Министерство науки и высшего образования Российской Федерации
Южно-Уральский государственный университет
Кафедра «Информационные технологии в экономике»

004(07)
М749

В.В. Мокеев

WEB-АНАЛИТИКА НА PYTHON

Практикум

Челябинск
Издательский центр ЮУрГУ
2020

УДК 004(075.8) + 519.86(075.8)
М749

Одобрено
учебно-методической комиссией
Высшей школы экономики и управления

Рецензенты:
генеральный директор ООО «ЛАНИТ-УРАЛ» А.А. Глазунова,
директор ООО «Диджитал-продакшн» Д.А. Жердев

Мокеев, В.В.

М749 Web-аналитика на Python: практикум / В.В. Мокеев. – Челябинск: Издательский центр ЮУрГУ, 2020. – 144 с.

Пособие предназначено для изучения методов машинного обучения. Рассматривается технология решения задач с использованием таких методов классификации как дерево решений Random Forest. Пособие ориентировано на студентов, изучающих дисциплины «Проектирование информационных систем», «Методы анализа и синтеза информационных систем», «Практикум по видам профессиональной деятельности».

Пособие предназначено для студентов образовательного направления «Бизнес-информатика».

УДК 004(075.8) + 519.86(075.8)

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Классический аппарат анализа и прогнозирования поведения социально-экономических систем (предприятия, организации, рынки и т.п. в дальнейшем СЭС) не удовлетворяет современным требованиям к качеству управленческих решений, оперативности их принятия и реализации. Так же на эффективность управленческих решений кроме внутренних факторов значительное влияние оказывают факторы внешней среды, их нестабильность и неопределенность.

Пособие предназначено для обучения по дисциплинам «Проектирование информационных систем», «Практикум по видам профессиональной деятельности», «Управление жизненным циклом информационных систем» в рамках бакалаврской образовательной программы по направлению «Бизнес-информатика.

Цель пособия – помочь студентам быстро и эффективно освоить теоретические и прикладные основы применения методов анализа данных, парсинга веб-сайтов. В первой главе представлены основные элементы языка Python, способы загрузки данных из файла, сведения об обработке и визуализации данных. Во второй главе изучаются основы методы машинного обучения. Третья глава посвящена изучению средств для парсинга веб-сайтов. Четвертая глава посвящена изучению соредств извлечения данных их баз данных интьернет ресурсов.

Методическая последовательность изложения материала дает студентам схему организации самостоятельной работы в области моделирования бизнес-процессов и проектирования баз данных. Для успешного освоения материала каждого практического занятия необходимо внимательно изучить теоретические материалы, затем выполнить учебное задание, для которого имеется подробная технология выполнения. Если студент успешно разобрался с учебным заданием, то он может переходить к выполнению самостоятельного задания.

Глава 1. АНАЛИЗ И ВИЗУАЛИЗАЦИЯ ДАННЫХ

1.1. Установка Python

На сегодняшний день существуют две версии Python – это Python 2 и Python 3, у них отсутствует полная совместимость друг с другом. Мы будем использовать Python 3, и, в дальнейшем, если где-то будет встречаться слово Python, то под ним следует понимать Python 3.

Для удобства запуска примеров и изучения языка Python, советуем установить на свой ПК пакет Anaconda. Этот пакет включает в себя интерпретатор языка Python, 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. Рассмотрим установку Anaconda в Windows.

1. Запустите скачанный инсталлятор. В первом появившемся окне необходимо нажать Next (рис. 1.1).

Рис. 1.1. Шаг 1 – начало установки

2. Далее следует принять лицензионное соглашение (рис. 1.2).

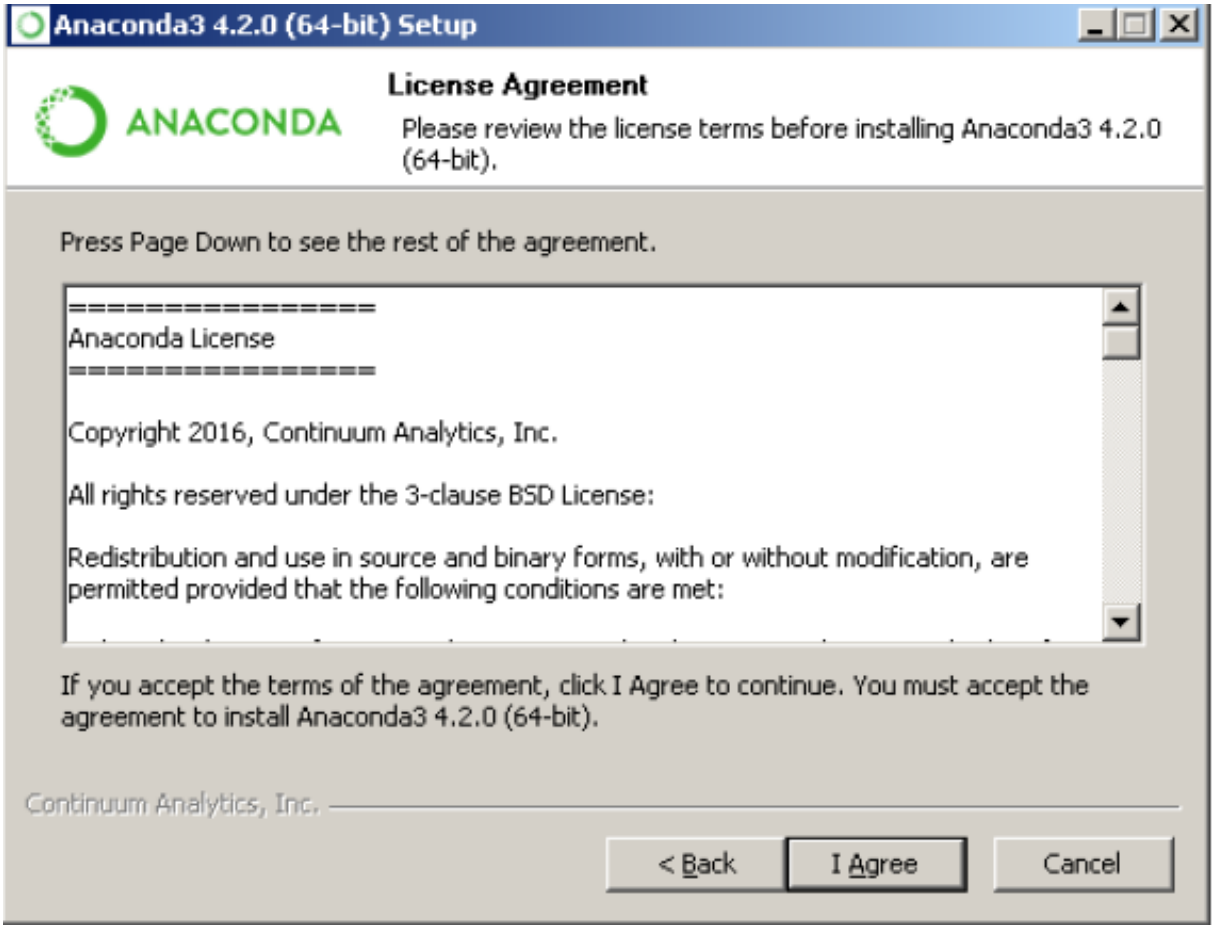

Рис. 1.2. Шаг 2 – принять лицензионное соглашение

3. На третьем шаге выберете одну из опций установки (рис. 1.3):

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

4. На четвертом шаге укажите путь, по которому будет установлена Anaconda (рис.1.4).

5. На пятом шаге укажите дополнительные опции (нужно поставить галочки, рис. 1.5):

- Add Anaconda to the system PATH environment variable – добавить Anaconda в системную переменную PATH;
- Register Anaconda as the system Python 3.5 – использовать Anaconda, как интерпретатор Python 3.5 по умолчанию;
- для начала установки нажмите на кнопку Install.

6. После этого будет произведена установка Anaconda на ваш компьютер (рис. 1.6).

