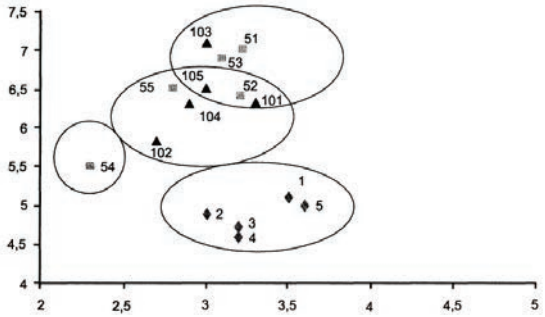


Рис. 61. Пример визуализации разделения на выпуклые области

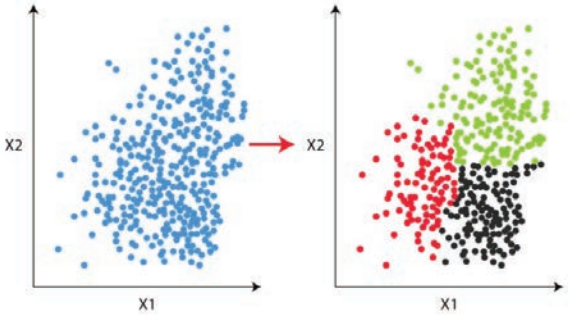


Рис. 62. Пример раскрашивания выделенных кластеров

Для выявления структуры кластеров независимо от размерности данных, удобно использовать метод построения дендрограмм. Под дендрограммой обычно понимается дерево, построенное по матрице мер близости. Дендрограмма позволяет изобразить взаимные связи между объектами из заданного множества. Для создания дендрограммы требуется матрица сходства, которая определяет уровень сходства между парами кластеров. Чаще используются агломеративные методы, в которых новые кластеры создаются путем объединения более мелких кластеров и, таким образом, дерево создается от листьев к корню [49].

Листья дендрограммы представляют собой объекты кластеризации, а ветви – связи между наиболее близкими объектами, объектами и кластерами или между кластерами. Длинна ветки на дендрограмме пропорциональна межкластерному расстоянию, что позволяет судить о том, какие объекты или группы объектов находятся ближе или дальше в пространстве признаков. Пример дендрограммы показан на рис. 63.

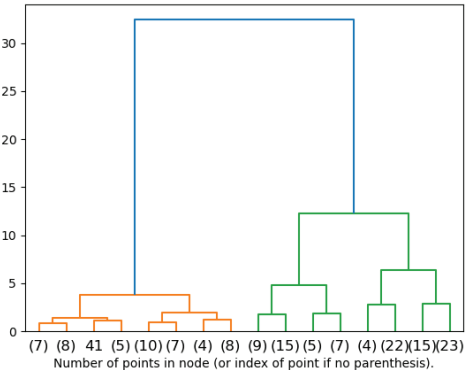


Рис. 63. Пример дендрограммы [50]

Часто, если число объектов велико, дендрограмма будет иметь большой размер и ее сложно интерпретировать. В таком случае, дерево обрезают на некотором достаточно глубоком уровне, а в листьях отображают не отдельные объекты, а число или полный перечень объектов обрезанной ветви.

По дендрограмме удобно судить о структуре кластеров и выбирать их число.

Еще одним подходом визуального анализа числа кластеров является «метод локтя». Суть заключается в следующем. Задачу кластеризации многократно решают выбранным методом с разным числом кластеров. Для каждого числа кластеров оценивается метрика качества, например, среднее внутрикластерное расстояние. Далее строится график, в котором по оси x откладывается число кластеров, а по оси y – значение меры качества кластеризации (рис. 64). Во многих прикладных задачах эффективным оказывается число кластеров, при котором происходит резкий «перелом» графика изменения меры качества, т.е. дальнейшее увеличение числа кластеров дает значительно меньшее изменение меры качества.

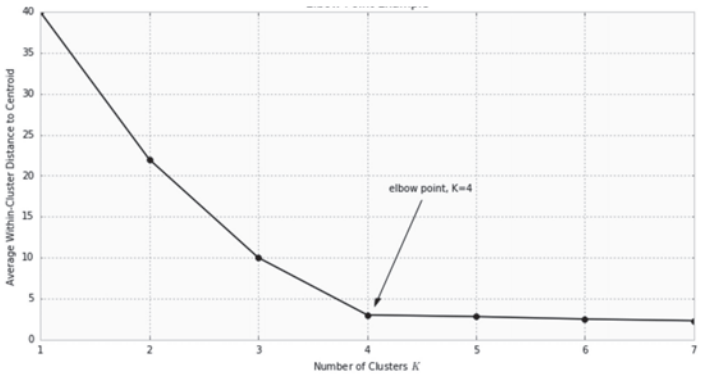


Рис. 64. Пример метода локтя, эффективное число кластеров равно 4

Одной из наиболее популярных библиотек для анализа данных и машинного обучения на Python является пакет scikit-learn [50]. scikit-learn – это библиотека, предназначенная для машинного обучения, написанная на языке программирования Python и распространяемая в виде свободного программного обеспечения. В ее состав входят различные алгоритмы, в том числе предназначенные для задач классификации, регрессионного и кластерного анализа данных. Библиотека была разработана для взаимодействия с численными и научными библиотеками языка программирования Python NumPy и SciPy. В частности, в scikit-learn реализованы различные методы кластерного анализа. На рис. 65 показаны примеры решения задачи кластеризации разными алгоритмами (столбцы) для разных наборов данных (строки).

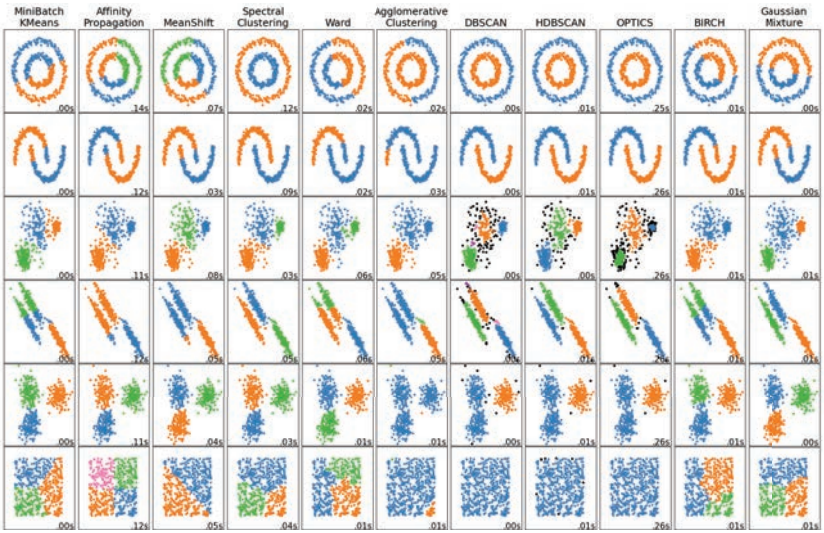


Рис. 65. Результаты работы разных алгоритмов кластеризации

Не вдаваясь в подробности работы алгоритмов, основной вывод, который мы можем сделать – разные подходы, основываясь на разных идеях, использующие разные меры качества кластеризации дают различные результаты. Поэтому на практике, после обнаружения кластеров, необходимо продемонстрировать их и обсудить с предметными экспертами. Если выделенная кластерная структура не может быть интерпретирована и не несет практической пользы для дальнейших этапов анализа, то исследование продолжается – процедуру повторяют с новыми настройками алгоритмов (другое число кластеров, другие меры расстояния и меры качества кластеризации) или констатируют, что данные однородные, и нет смысла делить из на подгруппы.

3.3. EDA в задаче регрессии

Разведочный анализ данных EDA проводится на примере инженерной задачи моделирования уровня шума крыла самолета (Airfoil Self-Noise Data Set), представленной в репозитории UCI [38].

Задача поставлена как задача регрессии, т.е. требуется построить модель прогнозирования значений одного числового параметра по значениям других числовых параметров.

Процедура EDA является довольно унифицированной, но для задач классификации, некоторые шаги будут реализованы иначе и потребуется иная интерпретация результатов.

Краткое описание задачи. Набор данных включает базу данных анализ аэродинамических поверхностей авиационного крыла типа NACA 0012 [51] разного размера при различных скоростях набегания потока воздуха в аэродинамической трубе и разных углах атаки. Во всех экспериментах размах профиля крыла и положение наблюдателя были одинаковыми.

Набор данных содержит следующие независимые переменные:

– частота, в герцах (Frequency),
– угол атаки, в градусах (Angle of attack),
– длина хорды, в метрах (Chord length),
– скорость набегающего потока, в метрах в секунду (Free-stream velocity),
– мощность вытеснения на стороне всасывания, в метрах (Suction side displacement thickness).

Целевая переменная:

– Масштабированный уровень звукового давления, в децибелах (Scaled sound pressure level).

Все переменные – это вещественные числа. Число записей базы данных равно 1503. Файл базы данных представлен на сайте репозитория UCI [38].

Процедура EDA для задачи регрессии включает следующие этапы:

1. Импорт данных.
2. Визуализация данных в табличном виде. Описательная статистика.
3. Визуальный анализ распределений и разброса данных.
4. Заполнение пропусков (при наличии).
5. Визуализация парных графиков. Анализ корреляций.
6. Масштабирование данных.
7. Применение метода и анализ главных компонент.
8. Построение линейной регрессии. Оценка важности переменных.

Для выполнения основных этапов EDA достаточно использовать ранее рассмотренные пакеты Numpy, Pandas и Matplotlib (Seaborn). Однако некоторые шаги потребуют вовлечение алгоритмов машинного обучения и специальных инструментов анализа, для чего будет применяться ранее упомянутая библиотека scikit-learn:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# заполнение пробелов методом kNN
from sklearn.impute import KNNImputer
# МГК
from sklearn.decomposition import PCA
# масштабирование данных
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# линейная регрессия
from sklearn.linear_model import LinearRegression
# разбивка на обучающее и тестовое множество
from sklearn.model_selection import train_test_split
# метрики оценки качества регрессии
from sklearn.metrics import mean_squared_error, r2_score

# функции вывода в ячейки блокнота
from IPython.core.display import display, HTML
```

3.3.1. Импорт данных

Импорт данных проще всего осуществлять средствами Pandas. Сразу же необходимо визуально оценить корректность импорта данных, а именно, наличие индексов и заголовков, соответствующие значения в столбцах, тип разделителя разрядов. Необходимо исключить технические ошибки импорта.

Примечание. Оригинальный набор данных не имеет пропущенных значений. Для данного примера используется модифицированный набор данных, в котором имеются пропуски в значениях признаков.

Импортируем данные в DataFrame с именем raw, подчеркивая, что это «сырые» данные, требующие анализа, исправления, предварительной обработки и т.д. прежде чем их можно будет использовать для целей машинного обучения.

```
raw = pd.read_csv(https://archive.ics.uci.edu/static/public/291/air-
foil+self+noise.csv', header=None)
display(raw.head(3))
display(raw.tail(3))
```

Вывод

	0	1	2	3	4	5
0	800.0	NaN	0.3048	71.3	0.002663	126.201
1	1000.0	0.0	0.3048	71.3	0.002663	125.201
2	1250.0	0.0	0.3048	71.3	0.002663	125.951

	0	1	2	3	4	5
1500	4000.0	15.6	0.1016	NaN	0.052849	106.604
1501	5000.0	15.6	0.1016	39.6	0.052849	106.224
1502	6300.0	15.6	0.1016	39.6	0.052849	104.204

Далее необходимо составить отчет о структуре импортированного DataFrame с помощью функции `pd.info()`.

```
raw.info()
```

Вывод

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1503 entries, 0 to 1502
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       1486 non-null   float64
 1   1       1451 non-null   float64
 2   2       1456 non-null   float64
 3   3       1426 non-null   float64
 4   4       1449 non-null   float64
 5   5       1503 non-null   float64
dtypes: float64(6)
memory usage: 70.6 KB
```

Из отчета следует, что у нас импортировано 1503 записи, представленных 6 признаками, которые имеют тип float64 (вещественное число). Сами данные импортированы корректно с технической точки зрения, так как структура первых и последних строк таблицы совпадает. Заголовки с именами признаков отсутствуют. В данных имеются пропуски во всех столбцах кроме последнего (целевого).

