РОСЖЕЛДОР

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Ростовский государственный университет путей сообщения»

(ФГБОУ ВО РГУПС)

О. В. Игнатьева, М. И. Мялова

БОЛЬШИЕ ДАННЫЕ И ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗ

Учебно-методическое пособие
для лабораторных работ

Ростов-на-Дону
РГУПС
2022

УДК 004.4(07) + 06

Рецензент – кандидат технических наук, доцент В. В. Жуков

Игнатьева, О. В.

Большие данные и интеллектуальный анализ : учебно-методическое пособие для лабораторных работ / О. В. Игнатьева, М. И. Мялова ; ФГБОУ ВО РГУПС. – Ростов-на-Дону : РГУПС, 2022. – 155 с.

Приведены требования к оформлению и указания по организации выполнения лабораторных работ по дисциплине «Большие данные и интеллектуальный анализ». Содержит задания и методические рекомендации для выполнения лабораторных работ.

Для студентов направлений «Информатика и вычислительная техника» и «Информационные системы и технологии» в целях углубленного изучения анализа больших данных на аудиторных занятиях и самостоятельного изучения материала по дисциплине «Большие данные и интеллектуальный анализ», а также для всех обучающихся магистратуры, бакалавриата, специалитета различных направлений и спецкурсов, изучающих дисциплины по большим данным, интеллектуальному анализу, технологиям искусственного интеллекта.

Одобрено к изданию кафедрой «Вычислительная техника и автоматизированные системы управления».

СОДЕРЖАНИЕ

Лабораторная работа № 1. Введение в науку о данных. Установка инструментов для анализа данных

1 Методические рекомендации

1.1 Python для анализа данных

Анализ данных − процесс исследования, фильтрации, преобразования и моделирования данных с целью извлечения полезной информации и принятия решений.

Анализ больших данных включает:

1 Сбор данных.

2 Подготовка и очистка данных.

3 Нахождение скрытых зависимостей.

4 Разработка моделей.

5 Прочее

Специалисты по анализу данных:

- BI-аналитик - решает срочные задачи, работает с базой данных, готовит дашборды, отвечает за визуализацию данных;
- аналитик - отлично знает предметную область, анализирует метрики, проводит эксперименты, составляет прогнозы, глубоко закапывается в имеющиеся данные;
- Data Scientist - структурирует и анализирует большие объемы данных, применяет машинное обучение для предсказания событий и обнаружения неочевидных закономерностей.

Python был разработан в конце 1989 г. Гуидо ван Россумом (Guido van Rossum) во время рождественских каникул, когда его исследовательская лаборатория была закрыта и ему просто некуда было деваться. Он позаимствовал многие средства программирования, присущие другим языкам.

IPython − мощный инструмент для работы с языком python. Jupiter notebook − графическая веб-оболочка для IPython, которая расширяет идею консольного подхода к интерактивным вычислениям, популярнейшая бесплатная интерактивная оболочка, позволяющая объединить код на python, текст и диаграммы и распространять их для других пользователей.

Рекомендации для изучения языка Python для анализа данных:

1 Освоение основных принципов программирования.

Например, М. Лутц «Изучаем Python», W. McKenney «Python for Data Analysis».

2 Изучение библиотек, необходимых для анализа данных.

− NumPy и Pandas − основные, для вычислений (документация NumPy, документация Pandas).

− Matplotlib и Seaborn − для визуализации данных (документация Matplotlib, документация Seaborn).

− Scipy и StatsModels − для статистического анализа (документация Scipy).

− SciKit − для работы с методами машинного обучения.

3 Закрепление знаний на практике: kaggle.com, pythonchallenge.com.

Запуск программы на Python

- Пакетный режим

1 Создать файл test.py с исходным кодом (например, в Блокноте);

2 Запустить файл через консоль с помощью команды: > python test.py

- Интерактивный режим

В интерактивный режим можно войти, набрав в командной строке > python

- Работа в оболочке Jupiter Notebook

В Jupiter Notebook есть два режима: Command Mode (можно делать операции с ячейками ноутбука: добавлять, удалять, разделять, запускать и тд.) и Edit Mode (работа в самой ячейке, написание кода).

Esc − перейти в режим Command Mode;

Y \ M − быстро переключаться между типом ячеек (M − markdown, Y − code).

CTRL Enter − выполнить ячейку (на Windows);

Блоки в python всегда выделяются отступом.

1.2 Среды разработки

Среды разработки для анализа данных: дистрибутив Anaconda, Jupyter Notebook, Google, Colaboratory

Версии Python

На сегодняшний день существуют две версии Python – это Python 2 и Python 3, у них отсутствует полная совместимость друг с другом. На данный момент вторая версия Python ещё широко используется, но, судя по изменениям, которые происходят, со временем, она останется только для запуска старого кода. Мы будем Python 3, и, в дальнейшем, если где-то будет встречаться слово Python, то под ним следует понимать Python 3. Случаи применения Python 2 будут специально оговариваться.

Установка Python

Для установки интерпретатора Python на ваш компьютер, первое, что нужно сделать − это скачать дистрибутив. Загрузить его можно с официального сайта, перейдя по ссылке https://www.python.org/downloads/

Установка Python в Windows

Для операционной системы Windows дистрибутив распространяется либо в виде исполняемого файл, либо в виде архивного файла.

Порядок установки.

1 Запустите скачанный установочный файл.

2 Выберете способ установки.

Рис. 1.1. Установка Python

В данном окне предлагается два варианта *Install Now* и *Customize installation*. При выборе *Install Now*, Python установится в папку по указанному пути. Помимо самого интерпретатора будет установлен IDLE (интегрированная среда разработки), pip (пакетный менеджер) и документация, а также будут созданы соответствующие ярлыки и установлены связи файлов, имеющие расширение .py с интерпретатором Python. *Customize installation* – это вариант настраиваемой установки. Опция *Add python 3.* to PATH нужна для того, чтобы появилась возможность запускать интерпретатор без указания полного пути до исполняемого файла при работе в командной строке.

1 Отметьте необходимые опций установки (доступно при выборе *Customize installation*).

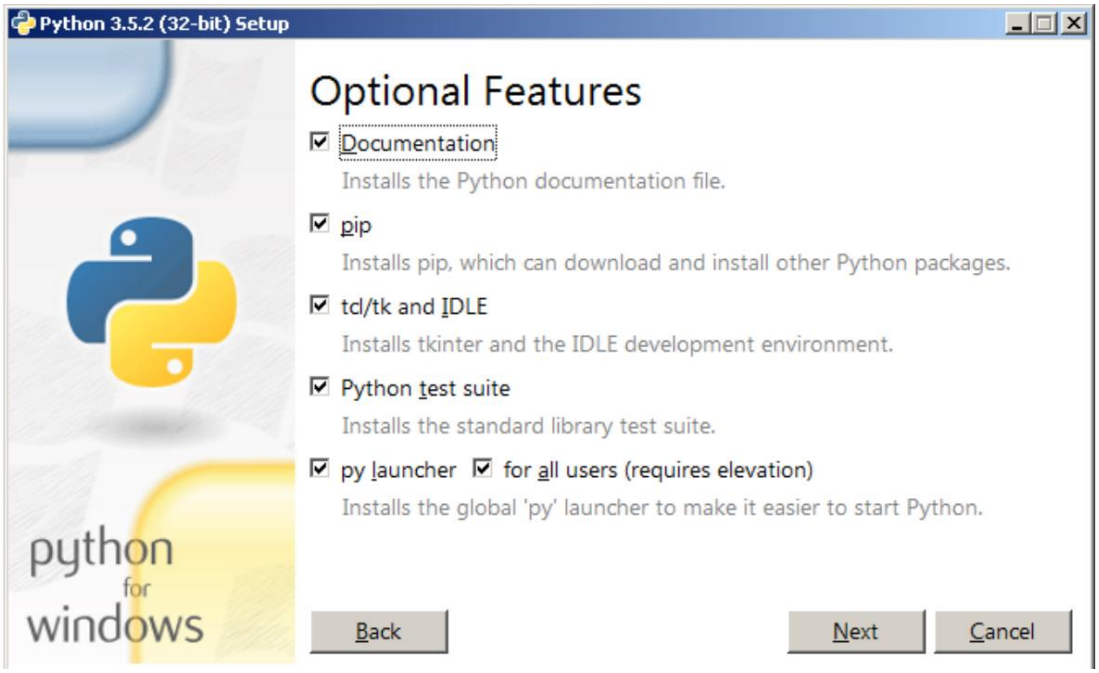


Рис. 1.2. Установка Python

На этом шаге нам предлагается отметить дополнения, устанавливаемые вместе с интерпретатором Python. Рекомендуем выбрать все опции:

- Documentation – установка документаций.
- pip – установка пакетного менеджера pip.
- tcl/tk and IDLE – установка интегрированной среды разработки (IDLE) и библиотеки для построения графического интерфейса (tkinter).

2 Выберете место установки (доступно при выборе Customize installation).

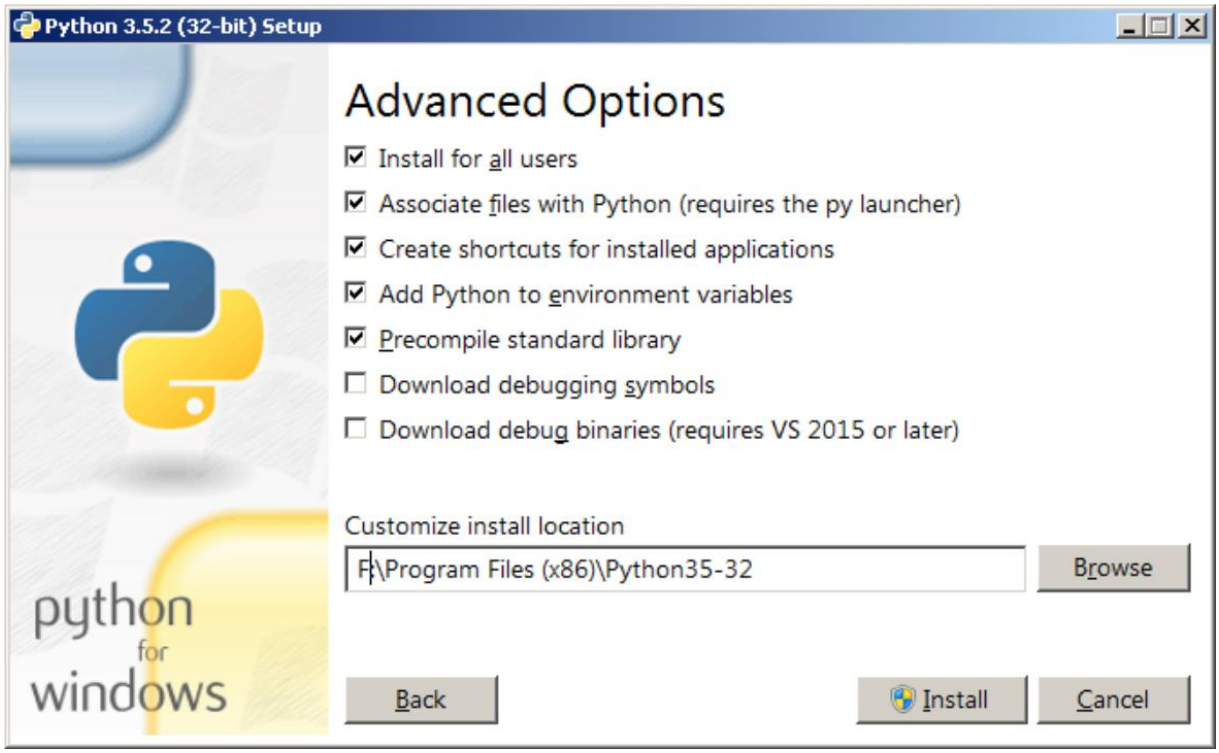


Рис. 1.3. Установка Python

Помимо указания пути, данное окно позволяет внести дополнительные изменения в процесс установки с помощью опций:

- Install for all users – Установить для всех пользователей. Если не выбрать данную опцию, то будет предложен вариант инсталляции в папку пользователя, устанавливающего интерпретатор.
- Associate files with Python – Связать файлы, имеющие расширение .py, с Python. При выборе данной опции будут внесены изменения в Windows, позволяющие запускать Python скрипты по двойному щелчку мыши.
- Create shortcuts for installed applications – Создать ярлыки для запуска приложений.
- Add Python to environment variables – Добавить пути до интерпретатора Python в переменную PATH.
- Precomple standard library – Провести прекомпиляцию стандартной библиотеки.

Последние два пункта (Download debugging symbols, Download debug binaries) связаны с загрузкой компонентов для отладки, их мы устанавливать не будем.

3 После успешной установки вас ждет следующее сообщение.

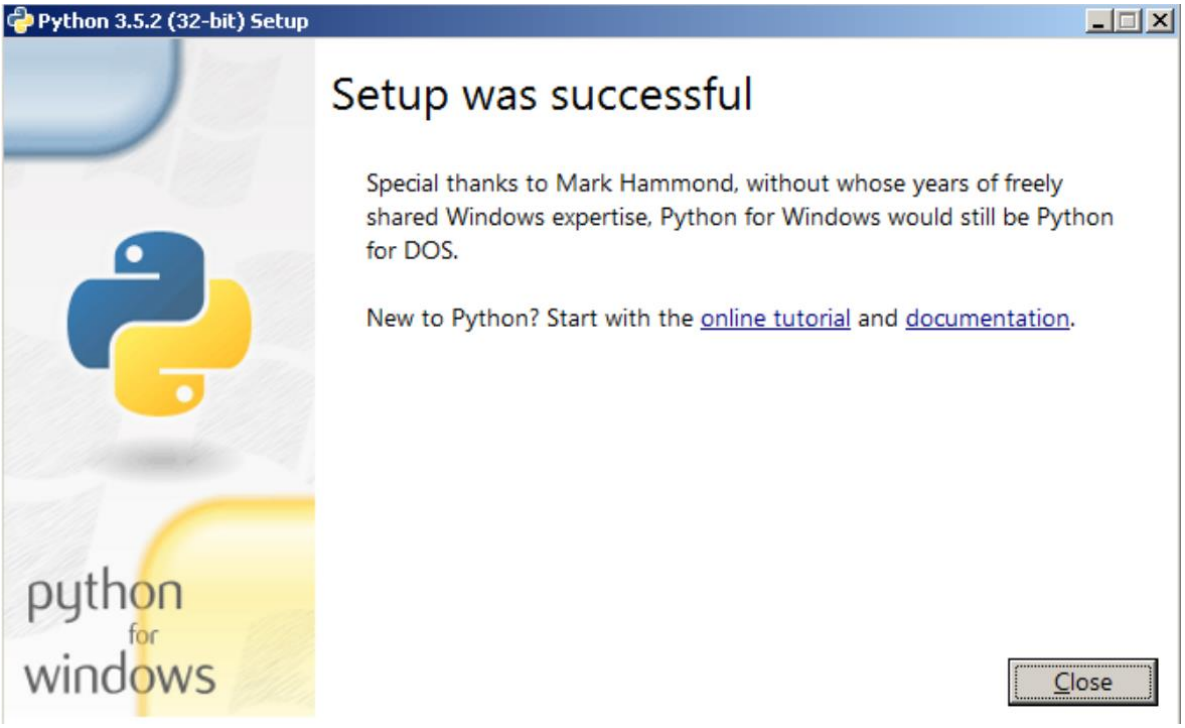


Рис. 1.4. Установка Python

Установка Anaconda

Для удобства запуска примеров и изучения языка Python, советуем установить на свой ПК пакет Anaconda. Этот пакет включает в себя интерпретатор языка Python (есть версии 2 и 3), набор наиболее часто используемых библиотек и удобную среду разработки и исполнения, запускаемую в браузере.

Для установки этого пакета, предварительно нужно скачать дистрибутив https://www.continuum.io/downloads.

Есть варианты под Windows, Linux и Mac OS.

Установка Anaconda в Windows

1 Запустите скачанный инсталлятор. В первом появившемся окне необходимо нажать «Next».

2 Далее следует принять лицензионное соглашение.

Рис. 1.5. Установка Anaconda

3 Выберете одну из опций установки:

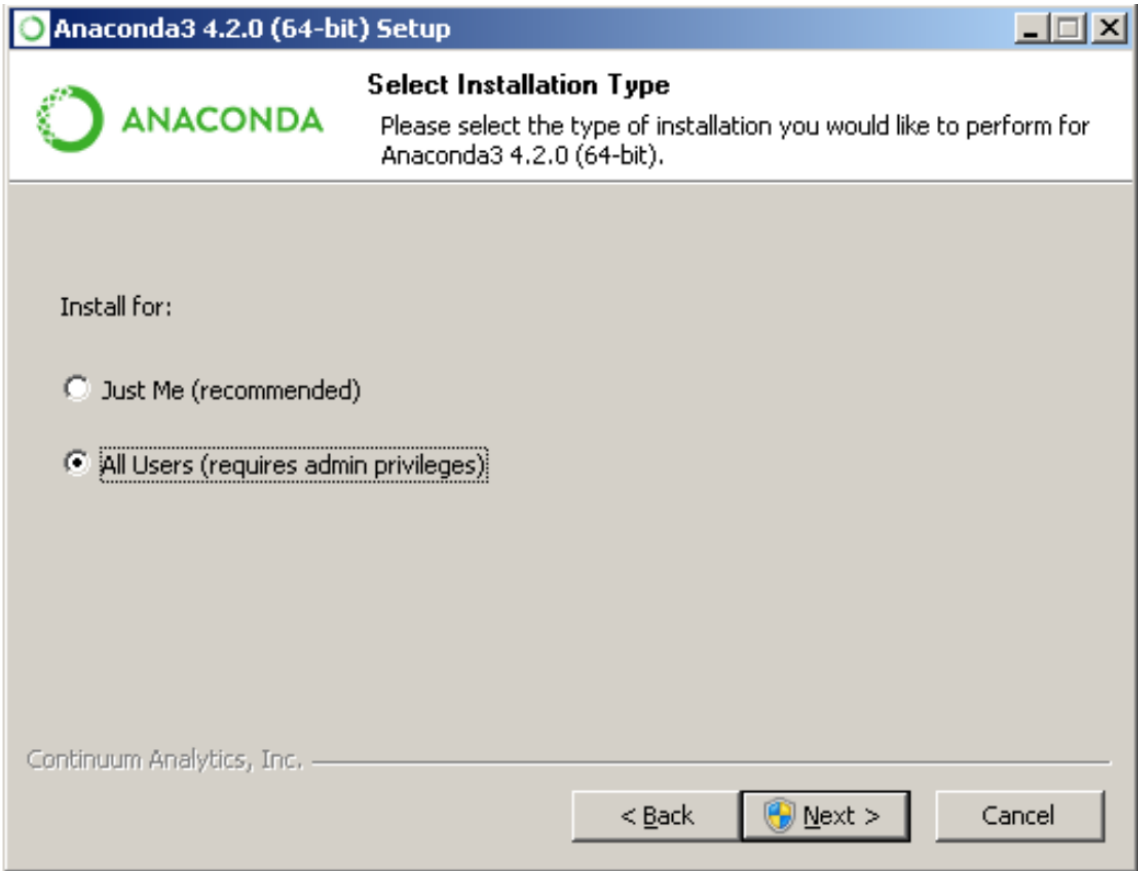


Рис. 1.6. Установка Anaconda

- Just Me – только для пользователя, запустившего установку;
- All Users – для всех пользователей.

4 Укажите путь, по которому будет установлена Anaconda.

5 Укажите дополнительные опции:

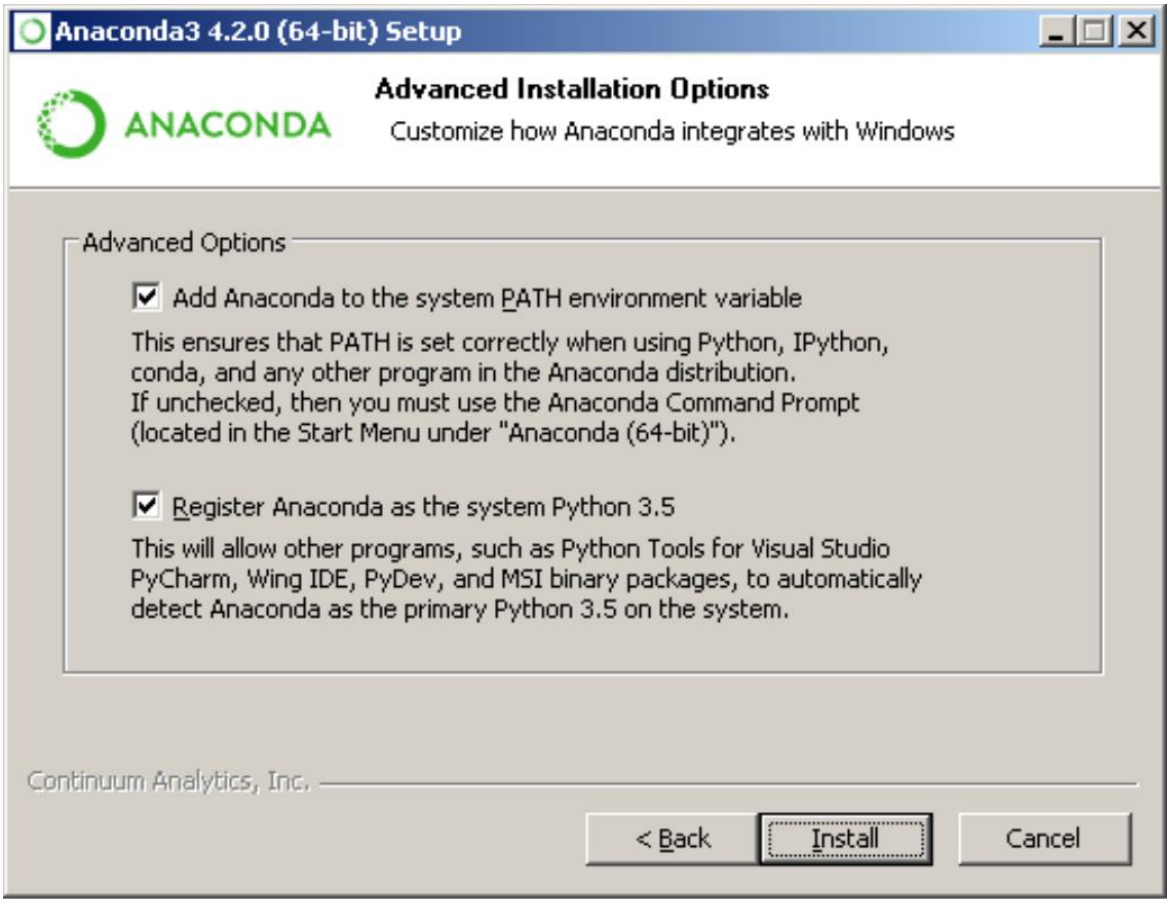


Рис. 1.7. Установка Anaconda

- Add Anaconda to the system PATH environment variable – добавить Anaconda в системную переменную PATH;
- Register Anaconda as the system Python 3 – использовать Anaconda, как интерпретатор Python 3 по умолчанию.

Для начала установки нажмите на кнопку «Install».

5 После этого будет произведена установка Anaconda на ваш компьютер.

Установка библиотек

В зависимости от вашей системы и предыдущих инсталляций среда Python может оказаться неукомплектованной всем тем, что может понадобиться.

Чтобы установить любую нужную библиотеку, можно применить команду pip. Инструмент установки библиотек Python pip непосредственно получает доступ к Интернету и получает их из каталога библиотек Python PyPI. PyPI представляет собой репозиторий, содержащий сторонние библиотеки с открытым исходным кодом, которые постоянно поддерживаются в работоспособном состоянии и сохраняются в репозитории их автором.

Устанавливать библиотеки лучше всего при помощи pip по следующим причинам:

− он является предпочтительным диспетчером библиотек Python и начиная с Python 2.7.9 и Python 3.4 по умолчанию включен в дистрибутивы Python;

− он обеспечивает функциональность по деинсталляции библиотек;

− он возвращает вашу систему в исходное состояние и оставляет ее чистой, если по какой-либо причине установленная библиотека перестала работать.

Команда pip работает в командной строке. Чтобы удостовериться в том, что инструмент pip установлен на локальной машине, выполните следующую команду:

Terminal> pip -V

В некоторых инсталляциях в Linux и Mac OS устанавливается и Python 3 и Python 2, в результате чего могут присутствовать команды pip3 и pip2. Если это так, то pip2 подходит только для установки библиотек в Python 2, а команды pip и pip3 − только для библиотек Python 3.

Если проверка закончилась ошибкой, то вам действительно нужно установить pip с нуля. Для установки pip следуйте инструкциям на https://pip.pypa.io/en/stable/installing/. Самый безопасный путь состоит в том, чтобы скачать сценарий get-pip.py по прямой ссылке с get-pip.py и затем выполнить его при помощи следующей команды:

Terminal> python get-pip.py

После того, как вы удостоверитесь, что инструмент pip установлен, можно будет устанавливать дополнительные библиотеки Python. Чтобы установить типовую библиотеку <lib>, нужно просто выполнить команду:

Terminal> pip install <lib>

После этого библиотека <lib> и все библиотеки, от которых она зависит, будут скачаны и установлены.

Если вы установили дистрибутив Anaconda, то в нем для управления установкой библиотек используется инструмент conda

Способы обновления библиотек

Как правило, может возникнуть ситуация, когда необходимо обновить библиотеку, потому что некая связанная с ней другая библиотека, т.н. зависимость, требует наличия более новой версии, либо имеется дополнительный функционал, который требуется задействовать. Для этого сначала нужно проверить версию установленной библиотеки, обратившись к атрибуту __version__, как показано в примере с библиотекой NumPy ниже:

import numpy

>>> numpy.__version__ # 2 символа подчеркивания перед ним и после него

Далее если нужно ее обновить до более новой версии, скажем в точности до версии 1.9.2, то из командной строки можно выполнить следующую ниже команду:

Terminal> pip install -U numpy==1.9.2

Если вы просто заинтересованы в обновлении до последней доступной версии, то просто выполните команду

Terminal> pip install -U numpy

Установка инструментальной среды Wing IDE

Сам по себе Питон – это только интерпретатор кода. Он запускает ваши программы, но не содержит удобного редактора. Поэтому для написания программ советуем использовать среду разработки Wing IDE.

Wing IDE – это, к сожалению, не свободное ПО, но у него существует официально бесплатная версия для образовательных целей, называется Wing IDE 101. Она доступна как для Windows, так и для Linux и macOS.

Все программы для установки можно скачать с официального сайта Wing IDE (http://wingware.com/, через пункт Download – Wing IDE 101). Обратите внимание, что вам нужна именно версия 101, а не какая-нибудь другая! Установите Wing IDE с помощью этого установщика.

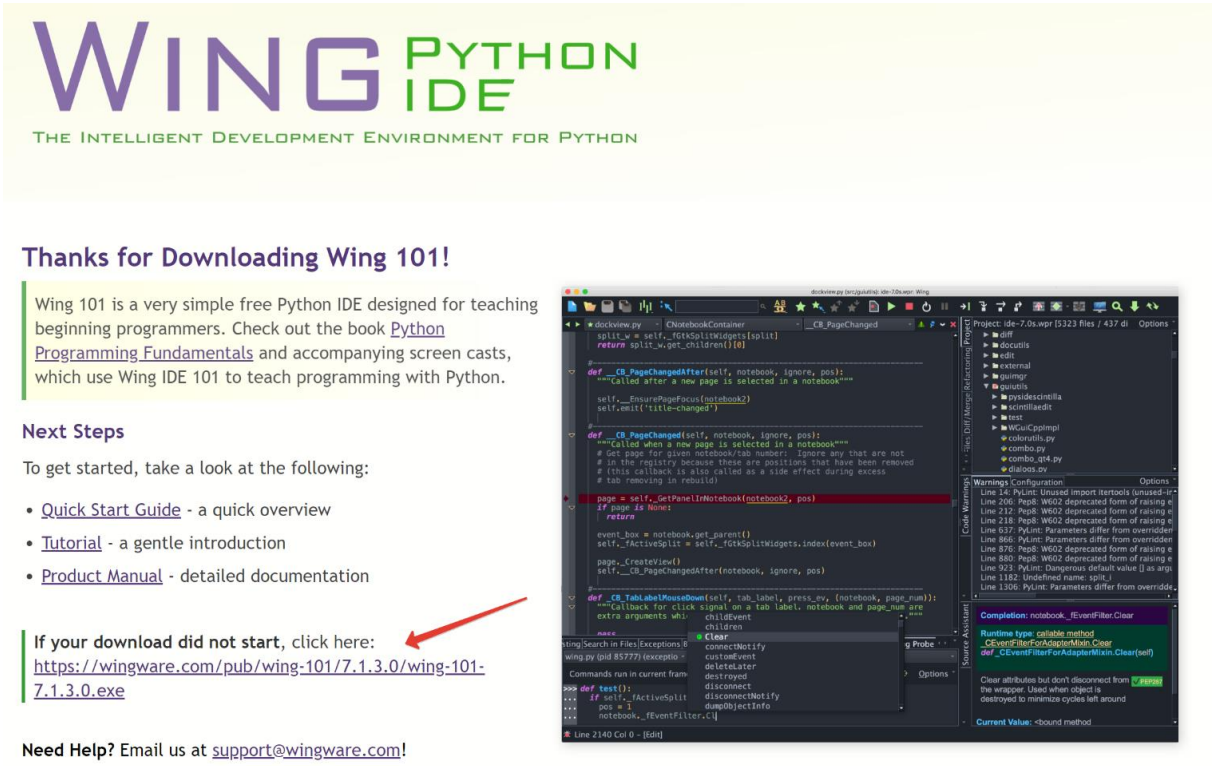


Рис. 1.8. Установка Wing IDE

Wing IDE – это просто среда разработки (IDE) для Python, т.е. удобный редактор программ, позволяющий легко запускать программы с помощью питона (именно поэтому надо отдельно устанавливать сам Python – Wing IDE его не включает в себя). В принципе, вы можете использовать и какую-нибудь другую среду разработки, но тогда разбирайтесь с ней сами. В частности, сам Python включает простенькую среду разработки Python IDLE, ее описание вы можете встретить во многих книжках по Python, но она слишком простая и потому не очень удобная. Так же есть популярная среда PyCharm, но на мой вкус она слишком сложная.

Проверка установки
Запустите Wing IDE. Появится следующее окошко:

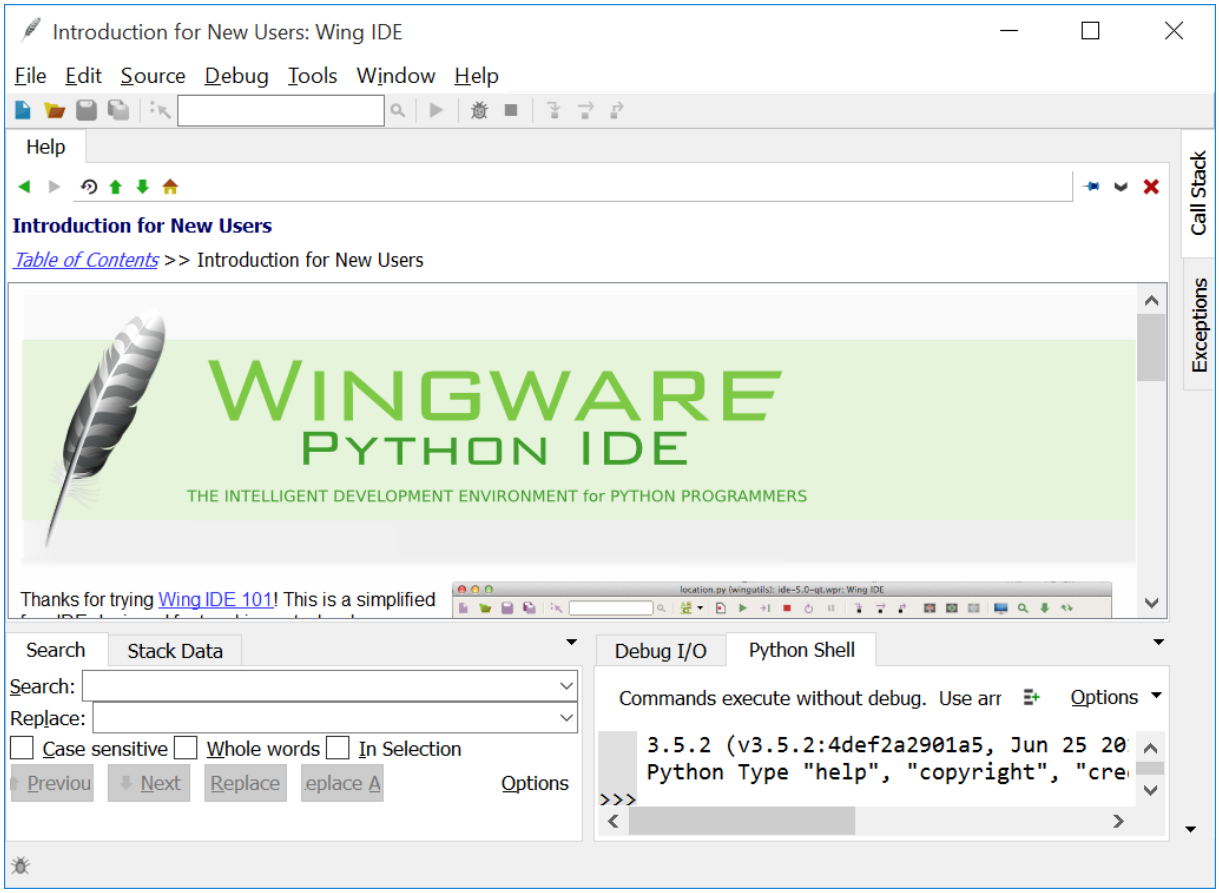


Рис. 1.9. Запуск Wing IDE

Во-первых, убедитесь, что в правом нижнем углу, на панели, озаглавленной Python Shell, появился текст, похожий на приведенный на рисунке; в частности, там должна быть указана версия питона, которую вы устанавливали. Убедитесь, что это версия 3 (на рисунке это версия 3.5.2). Если это не так, то попробуйте через меню Edit – Configure Python указать путь к питону вручную (см. рисунок ниже) – в пункте Python Executable надо указать что-нибудь типа C:\Python3\python.exe, если вы установили питон в каталог C:\Python3, возможно, также в список Python Path надо добавить C:\Python3. Возможно, вам придется поэкспериментировать, чтобы найти правильные настройки. Если у вас на компьютере установлены обе версии питона (и 2, и 3), то, возможно, Wing IDE по умолчанию «подцепит» версию 2, тогда тоже вручную укажите, что вам надо работать с версией 3.

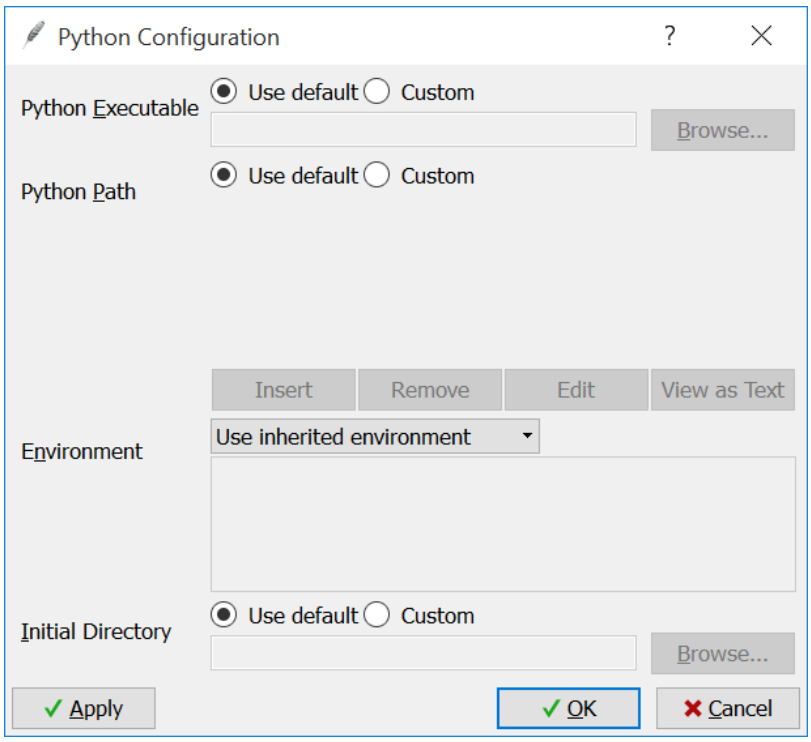


Рис. 1.10. Настройки Wing IDE

1.3 Наука о данных: основные понятия и определения

В отличие от термина «большие данные», который стал популярен с 2010-х гг., наука о данных зародилась намного раньше, во второй половине 20-го века. Первое упоминание этого понятия датируется 1974 годом, когда вышла книга Петера Наура. В этой публикации Data Science определяется как дисциплина по изучению жизненного цикла цифровых данных, от момента их появления до преобразования и использования в других областях знаний. Тем не менее, широкое употребление этот термин получил лишь в 1990-е годы, а общепризнанным стал только в начале 2000-х. В частности, в 2002 году междисциплинарный Комитет по данным для науки и техники начал выпускать журнала CODATA Data Science Journal, а в январе 2003 года вышел первый номер The Journal of Data Science Колумбийского университета [1].