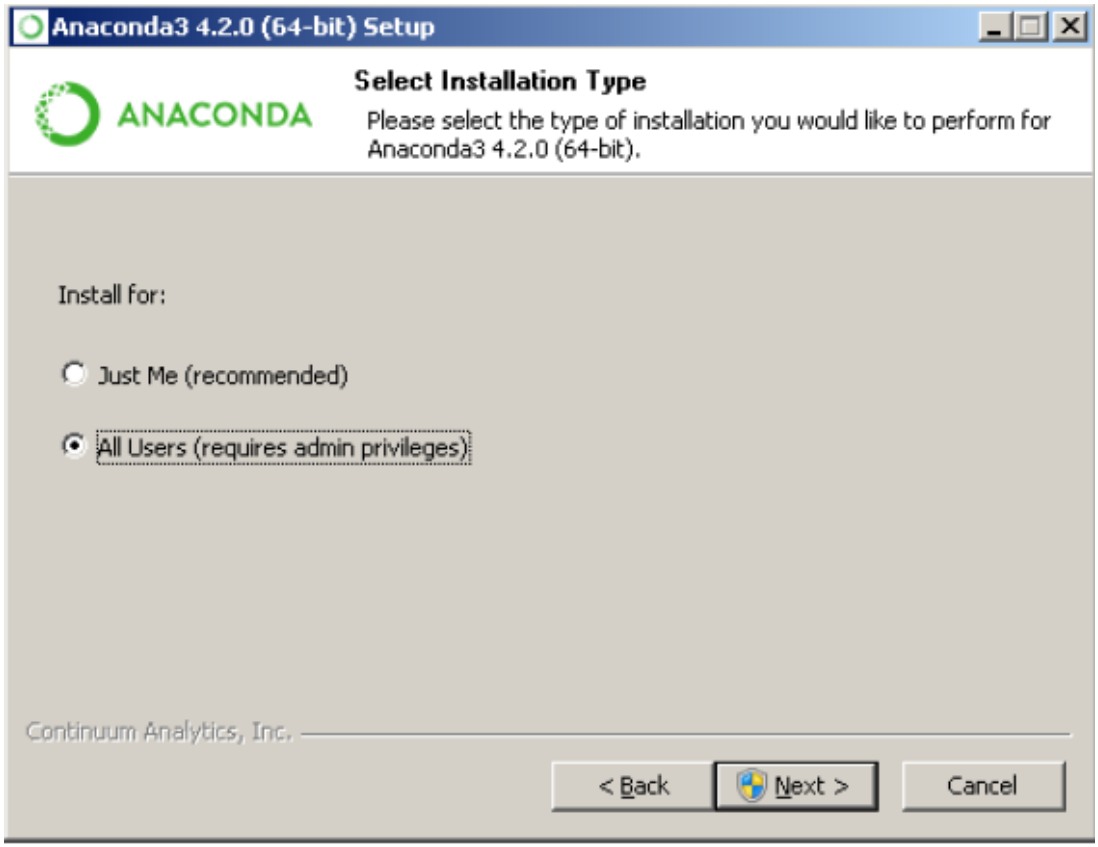

Рис. 1.3. Шаг 3 – выбор опций установки

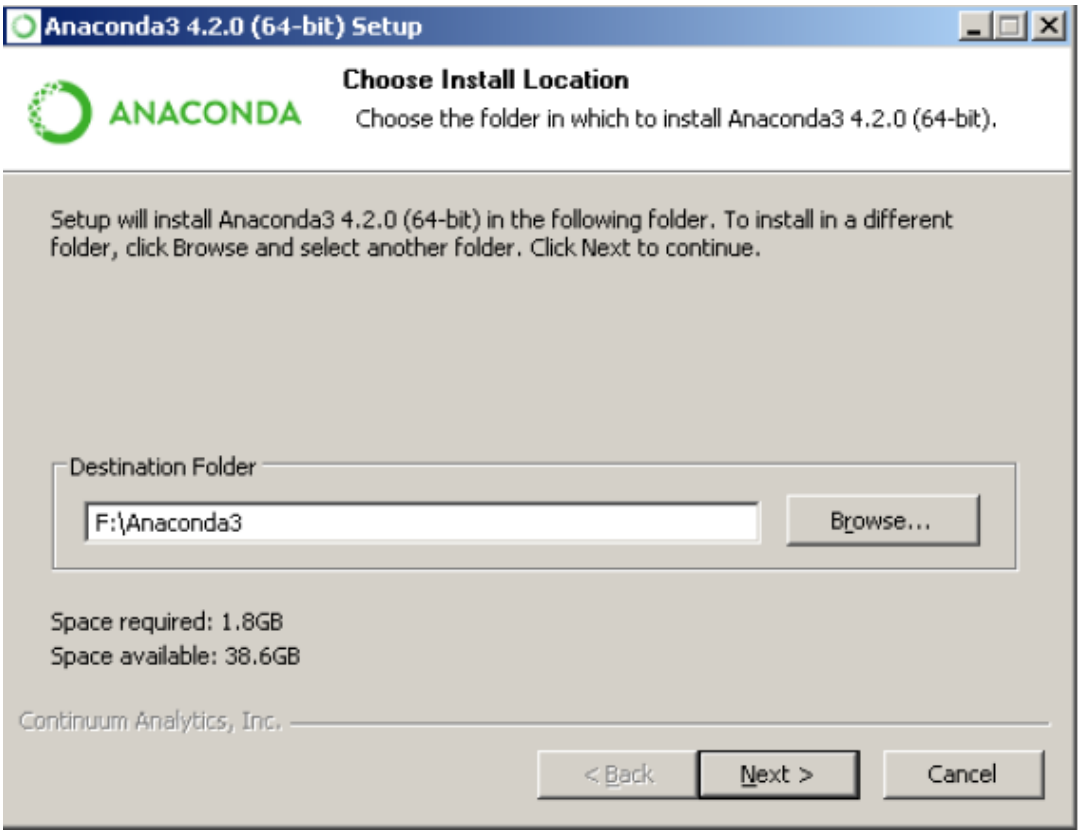

Рис. 1.4. Шаг 4 – выбор пути установки

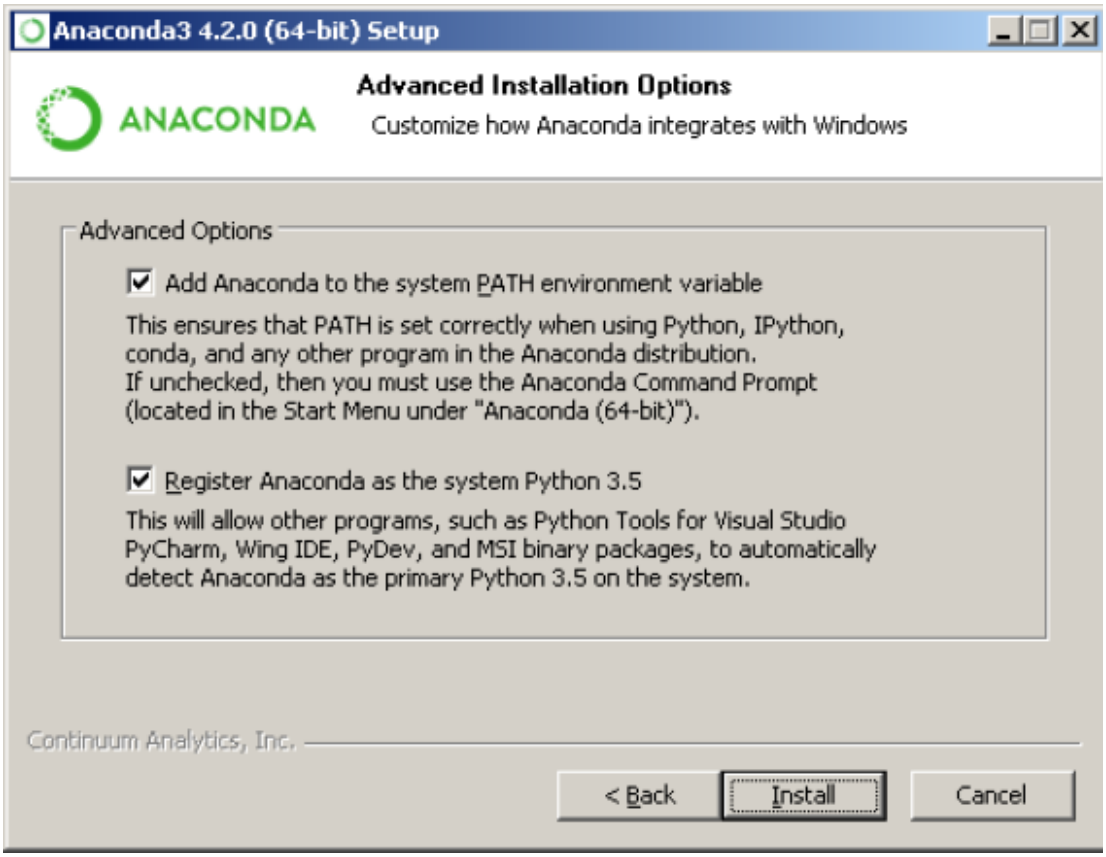

Рис. 1.5. Шаг 5 – выбор дополнительных опций

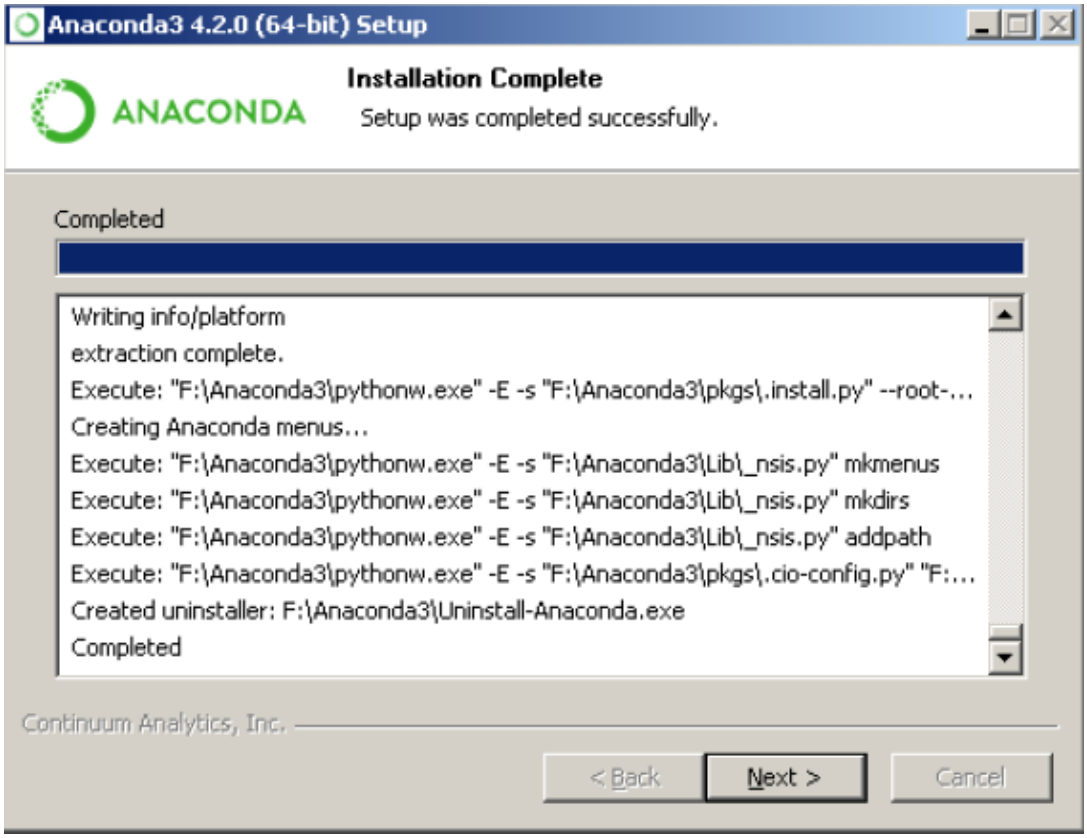

Рис. 1.6. Завершение установки

1.2. Работа с Jupyter Notebook

IPython представляет собой мощный инструмент для работы с языком Python. Базовые компоненты IPython – это интерактивная оболочка с широким набором возможностей и ядро для Jupyter. Jupyter notebook является графической веб-оболочкой для IPython, которая расширяет идею консольного подхода к интерактивным вычислениям.

Основные отличительные особенности данной платформы – это комплексная интроспекция объектов, сохранение истории ввода на протяжении всех сеансов, кэширование выходных результатов, расширяемая система «магических» команд, логирование сессии, дополнительный командный синтаксис, подсветка кода, доступ к системной оболочке, стыковка с отладчиком и Python профайлером. IPython позволяет подключаться множеству клиентов к одному вычислительному ядру и, благодаря своей архитектуре, может работать в параллельном кластере. В Jupyter notebook вы можете разрабатывать, документировать и выполнять приложения на языке Python, он состоит из двух компонентов: веб-приложение, запускаемое в браузере, и ноутбуки – файлы, в которых можно работать с исходным кодом программы, запускать его, вводить и выводить данные и т.п.

Веб приложение позволяет:

- редактировать Python код в браузере, с подсветкой синтаксиса, автоотступами и автодополнением;
- запускать код в браузере;
- отображать результаты вычислений с медиа представлением (схемы, графики);
- работать с языком разметки Markdown и LaTeX.

Ноутбуки – это файлы, в которых сохраняются исходный код, входные и выходные данные, полученные в рамках сессии. Фактически он является записью вашей работы, но при этом позволяет заново выполнить код, присутствующий на нем. Ноутбуки можно экспортировать в форматы PDF, HTML.

1.2.1. Запуск Python

Jupyter Notebook входит в состав Anaconda. Описание процесса установки можно найти в первом уроке. Для запуска Jupyter Notebook в командной строке наберите команду:

<CatPyth>python.exe <CatPyth>cwp.py <CatPyth> <CatPyth>python.exe <CatPyth>Scripts\jupyter-notebook-script.py <CatCur>

где <CatPyth> – каталог Anaconda, <CatCur> – рабочий каталог.

Пример:

C:\Anaconda\python.exe C:\Anaconda\cwp.py C:\Anaconda3 C:\Anaconda\python.exe C:\Anaconda\Scripts\jupyter-notebook-script.py E:\Python\Notebook\

Вставьте команду в файл с расширением bat и поместите на рабочий стол. В результате вы получите ярлык для запуска Python. Щелкните по ярлыку в результате будет запущена оболочка в браузере (рис.1.7).

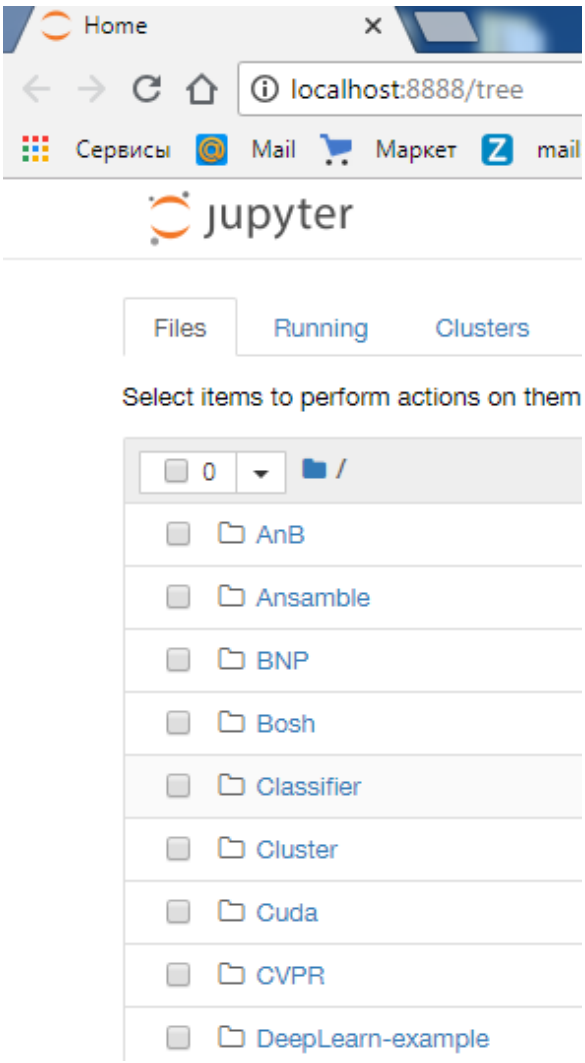

Рис. 1.7. Оболочка Python в браузере

1.2.2. Примеры работы

Будем следовать правилу: лучше один раз увидеть! Рассмотрим несколько примеров, выполнив которые, вы сразу поймете принцип работы с Jupyter notebook. Запустите Jupyter notebook и создайте папку для наших примеров, для этого нажмите на New в правой части экрана и выберите в выпадающем списке Folder (рис. 1.8).

По умолчанию папке присваивается имя Untitled folder, переименуем ее в notebooks: поставьте галочку напротив имени папки и нажмите на кнопку Rename (рис. 1.9).

Зайдите в эту папку и создайте в ней ноутбук, воспользовавшись той же кнопкой New, только на этот раз нужно выбрать Python 3.

В результате будет создан ноутбук (рис.1.10).

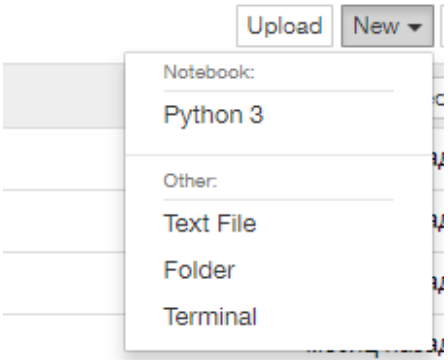

Рис. 1.8. Создание папки

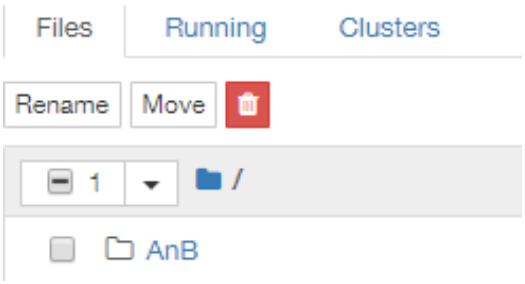

Рис. 1.9. Переименование папки

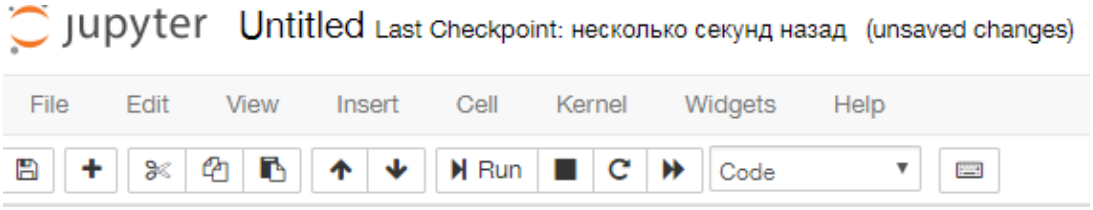

Рис. 1.10. Интерфейс ноутбука

Код на языке Python или текст в нотации Markdown нужно вводить в ячейки (рис.1.11).

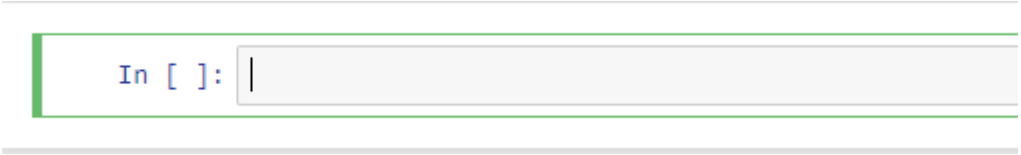

Рис. 1.11. Ячейка ноутбука

Если это код Python, то на панели инструментов нужно выставить свойство Code (рис.1.12).

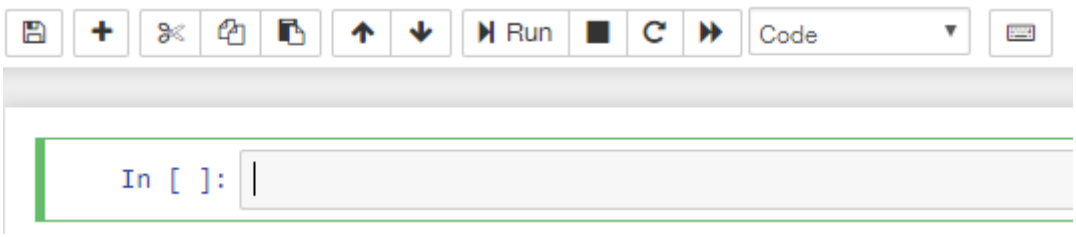

Рис. 1.12. Ячейка ноутбука: свойство Code

Если это Markdown текст – выставить Markdown (рис. 1.13).

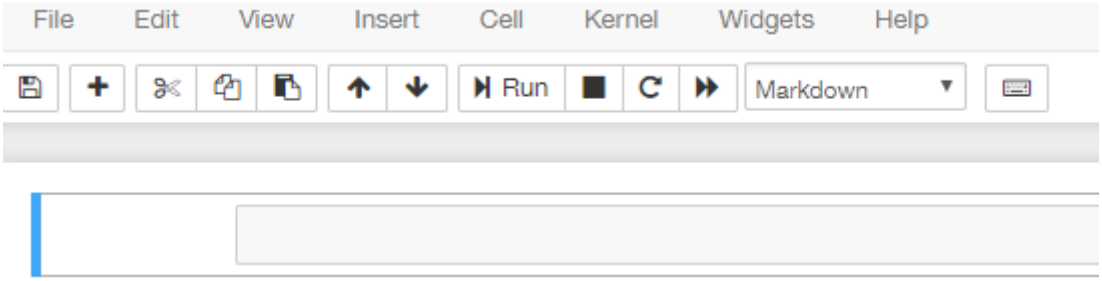

Рис. 1.13. Ячейка ноутбука: свойство Markdown

Для начала решим простую арифметическую задачу: выставите свойство Code, введите в ячейке «2 + 2» без кавычек и нажмите Ctrl+Enter или Shift+Enter, в первом случае введенный вами код будет выполнен интерпретатором Python, во втором – будет выполнен код и создана новая ячейка, которая расположится уровнем ниже так, как показано на рис. 1.14.

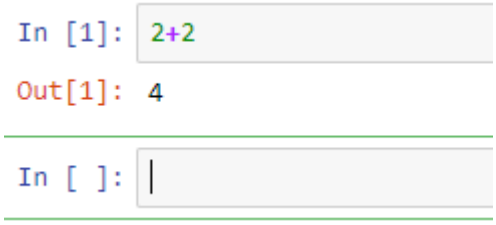

Рис. 1.14. Сложение чисел

Если у вас получилось это сделать, выполните еще несколько примеров.

1.2.3. Основные элементы интерфейса Jupyter notebook

У каждого ноутбука есть имя, оно отображается в верхней части экрана. Для изменения имени нажмите на его текущее имя и введите новое (рис. 1.15).

Рис. 1.15. Имя ноутбука

Из элементов интерфейса можно выделить панель меню (рис. 1.16), панель инструментов (рис. 1.17).

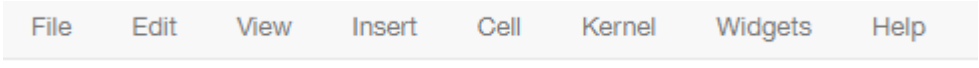

Рис. 1.16. Меню ноутбука

Рис. 1.17. Панель инструментов ноутбука

Ноутбук может находиться в одном из двух режимов – это режим правки (Edit mode) и командный режим (Command mode). Текущий режим отображается на панели меню в правой части, в режиме правки появляется изображение карандаша, отсутствие этой иконки значит, что ноутбук находится в командном режиме.

Для открытия справки по сочетаниям клавиш нажмите Help->Keyboard Shortcuts. В самой правой части панели меню находится индикатор загруженности ядра Python. Если ядро находится в режиме ожидания, то индикатор представляет собой окружность.

Если оно выполняет какую-то задачу, то изображение измениться на закрашенный круг.

1.2.4. Запуск и прерывание выполнения кода

Если ваша программа зависла, то можно прервать ее выполнение выбрав на панели меню пункт Kernel -> Interrupt.

Для добавления новой ячейки используйте Insert->Insert Cell Above и Insert->Insert Cell Below.

Для запуска ячейки используете команды из меню Cell, либо следующие сочетания клавиш:

- Ctrl+Enter – выполнить содержимое ячейки.
- Shift+Enter – выполнить содержимое ячейки и перейти на ячейку ниже.
- Alt+Enter – выполнить содержимое ячейки и вставить новую ячейку ниже.

1.3. Типы и модель данных Python

В данном уроке разберем как Python работает с переменными и определим, какие типы данных можно использовать в рамках этого языка. Подробно рассмотрим модель данных Python, а также механизмы создания и изменения значения переменных.

Если достаточно формально подходить к вопросу о типизации языка Python, то можно сказать, что он относится к языкам с неявной сильной динамической типизацией. Неявная типизация означает, что при объявлении переменной вам не нужно указывать её тип, при явной – это делать необходимо. В качестве примера языков с явной типизацией можно привести Java, C++. Вот как будет выглядеть объявление целочисленной переменной в Java и Python. Пример Java: int a = 1; Python: a = 1.