Для дальнейшей работы мы добавим названия столбцов, а также выделим массивы с именами переменных, сгруппированными по смыслу: числовые и категориальные, входы (независимые переменные) и выходы (целевые). Можно аналогично создать любые другие списки столбцов, которые потребуется обрабатывать отдельно. Далее можно выбирать данные целой группой, а не перечисляя их имена каждый раз заново.

```
numerical_names = ['Frequency', 'Angle', 'Chord', 'Free_stream', 'Suc-
tion', 'Sound_level']
categorical_names = []
inputs = ['Frequency', 'Angle', 'Chord', 'Free_stream', 'Suction']
targets = ['Sound_level']
raw.columns = ['Frequency', 'Angle', 'Chord',
               'Free_stream', 'Suction', 'Sound_level']
raw.head(3)
```

Вывод

	Frequency	Angle	Chord	Free_stream	Suction	Sound_level
0	800.0	NaN	0.3048	71.3	0.002663	126.201
1	1000.0	0.0	0.3048	71.3	0.002663	125.201
2	1250.0	0.0	0.3048	71.3	0.002663	125.951

3.3.2. Визуализация данных в табличном виде. Описательная статистика

Чтобы делать заключения о свойствах данных, необходимо оценить объем пропусков. Если отсутствует существенный объем данных, то стоит констатировать потерю информации о подобных признаках и исключить их из анализа, либо попытаться повторно собрать данные (если возможно). Функция `pd.isna()` возвращает маску со значениями `True` для записей, в которых есть пропуски. Сумма значений `True` даст представление о том, сколько пропусков в каждом из признаков, а разделив на общее число записей в базе данных можно получить долю пропусков в общем объеме данных.

```
df_len = raw.shape[0]
for col in raw.columns:
  missing_value_counts = raw[col].isna().sum()
  percentage = missing_value_counts/df_len*100
  print("Столбец \"%s\" содержит %d пропусков из %d (процент от объема - 
%.2f %%)."%(col, missing_value_counts, df_len, percentage))
```

Вывод

```
Столбец "Frequency" содержит 17 пропусков из 1503 (процент от объема - 
1.13 %).
Столбец "Angle" содержит 52 пропусков из 1503 (процент от объема - 3.46 
%).
Столбец "Chord" содержит 47 пропусков из 1503 (процент от объема - 3.13 
%).
Столбец "Free_stream" содержит 77 пропусков из 1503 (процент от объема - 
5.12 %).
Столбец "Suction" содержит 54 пропусков из 1503 (процент от объема - 3.59 
%).
Столбец "Sound_level" содержит 0 пропусков из 1503 (процент от объема - 
0.00 %).
```

Как видно из отчета выше, процент пропусков не велик, а значит со всеми признаками можно работать дальше. Пропущенные значения можно будет попробовать восстановить.

Представим описательную статистику для количественных данных.

```
raw[numerical_names].describe()
```

Вывод

	Frequency	Angle	Chord	Free_stream	Suction	Sound_level
count	1486.000000	1451.000000	1456.000000	1426.000000	1449.000000	1503.000000
mean	2878.287349	6.753894	0.136316	50.840042	0.011249	124.835943
std	3137.745026	5.884402	0.093747	15.541817	0.013251	6.898657
min	200.000000	0.000000	0.025400	31.700000	0.000401	103.380000
25%	800.000000	2.000000	0.050800	39.600000	0.002535	120.191000
50%	1600.000000	5.400000	0.101600	39.600000	0.004957	125.721000
75%	4000.000000	9.900000	0.228600	71.300000	0.016104	129.995500
max	20000.000000	22.200000	0.304800	71.300000	0.058411	140.987000

Аналогично нужно построить отчет для категориальных данных, если они присутствуют в наборе (в данном примере – нет).

```
data_raw[categorical_names].describe()
```

Описательная статистика позволяет сделать некоторые заключения о свойствах данных.

Во-первых, диапазоны значений (min, max) и типичное (нормальное, центральное) значение нужно показать предметным экспертам для анализа корректности сбора данных. Если данные выбиваются за границы допустимых значений, то надо или перепроверять всю выборку, или, как минимум, обратить внимание на аномальные значения (выбросы).

Во-вторых, как мы знаем, медиана – является робастной оценкой математического ожидания, в то время как оценка среднего менее устойчивая. Равенство или близость значений среднего и медианы говорит о том, что данные могут иметь нормальный закон распределения (требуется проверка), а сильное расхождение – о наличии шума и выбросов в данных.

В нашем примере, среднее не совпадает с медианой во всех столбцах, кроме целевого.

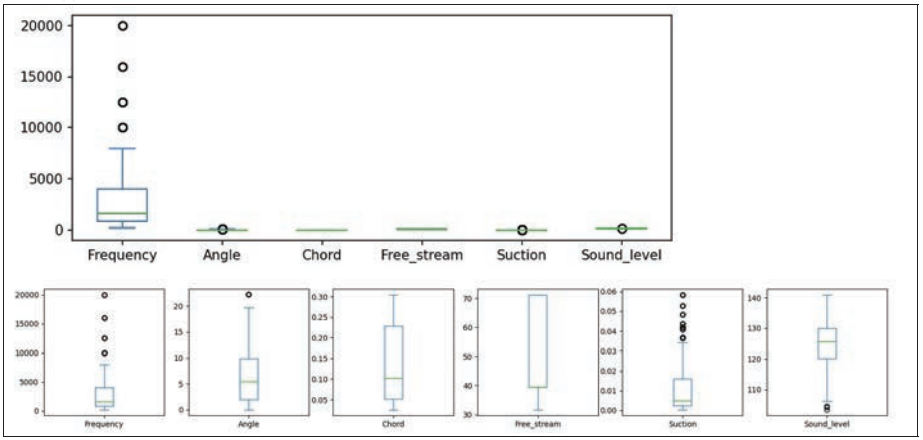
3.3.3. Визуальный анализ распределений и разброса данных

Вначале мы построим графики box-plot для количественных данных и bar-plot для категориальных данных. Если данные можно разделить на подгруппы по какой-то другой переменной, то лучше группы проанализировать отдельно и отразить на графике разным цветом. Возможно – это разные наборы данных в одной базе данных и их стоит анализировать дальше по отдельности (например, анализировать значения роста для мужчин и женщин, а не рост людей в целом).

Построим график box-plot средствами Pandas для всех переменных сразу, чтобы оценить масштаб значений, и для каждой переменной по-отдельности.

```
fig, ax = plt.subplots(figsize=(8,3), dpi=150)
raw.plot(kind='box', ax=ax)
plt.show()
raw.plot(kind='box', subplots=True, figsize=(20,3))
plt.show()
```

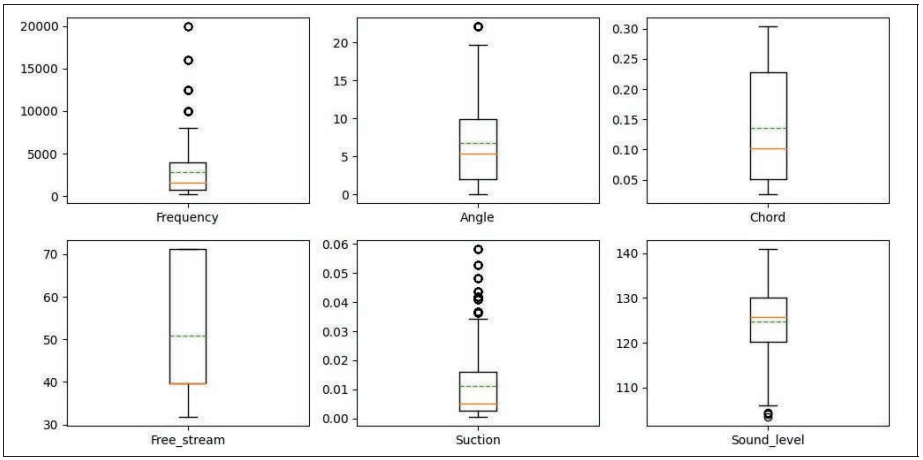
Вывод



Также создадим графики средствами Matplotlib, представив их удобной сеткой 2 на 3 и добавив на график среднее значение (зеленый пунктрир).

```
fig, ax = plt.subplots(figsize=(12,6), ncols=3, nrows=2)
for i in range(2):
  for j in range(3):
    ax[i][j].boxplot(raw[numerical_names[j + 3*i]][~np.isnan(raw[numeri-
cal_names[j + 3*i]])],
                     labels = [numerical_names[j + 3*i]],
                     showmeans=True, meanline=True)
plt.show()
```

Вывод



Из графиков видно, что часть данных довольно плотно сгруппирована вокруг их центрального значения и имеются выбросы, особенно для переменных Frequency и Suction. Наглядно показаны расхождения значений среднего и

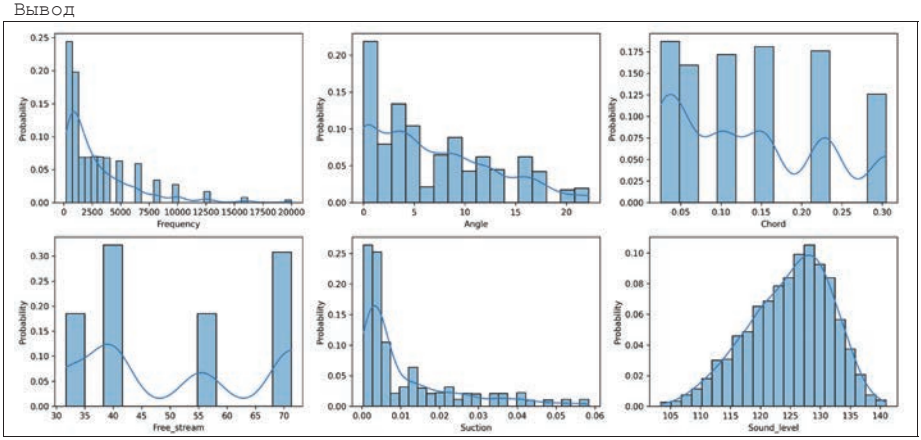
медианы. Разброс данных вокруг близких значений медианы и среднего у целевой переменной Sound_level может свидетельствовать о нормальности распределения этой переменной.

Поскольку категориальных переменных для построения графиков bar-plot у нас нет, перейдем к следующему шагу.

Оценим плотность распределения вероятностей для значений каждого признака с помощью гистограмм. Лучше всего воспользоваться пакетом Seaborn, так как он имеет встроенную возможность отображения ядерной оценки плотности на в виде графика функции плотности.

```
fig, ax = plt.subplots(figsize=(18,8), ncols=3, nrows=2)
for i in range(2):
  for j in range(3):
    sns.histplot(data=raw, x=numerical_names[j + 3*i], ax = ax[i][j],
                 kde=True, stat='probability')
plt.show()
```

Вывод



По гистограммам видно, что распределение близкое к нормальному только, как предполагалось, у целевой переменной. У остальных наибольшая плотность наблюдается в области нижней границы значений.

Также мы видим, что 2 переменные Chord и Free_stream имеют узкие «всплески» лишь в нескольких местах, что может свидетельствовать, что это дискретные переменные. Поскольку благодаря визуальному анализу возникла такая гипотеза, проверим это с помощью функции `pd.value_counts()`.

```
raw['Chord'].value_counts()
```

Вывод

```
0.0254    272
0.1524    263
0.2286    256
0.1016    250
0.0508    232
0.3048    183
Name: Chord, dtype: int64
```

```
raw['Free_stream'].value_counts()
```

Вывод

```
39.6    459
71.3    439
55.5    264
31.7    264
Name: Free_stream, dtype: int64
```

Как видно из отчета, переменная Chord имеет 6 фиксированных значений, при которых происходили эксперименты. А переменная Free_stream – 4 значения. Эти 2 переменные стоит попробовать представить как категории и рассмотреть распределения остальных данных, сгруппировав их по этим категориям. Возможно, для разных категорий будут наблюдаться разные распределения.

Создадим копию DataFrame и поменяем тип 2-х переменных на категориальные.

```
numerical_names_2 = ['Frequency', 'Angle', 'Suction', 'Sound_level']
categorical_names_2 = ['Chord', 'Free_stream']
raw_2 = raw.copy()
raw_2 = raw_2.astype({"Chord":'category', "Free_stream":'category'})
raw_2.info()
```

Вывод

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1503 entries, 0 to 1502
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Frequency    1486 non-null   float64
 1   Angle        1451 non-null   float64
 2   Chord        1456 non-null   category
 3   Free_stream  1426 non-null   category
 4   Suction      1449 non-null   float64
 5   Sound_level  1503 non-null   float64
dtypes: category(2), float64(4)
memory usage: 50.4 KB
```

Для удобства, категории можно переименовать.