Следующая волна интереса к DS возникла при популяризации понятия Big Data, с 2010 года, когда вычислительные мощности даже бытовых компьютеров стали позволять работать с большими объемами данных. Примерно с этого же времени стали проводиться многочисленные профессиональные конференции, а университеты по всему миру включили эту дисциплину в свои учебные курсы, разработав соответствующие образовательные программы.

Сегодня Data Science активно применяется в широком спектре прикладных областей деятельности: от астрономии до медицины, включая коммерческие кейсы: маркетинг, ритейл, менеджмент, финансовый анализ, предиктивная аналитика чрезвычайных ситуаций и т. д.

ЧТО ЗНАЕТ, УМЕЕТ И СКОЛЬКО СТОИТ DATA SCIENTIST

Специалисты в области Data Science называются учеными или исследователями по данным (Data Scientist'ами). В настоящее время это одна из самых востребованных и высокооплачиваемых ИТ-профессий. Например, в Москве на январь 2020 года месячный труд ученого по данным оценивается около 200 тысяч рублей (от 70 до 250 т.р.). В США оплата выше – $110 – $140 тысяч в год [2].

Основная практическая цель работы ученого по данным – это извлечение полезных для бизнеса сведений из больших массивов информации, выявление закономерностей, разработка и проверка гипотез путем моделирования и разработки нового программного обеспечения [3].

Для достижения этой цели Data Scientist использует следующие инструменты:

- пакеты статистического моделирования (R-Studio, Matlab);
- технологии больших данных (Apache Hadoop, HDFS, Spark, Kafka), NoSQL-СУБД (Cassandra, HBase, MongoDB, DynamoDB и прочие нереляционные решения);
- SQL для работы с классическими реляционными базами данных и формирования структурированных запросов к NoSQL-решениям с помощью Apache Phoenix, Drill, Impala, Hive и пр.
- языки программирования (Python, R, Java, Scala) для разработки моделей машинного обучения и прототипов программного обеспечения;
- информационные системы класса Business Intelligence (дэшборды, витрины данных) для визуализации бизнес-показателей из информационных массивов.

Таким образом, можно сделать вывод, что Data Science включает следующие области знаний:

- математика – математический анализ, матстатистика и матлогика;
- информатика - разработка программного обеспечения, баз данных, моделей и алгоритмов машинного обучения (нейросети, байесовские алгоритмы, регрессионные ряды и пр.), Data Mining;
- системный анализ - методы анализа предметной области, Business Intelligence.

2 Задания для выполнения

Выполнить следующее:

1 Установить Python.
2 Установить инструментальную среду программирования Wing.
3 Установить пакет Anaconda.
4 Установить библиотеку numpy.

Лабораторная работа № 2. Введение в язык Python. Ввод и вывод данных. Структуры данных

1 Методические рекомендации

1.1 Создание программ на Python в Wing

Простейшая программа

В основном меню Wing IDE выберите пункт File – New. Появится окно для редактирования текста программы. В этом окне наберите следующий текст:

```
print("Test", 2*2)
```

(Здесь " – это символ кавычек.)

Должно получиться так:

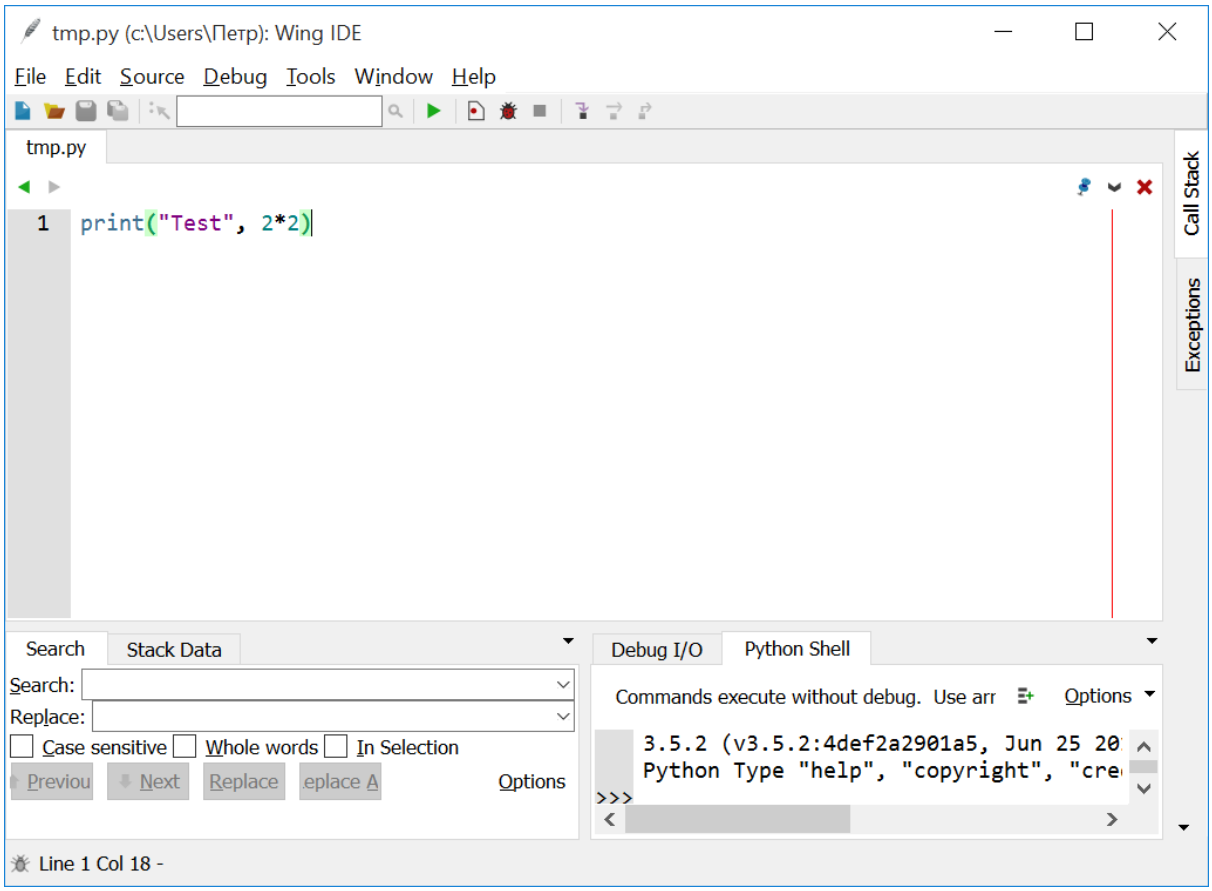


Рис. 2.1. Окно Wing IDE

Убедитесь, что опечаток нет. Сохраните программу: нажмите Ctrl-S или выберите пункт меню File – Save As. Wing IDE предложит выбрать имя файла для сохранения, для первой программы можно выбрать любое имя.

Примечание

Обратите внимание, что Wing IDE раскрашивает вашу программу. Это делается для того, чтобы ее было удобнее читать; на самом деле для питона

цвет не важен, он сделан только для того, чтобы вам было удобнее читать. Аналогично, в этом тексте код тоже раскрашен, причем раскраска может быть немного другой (это просто обусловлено системой, которую я использую для написания текста). Но еще раз: цвета только для удобства чтения, никакой больше нагрузки они не несут, в частности, Wing может раскрашивать не так, как вы видите в этом тексте, это не страшно.

После этого запустите программу, нажав на кнопку с зеленым треугольничком–стрелочкой на панели инструментов над текстом программы. Результат выполнения программы появляется в правой нижней части экрана, в панели «Python Shell» А именно, там вы можете увидеть один из двух возможных результатов, показанных на двух рисунках ниже.

Если там появилась надпись «Test 4», значит, все хорошо, программа успешно выполнилась.

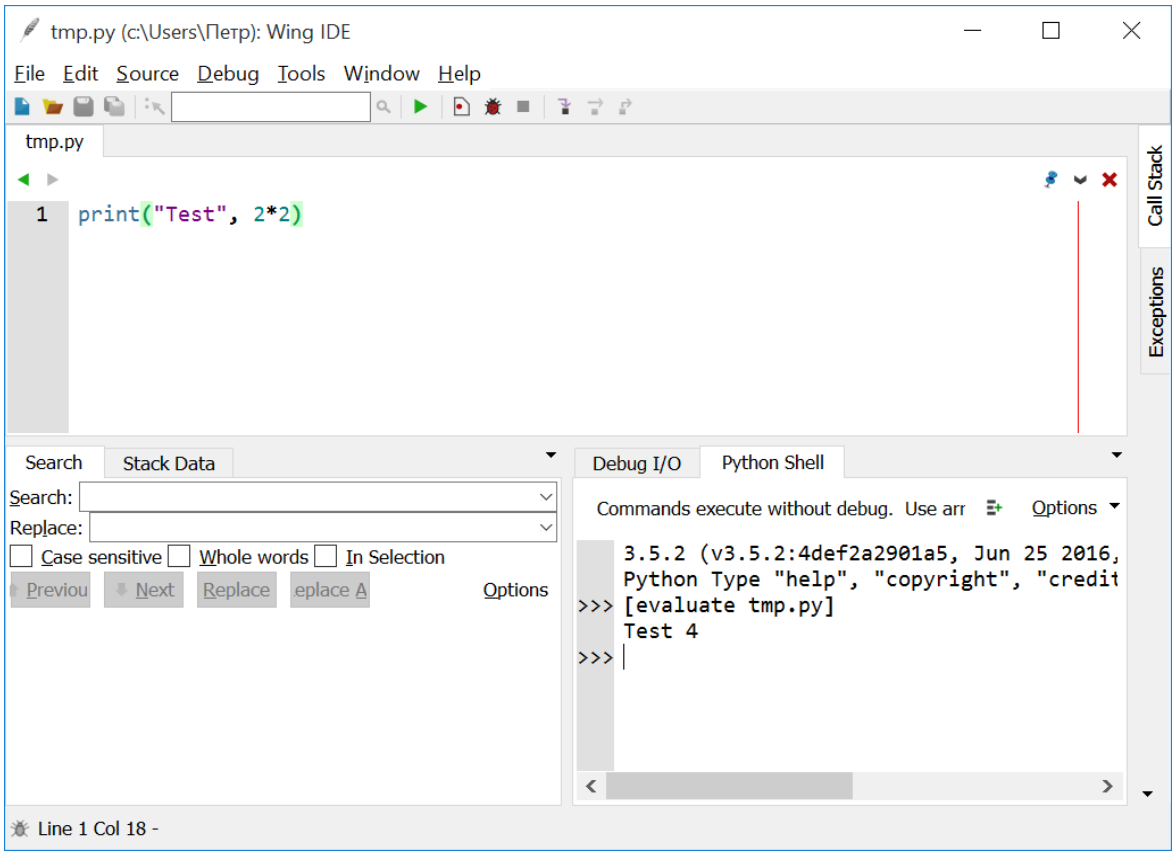


Рис. 2.2. Оператор вывода

Если же там появился длинный текст со словами «Traceback» (в начале) и «Error» (в конце), значит, в вашей программе есть ошибки.

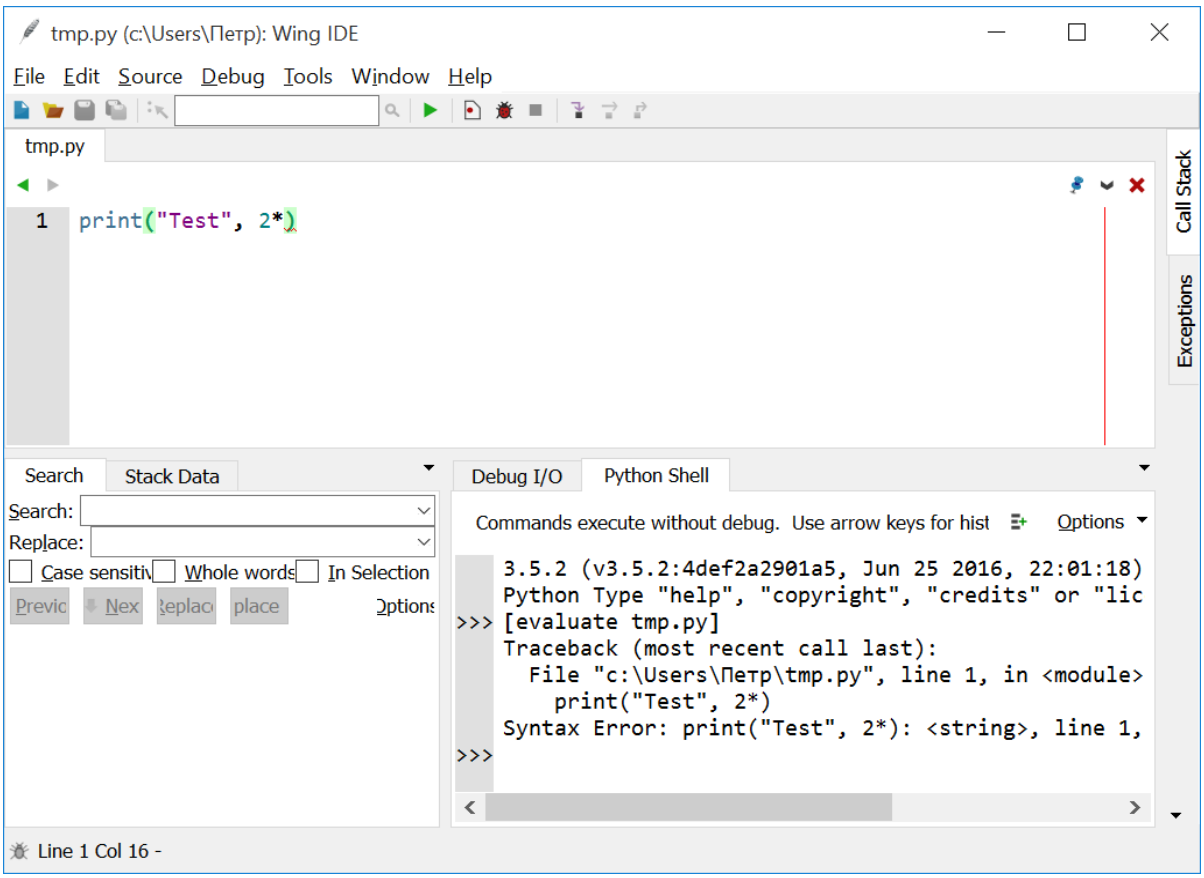


Рис. 2.3. Ошибка в Wing IDE

Подробнее про ошибки ниже (раздел Ошибки в программе), а пока, если вы увидели ошибку, то просто внимательно проверьте, не ошиблись ли вы где-нибудь в наборе программы.

Добейтесь того, чтобы ваша программа отработала успешно (внимательно проверив, не допустили ли вы ошибок), и посмотрите, что же именно пишется в этом окошке «Python Shell». Там, во-первых, виден заголовок питона (включающий номер версии), дальше строка >>> [evaluate tmp.py] (вместо tmp.py здесь будет имя файла, куда вы сохранили программу). Эта строка была выведена в тот момент, когда Wing IDE начал запускать вашу программу. И, наконец, есть строка Test 4, которую и напечатала наша программа. Почему она напечатала именно это, обсудим чуть ниже.

Позапускайте программу (зеленой стрелочкой) ещё несколько раз и посмотрите на результаты. Вы увидите, что Wing IDE каждый раз печатает строку evaluate... перед запуском программы, потом программа печатает свою строку. Вывод программы перемешивается с выводом Wing IDE – ничего страшного, это нормально.

Можно также запускать программу нажатием на кнопку с картинкой типа красного жучка. Это немного другой режим запуска, более удобный для поиска ошибок. Попробуйте позапускать и так, и так, посмотрите на отличия (основное отличие пока – при запуске через «красного жучка» вывод предыдущих программ затирается).

Ошибки в программе

В вашей программе могут быть серьёзные ошибки – такие, что питон «не понимает», что вы от него хотите (а могут быть и не столь серьёзные – программа отработает как бы нормально, но выдаст неверный результат). В случае таких серьезных ошибок питон выдаст сообщение, похожее на сообщение, показанное на рисунке выше. Оно обычно начинается со слова Traceback, а ближе к концу в нем есть слово Error.

С ошибками удобнее разбираться, запуская программу в режиме «красного жучка». В таком случае Wing IDE подсвечивает строку около ошибки красным, а подробную информацию пишет в особом окошке справа.

Пока для вас важным будет то, какую строку Wing IDE подсветила красным – примерно в том месте и ошибка. Важен также текст («сообщение об ошибке»), обычно содержащий слово «Error» (в примере на рисунке Syntax Error ...), там же рядом указан и номер строки с ошибкой (line 1). Поначалу сообщения об ошибке сложно понимать, но со временем вы выучите наиболее часто встречающиеся и будете сразу понимать, что не так.

А пока посмотрите внимательно на строчку с ошибкой (при запуске через жучка питон подсвечивает ее красным, при запуске через стрелочку – только пишет номер строки), и на строчки рядом – и попробуйте понять, что там не так. В примере на рисунке я забыл вторую цифру 2 (в результате чего питону стало непонятно, на что надо умножать). (В примере на рисунке я запускал программу через зеленую стрелочку, а не через «красного жучка», поэтому там нет подсвеченной красным строки.)

Имейте в виду, что питон не телепат и не может точно определить, где вы допустили ошибку. Он подсвечивает красным ту строку, где текст программы впервые разошёлся с правилами языка. Поэтому бывает, что на самом деле ваша ошибка чуть выше, чем подсвеченная строка (а иногда – и намного выше). Но тем не менее место, которое выделил питон, обычно бывает полезно при поиске ошибки.

Попробуйте в своей программе поделать разные ошибки и посмотрите, как на них отреагирует питон.

Как работает эта программа

Давайте разберём, как эта программа работает. Напомню её текст:

```
print("Test", 2*2)
```

Вообще, любая программа – это, в первую очередь, последовательность команд, которые программист даёт компьютеру, а компьютер будет последовательно их выполнять.

В нашей программе одна команда – print("Test", 2*2). Команда print обозначает «вывести на экран» (английское слово «print» обозначает «печатать»). В скобках после слова print указываются, как говорят, аргументы команды. Они разделяются запятыми, в данном случае у команды два аргумента: первый – "Test", и второй – 2*2.

Если аргументом команды print является некоторая строка, заключённая в кавычки (символы "), то команда print выводит эту строку на экран как есть (без кавычек). Поэтому первым делом наша команда выводит на экран текст Test.

Вторым аргументом команды print в нашем примере является арифметическое выражение 2*2. Если аргументом команды (любой команды, не обязательно именно print, просто других мы пока не знаем) является арифметические выражение, то компьютер сначала вычислит его, а потом передаст команде. Поэтому в данном случае сначала компьютер вычислит, получит 4, а потом передаст результат команде print, которая выведет его на экран.

Команда print разделяет выводимые элементы пробелами, поэтому между Test и 4 выведен пробел.

В итоге получается, что наша программа выводит Test 4.

1.2 Ввод и вывод данных

Мы уже встречались с функцией print(). Она отвечает за вывод данных, по-умолчанию на экран. Если код содержится в файле, то без нее не обойтись. В интерактивном режиме в ряде случаев можно обойтись без нее.

Ввод данных в программу и их вывод важны в программировании. Без ввода программы делали бы одно и то же, исключая случаи, когда в них самих генерируются случайные значения. Вывод позволяет увидеть, использовать, передать дальше результат работы программы.

Обычно требуется, чтобы программа обрабатывала какой-то диапазон различных входных данных, которые поступают в нее из внешних источников. В качестве последних могут выступать файлы, клавиатура, сеть, выходные данные из другой программы. В свою очередь вывод данных, например, возможен в файл, по сети, в базу данных, на принтер. Однако нередко информацию просто выводят на экран монитора.

Можно сказать, что программа – это открытая система, которая обменивается чем-либо с внешней для нее средой. Если живой организм в основном обменивается веществом и энергией, то программа – данными, информацией.

Вывод данных. Функция print()

Что такое функция в программировании, узнаем позже. Пока будем считать, что print() – это такая команда языка Python, которая выводит то, что в ее скобках на экран.

```
>>> print(1032)
1032
>>> print(2.34)
2.34
>>> print("Hello")
Hello
```

В скобках могут быть любые типы данных. Кроме того, количество данных может быть различным:

```
>>> print("a:", 1)
a: 1
>>> one = 1
>>> two = 2
>>> three = 3
>>> print(one, two, three)
1 2 3
```

Можно передавать в функцию print() как непосредственно литералы (в данном случае "a:" и 1), так и переменные, вместо которых будут выведены их значения. Аргументы функции (то, что в скобках), разделяются между собой запятыми. В выводе вместо запятых значения разделены пробелом.

Если в скобках стоит выражение, то сначала оно выполняется, после чего print() уже выводит результат данного выражения:

```
>>> print("hello" + " " + "world")
hello world
>>> print(10 - 2.5/2)
8.75
```

В print() предусмотрены дополнительные параметры. Например, через параметр sep можно указать отличный от пробела разделитель строк:

```
>>> print("Mon", "Tue", "Wed", "Thu",
... "Fri", "Sat", "Sun", sep="-")
Mon-Tue-Wed-Thu-Fri-Sat-Sun
>>> print(1, 2, 3, sep="//")
1//2//3
```

Параметр end позволяет указывать, что делать, после вывода строки. По-умолчанию происходит переход на новую строку. Однако это действие можно отменить, указав любой другой символ или строку:

```
>>> print(10, end="")
10>>>
```

Обычно end используется не в интерактивном режиме, а в скриптах, когда несколько выводов подряд надо разделить не переходом на новую строку, а, скажем, запятыми. Сам переход на новую строку обозначается символом '\n'. Если присвоить это значение параметру end, то никаких изменений в работе функции print вы не увидите, так как это значение и так присвоено по-умолчанию:

```
>>> print(10, end='\n')
10
>>>
```

Однако, если надо отступить на одну дополнительную строку после вывода, то можно сделать так:

```
>>> print(10, end='\n\n')
10


>>>
```

В функцию print нередко передаются так называемые форматированные строки, хотя по смыслу их правильнее называть строки-шаблоны. Никакого отношения к самому print они не имеют. Когда такая строка находится в скобках print(), интерпретатор сначала согласно заданному в ней формату преобразует ее к обычной строке, после чего передает результат в print().

Форматирование может выполняться в так называемом старом стиле или с помощью строкового метода format. Старый стиль также называют Си-стилем, так как он схож с тем, как происходит вывод на экран в языке C. Рассмотрим пример:

```
>>> pupil = "Ben"
>>> old = 16
>>> grade = 9.2
>>> print("It's %s, %d. Level: %f" %
... (pupil, old, grade))
It's Ben, 16. Level: 9.200000
```

Здесь вместо трех комбинаций символов %s, %d, %f подставляются значения переменных *pupil*, *old*, *grade*. Буквы s, d, f обозначают типы данных – строку, целое число, вещественное число. Если бы требовалось подставить три строки, то во всех случаях использовалось бы сочетание %s.

Хотя в качестве значения переменной *grade* было указано число 9.2, на экран оно вывелось с дополнительными нулями. Чтобы указать, сколько требуется знаков после запятой, надо перед f поставить точку, после нее указать желаемое количество знаков в дробной части:

```
>>> print("It's %s, %d. Level: %.1f"
...  % (pupil, old, grade))
It's Ben, 16. Level: 9.2
```

Теперь посмотрим на метод format():

```
>>> print("This is a {0}. It's {1}."
... .format("ball", "red"))
This is a ball. It's red.
>>> print("This is a {0}. It's {1}."
... .format("cat", "white"))
This is a cat. It's white.
>>> print("This is a {0}. It's {1} {2}."
... .format(1, "a", "number"))
This is a 1. It's a number.
```

В строке в фигурных скобках указаны номера данных, которые будут сюда подставлены. Далее к строке применяется метод format(). В его скоб-

ках указываются сами данные (можно использовать переменные). На нулевое место подставится первый аргумент метода format(), на место с номером 1 – второй и т. д.

На самом деле возможности метода format() существенно шире, и для их изучения понадобился бы отдельный урок. Нам пока будет достаточно этого.

В новых релизах Питона появился третий способ создания форматированных строк – f-строки. Перед их открывающей кавычкой прописывается буква f. В самой строке внутри фигурных скобок записываются выражения на Python, которые исполняются, когда интерпретатор преобразует строку-шаблон в обычную.

```
>>> a = 10
>>> b = 1.33
>>> c = 'Box'
>>> print(f'qty - {a:5}, goods - {c}')
qty -    10, goods - Box
>>> print(f'price - {b + 0.2:.1f}')
price - 1.5
```

В примере число 5 после переменной *a* обозначает количество знакомест, отводимых под вывод значения переменной. В выражении b + 0.2:.1f сначала выполняется сложение, после этого значение округляется до одного знака после запятой.

Ввод данных. Функция input()

За ввод в программу данных с клавиатуры в Python отвечает функция **input**. Когда вызывается эта функция, программа останавливает свое выполнение и ждет, когда пользователь введет текст. После этого, когда он нажмет Enter, функция input() заберет введенный текст и передаст его программе, которая уже будет обрабатывать его согласно своим алгоритмам.

Если в интерактивном режиме ввести команду input(), то ничего интересного вы не увидите. Компьютер будет ждать, когда вы что-нибудь введете и нажмете Enter или просто нажмете Enter. Если вы что-то ввели, это сразу же отобразиться на экране:

```
>>> input()
Yes!
'Yes!'
```

Функция input() передает введенные данные в программу. Их можно присвоить переменной. В этом случае интерпретатор не выводит строку сразу же:

```
>>> answer = input()
No, it is not.
```

В данном случае строка сохраняется в переменной *answer*, и при желании мы можем вывести ее значение на экран:

```
>>> answer
'No, it is not.'
```

При использовании функции print() кавычки в выводе опускаются:

>>> print(answer)

No, it is not.

Куда интересней использовать функцию input() в скриптах – файлах с кодом. Рассмотрим такую программу:

```
name_user = input()
city_user = input()
print(f'Вас зовут {name_user}. Ваш город {city_user}')


Серый
Белый
Вас зовут Серый. Ваш город Белый
```

Рис. 2.4. Пример программы

При запуске программы, компьютер ждет, когда будет введена сначала одна строка, потом вторая. Они будут присвоены переменным *name_user* и *city_user*. После этого значения этих переменных выводятся на экран с помощью форматированного вывода.

Вышеприведенный скрипт далек от совершенства. Откуда пользователю знать, что хочет от него программа? Чтобы не вводить человека в замешательство, для функции input предусмотрен специальный параметр-приглашение. Это приглашение выводится на экран при вызове input(). Усовершенствованная программа может выглядеть так:

```
name_user = input('Ваше имя: ')
city_user = input('Ваш город: ')
print(f'Вас зовут {name_user}. Ваш город {city_user}')


Ваше имя: Макс
Ваш город: Москва
Вас зовут Макс. Ваш город Москва
```

Рис. 2.5. Пример программы

Обратите внимание, что в программу поступает строка. Даже если ввести число, функция input() все равно вернет его строковое представление. Но что делать, если надо получить число? Ответ: использовать функции преобразования типов.

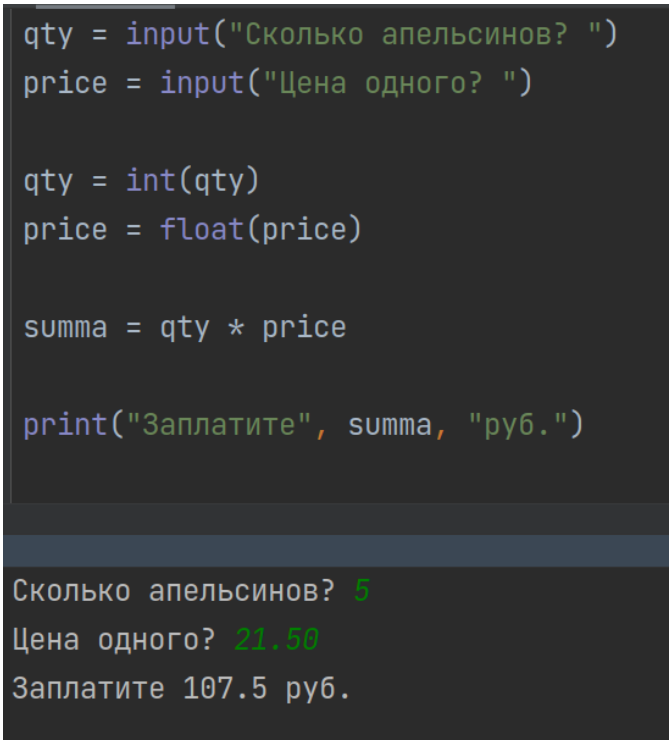


Рис. 2.6. Пример программы

В данном случае с помощью функций int() и float() строковые значения переменных *qty* и *price* преобразуются соответственно в целое число и вещественное число. После этого новые численные значения присваиваются тем же переменным.

Программный код можно сократить, если преобразование типов выполнить в тех же строках кода, где вызывается функция input():

```
qty = int(input("Сколько апельсинов? "))
price = float(input("Цена одного? "))

summa = qty * price

print("Заплатите", summa, "руб.")
```

Сначала выполняется функция input(). Она возвращает строку, которую функция int() или float() сразу преобразует в число. Только после этого происходит присваивание переменной, то есть она сразу получает численное значение.

1.3 Структуры данных

1 List(), dict() и float() используют круглые скобки, потому что они являются функциями;

2 Скобки сами по себе представляют кортеж, и их не стоит путать со скобками в функциях, например, list();

3 При создании пустого списка нужно использовать квадратные, а не круглые скобки: [].

Функция list()

У функции list() очень простой сценарий применения.

С помощью скобок создается список. После этого выводится переменная с присвоенным ей пустым списком. Выводится «[]», что указывает пусть и на пустой, но список. После этого выводится подтверждение того, что это действительно список.

```
my_first_list = []
print(my_first_list )
print(type(my_first_list ))
[]
<class 'list'>
```

В следующем примере можно добиться того же результата, что и в первом, но уже с помощью функции list().

```
my_first_list = list()
print(my_first_list)
print(type(my_first_list))
[]
<class 'list'>
```

Функция dict()

Рассмотрим функцию dict(), с помощью которой можно создать словарь Python.

```
my_first_dictionary = {}
print(my_first_dictionary)
print(type(my_first_dictionary))
{}
<class 'dict'>
```

И вот еще один пример создания пустого словаря. Тот же результат, но с помощью функции dict().

```
my_first_dictionary = dict()
print(my_first_dictionary)
print(type(my_first_dictionary))
{}
<class 'dict'>
```

Функция tuple()

Функция tuple() для создания **кортежа Python**. Пример с присваиванием переменной пустого кортежа.

```
my_first_tuple = ()
print(my_first_tuple)
print(type(my_first_tuple))
()
<class 'tuple'>
```

И снова тот же результат, но уже с помощью функции tuple().

```
my_first_tuple = tuple()
print(my_first_tuple)
print(type(my_first_tuple))
()
<class 'tuple'>
```

Советы:

1 Кортеж − отличная альтернатива списку для тех ситуаций, когда в дальнейшем не требуется менять значения внутри этой структуры данных.

2 Словари − идеальная структура для хранения пар ключ-значение.

Создадим список со значениями внутри.

```
mylist = ["лыжные ботинки", "лыжи", "перчатки"]
print(mylist)
print(type(mylist))
["лыжные ботинки", "лыжи", "перчатки"]
<class 'list'>
```

В этом примере список состоит из 3 строковых значений. Но все три типа − списки, словари и кортежи − могут включать разные типы данных.

Теперь создадим словарь с этими значениями.

```
mydict = {"лыжные ботинки": 3, "лыжи": 2, "перчатки": 5}
print(mydict)
print(type(mydict))
{"лыжные ботинки": 3, "лыжи": 2, "перчатки": 5}
<class 'dict'>
```

В этом примере создается словарь. Он присвоен переменной mydict, которая состоит из 3 ключей: “лыжные ботинки”, “лыжи” и “перчатки”. Каждому ключу присвоено свое значение: 3, 2 и 5 соответственно.

Теперь создадим список со значениями разных типов.

```
mylist = ["карабины", False, "порошок", 666, 25.25]
print(mylist)
print(type(mylist))
["карабины", False, "порошок", 666, 25.25]
<class 'list'>
```

В этом примере список включает 5 значений четырех разных типов: строка, булев тип, строка, целое число и число с плавающей точкой.

Наконец, создадим пример кортежа.

```
mytuple = ("карабины", False, "порошок", 666, 25.25)
print(mytuple)
print(type(mytuple))
("карабины", False, "порошок", 666, 25.25)
<class 'tuple'>
```

Единственное отличие здесь в том, что используются круглые (), а не квадратные [] скобки.

Мы кратко рассмотрели структуры данных, в следующих уроках разберем каждый подробно.

2 Задания для выполнения

Порядок выполнения работы

1 Перевести число "A", заданное в системе счисления "B", в систему счисления "C". Числа "A", "B" и "C" взять из представленных ниже таблиц. Вариант выбирается как сумма последнего числа в номере группы и номера в списке группы согласно ISU. Т. е. 13-му человеку из группы P3102 соответствует 15-й вариант (=2 + 13).

2 Всего нужно решить 11 примеров. Для примеров с 5-го по 7-й выполнить операцию перевода по сокращенному правилу (для системс основанием 2 в системы с основанием 2^k). Для примеров с 4-го по 6-й и с 8-го по 9-й найти ответ с точностью до 5 знака после запятой. В примере 11 группа символов {^1} означает -1 в симметричной системе счисления.

Требования и состав отчёта

1 Отчёт должен быть выполнен на листе размером А4.

2 Отчёт должен начинаться с титульного листа с названием вуза ифакультета, номером и названием лабораторной работы, вариантом, ФИО студента, № группы, ФИО преподавателя, городом и годом.

3 В отчёте нужно кратко описать задание, показать основные этапы вычисления при выполнении всех операций, сформулировать выводы.

4 Отчёт предоставить в бумажном или электронном виде.