Также языки бывают с динамической и статической типизацией. В первом случае тип переменной определяется непосредственно при выполнении программы, во втором – на этапе компиляции. Как уже было сказано Python – это динамически типизированный язык, такие языки как C, C#, Java – статически типизированные. Сильная типизация не позволяет производить операции в выражениях с данными различных типов, слабая – позволяет. В

языках с сильной типизацией вы не можете складывать, например, строки и числа, нужно все приводить к одному типу. К первой группе можно отнести Python, Java, ко второй – C и C++.

В Python типы данных можно разделить на встроенные в интерпретатор (built-in) и не встроенные, которые можно использовать при импортировании соответствующих модулей.

К основным встроенным типам относятся:

1. *None* (неопределенное значение переменной)
2. Логические переменные (*Boolean Type*)
3. Числа (*Numeric Type*):
    a. *int* – целое число,
    b. *float* – число с плавающей точкой,
    c. *complex* – комплексное число.
4. Списки (*Sequence Type*):
    a. *list* – список,
    b. *tuple* – кортеж,
    c. *range* – диапазон.
5. Строки (*Text Sequence Type*): *str* – строка.
6. Бинарные списки (*Binary Sequence Types*):
    a. *bytes* – байты,
    b. *bytearray* – массивы байт,
    c. *memoryview* – специальные объекты для доступа к внутренним данным объекта через *protocol buffer*.
7. Множества (*Set Types*):
    a. *set* – множество,
    b. *frozenset* – неизменяемое множество.
8. Словари (*Mapping Types*): *dict* – словарь.

Рассмотрим, как создаются объекты в памяти, их устройство, процесс объявления новых переменных и работу операции присваивания. Для того, чтобы объявить и сразу инициализировать переменную необходимо написать её имя, потом поставить знак равенства и значение, с которым эта переменная будет создана.

Например, строка: b = 5 объявляет переменную b и присваивает ей значение 5.

Целочисленное значение 5 в рамках языка Python по сути своей является объектом. Объект, в данном случае – это абстракция для представления данных, данные – это числа, списки, строки и т.п. При этом, под данными следует понимать как непосредственно сами объекты, так и отношения между ними (об этом чуть позже). Каждый объект имеет три атрибута – это идентификатор, значение и тип.

Идентификатор – это уникальный признак объекта, позволяющий отличать объекты друг от друга, а значение – непосредственно информация, хранящаяся в памяти, которой управляет интерпретатор. При инициализации переменной, на уровне интерпретатора, происходит следующее:

- создается целочисленный объект 5 (можно представить, что в этот момент создается ячейка и число 5 кладется в эту ячейку);
- данный объект имеет некоторый идентификатор – значение 5 и тип целое число;
- посредством оператора «=» создается ссылка между переменной b и целочисленным объектом 5 (переменная b ссылается на объект 5).

Имя переменной не должно совпадать с ключевыми словами интерпретатора Python. Список ключевых слов можно получить непосредственно в программе, для этого нужно подключить модуль keyword и воспользоваться командой keyword.kwlist.

```
import keyword
print "Python keywords: " , keyword.kwlist
```

Проверить является или нет идентификатор ключевым словом можно так:

```
keyword.iskeyword("try")
>>>True
keyword.iskeyword("b")
>>>False
```

Для того, чтобы посмотреть на объект с каким идентификатором ссылается данная переменная, можно использовать функцию *id()*.

```
a = 4
b = 5
id(a)
>>>1829984576
id(b)
>>>1829984592
a = b
id(a)
>>>1829984592
```

Как видно из примера, идентификатор – это некоторое целочисленное значение, посредством которого уникально адресуется объект. Изначально переменная a ссылается на объект 4 с идентификатором 1829984576, переменная *b* – на объект с *id* = 1829984592. После выполнения операции присваивания *a = b*, переменная *a* стала ссылаться на тот же объект, что и *b* (рис. 1.18).

Тип переменной можно определить с помощью функции *type()*. Пример использования приведен ниже.

```
a = 10
b = "hello"
```

```
c = (1 , 2)
type(a)
< class 'int' >
type (b)
< class 'str' >
type(c)
< class 'tuple' >
```

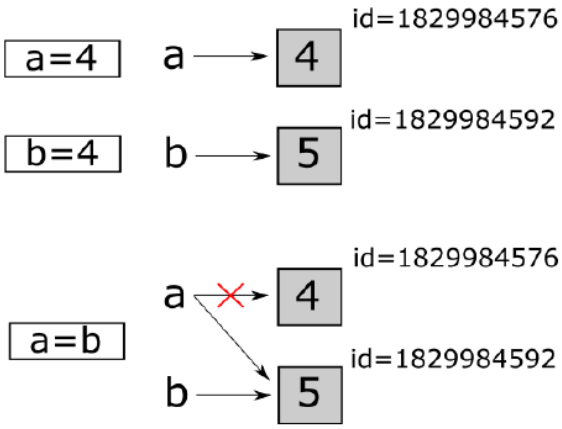

Рис. 1.18. Идентификатор переменной

В Python существуют изменяемые и неизменяемые типы. К неизменяемым (*immutable*) типам относятся:

- целые числа (*int*),
- числа с плавающей точкой (*float*),
- комплексные числа (*complex*),
- логические переменные (*bool*),
- кортежи (*tuple*),
- строки (*str*),
- неизменяемые множества (*frozen set*).

К изменяемым (*mutable*) типам относятся:

- списки (*list*),
- множества (set),
- словари (dict).

Как уже было сказано ранее, при создании переменной, вначале создается объект, который имеет уникальный идентификатор, тип и значение, после этого переменная может ссылаться на созданный объект. Неизменяемость типа данных означает, что созданный объект больше не изменяется. Например, если мы объявим переменную $k = 15$, то будет создан объект со значением 15, типа *int* и идентификатором, который можно узнать с помощью функции *id()*.

```
k = 15
id(k)
>>>1672501744
```

```
type(k)
< class 'int' >
```

Объект с *id* = 1672501744 будет иметь значение 15 и изменить его уже нельзя.

Если тип данных изменяемый, то можно менять значение объекта. Например, создадим список [1, 2], а потом заменим второй элемент на 3.

```
a = [1,2]
id(a)
>>> 47997336
a[1] = 3
a[1,3]
id(a)
>>> 47997336
```

Как видно, объект, на который ссылается переменная *a*, был изменен. Это можно проиллюстрировать рис. 1.19.

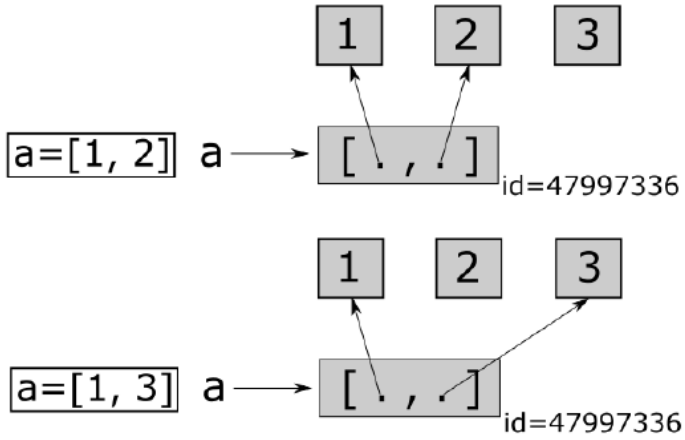

Рис. 1.19. Идентификатор списка

В рассмотренном случае, в качестве данных списка, выступают не объекты, а отношения между объектами. То есть в переменной *a* хранятся ссылки на объекты содержащие числа 1 и 3, а не непосредственно сами эти числа.

1.4. Арифметические операции

Язык Python, благодаря наличию огромного количества библиотек для решения разного рода вычислительных задач, является конкурентом таким пакетам как Matlab и Octave. Запущенный в интерактивном режиме, он, фактически, превращается в мощный калькулятор. В этом уроке речь пойдет об арифметических операциях, доступных в данном языке. Как было сказано в предыдущем уроке, посвященному типами и модели данных Python, в этом языке существует три встроенных числовых типа данных:

- целые числа (*int*),
- вещественные числа (*float*),
- комплексные числа (*complex*).

Если в качестве операндов некоторого арифметического выражения используются только целые числа, то результат тоже будет целое число. Исключением является операция деления, результатом которой является вещественное число. При совместном использовании целочисленных и вещественных переменных, результат будет вещественным.

1.4.1. Арифметические операции с целыми и вещественными числами

Все эксперименты будем проводить в Python, запущенном в интерактивном режиме.

Сложение.

Складывать можно непосредственно сами числа, например, 3 + 2, либо переменные, но они должны предварительно быть проинициализированы.

```
a = 3
b = 2
a + b
>>>5
```

Результат операции сложения можно присвоить другой переменной:

```
a = 3
b = 2
c = a + b
print (c)
>>>5
```

Либо ей же самой, в таком случае можно использовать полную или сокращенную запись, полная выглядит так:

```
a = 3
b = 2
a = a + b
print (a)
>>>5
```

сокращенная так:

```
a = 3
b = 2
a += b
print (a)
>>> 5
```

Все перечисленные выше варианты использования операции сложения могут быть применены для всех нижеследующих операций.

Вычитание.

```
4 - 2
>>>2
a = 5
b = 7
a - b
```

```
>>> – 2
```
Умножение.
```
5 * 8
>>>40
a = 4
a *= 10
print (a)
>>>40
```
Деление.
```
9 / 3
>>>3.0
a = 7
b = 4
a / b
>>>1.7 5
```
Получение целой части от деления.
```
9 // 3
>>>3
a = 7
b = 4
a // b
>>>1
```
Получение дробной части от деления.
```
9 % 5
>>>4
a = 7
b = 4
a % b
>>>3
```
Возведение в степень.
```
5 ** 4
>>>625
a = 4
b = 3
a ** b
>>> 64
```

1.4.2. Работа с комплексными числами

Для создания комплексного числа можно использовать функцию *complex(a, b)*, в которую, в качестве первого аргумента, передается действительная часть, в качестве второго – мнимая. Либо записать число в виде *a + bj*. Рассмотрим несколько примеров.

Создание комплексного числа.
```
z = 1 + 2j
print (z)
```

```
>>> (1 + 2j)
x = complex (3 , 2)
print (x)
>>> (3 + 2j)
```

Комплексные числа можно *складывать, вычитать, умножать, делить и возводить в степень*.

```
x + z
>>> (4 + 4j)
x – z
>>> (2 + 0j)
>>> x * z
>>> (– 1 + 8j)
x / z
>>> (1.4 – 0.8j)
x ** z
>>> (– 1.1122722036363393 – 0.0126351853553352 08j)
x ** 3
>>> (– 9 + 46j)
```

У комплексного числа можно *извлечь действительную и мнимую части*.

```
x = 3 + 2j
x.real
>>>3.0
x.imag
>>>2.0
```

Для *получения комплексносопряженного числа* необходимо использовать метод conjugate().

```
x.conjugate()
>>> (3 – 2j)
```

1.4.3. Битовые операции

В Python доступны битовые операции, их можно производить над целыми числами.

Побитовое И (AND).

```
p = 9
q = 3
p & q
>>>1
```

Побитовое ИЛИ (OR).

```
p | q
>>>11
```

Побитовое Исключающее ИЛИ (XOR).

```
p ^ q
>>>10
```

Инверсия.

```
~ p
>>>– 10
```

Сдвиг вправо и влево.

```
p << 1
>>>18
p >> 1
>>>4
```

1.4.4. Представление чисел в других системах счисления

В своей повседневной жизни мы используем десятичную систему исчисления, но программируя, очень часто, приходится работать с шестнадцатеричной, двоичной и восьмеричной.

Представление числа в шестнадцатеричной системе.

```
m = 124504
hex (m)
>>> '0x1e658'
```

Представление числа в восьмеричной системе.

```
oct (m)
>>>'0o363130'
```

Представление числа в двоичной системе.

```
bin(m)
>>>'0b11110011001011000'
```

1.4.5. Библиотека (модуль) math

В стандартную поставку Python входит библиотека math, в которой содержится большое количество часто используемых математических функций. Для работы с данным модулем его предварительно нужно импортировать: import math.

Рассмотрим наиболее часто используемые функции.

math.ceil(x)

Возвращает ближайшее целое число большее, чем x.

```
math.ceil(3.2)
>>>4
```

math.fabs(x)

Возвращает абсолютное значение числа.

```
math.fabs(– 7)
>>>7.0
```

math.factorial(x)

Вычисляет факториал x.

```
math.factorial(5)
>>>120
```

math.floor(x)

Возвращает ближайшее целое число меньшее, чем x.

```
math.floor(3.2)
>>>3
```

math.exp(x)

Вычисляет e**x.

```
math.exp(3)
>>>20.08553692318766
```

math.log2(x)

Логарифм по основанию 2.

math.log10(x)

Логарифм по основанию 10.

math.log(x[, base])

По умолчанию вычисляет логарифм по основанию e, дополнительно можно указать основание логарифма.

```
math.log2(8)
>>>3.0
math.log10(1000)
>>>3.0
math.log(5)
>>>1.609437912434100
math.log(4 , 8)
>>>0.666666666666666
```

math.pow(x, y)

Вычисляет значение x в степени y.

```
math.pow(3 , 4)
>>>81.0
```

math.sqrt(x)

Корень квадратный от x .

```
math.sqrt(25)
>>>5.0
```

Тригонометрические функции, их мы оставим без примера.

math.cos(x)

math.sin(x)

math.tan(x)

math.acos(x)

math.asin(x)

math.atan(x)

И напоследок пару констант.

math.pi

Число пи.

math.e

Число e.

Помимо перечисленных, модуль math содержит ещё много различных функций, за более подробной информацией можете обратиться на официальный сайт (https://docs.python.org/3/library/math.html).

1.5. Условные операторы и циклы

В этом уроке рассмотрим оператор ветвления *if* и операторы цикла *while* и *for*. Основная цель – это дать общее представление об этих операторах и на простых примерах показать базовые принципы работы с ними.

1.5.1. Условный оператор ветвления if

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

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*

Условный оператор ветвления 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*.

Условный оператор ветвления 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*.

1.5.2. 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")
```

1.5.3. 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 распечатать 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.

На этом закончим краткий обзор операторов ветвления и цикла.

1.5.4. Операторы break и continue

При работе с циклами используются операторы break и continue. Оператор break предназначен для досрочного прерывания работы цикла while.

```
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.

1.6. Работа со списками list

Одна из ключевых особенностей Python, благодаря которой он является таким популярным – это простота. Особенно подкупает простота работы с

различными структурами данных – списками, кортежами, словарями и множествами.

Список (list) – это структура данных для хранения объектов различных типов. Если вы использовали другие языки программирования, то вам должно быть знакомо понятие массива. Так вот, список очень похож на массив, только, как было уже сказано выше, в нем можно хранить объекты различных типов. Размер списка не статичен, его можно изменять. Список по своей природе является изменяемым типом данных. Про типы данных можно подробно прочитать в третьем уроке. Переменная, определяемая как список, содержит ссылку на структуру в памяти, которая в свою очередь хранит ссылки на какие-либо другие объекты или структуры.

Как уже было сказано выше, список является изменяемым типом данных. При его создании, в памяти резервируется область, которую можно условно назвать некоторым «контейнером», в котором хранятся ссылки на другие элементы данных в памяти. В отличии от таких типов данных как число или строка, содержимое «контейнера» списка можно менять. Для того, чтобы лучше визуально представлять себе этот процесс взгляните на рис. 1.20.

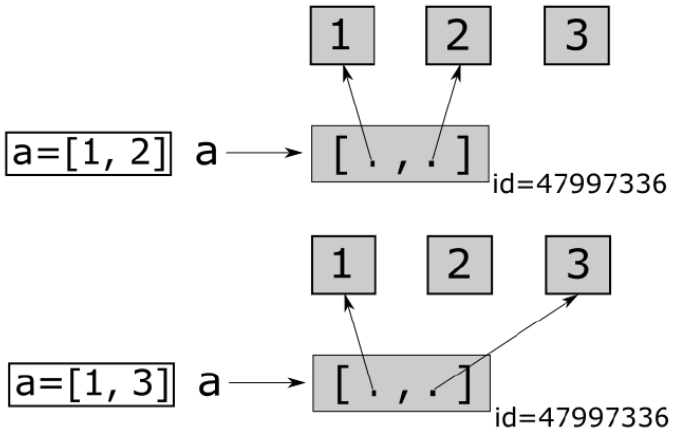

Рис. 1.20. Список: изменяемый тип данных

Изначально был создан список содержащий ссылки на объекты 1 и 2, после операции a[1] = 3, вторая ссылка в списке стала указывать на объект 3.