```
new_categories = []
for i in range(6):
  new_categories.append("Chord_" + str(i+1))
raw_2['Chord'] = raw_2['Chord'].cat.rename_categories(new_categories)

new_categories = []
for i in range(4):
  new_categories.append("Stream_" + str(i+1))
raw_2['Free_stream'] = raw_2['Free_stream'].cat.rename_catego-
ries(new_categories)
raw_2.head(3)
```

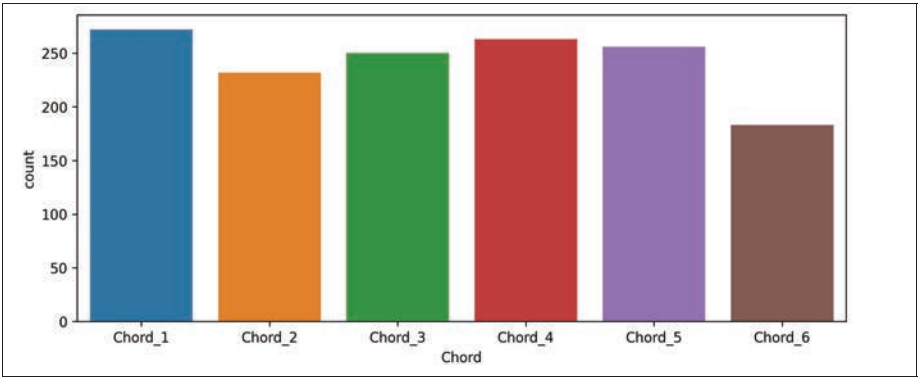
Вывод

	Frequency	Angle	Chord	Free_stream	Suction	Sound_level
0	800.0	NaN	Chord_6	Stream_4	0.002663	126.201
1	1000.0	0.0	Chord_6	Stream_4	0.002663	125.201
2	1250.0	0.0	Chord_6	Stream_4	0.002663	125.951

Вначале построим графики bar-plot, чтобы визуализировать то, как часто встречаются значений каждой категории. Если какая-то категория встречается сильно реже других, такие данные называются несбалансированными.

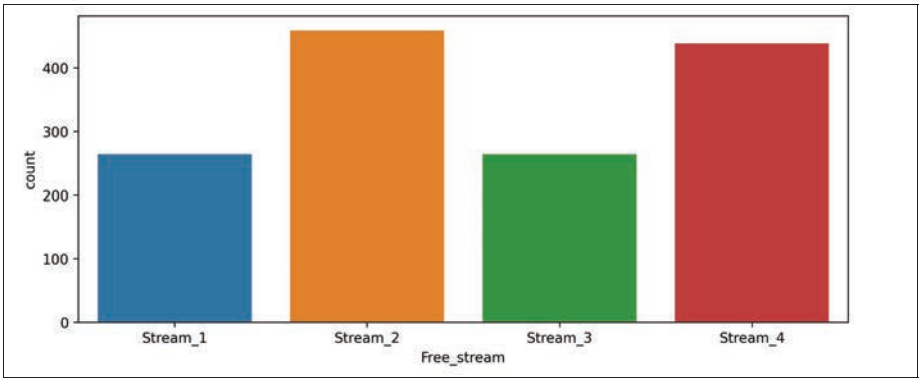
```
fig, ax = plt.subplots(figsize=(10,4), dpi=150)
sns.countplot(data=raw_2, x="Chord", ax=ax)
plt.show()
```

Вывод




```
fig, ax = plt.subplots(figsize=(10,4), dpi=150)
sns.countplot(data=raw_2, x="Free_stream", ax=ax)
plt.show()
```

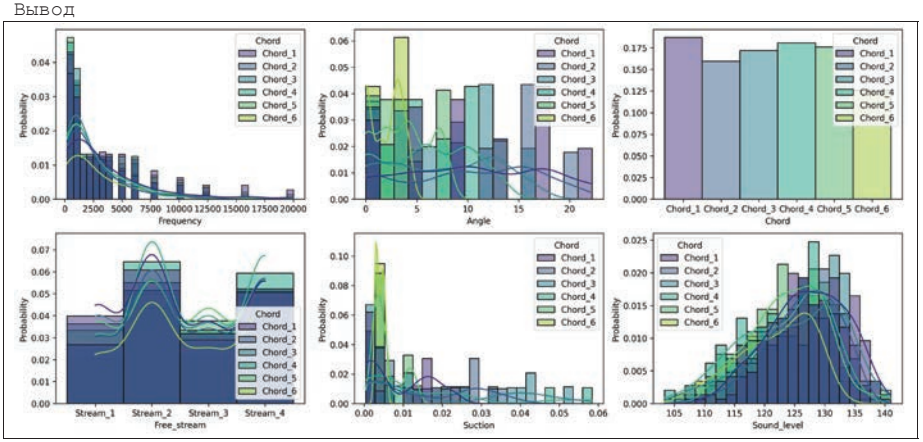
Вывод



Проверим распределения данных, сгруппированные по категориям, для чего при построении гистограмм будем использовать аргумент hue в функциях seaborn.

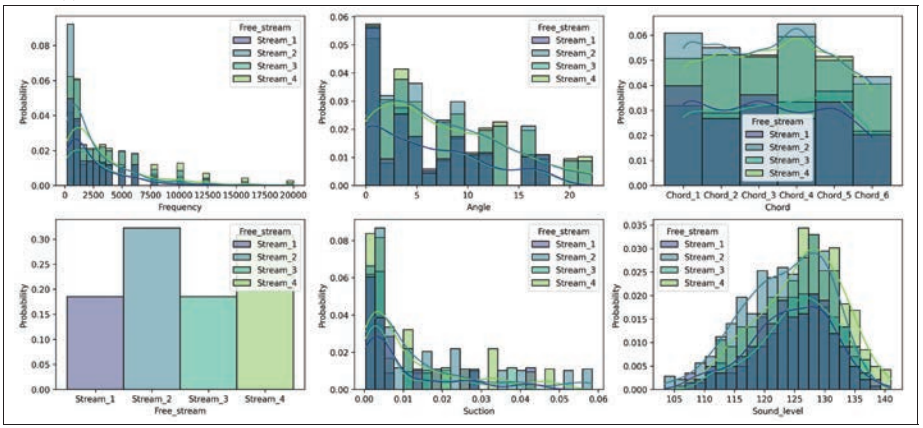
```
fig, ax = plt.subplots(figsize=(18,8), ncols=3, nrows=2)
for i in range(2):
  for j in range(3):
    sns.histplot(data=raw_2, x=numerical_names[j + 3*i], ax = ax[i][j],
                 kde=True, stat='probability',
                 hue='Chord', palette='viridis')
plt.show()
```

Вывод



```
fig, ax = plt.subplots(figsize=(18,8), ncols=3, nrows=2)
for i in range(2):
  for j in range(3):
    sns.histplot(data=raw_2, x=numerical_names[j + 3*i], ax = ax[i][j],
                 kde=True, stat='probability',
                 hue='Free_stream', palette='viridis')
plt.show()
```

Вывод



Из гистограмм видно, что распределения по категориям примерно похожи, за исключением переменной Angle сгруппированной по переменной Chord. В целом, нет смысла разделять данные по группам, так как мы не наблюдаем, что данные разных категорий получены из разных генеральных совокупностей.

3.3.4. Заполнение пропусков

Не существует универсального подхода для работы с пропущенными значениями. При наличии больших баз данных с достаточно репрезентативными данными, записки с пропусками просто удаляются. Но в большинстве случаев, поскольку часть данных записи все же присутствует, более выгодно спрогнозировать пропущенное значение, чтобы запись была использована при построении модели. Заполненные пропуски являются прогнозными значениями и поэтому содержат вероятные ошибки, поэтому в общем случае заполнение пропусков может исказить результаты анализа данных.

Самый простой, но часто применяемый на практике метод – это заполнение средним значением, поскольку среднее значение является оценкой математического ожидания – значения, которое появляется в выборке с наибольшей вероятностью. Другой вариант – заполнение медианным значением, так как медиана является робастной оценкой центрального положения, а значит меньше подвержен влиянию выбросов.

Более сложные подходы используют методы анализа и машинного обучения, чтобы сделать прогноз пропущенного значения по совокупности

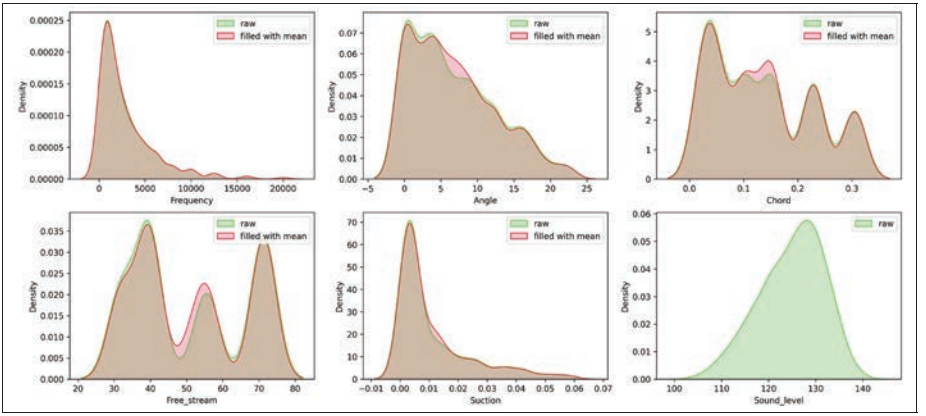
имеющихся значений записи. В частности, один из наиболее простых и достаточно эффективных подходов – метод kNN, k ближайших соседей.

К сожалению, нет какого-либо эффективного подхода предварительной оценки качества заполнения пропусков. Основной способ – оценка качества моделей, построенных по данным, в которых были спрогнозированы пропущенные значения. Однако, есть очевидное предположение – распределения данных до и после заполнения признаков должны быть похожими. Например, среднее значение – это вычисляемое значение, которое может отсутствовать в наборе данных, а значит на гистограмме могут появиться новые «пики» в областях значений, которые ранее не были представлены. Медиана – это значение из набора данных, а значит может существенно вырасти высота одного из «пиков» в области медианного значения.

Выполним заполнение пропусков средним и медианным значениями и методом kNN с помощью инструментов пакета Pandas, а затем визуально сравним результаты по оценке плотностей распределения вероятностей значений признаков.

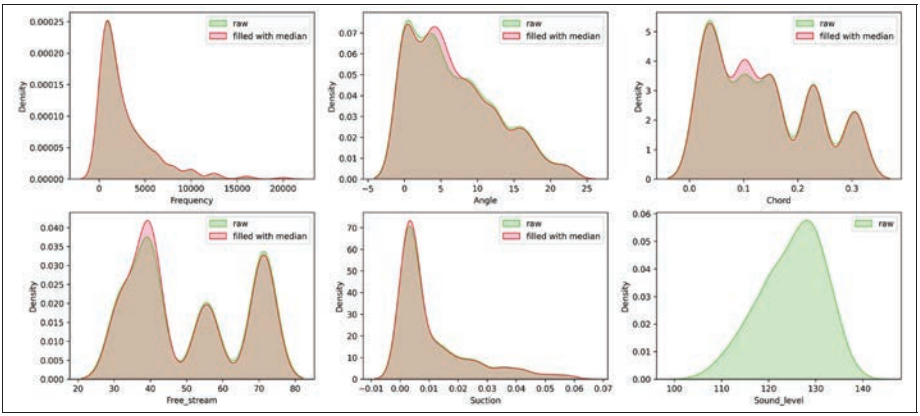
```
# заполнение средним
df_filled_mean = raw.copy()
df_filled_mean = df_filled_mean.fillna(df_filled_mean.mean())

fig, ax = plt.subplots(figsize=(18,8), ncols=3, nrows=2)
for i in range(2):
  for j in range(3):
    sns.kdeplot(data=raw, x=numerical_names[j + 3*i], ax = ax[i][j],
        color='limegreen', label='raw', fill=True)
    if numerical_names[j + 3*i] != "Sound_level":
      sns.kdeplot(data=df_filled_mean, x=numerical_names[j + 3*i], ax =
ax[i][j],
          color='crimson', label='filled with mean', fill=True)
    ax[i][j].legend()
plt.show()
```

Вывод



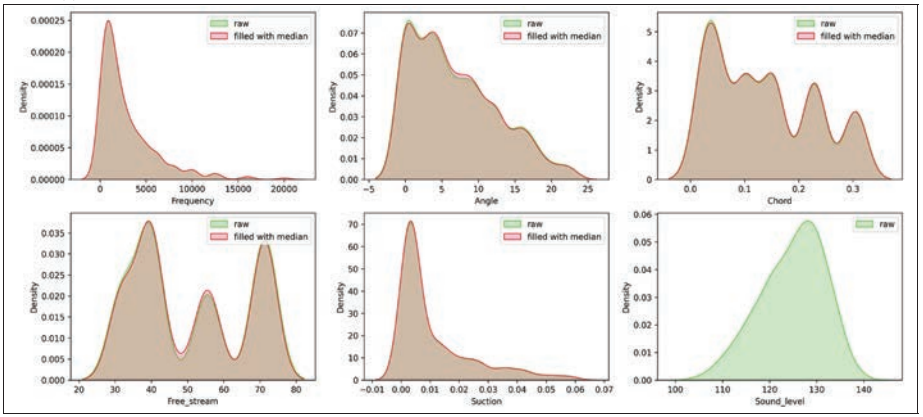
```
# заполнение медианой
df_filled_median = raw.copy()
df_filled_median = df_filled_median.fillna(df_filled_median.median())

fig, ax = plt.subplots(figsize=(18,8), ncols=3, nrows=2)
for i in range(2):
  for j in range(3):
    sns.kdeplot(data=raw, x=numerical_names[j + 3*i], ax = ax[i][j],
        color='limegreen', label='raw', fill=True)
    if numerical_names[j + 3*i] != "Sound_level":
      sns.kdeplot(data=df_filled_median, x=numerical_names[j + 3*i], ax =
ax[i][j],
          color='crimson', label='filled with median', fill=True)
    ax[i][j].legend()
plt.show()
```

Вывод