Варианты заданий

	1			2			3			4			5		
#	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
1	39275	10	7	55856	13	10	46320	7	13	35,34	10	2	2A,A3	16	2
2	40311	10	11	46200	7	10	370D1	15	5	93,64	10	2	FA,BC	16	2
3	20946	10	5	A4702	11	10	89358	13	7	67,95	10	2	B9,46	16	2
4	62740	10	5	56666	9	10	89618	11	9	46,96	10	2	32,22	16	2
5	49152	10	13	17566	9	10	799BC	15	5	99,27	10	2	E1,DB	16	2
6	29351	10	15	47658	11	10	C9120	15	5	56,37	10	2	33,25	16	2
7	35292	10	5	17A0A	11	10	13242	7	13	33,45	10	2	14,69	16	2
8	52261	10	7	14511	9	10	17008	9	11	30,91	10	2	48,4C	16	2
9	59047	10	15	33240	7	10	21300	9	11	94,85	10	2	CD,BC	16	2
10	17109	10	13	55404	9	10	25860	9	11	35,22	10	2	5F,26	16	2

	1			2			3			4			5		
#	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
11	36734	10	13	20046	7	10	30242	5	15	87,71	10	2	29,5B	16	2
12	57970	10	5	23143	5	10	11204	5	15	46,64	10	2	C2,59	16	2
13	38985	10	7	CAD9B	15	10	628ED	15	5	36,63	10	2	58,3C	16	2
14	76779	10	13	53255	7	10	53441	7	13	69,47	10	2	8A,63	16	2
15	69244	10	9	66875	9	10	12250	7	13	63,99	10	2	6B,51	16	2
16	35146	10	7	13608	11	10	12024	5	15	89,11	10	2	8C,9D	16	2
17	25334	10	9	22211	5	10	3CAAD	15	5	53,54	10	2	72,98	16	2
18	28593	10	5	868A3	13	10	495D7	15	5	48,77	10	2	28,A2	16	2
19	70013	10	9	A414C	15	10	41343	5	15	39,44	10	2	EC,42	16	2
20	68981	10	7	40403	5	10	B9235	15	5	58,88	10	2	BA,12	16	2

	1			2			3			4			5		
#	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
21	34106	10	15	16116	7	10	21104	5	15	51,96	10	2	41,6C	16	2
22	94118	10	15	9A977	13	10	95183	11	9	65,94	10	2	DE,86	16	2
23	31961	10	13	60678	9	10	74B55	13	7	96,87	10	2	FB,B1	16	2
24	74496	10	7	20021	5	10	27072	9	11	43,68	10	2	59,DF	16	2
25	46318	10	15	25115	7	10	29A13	11	9	26,48	10	2	5A,EF	16	2
26	85407	10	11	1A550	11	10	43455	7	13	36,19	10	2	83,E1	16	2
27	25307	10	9	10053	7	10	28D10	15	5	52,16	10	2	3B,64	16	2
28	25285	10	15	C2A41	15	10	40674	9	11	10,25	10	2	7D,F5	16	2
29	50822	10	9	85667	9	10	10101	5	15	68,82	10	2	25,23	16	2
30	95518	10	11	89373	11	10	2E6ED	15	5	68,41	10	2	B5,12	16	2

	1			2			3			4			5		
#	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
31	92934	10	11	A0661	13	10	71574	11	9	56,26	10	2	9B,AA	16	2
32	64073	10	7	31234	5	10	B0524	13	7	95,73	10	2	EA,D9	16	2
33	27162	10	7	84054	11	10	4435A	15	5	27,58	10	2	6A,36	16	2
34	88222	10	15	46632	7	10	66062	9	11	24,63	10	2	BA,B9	16	2
35	35069	10	5	36934	11	10	83488	9	11	94,76	10	2	47,48	16	2
36	83932	10	15	87238	13	10	4945C	13	7	46,33	10	2	68,76	16	2
37	21909	10	9	57A0A	11	10	BECD6	15	5	64,81	10	2	C7,A8	16	2
38	46302	10	11	6CD08	15	10	B3BC9	13	7	93,88	10	2	3E,9D	16	2
39	61196	10	9	20601	7	10	41230	5	15	12,83	10	2	3C,6F	16	2
40	18491	10	7	66305	11	10	B2E7D	15	5	40,56	10	2	F9,A2	16	2

	6			7			8			9		
#	A	B	C	A	B	C	A	B	C	A	B	C
1	34,17	8	2	0,011111	2	16	0,010011	2	10	BF,FA	16	10
2	22,17	8	2	0,000101	2	16	0,001101	2	10	47,C4	16	10
3	27,71	8	2	0,000011	2	16	0,010101	2	10	C3,71	16	10
4	15,26	8	2	0,001001	2	16	0,101111	2	10	D8,A5	16	10
5	72,32	8	2	0,000111	2	16	0,010101	2	10	BB,78	16	10
6	50,56	8	2	0,000101	2	16	0,110111	2	10	71,F1	16	10
7	23,77	8	2	0,110011	2	16	0,010001	2	10	79,87	16	10
8	24,22	8	2	0,011101	2	16	0,011111	2	10	25,4D	16	10
9	76,22	8	2	0,111111	2	16	0,100111	2	10	E3,AF	16	10
10	36,36	8	2	0,010001	2	16	0,010001	2	10	CF,A2	16	10

	6			7			8			9		
#	A	B	C	A	B	C	A	B	C	A	B	C
11	37,76	8	2	0,100101	2	16	0,001111	2	10	C9,CB	16	10
12	15,33	8	2	0,010001	2	16	0,000111	2	10	B4,CE	16	10
13	66,36	8	2	0,110111	2	16	0,001001	2	10	A6,CF	16	10
14	36,37	8	2	0,110111	2	16	0,111011	2	10	14,12	16	10
15	63,51	8	2	0,000101	2	16	0,010111	2	10	6E,D5	16	10
16	23,74	8	2	0,000101	2	16	0,100001	2	10	8C,E9	16	10
17	25,11	8	2	0,011111	2	16	0,000001	2	10	7A,87	16	10
18	31,42	8	2	0,110101	2	16	0,011001	2	10	69,18	16	10
19	36,43	8	2	0,000001	2	16	0,010001	2	10	86,86	16	10
20	34,43	8	2	0,111101	2	16	0,100001	2	10	52,A1	16	10

	6			7			8			9		
#	A	B	C	A	B	C	A	B	C	A	B	C
21	14,67	8	2	0,001101	2	16	0,001011	2	10	1B,08	16	10
22	10,55	8	2	0,110001	2	16	0,101011	2	10	DE,EF	16	10
23	43,71	8	2	0,001111	2	16	0,011101	2	10	68,88	16	10
24	13,36	8	2	0,100001	2	16	0,110011	2	10	81,76	16	10
25	44,12	8	2	0,011111	2	16	0,110011	2	10	2E,22	16	10
26	22,32	8	2	0,011101	2	16	0,001001	2	10	B7,F4	16	10
27	73,14	8	2	0,001001	2	16	0,011001	2	10	1F,1E	16	10
28	41,25	8	2	0,000001	2	16	0,000011	2	10	6F,09	16	10
29	63,56	8	2	0,110101	2	16	0,101111	2	10	B7,93	16	10
30	25,22	8	2	0,101001	2	16	0,101101	2	10	28,D2	16	10

	6			7			8			9		
#	A	B	C	A	B	C	A	B	C	A	B	C
31	55,63	8	2	0,010001	2	16	0,011001	2	10	AD,4D	16	10
32	41,17	8	2	0,100001	2	16	0,000001	2	10	45,19	16	10
33	35,47	8	2	0,011011	2	16	0,100101	2	10	FC,BD	16	10
34	65,21	8	2	0,101001	2	16	0,000101	2	10	FC,2C	16	10
35	61,25	8	2	0,010111	2	16	0,111101	2	10	CD,BF	16	10
36	10,56	8	2	0,011101	2	16	0,010001	2	10	8F,41	16	10
37	26,33	8	2	0,101101	2	16	0,110111	2	10	33,14	16	10
38	33,27	8	2	0,010011	2	16	0,000011	2	10	45,47	16	10
39	35,43	8	2	0,110111	2	16	0,010011	2	10	EE,3C	16	10
40	62,43	8	2	0,100001	2	16	0,111011	2	10	EF,10	16	10

	10			11		
#	A	B	C	A	B	C
1	249	10	Фиб	34{^2}1{^1}	9C	10
2	270	10	Фиб	1{^2}{^3}0{^4}	9C	10
3	292	10	Фиб	{^4}1{^3}22	9C	10
4	315	10	Фиб	703	-10	10
5	339	10	Фиб	814	-10	10
6	621	10	Факт	925	-10	10
7	732	10	Факт	136	-10	10
8	843	10	Факт	1001010	Фиб	10
9	954	10	Факт	1001001	Фиб	10
10	265	10	Факт	1010010	Фиб	10

	10			11		
#	A	B	C	A	B	C
11	651111	Факт	10	117	10	Фиб
12	262320	Факт	10	130	10	Фиб
13	543210	Факт	10	144	10	Фиб
14	430121	Факт	10	159	10	Фиб
15	140301	Факт	10	175	10	Фиб
16	354320	Факт	10	192	10	Фиб
17	142121	Факт	10	175	10	Фиб
18	611020	Факт	10	192	10	Фиб
19	244321	Факт	10	210	10	Фиб
20	613301	Факт	10	229	10	Фиб

			10			11
#	A	B	C	A	B	C
21	42	10	Фиб	147	-10	10
22	45	10	Фиб	258	-10	10
23	49	10	Фиб	369	-10	10
24	54	10	Фиб	470	-10	10
25	60	10	Фиб	581	-10	10
26	67	10	Фиб	692	-10	10
27	75	10	Фиб	33{^2}00	7C	10
28	84	10	Фиб	{^1)303{^2}	7C	10
29	94	10	Фиб	{^1}{^2}{^3}21	7C	10
30	105	10	Фиб	2{^1}33{^3}	7C	10

			10			11
#	A	B	C	A	B	C
31	121	10	Факт	1010101	Фиб	10
32	232	10	Факт	1001001	Фиб	10
33	343	10	Факт	1010010	Фиб	10
34	454	10	Факт	1001000	Фиб	10
35	565	10	Факт	1000101	Фиб	10
36	676	10	Факт	1001001	Фиб	10
37	787	10	Факт	1000100	Фиб	10
38	898	10	Факт	1010001	Фиб	10
39	909	10	Факт	1010010	Фиб	10
40	510	10	Факт	1001001	Фиб	10

Лабораторная работа № 3. Условные операторы и циклы в Python

1 Методические рекомендации

1.1 Условные операторы

Условный оператор ветвления *if*

Оператор ветвления *if* позволяет выполнить определенный набор инструкций в зависимости от некоторого условия. Возможны следующие варианты использования.

1 Конструкция *if*

Синтаксис оператора *if* выглядит так.

```
if выражение:
   инструкция_1
   инструкция_2
   ...
   инструкция_n
```

После оператора *if* записывается выражение. Если это выражение истинно, то выполняются инструкции, определяемые данным оператором. Выражение является истинным, если его результатом является число не равное нулю, непустой объект, либо логическое *True*. После выражения нужно поставить двоеточие ":".

ВАЖНО: блок кода, который необходимо выполнить, в случае истинности выражения, отделяется четырьмя пробелами слева!

Примеры:

```
if 1:
  print("hello 1")
```

Напечатает: *hello 1*

```
a = 3
if a == 3:
   print("hello 2")
```

Напечатает: *hello 2*

```
a = 3
if a > 1:
  print("hello 3")
```

Напечатает: *hello 3*

```
lst = [1, 2, 3]
if lst :
  print("hello 4")
```

Напечатает: *hello 4*

2 Конструкция *if – else*

Бывают случаи, когда необходимо предусмотреть альтернативный вариант выполнения программы. Т.е. при истинном условии нужно выполнить один набор инструкций, при ложном – другой. Для этого используется конструкция *if – else*.

```
if выражение:
```

```
    инструкция_1
    инструкция_2
    ...
    инструкция_n
else:
    инструкция_a
    инструкция_b
    ...
    инструкция_x
```

Примеры.

```
a = 3
if a > 2:
  print("H")
else:
  print("L")
```

Напечатает: *H*

```
a = 1
if a > 2:
  print("H")
else:
  print("L")
```

Напечатает: *L*

Условие такого вида можно записать в строчку, в таком случае оно будет представлять собой тернарное выражение.

```
a = 17
b = True if a > 10 else False
print(b)
```

В результате выполнения такого кода будет напечатано: *True*

3 Конструкция *if – elif – else*

Для реализации выбора из нескольких альтернатив можно использовать конструкцию *if – elif – else*.

```
if выражение_1:
    инструкции_(блок_1)
elif выражение_2:
    инструкции_(блок_2)
elif выражение_3:
    инструкции_(блок_3)
else:
    инструкции_(блок_4)
```

Пример.

```
a = int(input("введите число:"))
if a < 0:
```

```
  print("Neg")
elif a == 0:
  print("Zero")
else:
  print("Pos")
```

Если пользователь введет число меньше нуля, то будет напечатано “*Neg*“, равное нулю – “*Zero*“, большее нуля – “*Pos*“.

1.2 Циклы

Оператор цикла *while*

Оператор цикла *while* выполняет указанный набор инструкций до тех пор, пока условие цикла истинно. Истинность условия определяется также как и в операторе *if*. Синтаксис оператора *while* выглядит так.

```
while выражение:
   инструкция_1
   инструкция_2
   ...
   инструкция_n
```

Выполняемый набор инструкций называется телом цикла.

Пример.

```
a = 0
while a < 7:
  print("A")
  a += 1
```

Буква “A” будет выведена семь раз в столбик.

Пример бесконечного цикла.

```
a = 0
while a == 0:
  print("A")
```

Операторы *break* и *continue*

При работе с циклами используются операторы ***break*** и ***continue***.

Оператор ***break*** предназначен для досрочного прерывания работы цикла *while*.

Пример.

```
a = 0
while a >= 0:
  if a == 7:
     break
  a += 1
  print("A")
```

В приведенном выше коде, выход из цикла произойдет при достижении переменной *a* значения 7. Если бы не было этого условия, то цикл выполнялся бы бесконечно.

Оператор ***continue*** запускает цикл заново, при этом код, расположенный после данного оператора, не выполняется.

Пример.

```
a = -1
while a < 10:
  a += 1
  if a >= 7:
     continue
  print("A")
```

При запуске данного кода символ “A” будет напечатан 7 раз, несмотря на то что всего будет выполнено 11 проходов цикла.

Оператор цикла *for*

Оператор *for* выполняет указанный набор инструкций заданное количество раз, которое определяется количеством элементов в наборе.

Пример.

```
for i in range(5):
  print("Hello")
```

В результате *“Hello”* будет выведено пять раз.

Внутри тела цикла можно использовать операторы ***break*** и ***continue***, принцип работы их точно такой же как и в операторе *while*.

Если у вас есть заданный список, и вы хотите выполнить над каждым элементом определенную операцию (возвести в квадрат и напечатать получившееся число), то с помощью *for* такая задача решается так.

```
lst = [1, 3, 5, 7, 9]
for i in lst:
  print(i ** 2)
```

Также можно пройти по всем буквам в строке.

```
word_str = "Hello, world!"
for l in word_str:
  print(l)
```

Строка *“Hello, world!”* будет напечатана в столбик.

2 Задания для выполнения

Задача 1.

Вариант 1

Напишите код по следующему словесному алгоритму:

Попросить пользователя ввести число от 1 до 9. Полученные данные связать с переменной x.

Если пользователь ввел число от 1 до 3 включительно, то

попросить пользователя ввести строку. Полученные данные связать с переменной s;

попросить пользователя ввести число повторов строки. Полученные данные связать с переменной n, предварительно преобразовав их в целочисленный тип;

выполнить цикл повторения строки n раз;

вывести результат работы цикла.

Если пользователь ввел число от 4 до 6 включительно, то…

попросить пользователя ввести степень, в которую следует возвести число. Полученные данные связать с переменной m;

реализовать возведение числа x в степень m;

вывести полученный результат.

Если пользователь ввел число от 7 до 9, то выполнить увеличения числа x на единицу в цикле 10 раз, при этом на экран вывести все 10 чисел.

Во всех остальных случаях выводить надпись «Ошибка ввода».

Вариант 2

1. Создать произвольный словарь

2. Добавить новый элемент с ключом типа str и значением типа int

3. Добавить новый элемент с ключом типа кортеж(tuple) и значением типа список(list)

4. Получить элемент по ключу

5. Удалить элемент по ключу

6. Получить список ключей словаря

Вариант 3

Проверить, будет ли строка читаться одинаково справа налево и слева направо (т. е. является ли она палиндромом).

Вариант 4

Дана строка. Подсчитать самую длинную последовательность подряд идущих букв «н». Преобразовать ее, заменив точками все восклицательные знаки.

Вариант 5

1. Начав тренировки, лыжник в первый день пробежал 10 км. Каждый следующий день он увеличивал пробег на 10% от пробега предыдущего дня. Определить:

а) пробег лыжника за второй, третий, ..., десятый день тренировок;

б) какой суммарный путь он пробежал за первые 7 дней тренировок.

Решить задачу используя циклическую конструкцию for.

2. Найти сумму и произведение цифр, введенного целого числа. Например, если введено число 325, то сумма его цифр равна 10 (3+2+5), а произведение 30 (3*2*5).

Решить задачу используя циклическую конструкцию while.

Вариант 6

1. Одноклеточная амеба каждые 3 часа делится на 2 клетки. Определить, сколько клеток будет через 3, 6, 9, ..., 24 часа, если первоначально была одна амеба. Решить задачу используя циклическую конструкцию for.

2. Вывести таблицу значений функции y = -0.5x + x. Значения аргумента (x) задаются минимумом, максимумом и шагом. Например, если минимум задан как 1, максимум равен 3, а шаг 0.5. То надо вывести на экран изменение x от 1 до 3 с шагом 0.5 (1, 1.5, 2, 2.5, 3) и значения функции (y) при каждом значении x.

Решить задачу используя циклическую конструкцию while.

Вариант 7

Определить, существует ли треугольник с длинами сторон a, b, c. Если – да, вычислить его площадь по формуле Герона.

Формула Герона имеет вид:

S=p(p-a)(p-b)(p-c), где p=12(a+b+c)

Задача 2.

Варианты:

1 Катеты прямоугольного треугольника равны 7 и 10. Вычислите длину гипотенузы этого треугольника. Ответ дайте с точностью до 10 знаков после запятой.

2 Решите уравнение: $\frac{x}{3}+\frac{x-1}{2}=2$

3 Решите уравнение: $\frac{x-9}{2}-\frac{(x-1)*6}{7}=97$

4 Катеты прямоугольного треугольника равны 6 и 18. Вычислите длину гипотенузы этого треугольника. Ответ дайте с точностью до 9 знаков после запятой.

5 Решите уравнение: $\frac{x-9}{2}-\frac{(x-1)56}{2}+\frac{(x+5)*8}{9}=8$

6 Решите уравнение: $100+\frac{(x+3)5}{2}+\frac{(x/8)*8}{2}=9$

7 Найти периметр, площадь и сумму сторон многогранника n=9

8 Решите уравнение: $\frac{x-9}{2}+\frac{(x-1)*9}{5}=97$

9 Решите уравнение: $2+\frac{(x-1)*9}{5}=7*\frac{x-9}{2}$

10 Найти периметр, площадь и сумму сторон ромба

11 Решите уравнение: $67+x+\frac{(x-1)*7}{9}=\frac{9+x}{7}$

12 Решите уравнение: $9+9*x+\frac{(x+1)*7}{15}=\frac{2+x}{9}$

13 Найти радиус круга, если площадь равна 188. (π принять за 3,14)

14 Решите уравнение: $8*(x+8-1)+\frac{(x+1)*7}{9}=\frac{2+x}{9}$

15 Решите уравнение: $x+\frac{(x+1)*7}{15}=\frac{2+x}{x+9}$+8+x*5

Лабораторная работа № 4. Встроенные и пользовательские функции, lambda-выражения, пакеты и модули функций

1 Методические рекомендации

1.1 Функции

Python делится на системные функции (также известные как встроенные функции, встроенные функции) и пользовательские функции.

Как правило, размер функции должен составлять от 70 до 200 строк кода. Если он меньше этого диапазона, следует рассмотреть вопрос о том, должна ли функция предлагаться отдельно. Если она больше этого диапазона, следует рассмотреть вопрос о целесообразности уточнения функции.

Преимущества функции:

1. Упростить структуру программы и улучшить читаемость кода

2. Функцию можно вызывать повторно, уменьшая запись повторяющегося кода в программе

3. Упростить отладку, модификацию и сопровождение приложения.

4. Способствовать совместному развитию нескольких людей

Объявление функции и вызов

Формат:

def <имя функции> (список параметров):

Тело функции

Пример:

```
def sayHello():
    print 'hello world'

sayHello()
```

```
hello world
```

Примечание:

1. Каждый оператор под количеством строк должен иметь отступ с помощью клавиши табуляции, первая строка без отступа рассматривается как оператор вне тела функции, оператор программы на том же уровне, что и функция

2. Функция не определяет возвращаемый тип данных, функция возвращает указанное значение с помощью оператора return, в противном случае она возвращает нулевое значение (Нет)

3. передача параметров Python

В Python строки, кортежи и числа являются неизменяемыми объектами, в то время как list, dict и т. Д. Являются объектами, которые можно изменять.

- **Неизменные типы:** Переменная присваивания a=5 Назначить позже a=10, Здесь фактически генерируется новый объект 10 значения int, и пусть указывается на него, а 5 отбрасывается, не меняя значения a, что эквивалентно новому генерированию a.

- **Типы переменных:** Переменная присваивания la=[1,2,3,4] Назначить позже la[2]=5 Это изменение значения третьего элемента списка la, который сам не перемещается, изменена только часть его внутреннего значения.

Передача параметров функции python:

- **Неизменные типы:** C ++ - подобные значения передаются, такие как целые числа, строки и кортежи. Как и fun (a), передается только значение a, и сам объект не затрагивается. Например, изменение значения внутри внутри fun (a) изменяет только другой скопированный объект, не затрагивая самого себя.
- **Типы переменных:** C ++ - как передача ссылок, таких как списки, словари. Если fun (la), la передается, и la вне измененного веселья также будет затронуто

Все в Python является объектом. Строго говоря, мы не можем сказать передачу значения или передачу ссылки. Мы должны сказать передачу неизменяемых объектов и передачу изменяемых объектов.

```
def swapRef(x,y):
    t=x
    x=y
    y=t

a=10
b=20
print "exchange before a=",a,"b=",b
swapRef(a,b)
print "exchange after a=",a,"b=",b
exchange before a= 10 b= 20
exchange after a= 10 b= 20
```

Возвращение заявления

Используется для возврата из функции, то есть выпрыгивания из функции и возврата значения из функции.

Функции являются объектами

Все в Python является объектом, поэтому функция также является объектом и может использоваться как обычный объект, назначая функцию другой переменной.

Два типа параметров

Параметры Python делятся на обязательные параметры, параметры по умолчанию, ключевые параметры и параметры переменной длины и т. д.

1 Обязательные параметры

Требуемые параметры должны быть переданы в функцию в правильном порядке, а количество параметров должно быть таким же, как и при объявлении.

```
def printMe(str):
    print str
```

```
        return
printMe()
Traceback (most recent call last):
  File "/tmp/039209824/main.py", line 6, in
    printMe()
TypeError: printMe() takes exactly 1 argument (0 given)

exit status 1
```

Параметры по умолчанию

Параметры по умолчанию допускают параметры функции и значения по умолчанию. Если вы не передадите значения параметрам при вызове функции, параметры получат значения по умолчанию. Python назначает значение параметра по умолчанию для формального параметра, добавляя оператор присваивания (=) и значение по умолчанию после имени формального параметра определения функции

Примечание: значение параметра по умолчанию является неизменным параметром

```
def say(message,times=1):
        print message * times

say('hello')
say('hello',5)
hello
hellohellohellohellohello
```

Ключевые параметры

Множественные значения параметров функции обычно передаются слева направо по умолчанию, но Python также предоставляет гибкий порядок передачи параметров. Кламентальные параметры также называются именованными параметрами, и параметры можно указывать в любом порядке.

```
def func(a,b,c):
        print "a=",a,"and b=",b,"and c=",c
func(1,2,3)
func(a=2,b=3,c=1)

a= 1 and b= 2 and c= 3
a= 2 and b= 3 and c= 1
```

Параметры переменной длины

Параметры переменной длины также называются параметрами неопределенной длины. Параметры, начинающиеся со знака *, представляют предка любой длины и могут получать непрерывный ряд параметров. Параметры, начинающиеся с двух * s, представляют собой словарь, а формат

принимаемых параметров - «ключ = значение», который может принимать любое количество параметров.

```
def foo(x,*y,**z):
    print x
    print y
    print z

foo(1)
foo(1,2,3,4)
foo(1,2,3,a="a",b="b")
1
()
{}
1
(2, 3, 4)
{}
1
(2, 3)
{'a': 'a', 'b': 'b'}
```

Специальные функции

Функция лямбда

Лямбда-функция – это простая анонимная функция. Лямбда-функция возвращает значение выражения по умолчанию

Формат:

лямбда-параметр: выражение

```
f=lambda a,b:a+b
print f(1,2)
```

Приведенный выше пример эквивалентен

```
def f(a,b):
    return a+b
print f(1,2)
```

Рекурсивные функции

Рекурсивная функция - вызывать себя внутри пользовательской функции.

1. Конечное условие рекурсии

2. Может быть выражен в рекурсивной форме, и рекурсия развивается к конечному состоянию

Пример 1: Расчет факториала 4

```
def fact(n):
    if n==1:
        return 1
    return n*fact(n-1)
print fact(4)
```

Области переменных

1 Локальная переменная

Относится к переменной, определенной в теле функции, которая может использоваться только функцией, и не имеет отношения к другим переменным с таким же именем вне функции

2 Глобальные переменные

Глобальные переменные определены вне функции

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

total = 0; # Это глобальная переменная
 # Описание функции записи
def sum( arg1, arg2 ):
      # Возврат суммы двух параметров. "
      total = arg1 + arg2; # total является локальной переменной здесь.
      print "Функция является локальной переменной:", всего
  return total;
 # Вызовите функцию суммы
sum( 10, 20 );
 print "Функция является глобальной переменной:", всего
Локальные переменные в функции: 30
 За пределами функции находятся глобальные переменные: 0
```

1.2 Пространства имен и модули

Пространство имен

Весь код в Python связан с пространством имен.

Пространство имен можно понимать как контейнер. Многие идентификаторы загружаются в контейнер, и идентификаторы с одинаковым именем в разных контейнерах не будут конфликтовать друг с другом.

Поиск порядка имен пространства идентификатора (LEGB):

1 L, сказал в определении функции, и эта функция больше не содержит определение функции

2 E, сказано в определении функции, но эта функция также содержит определение других функций

3 G относится к пространству имен модуля, то есть к идентификатору, определенному в файле .py, но не в функции

4 B. Интерпретатор python автоматически загрузит модуль __builtin__ при запуске, который имеет встроенные функции, такие как list и str.

Модуль определения и экспорта

Модуль является программным подразделением высшего уровня. Модули более детализированы, чем функции. Модуль может содержать несколько функций. Модуль также разделен на системный модуль и пользовательский модуль. Пользовательский модуль представляет собой файл .py

Способ импорта модуля:

1 Модуль импорта

import moduleName

На этом этапе оператор import создает новое пространство имен, которое содержит все объекты в модуле. При доступе к этому пространству имен необходимо использовать имя модуля в качестве префикса.

Пример:

2 Вывод функции под модулем

from moduleName import function1,function2

Используйте метод 2 для импорта указанного объекта в текущее пространство имен, используйте оператор from, и имя модуля не используется в качестве префикса

3 Вывод всех функций модуля

from moduleName import *

2 Задания для выполнения

Разработать функцию для реализации алгоритма сортировки.

Применить метод сортировки для случайно сканированного списка данных (не массива).

Сгенерировать заполнение случайными числами и сохранение их в текстовый файл.

Применить сортировку над массивом, где данные, которого извлечены из текстового файла.

Варианты заданий к лабораторной работе

Вариант 1	Сортировка вставками	Строго случайные данные и «почти отсортированные» случайные данные
Вариант 2	Сортировка выбором	Строго случайные данные и «отсортированные наоборот» случайные данные
Вариант 3	Сортировка пузырьком	Строго случайные данные и случайные данные с малым числом уникальных значений
Вариант 4	Сортировка Шелла	Строго случайные данные и «почти отсортированные» случайные данные
Вариант 5	Сортировкаслиянием	Строго случайные данные и «отсортированные наоборот» случайные данные

Вариант 6	Пирамидальная сортировка	Строго случайные данные и случайные данные с малым числом уникальных значений
Вариант 7	Быстрая сортировка	Строго случайные данные и «почти отсортированные» случайные данные
Вариант 8	Быстрая сортировка разделением на 3 части	Строго случайные данные и «отсортированные наоборот» случайные данные
Вариант 9	Сортировка вставками	Строго случайные данные и случайные данные с малым числом уникальных значений
Вариант 10	Сортировка выбором	Строго случайные данные и «почти отсортированные» случайные данные
Вариант 11	Сортировка	Строго случайные данные и
Вариант 12	Сортировка Шелла	Строго случайные данные и случайные данные с малым числом уникальных значений
Вариант 13	Сортировка слиянием	Строго случайные данные и «почти отсортированные» случайные данные
Вариант 14	Пирамидальная сортировка	Строго случайные данные и «отсортированные наоборот» случайные данные
Вариант 15	Быстрая сортировка	Строго случайные данные и случайные данные с малым числом уникальных значений
Вариант 16	Быстрая сортировка разделением на 3 части	Строго случайные данные и «почти отсортированные» случайные данные

Лабораторная работа № 5. Генераторы списков

1 Методические рекомендации

1.1 Генератор списков

Генератор списков – это простой для чтения, компактный и элегантный способ создания списка из любого существующего итерируемого объекта. По сути, это более простой способ создания нового списка из значений уже имеющегося списка.

Обычно это одна строка кода, заключенная в квадратные скобки. Вы можете использовать генератор для фильтрации, форматирования, изменения или выполнения других небольших задач с существующими итерируемыми объектами, такими как строки, кортежи, множества, списки и т. д.

Сегодня мы разберем несколько способов создания генератора списков и увидим некоторые их вариации, например:

1 Простой генератор списка
2 Генераторы списков с одиночными и вложенными условиями if
3 Генератор списка с одним и несколькими условиями if и else
4 Генератор списков с вложенными циклами for

Помимо этого, мы также рассмотрим следующие концепции:

- Цикл for vs. генератор списка
- Каковы преимущества генератора списка
- Когда использовать, а когда лучше избегать генератора списков

Пример простого генератора списка

Приведенный ниже фрагмент кода является примером простейшего генератора списка. Здесь мы просто перебираем lst и сохраняем все его элементы в списке a:

```
lst = [1,2,3,4,5,6,7,8,9,10]
# простой генератор списка
a = [x for x in lst]
print(a)
# Результат:
# [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Этот код полностью эквивалентен следующему:

```
for x in lst:
a.append(x)
```

Идем дальше. В приведенном выше генераторе списка можно использовать любое выражение для изменения исходных элементов lst, например:

```
# добавить любое число к каждому элементу lst и сохранить результат в a
a = [x+1 for x in lst]
# вычесть любое число из каждого элемента lst и сохранить в a
a = [x-1 for x in lst]
# умножить каждый элемент lst на любое число и сохранить в a
```

1.2 Генератор списка с одиночным и вложенным условием if

В генератор списка также можно добавить if-условие, которое может помочь нам отфильтровать данные. Например, в приведенном ниже коде мы сохраняем в список с все значения lst, большие 4:

```
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
c = [x for x in lst if x > 4]
print(c)
# Результат:
# [5, 6, 7, 8, 9, 10]
```

Этот код выполняет то же самое, что и приведенный ниже:

```
for x in lst:
if x > 4:
a.append(x)
```

Мы также можем добавить в наш генератор списка вложенное условие if. Например, в приведенном ниже коде мы сохраняем в список d все элементы lst, значения которых больше 4 и кратны 2 :

```
d = [x for x in lst if x > 4 if x%2 == 0]
# Результат:
# [6, 8, 10]
```

1.3 Генератор списка с вложенным циклом for

Разберем использование вложенного цикла for в генераторе списка.

Чтобы понять, как это работает, давайте рассмотрим приведенный ниже пример. Здесь мы генерируем все возможные комбинации элементов двух списков: [1, 2, 3] и [3, 2, 1].

```
lst = [1,2,3]
lst_rev = [3,2,1]
g = [(x,y) for x in lst for y in lst_rev]
print(g)

# Результат:
# [(1, 3), (1, 2), (1, 1), (2, 3), (2, 2), (2, 1), (3, 3), (3, 2), (3, 1)]
```

Традиционным способом эта задача решалась бы так:

```
for x in lst:
   for y in lst_rev:
      f.append((x,y))
```

Теперь сравним обычный цикл for и генератор списков.

1.4 Циклы vs. генератор списков

Выше мы видели, как генератор списков позволяет выполнять задачу всего в одну строчку, в то время как цикл for требует написания нескольких строк.

Генератор списков не только более компактен, но также его эффективность выше. В некоторых случаях он оказывается в два раза быстрее, чем цикл for.

Однако если вы хотите выполнить более одного простого условия, генератор списков не сможет справиться с этим без ущерба для удобочитаемости. Это одна из его основных проблем.

1.5 Преимущества генераторов списков

Генератор списков – не только простое, компактное и быстрое, но и надежное решение во многих ситуациях. Его можно использовать в самых разных обстоятельствах. Например, для сопоставления и фильтрации в дополнение к генерации базового списка. Вам не нужно каждый раз изобретать велосипед. Это одна из причин, по которой генераторы списков считаются более «питоничными», чем цикл for.

2 Задания для выполнения

Написать программы в соответствии с номером своего варианта.

Номер	Задание № 1	Задание № 2
1	Вводятся вещественные числа в строку через пробел. Необходимо на их основе сформировать список с помощью list comprehension (генератора списков) из модулей введенных чисел (в списке должны храниться именно числа, а не строки). Результат вывести на экран.	Задается двумерный (вложенный) список, представляющий таблицу целых чисел. Необходимо с помощью list comprehension преобразовать его водномерный так, чтобы значения элементов шли в обратном порядке. Результат преобразования отобразить на экране.
2	Вводится семизначное целое положительное число. С помощью list comprehension сформировать список, содержащий цифры этого числа (в списке должны бытьзаписаны числа, а не строки). Результат вывести на экран водну строку через пробел.	Вводится список целых чисел в строку через пробел. Количество чисел равно N^2. С помощью list comprehension сформировать из них двумерный (вложенный) список размером N x N (квадратную таблицу чисел). Гарантируется, что из набора введенных чисел можно сформировать квадратную матрицу (таблицу). Результат отобразить на экране.

3	Вводится натуральное число N. С помощью list comprehension сформировать двумерный список размером N x N, состоящий из нулей, апо главной диагонали - единицы. (Главная диагональ – это элементы, имеющие одинаковые индексы, например, a[1][1], a[2][2], …). Результат вывести на экран.	Имеется список из строк: t = ["– Скажи-ка, дядя, ведь не даром", "Я Python выучил с каналом", "Наместников что раздавал?"] Необходимо преобразовать его в двумерный (вложенный) список, где каждая строка представляется спискомиз слов (слова разделяются пробелом). При этом сохранять слова только длиной более трех символов. Решить данную задачу с использованием list comprehension. Результат отобразить на экране.
4	Вводятся названия городов в строку через пробел. Необходимо сформировать список с помощью list comprehension, содержащий названия городов длиной более пяти символов. Результат вывести на экран.	Вводятся строки из целых чисел через пробел, пока пользователь не введет пустую строку. Необходимо все введенные строки вначале сохранить всписок. Затем, на основе этого списка,используя list comprehension, сформировать двумерный список, где каждый элемент будет представлять одно отдельное число. Результат вывести на экран.
5	Вводится натуральное число n. Необходимо сформироватьсписок с помощью list comprehension, состоящий из делителей числа n (включая исамо число n). Результат вывести на экран.	Используя вложенный list comprehension, сформируйте двумерный список, представляющийследующую квадратную таблицу чисел размером 4 x 4: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 Результат выведите на экран.

6	Вводится натуральное число N. Необходимо сгенерировать вложенный список с помощью list comprehension, размером N x N, где первая строка содержала бы все нули, вторая - все единицы, третья - все двойки и так до N-й строки. Результат вывести наэкран.	Вводятся два вещественных значения a, b (a < b). С помощью list comprehension сформируйте список со значениями синусов от аргументов в диапазоне [a; b] с шагом 0.1. Результат выведите на экран в виде списка чиселс точностью до сотых.
7	Вводится список вещественных чисел. С помощью list comprehension сформировать список, состоящий из элементов введенного списка, имеющих четные индексы (то есть, выбрать все элементы с четными индексами). Результат вывести на экран.	Вводятся названия три строки: первая строка содержит названия городов, вторая – названия стран, а третья – названия рек. Все названия следуют в строке через пробел. С помощью list comprehension сформируйте единый список из слов, длины которых больше пяти. Результат выведите на экран.
8	Вводятся два списка целых чисел одинаковой длины каждый с новой строки. С помощью list comprehension сформировать третий список,состоящий из суммы соответствующих пар чисел введенных списков. Результат вывести на экран.	Имеется двумерный список чисел.Например: d = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [8, 7, 6]] С помощью list comprehension необходимо сформировать новый список, в котором строки идут в обратном порядке. Результат выведите на экран.

9	Вводятся названия стран в одну строчку через пробел. С помощью list comprehension сформировать список, состоящий из названий стран,в которых присутствует фрагмент «ро» (без учёта регистра). Результат вывести на экран.	Имеется трехмерный список.Например: t = [[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [9, 8, 7]], [[0, 1, 2], [-1, -2]]] С помощью list comprehension необходимо сформировать новый одномерный список, состоящий из значений элементов списка t. Результат выведите на экран.
10	Вводятся два натуральных числа a, b (a < b). С помощью list comprehension сформировать список чисел вдиапазоне [a; b] с шагом 0.1. Результат вывести на экран.	Вводится строка с координатами точекв формате (пример): 5;4 -3;2 7;56 -4;-10 … То есть, пары координат разделены пробелом, а сами координаты – точкойс запятой. При этом все числовые значения – целые числа. Необходимо с помощью list comprehension преобразовать эту строку в двумерныйсписок вида (пример): [[5, 4], [-3, 2], [7, 56], …] Результат вывести на экран.

Лабораторная работа № 6. Библиотека Pandas для анализа данных

1 Методические рекомендации

1.1 Библиотека Pandas. Структуры данных Series и DataFrame

Ядром pandas являются две **структуры данных**, в которых происходят все операции:

- Series
- Dataframes

Series − это структура, используемая для работы с последовательностью одномерных данных, а Dataframe − более сложная и подходит для нескольких измерений.

Пусть они и не являются универсальными для решения всех проблем, предоставляют отличный инструмент для большинства приложений. При этом их легко использовать, а множество более сложных структур можно упросить до одной из этих двух.

Однако особенности этих структур основаны на одной черте − интеграции в их структуру объектов index и labels (метки). С их помощью структурами становится очень легко манипулировать.

Series (серии)

Series − это объект **библиотеки pandas**, спроектированный для представления одномерных структур данных, похожих на массивы, но с дополнительными возможностями. Его структура проста, ведь он состоит из двух связанных между собой массивов. Основной содержит данные (данные любого типа NumPy), а в дополнительном, index, хранятся метки.

Series	
index	value
0	12
1	-4
2	7
3	9

Рис. 6.1. Хранение данных

Создание объекта Series

Для создания объекта Series с предыдущего изображения необходимо вызвать конструктор Series() и передать в качестве аргумента массив, содержащий значения, которые необходимо включить.

```
>>> s = pd.Series([12,-4,7,9])
>>> s
0  12
1  -4
```

```
2    7
3    9
dtype: int64
```

Как можно увидеть по выводу, слева отображаются значения индексов, а справа − сами значения (данные).

Если не определить индекс при объявлении объекта, метки будут соответствовать индексам (положению в массиве) элементов объекта Series.