Создание, изменение, удаление списков и работа с его элементами

Создать список можно одним из следующих способов.

```
>>> a = []
>>> type (a)
< class 'list' >
>>> b = list ()
>>> type (b)
< class 'list' >
```

Также можно создать список с заранее заданным набором данных.

```
>>> a = [ 1 , 2 , 3 ]
>>> type (a)
< class 'list' >
```

Если у вас уже есть список и вы хотите создать его копию, то можно воспользоваться следующим способом:

```
>>> a = [ 1 , 3 , 5 , 7 ]
>>> b = a[:]
>>> print (a)
[ 1 , 3 , 5 , 7 ]
>>> print (b)
[ 1 , 3 , 5 , 7 ]
```

или сделать это так:

```
>>> a = [ 1 , 3 , 5 , 7 ]
>>> b = list (a)
>>> print (a)
[ 1 , 3 , 5 , 7 ]
>>> print (b)
[ 1 , 3 , 5 , 7 ]
```

В случае, если вы выполните простое присвоение списков друг другу, то переменной *b* будет присвоена ссылка на тот же элемент данных в памяти, на который ссылается *a*, а не копия списка *a*. Т.е., если вы будете изменять список a, то и b тоже будет меняться.

```
>>> a = [ 1 , 3 , 5 , 7 ]
>>> b = a
>>> print (a)
[ 1 , 3 , 5 , 7 ]
>>> print (b)
[ 1 , 3 , 5 , 7 ]
>>> a[ 1 ] = 10
>>> print (a)
[ 1 , 10 , 5 , 7 ]
>>> print (b)
[ 1 , 10 , 5 , 7 ]
```

Добавление элемента в список осуществляется с помощью метода *append()* .

```
>>> a = []
>>> a.append(3)
>>> a.append("hello")
>>> print (a)
[ 3 , 'hello' ]
```

Для удаления элемента из списка, в случае, если вы знаете его значение, используйте метод *remove(x)*, при этом будет удалена первая ссылка на данный элемент.

```
>>> b = [ 2 , 3 , 5 ]
```

```
>>> print (b)
[ 2 , 3 , 5 ]
>>> b.remove(3)
>>> print (b)
[ 2 , 5 ]
```

Если необходимо удалить элемент по его индексу, воспользуйтесь командой *del имя_списка[индекс]*.

```
>>> c = [ 3 , 5 , 1 , 9 , 6 ]
>>> print (c)
[ 3 , 5 , 1 , 9 , 6 ]
>>> del c[ 2 ]
>>> print (c)
[ 3 , 5 , 9 , 6 ]
```

Изменить значение элемента списка, зная его индекс, можно, обратившись напрямую к нему.

```
>>> d = [ 2 , 4 , 9 ]
>>> print (d)
[ 2 , 4 , 9 ]
>>> d[ 1 ] = 17
>>> print (d)
[ 2 , 17 , 9 ]
```

Очистить список можно просто заново его проинициализировав, так как будто вы его вновь создаете. Для получения доступа к элементу списка укажите индекс этого элемента в квадратных скобках.

```
>>> a = [ 3 , 5 , 7 , 10 , 3 , 2 , 6 , 0 ]
>>> a[ 2 ]
7
```

Можно использовать отрицательные индексы, в таком случае счет будет идти с конца, например, для доступа к последнему элементу списка можно использовать вот такую команду:

```
>>> a[ - 1 ]
0
```

Для получения из списка некоторого подсписка в определенном диапазоне индексов укажите начальный и конечный индекс в квадратных скобках, разделив их двоеточием:

```
>>> a[ 1 : 4 ]
[ 5 , 7 , 10 ]
```

Методы списков

list.append(x)

Добавляет элемент в конец списка. Ту же операцию можно сделать так: a[len(a):] = [x].

```
>>> a = [ 1 , 2 ]
```

```
>>> a.append(3)
>>> print (a)
[ 1 , 2 , 3 ]
```

list.extend(L)

Расширяет существующий список за счет добавления всех элементов из списка *L*. Эквивалентно команде *a[len(a):] = L*.

```
>>> a = [ 1 , 2 ]
>>> b = [ 3 , 4 ]
>>> a.extend(b)
>>> print (a)
[ 1 , 2 , 3 , 4 ]
```

list.insert(i, x)

Вставить элемент *x* в позицию *i*. Первый аргумент – индекс элемента, после которого будет вставлен элемент *x*.

```
>>> a = [ 1 , 2 ]
>>> a.insert(0 , 5)
>>> print (a)
[ 5 , 1 , 2 ]
>>> a.insert(len (a), 9)
>>> print (a)
[ 5 , 1 , 2 , 9 ]
```

list.remove(x)

Удаляет первое вхождение элемента *x* из списка.

```
>>> a = [ 1 , 2 , 3 ]
>>> a.remove(1)
>>> print (a)
[ 2 , 3 ]
```

list.pop([i])

Удаляет элемент из позиции *i* и возвращает его. Если использовать метод без аргумента, то будет удален последний элемент из списка.

```
>>> a = [ 1 , 2 , 3 , 4 , 5 ]
>>> print (a.pop(2))
3
>>> print (a.pop())
5
>>> print (a)
[ 1 , 2 , 4 ]
```

list.clear()

Удаляет все элементы из списка. Эквивалентно *del a[:]*.

```
>>> a = [ 1 , 2 , 3 , 4 , 5 ]
>>> print (a)
[ 1 , 2 , 3 , 4 , 5 ]
>>> a.clear()
```

```
>>> print (a)
[]
```

list.index(x[, start[, end]])

Возвращает индекс элемента.

```
>>> a = [ 1 , 2 , 3 , 4 , 5 ]
>>> a.index(4)
3
```

list.count(x)

Возвращает количество вхождений элемента x в список.

```
>>> a = [ 1 , 2 , 2 , 3 , 3 ]
>>> print (a.count(2))
2
```

list.sort(key=None, reverse=False)

Сортирует элементы в списке по возрастанию. Для сортировки в обратном порядке используйте флаг *reverse=True*. Дополнительные возможности открывает параметр *key*, за более подробной информацией обратитесь к документации.

```
>>> a = [ 1 , 4 , 2 , 8 , 1 ]
>>> a.sort()
>>> print (a)
[ 1 , 1 , 2 , 4 , 8 ]
```

list.reverse()

Изменяет порядок расположения элементов в списке на обратный.

```
>>> a = [ 1 , 3 , 5 , 7 ]
>>> a.reverse()
>>> print (a)
[ 7 , 5 , 3 , 1 ]
```

list.copy()

Возвращает копию списка. Эквивалентно *a[:]* .

```
>>> a = [ 1 , 7 , 9 ]
>>> b = a.copy()
>>> print (a)
[ 1 , 7 , 9 ]
>>> print (b)
[ 1 , 7 , 9 ]
>>> b[ 0 ] = 8
>>> print (a)
[ 1 , 7 , 9 ]
>>> print (b)
[ 8 , 7 , 9 ]
```

List Comprehensions

List Comprehensions чаще всего на русский язык переводят как абстракция списков или списковое включение, является частью синтаксиса языка, которая предоставляет простой способ построения списков. Проще всего работу *list comprehensions* показать на примере. Допустим, вам необходимо создать список целых чисел от 0 до *n*, где *n* предварительно задается. Классический способ решения данной задачи выглядел бы так:

```
n = int (input ())
a = []
for i in range (n):
a.append(i)
print (a)
```

Использование *list comprehensions* позволяет сделать это значительно проще:

```
n = int (input ())
a = [i for i in range (n)]
print (a)
```

Или, вообще, вот так, в случае если вам не нужно больше использовать n:

```
a = [i for i in range (int (input ()))]
print (a)
```

1.7. Практическое занятие № 1. Предварительная обработка входных данных

Цель занятия:

Изучить основные элементы языка Python.

Изучить способы загрузки данных в структуру DataFrame из файла.

Уметь заполнять пропущенные данные в таблицах.

Учебное задание

Крушение парохода «Титаник» представляет крупнейшую морскую катастрофу, произошедшей в ночь с 14 на 15 апреля 1912 года в северной части Атлантического океана. Трагедия случилась под конец пятого дня следования «Титаника» по трансатлантическому маршруту Саутгемптон – Нью-Йорк. В 23часа 40 минут 14 апреля во время первого рейса самый большой на тот момент океанский лайнер с 2208 людьми на борту по касательной столкнулся с айсбергом и получил серьёзные повреждения обшивки корпуса. Спустя 2 часа 40 минут полностью ушёл под воду. Катастрофа унесла жизни, по разным данным, от 1495 до 1635 человек.

Выполните анализ вероятности спасения пассажиров Титаника.

К задаче прилагается файл: *data.csv* – набор данных на основании, которого будет строиться и проверятся модель (с разделителем чисел – запятая, разделитель десятичной части – точка).

Пояснения по некоторым полям:

PassengerId – идентификатор пассажира;

Survival – поле в котором указано спасся человек (1) или нет (0);

Pclass – содержит социально-экономический статус: высокий, средний, низкий;

Name – имя пассажира;

Sex – пол пассажира;

Age – возраст пассажира;

SibSp – содержит информацию о количестве родственников 2-го порядка (муж, жена, братья, сестры);

Parch – содержит информацию о количестве родственников на борту 1-го порядка (мать, отец, дети);

Ticket – номер билета;

Fare – цена билета;

Cabin – каюта;

Embarked – порт посадки: C – Cherbourg, Q – Queenstown, S – Southampton.

Технология выполнение учебного здания

Ввод данных

Создайте новый файл notebook. Для этого нажмите кнопку New в правом верхнем углу и выберите Python 3. Нажмите на заголовок и в открывшемся окне задайте имя Pract01.

Загружаем необходимые для решения задачи библиотеки pandas, numpy, matplotlib.

Для анализа понадобятся модули pandas, numpy и sklearn. С помощью pandas и numpy мы проведем начальный анализ данных, а sklearn поможет в вычислении прогнозной модели.

Библиотека pandas дает возможность строить сводные таблицы, выполнять группировки, предоставляет удобный доступ к табличным данным, а наличие пакета matplotlib дает возможность рисовать графики на полученных наборах данных

Библиотека numpy включает поддержку больших многомерных массивов и матриц, а также высокоуровневых математических функций для операций с этими массивами.

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

Для начала загрузим набор данных и посмотрим, как он выглядит.

Основными структурами хранения данных в Pandas являются Series и DataFrame. Series – это проиндексированный одномерный массив значений. Он похож на простой словарь, где имя элемента будет соответствовать индексу, а значение – значению записи.

DataFrame – это таблица, представляющая проиндексированный многомерный массив значений.

Создать структуру DataFrame можно с помощью конструктора либо это можно сделать функцией read_csv():

```
tall = pd.read_csv("../Titanic/Data/data.csv")
tall.head(8)
```

Обратите внимание, что вызов функции read_csv() осуществляется через библиотеку pandas (сокращенно pd).

Очень важными аргументами функции read_csv() являются sep и decimal:

sep – символ, задающий разделитель данных в файле, по умолчанию запятая (',');

decimal – символ, который используется для отделения десятичной части от целой в числах с плавающей запятой (float). По умолчанию это точка ('.').

Пример использование функции для чтения данных в кодировке, принятой в РФ:

```
tall = pd.read_csv("../SMS/Data/2itest.csv", sep=';', decimal=',')
```

Функция head(n) структуры DataFrame используется для отображения заголовка файла, n – число строк в заголовке.

Прежде всего, определим размеры таблицы tall, которая в Python описывается структурой DataFrame. С помощью атрибута shape[] структуры DataFrame можно определить число строк и столбцов. Первый элемент массива shape (индекс 0) равен числу строк, а второй (индекс 1) – числу столбцов таблицы DataFrame. Для распечатки характеристик таблицы используйте оператор:

```
print( "число строк=", tall.shape[0], "число столбцов=", tall.shape[1])
```

В результате получаем «*число строк = 891, число стролбцов = 12*».

Результат выполнения операции показан на рис. 1.21.

Проверка данных

Полученные данные могут иметь пропуски (незаполненные значения). Нам нужно найти такие значения и либо их заполнить, либо удалить.

Сначала получим информацию о типах и количестве пропущенных значений данных, которые хранятся в нашей таблице tall. Для этого используем функцию info() структуры DataFrame.

Код команды:

```
tall.info()
```

Результат представлен на рис. 1.22.

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S

Рис. 1.21. Заголовок таблицы tall

Как видно из рисунка из 12 показателей, хранящихся в таблице, пять показателей имеют тип object, т.е. не числовой тип. Такие показатели нужно либо преобразовать в числовые показатели, либо удалять.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

Рис. 1.22. Информация о таблице tall

Мы можем увидеть, что пропуски есть для полей Age, Cabin, Embarked. Два пропусков найдено в поле Embarked, 177 значений пропущено для поля Age и 687 значений показателей не определены для поля Cabin.

Изучим структуру таблицы. Будем использовать для этого цикл:

```
cols=tall.columns
for col in cols:
    s=tall[col].dtypes
    print(col, s)
```

Результат представлен на рис. 1.23.

```
PassengerId int64
Survived int64
Pclass int64
Name object
Sex object
Age float64
SibSp int64
Parch int64
Ticket object
Fare float64
Cabin object
Embarked object
```

Рис. 1.23. Информация о типах полей таблицы

Для определения числа уникальных значений этого используйте метод nunique() объекта DataFrame. Уникальные значения будем определять только для текстовых и числовых целых полей:

```
cols=tall.columns
for col in cols:
    s=tall[col].dtypes
    if s == 'object' or s == 'int64':
        n=tall[col].nunique()
        print(col, s, n)
```

Результат представлен на рис. 1.24.

```
PassengerId int64 891
Survived int64 2
Pclass int64 3
Name object 891
Sex object 2
SibSp int64 7
Parch int64 7
Ticket object 681
Cabin object 147
Embarked object 3
```

Рис. 1.24. Информация о типах и количестве уникальных значений полей таблицы

Таким образом, несколько полей (Survived, Pclass, Sex, SibSb, Parch, Embarked) имеют небольшое число уникальных значений.

Далее вычислим для каждого числового поля простейшие статистические характеристики: среднее значение, медиану, максимальное и минимальное значение. Для этого используем соответствующие методы DataFrame.

```
for col in cols:
    s=tall[col].dtypes
    if s != 'object':
        mean=tall[col].mean()
        median=tall[col].median()
        maxx=tall[col].max()
        minn=tall[col].min()
        print(col, "mean =", mean ,"median=", median, "max =", maxx, "min=",
        minn)
```

Результат представлен на рис. 1.25.

Рассмотрим поле Cabin, которое содержит номера кают. Для определения, у каких пассажиров отсутствует информация о номерах кают, Используйте метод notnull() структуры DataFrame, который выбирает записи с пропущенными значениями указанного поля.

```
PassengerId mean = 446.0 median= 446.0 max = 891 min= 1
Survived mean = 0.3838383838383838 median= 0.0 max = 1 min= 0
Pclass mean = 2.308641975308642 median= 3.0 max = 3 min= 1
Age mean = 29.69911764705882 median= 28.0 max = 80.0 min= 0.42
SibSp mean = 0.5230078563411896 median= 0.0 max = 8 min= 0
Parch mean = 0.38159371492704824 median= 0.0 max = 6 min= 0
Fare mean = 32.2042079685746 median= 14.4542 max = 512.3292 min= 0.0
```

Рис. 1.25. Статистические показатели полей таблицы

Используйте оператор, который выбирает значения поля PassengerId, которым соответствуют неопреленные значения поля Cabin.

```
trCab=tall['PassengerId'][tall['Cabin'].notnull()]
trCab.head(5)
```

Первые пять найденных записей показаны на рис. 1.26.

```
1      2
3      4
6      7
10    11
11    12
Name: PassengerId, dtype: int64
```

Рис. 1.26. Заголовок о таблице trCab

Число неопределенных значений можно определить с помощью оператора shape структуры DataFrame.

```
print(trCab.shape[0])
```

В итоге получаем подтверждение уже известной информации: заполнено всего 204 записи и 890, на основании этого можно сделать вывод, что данное поле при анализе может быть бесполезным.

Следующее поле, которое мы разберем, будет поле с возрастом (Age). Посмотрим на сколько оно заполнено:

```
tall['PassengerId'][tall['Age'].notnull()].count()
```

Интересно познакомится с альтернативной формой записи выше представленного оператора:

```
tall.PassengerId[tall.Age.notnull()].count()
```

В этой форме имя поля записи задается не в апострофах, а через точку и без квадратных скобок.

Результат выполнения операции показывает, что данное поле практически все заполнено (714 непустых записей), хотя и есть пустые значения, которые не определены.