```
# метод kNN
imputer = KNNImputer(n_neighbors=3)
rez = imputer.fit_transform(raw)
data_filled_knn = pd.DataFrame(rez, columns=numerical_names)

fig, ax = plt.subplots(figsize=(18,8), ncols=3, nrows=2)
for i in range(2):
  for j in range(3):
    sns.kdeplot(data=raw, x=numerical_names[j + 3*i], ax = ax[i][j],
        color='limegreen', label='raw', fill=True)
    if numerical_names[j + 3*i] != "Sound_level":
      sns.kdeplot(data=data_filled_knn, x=numerical_names[j + 3*i], ax =
ax[i][j],
          color='crimson', label='filled with median', fill=True)
    ax[i][j].legend()
plt.show()
```

Вывод



Визуально, метод kNN не изменил график плотности распределения значений. Поэтому, далее в будем использовать набор данных, в котором пропуски заполнены с помощью kNN.

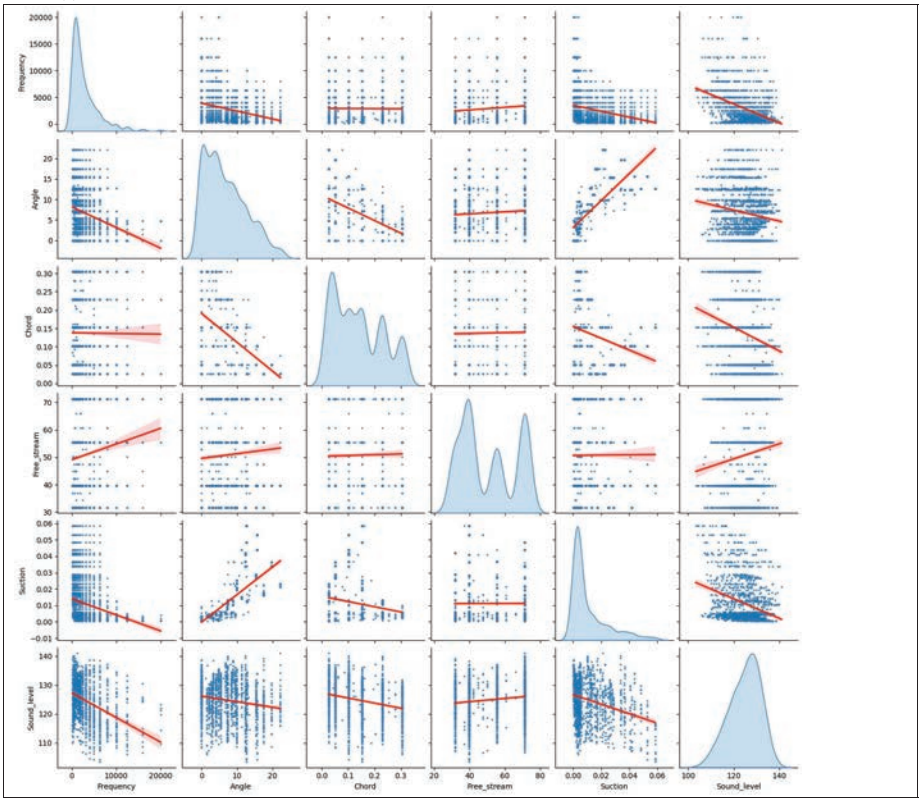
3.3.5. Визуализация парных графиков. Анализ корреляций

Парные графики могут обнаружить связи между парой признаков, поэтому есть смысл визуализировать все возможные пары и, в случае обнаружения зависимости, отдельно представить наиболее интересные пары.

Диаграмму рассеяния всех пар признаков можно построить с помощью функции Seaborn `sns.pairplot()`. Более того, Seaborn позволяет отобразить оценку линейной регрессии на каждом из графиков для оценки общего тренда.

```
sns.pairplot(data = data_filled_knn, kind='reg', diag_kind='kde',
             plot_kws={'line_kws':{'color':'red'}, "scatter_kws":{"s":
3}})
plt.show()
```

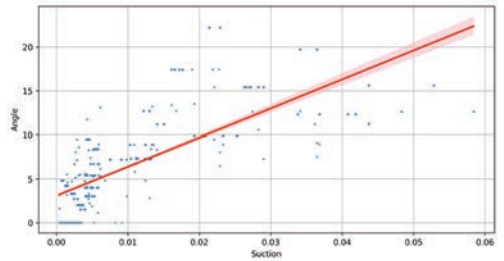
Вывод



На графиках выше, мы можем заметить, что данные довольно сильно разбросаны, что может говорить, что связи между признаками многомерные и проекции на плоскость не показательные, за исключением пары Suction и Angle. Покажем эту пару отдельно.

```
fig, ax = plt.subplots(figsize=(10,5))
sns.regplot(data=data_filled_knn, x= 'Suction', y= 'Angle', ax=ax,
            scatter_kws={"s": 3}, line_kws={"color": "red"})
ax.grid()
plt.show()
```

Вывод



Далее выполним корреляционный анализ. Вначале построим корреляционную матрицу. Если признаков не очень много, то можно проанализировать только значения матрицы, однако визуализация корреляционной матрицы более показательна, и особенно, когда признаков много.

```
data_filled_knn.corr()
```

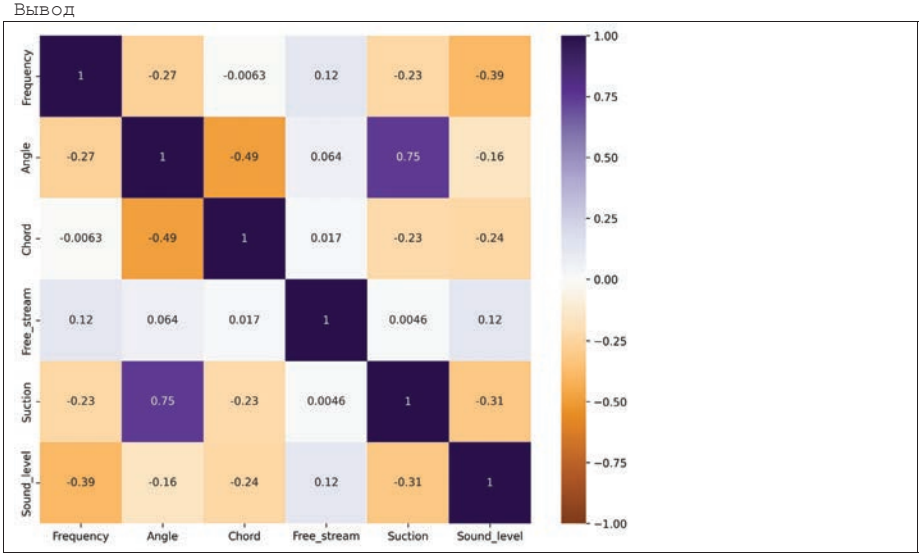
Вывод

	Frequency	Angle	Chord	Free_stream	Suction	Sound_level
Frequency	1.000000	-0.272447	-0.006302	0.117569	-0.233336	-0.388043
Angle	-0.272447	1.000000	-0.488873	0.064335	0.745879	-0.159946
Chord	-0.006302	-0.488873	1.000000	0.017301	-0.225878	-0.236973
Free_stream	0.117569	0.064335	0.017301	1.000000	0.004566	0.124279
Suction	-0.233336	0.745879	-0.225878	0.004566	1.000000	-0.312594
Sound_level	-0.388043	-0.159946	-0.236973	0.124279	-0.312594	1.000000

При построении тепловой карты необходимо учесть следующие особенности. Пропорции изображения должны быть такими, чтобы карта была квадратной – все оси равнозначны. Поскольку графики строятся по данным, а коэффициенты в матрице корреляций могут быть больше -1 и меньше 1, необходимо установить лимиты отображения в диапазон $[-1, 1]$. Необходимо выбрать палитру с бледным цветом в центре в нуле и различными яркими цветами на границах (около -1 и 1).

```
fig, ax = plt.subplots(figsize=(10,8))
sns.heatmap(data=data_filled_knn.corr(),
            annot=True, vmin=-1,vmax=1, cmap='PuOr')
plt.show()
```

Вывод



Не существует строгого критерия, какие корреляции считать сильными или слабыми, для каждой задачи стоит принимать решение отдельно и привлекать для этого предметных экспертов. В любом случае, на графике выше видно, что сильные корреляции между независимыми и зависимой переменными отсутствуют. Это означает, что мы не наблюдаем зависимостей от одной переменной или эти зависимости далеки от линейной. В тоже время есть сильная корреляция между признаками Angle и Suction, возможно, в дальнейшем одну из них можно исключить без больших потерь точности при построении модели регрессии.

3.3.6. Масштабирование данных

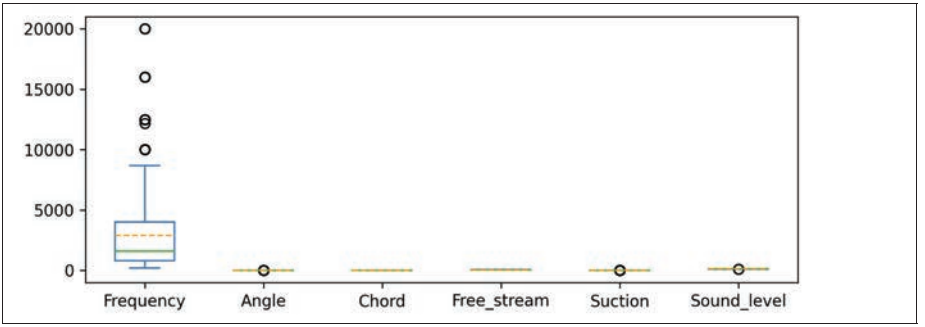
Исходные масштабы (диапазоны значений) могут существенно отличаться, часто на порядки. Для анализа зависимостей важны не абсолютные значения, а относительные (например, если 1-я переменная увеличилась в 2 раза, то вторая уменьшилась в 3 раза). Более того, некоторые методы машинного обучения, которые используют ядра (функции, обрабатывающие данные), могут оперировать лишь в ограниченном диапазоне, который может быть уже, чем диапазон изменения переменных. Поэтому все данные желательно привести к одному интервалу. Это называется масштабирование.

Обычно данные приводят к интервалу $[0, 1]$ или к нормальному виду, чтобы описательная статистика данных соответствовала стандартному нормальному распределению $N(0,1)$, последнее часто называют нормированием данных.

Отобразим исходные масштабы значений данных на графике box-plot.