Однако лучше создавать Series, используя метки с неким смыслом, чтобы в будущем отделять и идентифицировать данные вне зависимости от того, в каком порядке они хранятся.

В таком случае необходимо будет при вызове конструктора включить параметр index и присвоить ему массив строк с метками.

```
>>> s = pd.Series([12,-4,7,9], index=['a','b','c','d'])
>>> s
a   12
b   -4
c    7
d    9
dtype: int64
```

Если необходимо увидеть оба массива, из которых состоит структура, можно вызвать два атрибута: index и values.

```
>>> s.values
array([12, -4, 7, 9], dtype=int64)
>>> s.index
Index(['a', 'b', 'c', 'd'], dtype='object')
```

Выбор элементов по индексу или метке

Выбирать отдельные элементы можно по принципу обычных массивов numpy, используя для этого индекс.

```
>>> s[2]
7
```

Или же можно выбрать метку, соответствующую положению индекса.

```
>>> s['b']
-4
```

Таким же образом можно выбрать несколько элементов массива numpy с помощью следующей команды:

```
>>> s[0:2]
a   12
b   -4
dtype: int64
```

В этом случае можно использовать соответствующие метки, но указать их список в массиве.

```
>>> s[['b','c']]
b   -4
c    7
```

```
dtype: int64
```

Присваивание значений элементам

Понимая как выбирать отдельные элементы, важно знать и то, как присваивать им новые значения. Можно делать это по индексу или по метке.

```
>>> s[1] = 0
>>> s
a   12
b    0
c    7
d    9
dtype: int64
>>> s['b'] = 1
>>> s
a   12
b    1
c    7
d    9
dtype: int64
```

Создание Series из массивов NumPy

Новый объект Series можно создать из массивов NumPy и уже существующих Series.

```
>>> arr = np.array([1,2,3,4])
>>> s3 = pd.Series(arr)
>>> s3
0   1
1   2
2   3
3   4
dtype: int32
>>> s4 = pd.Series(s)
>>> s4
a   12
b    1
c    7
d    9
dtype: int64
```

Важно запомнить, что значения в массиве NumPy или оригинальном объекте Series не копируются, а передаются по ссылке. Это значит, что элементы объекта вставляются динамически в новый Series. Если меняется оригинальный объект, то меняются и его значения в новом.

```
>>> s3
0   1
```

```
1    2
2    3
3    4
dtype: int32
>>> arr[2] = -2
>>> s3
0    1
1    2
2   -2
3    4
dtype: int32
```

На этом примере можно увидеть, что при изменении третьего элемента массива arr, меняется соответствующий элемент и в s3.

Фильтрация значений

Благодаря тому что основной библиотекой в **pandas** является NumPy, многие операции, применяемые к массивам NumPy, могут быть использованы и в случае с Series. Одна из таких − фильтрация значений в структуре данных с помощью условий.

Например, если нужно узнать, какие элементы в Series больше 8, то можно написать следующее:

```
>>> s[s > 8]
a   12
d    9
dtype: int64
```

Операции и математические функции

Другие операции, такие как операторы (+, -, * и /), а также математические функции, работающие с массивами **NumPy**, могут использоваться и для Series.

Для операторов можно написать простое арифметическое уравнение.

```
>>> s / 2
a   6.0
b   0.5
c   3.5
d   4.5
dtype: float64
```

Но в случае с математическими функциями NumPy необходимо указать функцию через np, а Series передать в качестве аргумента.

```
>>> np.log(s)
a   2.484907
b   0.000000
c   1.945910
d   2.197225
dtype: float64
```

Количество значений

В Series часто встречаются повторения значений. Поэтому важно иметь информацию, которая бы указывала на то, есть ли дубликаты или конкретное значение в объекте.

Так, можно объявить Series, в котором будут повторяющиеся значения.

```
>>> serd = pd.Series([1,0,2,1,2,3], index=['white','white','blue','green','
green','yellow'])
>>> serd
white    1
white    0
blue     2
green    1
green    2
yellow   3
dtype: int64
```

Чтобы узнать обо всех значениях в Series, не включая дубли, можно использовать функцию unique(). Возвращаемое значение − массив с уникальными значениями, необязательно в том же порядке.

```
>>> serd.unique()
array([1, 0, 2, 3], dtype=int64)
```

На unique() похожа функция value_counts(), которая возвращает не только уникальное значение, но и показывает, как часто элементы встречаются в Series.

```
>>> serd.value_counts()
2   2
1   2
3   1
0   1
dtype: int64
```

Наконец, isin() показывает, есть ли элементы на основе списка значений. Она возвращает булевые значения, которые очень полезны при фильтрации данных в Series или в колонке Dataframe.

```
>>> serd.isin([0,3])
white    False
white     True
blue     False
green    False
green    False
yellow    True
dtype: bool
>>> serd[serd.isin([0,3])]
white    0
yellow   3
dtype: int64
```

Значения NaN

В предыдущем примере мы попробовали получить логарифм отрицательного числа и результатом стало значение NaN. Это значение (Not a Number) используется в структурах данных pandas для обозначения наличия пустого поля или чего-то, что невозможно обозначить в числовой форме.

Как правило, NaN − это проблема, для которой нужно найти определенное решение, особенно при работе с анализом данных. Эти данные часто появляются при извлечении информации из непроверенных источников или когда в самом источнике недостает данных. Также значения NaN могут генерироваться в специальных случаях, например, при вычислении логарифмов для отрицательных значений, в случае исключений при вычислениях или при использовании функций. Есть разные стратегии работы со значениями NaN.

Несмотря на свою «проблемность» pandas позволяет явно определять NaN и добавлять это значение в структуры, например, в Series. Для этого внутри массива достаточно ввести np.NaN в том месте, где требуется определить недостающее значение.

```
>>> s2 = pd.Series([5,-3,np.NaN,14])
>>> s2
0     5.0
1    -3.0
2     NaN
3    14.0
dtype: float64
```

Функции isnull() и notnull() очень полезны для определения индексов без значения.

```
>>> s2.isnull()
0    False
1    False
2     True
3    False
dtype: bool
>>> s2.notnull()
0     True
1     True
2    False
3     True
dtype: bool
```

Они возвращают два объекта Series с булевыми значениями, где True указывает на наличие значение, а NaN − на его отсутствие. Функция isnull() возвращает True для значений NaN в Series, а notnull() − True в тех местах, где значение не равно NaN. Эти функции часто используются в фильтрах для создания условий.

```
>>> s2[s2.notnull()]
0     5.0
```

```
1   -3.0
3   14.0
dtype: float64
s2[s2.isnull()]
2   NaN
dtype: float64
```

Series из словарей

Series можно воспринимать как объект dict (словарь). Эта схожесть может быть использована на этапе объявления объекта. Даже создавать Series можно на основе существующего dict.

```
>>> mydict = {'red': 2000, 'blue': 1000, 'yellow': 500,
 'orange': 1000}
>>> myseries = pd.Series(mydict)
>>> myseries
blue      1000
orange    1000
red       2000
yellow     500
dtype: int64
```

На этом примере можно увидеть, что массив индексов заполнен ключами, а данные − соответствующими значениями. В таком случае соотношение будет установлено между ключами dict и метками массива индексов. Если есть несоответствие, pandas заменит его на NaN.

```
>>> colors = ['red','yellow','orange','blue','green']
>>> myseries = pd.Series(mydict, index=colors)
>>> myseries
red       2000.0
yellow     500.0
orange    1000.0
blue      1000.0
green        NaN
dtype: float64
```

Операции с сериями

Вы уже видели, как выполнить арифметические операции на объектах Series и скалярных величинах. То же возможно и для двух объектов Series, но в таком случае в дело вступают и метки.

Одно из главных достоинств этого типа структур данных в том, что он может выравнивать данные, определяя соответствующие метки.

В следующем примере добавляются два объекта Series, у которых только некоторые метки совпадают.

```
>>> mydict2 = {'red':400,'yellow':1000,'black':700}
>>> myseries2 = pd.Series(mydict2)
>>> myseries + myseries2
```

```
black     NaN
blue      NaN
green     NaN
orange    NaN
red     2400.0
yellow  1500.0
dtype: float64
```

Новый объект получает только те элементы, где метки совпали. Все остальные тоже присутствуют, но со значением NaN.

DataFrame (датафрейм)

Dataframe − это табличная структура данных, напоминающая таблицы из Microsoft Excel. Ее главная задача − позволить использовать многомерные Series. Dataframe состоит из упорядоченной коллекции колонок, каждая из которых содержит значение разных типов (числовое, строковое, булевое и так далее).

DataFrame

	columns		
index	color	object	price
0	blue	ball	1.2
1	green	pen	1.0
2	yellow	pencil	0.6
3	red	paper	0.9
4	white	mug	1.7

Рис. 6.2. DataFrame

В отличие от Series у которого есть массив индексов с метками, ассоциированных с каждым из элементов, Dataframe имеет сразу два таких. Первый ассоциирован со строками (рядами) и напоминает таковой из Series. Каждая метка ассоциирована со всеми значениями в ряду. Второй содержит метки для каждой из колонок.

Dataframe можно воспринимать как dict, состоящий из Series, где ключи − названия колонок, а значения − объекты Series, которые формируют колонки самого объекта Dataframe. Наконец, все элементы в каждом объекте Series связаны в соответствии с массивом меток, называемым index.

Создание Dataframe

Простейший способ создания Dataframe − передать объект dict в конструктор DataFrame(). Объект dict содержит ключ для каждой колонки, которую требуется определить, а также массив значений для них.

Если объект dict содержит больше данных, чем требуется, можно сделать выборку. Для этого в конструкторе Dataframe нужно определить последовательность колонок с помощью параметра column. Колонки будут созданы в заданном порядке вне зависимости от того, как они расположены в объекте dict.

```
>> data = {'color' : ['blue', 'green', 'yellow', 'red', 'white'],
      'object' : ['ball', 'pen', 'pencil', 'paper', 'mug'],
      'price' : [1.2, 1.0, 0.6, 0.9, 1.7]}
>>> frame = pd.DataFrame(data)
>>> frame
```

	color	**object**	**price**
0	blue	ball	1.2
1	green	pen	1.0
2	yellow	pencil	0.6
3	red	paper	0.9
4	white	mug	1.7

Даже для объектов Dataframe если метки явно не заданы в массиве index, pandas автоматически присваивает числовую последовательность, начиная с нуля. Если же индексам Dataframe нужно присвоить метки, необходимо использовать параметр index и присвоить ему массив с метками.

```
>>> frame2 = pd.DataFrame(data, columns=['object', 'price'])
>>> frame2
```

	object	**price**
0	ball	1.2
1	pen	1.0
2	pencil	0.6
3	paper	0.9
4	mug	1.7

Теперь, зная о параметрах index и columns, проще использовать другой способ определения Dataframe. Вместо использования объекта dict можно определить три аргумента в конструкторе в следующем порядке: матрицу данных, массив значений для параметра index и массив с названиями колонок для параметра columns.

В большинстве случаев простейший способ создать матрицу значений − использовать np.arrange(16).reshape((4,4)). Это формирует матрицу размером 4x4 из чисел от 0 до 15.

```
>>> frame3 = pd.DataFrame(np.arange(16).reshape((4,4)),
...                 index=['red', 'blue', 'yellow', 'white'],
...                 columns=['ball', 'pen', 'pencil', 'paper'])
>>> frame3
```

Выбор элементов

Если нужно узнать названия всех колонок Dataframe, можно вызвать атрибут columns для экземпляра объекта.

```
>>> frame.columns
Index(['color', 'object', 'price'], dtype='object')
```

То же можно проделать и для получения списка индексов.

```
>>> frame.index
RangeIndex(start=0, stop=5, step=1)
```

Весь же набор данных можно получить с помощью атрибута values.

```
>>> frame.values
array([['blue', 'ball', 1.2],
       ['green', 'pen', 1.0],
       ['yellow', 'pencil', 0.6],
       ['red', 'paper', 0.9],
       ['white', 'mug', 1.7]], dtype=object)
```

Указав в квадратных скобках название колонки, можно получить значений в ней.

```
>>> frame['price']
0  1.2
1  1.0
2  0.6
3  0.9
4  1.7
Name: price, dtype: float64
```

Возвращаемое значение − объект Series. Название колонки можно использовать и в качестве атрибута.

```
>>> frame.price
0  1.2
1  1.0
2  0.6
3  0.9
4  1.7
Name: price, dtype: float64
```

Для строк внутри Dataframe используется атрибут loc со значением индекса нужной строки.

```
>>> frame.loc[2]
color     yellow
object    pencil
price        0.6
Name: 2, dtype: object
```

Возвращаемый объект − это снова Series, где названия колонок − это уже метки массива индексов, а значения − данные Series.

Для выбора нескольких строк можно указать массив с их последовательностью.

```
>>> frame.loc[[2,4]]
```

	color	object	price
2	yellow	pencil	0.6
4	white	mug	1.7

Если необходимо извлечь часть Dataframe с конкретными строками, для этого можно использовать номера индексов. Она выведет данные из соответствующей строки и названия колонок.

>>> frame[0:1]

	color	object	price
2	yellow	pencil	0.6
4	white	mug	1.7

Возвращаемое значение − объект Dataframe с одной строкой. Если нужно больше одной строки, необходимо просто указать диапазон.

>>> frame[1:3]

	color	object	price
0	blue	ball	1.2

Наконец, если необходимо получить одно значение из объекта, сперва нужно указать название колонки, а потом − индекс или метку строки.

```
>>> frame['object'][3]
'paper'
```

Присваивание и замена значений

Разобравшись с логикой получения доступа к разным элементам Dataframe, можно следовать ей же для добавления новых или изменения уже существующих значений.

Например, в структуре Dataframe массив индексов определен атрибутом index, а строка с названиями колонок − columns. Можно присвоить метку с помощью атрибута name для этих двух подструктур, чтобы идентифицировать их.

```
>>> frame.index.name = 'id'
>>> frame.columns.name = 'item'
>>> frame
```

item	color	object	price
id			
0	blue	ball	1.2
1	green	pen	1.0
2	yellow	pencil	0.6
3	red	paper	0.9
4	white	mug	1.7

Одна из главных особенностей структур данных pandas − их гибкость. Можно вмешаться на любом уровне для изменения внутренней структуры данных. Например, добавление новой колонки − крайне распространенная операция.

Ее можно выполнить, присвоив значение экземпляру Dataframe и определив новое имя колонки.

>>> frame['new'] = 12

>>> frame

item	color	object	price	new
id				
0	blue	ball	1.2	12
1	green	pen	1.0	12
2	yellow	pencil	0.6	12
3	red	paper	0.9	12
4	white	mug	1.7	12

Здесь видно, что появилась новая колонка new со значениями 12 для каждого элемента.

Для обновления значений можно использовать массив.

frame['new'] = [3.0, 1.3, 2.2, 0.8, 1.1]

frame

item	color	object	price	new
id				
0	blue	ball	1.2	3.0
1	green	pen	1.0	1.3
2	yellow	pencil	0.6	2.2
3	red	paper	0.9	0.8
4	white	mug	1.7	1.1

Тот же подход используется для обновления целой колонки. Например, можно применить функцию np.arrange() для обновления значений колонки с помощью заранее заданной последовательности.

Колонки Dataframe также могут быть созданы с помощью присваивания объекта Series одной из них, например, определив объект Series, содержащий набор увеличивающихся значений с помощью np.arrange().

```
>>> ser = pd.Series(np.arange(5))
>>> ser
0  0
1  1
2  2
3  3
4  4
dtype: int32
frame['new'] = ser
frame
```

item	color	object	price	new
id				
0	blue	ball	1.2	0
1	green	pen	1.0	1
2	yellow	pencil	0.6	2
3	red	paper	0.9	3
4	white	mug	1.7	4

Наконец, для изменения одного значения нужно лишь выбрать элемент и присвоить ему новое значение.

```
>>> frame['price'][2] = 3.3
```

Вхождение значений

Функция isin() используется с объектами Series для определения вхождения значений в колонку. Она же подходит и для объектов Dataframe.

```
>>> frame.isin([1.0,'pen'])
```

item	color	object	price	new
id				
0	False	False	False	False
1	False	True	True	True
2	False	False	False	False
3	False	False	False	False
4	False	False	False	False

Возвращается Dataframe с булевыми значениями, где True указывает на те значения, где членство подтверждено. Если передать это значение в виде условия, тогда вернется Dataframe, где будут только значения, удовлетворяющие условию.

```
>>> frame[frame.isin([1.0,'pen'])]
```

item	color	object	price	new
id				
0	NaN	NaN	NaN	NaN
1	NaN	pen	1.0	1.0
2	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN

Удаление колонки

Для удаления целой колонки и всего ее содержимого используется команда del.

```
>>> del frame['new']
>>> frame
```

item	color	object	price
id			
0	blue	ball	1.2
1	green	pen	1.0
2	yellow	pencil	3.3
3	red	paper	0.9
4	white	mug	1.7

Фильтрация

Даже для Dataframe можно применять фильтры, используя определенные условия. Например, вам нужно получить все значения меньше определенного числа (допустим, 1,2).

```
>>> frame[frame  <  1.2]
```

item	color	object	price
id			
0	blue	ball	NaN
1	green	pen	1.0
2	yellow	pencil	NaN
3	red	paper	0.9
4	white	mug	NaN

Результатом будет Dataframe со значениями меньше 1,2 на своих местах. На месте остальных будет NaN.

Dataframe из вложенного словаря

В Python часто используется вложенный dict:

```
>>> nestdict = {'red': { 2012: 22, 2013: 33},
...          'white': { 2011: 13, 2012: 22, 2013: 16},
...          'blue': { 2011: 17, 2012: 27, 2013: 18}}
```

Эта структура данных, будучи переданной в качестве аргумента в DataFrame(), интерпретируется pandas так, что внешние ключи становятся названиями колонок, а внутренние − метками индексов.

При интерпретации вложенный структуры возможно такое, что не все поля будут совпадать. pandas компенсирует это несоответствие, добавляя NaN на место недостающих значений.

```
>>> nestdict = {'red': { 2012: 22, 2013: 33},
...          'white': { 2011: 13, 2012: 22, 2013: 16},
...          'blue': { 2011: 17, 2012: 27, 2013: 18}}
>>> frame2 = pd.DataFrame(nestdict)
>>> frame2
```

	blue	red	white
2011	17	NaN	13
2012	27	22.0	22
2013	18	33.0	16

Транспонирование Dataframe

При работе с табличным структурами данных иногда появляется необходимость выполнить операцию перестановки (сделать так, чтобы колонки стали рядами и наоборот). pandas позволяет добиться этого очень просто. Достаточно добавить атрибут T.

```
>>> frame2.T
```

	2011	2012	2013
blue	17.0	27.0	18.0

red	NaN	22.0	33.0
white	13.0	22.0	16.0

Объекты Index

Зная, что такое Series и Dataframes, и понимая как они устроены, проще разобраться со всеми их достоинствами. Главная особенность этих структур − наличие объекта Index, который в них интегрирован.

Объекты Index являются метками осей и содержат другие метаданные. Вы уже знаете, как массив с метками превращается в объект Index, и что для него нужно определить параметр index в конструкторе.

```
>>> ser = pd.Series([5,0,3,8,4], index=['red','blue','yellow','white','green'])
>>> ser.index
Index(['red', 'blue', 'yellow', 'white', 'green'], dtype='object')
```

В отличие от других элементов в структурах данных pandas (Series и Dataframe) объекты index − неизменяемые. Это обеспечивает безопасность, когда нужно передавать данные между разными структурами.

У каждого объекта Index есть методы и свойства, которые нужны, чтобы узнавать значения.

Методы Index

Есть методы для получения информации об индексах из структуры данных. Например, idmin() и idmax() − структуры, возвращающие индексы с самым маленьким и большим значениями.

```
>>> ser.idxmin()
'blue'
>>> ser.idxmax()
'white'
```

Индекс с повторяющимися метками

Пока что были только те случаи, когда у индексов одной структуры лишь одна, уникальная метка. Для большинства функций это обязательное условие, но не для структур данных pandas.

Определим, например, Series с повторяющимися метками.

```
>>> serd = pd.Series(range(6), index=['white','white','blue','green',
'green','yellow'])
>>> serd
white    0
white    1
blue     2
green    3
green    4
yellow   5
dtype: int64
```

Если метке соответствует несколько значений, то она вернет не один элемент, а объект Series.

```
>>> serd['white']
white    0
white    1
dtype: int64
```

То же применимо и к Dataframe. При повторяющихся индексах он возвращает Dataframe.

В случае с маленькими структурами легко определять любые повторяющиеся индексы, но если структура большая, то растет и сложность этой операции. Для этого в pandas у объектов Index есть атрибут is_unique. Он сообщает, есть ли индексы с повторяющимися метками в структуре (Series или Dataframe).

```
>>> serd.index.is_unique
False
>>> frame.index.is_unique
True
```

Операции между структурами данных

Теперь когда вы знакомы со структурами данных, Series и Dataframe, а также базовыми операциями для работы с ними, стоит рассмотреть операции, включающие две или более структур.

Гибкие арифметические методы

Уже рассмотренные операции можно выполнять с помощью гибких арифметических методов:

- add()
- sub()
- div()
- mul()

Для их вызова нужно использовать другую спецификацию. Например, вместо того чтобы выполнять операцию для двух объектов Dataframe по примеру frame1 + frame2, потребуется следующий формат:

```
>>> frame1.add(frame2)
```

	ball	**mug**	**paper**	**pen**	**pencil**
blue	6.0	NaN	NaN	6.0	NaN
green	NaN	NaN	NaN	NaN	NaN
red	NaN	NaN	NaN	NaN	NaN
white	20.0	NaN	NaN	20.0	NaN
yellow	19.0	NaN	NaN	19.0	NaN

Результат такой же, как при использовании оператора сложения +. Также стоит обратить внимание, что если названия индексов и колонок сильно отличаются, то результатом станет новый объект Dataframe, состоящий только из значений NaN.

Операции между Dataframe и Series

Pandas позволяет выполнять переносы между разными структурами, например, между Dataframe и Series. Определить две структуры можно следующим образом.

```
>>> frame = pd.DataFrame(np.arange(16).reshape((4,4)),
...          index=['red', 'blue', 'yellow', 'white'],
...          columns=['ball','pen','pencil','paper'])
>>> frame
```

	ball	**pen**	**pencil**	**paper**
red	0	1	2	3
blue	4	5	6	7
yellow	8	9	10	11
white	12	13	14	15

```
>>> ser = pd.Series(np.arange(4), index=['ball','pen','pencil','paper'])
>>> ser
ball      0
pen       1
pencil    2
paper     3
dtype: int32
```

Они были специально созданы так, чтобы индексы в Series совпадали с названиями колонок в Dataframe. В таком случае можно выполнить прямую операцию.

```
>>> frame – ser
```

	ball	**pen**	**pencil**	**paper**
red	0	0	0	0
blue	4	4	4	4
yellow	8	8	8	8
white	12	12	12	12

По результату видно, что элементы Series были вычтены из соответствующих тому же индексу в колонках значений Dataframe.

Если индекс не представлен ни в одной из структур, то появится новая колонка с этим индексом и значениями NaN.

```
>>> ser['mug'] = 9
>>> ser
ball      0
pen       1
pencil    2
paper     3
mug       9
dtype: int64
>>> frame – ser
```

	ball	mug	paper	pen	pencil
red	0	NaN	0	0	0
blue	4	NaN	4	4	4
yellow	8	NaN	8	8	8
white	12	NaN	12	12	12

1.2 Основные функции для работы с DataFrame

Создайте фрейм данных

DataFrame([data, index, columns, dtype, copy]) *# Создать фрейм данных*

Атрибуты и данные

DataFrame.axes #index: метки строк; столбцы: метки столбцов

DataFrame.as_matrix([columns]) # Преобразовать в матрицу

DataFrame.dtypes # Возврат типа данных

DataFrame.ftypes # Возвращаем тип данных каждого столбца float64: density

DataFrame.get_dtype_counts() # Возвращает количество типов данных фрейма данных

DataFrame.get_ftype_counts() # Возвращает количество фреймов данных типа данных float64: density

DataFrame.select_dtypes([include, include]) # Выберите подкадр данных в соответствии с типом данных

DataFrame.values #Numpy как отображать

DataFrame.axes # Вернуть название метки горизонтальных и вертикальных координат

DataFrame.ndim # Вернуться к широте фрейма данных

DataFrame.size # Вернуть количество элементов фрейма данных

DataFrame.shape # Вернуться к форме фрейма данных

DataFrame.memory_usage() # Хранение каждого столбца

Преобразование типов

DataFrame.astype(dtype[, copy, errors]) # Преобразование типа данных

DataFrame.copy([deep]) #deepDeep копирование данных

DataFrame.isnull() # Возвращать нулевое значение логическим способом

DataFrame.notnull() # Возврат ненулевого значения логическим способом

Индекс и итерация

DataFrame.head([n]) # Вернуться к первым n строкам данных

DataFrame.at # Quick label постоянный метод доступа

DataFrame.iat # Быстрый метод доступа к целочисленным константам

DataFrame.loc # Позиционирование ярлыка, используйте название

DataFrame.iloc # Интегральное позиционирование, используйте числа

DataFrame.insert(loc, column, value) # В специальном месте loc [номер] вставить столбец [имя столбца] определенный столбец данных

DataFrame.iter() #Iterate over infor axis

DataFrame.iteritems() # Возврат итератора имен столбцов и последовательностей

DataFrame.iterrows() # Возврат индекса и итератора последовательности

DataFrame.itertuples([index, name]) #Iterate over DataFrame rows as namedtuples, with index value as first element of the tuple.

DataFrame.lookup(row_labels, col_labels) #Label-based “fancy indexing” function for DataFrame.

DataFrame.pop(item) # Вернуться к удаленным элементам

DataFrame.tail([n]) # Вернуться к последним n строкам

DataFrame.xs(key[, axis, level, drop_level]) #Returns a cross-section (row(s) or column(s)) from the Series/DataFrame.

DataFrame.isin(values) # Включать ли элементы во фрейм данных

DataFrame.where(cond[, other, inplace, …]) # Условный фильтр

DataFrame.mask(cond[, other, inplace, …]) #Return an object of same shape as self and whose corresponding entries are from self where cond is False and otherwise are from other.

DataFrame.query(expr[, inplace]) #Query the columns of a frame with a boolean expression.

Бинарная операция

DataFrame.add(other[,axis,fill_value]) # Дополнение, элементные баллы

DataFrame.sub(other[,axis,fill_value]) # Вычитание, элементные баллы

DataFrame.mul(other[, axis,fill_value]) # Умножение, элементные очки

DataFrame.div(other[, axis,fill_value]) # Десятичное деление, элементные баллы

DataFrame.truediv(other[, axis, level, …]) # Истинное деление, элементные баллы

DataFrame.floordiv(other[, axis, level, …]) # Метод десятичного деления, элементные баллы

DataFrame.mod(other[, axis,fill_value]) # Модульная работа, указатель элемента

DataFrame.pow(other[, axis,fill_value]) # Силовой режим, элементные баллы

DataFrame.radd(other[, axis,fill_value]) # Дополнение справа, элемент, указывающий

DataFrame.rsub(other[, axis,fill_value]) # Правильное вычитание, элементные баллы

DataFrame.rmul(other[, axis,fill_value]) # Умножение справа, элементные точки

DataFrame.rdiv(other[, axis,fill_value]) # Правая часть десятичного деления, элемент указывает на

DataFrame.rtruediv(other[, axis, …]) # Истинное деление справа, элементные баллы

DataFrame.rfloordiv(other[, axis, …]) # Правая часть округлена в меньшую сторону, а элемент указывает на

DataFrame.rmod(other[, axis,fill_value]) # Модульная операция справа, элемент указывает

DataFrame.rpow(other[, axis,fill_value]) # Правостороннее управление мощностью, элементные баллы

DataFrame.lt(other[, axis, level]) # Аналогично Array.lt

DataFrame.gt(other[, axis, level]) # Аналогично Array.gt

DataFrame.le(other[, axis, level]) # Аналогично Array.le

DataFrame.ge(other[, axis, level]) # Аналогично Array.ge

DataFrame.ne(other[, axis, level]) # Аналогично Array.ne

DataFrame.eq(other[, axis, level]) # Аналогично Array.eq

DataFrame.combine(other,func[,fill_value, …]) #Add two DataFrame objects and do not propagate NaN values, so if for a

DataFrame.combine_first(other) #Combine two DataFrame objects and default to non-null values in frame calling the method.

Приложение функции и группировка и окно

DataFrame.apply(func[, axis, broadcast, …]) #Application function

DataFrame.applymap(func) #Apply a function to a DataFrame that is intended to operate elementwise, i.e.

DataFrame.aggregate(func[, axis]) #Aggregate using callable, string, dict, or list of string/callables

DataFrame.transform(func, *args, **kwargs) #Call function producing a like-indexed NDFrame

DataFrame.groupby([by, axis, level, …]) #Группа

DataFrame.rolling(window[, min_periods, …]) # Прокручивающееся окно

DataFrame.expanding([min_periods, freq, …]) # Развернуть окно

DataFrame.ewm([com, span, halflife, …]) # Окно веса индекса

Описательная статистика

DataFrame.abs() # Вернуть абсолютное значение

DataFrame.all([axis, bool_only, skipna]) #Return whether all elements are True over requested axis

DataFrame.any([axis, bool_only, skipna]) #Return whether any element is True over requested axis

DataFrame.clip([lower, upper, axis]) #Trim values at input threshold(s).

DataFrame.clip_lower(threshold[, axis]) #Return copy of the input with values below given value(s) truncated.

DataFrame.clip_upper(threshold[, axis]) #Return copy of input with values above given value(s) truncated.

DataFrame.corr([method, min_periods]) # Вернуться к коэффициенту корреляции парных столбцов этого фрейма данных

DataFrame.corrwith(other[, axis, drop]) # Вернуться к корреляции разных фреймов данных

DataFrame.count([axis, level, numeric_only]) # Вернуть количество непустых элементов

DataFrame.cov([min_periods]) # Вычислить ковариацию

DataFrame.cummax([axis, skipna]) #Return cumulative max over requested axis.

DataFrame.cummin([axis, skipna]) #Return cumulative minimum over requested axis.

DataFrame.cumprod([axis, skipna]) # Вернуться к накоплению

DataFrame.cumsum([axis, skipna]) # Вернуться к уставшим и

DataFrame.describe([percentiles,include, …]) # Общее описание фрейм данных

DataFrame.diff([periods, axis]) #1st discrete difference of object

DataFrame.eval(expr[, inplace]) #Evaluate an expression in the context of the calling DataFrame instance.

DataFrame.kurt([axis, skipna, level, …]) # Возврат к беспристрастному эксцессу Фишера (нормальный эксцесс == 0,0).

DataFrame.mad([axis, skipna, level]) # Отклонение возврата

DataFrame.max([axis, skipna, level, …]) # Максимум возврата

DataFrame.mean([axis, skipna, level, …]) # Вернуться к значению

DataFrame.median([axis, skipna, level, …]) # Возврат медианы

DataFrame.min([axis, skipna, level, …]) # Возврат к минимуму

DataFrame.mode([axis, numeric_only]) # Вернуться в режим

DataFrame.pct_change([periods, fill_method]) # Возврат к процентному изменению

DataFrame.prod([axis, skipna, level, …]) # Вернуться к продукту

DataFrame.quantile([q, axis, numeric_only]) # Квантиль возврата

DataFrame.rank([axis, method, numeric_only]) # Вернуть сортировку чисел

DataFrame.round([decimals]) #Round a DataFrame to a variable number of decimal places.

DataFrame.sem([axis, skipna, level, ddof]) # Возврат к объективным стандартным ошибкам

DataFrame.skew([axis, skipna, level, …]) # Возврат к беспристрастной асимметрии

DataFrame.sum([axis, skipna, level, …]) #

DataFrame.std([axis, skipna, level, ddof]) # Вернуться к стандартной ошибке

DataFrame.var([axis, skipna, level, ddof]) # Вернуться к объективной ошибке

Из нового индекса и операций выбора и меток

DataFrame.add_prefix(prefix) # Добавить префикс

DataFrame.add_suffix(suffix) # Добавить суффикс

DataFrame.align(other[, join, axis, level]) #Align two object on their axes with the

DataFrame.drop(labels[, axis, level, …]) # Вернуться к удаленному столбцу

DataFrame.drop_duplicates([subset, keep, …]) #Return DataFrame with duplicate rows removed, optionally only

DataFrame.duplicated([subset, keep]) #Return boolean Series denoting duplicate rows, optionally only

DataFrame.equals(other) # Два фрейма данных одинаковы

DataFrame.filter([items, like, regex, axis]) # Отфильтровать определенные подфреймы данных

DataFrame.first(offset) #Convenience method for subsetting initial periods of time series data based on a date offset.

DataFrame.head([n]) # Вернуться к первым n строкам

DataFrame.idxmax([axis, skipna]) #Return index of first occurrence of maximum over requested axis.

DataFrame.idxmin([axis, skipna]) #Return index of first occurrence of minimum over requested axis.

DataFrame.last(offset) #Convenience method for subsetting final periods of time series data based on a date offset.

DataFrame.reindex([index, columns]) #Conform DataFrame to new index with optional filling logic, placing NA/NaN in locations having no value in the previous index.

DataFrame.reindex_axis(labels[, axis, …]) #Conform input object to new index with optional filling logic, placing NA/NaN in locations having no value in the previous index.

DataFrame.reindex_like(other[, method, …]) #Return an object with matching indices to myself.

DataFrame.rename([index, columns]) #Alter axes input function or functions.

DataFrame.rename_axis(mapper[, axis, copy]) #Alter index and / or columns using input function or functions.

DataFrame.reset_index([level, drop, …]) #For DataFrame with multi-level index, return new DataFrame with labeling information in the columns under the index names, defaulting to ‘level_0’, ‘level_1’, etc.

DataFrame.sample([n, frac, replace, …]) # Вернуться к случайной выборке

DataFrame.select(crit[, axis]) #Return data corresponding to axis labels matching criteria

DataFrame.set_index(keys[, drop, append]) #Set the DataFrame index (row labels) using one or more existing columns.

DataFrame.tail([n]) # Вернуться к последним строкам

DataFrame.take(indices[, axis, convert]) #Analogous to ndarray.take

DataFrame.truncate([before, after, axis]) #Truncates a sorted NDFrame before and/or after some particular index value.

Работа с отсутствующими значениями

DataFrame.dropna([axis, how, thresh, …]) #Return object with labels on given axis omitted where alternately any

DataFrame.fillna([value, method, axis, …]) # Заполняем пустые значения

DataFrame.replace([to_replace, value, …]) #Replace values given in ‘to_replace’ with ‘value’.

От новых стереотипов, сортировки и трансформации

DataFrame.pivot([index, columns, values]) #Reshape data (produce a “pivot” table) based on column values.

DataFrame.reorder_levels(order[, axis]) #Rearrange index levels using input order.

DataFrame.sort_values(by[, axis, ascending]) #Sort by the values along either axis

DataFrame.sort_index([axis, level, …]) #Sort object by labels (along an axis)

DataFrame.nlargest(n, columns[, keep]) #Get the rows of a DataFrame sorted by the n largest values of columns.

DataFrame.nsmallest(n, columns[, keep]) #Get the rows of a DataFrame sorted by the n smallest values of columns.

DataFrame.swaplevel([i, j, axis]) #Swap levels i and j in a MultiIndex on a particular axis

DataFrame.stack([level, dropna]) #Pivot a level of the (possibly hierarchical) column labels, returning a DataFrame (or Series in the case of an object with a single level of column labels) having a hierarchical index with a new inner-most level of row labels.

DataFrame.unstack([level, fill_value]) #Pivot a level of the (necessarily hierarchical) index labels, returning a DataFrame having a new level of column labels whose inner-most level consists of the pivoted index labels.

DataFrame.melt([id_vars, value_vars, …]) #“Unpivots” a DataFrame from wide format to long format, optionally

DataFrame.T #Transpose index and columns

DataFrame.to_panel() #Transform long (stacked) format (DataFrame) into wide (3D, Panel) format.

DataFrame.to_xarray() #Return an xarray object from the pandas object.

DataFrame.transpose(*args, **kwargs) #Transpose index and columns

Combining&joining&merging

DataFrame.append(other[, ignore_index, …]) # Добавить данные

DataFrame.assign(**kwargs) #Assign new columns to a DataFrame, returning a new object (a copy) with all the original columns in addition to the new ones.

DataFrame.join(other[, on, how, lsuffix, …]) #Join columns with other DataFrame either on index or on a key column.

DataFrame.merge(right[, how, on, left_on, …]) #Merge DataFrame objects by performing a database-style join operation by columns or indexes.

DataFrame.update(other[, join, overwrite, …]) #Modify DataFrame in place using non-NA values from passed DataFrame.

Последовательно

DataFrame.asfreq(freq[, method, how, …]) # Преобразование временного ряда на определенную частоту

DataFrame.asof(where[, subset]) #The last row without any NaN is taken (or the last row without

DataFrame.shift([periods, freq, axis]) #Shift index by desired number of periods with an optional time freq

DataFrame.first_valid_index() #Return label for first non-NA/null value

DataFrame.last_valid_index() #Return label for last non-NA/null value

DataFrame.resample(rule[, how, axis, …]) #Convenience method for frequency conversion and resampling of time series.