Пустые значения можно заполнить медианным значением возраста пассажиров. Это позволит создать более адекватную модель. Вычислим медиану возраста пассажиров. Для этого используем оператор:

```
medianAge = tall.Age.median()
print( medianAge)
```

Для замены пустых значений медианным значением используйте оператор:

```
tall.Age[tall.Age.isnull()] =medianAge
```

Разберем поле Embarked, которое содержит порт посадки. Проверим, есть ли такие пассажиры, у которых порт не указан. Для этого выполним оператор:

```
tall[tall.Embarked.isnull()].shape[0]
```

Итак, у нас нашлось 2 таких пассажира. Давайте присвоим этим пассажирам порт, в котором село больше всего людей.

Для этого нам нужно опять сформировать сводную таблицу, показывающей число пассажиров, севших на корабль в каждом порту. Можно использовать метод pivot_table библиотеки pandas.

Синтаксис функции:

pivot_table(data, values=None, index=None, columns=None, aggfunc='mean', fill_value=None, margins=False, dropna=True, margins_name='All')

Функция *pivot_table* возвращает сводную таблицу в стиле *DataFrame*.

Аргумент *data* – это исходная таблица типа *DataFrame*.

Аргумент values задает имя столбцы, который агрегируется. Для нашей таблицы это будет столбец *PassengerId*.

Аргумент *index* задает имя столбца, по которому осуществляется группировка. Используется для идентификации строк агрегированной таблицы. В нашем случае будем использовать *Pclass*.

Аргумент *columns* задает имя столбца, по которому осуществляется группировка. Используется для идентификации столбцов агрегированной таблицы. В нашем случае будем использовать *Survived*.

Аргумент *aggfunc* – определяет имя функции агрегирования. В качестве имени может использоваться *mean* (среднее значение), *sum* (сумма значений), *count* (количество значение).

Аргумент *fill_value* – позволяет заполнять заданными значениями ячейки таблицы типа *NAN* (неопределенное значение).

В качестве столбца, который агрегируется используйте поле *PassengerId,* а в качестве столбца, по которому осуществляется группировка, поле *Embarked*. функция агрегирования должна подсчитывать количество значений поля *PassengerId*.

Таким образом, сводная таблица формируется с помощью следующей команды:

```
MaxPassEmb=tall.pivot_table(values='PassengerId', index='Embarked', aggfunc='count')
MaxPassEmb.head()
```

В результате формируется таблица, показанная на рис. 1.27.

```
: Embarked
  C    168
  Q     77
  S    644
  Name: PassengerId, dtype: int64
```

Рис. 1.27. Количество пассажиров, севших в разных портах

Для создания сводной таблицы также может использоваться функция groupby(), которая возвращает объект GroupBy. В нашем случае это сводная таблица. Функции типа count(), max(), min(), mean(), first(), last() могут быть быстро применены к объекту GroupBy для получения сводной статистики для каждой группы.

```
MaxPassEmb = tall.groupby('Embarked').count()['PassengerId']
MaxPassEmb
```

Результат будет аналогичен, показанному на рис. 1.27.

```
Port=MaxPassEmb[MaxPassEmb == MaxPassEmb.max()].index[0]
print(Port)
```

Далее присваиваем пропущенным данным название порта, в котором осуществило посадку максимальное число пассажиров.

```
tall.Embarked[tall.Embarked.isnull()] = Port
```

Из оставшихся полей нам интересна только цена (*Fare*), т.к. она в какой-то мере определяем ранжирование внутри классов поля *Pclass*. Например, люди внутри среднего класса могут быть разделены на тех, кто ближе к первому(высшему) классу, а кто к третьему(низший). Проверим это поле на пустые значения и, если таковые имеются, заменим цену медианой по цене из всей выборки:

```
tall.PassengerId[tall.Fare.isnull()].shape[0]
```

В нашем случае пустых записей нет.

Сохраним полученную таблицу в файле *Prak01.scv*. Для этого используйте функцию *to_csv()*.

```
tall.to_csv('../Titanic/Data/Prak01.csv', index=False, sep=';', float_for-
    mat='%.5f', decimal=',' )
```

Обратите внимание, что мы сохраняем таблицу в формате, который используется в России: в качестве разделителей чисел и десятичной части используется точка с запятой и запятая.

Для проверки правильности записанного файла используется оператор *tall.head(5)*.

Результат представлен на рис. 1.28.

Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	male	28	1	0	7.2500	S
1	1	female	28	1	0	71.2833	C
1	3	female	28	0	0	7.9250	S
1	1	female	28	1	0	53.1000	S
0	3	male	28	0	0	8.0500	S

Рис. 1.28. Таблица после предварительной обработки

Преобразование тестовых полей в числовые поля

Предварительный анализ данных завершен, и по его результатам у нас получилась некая выборка, в которой содержатся несколько полей и можно преступить к построению модели, если бы не одно «но»: наши данные содержат не только числовые, но и текстовые данные.

Поэтому перед тем, как строить модель, нужно превратить наши текстовые поля в числовые. В реальных данных не так редки категориальные признаки, принимающие дискретные значения, такие как да/нет или январь/февраль/.../декабрь. В нашем случае таким категориальными признаками (показателями) являются поля Sex и Embarked.

Преобразовать категориальные признаки в числовые можно с помощью класса LabelEncoder(). Суть функции преобразования данного класса заключается в том, что на вход ей подается список значений, который надо закодировать (например, текстовые переменные), а на выходе получается список индексов, каждый из которых являются кодом уникальных значений поданного на вход списка. Первым делом нужно объявить класс LabelEncoder.

```
from sklearn.preprocessing import LabelEncoder
la = LabelEncoder()
```

Показатель Sex представляет массив с двумя уникальными значениями female и male. Таким образом, если обозначить один индексом 0, а другой индексом 1, то мы легко можем преобразовать это поле в числовое. Для начала проверим как преобразуются наши данные. Для этого будем использовать функцию *fit_transform().* В результате получим массив, который отформатируем как структуру Series.

Структура Series представляет из себя объект, похожий на одномерный массив (список, например), но отличительной его чертой является наличие ассоциированных меток, так называемых индексов, вдоль каждого элемента из списка.

В строковом представлении объекта Series, индекс находится слева, а сам элемент справа. Объект Series напоминает словарь, где ключом является индекс, а значением сам элемент.

Для того чтобы определить количество уникальных значений используйте функцию *value_counts()*, которая является методом объекта Series. Функция *value_counts()* возвращает объект Series, который содержит количество уникальных значений массива *mapped_sex*. Для графически отображения используйте функцию *plot.barh()*.

```
mapped_sex = pd.Series(la.fit_transform(tall['Sex']))
mapped_sex.value_counts().plot.barh()
plt.show()
```

В результате получим график, показанный на рис. 1.29.

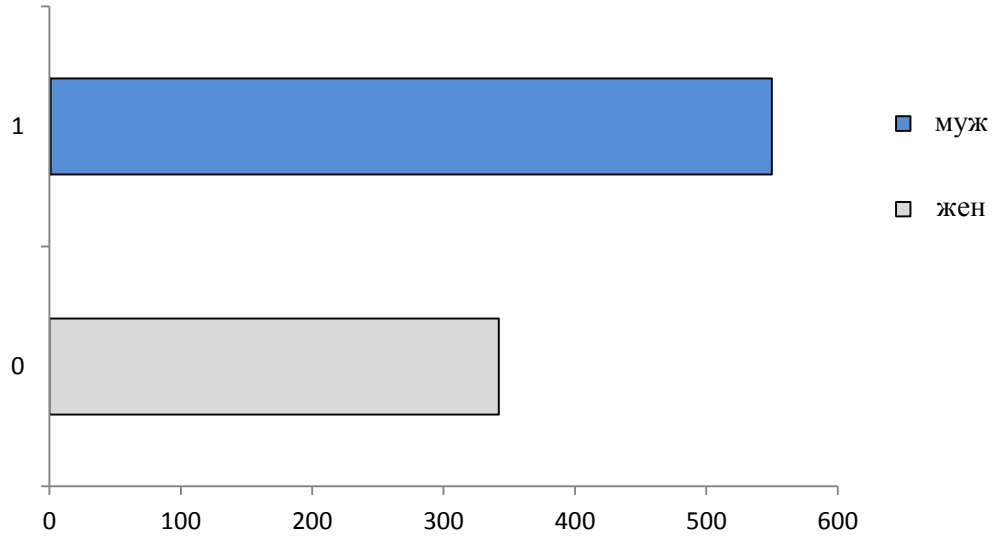

Рис. 1.29. Численность пассажиров мужского и женского пола

Распечатаем уникальные индексы поля Sex и их текстовые значения. Для потребуется словарь, поэтому создадим словарь с именем dicts.

```
dicts = {}
dicts=dict(enumerate(la.classes_))
print(dicts)
```

В результате мы получили mapped_sex (объект типа Series) содержащий 0 и 1. На рис. 1.30 представлены индексы поля Sex и их индексы.

```
{0: 'female', 1: 'male'}
```

Рис. 1.30. Значения поля Sex и их индексы

Полученный массив числовых индексов поля Sex загружаем в таблицу tall заменяя текстовые значения поля.

```
tall['Sex'] = mapped_sex
tall.head(5)
```

Преобразуйте значения поля Embarked в числовые значения.

```
tall['Embarked'] = la.fit_transform(tall['Embarked'])
tall.head(5)
```

В итоге наши исходные данные будут выглядеть так (рис. 1.31):

Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	1	28	1	0	7.2500	2
1	1	0	28	1	0	71.2833	0
1	3	0	28	0	0	7.9250	2
1	1	0	28	1	0	53.1000	2
0	3	1	28	0	0	8.0500	2

Рис. 1.31. Окончательный вид таблицы

Очень часто для анализа данных строится корреляционная матрица, которая очень легко строится с помощью функции *corr()* класса DataFrame.

```
corr =tall.corr()
corr
```

Результат представлен на рис. 1.32. Обратите внимание что текстовые поля игнорируются при вычислении корреляционной матрицы.

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652	0.012658
Survived	-0.005007	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307
Pclass	-0.035144	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500
Age	0.036847	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067
SibSp	-0.057527	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651
Parch	-0.001652	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225
Fare	0.012658	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000

Рис.1.32. Корреляционная матрица

Самостоятельное задание 1

Рассматривается задача анализа экологической ситуации в различных странах. Для описания экологического состояния стран используются следующие показатели:

HDI2012 – рейтинг стран;

Name – название страны;

FFU2009 – использование ископаемого топлива 2009 год;

REU2009 – использование возобновляемой энергии 2008 год;

CDE2008 – выбросы углекислого газа 2008 года;

CDEPC2008 – выбросы углекислого газа на душу населения 2008 год;

CDEAverAnnuPerCapGrow70/08 – рост выбросов двуокиси углерода средний годовой на душу населения 1970/2008 ;

GGE per capita2005 – выбросы парниковых газов на душу населения 2005 год;

NRD2010 – истощение природных ресурсов 2010 год;

FA2010 – лесная зона 2010 год;

ChangeFA1990/2010 – изменение лесной территории 1990 / 2010;

FWW2003-2012 – извлечение пресной воды 2003–2012 годы;

ES2011 – виды, находящиеся под угрозой исчезновения в 2011 году;

AL2009 – сельскохозяйственные земли 2009 год;

NDdueND2005/2011 – число смертей из-за стихийных бедствий 2005/2011 годы;

PLonDL2010 – население, живущее на деградировавших землях 2010 год.

Все данные хранятся в файле Environment.csv. Файл подготовлен в российском формате (разделители точка с запятой).

1) Загрузите данные в структуру DataFrame.

2) Для каждого поля определите наличие пропущенных значений.

3) Заполните пропущенные значения средними значениями.

Самостоятельное задание 2

Сегодня Сан-Франциско известен скорее как город высоких технологий, а не криминальный город. Но, несмотря на неравенство в отношении богатства, нехватку жилья и распространение дорогостоящих цифровых игрушек, в городе по-прежнему нет недостатка в преступности. Предлагаемый для анализа набор данных содержит почти 12-летний отчет о преступлениях во всех районах Сан-Франциско. Учитывая время и место, вы должны предсказать категорию преступления, которое произошло.

Этот набор данных содержит описание правонарушений, полученных из системы отчетности о правонарушениях США. Он содержит следующие показатели:

Dates – отметка времени преступления;

Category – категория преступления (только в train.csv). Это целевая переменная, которую вы собираетесь прогнозировать;

Descript – подробное описание инцидента с преступностью (только в train.csv);

DayOfWeek – день недели;

PdDistrict – название округа полицейского управления;

Resolution – как был разрешен инцидент с преступностью (только в train.csv);

Address – приблизительный уличный адрес преступления;

X – долгота;

Y – широта.

Все данные хранятся в файле Crimin01.csv. Файл подготовлен в Европейском формате (разделители запятые).

1. Загрузите данные в структуру DataFrame.
2. Преобразуйте все текстовые поля в числовые данные.
3. Попробуйте выполнить обработку в цикле.

1.8. Практическое занятие № 2. Агрегирование данных

Цель занятия:

Изучить основные элементы языка Python.

Изучить методы создания сводных таблиц и построения графиков.

Учебное задание

Крушение парохода «Титаник» представляет крупнейшую морскую катастрофу, произошедшей в ночь с 14 на 15 апреля 1912 года в северной части Атлантического океана. Трагедия случилась под конец пятого дня следования «Титаника» по трансатлантическому маршруту Саутгемптон – Нью-Йорк. В 23часа 40 минут 14 апреля во время первого рейса самый большой на тот момент океанский лайнер с 2208 людьми на борту по касательной столкнулся с айсбергом и получил серьёзные повреждения обшивки корпуса. Спустя 2 часа 40 минут полностью ушёл под воду. Катастрофа унесла жизни, по разным данным, от 1495 до 1635 человек.

Выполните анализ вероятности спасения пассажиров Титаника.

Технология выполнения учебного задания

Загрузим данные из файла, который мы сохранили на предыдущем занятии.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
tall = pd.read_csv("../Titanic/Data/Prac02.csv")
tall.head(8)
```

Как показывает предварительный анализ номер билета и имя пассажира нам никак не помогут, т.к. имя – это просто справочная информация, а по полю номер билита очень много пропущенной информации. Поэтому их нужно просто удалить. Для этого используем функцию *drop()* класса DataFrame.

```
tall = tall.drop(['Name','Ticket','Cabin'], axis=1)
tall.head(8)
```

Аргумент axis используется для указания оси удаления. В нашем случае это столбцы, поэтому axis=1. Теперь, после удаления всех ненужных полей, наш набор выглядит так как показано на рис.1.31.

Можно предположить, что чем выше социальный статус пассажира, тем выше вероятность его спасения. Давайте проверим это предположение. Для этого вычислим, сколько пассажиров каждого класса спаслись и сколько утонули. Такой анализ можно легко выполнить путем построения сводной таблицы с помощью функции *pivot_table* библиотеки pandas. Синтаксис функции:

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

Функция *pivot_table* возвращает сводную таблицу в стиле DataFrame.

Аргумент *data* – это исходная таблица типа DataFrame.

Аргумент *values* задает имя столбцы, который агрегируется. Для нашей таблицы это будет столбец PassengerId.

Аргумент *index* задает имя столбца, по которому осуществляется группировка. Используется для идентификации строк агрегированной таблицы. В нашем случае будем использовать Pclass.

Аргумент *columns* задает имя столбца, по которому осуществляется группировка. Используется для идентификации столбцов агрегированной таблицы. В нашем случае будем использовать Survived.

Аргумент *aggfunc* – определяет имя функции агрегирования. В качестве имени может использоваться mean (среднее значение), sum (сумма значений), count (количество значение).

Аргумент *fill_value* – позволяет заполнять заданными значениями ячейки таблицы типа NAN (неопределенное значение).

Для агрегирования таблицы tall следующий оператор:

```
tall_aggr1=pd.pivot_table(tall,values='PassengerId', index='Pclass', columns='Survived',  aggfunc='count')
tall_aggr1
```

В итоге получаем следующую таблицу (рис. 1.33).

Survived	0	1
Pclass		
1	80	136
2	97	87
3	372	119

Рис. 1.33. Сводная таблица

Таким образом, агрегированная таблица показывает, что 136 из 226 пассажиров 1-го класса спаслись.

Попробуйте записать оператор немного по-другому, используя функцию pivot_table как метод класса DataFrame.

```
tall_aggr1=tall.pivot_table(values='PassengerId', index='Pclass', columns='Survived',  aggfunc='count')
tall_aggr1
```

Можно сократить запись.