```
fig, ax = plt.subplots(figsize=(8,3), dpi=150)
data_filled_knn.plot(kind='box', ax=ax,
                     showmeans=True, meanline=True,
                     meanprops={'color': 'orange', 'ls': '--', 'lw': 1})
plt.show()
```

Вывод



Выполним масштабирование в интервал [0,1].

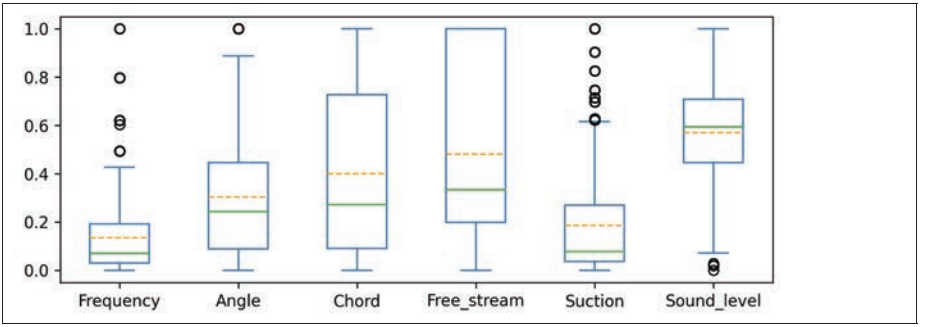
```
scaler1 = MinMaxScaler()
rez = scaler1.fit_transform(data_filled_knn)
data_scalled_01 = pd.DataFrame(rez, columns=numerical_names)
data_scalled_01.describe().loc[['min', 'max']]
```

Вывод

	Frequency	Angle	Chord	Free_stream	Suction	Sound_level
min	0.0	0.0	0.0	0.0	0.0	0.0
max	1.0	1.0	1.0	1.0	1.0	1.0

```
fig, ax = plt.subplots(figsize=(8,3), dpi=150)
data_scalled_01.plot(kind='box', ax=ax,
                     showmeans=True, meanline=True,
                     meanprops={'color': 'orange', 'ls': '--', 'lw': 1})
plt.show()
```

Вывод



Нормируем данные.

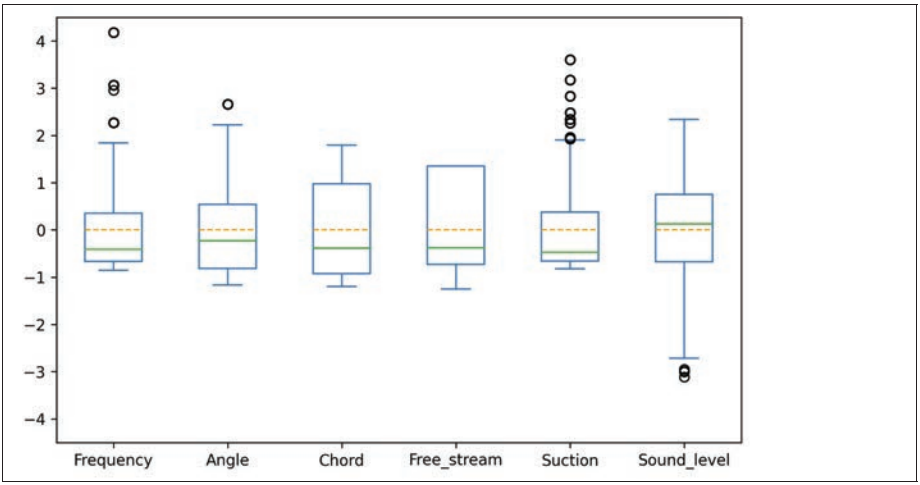
```
scaler2 = StandardScaler()
rez = scaler2.fit_transform(data_filled_knn)
data_scalled_norm = pd.DataFrame(rez, columns=numerical_names)
data_scalled_norm.describe().loc[['mean', 'std']]
```

Вывод

	Frequency	Angle	Chord	Free_stream	Suction	Sound_level
mean	1.654624e-16	3.781997e-17	-2.269198e-16	1.134599e-16	0.000000	-2.968868e-15
std	1.000333e+00	1.000333e+00	1.000333e+00	1.000333e+00	1.000333	1.000333e+00

```
fig, ax = plt.subplots(figsize=(8,5), dpi=150)
data_scalled_norm.plot(kind='box', ax=ax,
                       showmeans=True, meanline=True,
                       meanprops={'color': 'orange', 'ls': '--', 'lw': 1})
ax.set_ylim(-5, 5)
plt.show()
```

Вывод



Масштабированные данные можно использовать на следующих этапах в зависимости от применяемых алгоритмов.

3.3.7. Применение метода и анализ главных компонент

Метод главных компонент, МГК (principal component analysis, PCA) – один из основных способов уменьшить размерность данных, потеряв наименьшее количество информации. Задача анализа главных компонент состоит в том, чтобы найти подпространства меньшей размерности, в ортогональной проекции на которые разброс данных (то есть среднеквадратичное отклонение от среднего значения) максимален [52].

Иными словами, ищется такой поворот осей в пространстве данных, чтобы разброс данных вдоль 1-й новой оси (вдоль 1-й главной компоненты) был наибольший, 2-я ось располагается ортогонально 1-й и вдоль наибольшего разброса данных и т.д. В итоге первые главные компоненты сохраняют наиболее разные значения, т.е. большее количество информации, а последними компонентами можно пренебречь, т.е. оставить меньшее количество координат для описания точки данных. Снижение размерности сопровождается некоторой потерей информацией, но часто незначительной.

После построения главных компонент необходимо оценить процент объясненной дисперсии каждой из компонент. Далее можно оставить столько компонент, сколько обеспечат некоторое допустимое количество информации. Если 1 или 2 компоненты содержат достаточно большое значение объясненной дисперсии, то можно попробовать визуализировать набор данных.

МКГ может выделить в качестве главной компоненты ту переменную, которая имеет большее значение дисперсии из-за своего масштаба. Для корректного результат, необходимо использовать нормализованные данные.

```
names_pca = []
for i in range(5):
  names_pca.append("pca_0"+str(i+1))

pca = PCA()
rez = pca.fit_transform(data_scalled_norm[inputs])
data_pca = pd.DataFrame(rez, columns=names_pca)
data_pca[targets] = data_filled_knn[targets].copy()
data_pca.head(3)
```

Вывод

	pca_01	pca_02	pca_03	pca_04	pca_05	Sound_level
0	-1.094607	-0.025726	-2.119528	-0.125045	-0.523228	126.201
1	-1.683311	-0.068906	-2.070346	-0.180891	0.132961	125.201
2	-1.705723	-0.019108	-2.035151	-0.135433	0.125102	125.951

```
print(pca.explained_variance_ratio_*100)
```

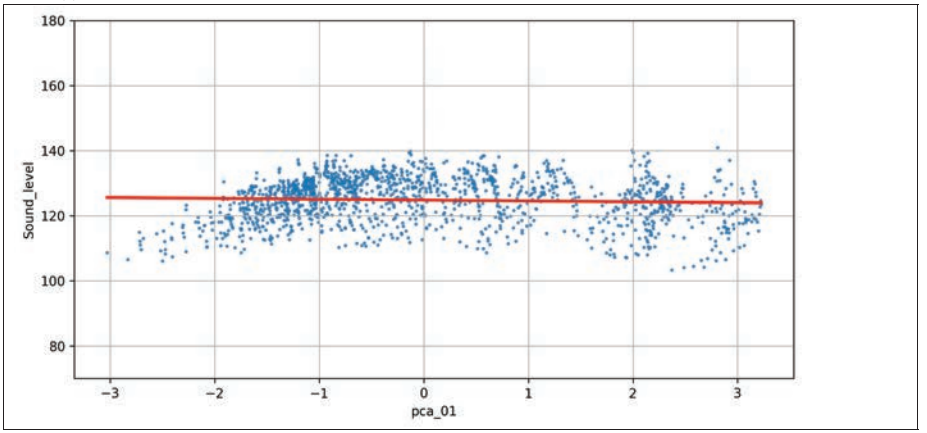
Вывод

```
[41.96070301 22.08658184 18.79030971 13.36718367  3.79522178]
```

Первая главная компонента содержит в себе почти 42% от общей дисперсии значений. Построим график зависимости целевой переменной от 1-й компоненты.

```
fig, ax = plt.subplots(figsize=(10,5))
sns.regplot(data=data_pca, x="pca_01", y='Sound_level', ax=ax,
            scatter_kws={"s": 3}, line_kws={"color": "red"})
ax.grid()
ax.set_ylim(70, 180)
plt.show()
```

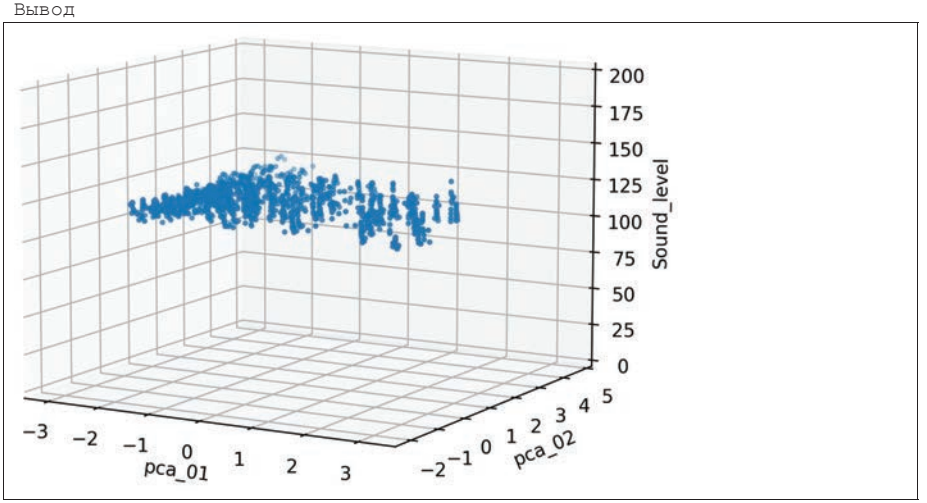
Вывод



Первых две главные компоненты в сумме содержат примерно 63% объясненной дисперсии, построим график в 3х измерениях.

```
fig = plt.figure(figsize=(10,7))
ax = fig.add_subplot(projection='3d')
ax.scatter(data_pca['pca_01'], data_pca['pca_02'],
           data_pca['Sound_level'], s=4)
ax.set_xlabel('pca_01')
ax.set_ylabel('pca_02')
ax.set_zlabel('Sound_level')
ax.set_zlim(0,200)

ax.elev = 10
ax.set_box_aspect(None, zoom=0.9)
plt.show()
```

Вывод



В пространстве меньшей размерности можно заметить, что данные расположены практически в плоскости, а разброс можно объяснить шумом, который почти всегда присутствует в реальных данных. Тогда можно предположить, что линейная модель должна быть достаточно адекватной.

3.3.8. Построение линейной регрессии. Оценка важности переменных

Построим модель линейной регрессии средствами scikit-learn. Преимуществом линейно модели является возможность оценить важность переменных по коэффициентам модели. Знак и значение коэффициентов позволяют интерпретировать модель. Для немасштабированных данных, коэффициенты могут зависеть от масштаба, а не отражать важность переменной, поэтому построим модель, используем нормированные данные.

Также необходимо оценить качество модели с помощью тестовых данных, которые не использовались при построении модели. Такая оценка – это оценка обобщающей способности модели, в отличии от простого запоминания данных. Для этого данные случайным образом разбиваются на обучающие и тестовые, обычно в пропорции 75% и 25%, соответственно.

Для оценки качества построенной модели используем две меры: среднеквадратическую ошибку (mean squared error, MSE) и коэффициент детерминации R^2.

```
train, test = train_test_split(data_scalled_norm, test_size=0.25)


x = train[inputs]
y = train[targets]
linear_model = LinearRegression()
linear_model.fit(x, y)

print("Обучающая выборка")
predict_train = linear_model.predict(x)
print('MSE =', mean_squared_error(y, predict_train))
print('R2 =', r2_score(y, predict_train))
print("Тестовая выборка")
predict_test = linear_model.predict(test[inputs])
print('MSE =', mean_squared_error(test[targets], predict_test))
print('R2 =', r2_score(test[targets], predict_test))
```

Вывод

```
Обучающая выборка
MSE = 0.4919670338749368
R2 = 0.5108988484882493
Тестовая выборка
MSE = 0.45926612423518187
R2 = 0.5321794529646748
```

Коэффициент детерминации – это доля дисперсии зависимой переменной, объясняемая рассматриваемой моделью зависимости, то есть объясняющими переменными. Коэффициент детерминации принимает значения от 0 до 1. Чем

ближе значение коэффициента к 1, тем сильнее зависимость. При оценке регрессионных моделей это интерпретируется как соответствие модели данным.

Для приемлемых моделей предполагается, что коэффициент детерминации должен быть хотя бы не меньше 50% [53].

Оценки коэффициента детерминации для нашей модели позволяет считать ее приемлемой по качеству, а значит мы можем сделать выводы о значимости переменных по коэффициентам регрессии.