DataFrame.to_period([freq, axis, copy]) #Convert DataFrame from DatetimeIndex to PeriodIndex with desired

DataFrame.to_timestamp([freq, how, axis]) #Cast to DatetimeIndex of timestamps, at beginning of period

DataFrame.tz_convert(tz[, axis, level, copy]) #Convert tz-aware axis to target time zone.

DataFrame.tz_localize(tz[, axis, level, …]) #Localize tz-naive TimeSeries to target time zone.

Рисунок

DataFrame.plot([x, y, kind, ax, ….]) #DataFrame plotting accessor and method

DataFrame.plot.area([x, y]) # Площадь участка

DataFrame.plot.bar([x, y]) # Вертикальная гистограмма

DataFrame.plot.barh([x, y]) # Горизонтальная диаграмма

DataFrame.plot.box([by]) # Boxplot

DataFrame.plot.density(**kwds) #Core DensityКонструкция оценки плотности ядра

DataFrame.plot.hexbin(x, y[, C, …]) #Hexbin plot

DataFrame.plot.hist([by, bins]) #HistogramHistogram

DataFrame.plot.kde(**kwds) #Core DensityКонструкция оценки плотности ядра

DataFrame.plot.line([x, y]) # Линейный график

DataFrame.plot.pie([y]) #Круговая диаграмма

DataFrame.plot.scatter(x, y[, s, c]) # Точечная диаграмма

DataFrame.boxplot([column, by, ax, …]) #Make a box plot from DataFrame column optionally grouped by some columns or

DataFrame.hist(data[, column, by, grid, …]) #Draw histogram of the DataFrame's series using matplotlib / pylab.

Конвертировать в другие форматы

DataFrame.from_csv(path[, header, sep, …]) #Read CSV file (DEPRECATED, please use pandas.read_csv() instead).

DataFrame.from_dict(data[, orient, dtype]) #Construct DataFrame from dict of array-like or dicts

DataFrame.from_items(items[,columns,orient]) #Convert (key, value) pairs to DataFrame.

DataFrame.from_records(data[, index, …]) #Convert structured or record ndarray to DataFrame

DataFrame.info([verbose, buf, max_cols, …]) #Concise summary of a DataFrame.

DataFrame.to_pickle(path[, compression, …]) #Pickle (serialize) object to input file path.

DataFrame.to_csv([path_or_buf, sep, na_rep]) #Write DataFrame to a comma-separated values (csv) file

DataFrame.to_hdf(path_or_buf, key, **kwargs) #Write the contained data to an HDF5 file using HDFStore.

DataFrame.to_sql(name, con[, flavor, …]) #Write records stored in a DataFrame to a SQL database.

DataFrame.to_dict([orient, into]) #Convert DataFrame to dictionary.

DataFrame.to_excel(excel_writer[, …]) #Write DataFrame to an excel sheet

DataFrame.to_json([path_or_buf, orient, …]) #Convert the object to a JSON string.

DataFrame.to_html([buf, columns, col_space]) #Render a DataFrame as an HTML table.

DataFrame.to_feather(fname) #write out the binary feather-format for DataFrames

DataFrame.to_latex([buf, columns, …]) #Render an object to a tabular environment table.

DataFrame.to_stata(fname[, convert_dates, …]) #A class for writing Stata binary dta files from array-like objects

DataFrame.to_msgpack([path_or_buf, encoding]) #msgpack (serialize) object to input file path

DataFrame.to_sparse([fill_value, kind]) #Convert to SparseDataFrame

DataFrame.to_dense() #Return dense representation of NDFrame (as opposed to sparse)

DataFrame.to_string([buf, columns, …]) #Render a DataFrame to a console-friendly tabular output.

DataFrame.to_clipboard([excel, sep]) #Attempt to write text representation of object to the system clipboard This can be pasted into Excel, for example.

Пример

Импортировать пакет

```
# Представляем DataFrame
import pandas as pd
import numpy as np
pandas.DataFrame
date_range () ссылка
dates = pd.date_range('20130101', periods=6)
print("dates:\n",dates)

# np.random.seed (0) фиксированное случайное число
df = pd.DataFrame(np.random.randn(6, 4), index = dates,columns=list('ABCD')) # Индекс здесь - даты выше
print("df:\n",df)
```

```
df:
                   A         B         C         D
2013-01-01 -0.760186 -0.835171 -0.321577 -1.314606
2013-01-02 -1.564905 -0.076478 -1.438768  0.475397
2013-01-03 -0.456697  0.855267  0.568851  0.925598
2013-01-04  1.640094 -0.616764 -0.988467 -0.881574
2013-01-05 -0.092978  0.157447 -0.945336 -1.015510
2013-01-06 -0.214795 -0.816972 -0.390839  0.627917
```

Рис. 6.3. Таблица df

```
df2 = pd.DataFrame({'A': 1.,
                    'B': pd.Timestamp('20130102'),
                    'C': pd.Series(1, index=list(range(4)), dtype='float32'),
                    'D': np.array([3] * 4, dtype='int32'),
                    'E': pd.Categorical(["test", "train", "test", "train"]),
                    'F': 'foo'})
print("df2:\n",df2)
```

```
df2:
     A          B    C  D      E    F
0  1.0 2013-01-02  1.0  3   test  foo
1  1.0 2013-01-02  1.0  3  train  foo
2  1.0 2013-01-02  1.0  3   test  foo
3  1.0 2013-01-02  1.0  3  train  foo
```

Рис. 6.4. Таблица df2

pandas.DataFrame.dtypes
print("df2.dtypes:\n",df2.dtypes)

```
df2.dtypes:
 A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
```

Рис. 6.5. Таблица df2

pandas.DataFrame.head
print("df.head(2):\n",df.head(2))

```
                   A         B         C         D
2013-01-01 -0.736218 -1.791755 -1.117054 -0.226795
2013-01-02 -0.373838 -2.079106 -0.849860  2.530153
```

Рис. 6.6. Таблица df

pandas.DataFrame.tail
print("df.tail(3):\n",df.tail(3))
pandas.DataFrame.index
print("df.index:\n",df.index)

```
df.index:
 DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
                '2013-01-05', '2013-01-06'],
               dtype='datetime64[ns]', freq='D')
```

Рис. 6.7. Таблица df

pandas.DataFrame.to_numpy
print("df.to_numpy():\n",df.to_numpy())

```
df.to_numpy():
 [[ 0.2141096  -1.05533289 -0.10135974  0.09923558]
 [ 0.82939157  0.27569447 -0.60106414 -0.38447009]
 [ 1.83846855  0.13537906 -1.01557383 -1.61325547]
 [ 1.56241501  2.12921685  2.63777735  0.2400643 ]
 [-0.24286062  1.15382638 -1.44499973  0.54341177]
 [-1.41707598 -0.01297177 -0.18833385  0.49202035]]
```

Рис. 6.8. Таблица df

1.3 Операции над данными. Комбинирование данных из разных источников. Обработка пропущенных значений

1 Набор данных диабета индейцев пима

Набор данных диабета индейцев пима предполагает прогнозирование диабета в течение 5 лет у индейцев Пима с учетом медицинских данных.

Это бинарная (2-классная) задача классификации. Количество наблюдений для каждого класса не сбалансировано. Есть 768 наблюдений с 8 входными переменными и 1 выходной переменной. Имена переменных следующие:

- 0. Количество беременных.
- 1. Концентрация глюкозы в плазме в течение 2 часов при оральном тесте на толерантность к глюкозе.
- 2. Диастолическое артериальное давление (мм рт. Ст.).
- 3. Толщина трехглавой кожи (мм).
- 4. 2-часовой сывороточный инсулин (мю Ед / мл).
- 5. Индекс массы тела (вес в кг / (рост в м) ^ 2).
- 6. Диабет родословной.
- 7. Возраст (годы).

- 8. Переменная класса (0 или 1).

Базовая эффективность прогнозирования наиболее распространенного класса - точность классификации приблизительно 65%. Лучшие результаты достигают точности классификации примерно 77%.

Образец первых 5 строк приведен ниже.

6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1

Этот набор данных, как известно, имеет пропущенные значения.

В частности, отсутствуют некоторые наблюдения для некоторых столбцов, которые помечены как нулевое значение.

Мы можем подтвердить это определением этих столбцов и знанием предметной области, что нулевое значение недопустимо для этих мер, например, ноль для индекса массы тела или артериального давления недопустим.

Загрузите набор данных здесь и сохраните его в текущем рабочем каталоге с именем файла *Пим-индусы-diabetes.csv*(Обновить:скачать отсюда).

2 Отметьте отсутствующие значения

В этом разделе мы рассмотрим, как мы можем идентифицировать и пометить значения как пропущенные.

Мы можем использовать графики и сводную статистику, чтобы помочь идентифицировать отсутствующие или поврежденные данные.

Мы можем загрузить набор данных как Pandas DataFrame и распечатать сводную статистику по каждому атрибуту.

```
from pandas import read_csv
dataset = read_csv('pima-indians-diabetes.csv', header=None)
print(dataset.describe())
```

Запуск этого примера приводит к следующему выводу:

```
0           1           2           3           4           5  \
count  768.000000  768.000000  768.000000  768.000000  768.000000  768.000000
mean     3.845052  120.894531   69.105469   20.536458   79.799479   31.992578
std      3.369578   31.972618   19.355807   15.952218  115.244002    7.884160
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      1.000000   99.000000   62.000000    0.000000    0.000000   27.300000
50%      3.000000  117.000000   72.000000   23.000000   30.500000   32.000000
75%      6.000000  140.250000   80.000000   32.000000  127.250000   36.600000
```

max 17.000000 199.000000 122.000000 99.000000 846.000000 67.100000

```
              6          7          8
count  768.000000  768.000000  768.000000
mean     0.471876   33.240885    0.348958
std      0.331329   11.760232    0.476951
min      0.078000   21.000000    0.000000
25%      0.243750   24.000000    0.000000
50%      0.372500   29.000000    0.000000
75%      0.626250   41.000000    1.000000
max      2.420000   81.000000    1.000000
```

Это полезно

Мы можем видеть, что есть столбцы, которые имеют минимальное значение ноль (0). В некоторых столбцах нулевое значение не имеет смысла и указывает на недопустимое или отсутствующее значение.

В частности, следующие столбцы имеют недопустимое нулевое минимальное значение:

- 1: концентрация глюкозы в плазме
- 2: диастолическое артериальное давление
- 3: толщина трицепса
- 4: 2-часовой сывороточный инсулин
- 5: индекс массы тела

Давайте подтвердим это, глядя на необработанные данные, в примере печатаются первые 20 строк данных.

```
from pandas import read_csv
import numpy
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# print the first 20 rows of data
print(dataset.head(20))
```

Запустив пример, мы ясно видим значения 0 в столбцах 2, 3, 4 и 5.

```
0   1   2   3    4    5     6   7  8
0   6  148  72  35    0  33.6  0.627  50  1
1   1   85  66  29    0  26.6  0.351  31  0
2   8  183  64   0    0  23.3  0.672  32  1
3   1   89  66  23   94  28.1  0.167  21  0
4   0  137  40  35  168  43.1  2.288  33  1
5   5  116  74   0    0  25.6  0.201  30  0
6   3   78  50  32   88  31.0  0.248  26  1
7  10  115   0   0    0  35.3  0.134  29  0
8   2  197  70  45  543  30.5  0.158  53  1
9   8  125  96   0    0   0.0  0.232  54  1
10  4  110  92   0    0  37.6  0.191  30  0
11 10  168  74   0    0  38.0  0.537  34  1
12 10  139  80   0    0  27.1  1.441  57  0
```

```
13   1  189  60  23  846  30.1  0.398  59  1
14   5  166  72  19  175  25.8  0.587  51  1
15   7  100   0   0    0  30.0  0.484  32  1
16   0  118  84  47  230  45.8  0.551  31  1
17   7  107  74   0    0  29.6  0.254  31  1
18   1  103  30  38   83  43.3  0.183  33  0
19   1  115  70  30   96  34.6  0.529  32  1
```

Мы можем получить количество пропущенных значений в каждом из этих столбцов. Мы можем сделать это, пометив все значения в подмножестве интересующего нас DataFrame с нулевыми значениями как True. Затем мы можем посчитать количество истинных значений в каждом столбце.

Мы можем сделать это, пометив все значения в подмножестве интересующего нас DataFrame с нулевыми значениями как True. Затем мы можем посчитать количество истинных значений в каждом столбце.

```
from pandas import read_csv
dataset = read_csv('pima-indians-diabetes.csv', header=None)
print((dataset[[1,2,3,4,5]] == 0).sum())
```

При выполнении примера выводится следующий вывод:

```
1 5
2 35
3 227
4 374
5 11
```

Мы видим, что столбцы 1,2 и 5 имеют только несколько нулевых значений, тогда как столбцы 3 и 4 показывают намного больше, почти половину строк.

Это подчеркивает, что для разных столбцов могут потребоваться разные стратегии «пропущенного значения», например, чтобы убедиться, что осталось еще достаточное количество записей для обучения модели прогнозирования.

В Python, в частности Pandas, NumPy и Scikit-Learn, мы отмечаем пропущенные значения как NaN.

Значения со значением NaN игнорируются в таких операциях, как sum, count и т. д.

Мы можем легко пометить значения как NaN с помощью Pandas DataFrame, используя функцию replace ()_на подмножестве интересующих нас столбцов.

После того, как мы отметили пропущенные значения, мы можем использовать функцию isnull () пометить все значения NaN в наборе данных как True и получить счетчик пропущенных значений для каждого столбца.

```
from pandas import read_csv
import numpy
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)
```

```
# count the number of NaN values in each column
print(dataset.isnull().sum())
```

При выполнении примера выводится количество пропущенных значений в каждом столбце. Мы видим, что столбцы 1: 5 имеют то же количество пропущенных значений, что и нулевые значения, указанные выше. Это признак того, что мы правильно отметили идентифицированные пропущенные значения.

Мы можем видеть, что столбцы с 1 по 5 имеют то же количество пропущенных значений, что и нулевые значения, указанные выше. Это признак того, что мы правильно отметили идентифицированные пропущенные значения.

```
0      0
1      5
2     35
3    227
4    374
5     11
6      0
7      0
8      0
```

Это полезное резюме. Мне всегда нравится смотреть на фактические данные, чтобы подтвердить, что я не обманывал себя.

Ниже приведен тот же пример, за исключением того, что мы печатаем первые 20 строк данных.

```
from pandas import read_csv
import numpy
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)
# print the first 20 rows of data
print(dataset.head(20))
```

Запустив пример, мы ясно видим значения NaN в столбцах 2, 3, 4 и 5. В столбце 1 пропущено только 5 значений, поэтому неудивительно, что мы не увидели пример в первых 20 строках.

Из необработанных данных ясно, что маркировка пропущенных значений имела ожидаемый эффект.

```
0      1     2     3      4     5      6   7  8
0    6  148.0  72.0  35.0    NaN  33.6  0.627  50  1
1    1   85.0  66.0  29.0    NaN  26.6  0.351  31  0
2    8  183.0  64.0   NaN    NaN  23.3  0.672  32  1
3    1   89.0  66.0  23.0   94.0  28.1  0.167  21  0
4    0  137.0  40.0  35.0  168.0  43.1  2.288  33  1
5    5  116.0  74.0   NaN    NaN  25.6  0.201  30  0
6    3   78.0  50.0  32.0   88.0  31.0  0.248  26  1
7   10  115.0   NaN   NaN    NaN  35.3  0.134  29  0
```

```
8   2  197.0  70.0  45.0  543.0  30.5  0.158  53  1
9   8  125.0  96.0   NaN    NaN   NaN  0.232  54  1
10  4  110.0  92.0   NaN    NaN  37.6  0.191  30  0
11 10  168.0  74.0   NaN    NaN  38.0  0.537  34  1
12 10  139.0  80.0   NaN    NaN  27.1  1.441  57  0
13  1  189.0  60.0  23.0  846.0  30.1  0.398  59  1
14  5  166.0  72.0  19.0  175.0  25.8  0.587  51  1
15  7  100.0   NaN   NaN    NaN  30.0  0.484  32  1
16  0  118.0  84.0  47.0  230.0  45.8  0.551  31  1
17  7  107.0  74.0   NaN    NaN  29.6  0.254  31  1
18  1  103.0  30.0  38.0   83.0  43.3  0.183  33  0
19  1  115.0  70.0  30.0   96.0  34.6  0.529  32  1
```

Прежде чем мы рассмотрим обработку пропущенных значений, давайте сначала продемонстрируем, что наличие пропущенных значений в наборе данных может вызвать проблемы.

3 Отсутствие значений вызывает проблемы

Отсутствие значений в наборе данных может привести к ошибкам в некоторых алгоритмах машинного обучения.

В этом разделе мы попытаемся оценить алгоритм линейного дискриминантного анализа (LDA) для набора данных с пропущенными значениями.

Это алгоритм, который не работает, если в наборе данных отсутствуют значения.

В приведенном ниже примере отмечены отсутствующие значения в наборе данных, как мы это делали в предыдущем разделе, затем предпринимаются попытки оценить LDA с использованием 3-кратной перекрестной проверки и вывести среднюю точность.

```
from pandas import read_csv
import numpy
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)
# split dataset into inputs and outputs
values = dataset.values
X = values[:,0:8]
y = values[:,8]
# evaluate an LDA model on the dataset using k-fold cross validation
model = LinearDiscriminantAnalysis()
kfold = KFold(n_splits=3, random_state=7)
result = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
print(result.mean())
```

Выполнение примера приводит к ошибке, как показано ниже:

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

Это как мы и ожидаем.

Нам запрещено оценивать алгоритм LDA (и другие алгоритмы) в наборе данных с пропущенными значениями.

Теперь мы можем взглянуть на методы для обработки пропущенных значений.

4 Удалить строки с пропущенными значениями

Самая простая стратегия обработки пропущенных данных - удалить записи, содержащие пропущенное значение.

Мы можем сделать это, создав новый DataFrame Pandas, в котором удалены строки, содержащие пропущенные значения.

Панды обеспечивает функция dropna () – это можно использовать для удаления столбцов или строк с отсутствующими данными. Мы можем использовать dropna (), чтобы удалить все строки с отсутствующими данными, следующим образом:

```
from pandas import read_csv
import numpy
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)
# drop rows with missing values
dataset.dropna(inplace=True)
# summarize the number of rows and columns in the dataset
print(dataset.shape)
```

Запустив этот пример, мы увидим, что количество строк было агрессивно сокращено с 768 в исходном наборе данных до 392, при этом все строки, содержащие NaN, были удалены.

```
(392, 9)
```

Теперь у нас есть набор данных, который мы могли бы использовать для оценки алгоритма, чувствительного к пропущенным значениям, такого как LDA.

```
from pandas import read_csv
import numpy
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)
# drop rows with missing values
dataset.dropna(inplace=True)
# split dataset into inputs and outputs
values = dataset.values
X = values[:,0:8]
```

```
y = values[:,8]
# evaluate an LDA model on the dataset using k-fold cross validation
model = LinearDiscriminantAnalysis()
kfold = KFold(n_splits=3, random_state=7)
result = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
print(result.mean())
```

Пример выполняется успешно и печатает точность модели.

0.78582892934

Удаление строк с пропущенными значениями может быть слишком ограничивающим для некоторых задач прогнозного моделирования, альтернативой является вменение пропущенных значений.

5 Вменять недостающие значения

Вменение относится к использованию модели для замены пропущенных значений.

Есть много вариантов, которые мы могли бы рассмотреть при замене отсутствующего значения, например:

- Постоянное значение, имеющее значение в домене, например 0, отличное от всех других значений.
- Значение из другой случайно выбранной записи.
- Среднее значение, медиана или значение режима для столбца.
- Значение оценивается другой прогнозной моделью.

Любое вменение, выполненное в обучающем наборе данных, должно быть выполнено на новых данных в будущем, когда необходимы прогнозы из окончательной модели. Это необходимо учитывать при выборе способа вменения пропущенных значений.

Например, если вы решили вменять средние значения столбца, эти средние значения столбца необходимо будет сохранить в файле для последующего использования в новых данных, в которых отсутствуют значения.

Панды обеспечивает функция fillna () для замены пропущенных значений конкретным значением.

Например, мы можем использовать fillna (), чтобы заменить отсутствующие значения средним значением для каждого столбца следующим образом:

```
from pandas import read_csv
import numpy
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)
# fill missing values with mean column values
dataset.fillna(dataset.mean(), inplace=True)
# count the number of NaN values in each column
print(dataset.isnull().sum())
```

Выполнение примера обеспечивает подсчет количества пропущенных значений в каждом столбце, показывая ноль пропущенных значений.

```
0    0
```

1 0
2 0
3 0
4 0
5 0
6 0
7 0
8 0

Библиотека Scikit-Learn обеспечивает Imputer () класс предварительной обработки – это может быть использовано для замены отсутствующих значений.

Это гибкий класс, который позволяет вам указать значение для замены (это может быть что-то отличное от NaN) и метод, используемый для его замены (например, среднее значение, медиана или мода). Класс Imputer работает непосредственно с массивом NumPy вместо DataFrame

В приведенном ниже примере класс Imputer используется для замены отсутствующих значений средним значением каждого столбца, а затем выводит число значений NaN в преобразованной матрице.

```
from pandas import read_csv
from sklearn.preprocessing import Imputer
import numpy
dataset = read_csv('pima-indians-diabetes.csv', header=None)
# mark zero values as missing or NaN
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, numpy.NaN)
# fill missing values with mean column values
values = dataset.values
imputer = Imputer()
transformed_values = imputer.fit_transform(values)
# count the number of NaN values in each column
print(numpy.isnan(transformed_values).sum())
```

Выполнение примера показывает, что все значения NaN были вменены успешно.

При выполнении примера печатается точность LDA на преобразованном наборе данных.

0.766927083333

Попробуйте заменить отсутствующие значения другими значениями и посмотрите, сможете ли вы повысить производительность модели.

Возможно, отсутствующие значения имеют значение в данных.

Далее мы рассмотрим использование алгоритмов, которые рассматривают пропущенные значения как просто другое значение при моделировании.

6. Алгоритмы, поддерживающие пропущенные значения

Не все алгоритмы дают сбой, когда отсутствуют данные.

Существуют алгоритмы, которые можно сделать устойчивыми к отсутствующим данным, например, k-Nearest Neighbors, которые могут игнорировать столбец из меры расстояния, если значение отсутствует.

Существуют также алгоритмы, которые могут использовать отсутствующее значение в качестве уникального и другого значения при построении прогнозирующей модели, например деревья классификации и регрессии.

К сожалению, scikit-learn реализации деревьев решений и k-Nearest Neighbours не устойчивы к отсутствующим значениям.Хотя это рассматривается,

Тем не менее, это остается вариантом, если вы планируете использовать другую реализацию алгоритма (например, xgboost) или разработка собственной реализации.

2 Задания для выполнения

Вариант 1

Реализовать клиент-серверное приложение с использованием API GoogleMaps, позволяющее пользователю искать географические данные по введенному названию. Для этого воспользоваться сетевой библиотекой языка Python urllib.

Вариант 2

1. Задайте результаты статистического исследования в виде двумерного представления данных. 2. Определите несколько выборок данных по различным критериям. 3. Определите стандартную описательную статистику, используя методы библиотеки Pandas. 4. Создайте двумерное представление, имеющее не заполненные данные. Проведите обработку массива для получения новой выборки без пропущенных значений.

Вариант 3

Создайте файл excel с колонками: дата и цена. Заполните не менее 100 строк. Далее напишите программу, получающую данные из файла, и организуйте поиск средней цены за каждую неделю.

Вариант 4

Из представленных данных подсчитать, сколько женщин и мужчин выжило, а сколько нет. В этом вам поможет метод *.groupby*. Скачать CSV файл можно тут.

Вариант 5

Посчитать сколько всего женщин и мужчин было в конкретном классе корабля. Скачать CSV файл можно тут.

Вариант 6

Используя цену на акции корпорации Apple за 5 лет по дням, узнать среднюю цену акции (mean) на закрытии (Close): Файл с данными можно скачать тут.

Вариант 7

Создать Dataframe, заполнить повторяющимися данными. Далее необходимо найти и удалить все повторения

Вариант 8

В файле excel заполнить колонки целочисленными данными, далее необходимо написать программу, которая будет суммировать несколько столбцов и добавить итоговый.

Лабораторная работа № 7. Визуализация с помощью библиотеки Matplotlib

1 Методические рекомендации

1.1 Библиотеки Python для визуализации данных: Matplotlib, Seaborn, Plotly

Интерактивность

Хотите ли вы, чтобы ваша визуализация была интерактивной?

Визуализация в некоторых библиотеках, таких как Matplotlib, является простым статичным изображением, что хорошо подходит для объяснения концепций (в документе, на слайдах или в презентации).

Другие библиотеки, такие как Altair, Bokeh и Plotly, позволяют создавать интерактивные графики, которые пользователи могут изучать, взаимодействуя с ними.

Синтаксис и гибкость

Чем отличается синтаксис каждой библиотеки? Библиотеки низкого уровня, такие как Matplotlib, позволяют делать все, что вы захотите, но за счет более сложного API. Некоторые библиотеки, такие как Altair, очень *декларативны*, что упрощает построение графиков по вашим данным.

Тип данных и визуализации

Приходилось ли вам сталкиваться в работе с нестандартными юзкейсами, например, с географическим графиком, включающим большой набор данных или с типом графика, который поддерживается только определенной библиотекой?

Данные

Чтобы было проще сравнивать библиотеки, здесь представлены реальные данные с Github из этой статьи:

I Scraped more than 1k Top Machine Learning Github Profiles and this is what I Found

user_name	name	type_user	html_url	bio	company	email	followers	following	hireable
josephmisiti	Joseph Misiti	Owner	https://gith...	Mathemati...	Math & Pen...	None	2495	273	True
wepe	wepon	Owner	https://gith...	None	AntFin	wepon@pk...	4508	47	
ZuzooVn	Nam Vu	Owner	https://gith...	A Vietname...	None	zuzoovn@g...	1192	91	True
rasbt	Sebastian ...	Owner	https://gith...	Machine Le...	UW-Madison	mail@seba...	12725	33	
lazyprogra...	LazyProgra...	Owner	https://gith...	https://dee...	None	None	2809	0	
lawlite19	lawlite	Owner	https://gith...	It's really ni...	Southeast ...	lawlitewan...	616	48	True
Jack-Cherish	Jack Cui	Owner	https://gith...	:octocat:公...	Northeaste...	c41118400...	2902	28	
ujjwalkarn	Ujjwal Karn	Owner	https://gith...	None	None	None	1952	224	
trekhleb	Oleksii Trek...	Owner	https://gith...	Software E...	Uber	None	4934	7	True
Vay-keen	Wei Ao	Owner	https://gith...	Talk is chea...	Shenzhen ...	aowei2016...	192	0	

Рис. 7.1. Данные таблицы Github

В статью включены визуализации из каждой библиотеки с помощью Datapane, который представляет собой Python фреймворк и API для публикации и совместного использования Python-отчетов. Больше реальных примеров вы можете найти в пользовательских отчетах в галерее Datapane.

```
import datapane as dp
dp.Blob.get(name='github_data', owner='khuyentran1401').download_df()
```

Не забудьте залогиниться со своим токеном авторизации в Datapane, если вы хотите использовать Blob. Это займет менее минуты.

Matplotlib

Matplotlib, вероятно, является самой популярной библиотекой Python для визуализации данных. Все, кто интересуется data science, наверняка хоть раз сталкивались с Matplotlib.

Плюсы

1. Четко отображены свойства данных

При анализе данных возможность быстро посмотреть распределение может быть очень полезной.

Например, если я хочу быстро посмотреть распределение топ 100 пользователей с наибольшим количеством подписчиков, обычно Matplotlib мне будет вполне достаточно:

```
import matplotlib.pyplot as plt

top_followers = new_profile.sort_values(by='followers', axis=0, ascending=False)[:100]

fig = plt.figure()

plt.bar(top_followers.user_name,
    top_followers.followers)
```

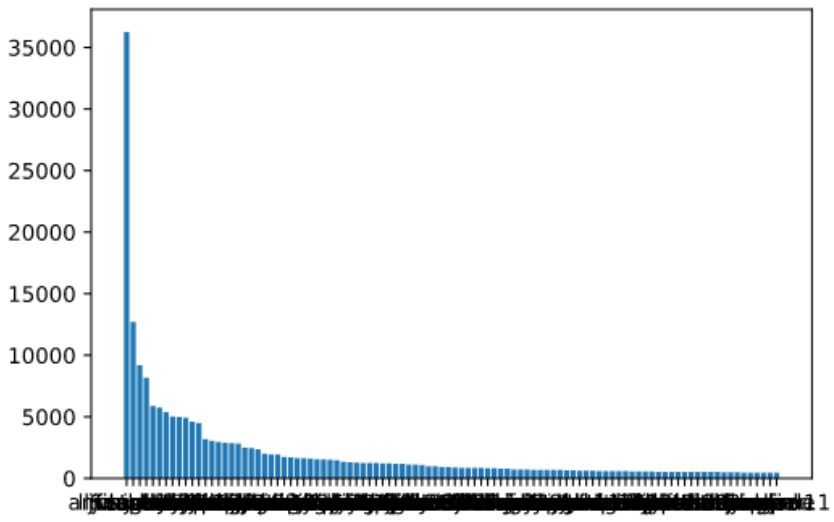


Рис. 7.2. Диаграмма таблицы Github

Даже что-то вроде этого:

```
fig = plt.figure()

plt.text(0.6, 0.7, "learning", size=40, rotation=20.,
```

```
        ha="center", va="center",
        bbox=dict(boxstyle="round",
                ec=(1., 0.5, 0.5),
                fc=(1., 0.8, 0.8),
                )
        )

    plt.text(0.55, 0.6, "machine", size=40, rotation=-25.,
        ha="right", va="top",
        bbox=dict(boxstyle="square",
                ec=(1., 0.5, 0.5),
                fc=(1., 0.8, 0.8),
                )
        )

    plt.show()
```

Минусы

Matplotlib может создать любой график, но с его помощью может быть сложно построить или подогнать сложные графики, чтобы они выглядели презентабельно.

Несмотря на то, что график достаточно хорошо подходит для визуализации распределений, если вы хотите презентовать его публике, вам нужно будет откорректировать оси X и Y, что потребует больших усилий, потому что Matplotlib имеет чрезвычайно низкоуровневый интерфейс.

```
    correlation = new_profile.corr()

    fig, ax = plt.subplots()
    im = plt.imshow(correlation)

    ax.set_xticklabels(correlation.columns)
    ax.set_yticklabels(correlation.columns)

    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
        rotation_mode="anchor")
```

Вывод: с помощью Matplotlib можно создать что угодно, но для сложных графиков может потребоваться гораздо больше кода, чем другим библиотекам.

Seaborn

Seaborn - это библиотека Python для визуализации данных, построенная на базе Matplotlib. Она более высокоуровневая, что упрощает ее использование.

Плюсы

1 Меньше кода

Предоставляет интерфейс более высокого уровня для построения похожих графиков. Другими словами, seaborn обычно строит графики, аналогичные matplotlib, но с меньшим количеством кода и более красивым дизайном.

Мы используем те же данные, что и раньше, чтобы построить аналогичный график пользовательской активности.

```
correlation = new_profile.corr()

sns.heatmap(correlation, annot=True)
```

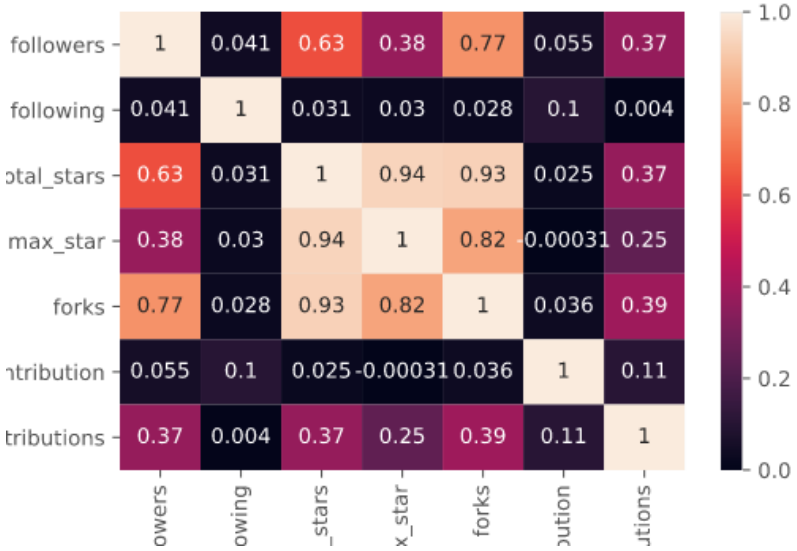


Рис. 7.3. График пользовательской активности

Мы получаем лучший график пользовательской активности без возни x и y!

2 Делает стандартные графики красивее

Многие люди выбирают seaborn для создания широко используемых графиков, таких как столбчатые и прямоугольные диаграммы, расчетные графики, гистограммы и т. д., но не только потому, что это потребует меньше кода, они еще и визуально приятнее. Как видно на примере выше, цвета выглядят лучше, чем цвета по умолчанию в Matplotlib.

```
sns.set(style="darkgrid")
titanic = sns.load_dataset("titanic")
ax = sns.countplot(x="class", data=titanic)
```

Минусы

Seaborn более ограничен и не имеет такой широкой коллекции графиков, как matplotlib.

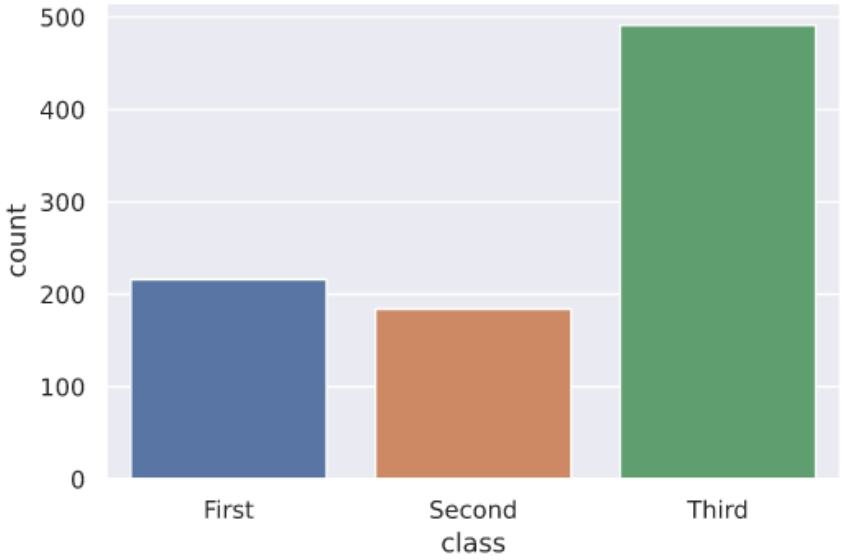


Рис. 7.4. Диаграмма Seaborn

Вывод: Seaborn − это версия Matplotlib более высокого уровня. Несмотря на то, что коллекция графиков не настолько большая, как в Matplotlib, созданные с помощью seaborn широко используемые графики (например, столбчатая диаграмма, прямоугольная диаграмма, график пользовательской активности и т. д.), при меньшем количестве кода будет выглядеть визуально приятнее.

Plotly

Python библиотека Plotly упрощает создание интерактивных графиков типографского качества. Он также может создавать диаграммы, аналогичные Matplotlib и seaborn, такие как линейные графики, точечные диаграммы, диаграммы с областями, столбчатые диаграммы и т. д.

Плюсы

1 Похож на R

Если вы поклонник графиков в R и вам не хватает его функционала при переходе на Python, Plotly даст вам такое же качество графиков с использованием Python!

Мой любимая версия - Plotly Express, потому что с ней можно легко и быстро создавать отличные графики одной строчкой в Python.

```
import plotly.express as px

fig = px.scatter(new_profile[:100],
        x='followers',
        y='total_stars',
        color='forks',
        size='contribution')
fig.show()
```

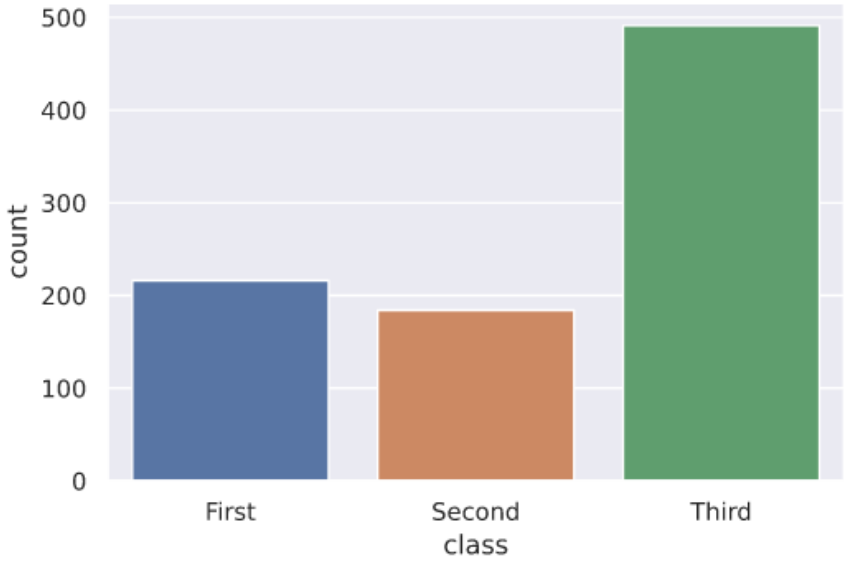


Рис. 7.5. Диаграмма – **Plotly Express**

2 Простота создания интерактивных графиков

Plotly также упрощает создание интерактивных графиков. Интерактивные графики не только красиво выглядят, но и позволяют публике более внимательно изучить каждую точку на графике.