```
tall_aggr1=tall.pivot_table('PassengerId', 'Pclass', 'Survived',  'count')
tall_aggr1
```

Отобразим данные сводной таблицы в виде диаграммы. Для этого будем использовать метод plot структуры DataFrame. Описание структуры DataFrame:

DataFrame.plot(x=None, y=None, kind='line', ax=None, subplots=False, sharex=None, sharey=False, layout=None, figsize=None, use_index=True, title=None, grid=None, legend=True, style=None, logx=False, logy=False, loglog=False, xticks=None, yticks=None, xlim=None, ylim=None, rot=None, fontsize=None, colormap=None, table=False, yerr=None, xerr=None, secondary_y=False, sort_columns=False, **kwds)¶

Аргумент *kind* задает тип графического отображения и может принимать значения:

line – график линейный (по умолчанию);

bar – вертикальная столбчатая диаграмма;

barh – горизонтальная столбчатая диаграмма;

hist – гистограмма.

Аргумент *figsize* задает размеры рисунка.

Аргумент *fontsize* задает размеры шрифта.

Аргумент *colormap* принимает значения либо палитры Matplotlib, либо строки, которая является именем colormap, зарегистрированным в Matplotlib.

Примеры имен colormap: gist_rainbow, gist_gray, winter, spring, summer, autumn, spectral, cubehelix, prism.

Аргумент *stacked* позволяет задавать сложные диаграммы. Для этого он должен иметь значение True.

Аргумент *title* позволяет задать название графика или диаграммы, которое расположено в верхней части рисунка.

Постройте диаграмму, используя оператор:

```
tall_aggr1.plot(kind='bar', stacked=True)
plt.show()
```

В результате получаем диаграмму (рис. 1.34).

Вышеописанное предположение про то, что чем выше у пассажиров их социальное положение, тем выше их вероятность спасения, оказалось правильным.

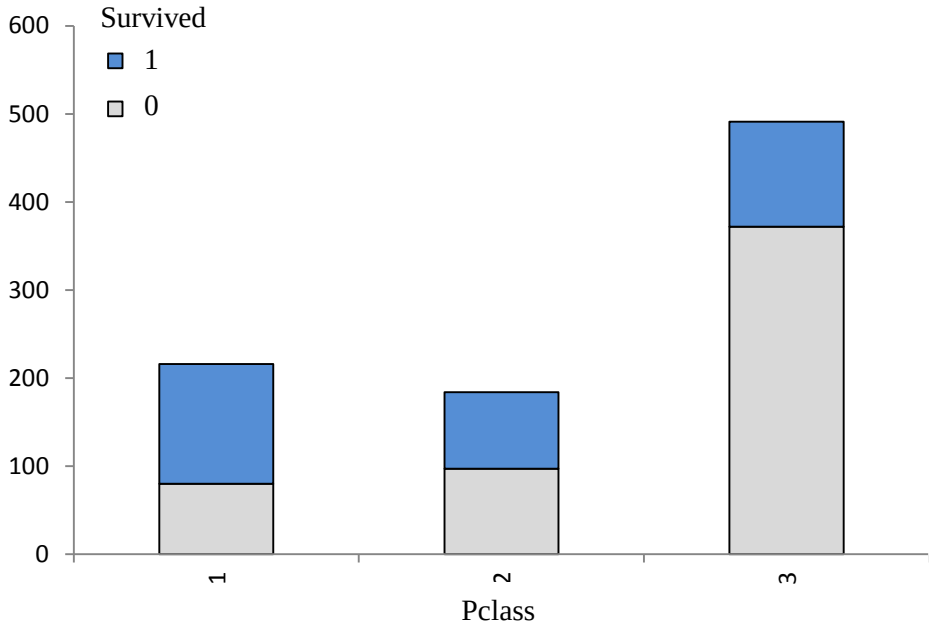

Рис. 1.34. Диаграмма числа спасенных и утонувших пассажиров в зависимости от социального класса пассажира

Для изменения размеров рисунка используйте аргументы figsize, fontsize. Кроме того, угол поворота текста зададим равным 0, код будет выглядеть следующим образом:

```
tall_aggr1.plot(kind='bar', stacked=True, figsize=(16,10),  fontsize =18,
    rot=0)
plt.show()
```

Результат представлен на рис. 1.35.

Задайте подписи по осям. Для этого будем использовать функции xlabel() и ylabel(). В качестве аргументов указываем строку с названием оси и размер шрифта. Далее откорректируем легенду с помощью функции legend(). В качестве аргументов задаем labels и fontsize.

```
tall_aggr1.plot(kind='bar', stacked=True, figsize=(16,10),  fontsize =18,
    rot=0)
plt.legend(labels=['Спаслось','Утонуло'], fontsize=18)
plt.xlabel("Социальный статус пассажира", fontsize=18)
plt.ylabel("Число пассажиров", fontsize=18)
plt.show()
```

В результате выполнения команды получаем график (рис. 1.36).

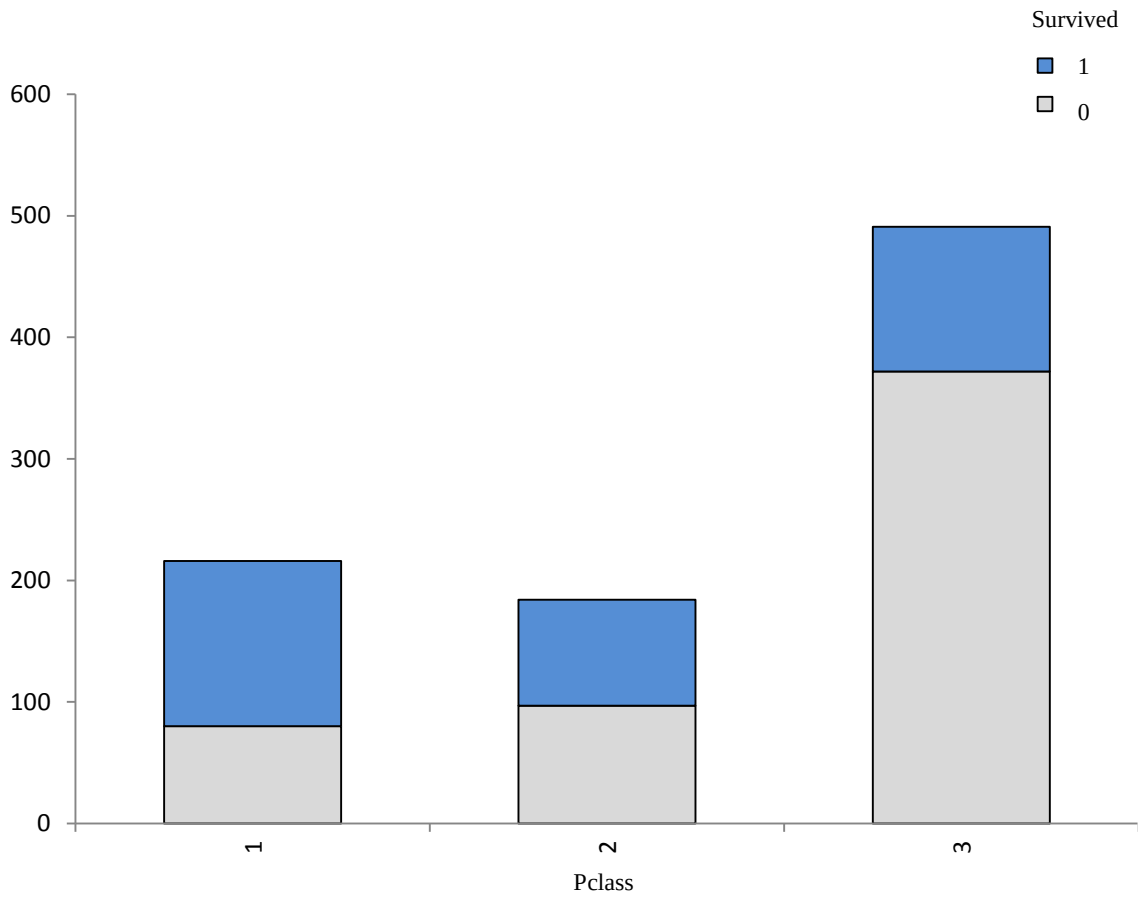

Рис. 1.35. Диаграмма числа спасенных и утонувших пассажиров в зависимости от социального класса пассажира

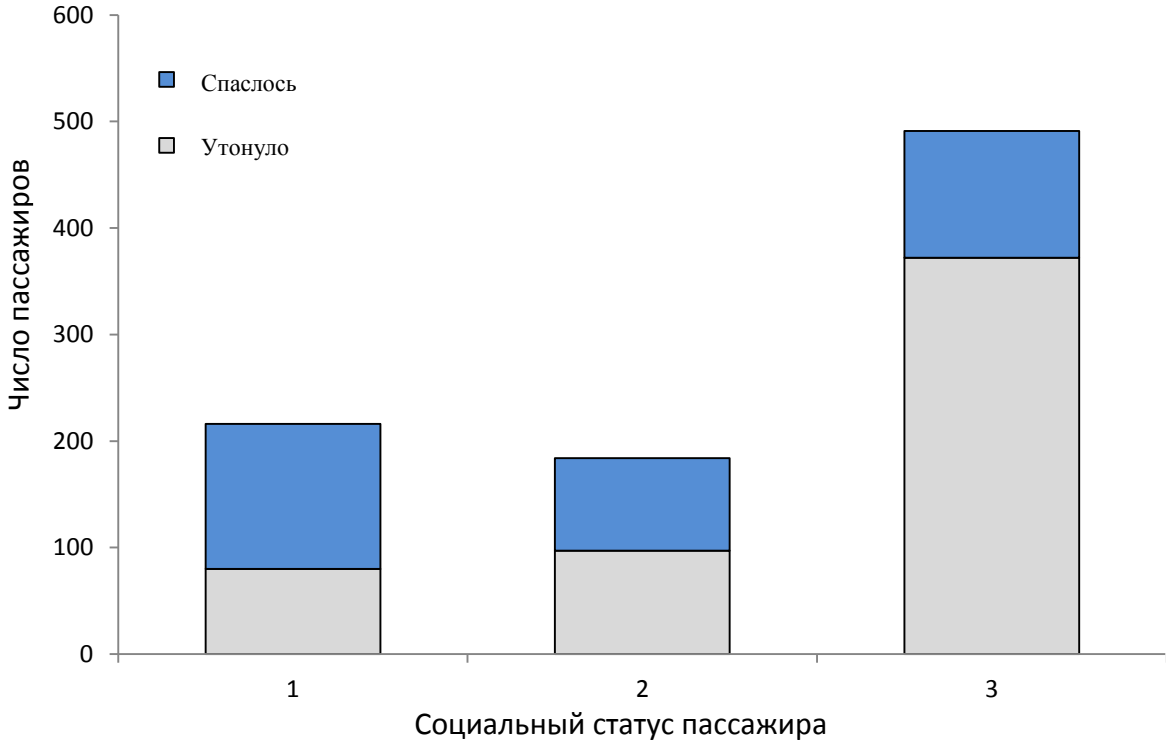

Рис. 1.36. Диаграмма числа спасенных и утонувших пассажиров в зависимости от социального класса пассажира

Изменим цветовую гамму нашей диаграммы. для этого используем аргумент colormap.

```
tall_aggr1.plot(kind='bar', stacked=True, figsize=(16,10),  fontsize =18,
    rot=0, colormap='winter')
plt.legend(labels=['Спаслось','Утонуло'], fontsize=18)
plt.xlabel("Социальный статус пассажира", fontsize=18)
plt.ylabel("Число пассажиров", fontsize=18)
plt.show()
```

Поменяйте цветовую гамму используя другие карты цветов gist_rainbow, gist_gray, spring, summer, autumn, spectral, cubehelix, prism.

Результат представлен на рис. 1.37.

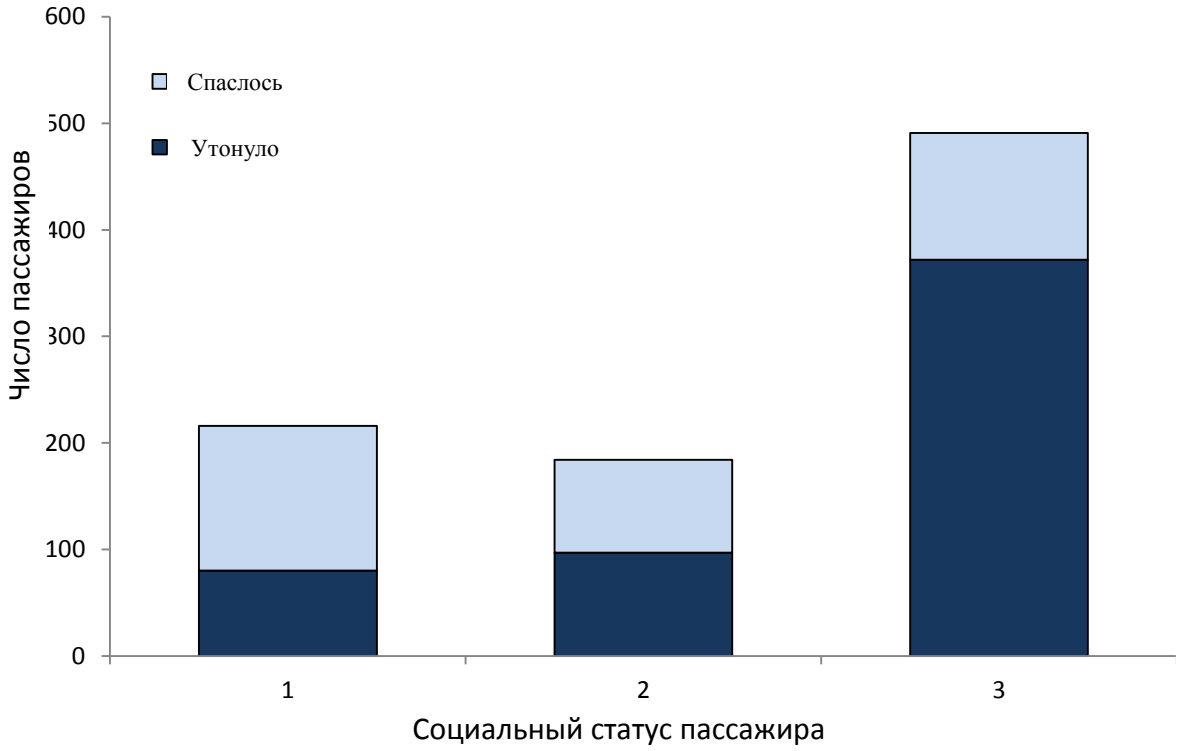

Рис. 1.37. Диаграмма числа спасенных и утонувших пассажиров в зависимости от социального класса пассажира

Для того чтобы сохранить рисунок в виде файла используйте функцию savefig()

```
plt.savefig("PClass.jpg", dpi=300)
```

Сохраните сводную таблицу в Excel. Для этого создайте файл с именем AggrTitanic.xlsx и сохраните таблицу в виде листа PClass. Для создания файла будем использовать метод ExcelWriter библиотеки pandas

```
writer = pd.ExcelWriter('AggrTitanic.xlsx', engine='xlsxwriter')
tall_aggr1.to_excel(writer, sheet_name='PClass', index=True, index_label
    = True)
```

Здесь writer представляет указатель на открытый файл. В дальнейшем его мы будем использовать при добавлении в этот файл других страниц.

Теперь исследуйте, как количество родственников влияет на факт спасения. Для исследования влияние родственников 2-го порядка (муж, жена, братья, сестры), которые хранятся в поле SibSp.

```
tall_aggr2=tall.pivot_table('PassengerId', 'SibSp', 'Survived', 'count')
tall_aggr2
```

В результате получим следующую таблицу (рис. 1.38).

Survived	0	1
SibSp		
0	398.0	210.0
1	97.0	112.0
2	15.0	13.0
3	12.0	4.0
4	15.0	3.0
5	5.0	NaN
8	7.0	NaN

Рис. 1.38. Сводная таблица для исследования влияния количества родственников 2-го порядка у пассажира на вероятность его спасения

Мы видим, что в последних строках сводной таблицы появились неопределенные значения. Заменим их нулями, для этого дополним наш список аргументов.

```
tall_aggr2=tall.pivot_table('PassengerId', 'SibSp', 'Survived', 'count',
    fill_value=0)
tall_aggr23
```

Также изучим влияние родственников 1-го порядка, которые хранятся в поле 'Parch'.