```
print('Коэффициенты при x (slope):')
for i in range(5):
  print('  ', inputs[i],':',linear_model_2.coef_[0][i])
print('Свободный член (intercept)', linear_model_2.intercept_)
```

Вывод

```
Коэффициенты при x (slope):
   Frequency : -0.570612384683523
   Angle : -0.33395177092686257
   Chord : -0.49120844724303525
   Free_stream : 0.20397461769857392
   Suction : -0.3029974987706032
Свободный член (intercept) [0.01082636]
```

Полученная зависимость в виде формулы:

$$Sound\_level = -0.57 \cdot Frequency - 0.33 \cdot Angle +$$
$$-0.49 \cdot Chord + 0.20 \cdot Free\_stream\ - 0.30 \cdot Suction + 0.01$$

По значениям коэффициентов регрессии переменные имеют примерный равный вклад в линейную модель, наибольший имеют переменные Frequency и Chord. Тем не менее, коэффициент R^2 принимает минимально допустимое значение, а значит есть смысл строить более сложную модель. Построение моделей машинного обучения не является предметом обсуждения в данном пособии.

3.4. EDA в задаче классификации

Пример проведения EDA для задачи классификации проводится на примере классического набора данных Ирисы Фишера (Iris Dataset). Задача поставлена как задача классификации, требуется построить способ назначения меток классов по признаковому описанию объектов.

Ирисы Фишера – набор данных для задачи классификации, на примере которого Рональд Фишер в 1936 году продемонстрировал работу разработанного им метода дискриминантного анализа. Иногда его также называют ирисами Андерсона, так как данные были собраны американским ботаником Эдгаром Андерсоном. Этот набор данных стал классическим и часто используется в литературе для иллюстрации работы различных статистических алгоритмов [54].

Набор данных Ирисы Фишера состоит из данных о 150 экземплярах цветка Ириса, по 50 экземпляров каждого из трех видов: Ирис щетинистый (Iris setosa), Ирис виргинский (Iris virginica) и Ирис разноцветный (Iris versicolor). Для каждого экземпляра измерялись четыре характеристики (в сантиметрах):

– длина наружной доли околоцветника (англ. sepal length),
– ширина наружной доли околоцветника (англ. sepal width),
– длина внутренней доли околоцветника (англ. petal length),
– ширина внутренней доли околоцветника (англ. petal width).

Набор данных представлен практически по всех репозиториях, а также в пакетах Python для анализа данных, например, в пакете Seaborn.

Для проведения EDA импортируем следующие пакеты.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import accuracy_score, f1_score, classification_re-
port
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

from IPython.core.display import display, HTML
```

3.4.1. Импорт данных

Импортируем данные из пакета Seaborn. Набор данных в пакете полностью подготовлен для работы, тем не менее, проконтролируем корректность импорта, показав начало и конец DataFrame и сформировав отчет с помощью функции `pd.info()`.

```
raw = sns.load_dataset('iris')
display(raw.head(3))
display(raw.tail(3))
```

Вывод

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa

	sepal_length	sepal_width	petal_length	petal_width	species
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

```
raw.info()
```

Вывод

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

В данных нет пропусков. Есть названия столбцов и имена меток классов. Создадим списки числовых и категориальных данных, списки независимых и целевых переменных.

```
numerical_names = ['sepal_length', 'sepal_width', 'petal_length',
'petal_width']
categorical_names = ['species']
inputs = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
targets = ['species']
```

3.4.2. Визуализация данных в табличном виде. Описательная статистика

Как и в задаче регрессии цель этапа – проверить объем, форматы, диапазоны данных, наличие пропусков и сделать выводы о качестве данных.

Описательная статистика для количественных данных.

```
raw[numerical_names].describe()
```

Вывод

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Описательная статистика для категориальных данных.

```
raw[categorical_names].describe()
```

Вывод

	species
count	150
unique	3
top	setosa
freq	50

Среднее значение независимых признаков в большинстве случаев совпадает с медианой, следовательно распределение может соответствовать нормальному закону распределения, но требуется проверка.

Диапазоны значений, как и в других задачах, нужно показать предметным экспертам для анализа корректности сбора данных.

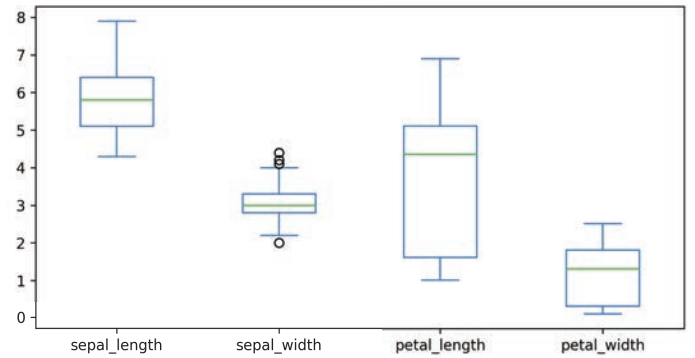
3.4.3. Визуальный анализ распределений и разброса данных

Построим графики box-plot для количественных данных и bar-plot для категориальных данных. В задаче классификации стоит построить отдельные графики для каждого класса.

Вначале покажем разброс данных без разбивки по классам.

```
fig, ax = plt.subplots(figsize=(8,4))
raw.plot(kind='box', ax=ax)
plt.show()
```

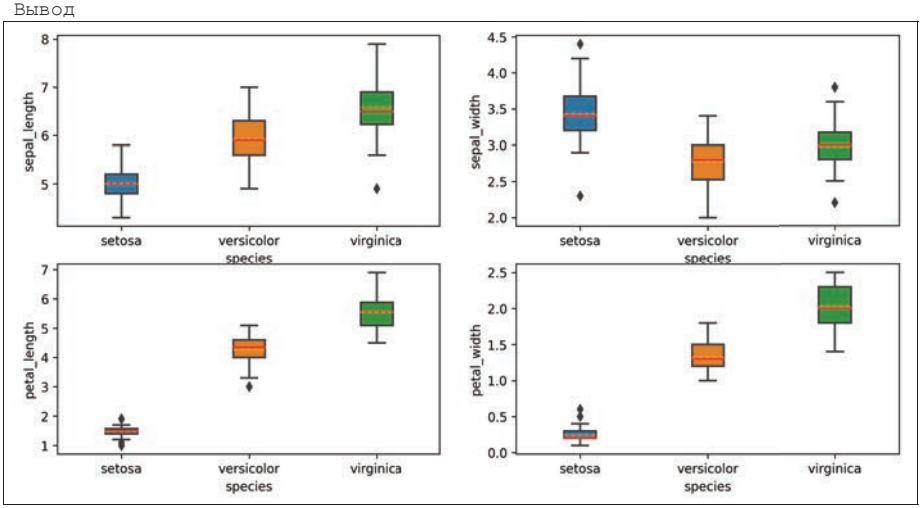
Вывод



Построим график box-plot для каждого признака с разбивкой по классам.

```
fig, ax = plt.subplots(figsize=(12,6), ncols=2, nrows=2)
for i in range(2):
  for j in range(2):
    sns.boxplot(data=raw, y=numerical_names[j + 2*i], x=targets[0],
                showmeans=True, meanline=True,
                meanprops={'color': 'orange', 'ls': '--', 'lw': 1},
                medianprops={'color': 'red', 'ls': '-', 'lw': 1.5},
                ax=ax[i][j], width=0.25)
plt.show()
```

Вывод

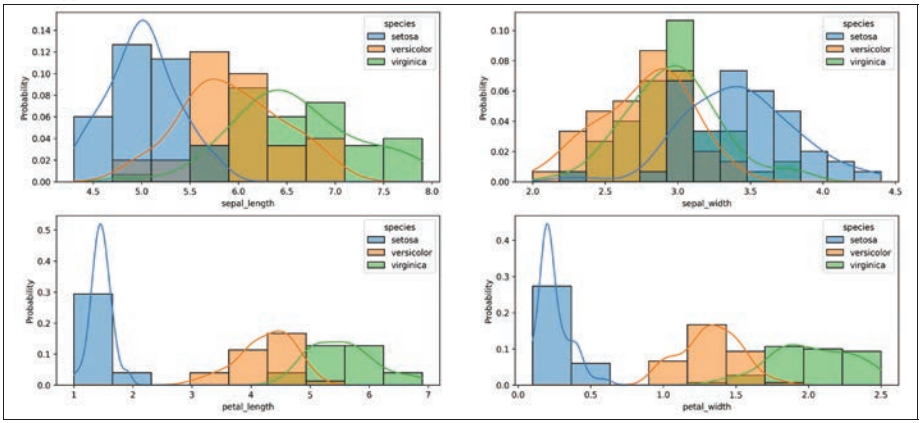


По графикам мы видим, что разброс значений признаков имеет свои особенности, практически всегда значения для класса Setosa имеют значения меньше, чем у других классов, кроме признака sepal_width. Медианы классов Versicolor и Virginica различаются, но разброс таков, что значения часто пересекаются.

Оценим плотности распределения вероятностей значений признаков для каждого класса.

```
fig, ax = plt.subplots(figsize=(18,8), ncols=2, nrows=2)
for i in range(2):
  for j in range(2):
    sns.histplot(data=raw, x=numerical_names[j + 2*i],
                 hue = targets[0],
                 ax = ax[i][j],
                 kde=True, stat='probability')
plt.show()
```

Вывод

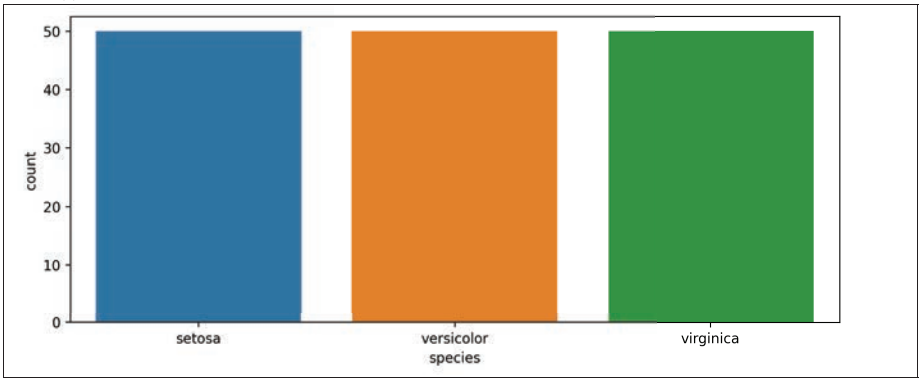


Частные распределения близки к нормальному у всех признаков для всех классов. Более того, распределения таковы, что по распределениям можно эффективно принимать решения о принадлежности к классам, используя вероятностные подходы. По крайней мере, по трем признакам их четырех классы вполне разделимы.

Для целевой переменной с метками классов необходимо построить график bar-plot, чтобы оценить сбалансированность выборки.

```
fig, ax = plt.subplots(figsize=(10,4), dpi=150)
sns.countplot(data=raw, x=categorical_names[0], ax=ax)
plt.show()
```

Вывод



В данном наборе данные сбалансированы, т.е. есть возможность построить модель машинного обучения, которая в равной степени сможет обобщить данные каждого из классов.

3.4.4. Заполнение пропусков

Процедура заполнения пропусков для задачи классификации не отличается от таковой для задачи регрессии. Единственное отличие – стоит заполнять пропуски на основе анализа данных соответствующего класса, а не по всему набору данных.

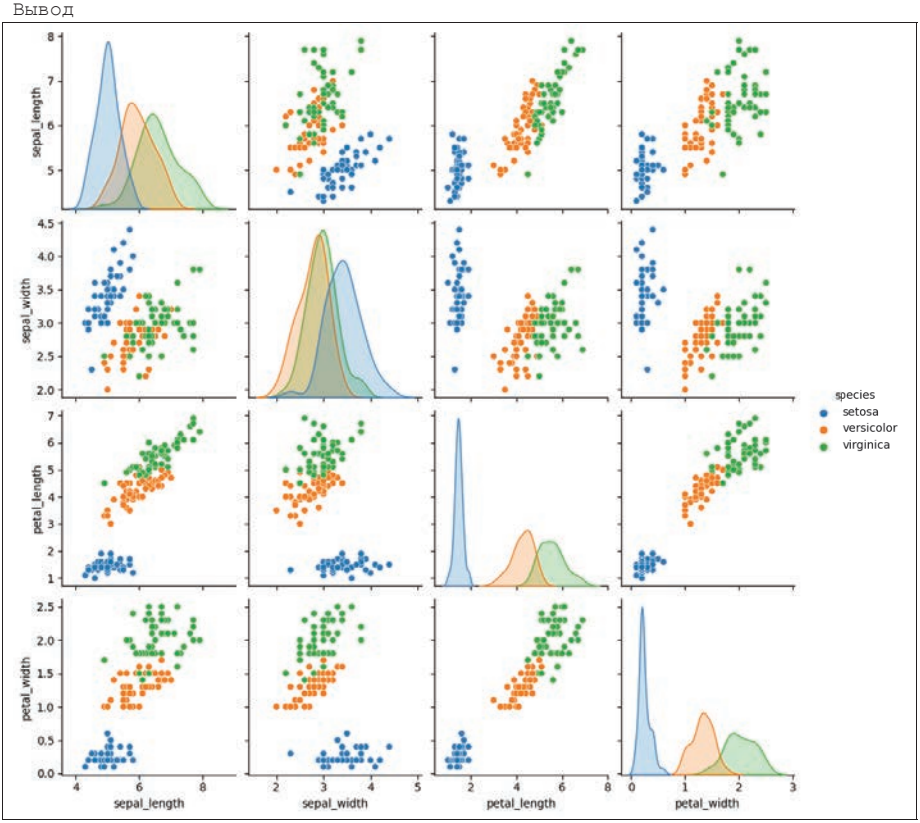
В исследуемом наборе пропущенных значений нет, пропусти этот этап.