Помните столбчатую диаграмму, которую мы показывали ранее в matplotlib? Давайте посмотрим, как она получится с помощью Plotly

```
import plotly.express as px

top_followers = new_profile.sort_values(by='followers', axis=0, ascending=False)[:100]

fig = px.bar(top_followers,
             x='user_name',
             y='followers',
            )

fig.show()
```

Примерно за столько же строк кода мы создали интерактивный график, на котором можно навести указатель мыши на каждый столбец, чтобы увидеть, кому он принадлежит и сколько подписчиков у этого пользователя. Это означает, что пользователь вашей визуализации может изучить ее самостоятельно.

3 Легко делать сложные графики

С помощью Plotly достаточно легко создавать сложные графики.

Например, если мы хотим создать карту для визуализации местоположения пользователей GitHub, мы можем найти широту и долготу их расположения как показано здесь, а затем использовать эти данные чтобы отметить местоположение пользователей уже на карте:

```
import plotly.express as px
import datapane as dp

location_df = dp.Blob.get(name='location_df', owner='khuyen-
tran1401').download_df()

m = px.scatter_geo(location_df, lat='latitude', lon='longitude',
                   color='total_stars', size='forks',
                   hover_data=['user_name','followers'],
                   title='Locations of Top Users')

m.show()
```

И, написав всего несколько строк кода, местоположения всех пользователей красиво представлены на карте. Цвет окружностей представляет количество форков, а размер - общее количество звезд.

Вывод: Plotly отлично подходит для создания интерактивных и качественных графиков при помощи всего нескольких строк кода.

Altair

Altair – это библиотека Python декларативной статистической визуализации, которая основана на vega-lite, что идеально подходит для графиков, требующих большого количества статистических преобразований.

Плюсы

1 Простая грамматика визуализации

Грамматика, используемая для визуализации, невероятно проста для понимания. Необходимо только обозначить связи между столбцами данных и каналами их преобразования, а остальная часть построения графиков обрабатывается автоматически. Это звучит довольно абстрактно, но имеет решающее значение, когда вы работаете с данными, и делает визуализацию информации очень быстрой и интуитивно понятной.

Например, для данных о Титанике выше мы хотели бы подсчитать количество людей в каждом классе. Все, что нам нужно, это использовать count() в y_axis

```
import seaborn as sns
import altair as alt

titanic = sns.load_dataset("titanic")

alt.Chart(titanic).mark_bar().encode(
   alt.X('class'),
   y='count()'
)
```

2 Простота преобразования данных

Altair также упрощает преобразование данных при создании диаграммы.

Например, мы хотим определить средний возраст каждого пола на Титанике и вместо того, чтобы выполнять преобразование заранее, как в Plotly, в Altair есть возможность выполнить преобразование в коде, описывающем диаграмму.

```
hireable = alt.Chart(titanic).mark_bar().encode(
    x='sex:N',
    y='mean_age:Q'
).transform_aggregate(
    mean_age='mean(age)',
    groupby=['sex'])

hireable
```

Логика здесь состоит в том, чтобы использовать transform_aggregate() для взятия среднего значения возраста (mean(age)) каждого пола (groupby=['sex']) и сохранить его в переменной mean_age). За ось Y мы берем переменную.

Мы также можем убедиться, что класс - это номинальные данные (категорийные данные в произвольном порядке), используя :N, или что mean_age - это количественные данные (меры значений, такие как числа), используя :Q.

Полный список преобразований данных можно найти здесь.

3 Связывание нескольких графиков

Altair также позволяет создавать впечатляющие связи между графиками, например, с возможностью использовать выбор интервала для фильтрации содержимого прикрепленной гистограммы.

Например, мы хотим визуализировать количество людей из каждого класса в пределах значений, ограниченных выделенным интервалом в точечной диаграмме по возрасту и плате за проезд. Тогда нам нужно написать что-то вроде этого:

```
brush = alt.selection(type='interval')

points = alt.Chart(titanic).mark_point().encode(
    x='age:Q',
    y='fare:Q',
    color=alt.condition(brush, 'class:N', alt.value('lightgray'))
).add_selection(
    brush
)

bars = alt.Chart(titanic).mark_bar().encode(
    y='class:N',
    color='class:N',
    x = 'count(class):Q'
).transform_filter(
    brush
```

)

points & bars

Когда мы перетаскиваем мышь, чтобы выбрать интервал на корреляционной диаграмме, мы можем наблюдать изменения на гистограмме ниже. В сочетании с преобразованиями и вычислениями, сделанными ранее, это означает, что вы можете создавать несколько чрезвычайно интерактивных графиков, которые выполняют вычисления на лету - даже не требуя работающего сервера Python!

Минусы

Если вы не задаете пользовательский стиль, простые диаграммы, такие как, например, столбчатые, не будут оформлены стилистически так же хорошо, как в seaborn или Plotly. Altair также не рекомендует использовать наборы данных с более чем 5000 экземплярами и рекомендует вместо этого агрегировать данные перед визуализацией.

Вывод: Altair идеально подходит для создания сложных графиков для отображения статистики. Altair не может обрабатывать данные, превышающие 5000 экземпляров, и некоторые простые диаграммы в нем уступают по стилю Plotly или Seaborn.

Bokeh

Bokeh – это интерактивная библиотека для визуализации, предназначенная для презентации данных в браузерах.

Плюсы

1 Интерактивная версия Matplotlib

Если мы будем будем составлять топы интерактивных библиотек для визуализации, Bokeh, вероятно, займет первое место в категории сходства с Matplotlib.

Matplotlib позволяет создать любой график, так как эта библиотека предназначена для визуализации на достаточно низком уровне. Bokeh можно использовать как с высокоуровневым, так и низкоуровневым интерфейсом; таким образом, она способна создавать множество сложных графиков, которые создает Matplotlib, но с меньшим количеством строк кода и более высоким разрешением.

Например, круговой график Matplotlib,

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

x = [1, 2, 3, 4, 5]
y = [2, 5, 8, 2, 7]

for x,y in zip(x,y):
    ax.add_patch(plt.Circle((x, y), 0.5, edgecolor = "#f03b20",facecolor='#9ebcda', alpha=0.8))
```

```
#Use adjustable='box-forced' to make the plot area square-shaped as well.
ax.set_aspect('equal', adjustable='datalim')
ax.set_xbound(3, 4)

ax.plot()   #Causes an autoscale update.
plt.show()
```

который, в Bokeh, может быть создан с лучшим разрешением и функциональностью:

```
from bokeh.io import output_file, show
from bokeh.models import Circle
from bokeh.plotting import figure

reset_output()
output_notebook()

plot = figure(plot_width=400, plot_height=400, tools="tap", title="Select a circle")
renderer = plot.circle([1, 2, 3, 4, 5], [2, 5, 8, 2, 7], size=50)

selected_circle = Circle(fill_alpha=1, fill_color="firebrick", line_color=None)
nonselected_circle = Circle(fill_alpha=0.2, fill_color="blue", line_color="firebrick")

renderer.selection_glyph = selected_circle
renderer.nonselection_glyph = nonselected_circle

show(plot)
```

2 Связь между графиками

В Bokeh также можно достаточно просто связывать графики. Изменение, примененное к одному графику, будет применено к другому графику с этой же переменной.

Например, если мы создаем 3 графика рядом и хотим наблюдать их взаимосвязь, мы можем связанное закрашивание

```
from bokeh.layouts import gridplot, row
from bokeh.models import ColumnDataSource

reset_output()
output_notebook()

source = ColumnDataSource(new_profile)

TOOLS = "box_select,lasso_select,help"
TOOLTIPS = [('user', '@user_name'),
        ('followers', '@followers'),
```

```
            ('following', '@following'),
            ('forks', '@forks'),
            ('contribution', '@contribution')]

    s1 = figure(tooltips=TOOLTIPS, plot_width=300, plot_height=300, title=None, tools=TOOLS)
    s1.circle(x='followers', y='following', source=source)

    s2 = figure(tooltips=TOOLTIPS, plot_width=300, plot_height=300, title=None, tools=TOOLS)
    s2.circle(x='followers', y='forks', source=source)

    s3 = figure(tooltips=TOOLTIPS, plot_width=300, plot_height=300, title=None, tools=TOOLS)
    s3.circle(x='followers', y='contribution', source=source)

    p = gridplot([[s1,s2,s3]])
    show(p)
```

Минусы

Поскольку Bokeh – это библиотека, которая имеет интерфейс среднего уровня, она часто требует меньше кода, чем Matplotlib, но требует больше кода для создания того же графика, чем Seaborn, Altair или Plotly.

Например, для создания такого же расчетного графика с данными с Титаника, помимо преобразования данных заранее, мы также должны установить ширину столбца и цвет если мы хотим, чтобы график выглядел красиво.

Если мы не добавим ширину столбцов графика, то он будет выглядеть так:

```
    from bokeh.transform import factor_cmap
    from bokeh.palettes import Spectral6

    p = figure(x_range=list(titanic_groupby['class']))
    p.vbar(x='class', top='survived', source = titanic_groupby,
        fill_color=factor_cmap('class', palette=Spectral6, factors=list(titanic_groupby['class'])
        ))
    show(p)
```

Таким образом, нам нужно вручную настраивать параметры, чтобы сделать график более красивым:

```
    from bokeh.transform import factor_cmap
    from bokeh.palettes import Spectral6

    p = figure(x_range=list(titanic_groupby['class']))
    p.vbar(x='class', top='survived', width=0.9, source = titanic_groupby,
```

```
        fill_color=factor_cmap('class', palette=Spectral6, factors=list(ti-
tanic_groupby['class'])
        ))
    show(p)
```

Если вы хотите создать красивую столбчатую диаграмму, используя меньшее количеством кода, то для вас это может быть недостатком Bokeh по сравнению с другими библиотеками

Вывод: Bokeh - единственная библиотека, чей интерфейс варьируется от низкого до высокого, что позволяет легко создавать как универсальные, так и сложные графики. Однако цена этого заключается в том, что для создания графиков с качеством, аналогичным другим библиотекам, обычно требуется больше кода.

Folium

Folium позволяет легко визуализировать данные на интерактивной встраиваемой карте. В библиотеке есть несколько встроенных тайлсетов из OpenStreetMap, Mapbox и Stamen

Плюсы

1 Очень легко создавать карты с маркерами

Несмотря на то, что Plotly, Altair и Bokeh также позволяют нам создавать карты, Folium использует открытую уличную карту, что-то близкое к Google Map, с помощью минимального количества кода

Помните, как мы создавали карту для визуализации местоположения пользователей Github с помощью Plotly? Мы могли бы сделать карту еще лучше с помощью Folium:

```
    import folium

    # Load data
    location_df = dp.Blob.get(name='location_df', owner='khuyen-
tran1401').download_df()

    # Save latitudes, longitudes, and locations' names in a list
    lats = location_df['latitude']
    lons = location_df['longitude']
    names = location_df['location']

    # Create a map with an initial location
    m = folium.Map(location=[lats[0], lons[0]])
    for lat, lon, name in zip(lats, lons, names):

       # Create marker with other locations
       folium.Marker(location=[lat, lon],
                     popup= name,
                    icon=folium.Icon(color='green')
    ).add_to(m)
```

m

«Живой» вариант карты можно посмотреть в оригинале: https://towardsdatascience.com/top-6-python-libraries-for-visualization-which-one-to-use-fe43381cd658

2 Добавление потенциального местоположения

Если мы хотим добавить потенциальные местоположения других пользователей, Folium упрощает это, позволяя пользователям добавлять маркеры:

```
# Code to generate map here
#....

# Enable adding more locations in the map
m = m.add_child(folium.ClickForMarker(popup='Potential Location'))
```

«Живой» вариант карты можно посмотреть в оригинале: https://towardsdatascience.com/top-6-python-libraries-for-visualization-which-one-to-use-fe43381cd658

Кликните на карту, чтобы увидеть новое местоположение, созданное прямо там, где вы кликнули.

3 Плагины

У Folium есть ряд плагинов, которые вы можете использовать со своей картой, в том числе плагин для Altair. Что, если мы хотим увидеть карту пользовательской активности общего количества звездных пользователей Github в мире, чтобы определить, где находится большое количество пользователей Github с большим количеством звезд? Карта пользовательской активности в плагинах Folium позволяет вам это сделать:

```
from folium.plugins import HeatMap
m = folium.Map(location=[lats[0], lons[0]])
HeatMap(data=location_df[['latitude', 'longitude', 'total_stars']]).add_to(m)
```

«Живой» вариант карты можно посмотреть в оригинале: https://towardsdatascience.com/top-6-python-libraries-for-visualization-which-one-to-use-fe43381cd658

Уменьшите масштаб, чтобы увидеть полное отображение пользовательской активности на карте.

Вывод: Folium позволяет создавать интерактивную карту в несколько строк кода. Он дает вам ощущения близкие к использованию Google Map.

1.2 Виды графиков и функции для их построения с помощью Matplotlib

В прошлых материалах вы встречали примеры, демонстрирующие архитектуру **библиотеки matplotlib**. После знакомства с основными графическими элементами для графиков время рассмотреть примеры разных типов

графиков, начиная с самых распространенных, таких как линейные графики, гистограммы и круговые диаграммы, и заканчивая более сложными, но все равно часто используемыми.

Поскольку визуализация − основная цель библиотеки, то этот раздел является очень важным. Умение выбрать правильный тип графика является фундаментальным навыком, ведь неправильная репрезентация может привести к тому, что данные, полученные в результате качественного анализа данных, будет интерпретированы неверно.

Для выполнения кода импортируйте pyplot и numpy

```
import matplotlib.pyplot as plt
import numpy as np
```

Линейные графики

Линейные графики являются самыми простыми из всех. Такой график − это последовательность точек данных на линии. Каждая точка состоит из пары значений (x, y), которые перенесены на график в соответствии с масштабами осей (x и y).

В качестве примера можно вывести точки, сгенерированные математической функцией. Возьмем такую: y = sin (3 * x) / x

Таким образом для создания последовательности точек данных нужно создать два массива NumPy. Сначала создадим массив со значениями x для оси x. Для определения последовательности увеличивающихся значений используем функцию np.arrange(). Поскольку функция синусоидальная, то значениями должны быть числа кратные π (np.pi). Затем с помощью этой последовательности можно получить значения y, применив для них функцию np.sin() (и все благодаря NumPy).

После этого остается лишь вывести все точки на график с помощью функции plot(). Результатом будет линейный график.

```
x = np.arange(-2*np.pi,2*np.pi,0.01)
y = np.sin(3*x)/x
plt.plot(x,y)
plt.show()
```

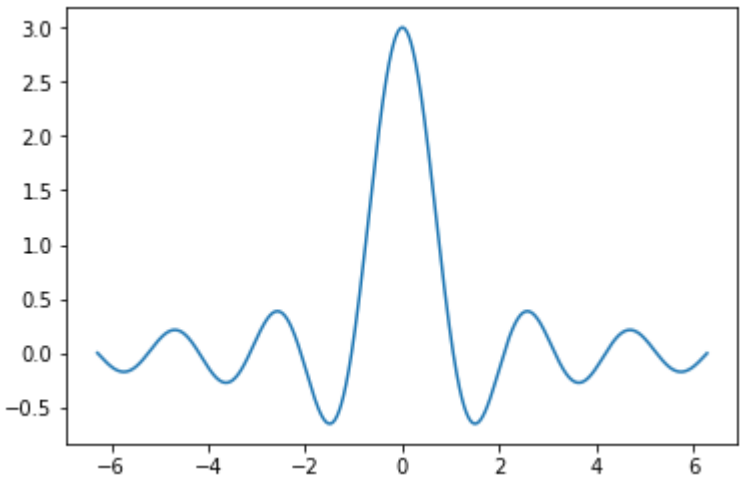


Рис. 7.6. Линейный график

Этот пример можно расширить для демонстрации семейства функций, например, такого (с разными значениями n):

```
x = np.arange(-2*np.pi,2*np.pi,0.01)
y = np.sin(3*x)/x
y2 = np.sin(2*x)/x
y3 = np.sin(x)/x
plt.plot(x,y)
plt.plot(x,y2)
plt.plot(x,y3)
plt.show()
```

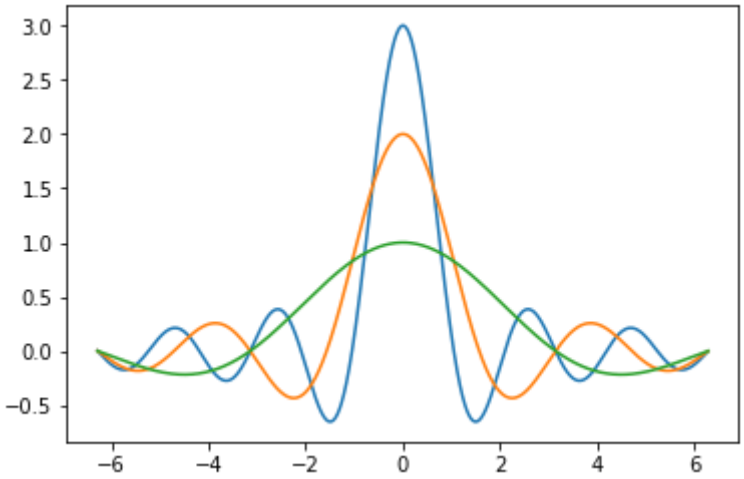


Рис. 7.7. Линейный график

Как можно увидеть на изображении, каждой линии автоматически присваивается свой цвет. При этом все графики представлены в одном масштабе. Это значит, что точки данных связаны с одними и теми же осями x и y. Вот почему каждый вызов функции plot() учитывает предыдущие вызовы, так что объект Figure применяет изменения с учетом прошлых команд еще до вывода (для вывода используется show()).

```
x = np.arange(-2*np.pi,2*np.pi,0.01)
y = np.sin(3*x)/x
y2 = np.sin(2*x)/x
y3 = np.sin(x)/x
plt.plot(x,y,'k--',linewidth=3)
plt.plot(x,y2,'m-.')
plt.plot(x,y3,color='#87a3cc',linestyle='--')
plt.show()
```

Как уже говорилось в прошлых в разделах, вне зависимости от настроек по умолчанию можно выбрать тип начертания, цвет и так далее.

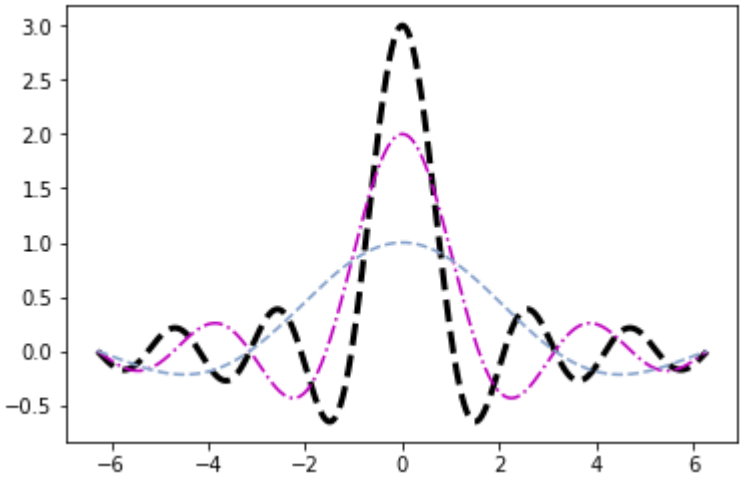


Рис. 7.8. Линейный график

Третьим аргументом функции plot() можно указать коды цветов, типы линий и все этой в одной строке. Также можно использовать два именованных аргумента отдельно: color − для цвета и linestyle − для типа линии.

Код	**Цвет**
b	голубой
g	зеленый
r	красный
c	сине-зеленый
m	пурпурный
y	желтый
k	черный
w	белый

На графике определен диапазон от − 2π до 2π на оси x, но по умолчанию деления обозначены в числовой форме. Поэтому их нужно заменить на множители числа π. Также можно поменять делители на оси y. Для этого используются функции xticks() и yticks(). Им нужно передать список значений. Первый список содержит значения, соответствующие позициям, где деления будут находиться, а второй − их метки. В этом случае будут использоваться LaTeX-выражения, что нужно для корректного отображения π. Важно не забыть добавить знаки $ в начале и конце, а также символ r в качестве префикса.

```
x = np.arange(-2*np.pi,2*np.pi,0.01)
y = np.sin(3*x)/x
y2 = np.sin(2*x)/x
y3 = np.sin(x)/x
plt.plot(x,y,color='b')
plt.plot(x,y2,color='r')
plt.plot(x,y3,color='g')
plt.xticks([-2*np.pi,-np.pi,0, np.pi, 2*np.pi],
```

```
        [r'$-2\pi$',r'$-\pi$',r'$0$',r'$+\pi$',r'$+2\pi$'])
plt.yticks([-1,0,1,2,3],
        [r'$-1$',r'$0$',r'$+1$',r'$+2$',r'$+3$'])
plt.show()
```

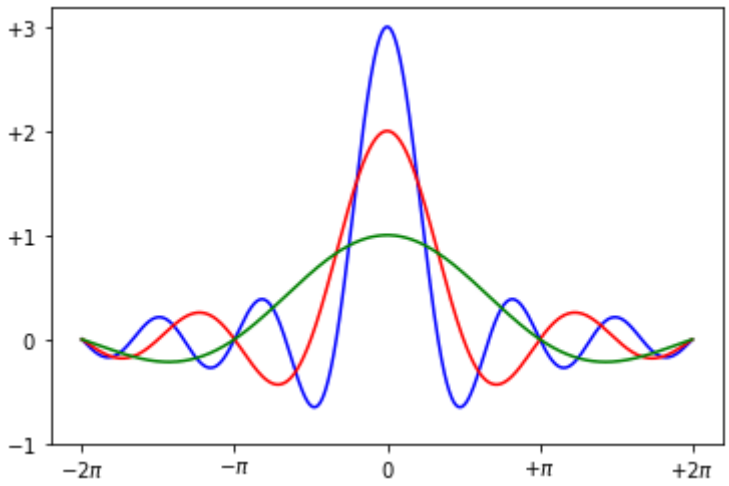


Рис. 7.9. Линейный график

Пока что на всех рассмотренных графиках оси x и y изображались на краях объекта Figure (по границе рамки). Но их же можно провести так, чтобы они пересекались − то есть, получит декартову система координат.

Для этого нужно сперва получить объект Axes с помощью функцию gca. Затем с его помощью можно выбрать любую из четырех сторон, создав область с границами и определив положение каждой: справа, слева, сверху и снизу. Ненужные части обрезаются (справа и снизу), а с помощью функции set_color() задается значение none. Затем стороны, которые соответствуют осям x и y, проходят через начало координат (0, 0) с помощью функции set_position().

```
x = np.arange(-2*np.pi,2*np.pi,0.01)
y = np.sin(3*x)/x
y2 = np.sin(2*x)/x
y3 = np.sin(x)/x
plt.plot(x,y,color='b')
plt.plot(x,y2,color='r')
plt.plot(x,y3,color='g')
plt.xticks([-2*np.pi,-np.pi,0, np.pi, 2*np.pi],
        [r'$-2\pi$',r'$-\pi$',r'$0$',r'$+\pi$',r'$+2\pi$'])
plt.yticks([-1,0,1,2,3],
        [r'$-1$',r'$0$',r'$+1$',r'$+2$',r'$+3$'])
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
```

```
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
plt.show()
```

Теперь график будет состоять из двух пересекающихся в центре осей, который представляет собой начало декартовой системы координат.

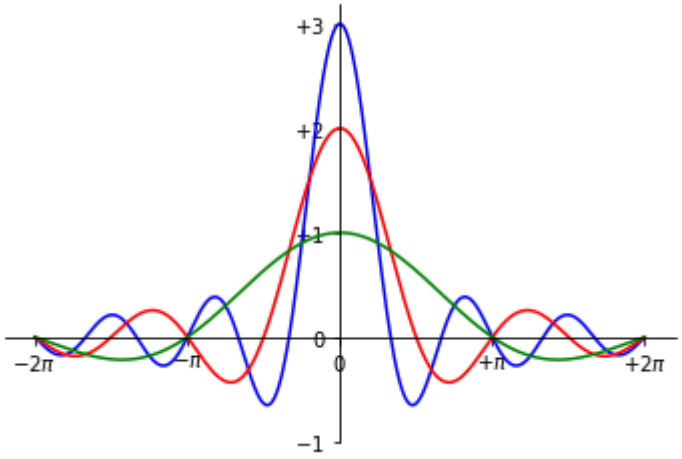


Рис. 7.10. Линейный график

Также есть возможность указать на определенную точку с помощью дополнительных обозначений и стрелки. Обозначением может выступать LaTeX-выражение, например, формула предела функции sinx/x, стремящейся к 0.

Для этого в matplotlib есть функция annotate(). Ее настройка кажется сложной, но большое количество kwargs обеспечивает требуемый результат. Первый аргумент − строка, представляющая собой LaTeX-выражение, а все остальные − опциональные. Точка, которую нужно отметить на графике представлена в виде списка, включающего ее координаты (x и y), переданные в аргумент xy. Расстояние заметки до точки определено в xytext, а стрелка − с помощью arrowprops.

```
x = np.arange(-2*np.pi,2*np.pi,0.01)
y = np.sin(3*x)/x
y2 = np.sin(2*x)/x
y3 = np.sin(x)/x
plt.plot(x,y,color='b')
plt.plot(x,y2,color='r')
plt.plot(x,y3,color='g')
plt.xticks([-2*np.pi,-np.pi,0, np.pi, 2*np.pi],
        [r'$-2\pi$',r'$-\pi$',r'$0$',r'$+\pi$',r'$+2\pi$'])
plt.yticks([-1,0,1,2,3],
        [r'$-1$',r'$0$',r'$+1$',r'$+2$',r'$+3$'])
plt.annotate(r'$\lim_{x\to 0}\frac{\sin(x)}{x}= 1$', xy=[0,1],xy-
coords='data',
```

```
            xytext=[30,30],fontsize=16, textcoords='offset points', ar-
rowprops=dict(arrowstyle="->",
            connectionstyle="arc3,rad=.2"))
    ax = plt.gca()
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    ax.xaxis.set_ticks_position('bottom')
    ax.spines['bottom'].set_position(('data',0))
    ax.yaxis.set_ticks_position('left')
    ax.spines['left'].set_position(('data',0))
    plt.show()
```

В итоге этот код сгенерирует график с математической формулой предела, представленной точкой, на которую указывает стрелка.

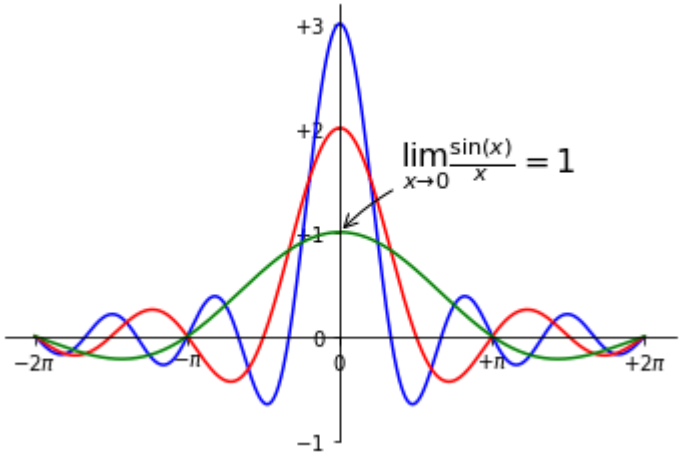


Рис. 7.11. Линейный график

Линейные графики с pandas

Рассмотрим более практический и приближенный к анализу данных пример. С ним будет видно, насколько просто использовать библиотеку matplotlib для объектов Dataframe из библиотеки pandas. Визуализация данных в виде линейного графика − максимально простая задача. Достаточно передать объект в качестве аргумента функции plot() для получения графика с несколькими линиями.

```
    import pandas as pd

    data = {'series1':[1,3,4,3,5],
            'series2':[2,4,5,2,4],
            'series3':[3,2,3,1,3]}
    df = pd.DataFrame(data)
    x = np.arange(5)
    plt.axis([0,5,0,7])
    plt.plot(x,df)
    plt.legend(data, loc=2)
    plt.show()
```

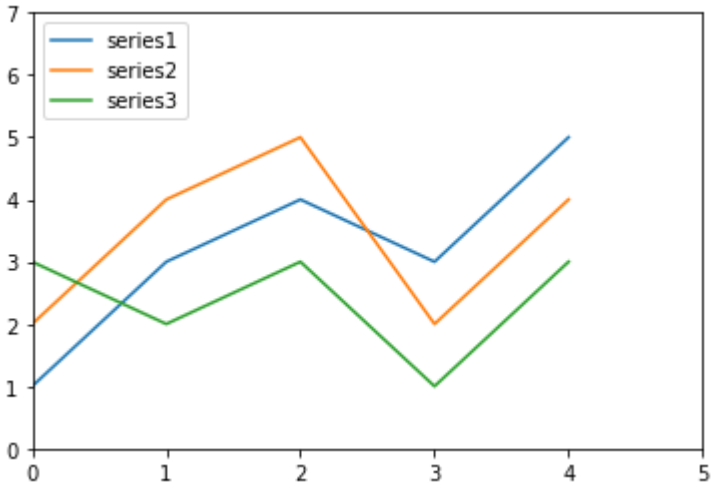


Рис. 7.12. Линейный график

Гистограммы

Гистограмма состоит из примыкающих прямоугольников, расположенных вдоль оси x, которые разбиты на дискретные интервалы, их называют bins. Их площадь пропорциональна частоте конкретного интервала. Такой способ визуализации часто используют в статистике для демонстрации распределения.

Для представления гистограммы в pyplot есть функция hist(). У нее также есть особенности, которых не найти у других функций, отвечающих за создание графиков. hist() не только рисует гистограмму, но также возвращает кортеж значений, представляющих собой результат вычислений гистограммы. Функция hist() может реализовывать вычисление гистограммы, чего достаточно для предоставления набора значений и количества интервалов, на которых их нужно разбить. Наконец hist() отвечает за разделение интервала на множество и вычисление частоты каждого. Результат этой операции не только выводится в графической форме, но и возвращается в виде кортежа.

Для понимания операции лучше всего воспользоваться практическим примером. Сгенерируем набор из 100 случайных чисел от 0 до 100 с помощью random.randint().

```
pop = np.random.randint(0,100,100)
pop
array([33, 90, 10, 68, 18, 67,  6, 54, 32, 25, 90,  6, 48, 34, 59, 70, 37,
       50, 86,  7, 49, 40, 54, 94, 95, 20, 83, 59, 33,  0, 81, 18, 26, 69,
        2, 42, 51,  7, 42, 90, 94, 63, 14, 14, 71, 25, 85, 99, 40, 62, 29,
       42, 27, 98, 30, 89, 21, 78, 17, 33, 63, 80, 61, 50, 79, 38, 96,  8,
       85, 19, 76, 32, 19, 14, 37, 62, 24, 30, 19, 80, 55,  5, 94, 74, 85,
       59, 65, 17, 80, 11, 81, 84, 81, 46, 82, 66, 46, 78, 29, 40])
```

Дальше создаем гистограмму из этих данных, передавая аргумент функции hist(). Например, нужно разделить данные на 20 интервалов (значение по умолчанию − 10 интервалов). Для этого используется именованный аргумент bin.

```
n, bin, patches = plt.hist(pop, bins=20)
plt.show()
```

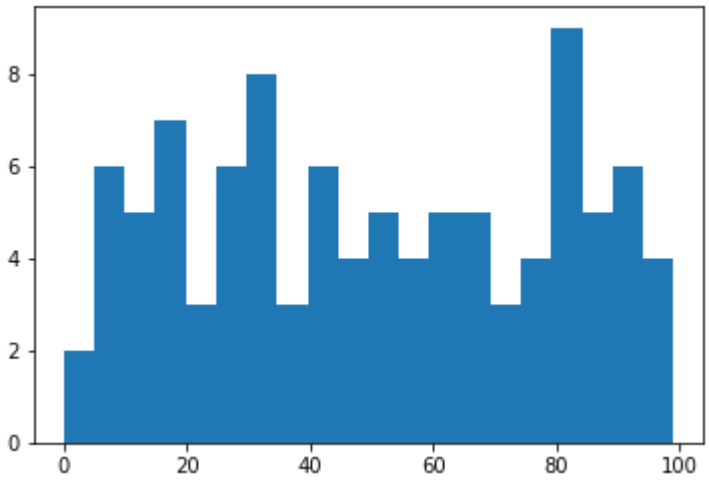


Рис. 7.13. Гистограмма

Столбчатые диаграммы

Еще один распространенный тип графиков − столбчатые диаграммы. Они похожа на гистограммы, но на оси x тут располагаются не числовые значения, а категории. В matplotlib для реализации столбчатых диаграмм используется функция bin().

```
index = [0,1,2,3,4]
values = [5,7,3,4,6]
plt.bar(index,values)
plt.show()
```

Всего нескольких строк кода достаточно для получения такой столбчатой диаграммы.

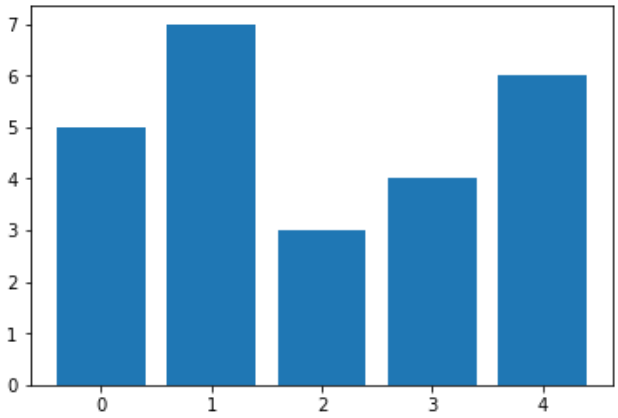


Рис. 7.14. Столбчатые диаграммы

На последней диаграмме видно, что метки на оси x написаны под каждым столбцом. Поскольку каждый из них относится к отдельной категории, правильнее обозначать их строками. Для этого используется функция xticks(). А для правильного размещения нужно передать список со значениями позиций в качестве первого аргумента в той же функции. Результатом будет такая диаграмма.

```
index = np.arange(5)
```

```
values1 = [5,7,3,4,6]
plt.bar(index, values1)
plt.xticks(index+0.4,['A','B','C','D','E'])
plt.show()
```

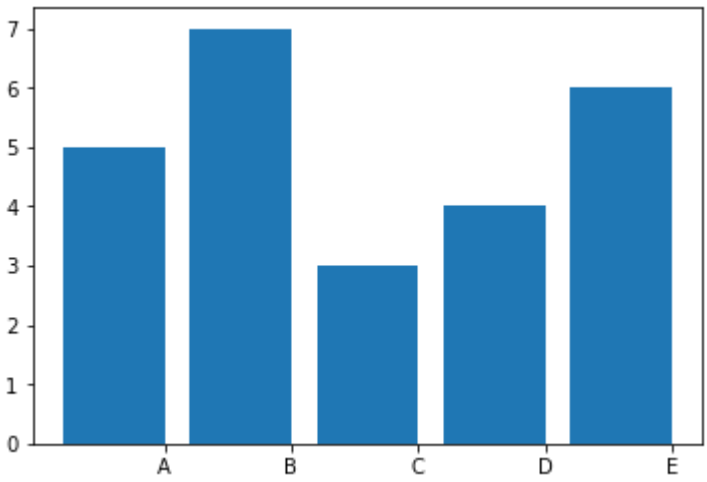


Рис. 7.15. Столбчатые диаграммы

Есть и множество других операций, которые можно выполнить для улучшения диаграммы. Каждая из них выполняется за счет добавления конкретного именованного аргумента в bar(). Например, можно добавить величины стандартного отклонения с помощью аргумента yerr вместе с соответствующими значениями. Часто этот аргумент используется вместе с error_kw, который принимает друге аргументы, отвечающие за представление погрешностей. Два из них − это eColor, который определяет цвета колонок погрешностей и capsize − ширину поперечных линий, обозначающих окончания этих колонок.

Еще один именованный аргумент − alpha. Он определяет степень прозрачности цветной колонки. Его значением может быть число от 0 до 1, где 0 − полностью прозрачный объект.

Также крайне рекомендуется использовать легенду, за которую отвечает аргумент label.

Результат − следующая столбчатая диаграмма с колонками погрешностей.