```
tall_aggr3=tall.pivot_table('PassengerId', 'Parch', 'Survived', 'count',
    fill_value=0)
tall_aggr3
```

В результате получим следующую таблицу (рис. 1.40).

Построим графические зависимости количества родственников 1-го и 2-го порядка у пассажира на вероятность его спасения. Для этого используем библиотеку matplotlib.pyplot (сокращенно plt). В качестве аргумента используйте дополнительно аргументы ax и title: ax – объект оси matplotlib.

Survived	0	1
Parch		
0	445.0	233.0
1	53.0	65.0
2	40.0	40.0
3	2.0	3.0
4	4.0	NaN
5	4.0	1.0
6	1.0	NaN

Рис. 1.40. Сводная таблица для исследования влияния количества родственников 1-го порядка у пассажира на вероятность его спасения

```
fig, axes = plt.subplots(ncols=2)
tall_aggr2.plot(ax=axes[0], title='SibSp')
tall_aggr3.plot(ax=axes[1], title='Parch')
plt.show()
```

Результат представлен на рис. 1.41.

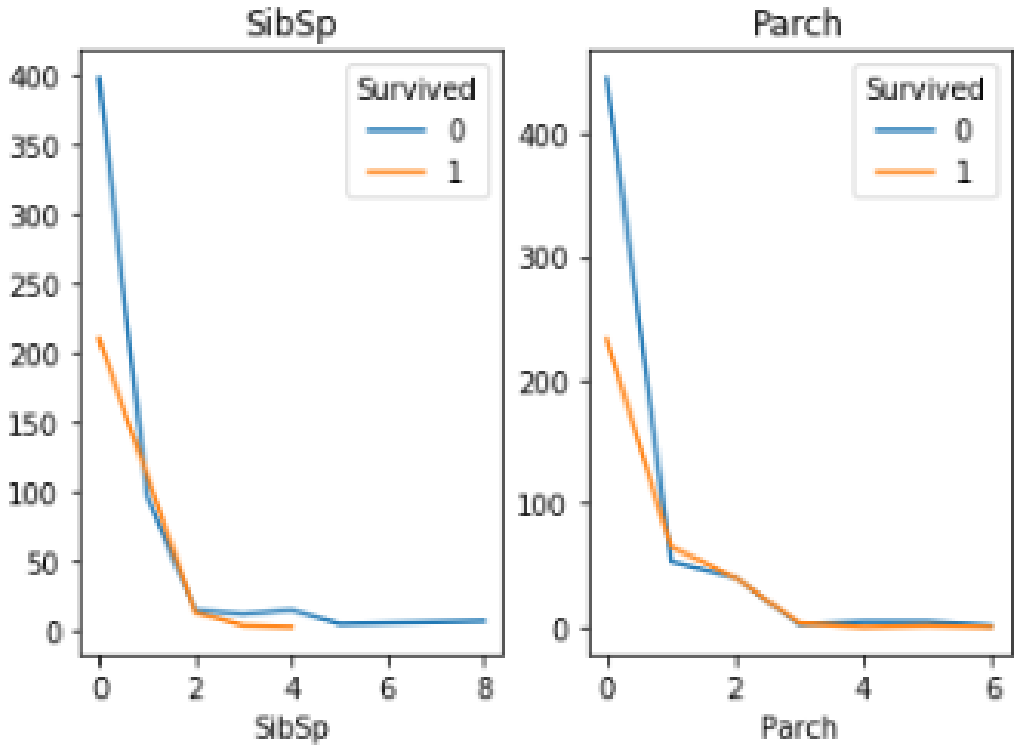

Рис. 1.41. Влияние количества родственников 1-го и 2-го порядка у пассажира на вероятность его спасения

Изменим тип графических зависимостей на вертикальную диаграмму.

```
fig, axes = plt.subplots(ncols=2)
tall_aggr2.plot(kind='bar',ax=axes[0], title='SibSp')
tall_aggr3.plot(kind='bar',ax=axes[1], title='Parch')
plt.show()
```

Результат представлен на рис. 1.42.

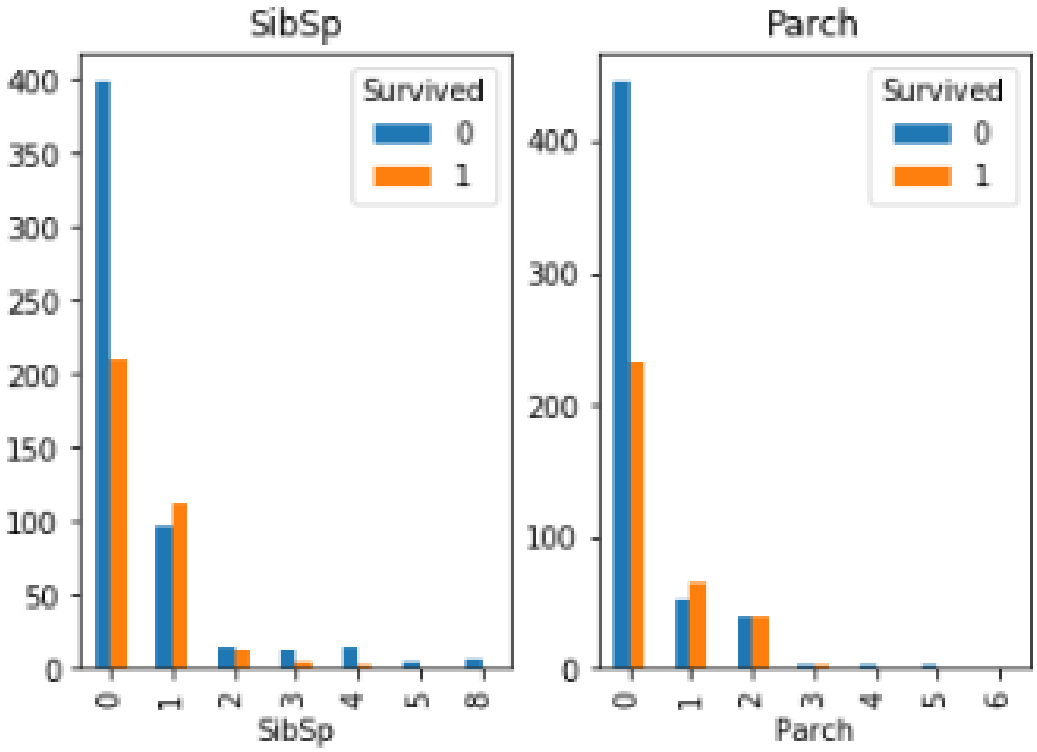

Рис. 1.42. Влияние количества родственников 1-го и 2-го порядка у пассажира на вероятность его спасения

Как видно из графиков наше предположение снова подтвердилось, и из людей, имеющих больше одного родственника спаслись немногие.

Сохраните агрегированные таблицы в виде листов SibSb и Parch файла AggrTitanic.xlsx:

```
tall_aggr2.to_excel(writer, sheet_name='SibSb', index=True, index_label =
    True)
tall_aggr3.to_excel(writer, sheet_name='Parch', index=True, index_label =
    True)
```

Так как больше в этом файле мы ничего сохранять не собираемся, то закроем этот файл.

```
writer.save()
```

Сохраним полученную таблицу в файле Prak02.scv. Для этого используйте функцию to_csv():