3.4.5. Визуализация парных графиков. Анализ корреляций

В отличии от задачи регрессии, парные графики для задачи регрессии должны отражать не зависимость изменения одной переменной от изменений другой, а показать возможность отделить значения переменных одного класса от значений других классов. Таким образом, нужно построить парные графики с разбивкой по группам.

```
sns.pairplot(data = raw, hue = targets[0], kind='scatter')
plt.show()
```

Вывод



Из графиков видно, что класс Setosa практически для любых пар признаков линейно отделим от других классов. В тоже время классы Versicolor и Virginica линейно не разделимы, но для каждой пары признаков имеют уникальное пересечение, что требует построение модели, учитывающей все признаки.

Построим матрицу и график корреляций.

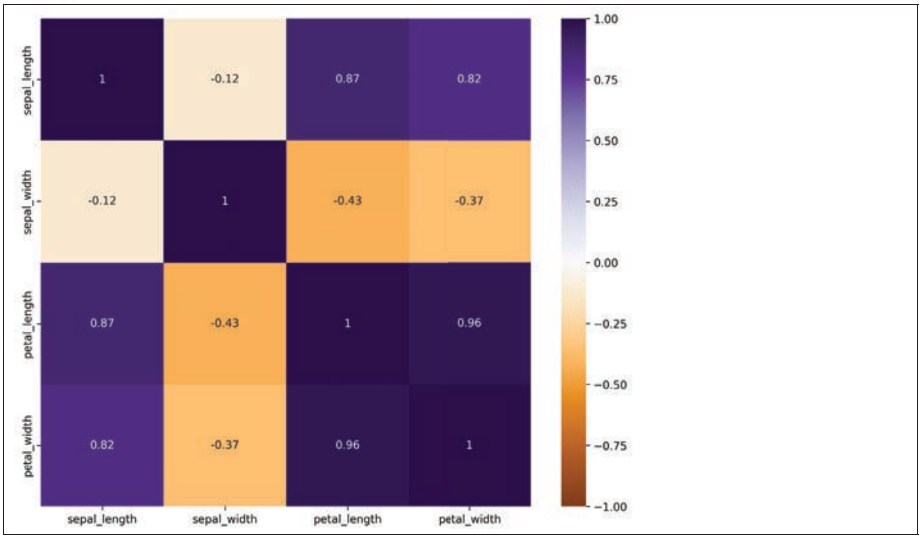
```
raw[numerical_names].corr()
```

Вывод

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.000000	-0.117570	0.871754	0.817941
sepal_width	-0.117570	1.000000	-0.428440	-0.366126
petal_length	0.871754	-0.428440	1.000000	0.962865
petal_width	0.817941	-0.366126	0.962865	1.000000

```
fig, ax = plt.subplots(figsize=(10,8))
sns.heatmap(data=raw[numerical_names].corr(),
            annot=True, vmin=-1,vmax=1,
            cmap='PuOr')
plt.show()
```

Вывод



Существуют сильные корреляции между многими признаками, а значит стоит проверить их важность и часть из них, возможно, стоит исключить.

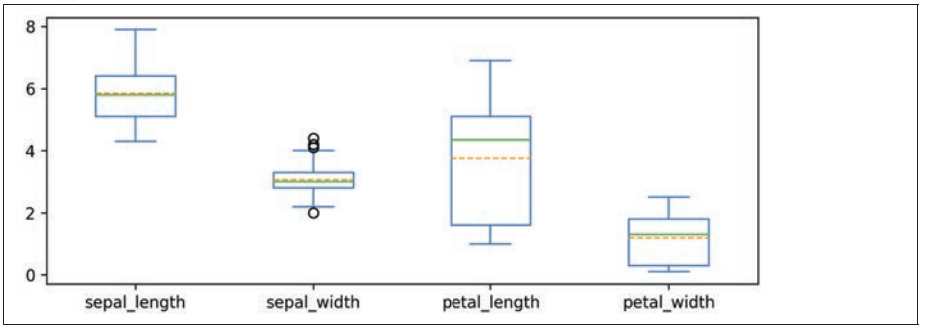
3.4.6. Масштабирование данных

Выполним масштабирование, как и в задаче регрессии. Масштабированные данные можно использовать на следующих этапах в зависимости от применяемых алгоритмов.

Исходные масштабы.

```
fig, ax = plt.subplots(figsize=(8,3), dpi=150)
raw.plot(kind='box', ax=ax,
                     showmeans=True, meanline=True,
                     meanprops={'color': 'orange', 'ls': '--', 'lw': 1})
plt.show()
```

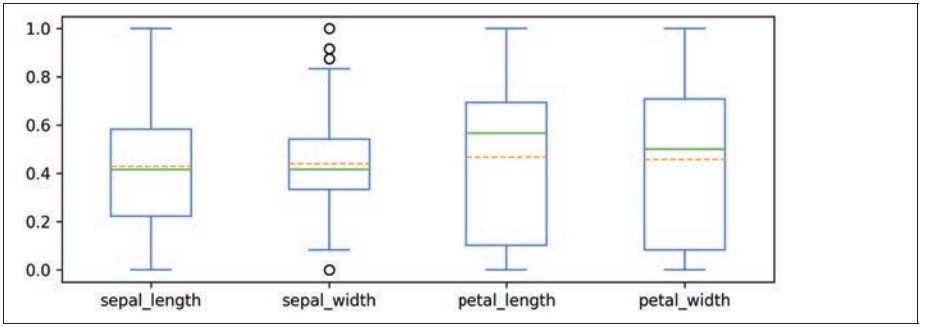
Вывод



Масштабирование в интервал $[0, 1]$.

```
scaler1 = MinMaxScaler()
rez = scaler1.fit_transform(raw[numerical_names])
data_scalled_01 = pd.DataFrame(rez, columns=numerical_names)
data_scalled_01[targets[0]] = raw[targets[0]].copy()

fig, ax = plt.subplots(figsize=(8,3), dpi=150)
data_scalled_01.plot(kind='box', ax=ax,
                     showmeans=True, meanline=True,
                     meanprops={'color': 'orange', 'ls': '--', 'lw': 1})
plt.show()
```

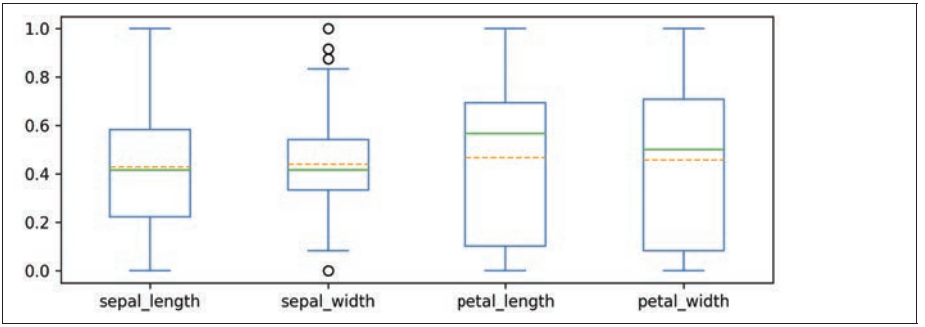
Вывод



Нормирование данных.

```
scaler2 = StandardScaler()
rez = scaler2.fit_transform(raw[numerical_names])
data_scalled_norm = pd.DataFrame(rez, columns=numerical_names)
data_scalled_norm[targets[0]] = raw[targets[0]].copy()

fig, ax = plt.subplots(figsize=(8,5), dpi=150)
data_scalled_norm.plot(kind='box', ax=ax,
                       showmeans=True, meanline=True,
                       meanprops={'color': 'orange', 'ls': '--', 'lw': 1})
ax.set_ylim(-5, 5)
plt.show()
```

Вывод



3.4.7. Применение метода и анализ главных компонент

Построим главных компоненты и оценим процент объясненной дисперсии каждой из компонент аналогично тому, как делали в задаче регрессии. При построении диаграмм рассеяния первой и первых двух главных компонент будем использовать разные цвета для разных классов, чтобы оценить возможность разделения классов по значениям признаков.

```
names_pca = []
for i in range(4):
  names_pca.append('pca_0'+str(i+1))

pca = PCA()
rez = pca.fit_transform(data_scalled_norm[inputs])
data_pca = pd.DataFrame(rez, columns=names_pca)
data_pca[targets[0]] = raw[targets[0]].copy()

print(pca.explained_variance_ratio_*100)
```

Вывод

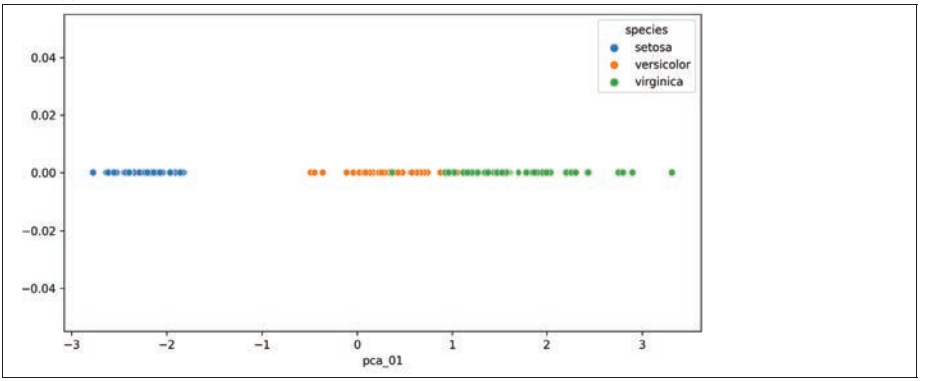
```
[72.96244541 22.85076179  3.66892189  0.51787091]
```

Как видно из отчета, первая компонента объясняет почти 73% дисперсии данных, а первые две – почти 96%. Таким образом можно надеяться, что

визуализация пространства главных компонент меньшей размерности адекватно отразит структуру классов.

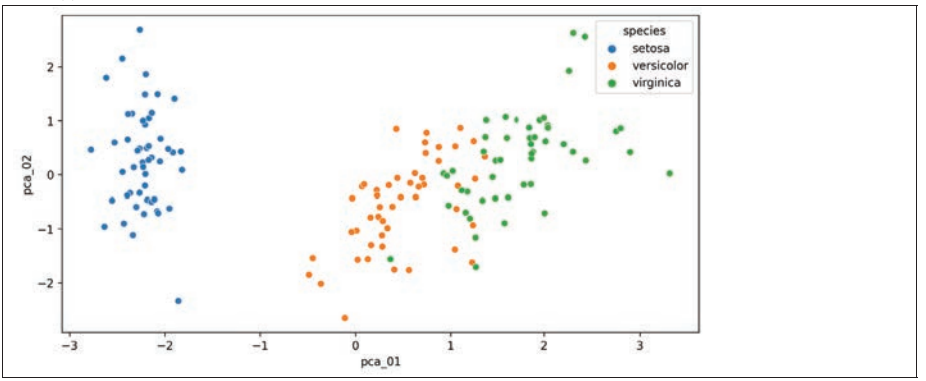
```
fig, ax = plt.subplots(figsize=(10,5))
sns.scatterplot(data=data_pca, x="pca_01", y=0,
                hue=targets[0], ax=ax)
plt.show()
```

Вывод




```
fig, ax = plt.subplots(figsize=(10,5))
sns.scatterplot(data=data_pca, x="pca_01", y="pca_02",
                hue=targets[0], ax=ax)
plt.show()
```

Вывод



Как видно из графиков, даже по одной главной компоненте можно принять решение о разделении классов. Безошибочно можно классифицировать класс Setosa, для оставшихся ошибка будет невелика. Ошибку можно снизить если использовать большее число компонент, но тогда придется строить нелинейные разделяющие функции.

3.4.8. Построение классификатора на основе метода случайного леса. Оценка важности переменных

Для оценки важности переменных необходимо выбрать метод, который помимо самой модели классификации способен оценить, какие признаки вносят больший вклад в принимаемые решения. Одним из таких подходов является метод построения деревьев решений, в который используются информационные критерии для выбора вершин ветвления в дереве так, чтобы в верхних узлах находились признаки, дающие наибольший прирост информации.

Для того чтобы делать адекватные выводы о важности переменных, выбранный метод классификации должен показать высокую точность обучения. Одним из наиболее эффективных подходов для решения задач классификации на основе деревьев решений является метод случайного леса (Random Forest). Воспользуемся его реализацией из пакета scikit-learn.