```
index = np.arange(5)
values1 = [5,7,3,4,6]
std1 = [0.8,1,0.4,0.9,1.3]
plt.title('A Bar Chart')
plt.bar(index, values1, yerr=std1, error_kw={'ecolor':'0.1','capsize':6},alpha=0.7,label='First')
plt.xticks(index+0.4,['A','B','C','D','E'])
plt.legend(loc=2)
plt.show()
```

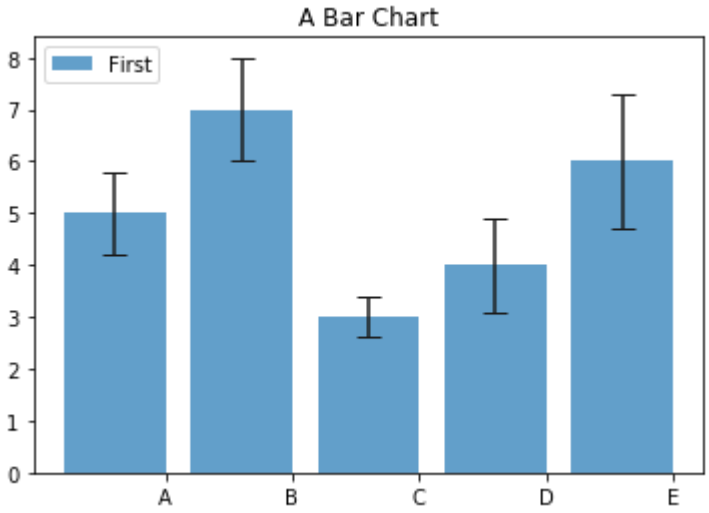


Рис. 7.16. Столбчатые диаграммы

Горизонтальные столбчатые диаграммы

В предыдущем разделе столбчатая диаграмма была вертикальной. Но блоки могут располагаться и горизонтально. Для этого режима есть специальная функция barh(). Аргументы и именованные аргументы, которые использовались для bar() будут работать и здесь. Единственное изменение в том, что поменялись роли осей. Категории теперь представлены на оси y, а числовые значения − на x.

```
index = np.arange(5)
values1 = [5,7,3,4,6]
std1 = [0.8,1,0.4,0.9,1.3]
plt.title('A Horizontal Bar Chart')
plt.barh(index, values1, xerr=std1, error_kw={'ecolor':'0.1','capsize':6},alpha=0.7,label='First')
plt.yticks(index+0.4,['A','B','C','D','E'])
plt.legend(loc=5)
plt.show()
```

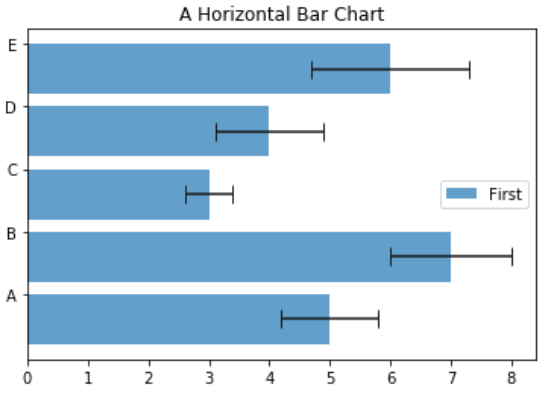


Рис. 7.17. Горизонтальные столбчатые диаграммы

Многорядные столбчатые диаграммы

Как и линейные графики, столбчатые диаграммы широко используются для одновременного отображения больших наборов данных. Но в слу-

чае с многорядными работает особая структура. До сих пор во всех примерах определялись последовательности индексов, каждый из которых соответствует столбцу, относящемуся к оси x. Индексы представляют собой и категории. В таком случае столбцов, которые относятся к одной и той же категории, даже больше.

Один из способов решения этой проблемы − разделение пространства индекса (для удобства его ширина равна 1) на то количество столбцов, которые к нему относятся. Также рекомендуется добавлять пустое пространство, которое будет выступать пропусками между категориями.

```
index = np.arange(5)
values1 = [5,7,3,4,6]
values2 = [6,6,4,5,7]
values3 = [5,6,5,4,6]
bw = 0.3
plt.axis([0,5,0,8])
plt.title('A Multiseries Bar Chart', fontsize=20)
plt.bar(index, values1, bw, color='b')
plt.bar(index+bw, values2, bw, color='g')
plt.bar(index+2*bw, values3, bw, color='r')
plt.xticks(index+1.5*bw,['A','B','C','D','E'])
plt.show()
```

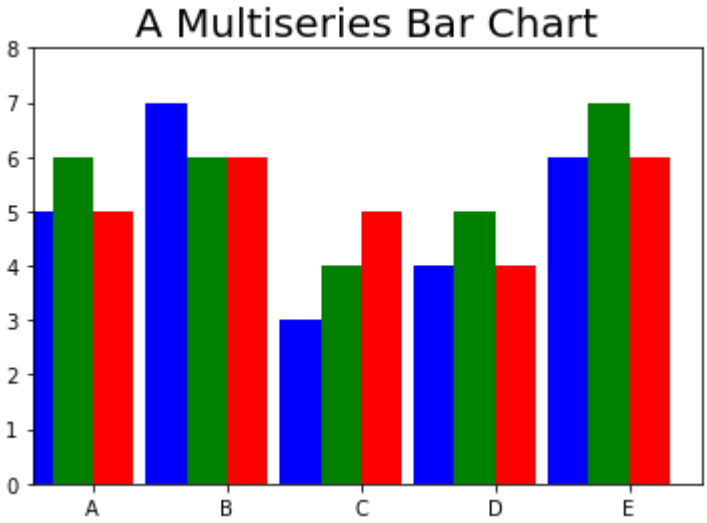


Рис. 7.17. Multisiries Bar Chart

В случае с горизонтальными многорядными столбчатыми диаграммами все работает по тому же принципу. Функцию bar() нужно заменить на соответствующую barh(), а также не забыть заменить xticks() на yticks(). И нужно развернуть диапазон значений на осях с помощью функции axis().

```
index = np.arange(5)
values1 = [5,7,3,4,6]
values2 = [6,6,4,5,7]
values3 = [5,6,5,4,6]
bw = 0.3
```

```
plt.axis([0,8,0,5])
plt.title('A Multiseries Bar Chart', fontsize=20)
plt.barh(index, values1, bw, color='b')
plt.barh(index+bw, values2, bw, color='g')
plt.barh(index+2*bw, values3, bw, color='r')
plt.yticks(index+0.4,['A','B','C','D','E'])
plt.show()
```

Многорядные столбчатые диаграммы с Dataframe из pandas

Как и в случае с линейными графиками matplotlib предоставляет возможность представлять объекты Dataframe с результатами анализа данных в форме столбчатых графиков. В этом случае все происходит даже быстрее и проще. Нужно лишь использовать функцию plot() по отношению к объекту Dataframe и указать внутри именованный аргумент kind, ему требуется присвоить тип графика, который будет выводиться. В данном случае это bar. Без дополнительных настроек результат должен выглядеть как на следующем изображении.

```
import pandas as pd

index = np.arange(5)
data = {'series1': [1,3,4,3,5],
        'series2': [2,4,5,2,4],
        'series3': [3,2,3,1,3]}
df = pd.DataFrame(data)
df.plot(kind='bar')
plt.show()
```

Но для еще большего контроля (или просто при необходимости) можно брать части Dataframe в виде массивов NumPy и описывать их так, как в предыдущем примере. Для этого каждый нужно передать в качестве аргумента функциям matplotlib.

К горизонтальной диаграмме применимы те же правила, но нужно не забыть указать значение barh для аргумента kind. Результатом будет горизонтальная столбчатая диаграмма как на следующем изображении.

Многорядные сложенные столбчатые графики

Еще один способ представления многорядного столбчатого графика − сложенная форма, где каждый столбец установлен поверх другого. Это особенно полезно в том случае, когда нужно показать общее значение суммы всех столбцов.

Для превращения обычного многорядного столбчатого графика в сложенный нужно добавить именованный аргумент bottom в каждую функцию bar(). Каждый объект Series должен быть присвоен соответствующему аргументу bottom. Результатом будет сложенный столбчатый график.

```
series1 = np.array([3,4,5,3])
```

```
series2 = np.array([1,2,2,5])
series3 = np.array([2,3,3,4])
index = np.arange(4)
plt.axis([-0.5,3.5,0,15])
plt.title('A Multiseries Stacked Bar Chart')
plt.bar(index,series1,color='r')
plt.bar(index,series2,color='b',bottom=series1)
plt.bar(index,series3,color='g',bottom=(series2+series1))
plt.xticks(index,['Jan18','Feb18','Mar18','Apr18'])
plt.show()
```

Здесь для создания аналогичного горизонтального графика нужно заменить bar() на barh(), не забыв про остальные параметры. Функцию xticks() необходимо поменять местами с yticks(), потому что метки категорий теперь будут расположены по оси y. После этого будет создан следующий горизонтальный график.

```
series1 = np.array([3,4,5,3])
series2 = np.array([1,2,2,5])
series3 = np.array([2,3,3,4])
index = np.arange(4)
plt.axis([0,15,-0.5,3.5])
plt.title('A Multiseries Horizontal Stacked Bar Chart')
plt.barh(index,series1,color='r')
plt.barh(index,series2,color='b',left=series1)
plt.barh(index,series3,color='g',left=(series2+series1))
plt.yticks(index,['Jan18','Feb18','Mar18','Apr18'])
plt.show()
```

До сих пор объекты Series разделялись только по цветам. Но можно использовать, например, разную штриховку. Для этого сперва необходимо сделать цвет столбца белым и использовать именованный аргумент hatch для определения типа штриховки. Все они выполнены с помощью символов (|, /, -, \, *), соответствующих стилю столбца. Чем чаще он повторяется, тем теснее будут расположены линии. Так, /// − более плотный вариант чем //, а этот, в свою очередь, плотнее /.

```
series1 = np.array([3,4,5,3])
series2 = np.array([1,2,2,5])
series3 = np.array([2,3,3,4])
index = np.arange(4)
plt.axis([0,15,-0.5,3.5])
plt.title('A Multiseries Horizontal Stacked Bar Chart')
plt.barh(index,series1,color='w',hatch='xx')
plt.barh(index,series2,color='w',hatch='///',left=series1)
plt.barh(index,series3,color='w',hatch='\\\\\\',left=(series2+series1))
plt.yticks(index,['Jan18','Feb18','Mar18','Apr18'])
```

```
plt.show()
```

Сложенные столбчатые графики с Dataframe из padans

В случае со сложенными столбчатыми графиками очень легко представлять значения объектов Dataframe с помощью функции plot(). Нужно лишь добавить в качестве аргумента stacked со значением True.

```
import pandas as pd

data = {'series1': [1,3,4,3,5],
        'series2': [2,4,5,2,4],
        'series3': [3,2,3,1,3]}
df = pd.DataFrame(data)
df.plot(kind='bar',stacked=True)
plt.show()
```

Другие представления столбчатых графиков

Еще один удобный тип представления данных в столбчатом графике − с использованием двух Series из одних и тех же категорий, где они сравниваются путем размещения друг напротив друга вдоль оси y. Для этого нужно разместить значения y одного из графиков в отрицательной форме. Также в этом примере показано, как поменять внутренний цвет другим способом. Это делается с помощью задания значения для аргумента facecolor.

Также вы увидите, как добавить значение y с меткой в конце каждого столбца. Это поможет улучшить читаемость всего графика. Это делается с помощью цикла for, в котором функция text() показывает значение y. Настроить положение метки можно с помощью именованных аргументов ha и va, которые контролируют горизонтальное и вертикальное выравнивание соответственно. Результатом будет следующий график.

```
x0 = np.arange(8)
y1 = np.array([1,3,4,5,4,3,2,1])
y2 = np.array([1,2,5,4,3,3,2,1])
plt.ylim(-7,7)
plt.bar(x0,y1,0.9, facecolor='g')
plt.bar(x0,-y2,0.9,facecolor='b')
plt.xticks(())
plt.grid(True)
for x, y in zip(x0, y1):
   plt.text(x, y + 0.05, '%d' % y, ha='center', va = 'bottom')
for x, y in zip(x0, y2):
   plt.text(x, -y - 0.05, '%d' % y, ha='center', va = 'top')
plt.show()
```

Круговая диаграмма

Еще один способ представления данных − круговая диаграмма, которую можно получить с помощью функции pie().

Даже для нее нужно передать основной аргумент, представляющий собой список значений. Пусть это будут проценты (где максимально значение − 100), но это может быть любое значение. А уже сама функция определит, сколько будет занимать каждое значение.

Также в случае с этими графиками есть другие особенности, которые определяются именованными аргументами. Например, если нужно задать последовательность цветов, используется аргумент colors. В таком случае придется присвоить список строк, каждая из которых будет содержать название цвета. Еще одна возможность − добавление меток каждой доле. Для этого есть labels, которой присваивает список строк с метками в последовательности.

А чтобы диаграмма была идеально круглой, необходимо в конце добавить функцию axix() со строкой equal в качестве аргумента. Результатом будет такая диаграмма.

```
labels = ['Nokia','Samsung','Apple','Lumia']
values = [10,30,45,15]
colors = ['yellow','green','red','blue']
plt.pie(values,labels=labels,colors=colors)
plt.axis('equal')
plt.show()
```

Чтобы сделать диаграмму более сложной, можно «вытащить» одну из частей. Обычно это делается с целью акцентировать на ней внимание. В этом графике, например, для выделения Nokia. Для этого используется аргумент explode. Он представляет собой всего лишь последовательность чисел с плавающей точкой от 0 до 1, где 1 − положение целиком вне диаграмма, а 0 − полностью внутри. Значение между соответствуют среднему градусу извлечения.

Заголовок добавляется с помощью функции title(). Также можно настроить угол поворота с помощью аргумента startangle, который принимает значение между 0 и 360, обозначающее угол поворота (0 – значение по умолчанию). Следующий график показывает все изменения.

```
labels = ['Nokia','Samsung','Apple','Lumia']
values = [10,30,45,15]
colors = ['yellow','green','red','blue']
explode = [0.3,0,0,0]
plt.title('A Pie Chart')
plt.pie(values,labels=labels,colors=colors,explode=explode,startan-
gle=180)
plt.axis('equal')
plt.show()
```

Но и это не все, что может быть на диаграмме. У нее нет осей, поэтому сложно передать точное разделение. Чтобы решить эту проблему, можно

использовать autopct, который добавляет в центр каждой части текст с соответствующим значением.

Чтобы сделать диаграмму еще более привлекательной визуально, можно добавить тень с помощью shadow со значением True. Результат − следующее изображение.

```
labels = ['Nokia','Samsung','Apple','Lumia']
values = [10,30,45,15]
colors = ['yellow','green','red','blue']
explode = [0.3,0,0,0]
plt.title('A Pie Chart')
plt.pie(values,labels=labels,colors=colors,explode=ex-
plode,shadow=True,autopct='%1.1f%%',startangle=180)
plt.axis('equal')
plt.show()
```

Круговые диаграммы с Dataframe из pandas

Даже в случае с круговыми диаграммами можно передавать значения из Dataframe. Однако каждая диаграмма будет представлять собой один Series, поэтому в примере изобразим только один объект, выделив его через df['series1'].

Указать тип графика можно с помощью аргумента kind в функции plot(), который в этом случае получит значение pie. Также поскольку он должен быть идеально круглым, обязательно задать figsize. Получится следующая диаграмма.

```
import pandas as pd

data = {'series1': [1,3,4,3,5],
        'series2': [2,4,5,2,4],
        'series3': [3,2,3,1,3]}
df = pd.DataFrame(data)
df['series1'].plot(kind='pie', figsize=(6,6))
plt.show()
```

2 Задания для выполнения

Визуализация данных с помощью Matplotlib

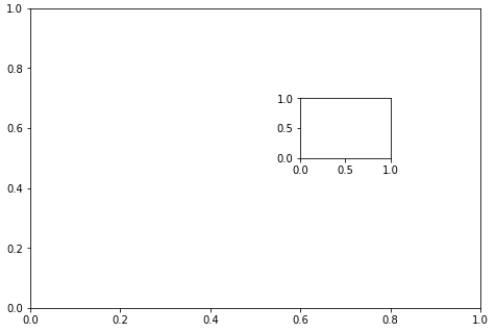
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
x = np.arange(0,100)
y = x*2
z = x**2
```

Вариант 1

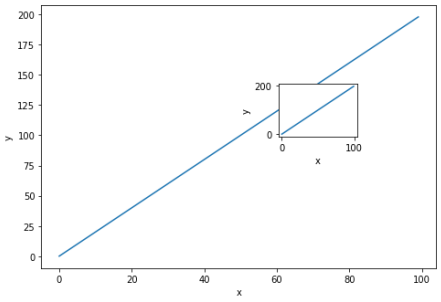
- Создайте объект класса Figure используя метод plt.figure() **
- ** Используйте add_axes чтобы добавить оси, занимающие все полотно фигуры.**
- ** Изобразите на полученном графике связь между x и y, чтобы получился похожий результат:**

Вариант 2

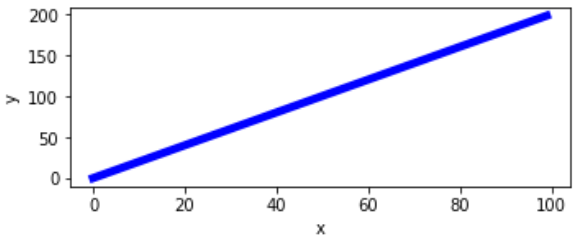
Создайте на одном полотне две пары осей, чтобы получилась похожая картинка:

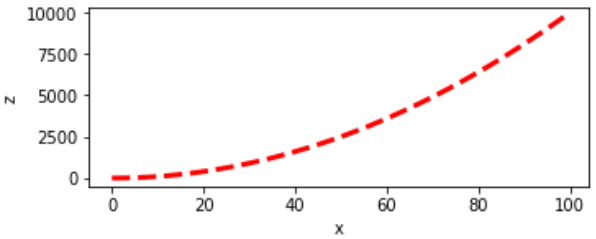


Визуализируйте данные на каждой паре осей



Напишите код, чтобы получить похожие графики:





Вариант 3

Написать программу для построения графика по данным

	Rank	Major_code	Major	Total	Men	Women	Major_category	ShareWomen	Sample_size	Employed	...	Part_time	Full_time_year_rou
0	1	2419	PETROLEUM ENGINEERING	2339.0	2057.0	282.0	Engineering	0.120564	36	1976	...	270	12
1	2	2416	MINING AND MINERAL ENGINEERING	756.0	679.0	77.0	Engineering	0.101852	7	640	...	170	3
2	3	2415	METALLURGICAL ENGINEERING	856.0	725.0	131.0	Engineering	0.153037	3	648	...	133	3
3	4	2417	NAVAL ARCHITECTURE AND MARINE ENGINEERING	1258.0	1123.0	135.0	Engineering	0.107313	16	758	...	150	6
4	5	2405	CHEMICAL ENGINEERING	32260.0	21239.0	11021.0	Engineering	0.341631	289	25694	...	5180	166

5 rows × 21 columns

Вариант 4

Написать программу для построения вертикальной диаграммы по данным

	Rank	Major_code	Major	Total	Men	Women	Major_category	ShareWomen	Sample_size	Employed	...	Part_time	Full_time_year_rou
0	1	2419	PETROLEUM ENGINEERING	2339.0	2057.0	282.0	Engineering	0.120564	36	1976	...	270	12
1	2	2416	MINING AND MINERAL ENGINEERING	756.0	679.0	77.0	Engineering	0.101852	7	640	...	170	3
2	3	2415	METALLURGICAL ENGINEERING	856.0	725.0	131.0	Engineering	0.153037	3	648	...	133	3
3	4	2417	NAVAL ARCHITECTURE AND MARINE ENGINEERING	1258.0	1123.0	135.0	Engineering	0.107313	16	758	...	150	6
4	5	2405	CHEMICAL ENGINEERING	32260.0	21239.0	11021.0	Engineering	0.341631	289	25694	...	5180	166

5 rows × 21 columns

Вариант 5

Написать программу для построения горизонтальной диаграммы по данным

	Rank	Major_code	Major	Total	Men	Women	Major_category	ShareWomen	Sample_size	Employed	...	Part_time	Full_time_year_rou
0	1	2419	PETROLEUM ENGINEERING	2339.0	2057.0	282.0	Engineering	0.120564	36	1976	...	270	12
1	2	2416	MINING AND MINERAL ENGINEERING	756.0	679.0	77.0	Engineering	0.101852	7	640	...	170	3
2	3	2415	METALLURGICAL ENGINEERING	856.0	725.0	131.0	Engineering	0.153037	3	648	...	133	3
3	4	2417	NAVAL ARCHITECTURE AND MARINE ENGINEERING	1258.0	1123.0	135.0	Engineering	0.107313	16	758	...	150	6
4	5	2405	CHEMICAL ENGINEERING	32260.0	21239.0	11021.0	Engineering	0.341631	289	25694	...	5180	166

5 rows × 21 columns

Лабораторная работа № 8. Элементы статистики. Методы подготовки и исследования данных

1 Методические рекомендации

1.1 Элементы статистического анализа данных

Произвольно сгенерируйте три набора данных.

```
import numpy as np
import pandas as pd

np.random.seed(1234)
d1 = pd.Series(2*np.random.normal(size = 100)+3)
d2 = np.random.f(2,4,size = 100)
d3 = np.random.randint(1,100,size = 100)
```

1.2 Функции, используемые в статистическом анализе

```
d1.count()        # Расчет непустого элемента
d1.min()          # Минимум
d1.max()          # Максимум
d1.idxmin()       # Положение минимального значения, аналогично функции which.min в R
d1.idxmax()       # Положение максимального значения, аналогично функции which.max в R
d1.quantile(0.1)  # 10% квантиль
d1.sum()          #
d1.mean()         #В среднем
d1.median()       #median
d1.mode()         # Режим
d1.var()          #variance
d1.std()          #Среднеквадратичное отклонение
d1.mad()          # Среднее абсолютное отклонение
d1.skew()         #Skewness
d1.kurt()         # Куртоз
d1.describe()     # Вывести сразу несколько описательных статистических показателей
```

- Следует отметить, что метод descirbe может использоваться только для последовательностей или фреймов данных, одномерные массивы не имеют этого метода.

Настройте функцию для агрегирования этих статистических показателей вместе:

```
def status(x) :
   return pd.Series([x.count(),x.min(),x.idxmin(),x.quantile(.25),x.median(),
                     x.quantile(.75),x.mean(),x.max(),x.idxmax(),x.mad(),x.var(),
```

x.std(),x.skew(),x.kurt()],index=['Всего',«Минимум»,'Минимальная позиция',"25% квантиль",
'медиана',"75% квантиль",'Значить',«Максимум»,'Максимальное количество цифр',«Среднее абсолютное отклонение»,'дисперсия','Среднеквадратичное отклонение','Асимметрия','Эксцесс'])

Выполните эту функцию и проверьте значения этих статистических функций набора данных d1:

```
df = pd.DataFrame(status(d1))
df
```

Результат:

	0
总数	100.000000
最小值	-2.089340
最小值位置	74.000000
25%分位数	1.998625
中位数	3.680687
75%分位数	4.791963
均值	3.403556
最大值	8.178326
最大值位数	29.000000
平均绝对偏差	1.770432
方差	4.986340
标准差	2.233011
偏度	-0.346918
峰度	-0.111452

Рис. 8.1. Результат работы

В реальной работе нам может потребоваться иметь дело с серией числовых фреймов данных. Как применить эту функцию к каждому столбцу фрейма данных? Вы можете использовать функцию apply, которая очень похожа на метод apply в R.

Создайте фрейм данных с ранее созданными данными d1, d2 и d3:

```
df = pd.DataFrame(np.array([d1,d2,d3]).T, columns=['x1','x2','x3'])
df.head()

df.apply(status)
```

Результат:

	x1	x2	x3
总数	100.000000	100.000000	100.000000
最小值	-0.993843	0.020867	1.000000
最小值位置	25.000000	18.000000	47.000000
25%分位数	1.868774	0.300948	21.750000
中位数	3.456328	0.671470	52.000000
75%分位数	5.173430	1.556927	75.250000
均值	3.417434	1.430218	50.390000
最大值	7.965557	14.712363	99.000000
最大值位数	0.000000	28.000000	0.000000
平均绝对偏差	1.661051	1.336215	25.174400
方差	4.079887	5.104443	862.321111
标准差	2.019873	2.259301	29.365305
偏度	0.008918	3.690352	-0.059582
峰度	-0.731357	16.310994	-1.202740

Рис. 8.2. Результат работы

3 Загрузить данные CSV.

```
import numpy as np
import pandas as pd

bank = pd.read_csv("D://bank/bank-additional-train.csv")
bank.head()    # Просмотр первых 5 строк
```

0

	age	job	marital	education	default	housing	loan	contact	month	day_of_week
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon
1	57	services	married	high.school	unknown	no	no	telephone	may	mon
2	37	services	married	high.school	no	yes	no	telephone	may	mon
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon
4	56	services	married	high.school	no	no	yes	telephone	may	mon

Рис. 8.3. Результат работы

Описательная статистика 1: описать ()

```
result = bank['age'].describe()
pd.DataFrame(result )   # Форматировать в DataFrame
```

	age
count	34999.000000
mean	39.764622
std	9.355707
min	18.000000
25%	32.000000
50%	38.000000
75%	47.000000
max	95.000000

Рис. 8.4. Результат работы

Описательная статистика 2: описать (include = [‘number’])

Include - это тип данных. Если вы хотите просмотреть статистические данные всех данных, вы можете заполнить объект, то есть include = ['object']; если вы хотите просмотреть данные типа float, тогда include = ['float'].

result = bank.describe(include=['object'])

	job	marital	education	default	housing	loan	contact	month	day_of_week
count	34999	34999	34999	34999	34999	34999	34999	34999	34999
unique	12	4	8	3	3	3	2	9	5
top	blue-collar	married	university.degree	no	yes	no	cellular	may	thu
freq	8704	21751	9735	26631	18152	28816	20731	12332	7601

Рис. 8.5. Результат работы

Значение:

- count: непустое общее количество указанного поля.
- уникальный: количество типов значений, хранящихся в этом поле. Например, если в столбце «Пол» хранятся мужские и женские значения, уникальным значением будет 2.
- вверху: наиболее многочисленное значение.
- freq: общее количество самых многочисленных значений.

bank.describe(include=['number'])

	age	duration	campaign	pdays	previous	emp.var.rate
count	34999.000000	34999.000000	34999.000000	34999.000000	34999.000000	34999.000000
mean	39.764622	255.621389	2.667905	993.013143	0.088774	0.506883
std	9.355707	260.531052	2.922104	76.874368	0.299302	1.264246
min	18.000000	0.000000	1.000000	0.000000	0.000000	-1.800000
25%	32.000000	99.000000	1.000000	999.000000	0.000000	-0.100000
50%	38.000000	175.000000	2.000000	999.000000	0.000000	1.100000
75%	47.000000	314.000000	3.000000	999.000000	0.000000	1.400000
max	95.000000	4918.000000	56.000000	999.000000	3.000000	1.400000

Рис. 8.6. Результат работы

Коэффициент корреляции непрерывных переменных (корр)
bank.corr()

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx
age	1.000000	-0.013953	0.016256	-0.003962	-0.021486	0.077069	0.046928
duration	-0.013953	1.000000	-0.074273	-0.029713	-0.007303	-0.020915	-0.000601
campaign	0.016256	-0.074273	1.000000	0.018813	-0.066620	0.126081	0.126844
pdays	-0.003962	-0.029713	0.018813	1.000000	-0.274155	0.124707	0.101393
previous	-0.021486	-0.007303	-0.066620	-0.274155	1.000000	-0.446758	-0.381965
emp.var.rate	0.077069	-0.020915	0.126081	0.124707	-0.446758	1.000000	0.815232
cons.price.idx	0.046928	-0.000601	0.126844	0.101393	-0.381965	0.815232	1.000000

Рис. 8.7. Результат работы

Матрица ковариации (cov)
bank.cov()

	age	duration	campaign	pdays	previous	emp.var.rate
age	87.529258	-34.010858	0.444403	-2.849469	-0.060166	0.911561
duration	-34.010858	67876.429111	-56.544048	-595.099715	-0.569451	-6.888859
campaign	0.444403	-56.544048	8.538690	4.226063	-0.058266	0.465775
pdays	-2.849469	-595.099715	4.226063	5909.668380	-6.307927	12.120019
previous	-0.060166	-0.569451	-0.058266	-6.307927	0.089582	-0.169049
emp.var.rate	0.911561	-6.888859	0.465775	12.120019	-0.169049	1.598319

Рис. 8.8. Результат работы

Удалить столбец

bank.drop('job', axis=1) # Удалить столбец возраста, ось = 1 имеет важное значение

Сортировать

bank.sort_values(by=['job','age']) # Сортировать по должностям и возрасту в порядке возрастания

bank.sort_values(by=['job','age'], ascending=False) # Сортировать по должностям, возрасту в порядке убывания

Соединение нескольких таблиц

Подготовьте данные:

```
import numpy as np
import pandas as pd

student = {'Name':['Bob','Alice','Carol','Henry','Judy','Robert','William'],
        'Age':[12,16,13,11,14,15,24],
        'Sex':['M','F','M','M','F','M','F']}

score = {'Name':['Bob','Alice','Carol','Henry','William'],
       'Score':[75,35,87,86,57]}

df_student = pd.DataFrame(student)
df_student

df_score = pd.DataFrame(score)
df_score
student :
```

	Age	Name	Sex
0	12	Bob	M
1	16	Alice	F
2	13	Carol	M
3	11	Henry	M
4	14	Judy	F
5	15	Robert	M
6	24	William	F

Рис. 8.9. Результат работы

score：

	Name	Score
0	Bob	75
1	Alice	35
2	Carol	87
3	Henry	86
4	William	57

Рис. 8.10. Результат работы

Внутреннее соединение

stu_score1 = pd.merge(df_student, df_score, on='Name')
stu_score1

- Обратите внимание, что по умолчанию функция слияния реализует внутреннее соединение между двумя таблицами, то есть возвращает общую часть данных в двух таблицах. Режим подключения можно установить с помощью параметра how, левое - левое соединение; правое - правое соединение; внешнее – внешнее соединение.

	Age	Name	Sex	Score
0	12	Bob	M	75
1	16	Alice	F	35
2	13	Carol	M	87
3	11	Henry	M	86
4	24	William	F	57

Рис. 8.11. Результат работы

Левое соединение

stu_score2 = pd.merge(df_student, df_score, on='Name',how='left')
stu_score2

	Age	Name	Sex	Score
0	12	Bob	M	75.0
1	16	Alice	F	35.0
2	13	Carol	M	87.0
3	11	Henry	M	86.0
4	14	Judy	F	NaN
5	15	Robert	M	NaN
6	24	William	F	57.0

Рис. 8.12. Результат работы

- В левой ссылке оценка студента без оценки - NaN.

Обработка отсутствующих значений

Данные в реальной жизни очень беспорядочные, и пропущенные значения также очень распространены. Существование пропущенных значений может повлиять на последующий анализ данных или работу по интеллектуальному анализу, так что же нам делать с этими пропущенными значениями? Обычно используются три метода, а именно: Удалить метод、Метод заполнениясучастием Интерполяция。

Удалить метод

Когда в большинстве значений переменной в данных отсутствуют значения, вы можете рассмотреть возможность удаления суммы изменения; когда отсутствующие значения распределены случайным образом, а количество отсутствующих значений невелико, вы также можете удалить эти отсутствующие наблюдения.

Замещающий метод

Для непрерывных переменных, если распределение переменной аналогично или нормально распределено, вы можете использовать среднее значение для замены этих недостающих значений; если переменная смещена, вы можете использовать медианное значение для замены этих отсутствующих значений; для дискретных переменных мы обычно используем режим для замены отсутствующих наблюдений.

1.3 Интерполяция

Метод интерполяции основан на методе моделирования Монте-Карло в сочетании с линейной моделью, обобщенной линейной моделью, деревом решений и другими методами для вычисления прогнозируемого значения для замены отсутствующего значения.

- В этом тесте для обработки используются указанные выше данные об успеваемости учащихся.

Запросить количество пустых данных в поле

```
sum(pd.isnull(stu_score2['Score']))
```

Результат: 2

Удалять отсутствующие значения напрямую

```
stu_score2.dropna()
```

- По умолчанию dropna удаляет все строки с пропущенными значениями.

Удалить все данные с пропущенными значениями

```
import numpy as np
import pandas as pd

df = pd.DataFrame([[1,2,3],[3,4,np.nan],
                   [12,23,43],[55,np.nan,10],
```

```
                [np.nan,np.nan,np.nan],[np.nan,1,2]],
                columns=['a1','a2','a3'])
```

df.dropna() # Эта операция удалит все строки с пропущенными значениями

df.dropna(how='all') # Эта операция удалит только данные строк, в которых все столбцы не содержат значений

Ввод данных

Чтобы использовать константу для заполнения отсутствующих значений, вы можете использовать функцию fillna для выполнения простой работы по заполнению:

1 Заполните все пропущенные значения 0

```
df.fillna(0)
```

2 Принять предыдущую заливку или обратную заливку

```
df.fillna(method='ffill') # заполняем предыдущим значением
df.fillna(method='bfill') # заполняем последним значением
```

3 Используйте константы для заполнения разных столбцов.

```
df.fillna({'a1':100,'a2':200,'a3':300})
```

4 Заполните соответствующие столбцы средним или медианным значением.

```
a1_median = df['a1'].median()  # Вычислить медианное значение столбца a1
a1_median=7.5

a2_mean = df['a2'].mean()  # Вычислить среднее значение столбца a2
a2_mean = 7.5

a3_mean = df['a3'].mean()  # Вычислить среднее значение столбца a3
a3_mean = 14.5

df.fillna({'a1':a1_median,'a2':a2_mean,'a3':a3_mean})  # Заполнить значение
```

- Очевидно, что при использовании метода заполнения более разумно использовать режим, среднее или медианное значение каждого столбца для заполнения режима, среднего или медианного значения каждого столбца по сравнению с постоянным заполнением или предшествующими и последующими терминами. Это также ярлык, обычно используемый в работе.

Перемешивание данных (перемешивание)

В реальной работе мы часто сталкиваемся с несколькими объединениями DataFrame и надеемся зашифровать данные. В пандах есть**sample**Функция может выполнить эту операцию.

df = df.sample(frac=1)

- Таким образом можно перемешать df. Параметр frac – это пропорция, которая должна быть возвращена. Например, в df 10 строк данных, и я хочу вернуть только 30% из них, тогда frac = 0,3.

Иногда нам может потребоваться отсортировать индекс набора данных после смешивания. Нам просто нужно это сделать

df = df.sample(frac=1).reset_index(drop=True)

2 Задания для выполнения

Статистический анализ. Подготовка и исследование данных

gender	F	M	diff
title			
E.T. the Extra-Terrestrial (1982)	4.089850	3.920264	-0.169586
Men in Black (1997)	3.817844	3.719000	-0.098844
Sixth Sense, The (1999)	4.477410	4.379944	-0.097465
Shakespeare in Love (1998)	4.181704	4.099936	-0.081768
Schindler's List (1993)	4.562602	4.491415	-0.071187
Toy Story (1995)	4.187817	4.130552	-0.057265
Princess Bride, The (1987)	4.342767	4.288942	-0.053826
Being John Malkovich (1999)	4.159930	4.113636	-0.046293
Shawshank Redemption, The (1994)	4.539075	4.560625	0.021550
Fargo (1996)	4.217656	4.267780	0.050124

Вариант 1

Рассчитать средний балл каждого фильма по полу

Вариант 2

Рассчитать количество оценок для каждого фильма

Вариант 3

Выберите фильмы с более чем 2000 оценок

Вариант 4

Показ 10 лучших фильмов, которые нравятся женщинам и мужчинам

Вариант 5

Рассчитайте разницу в рейтингах между мужчинами и женщинами (разница между мужчинами и женщинами в одном фильме)

Вариант 6

Выберите 10 лучших фильмов с наибольшим количеством разногласий, которые нравятся женщинам

Вариант 7

Выберите 10 лучших фильмов с наибольшим количеством разногласий, которые нравятся мужчинам

Лабораторная работа № 9. Машинное обучение в Scikit-learn

1 Методические рекомендации

1.1 Введение в машинное обучение. Обучение с учителем и без учителя

Машинное обучение: постановка вопроса

В общем, задача машинного обучения сводится к получению набора выборок данных и, в последствии, к попыткам предсказать свойства неизвестных данных. Если каждый набор данных − это не одиночное число, а например, многомерная сущность (multi-dimensional entry или multivariate data), то он должен иметь несколько признаков или фич.

Машинное обчение можно разделить на несколько больших категорий:

- **обучение с учителем** (или управляемое обучение). Здесь данные представлены вместе с дополнительными признаками, которые мы хотим предсказать. (Нажмите сюда, чтобы перейти к странице Scikit-Learn обучение с учителем). Это может быть любая из следующих задач:

1 **классификация:** выборки данных принадлежат к двум или более классам, и мы хотим научиться на уже размеченных данных предсказывать класс неразмеченной выборки. Примером задачи классификации может стать распознавание рукописных чисел, цель которого − присвоить каждому входному набору данных одну из конечного числа дискретных категорий. Другой способ понимания классификации − это понимание ее в качестве дискретной (как противоположность непрерывной) формы управляемого обучения, где у нас есть ограниченное количество категорий, предоставленных для N выборок; и мы пытаемся их пометить правильной категорией или классом.

2 **регрессионный анализ:** если желаемый выходной результат состоит из одного или более непрерывных переменных, тогда мы сталкиваемся с регрессионным анализом. Примером решения такой задачи может служить предсказание длинны лосося как результата функции от его возраста и веса.

3 **обучение без учителя** (или самообучение). В данном случае обучающая выборка состоит из набора входных данных X без каких-либо соответствующих им значений. Целью подобных задач может быть определение групп схожих элементов внутри данных. Это называется кластеризацией или кластерным анализом. Также задачей может быть установление распределения данных внутри пространства входов, называемое густотой ожидания (density estimation). Или это может быть выделение данных из высоко размерного пространства в двумерное или трехмерное с целью визуализации данных. (Нажмите сюда, чтобы перейти к странице Scikit-Learn обучение без учителя).

Обучающая выборка и контрольная выборка

Машинное обучение представляет собой обучение выделению некоторых свойств выборки данных и применение их к новым данным. Вот почему общепринятая практика оценки алгоритма в Машинном обучении − это разбиение данных вручную на два набора данных. Первый из них − это обучающая выборка, на ней изучаются свойства данных. Второй − контрольная выборка, на ней тестируются эти свойства.

Загрузка типовой выборки

Scikit-learn устанавливается вместе с несколькими стандартными выборками данных, например, iris и digits для классификации, и boston house prices dataset для регрессионного анализа.

Далее мы запускам Python интерпретатор из командной строки и загружаем выборки iris и digits. Установим условные обозначения: $ означает запуск интерпретатора Python, а >>> обозначает запуск командной строки Python:

```
$ python
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> digits = datasets.load_digits()
```

Набор данных − это объект типа «словарь», который содержит все данные и некоторые метаданных о них. Эти данные хранятся с расширением .data, например, массивы n_samples, n_features. При машинном обучении с учителем одна или более зависимых переменных хранятся с расширением .target. Для получения более полной информации о наборах данных перейдите в соответствующий раздел.

Например, набор данных digits.data дает доступ к фичам, которые можно использовать для классификации числовых выборок:

```
>>> print(digits.data)
[[  0.   0.   5. ...,   0.   0.   0.]
 [  0.   0.   0. ...,  10.   0.   0.]
 [  0.   0.   0. ...,  16.   9.   0.]
 ...,
 [  0.   0.   1. ...,   6.   0.   0.]
 [  0.   0.   2. ...,  12.   0.   0.]
 [  0.   0.  10. ...,  12.   1.   0.]]
```

a digits.target дает возможность определить в числовой выборке, какой цифре соответствует каждое числовое представление, чему мы и будем обучаться:

```
>>> digits.target
```

```
array([0, 1, 2, ..., 8, 9, 8])
```

Форма массива данных

Обычно данные представлены в виде двухмерного массива, такую форму имеют n_samples, n_features, хотя исходные данные могут иметь другую форму. В случае с числами, каждая исходная выборка − это представление формой (8, 8), к которому можно получить доступ, используя:

```
>>> digits.images[0]
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

Следующий простой пример с этим набором данных иллюстрирует, как, исходя из поставленной задачи, можно сформировать данные для использования в scikit-learn.

Обучение и прогнозирование

В случае с числовым набором данных цель обучения − это предсказать, принимая во внимание представление данных, какая цифра изображена. У нас есть образцы каждого из десяти возможных классов (числа от 0 до 9), на которым мы обучаем алгоритм оценки (estimator), чтобы он мог предсказать класс, к которому принадлежит неразмеченный образец.

В scikit-learn алгоритм оценки для классификатора − это Python объект, который исполняет методы fit(X, y) и predict(T). Пример алгоритма оценки − это класс sklearn.svm.SVC выполняет классификацию методом опорных векторов. Конструктор алгоритма оценки принимает в качестве аргументов параметры модели, но для сокращения времени, мы будем рассматривать этот алгоритм как черный ящик:

```
>>> from sklearn import svm
>>> clf = svm.SVC(gamma=0.001, C=100.)
```

Выбор параметров для модели

В этом примере мы установили значение gamma вручную. Также можно автоматически определить подходящие значения для параметров, используя такие инструменты как grid search и cross validation.

Мы назвали экземпляр нашего алгоритма оценки clf, так как он является классификатором. Теперь он должен быть применен к модели, т.е. он

должен обучится на модели. Это осуществляется путем прогона нашей обучающей выборки через метод fit. В качестве обучающей выборки мы можем использовать все представления наших данных, кроме последнего. Мы сделали эту выборку с помощью синтаксиса Python [:-1], что создало новый массив, содержащий все, кроме последней, сущности из digits.data:

```
>>> clf.fit(digits.data[:-1], digits.target[:-1])
SVC(C=100.0, cache_size=200, class_weight=None, coef0=0.0, degree=3,
  gamma=0.001, kernel='rbf', max_iter=-1, probability=False,
  random_state=None, shrinking=True, tol=0.001, verbose=False)
```

Теперь можно предсказать новые значения, в частности, мы можем спросить классификатор, какое число содержится в последнем представлении в наборе данных digits, которое мы не использовали в обучении классификатора:

```
>>> clf.predict(digits.data[-1])
array([8])
```

Соответствующее изображение представлено ниже:

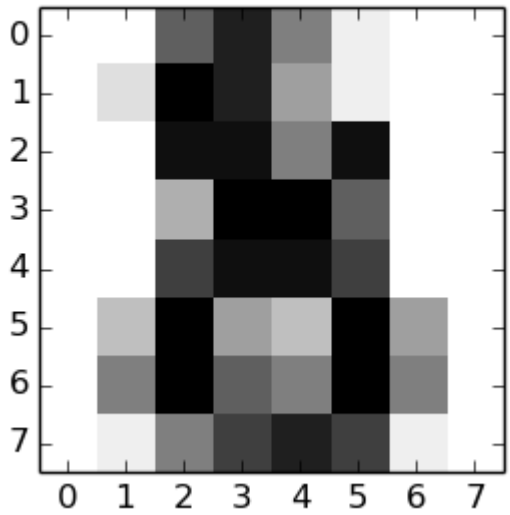


Рис. 9.1. Число классификатора

Как вы можете видеть, это сложная задача: представление в плохом разрешении. Вы согласны с классификатором?

Полное решение этой задачи классификации доступно в качестве примера, который вы можете запустить и изучить: Recognizing hand-written digits.

Сохранение модели

В scikit модель можно сохранить, используя встроенный модуль, названный pickle:

```
>>> from sklearn import svm
>>> from sklearn import datasets
>>> clf = svm.SVC()
>>> iris = datasets.load_iris()
>>> X, y = iris.data, iris.target
>>> clf.fit(X, y)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)

>>> import pickle
>>> s = pickle.dumps(clf)
>>> clf2 = pickle.loads(s)
>>> clf2.predict(X[0])
array([0])
>>> y[0]
0
```

В частном случае применения scikit, может быть полезнее заметить pickle на библиотеку joblib (joblib.dump & joblib.load), которая более эффективна для работы с большим объемом данных, но она позволяет сохранять модель только на диске, а не в строке:

```
>>> from sklearn.externals import joblib
>>> joblib.dump(clf, 'filename.pkl')
```

Потом можно загрузить сохраненную модель(возможно в другой Python процесс) с помощью:

```
>>> clf = joblib.load('filename.pkl')
```

Обратите внимание, что joblib.dump возвращает список имен файлов. Каждый отдельный массив numpy, содержащийся в clf объекте, сеарилизован как отдельный файл в файловой системе. Все файлы должны находиться в одной папке, когда вы снова загружаете модель с помощью joblib.load.

Обратите внимание, что у pickle есть некоторые проблемы с безопасностью и сопровождением. Для получения более детальной информации о хранении моделей в scikit-learn обратитесь к секции Model persistence.

1.2 Задачи кластеризации, классификации и регрессии

Датасет с домами на продажу

В качестве примера мы будем использовать датасет Kaggle, который содержит данные о домах на продажу в Бруклине с 2003 по 2017 года и до-

ступен для скачивания. Он содержит 111 атрибутов (столбцов) и 390883 записей (строк). В атрибуты включены: дата продажи, дата постройки, цена на дом, налоговый класс, соседние регионы, долгота, ширина и др.

```
# Если у вас Google Colab, то раскомментируйте
# import findspark
# findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
data = spark.read.csv(
'brooklyn_sales_map.csv',
inferSchema=True, header=True)
```

Если у вас Google Colab, то раскомментируйте # import findspark # findspark.init() from pyspark.sql import SparkSession spark = SparkSession.builder.master("local[*]").getOrCreate() data = spark.read.csv('brooklyn_sales_map.csv', inferSchema=True, header=True)

```
# Если у вас Google Colab, то раскомментируйте
# import findspark
# findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
data = spark.read.csv(
   'brooklyn_sales_map.csv',
   inferSchema=True, header=True)
```

О Kaggle API и о том, как установить PySpark в Google Colab, читайте здесь.

Готовим атрибут для последующей бинарной классификации

Допустим, требуется классифицировать налоговый класс на дом (tax_class). Всего имеется 10 таких классов. Поскольку данные распределены неравномерно (например, в классе 1 имеется 198969 записей, а в 3-м − только 18), мы разделим их на 2 категории: те, которые принадлежат классу 1, и остальные. В Python это делается очень просто, нужно просто вызвать метод replace:

```
by_1 = ['1', '1A', '1B', '1C']
by_others = ['2', '2A', '2B', '2C', '3', '4']
data = data.replace(by_others, '0', ['tax_class'])
data = data.replace(by_1, '1', ['tax_class'])
```

by_1 = ['1', '1A', '1B', '1C'] by_others = ['2', '2A', '2B', '2C', '3', '4'] data = data.replace(by_others, '0', ['tax_class']) data = data.replace(by_1, '1', ['tax_class'])

```
by_1 = ['1', '1A', '1B', '1C']
by_others = ['2', '2A', '2B', '2C', '3', '4']
data = data.replace(by_others, '0', ['tax_class'])
data = data.replace(by_1, '1', ['tax_class'])
```

Кроме того, алгоритмы Machine Learning в PySpark работают с числовым значениями, а не со строками. Поэтому преобразуем значения столбца tax_class в тип int:

```
data = data.withColumn('tax_class', data.tax_class.cast('int'))
data = data.withColumn('tax_class', data.tax_class.cast('int'))
data = data.withColumn('tax_class', data.tax_class.cast('int'))
```

Подбор признаков и преобразование категорий

Выберем следующие признаки для обучения модели Machine Learning: год постройки (year_of_sale), цена на дом (sale_price) и соседние регионы (neighborhood). Последние атрибут является категориальным признаком − в данных имеется 20 соседних регионов. Но опять же все значения этих категорий являются строковыми, поэтому нужно преобразовать их в числовые.

Можно воспользоваться методом replace, как это сделано выше, но придётся сначала извлечь названия всех 20 регионов. А можно использовать специальный класс StringIndexer из PySpark-модуля ML, который выполнит за нас всю работу. Объект этого класса принимает в качестве аргументов: название атрибута, который нужно преобразовать (inputCol), и название, которое будет иметь преобразованный атрибут (outputCol). Вот так это выглядит в Python:

```
from pyspark.ml.feature import StringIndexer
indexer = StringIndexer(inputCol="neighborhood", outputCol="neighborhood_id")
data = indexer.fit(data).transform(data)
from pyspark.ml.feature import StringIndexer indexer = StringIndexer(inputCol="neighborhood", outputCol="neighborhood_id") data = indexer.fit(data).transform(data)
from pyspark.ml.feature import StringIndexer

indexer = StringIndexer(inputCol="neighborhood", outputCol="neighborhood_id")
data = indexer.fit(data).transform(data)
```

Преобразованные категории имеют вид:

```
data.groupBy('neighborhood_id').count().show()
#
+---------------+-----+
|neighborhood_id|count|
+---------------+-----+
| 8.0|13215|
| 0.0|27279|
| 7.0|13387|
| 49.0| 2271|
| 29.0| 5074|
| 47.0| 2422|
```

42.0	3086
44.0	2802
35.0	4000
62.0	2
18.0	7342
1.0	21206
39.0	3396
37.0	3894
34.0	4037
25.0	5809
36.0	3984
41.0	3138
4.0	14608
23.0	6374
+---------------+-----+

data.groupBy('neighborhood_id').count().show() # +---------------+-----+ |neighborhood_id|count| +---------------+-----+ | 8.0|13215| | 0.0|27279| | 7.0|13387| | 49.0| 2271| | 29.0| 5074| | 47.0| 2422| | 42.0| 3086| | 44.0| 2802| | 35.0| 4000| | 62.0| 2| | 18.0| 7342| | 1.0|21206| | 39.0| 3396| | 37.0| 3894| | 34.0| 4037| | 25.0| 5809| | 36.0| 3984| | 41.0| 3138| | 4.0|14608| | 23.0| 6374| +---------------+-----+

data.groupBy('neighborhood_id').count().show()

#

+---------------+-----+

|neighborhood_id|count|

+---------------+-----+

| 8.0|13215|

| 0.0|27279|

| 7.0|13387|

| 49.0| 2271|

| 29.0| 5074|

| 47.0| 2422|

| 42.0| 3086|

| 44.0| 2802|

| 35.0| 4000|

| 62.0| 2|

| 18.0| 7342|

| 1.0|21206|

| 39.0| 3396|

| 37.0| 3894|

| 34.0| 4037|

| 25.0| 5809|

| 36.0| 3984|

| 41.0| 3138|

| 4.0|14608|

| 23.0| 6374|

```
+---------------+-----+
```

Теперь выберем необходимые признаки, а также отбросим строки с пустыми значениями с помощью метода dropna в PySpark:

```
features = ['year_of_sale', 'sale_price', 'neighborhood_id']
target = 'tax_class'
attributes = features + [target]
sample = data.select(attributes).dropna()
```

features = ['year_of_sale', 'sale_price', 'neighborhood_id'] target = 'tax_class' attributes = features + [target] sample = data.select(attributes).dropna()


```
features = ['year_of_sale', 'sale_price', 'neighborhood_id']
target = 'tax_class'
attributes = features + [target]
sample = data.select(attributes).dropna()
```


Векторизация признаков

Поскольку алгоритмы машинного обучения в PySpark принимают на вход только вектора, то нужно провести векторизацию. Для преобразования признаков в вектора используется класс VectorAssembler. Объект этого класса принимает в качестве аргументов список с названиями признаков, которые нужно векторизовать (inputCols), и название преобразованного признака (outputCol). После создания объекта VectorAssembler вызывается метод transform.

Для начала выберем в качестве признака для преобразования − цену на дом. Код на Python:

```
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=['sale_price'],
outputCol='features')
output = assembler.transform(sample)
```

from pyspark.ml.feature import VectorAssembler assembler = VectorAssembler(inputCols=['sale_price'], outputCol='features') output = assembler.transform(sample)


```
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(inputCols=['sale_price'],
                            outputCol='features')
output = assembler.transform(sample)
```


Полученный после векторизации DataFrame выглядит следующим образом:

```
output.show(5)
+------------+----------+---------------+---------+-------------+
|year_of_sale|sale_price|neighborhood_id|tax_class| features|
+------------+----------+---------------+---------+-------------+
| 2008| 499401179| 48.0| 0|[4.99401179E8]|
| 2016| 345000000| 41.0| 0| [3.45E8]|
| 2016| 340000000| 27.0| 0| [3.4E8]|
```

```
+------------+----------+--------------+---------+-------------+
output.show(5) +------------+----------+--------------+---------+-------------+
|year_of_sale|sale_price|neighborhood_id|tax_class| features| +------------+----------+--------------+---------+-------------+ | 2008| 499401179| 48.0| 0|[4.99401179E8]| | 2016| 345000000| 41.0| 0| [3.45E8]| | 2016| 340000000| 27.0| 0| [3.4E8]| +------------+----------+--------------+---------+-------------+
output.show(5)
+------------+----------+--------------+---------+-------------+
|year_of_sale|sale_price|neighborhood_id|tax_class|     features|
+------------+----------+--------------+---------+-------------+
|        2008| 499401179|          48.0|        0|[4.99401179E8]|
|        2016| 345000000|          41.0|        0|     [3.45E8]|
|        2016| 340000000|          27.0|        0|      [3.4E8]|
+------------+----------+--------------+---------+-------------+
```

Разделение датасета и обучение модели

Для решения задач Machine Learning всегда нужно иметь, как минимум, две выборки − обучающую и тестовую. На обучающей мы будем обучать модель, а на тестовой проверять эффективность обученной модели. В PySpark сделать это очень просто, нужно просто вызвать метод randomSplit, который разделит исходный датасет в заданной пропорции. Мы разделим в пропорции 80:20, в Python это выглядит так:

```
train, test = output.randomSplit([0.8, 0.2])
train, test = output.randomSplit([0.8, 0.2])
train, test = output.randomSplit([0.8, 0.2])
```

Теперь воспользуемся логистической регрессией (Logistic Regression) **[1]**, которая есть в PySpark, в качестве алгоритма Machine learning. Для этого нужно указать признаки, на которых модель обучается, и признак, который нужно классифицировать. Мы преобразовали цену на дом (sale price) в вектор под названием features, поэтому именно его и указываем в аргументе:

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol='features',
labelCol='tax_class')
model = lr.fit(train)
from pyspark.ml.classification import LogisticRegression lr = LogisticRegression(featuresCol='features', labelCol='tax_class') model = lr.fit(train)
from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(featuresCol='features',
                        labelCol='tax_class')
model = lr.fit(train)
```

Осталось только получить предсказания. Для этого вызывается метод transform, который принимает тестовую выборку:

```
predictions = model.transform(test)
predictions = model.transform(test)
predictions = model.transform(test)
```

Проверим эффективность модели, используя метрику качества. И в этом случае PySpark нас выручает, поскольку у него есть класс BinaryClassificationEvaluator. Нужно лишь указать целевой признак (tax class), а затем вызвать метод evaluate и передать в него наши предсказания. В Python это выглядит так:

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(labelCol='tax_class')
print('Evaluation:', evaluator.evaluate(predictions))
# Evaluation: 0.5242388483600111
```

from pyspark.ml.evaluation import BinaryClassificationEvaluator evaluator = BinaryClassificationEvaluator(labelCol='tax_class') print('Evaluation:', evaluator.evaluate(predictions)) # Evaluation: 0.5242388483600111

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator(labelCol='tax_class')
print('Evaluation:', evaluator.evaluate(predictions))
# Evaluation: 0.5242388483600111
```

Как видим, мы получили точность только 52%, что очень мало. Попробуем добавить ещё несколько признаков для обучения.

Добавление признаков

Векторизуем также год постройки (year_of_sale) и соседние регионы (neighborhood_id). Для этого нужно только в VectorAssembler указать выбранные признаки:

```
features = ['year_of_sale', 'sale_price', 'neighborhood_id']
assembler = VectorAssembler(inputCols=features,
outputCol='features')
output = assembler.transform(sample)
```

features = ['year_of_sale', 'sale_price', 'neighborhood_id'] assembler = VectorAssembler(inputCols=features, outputCol='features') output = assembler.transform(sample)

```
features = ['year_of_sale', 'sale_price', 'neighborhood_id']

assembler = VectorAssembler(inputCols=features,
                            outputCol='features')
output = assembler.transform(sample)
```

Python-код для остальных шагов − разделение на тестовую и обучающую выборки, обучение и оценивание модели − остаётся все тем же. В итоге, мы смогли повысить точность до 60%:

```
print('Evaluation:', evaluator.evaluate(predictions))
# Evaluation: 0.6019972898385996
```

print('Evaluation:', evaluator.evaluate(predictions)) # Evaluation: 0.6019972898385996

print('Evaluation:', evaluator.evaluate(predictions))

Evaluation: 0.6019972898385996

Отметим также, что при большем количестве классов в качестве метрики следует использовать MultilabelClassificationEvaluator, вместо BinaryClassificationEvaluator.

1.3 Библиотека Scikit-learn для машинного обучения

Scikit-learn− библиотека машинного обучения на языке программирования Python с открытым исходным кодом. Содержит реализации практически всех возможных преобразований, и нередко ее одной хватает для полной реализации модели. В данной библиотеки реализованы методы разбиения датасета на тестовый и обучающий, вычисление основных метрик над наборами данных, проведение Кросс-валидация. В библиотеке также есть основные алгоритмы машинного обучения: линейной регрессии и её модификаций Лассо, гребневой регрессии, опорных векторов, решающих деревьев и лесов и др. Есть и реализации основных методов кластеризации. Кроме того, библиотека содержит постоянно используемые исследователями методы работы с признаками: например, понижение размерности методом главных компонент. Частью пакета является библиотека imblearn, позволяющая работать с разбалансированными выборками и генерировать новые значения.

Линейная регрессия

```
# Add required imports
 import matplotlib.pyplot as plt
 import numpy as np
 from sklearn import datasets
 from sklearn.linear_model import LinearRegression
 from sklearn.metrics import mean_squared_error, r2_score
```

Загрузка датасета:

```
 diabetes = datasets.load_diabetes()
 # Use only one feature
 diabetes_X = diabetes.data[:, np.newaxis, 2]
```

Разбиение датасета на тренировочный и тестовый:

```
 # Split the data into training/testing sets
 x_train = diabetes_X[:-20]
 x_test = diabetes_X[-20:]

 # Split the targets into training/testing sets
 y_train = diabetes.target[:-20]
 y_test = diabetes.target[-20:]
```

Построение и обучение модели:

```
 lr = LinearRegression()
 lr.fit(x_train, y_train)
```

```
predictions = lr.predict(x_test)
```
Оценка алгоритма:
```
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(y_test, predictions))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, predictions))
> Mean squared error: 2548.07
  Variance score: 0.47
```
Построение графика прямой, получившейся в результате работы линейной регрессии:
```
plt.scatter(x_test, y_test,  color='black')
plt.plot(x_test, predictions, color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

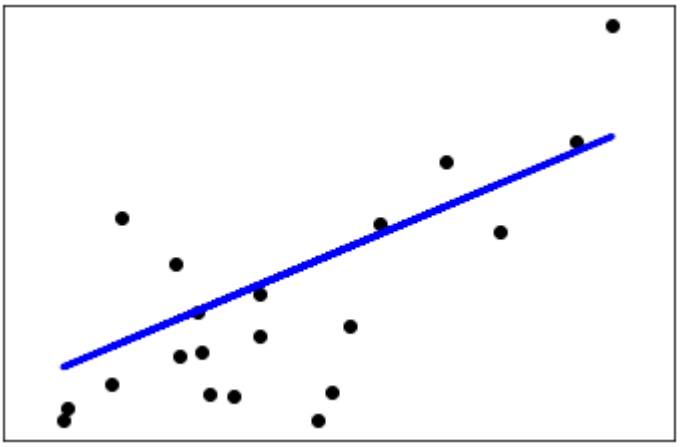


Рис. 9.2. Линейная регрессия

Логистическая регрессия

Загрузка датасета:
```
from sklearn.datasets import load_digits
digits = load_digits()
```
Вывод первых трех тренировочных данных для визуализации:
```
import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(20,4))
for index, (image, label) in enumerate(zip(digits.data[0:3], digits.target[0:3])):
  plt.subplot(1, 3, index + 1)
  plt.imshow(np.reshape(image, (8,8)), cmap=plt.cm.gray)
  plt.title('Training: %i\n' % label, fontsize = 20)
```

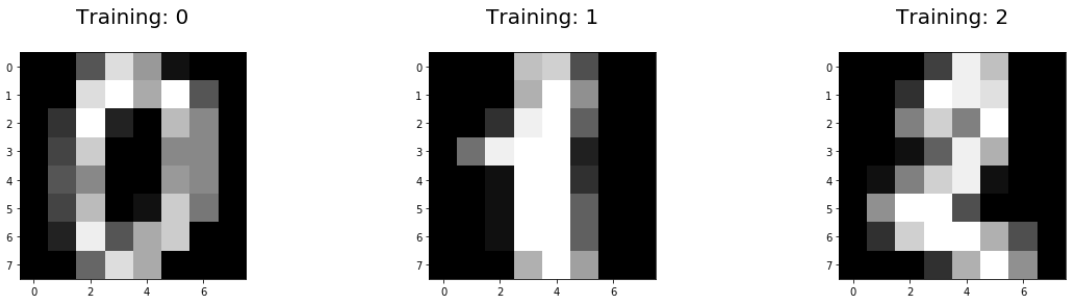


Рис. 9.3. Вывод первых трех тренировочных данных для визуализации

Разбиение датасета на тренировочный и тестовый:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.25, random_state=0)
```

Построение и обучение модели:

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(x_train, y_train)
predictions = lr.predict(x_test)
```

Оценка алгоритма:

```
score = lr.score(x_test, y_test)
print("Score: %.3f" % score)
> Score: 0.953
```

Перцептрон

Загрузка датасета:

```
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

Разбиение датасета на тренировочный и тестовый:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

Трансформация признаков:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Построение и обучение модели:

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
mlp.fit(X_train, y_train.values.ravel())
predictions = mlp.predict(X_test)
```

Оценка алгоритма:

```
from sklearn.metrics import classification_report, confusion_matrix
```

```
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
> [[ 7  0  0]
  [ 0  8  1]
  [ 0  2 12]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       0.80      0.89      0.84         9
           2       0.92      0.86      0.89        14

    micro avg       0.90      0.90      0.90        30
    macro avg       0.91      0.92      0.91        30
 weighted avg       0.90      0.90      0.90        30
```

Метрический классификатор и метод ближайших соседей

Дерево решений и случайный лес

Обработка естественного языка

Загрузка датасета:

```
from sklearn import fetch_20newsgroups
twenty_train = fetch_20newsgroups(subset='train', shuffle=True, random_state=42)
```

Вывод первых трех строк первого тренивочного файла и его класса:

```
print("\n".join(twenty_train.data[0].split("\n")[:3]))
print(twenty_train.target_names[twenty_train.target[0]])
> From: lerxst@wam.umd.edu (where's my thing)
  Subject: WHAT car is this!?
  Nntp-Posting-Host: rac3.wam.umd.edu

  rec.autos
```

Построение и обучение двух моделей. Первая на основе Байесовской классификации[на 28.01.19 не создан], а вторая использует метод опорных векторов:

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer

from sklearn.naive_bayes import MultinomialNB
text_clf1 = Pipeline([
  ('vect', CountVectorizer()),
  ('tfidf', TfidfTransformer()),
  ('clf', MultinomialNB()),
])

from sklearn.linear_model import SGDClassifier
text_clf2 = Pipeline([
```

```
  ('vect', CountVectorizer()),
  ('tfidf', TfidfTransformer()),
  ('clf', SGDClassifier(loss='hinge', penalty='l2',
                        alpha=1e-3, random_state=42,
                        max_iter=5, tol=None)),
])

text_clf1.fit(twenty_train.data, twenty_train.target)
text_clf2.fit(twenty_train.data, twenty_train.target)
```

Оценка алгоритмов:

```
twenty_test = fetch_20newsgroups(subset='test', shuffle=True, random_state=42)
docs_test = twenty_test.data
predicted1 = text_clf1.predict(docs_test)
predicted2 = text_clf2.predict(docs_test)
print("Score: %.3f" % np.mean(predicted1 == twenty_test.target))
print("Score: %.3f" % np.mean(predicted2 == twenty_test.target))
> Score for naive Bayes: 0.774
  Score for SVM: 0.824
```

Кросс-валилация и подбор параметров

Возьмем предыдущий пример с обработкой естественного языка и попробуем увеличить точность алгоритма за счет кросс-валидации и подбора параметров:

```
from sklearn.model_selection import GridSearchCV
parameters = {
   'vect__ngram_range': [(1, 1), (1, 2)],
   'tfidf__use_idf': (True, False),
   'clf__alpha': (1e-2, 1e-3),
}

gs_clf = GridSearchCV(text_clf2, parameters, cv=5, iid=False, n_jobs=-1)
gs_clf = gs_clf.fit(twenty_train.data, twenty_train.target)

print("Best score: %.3f" % gs_clf.best_score_)

for param_name in sorted(parameters.keys()):
  print("%s: %r" % (param_name, gs_clf.best_params_[param_name]))
> Best score: 0.904
  clf__alpha: 0.001
  tfidf__use_idf: True
  vect__ngram_range: (1, 2)
```

Метод опорных векторов (SVM)

Загрузка датасета:

```
from sklearn import datasets
iris = datasets.load_iris()
```
Разбиение датасета на тестовый и тренировочный:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.25, random_state=0)
```
Построение и обучение модели:
```
clf = svm.SVC(kernel='linear', C=1.0)
clf.fit(x_train, y_train)
predictions = clf.predict(x_test)
```
Оценка алгоритма:
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
> [[13  0  0]
  [ 0 15  1]
  [ 0  0  9]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      0.94      0.97        16
           2       0.90      1.00      0.95         9

    micro avg       0.97      0.97      0.97        38
    macro avg       0.97      0.98      0.97        38
 weighted avg       0.98      0.97      0.97        38
```
EM-алгоритм
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from sklearn import mixture

n_samples = 300

# generate random sample, two components
np.random.seed(0)

# generate spherical data centered on (20, 20)
shifted_gaussian = np.random.randn(n_samples, 2) + np.array([20, 20])

# generate zero centered stretched Gaussian data
C = np.array([[0., -0.7], [3.5, .7]])
stretched_gaussian = np.dot(np.random.randn(n_samples, 2), C)

# concatenate the two datasets into the final training set
```

```
X_train = np.vstack([shifted_gaussian, stretched_gaussian])

# fit a Gaussian Mixture Model with two components
clf = mixture.GaussianMixture(n_components=2, covariance_type='full')
clf.fit(X_train)

# display predicted scores by the model as a contour plot
x = np.linspace(-20., 30.)
y = np.linspace(-20., 40.)
X, Y = np.meshgrid(x, y)
XX = np.array([X.ravel(), Y.ravel()]).T
Z = -clf.score_samples(XX)
Z = Z.reshape(X.shape)

CS = plt.contour(X, Y, Z, norm=LogNorm(vmin=1.0, vmax=1000.0),
                 levels=np.logspace(0, 3, 10))
CB = plt.colorbar(CS, shrink=0.8, extend='both')
plt.scatter(X_train[:, 0], X_train[:, 1], .8)

plt.title('Negative log-likelihood predicted by a GMM')
plt.axis('tight')
plt.show()
```

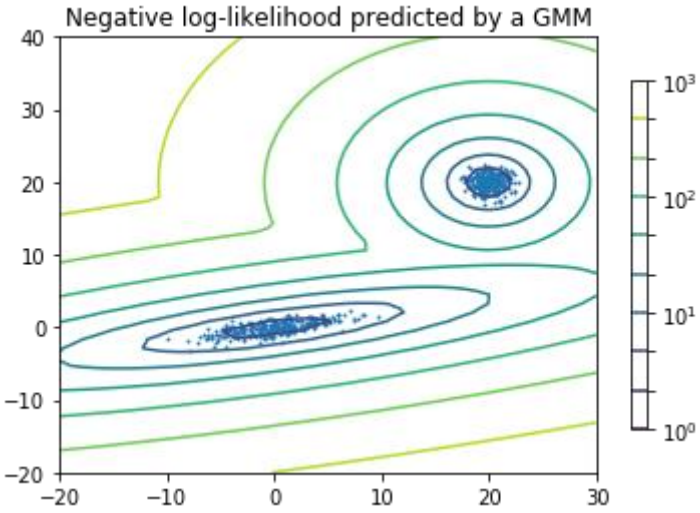


Рис. 9.4 График

2 Задания для выполнения

Кластерный анализ

для алгоритма является список некоторых элементов и функция схожести между ними («близости» элементов). Каждый элемент имеет «координаты» в определенном пространстве (например, в пространстве количеств слов), по которым и вычисляется схожесть между элементами.

элементы − блоги, 2 «координаты» − количество слов «php» и «xml» в содержимом блогов.

У блога № 1 – 7 раз встречается слово «php» и 5 раз «xml». Тогда «координаты» блога №1 в пространстве количества этих слов − (7;5).

У блога № 2 − 6 раз встречается слово «php» и 4 раза «xml». Тогда «координаты» блога №2 в пространстве количества этих слов − (6;4).

Схожесть определим с помощью функции − евклидова расстояния.

d(blog1, blog2) = sqrt((7-6)^2 + (5-4)^2) = 1.414

На выходе мы получаем дерево из кластеров (иерархию кластеров).

Задачи классификации и регрессии

Реализовать и обучить нейросеть.

Построение и оптимизация моделей Scikit-learn

загрузить содержимое файла и категории

выделить вектора признаков, подходящих для машинного обучения

обучить одномерную модель выполнять категоризацию

использовать стратегию grid search, чтобы найти наилучшую конфигурацию для извлечения признаков и для классификатора

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Основная литература

1 **Чернышев, С. А.** Основы программирования на Python : учебное пособие для вузов / С. А. Чернышев. – Москва : Юрайт, 2021. – 286 с. – ISBN 978-5-534-14350-8.

2 **Федоров, Д. Ю.** Программирование на языке высокого уровня Python : учебное пособие для вузов / Д. Ю. Федоров. – 3-е изд., перераб. и доп. – Москва : Юрайт, 2021. – 210 с. – ISBN 978-5-534-14638-7.

3 **Тузовский, А. Ф.** Объектно ориентированное программирование : учебное пособие для вузов / А. Ф. Тузовский. – Москва : Юрайт, 2020. – 206 с. – ISBN 978-5-534-00849-4.

4 **Гниденко, И. Г.** Технологии и методы программирования : учебное пособие для вузов / И. Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. – Москва : Юрайт, 2021. – 235 с. – ISBN 978-5-534-02816-4.

5 **Гниденко, И. Г.** Технология разработки программного обеспечения : учебное пособие для среднего профессионального образования / И. Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. – Москва : Юрайт, 2021. – 235 с. – ISBN 978-5-534-05047-9.

6 **Бизли, Д. М.** Язык программирования Python : справочник / Д. М. Бизли ; перевод с английского. – Киев : ДиаСофт, 2000.

7 **Гифт, Н.** Python в системном администрировании UNIX и Linux / Н. Гифт, Д. Джонс ; перевод с английского. – Санкт-Петербург : СимволПлюс, 2009.

8 **Лейнингем, И.** Освой самостоятельно Python за 24 часа / И. Лейнингем ; перевод с английского. – Москва : «Вильямс», 2001.

9 **Лесса, А.** Python. Руководство разработчика / А. Лесса ; перевод с английского. – Санкт-Петербург : ДиасофтЮП, 2001.

10 **Лутц, М.** Изучаем Python / М. Лутц ; перевод с английского. – Санкт-Петербург : Символ-Плюс, 2009.

11 **Лутц, М.** Программирование на Python / М. Лутц ; перевод с английского. – Санкт-Петербург : Символ-Плюс, 2002.

12 **Саммерфельд, М.** Программирование на Python 3. Подробное руководство / М. Саммерфельд ; перевод с английского. – Санкт-Петербург : Символ-Плюс, 2009.

13 **Сузи, Р. А.** Python / Р. А. Сузи. – Санкт-Петербург : БХВ-Петербург, 2002.

14 **Сузи, Р. А.** Язык Python и его применения : учебное пособие / Р. А. Сузи. – Москва : Интернет-Университет информационных технологий : БИНОМ, Лаборатория знаний, 2006.

15 Язык программирования Python / Г. Россум [и др.]. – Санкт-Петербург : АНО «Институт логики» – Невский диалект, 2001

Дополнительные источники

1 Хорошая шпаргалка по Python 3 на русском. – URL: https://pythonworld.ru/uploads/mementopython3-russian.pdf

2 Подборка книг о python для начинающих (на русском языке). – URL: https://pythonworld.ru/bookshop/category/beginners.
3 Стандартная библиотека python. – URL: https://pythonworld.ru/moduli.
4 Примеры программ на языке python. – URL: https://pythonworld.ru/primery-programm
5 Уроки по Python для начинающих. – URL: https://pythonru.com/uroki/vvedenie-uroki-po-python-dlja-nachinajushhih.
6 Учите Питон. Бесплатный курс по программированию с нуля. – URL: https://pythontutor.ru/.
7 Основные возможности Python. – URL: https://pythonchik.ru/osnovy
8 Лаборатория линуксоида – Python. Обучение языку программирования. – URL: https://younglinux.info/python/.

Учебное издание

Игнатьева Олеся Владимировна
Мялова Мария Ильинична

БОЛЬШИЕ ДАННЫЕ
И ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗ

Печатается в авторской редакции
Технический редактор К. И. Паханова

Подписано в печать 03.03.2022. Формат 60×84/16.
Усл. печ. л. 9,07. Тираж экз. Изд. № 507. Заказ .

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Ростовский государственный университет путей сообщения»
(ФГБОУ ВО РГУПС)

Адрес университета: 344038, г. Ростов н/Д, пл. Ростовского Стрелкового
Полка Народного Ополчения, д. 2, www.rgups.ru