Для оценки качества классификации будем использовать случайное разделение на обучающие и тестовые данные в пропорции 75% и 25%, соответственно. Для оценки качества модели используем точность (accuracy). Дополнительно сформируем отчет с поклассовой оценкой мер точности (precision), полноты (recall) и f1-меры.

```
train, test = train_test_split(data_scalled_norm, test_size=0.2)
x = train[inputs]
y = train[targets]

rf = RandomForestClassifier()
rf.fit(x, y.values.ravel())
print("Обучающая выборка")
predict_train = rf.predict(x)
print('Accuracy =', accuracy_score(y, predict_train)*100)

print("Тестовая выборка")
predict_test = rf.predict(test[inputs])
print('Accuracy =', accuracy_score(test[targets], predict_test)*100)
```

Вывод

```
Обучающая выборка
Accuracy = 100.0
Тестовая выборка
Accuracy = 93.33333333333333
```

```
print(classification_report(test[targets], predict_test))
```

Вывод

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       0.86      1.00      0.92         6
   virginica       1.00      0.91      0.95        11

    accuracy                           0.97        30
   macro avg       0.95      0.97      0.96        30
weighted avg       0.97      0.97      0.97        30
```

Как видно из оценок точности, классификация выполняется хорошо, а значит оценки важности переменных могут адекватно отражать вклад каждого из признаков.

```
print("Важность переменных, %:")
for i in range(4):
  print(' ',numerical_names[i],':',
        rf.feature_importances_[i]*100)
```

Вывод

```
Важность переменных, %:
  sepal_length : 10.370175602594845
  sepal_width : 2.3356353224145634
  petal_length : 46.193140104747414
  petal_width : 41.101048970243184
```

Как видно из отчета, наибольший вклад в классификацию алгоритма Random Forest вносят признаки petal_length и petal_width (более 87%), а потому в дальнейшем им стоит уделить особое внимание.

Контрольные вопросы

1. Для чего используется разведывательный анализ данных?
2. Какие преимущества дает EDA?
3. Назовите риски отказа от EDA.
4. Сравните одномерную, двумерную и многомерную визуализацию.
5. Для чего решается задача снижения размерности?
6. Опишите общую процедуру EDA.
7. Какие типы данных используются в анализе данных и зачем знать типы?
8. Что такое оценка центрального положения и вариабельность?
9. В каких случаях оценка медианы предпочтительнее оценки среднего? В каких наоборот?
10. Перечислите состав порядковых статистик.
11. Какие столбиковые диаграммы психологически воспринимаются как прогресс, какие как рост показателя?
12. Почему корреляция не означает причинно-следственную связь?
13. В чем преимущество построения тепловой карты вместо обычной корреляционной матрицы?
14. Какие способы визуального представления результатов используются в кластерном анализе?
15. Для чего используется дендрограмма?
16. В чем суть метода локтя?
17. Назовите основные шаги EDA в задаче регрессии?
18. Какие выводы можно сделать по описательной статистике набора данных?
19. Перечислите способы работы с пропусками, их плюсы и минусы.

20. Как интерпретировать сильную корреляцию между независимыми признаками?

21. В каких случаях можно доверять графикам одной или двух главных компонент?

22. Каким образом по модели линейной регрессии можно сделать заключение о важности переменных?

23. Какие принципиальные отличия EDA для задачи классификации в сравнении с EDA для задачи регрессии?

24. Что можно узнать из гистограмм, построенных по отдельным классам?

25. Какой алгоритм машинного обучения позволяет выяснить важность переменных в задаче классификации?

Заключение

В учебном пособии рассмотрены основные понятия анализа данных, включая особенности анализа больших данных для задач машинного обучения. Показаны инструменты визуального и разведочного анализа. Представлены основные фреймворки для анализа данных на языке Python.

Рассмотренные в главе 3 примеры разведочного анализа данных для задач регрессии и классификации являются универсальными шаблонами для выполнения прикладных исследований данных. А рекомендации по визуализации, анализу и интерпретации результатов анализа данных могут быть использованы в любых других прикладных и исследовательских задач.

Список использованных источников

1. Стратегия развития рынка больших данных 2024. Ассоциация больших данных РФ, 2023. https://rubda.ru/wp-content/uploads/2023/07/strategiya-rynka_abd_2023.pdf
2. Макшанов А. В. Большие данные. Big Data : учебник для вузов / А. В. Макшанов, А. Е. Журавлев, Л. Н. Тындыкарь. – 2-е изд., стер. - Санкт-Петербург : Лань, 2022. - 188 с.
3. The Demise of Big Data, Its Lessons and the State of Things to Come. Агентство Gartner. https://www.gartner.com/en/documents/3115022
4. Грас Д. Data Science. Наука о данных с нуля: Пер. с англ. - 2-е изд., перераб. и доп. - СПб.: БХВ-Петербург, 2021. - 416 с.
5. Низаметдинов, Ш. У. Анализ данных : учебное пособие / Ш. У. Низаметдинов, В. П. Румянцев. - Москва : НИЯУ МИФИ, 2012. — 288 с.
6. Барсегян А. А. Анализ данных и процессов: учеб. пособие / А. А. Барсегян, М. С. Куприянов, И. И. Холод, М. Д. Тесс, С. И. Елизаров. - 3-е изд., перераб. и доп. - СПб .: БХВ-Петербург, 2009. - 512 с.
7. Тьюки, Д.У. Анализ результатов наблюдений : Разведоч. анализ / Дж. Тьюки; Пер. с англ. А. Ф. Кушнира и др. - Москва : Мир, 1981. - 693 с.
8. Брюс П. Практическая статистика для специалистов Data Science: Пер. с англ. / П. Брюс, Э. Брюс, П. Гедек. - 2-е изд., перераб. и доп. - СПб .: БХВ-Петербург, 2021. - 352 с.
9. Паклин Н.Б. Бизнес-аналитика: от данных к знаниям [Текст] : учебное пособие / Н. Паклин, В. Орешков. - Москва [и др.] : Питер, 2013. - 701 с.
10. Tufte E.R. The Visual Display of Quantitative Information / Cheshire, CT: Graphics Press, 1983 -210 p.
11. Бурков А. Инженерия машинного обучения / А. Бурков ; перевод с английского А. А. Слинкина. - Москва : ДМК Пресс, 2022. - 306 с.
12. Харрисон М. Машинное обучение : карманный справочник : краткое руководство по методам структурированного машинного обучения на Python / Москва : Диалектика ; Санкт-Петербург : Диалектика, 2020. - 312 с.
13. Бишоп К.М. Распознавание образов и машинное обучение / Москва ; Санкт-Петербург : Диалектика, 2020. - 960 с.
14. Gift N., Deza A. Practical MLOps / O'Reilly Media, Inc., 2021 – 460 p.
15. Рашка, Себастьян. Python и машинное обучение : крайне необходимое издание по новейшей предсказательной аналитике для более глубокого понимания методологии машинного обучения / Москва : ДМК Пресс, 2017. - 414 с.
16. Сопов Е.А. Обобщенный метод синтеза гиперэвристических эволюционных алгоритмов оптимизации сложных систем : диссертация доктора технических наук : 05.13.01 / Красноярск, 2021. - 325 с.
17. Хайкин С. Нейронные сети [Текст] : полный курс : Изд. 2-е, испр. - Москва ; Санкт-Петербург : Диалектика, 2019. - 1103 с.
18. Новак В. Математические принципы нечеткой логики / 2-е изд., доп. и испр. - Москва : ФИЗМАТЛИТ, 2006. - 347 с.

19. Саймон Д. Алгоритмы эволюционной оптимизации / Москва : ДМК Пресс, 2020. — 1002 с.

20. Морроу Д. Как вытащить из данных максимум : навыки аналитики для неспециалистов / Москва : Альпина Паблишер, 2022. - 255 с.

21. Шарден, Б. Крупномасштабное машинное обучение вместе с Python : учись быстро создавать мощные модели машинного обучения и развертывать крупномасштабные приложения прогнозирования / Москва : ДМК Пресс, 2018. - 357 с.

22. The Apache Hadoop software library, https://hadoop.apache.org

23. Чак .Hadoop в действии [Текст] / Чак Лэм. - Москва : ДМК Пресс, 2012. - 423 с.

24. Рамальо Л. Python. К вершинам мастерства / Москва : ДМК Пресс, 2016. - 768 с.

25. Python Enhancement Proposals, https://peps.python.org

26. Мартелли А. Python : справочник : полное описание языка : 3-е изд. Москва ; Диалектика ; Санкт-Петербург : Диалектика, 2020. - 892 с.

27. Isolated Python environments, https://virtualenv.pypa.io

28. Markdown Guide, https://www.markdownguide.org

29. Маккинли, У. Python и анализ данных / Москва : ДМК Пресс, 2015. - 482 с.

30. NumPy, https://numpy.org

31. Тихонов А.И. Научно-технические расчеты на Python : учебное пособие / Национальный исследовательский университет "МЭИ". - Москва : Изд-во МЭИ, 2020. - 313 с.

32. Pandas - Python Data Analysis Library, https://pandas.pydata.org

33. Хейдт М. Изучаем pandas : высокопроизводительная обработка и анализ данных в Python / 2-е изд. - Москва ; Гевисста : ДМК Пресс, 2019. - 681 с.

34. Matplotlib - Visualization with Python, https://matplotlib.org

35. Плас Д. Python для сложных задач, наука о данных: и машинное обучение / Санкт-Петербург [и др.] : Питер, 2019. – 572 с.

36. Matplotlib. Anatomy of a figure, https://matplotlib.org/stable/gallery/showcase/anatomy.html

37. Seaborn: statistical data visualization, https://seaborn.pydata.org

38. UCI Airfoil Self-Noise Dataset, https://archive.ics.uci.edu/dataset/291/airfoil+self+noise

39. McGill R. Tukey J. Variations of Box Plots / The American Statistician. 32 (1), 1978.

40. Википедия. Ящик с усами, https://ru.wikipedia.org/wiki/Ящик_с_усами

41. Митропольский А. К.. Техника статистических вычислений. - 2 изд., перераб. и доп.. - М.: Наука, 1971. - 576 с.

42. Tukey J. The future of data analysis / The Annals of Mathematical Statistics, 1962. – 68 p.

43. Tukey J. Exploratory data analysis / Reading, Mass. : Addison-Wesley Pub. Co., 1977. – 688 p.

44. Гмурман В.Е. Теория вероятностей и математическая статистика : учебное пособие для студентов вузов / В. Е. Гмурман. - 12-е изд. / перераб. - Москва : Юрайт : Высш. образование, 2009. - 478 с.

45. Митина Т.В. Многомерные случайные величины. Корреляционный анализ : учебное пособие : Дубна : Государственный университет "Дубна", 2021. - 61 с.

46. Википедия. Кореляция, https://ru.wikipedia.org/wiki/корреляция

47. Тарасенко Ф.П. Прикладной системный анализ [Текст] : учебное пособие / Ф. П. Тарасенко. - 2-е изд., перераб и доп. - Москва : КНОРУС, 2016. - 319 с.

48. Мандель И.Д. Кластерный анализ / И. Д. Мандель; [Предисл. Б. Г. Миркина]. - Москва : Финансы и статистика, 1988. - 176 с.

49. Жамбю М. Иерархический кластер-анализ и соответствия. — М.: Финансы и статистика, 1988. - 345 с.

50. scikit-learn Machine Learning in Python, https://scikit-learn.org

51. Wikipedia. NACA airfoil, https://en.wikipedia.org/wiki/NACA_airfoil

52. Айвазян С.А., Бухштабер В. М., Енюков И. С., Мешалкин Л. Д. Прикладная статистика. Классификация и снижение размерности / М.: Финансы и статистика, 1989 - 607 с.

53. Айвазян С.А. Прикладная статистика и основы эконометрики / Гос. ун-т, Высш. шк. экономики. - Москва : ЮНИТИ, 1998. - 1022 с.

54. Википедия. Ирисы Фишера, https://ru.wikipedia.org/wiki/ирисы_фишера

ФЗ № 436-ФЗ	Издание не подлежит маркировке в соответствии с п. 1 ч. 2 ст. 1

Учебное издание

Сопов Евгений Александрович

ТЕХНОЛОГИИ ОБРАБОТКИ БОЛЬШИХ ДАННЫХ НА PYTHON

УЧЕБНИК

ООО «Научно-издательский центр ИНФРА-М»
127214, Москва, ул. Полярная, д. 31В, стр. 1
Тел.: (495) 280-15-96, 280-33-86. Факс: (495) 280-36-29
E-mail: books@infra-m.ru http://www.infra-m.ru

Подписано в печать 29.12.2023.
Формат 60×90/16. Бумага офсетная. Гарнитура Petersburg.
Печать цифровая. Усл. печ. л. 9,81.
Тираж 500 экз. (I – 50). Заказ № 00000

ТК 821271-2131445-291223

Отпечатано в типографии ООО «Научно-издательский центр ИНФРА-М»
127214, Москва, ул. Полярная, д. 31В, стр. 1
Тел.: (495) 280-15-96, 280-33-86. Факс: (495) 280-36-29