```
tall.to_csv('../Titanic/Data/Prak02.csv', index=False, sep=';', float_for-
    mat='%.5f', decimal=',' )
```

Самостоятельное задание 3

Отредактируйте последние два рисунка так, чтобы размер рисунков был (16, 10), размер шрифта 18. Подпишите оси «количества родственников 1-го порядка» и «количества родственников 2-го порядка». Откорректируйте легенду.

Самостоятельное задание 4

По 50 водителям были зарегистрированы:

X1 – возраст;

X2 – состояние зрения (имеются проблемы со зрением — значение переменной равно единице, нет проблем – нулю);

X3 – уровень подготовки водителя (если прошел курсы для водителей, X3 = 1, если нет, X3 = 0);

Y – наличие дорожно-транспортных происшествий (ДТП) в последний год (0 – нет; 1 – да).

Все данные хранятся в файле DTP.csv. Файл подготовлен в российском формате (разделители точка с запятой).

1) Загрузите данные в структуру DataFrame.

2) Можно предположить, что чем хуже зрение водителя, тем выше вероятность ДТП. Давайте проверим это предположение. Для этого вычислите сколько водителей каждого класса имели ДТП, а сколько не имели. Анализ выполнить путем построения сводной таблицы и диаграммы.

3) Можно предположить, что чем ниже уровень подготовки водителя, тем выше вероятность ДТП. Проверьте это предположение. Для этого вычислите сколько водителей каждого класса (прошли курсы или нет) имели ДТП, а сколько не имели. Анализ выполнить путем построения сводной таблицы и диаграммы.

Самостоятельное задание 5

Рассматривается задача анализа кредитоспособности заемщика. Для описания финансового и социального состояния заемщика используются следующие показатели:

ZAEM – брался ли кредит ранее (да, нет);

DOXOD – среднемесячный доход семьи заемщика, тыс. руб .;

TIME – период погашения кредита;

VOL – размер кредита;

FAMIL – состав семьи заемщика, чел.;

OLD – возраст заемщика, лет;

RETURN – вероятность погашения кредита, 3 – высокая, 2 – средняя, 1 – низкая.

Все данные хранятся в файле credit.csv. Файл подготовлен в российском формате (разделители точка с запятой).

1) Загрузите данные в структуру DataFrame.

2) Можно предположить, что если кредит брался ранее, то вероятность его погашения выше. Проверьте это предположение. Для этого вычислите сколько заемщиков, бравших кредит, имеют вероятность погашения кредита 3, 2 и 1. Анализ выполнить путем построения сводной таблицы и диаграммы.

Глава 2. МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ

2.1. Введение

В библиотеке scikit-learn реализована масса алгоритмов машинного обучения. Некоторые алгоритмы машинного обучения, реализованные в scikit-learn, представлены в табл. 2.1.

Таблица 2.1

Метод	Класс
kNN – kk ближайших соседей	sklearn.neighbors.KNeighborsClassifier
LDA – линейный дискриминантный анализ	sklearn.lda.LDA
QDA – квадратичный дискриминантный анализ	sklearn.qda.QDA
Logistic – логистическая регрессия	sklearn.linear_model.LogisticRegression
SVC – машина опорных векторов	sklearn.svm.SVC
Tree – деревья решений	sklearn.tree.DecisionTreeClassifier
RF – случайный лес	sklearn.ensemble.RandomForestClassifier
AdaBoost – адаптивный бустинг	sklearn.ensemble.AdaBoostClassifier
GBT – градиентный бустинг деревьев решений	sklearn.ensemble.GradientBoostingClassifier

Основные методы классов, реализующих алгоритмы машинного обучения. Все алгоритмы выполнены в виде классов, обладающих, по крайней мере, следующими методами (табл. 2.2 и табл. 2.3).

Таблица 2.2

Метод класса	Описание
fit(X, y)	обучение (тренировка) модели на обучающей выборке X, y
predict(X)	предсказание класса и регрессионного значения на данных X
predict_proba(X[, check_input])	предсказание вероятности класса на данных X
score(X, y[, sample_weight])	вычисляет среднюю точность на заданных тестовых данных и ответах
set_params(**params)	установка параметров алгоритма
get_params()	чтение параметров алгоритма

Таблица 2.3

Методы класса	kNN	LDA	QDA	Logistic	SVC	Tree	RF	AdaBoost	GBT
fit(X, y)	+	+	+	+	+	+	+	+	+
predict(X)	+	+	+	+	+	+	+	+	+
predict_proba(X)	+	+	+	+	–	+	+	+	+
predict_log_proba(X)	–	+	+	+	–	+	+	+	+
score(X, y)	+	+	+	+	+	+	+	+	+
decision_function(X)	–	+	+	+	+	–	–	+	+
transform(X)	–	+	–	–	–	+	+	–	+
staged_decision_function(X)	–	–	–	–	–	–	–	+	+
staged_predict(X)	–	–	–	–	–	–	–	+	+
staged_predict_proba(X)	–	–	–	–	–	–	–	+	+
staged_score(X, y)	–	–	–	–	–	–	–	+	+
set_params(**params)	+	+	+	+	+	+	+	+	+
get_params()	+	+	+	+	+	+	+	+	+

Заметим, что параметры алгоритмов обучения можно задавать как в конструкторе класса, так и с помощью метода set_params(**params**).**

2.2. Технология обучения модели

Главная задача обучаемых алгоритмов – их способность *обобщаться*, то есть хорошо работать на новых данных. Поскольку на новых данных мы сразу не можем проверить качество построенной модели (нам ведь надо для них сделать прогноз, то есть истинных значений целевого признака мы для них не знаем), то надо пожертвовать небольшой порцией данных, чтобы на ней проверить качество модели.

Чаще всего это делается одним из 2 способов:

- отложенная выборка (*held-out/hold-out set*). При таком подходе мы оставляем какую-то долю обучающей выборки (как правило от 20% до 40%), обучаем модель на остальных данных (60-80% исходной выборки) и считаем некоторую метрику качества модели (например, самое простое – долю правильных ответов в задаче классификации) на отложенной выборке;

• кросс-валидация (*cross-validation*, на русский еще переводят как скользящий или перекрестный контроль). Тут модель обучается k раз на разных k−1 подвыборках исходной выборки (белый цвет), а проверяется на одной подвыборке (каждый раз на разной, серый цвет). Получаются k оценок качества модели, которые обычно усредняются, выдавая среднюю оценку качества классификации/регрессии на кросс-валидации (рис. 2.1).

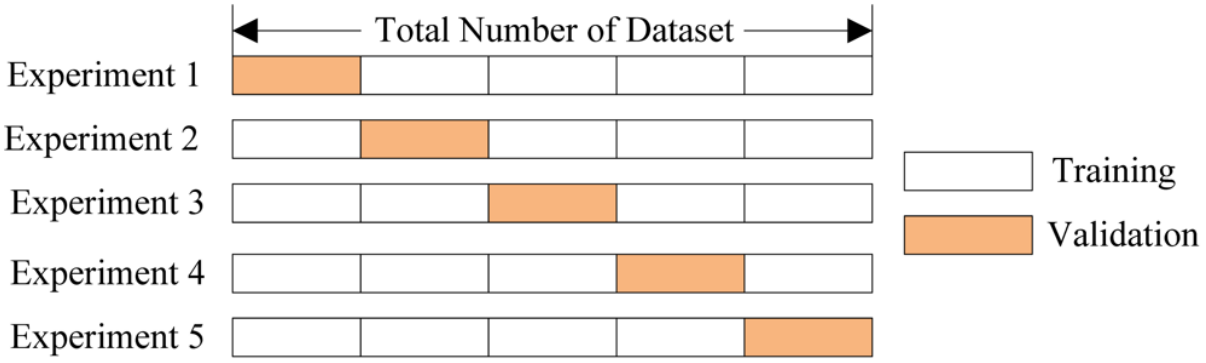

Рис. 2.1. Кросс-валидация

Кросс-валидация дает лучшую по сравнению с отложенной выборкой оценку качества модели на новых данных. Но кросс-валидация вычислительно дорогостоящая, если данных много.

Кросс-валидация – очень важная техника в машинном обучении (применяемая также в статистике и эконометрике), с ее помощью выбираются гиперпараметры моделей, сравниваются модели между собой, оценивается полезность новых признаков в задаче и т.д.

2.3. Дерево решений

Зачастую дерево решений служит обобщением опыта экспертов, средством передачи знаний будущим сотрудникам или моделью бизнес-процесса компании. Например, до внедрения масштабируемых алгоритмов машинного обучения в банковской сфере задача кредитного скоринга решалась экспертами. Решение о выдаче кредита заемщику принималось на основе некоторых интуитивно (или по опыту) выведенных правил, которые можно представить в виде дерева решений.

Дерево решений как алгоритм машинного обучения – по сути объединение логических правил вида «*Значение признака a меньше x И Значение признака b меньше y… => Класс 1*» в структуру данных «Дерево». Огромное преимущество деревьев решений в том, что они легко интерпретируемы, понятны человеку. Например, можно объяснить заемщику, почему ему было отказано в кредите. Скажем, потому, что у него нет дома и доход меньше 5000. Как мы увидим дальше, многие другие, хоть и более точные, модели не обладают этим свойством и могут рассматриваться скорее как «черный

ящик», в который загрузили данные и получили ответ. В связи с этой «понятностью» деревьев решений и их сходством с моделью принятия решений человеком, деревья решений получили огромную популярность.

Энтропия Шеннона определяется для системы с N возможными состояниями следующим образом:

$$S = -\sum_{i=1}^{N} p_i \log_2 p_i,$$

где p_i – вероятности нахождения системы в i-ом состоянии.

Это очень важное понятие, используемое в физике, теории информации и других областях. Опуская предпосылки введения (комбинаторные и теоретико-информационные) этого понятия, отметим, что, интуитивно, энтропия соответствует степени хаоса в системе. Чем выше энтропия, тем менее упорядочена система и наоборот. Это поможет нам формализовать «эффективное разделение выборки», про которое мы говорили в контексте игры «20 вопросов».

В основе популярных алгоритмов построения дерева решений лежит принцип жадной максимизации прироста информации – на каждом шаге выбирается тот признак, при разделении по которому прирост информации оказывается наибольшим. Дальше процедура повторяется рекурсивно, пока энтропия не окажется равной нулю или какой-то малой величине (если дерево не подгоняется идеально под обучающую выборку во избежание переобучения).

Мы разобрались в том, как понятие энтропии позволяет формализовать представление о качестве разбиения в дереве. Кроме энтропии может использоваться также критерий неопределенности Джини (Gini impurity):

$$G = 1 - \sum_{\mathrm{k}} (p_k)^2.$$

Максимизацию этого критерия можно интерпретировать как максимизацию числа пар объектов одного класса, оказавшихся в одном поддереве.

В принципе дерево решений можно построить до такой глубины, чтобы в каждом листе был ровно один объект. Но на практике это не делается (если строится только одно дерево) из-за того, что такое дерево будет *переобученным* – оно слишком настроится на обучающую выборку и будет плохо работать на прогноз на новых данных. Где-то внизу дерева, на большой глубине будут появляться разбиения по менее важным признакам (например, приехал ли клиент из Саратова или Костромы). Если утрировать, может оказаться так, что из всех 4 клиентов, пришедших в банк за кредитом в зеленых штанах, никто не вернул кредит. Но мы не хотим, чтобы наша модель классификации порождала такие специфичные правила.

Есть два исключения, ситуации, когда деревья строятся до максимальной глубины:

1) ансамбль деревьев (композиция многих деревьев) усредняет ответы деревьев, построенных до максимальной глубины;

2) стрижка дерева (*pruning*). При таком подходе дерево сначала строится до максимальной глубины, потом постепенно снизу вверх некоторые вершины дерева убираются за счет сравнения по качеству дерева с данным разбиением и без него (сравнение проводится с помощью *кросс-валидации*, о которой чуть ниже).

Основные способы борьбы с переобучением в случае деревьев решений:

1) искусственное ограничение глубины или минимального числа объектов в листе: построение дерева просто в какой-то момент прекращается;

2) стрижка дерева.

2.4. Случайный лес (Random forest)

Как мы только что отметили, основным недостатком деревьев решений является их склонность к переобучению. Случайный лес является одним из способов решения этой проблемы. По сути случайный лес – это набор деревьев решений, где каждое дерево немного отличается от остальных. Идея случайного леса заключается в том, что каждое дерево может довольно хорошо прогнозировать, но скорее всего переобучается на части данных. Если мы построим много деревьев, которые хорошо работают и переобучаются с разной степенью, мы можем уменьшить переобучение путем усреднения их результатов. Уменьшение переобучения при сохранении прогнозной силы деревьев можно проиллюстрировать с помощью строгой математики.

Для реализации вышеизложенной стратегии нам нужно построить большое количество деревьев решений. Каждое дерево должно на приемлемом уровне прогнозировать целевую переменную и должно отличаться от других деревьев. Случайные леса получили свое название из-за того, что в процесс построения деревьев была внесена случайность, призванная обеспечить уникальность каждого дерева. Существует две техники, позволяющие получить рандомизированные деревья в рамках случайного леса: сначала выбираем точки данных (наблюдения), которые будут использоваться для построения дерева, а затем отбираем признаки в каждом разбиении. Давайте разберем этот процесс более подробно.

Для построения модели случайных лесов необходимо определиться с количеством деревьев (параметр n_estimators для RandomForestRegressor или RandomForestClassifier). Допустим, мы хотим построить 10 деревьев. Эти деревья будут построены совершенно независимо друг от друга, и алгоритм будет случайным образом отбирать признаки для построения каждого дерева, чтобы получить непохожие друг на друга деревья.

Для построения дерева мы сначала сформируем *бутстреп-выборку* (*bootstrap sample*) *исходных* данных. То есть из n_samples примеров мы слу-

чайным образом выбираем пример с возвращением n_samples раз (поскольку отбор с возвращением, то один и тот же пример может быть выбран несколько раз). Мы получаем выборку, которая имеет такой же размер, что и исходный набор данных, однако некоторые примеры будут отсутствовать в нем (примерно одна треть), а некоторые попадут в него несколько раз.

Чтобы проиллюстрировать это, предположим, что мы хотим создать бутстреп-выборку списка ['a', 'b', 'c', 'd']. Возможная бутстреп-выборка может выглядеть как ['b', 'd', 'd', 'c']. Другой возможной бутстреп-выборкой может быть ['d', 'a', 'd', 'a'].

Далее на основе этой сформированной бутстреп-выборки строится дерево решений. Однако алгоритм, который мы описывали для дерева решений, теперь слегка изменен. Вместо поиска наилучшего теста для каждого узла, алгоритм для разбиения узла случайным образом отбирает подмножество признаков и затем находит наилучший тест, используя один из этих признаков. Количество отбираемых признаков контролируется параметром max_features. Отбор подмножества признаков повторяется отдельно для каждого узла, поэтому в каждом узле дерева может быть принято решение с использованием «своего» подмножества признаков.

Использование бутстрепа приводит к тому, что деревья решений в случайном лесе строятся на немного отличающихся между собой бутстреп-выборках. Из-за случайного отбора признаков в каждом узле все расщепления в деревьях будут основано на отличающихся подмножествах признаков. Вместе эти два механизма приводят к тому, что все деревья в случайном лесе отличаются друг от друга.

Критическим параметром в этом процессе является max_features. Если мы установим max_features равным n_features, это будет означать, что в каждом разбиении могут участвовать все признаки набора данных, в отбор признаков не будет привнесена случайность (впрочем, случайность в силу использования бутстрепа остается). Если мы установим max_features равным 1, это означает, что при разбиении не будет никакого отбора признаков для тестирования вообще, будет осуществляться поиск с учетом различных пороговых значений для случайно выбранного признака. Таким образом, высокое значение max_features означает, что деревья в случайном лесе будут весьма схожи между собой и они смогут легко аппроксимировать данные, используя наиболее дискриминирующие признаки. Низкое значение max_features означает, что деревья в случайном лесе будут сильно отличаться друг от друга и, возможно, каждое дерево будет иметь очень большую глубину, чтобы хорошо соответствовать данным.

Чтобы дать прогноз для случайного леса, алгоритм сначала дает прогноз для каждого дерева в лесе. Для регрессии мы можем усреднить эти результаты, чтобы получить наш окончательный прогноз. Для классификации используется стратегия «мягкого голосования». Это означает, что каждый алгоритм дает «мягкий» прогноз, вычисляя вероятности для каждого класса.

Эти вероятности усредняются по всем деревьям и прогнозируется класс с наибольшей вероятностью.

В отличие от отдельных деревьев случайный лес переобучается в меньшей степени и дает гораздо более чувствительную (гибкую) границу принятия решений. В реальных примерах используется гораздо большее количество деревьев (часто сотни или тысячи), что приводит к получению еще более чувствительной границы.

В настоящее время случайные леса регрессии и классификации являются одним из наиболее широко используемых методов машинного обучения. Они обладают высокой прогнозной силой, часто дают хорошее качество модели без утомительной настройки параметров и не требуют масштабирования данных.

По сути случайные леса обладают всеми преимуществами деревьев решений, хотя и не лишены некоторых их недостатков. Одна из причин, в силу которой деревья решений еще используются до сих пор, – это компактное представление процесса принятия решений. Детальная интерпретация десятков или сотен деревьев нвозможна в принципе, и, как правило, деревья в случайном лесе получаются более глубокими по сравнению с одиночными деревьями решений (из-за использования подмножеств признаков). Поэтому, если вам нужно в сжатом виде визуализировать процесс принятия решений для неспециалистов, одиночное дерево решений может быть оптимальным выбором. Несмотря на то, что построение случайных лесов на больших наборах данных может занимать определенное время, его можно легко распараллелить между несколькими ядрами процессора в компьютере.

Вы должны помнить, что случайный лес по своей природе является рандомизированным алгоритмом и установка различных стартовых значений генератора псевдослучайных чисел (или вообще отказ от использования random_state) может кардинально изменить построение модели. Чем больше деревьев в лесу, тем более устойчивым он будет к изменению стартового значения. Если вы хотите получить результаты, которые потом нужно будет воспроизвести, важно зафиксировать random_state.

Случайный лес плохо работает на данных очень высокой размерности, разреженных данных, например, на текстовых данных. Для подобного рода данных линейные модели подходят больше. Случайный лес, как правило, хорошо работает даже на очень больших наборах данных, и обучение могут легко распараллелить между многочисленными процессорными ядрами в рамках мощного компьютера. Однако случайный лес требует больше памяти и медленнее обучается и прогнозирует, чем линейные модели. Если время и память имеют важное значение, имеет смысл вместо случайного леса использовать линейную модель.

Важными параметрами настройки являются n_estimators, max_features и опции предварительной обрезки деревьев, например, max_depth. Что касается n_estimators, большее значение всегда дает лучший результат. Усреднение результатов по большему количеству деревьев позволит получить более устойчивый ансамбль за счет снижения переобучения. Однако обратная сторона увеличения числа деревьев заключается в том, что с ростом количества деревьев требуется больше памяти и больше времени для обучения. Общее правило заключается в том, чтобы построить «столько, сколько позволяет ваше время/память».

Как было описано ранее, max_features случайным образом определяет признаки, использующиеся при разбиении в каждом дереве, а меньшее значение max_features уменьшает переобучение. В общем, лучше взять за правило использовать значения, выставленные по умолчанию: max_features=sqrt(n_features) для классификации и max_features=n_features для регрессии. Увеличение значений max_features или max_leaf_nodes иногда может повысить качество модели. Кроме того, оно может резко снизить требования к пространству на диске и времени вычислений в ходе обучения и прогнозирования.

Одной из важных процедур предобработки данных в алгоритмах их анализа является отбор значимых признаков. Его цель заключается в том, чтобы отобрать наиболее существенные признаки для решения рассматриваемой задачи классификации.

Отбор признаков необходим для следующих целей:

1. для лучшего понимания задачи. Человеку легче разобраться с небольшим количеством признаков, чем с огромным их количеством.

2. для улучшения качества предсказания. Устранение шумовых признаков может уменьшить ошибку алгоритма на тестовой выборке, т.е. улучшить качество предсказания.

Отбор значимых признаков осуществляется как «вручную» – на основе анализа содержательной постановки задачи, так и «автоматически» – с помощью универсальных алгоритмов.

Отбор признаков «вручную» (как и «ручной» синтез новых признаков) – важный этап в анализе данных. К сожалению, нам не известны содержательные значения используемых в рассматриваемой задаче признаков, поэтому ограничимся только их автоматическим отбором. Для этого существует много различных алгоритмов. Рассмотрим только один из них – с помощью случайного леса.

Для отбора значимых признаков необходимо после вызова метода predict для случайного леса прочитать поле feature_importances_. Для каждого признака это поле содержит число, выражающее «важность» этого признака. Чем больше число, тем значимее признак. Сумма всех чисел равна 1.

2.5. Метрики в задачах классификации

Accuracy, precision и recall

Перед переходом к самим метрикам необходимо ввести важную концепцию для описания этих метрик в терминах ошибок классификации — *confusion matrix* (матрица ошибок). Допустим, что у нас есть два класса и алгоритм, предсказывающий принадлежность каждого объекта одному из классов, тогда матрица ошибок классификации будет выглядеть следующим образом:

факт / прогноз	y = 1	y = 0
$y^* = 1$	True Positive (TP)	False Positive (FP)
$y^* = 0$	False Negative (FN)	True Negative (TN)

Здесь y^* – это ответ алгоритма на объекте, а y – истинная метка класса на этом объекте. Таким образом, ошибки классификации бывают двух видов: False Negative (FN) и False Positive (FP).

Accuracy

Интуитивно понятной, очевидной и почти неиспользуемой метрикой является accuracy − доля правильных ответов алгоритма:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

Эта метрика бесполезна в задачах с неравновесными классами, и это легко показать на примере.

Допустим, мы хотим оценить работу спам-фильтра почты (рис. 2.2).

У нас есть 100 не-спам писем, 90 из которых наш классификатор определил верно (True Negative = 90, False Positive = 10), и 10 спам-писем, 5 из которых классификатор также определил верно (True Positive = 5, False Negative = 5). Тогда:

$$accuracy = \frac{5 + 90}{5 + 90 + 10 + 5} = 86{,}4.$$

Однако, если мы просто будем предсказывать все письма как не-спам, то получим более высокую accuracy:

$$accuracy = \frac{0 + 100}{0 + 100 + 0 + 10} = 90{,}9.$$

При этом, наша модель совершенно не обладает никакой предсказательной силой, так как изначально мы хотели определять письма со спамом.

Преодолеть это нам поможет переход с общей для всех классов метрики к отдельным показателям качества классов.

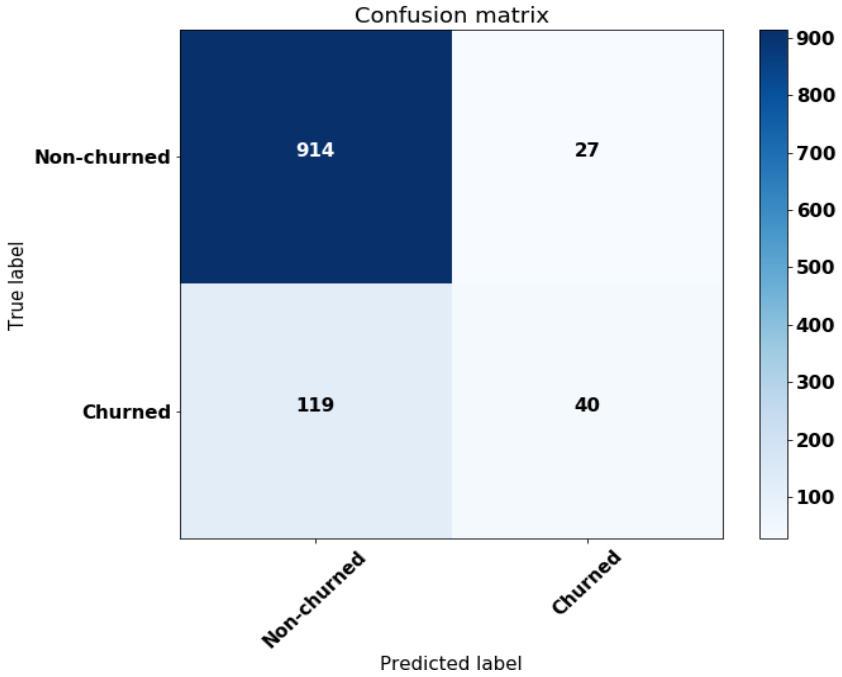

Рис. 2.2. Матрица ошибок

Precision, recall и F-мера

Для оценки качества работы алгоритма на каждом из классов по отдельности введем метрики precision (точность) и recall (полнота):

$$\text{precision} = \frac{TP}{TP + FP}, \quad \text{recall} = \frac{TP}{TP + FN}.$$

Precision можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм.

Именно введение precision не позволяет нам записывать все объекты в один класс, так как в этом случае мы получаем рост уровня False Positive. Recall демонстрирует способность алгоритма обнаруживать данный класс вообще, а precision – способность отличать этот класс от других классов.

Как мы отмечали ранее, ошибки классификации бывают двух видов: False Positive и False Negative. В статистике первый вид ошибок называют ошибкой I-го рода, а второй – ошибкой II-го рода. В нашей задаче по определению оттока абонентов, ошибкой первого рода будет принятие лояльного абонента за уходящего, так как наша *нулевая гипотеза* состоит в том, что никто из абонентов не уходит, а мы эту гипотезу отвергаем. Соответственно, ошибкой второго рода будет являться "пропуск" уходящего абонента и ошибочное принятие нулевой гипотезы.

Precision и recall не зависят, в отличие от accuracy, от соотношения классов и потому применимы в условиях несбалансированных выборок. Часто в реальной практике стоит задача найти оптимальный (для заказчика) баланс между этими двумя метриками. Классическим примером является задача определения оттока клиентов.

Обычно при оптимизации гиперпараметров алгоритма (например, в случае перебора по сетке *GridSearchCV*) используется одна метрика, улучшение которой мы и ожидаем увидеть на тестовой выборке. Существует несколько различных способов объединить precision и recall в агрегированный критерий качества. F-мера (в общем случае F_β) – среднее гармоническое precision и recall:

$$F_\beta = (1+\beta^2)\cdot\frac{\text{precision}\cdot\text{recall}}{(\beta^2\cdot\text{precision})+\text{recall}}.$$

β в данном случае определяет вес точности в метрике, и при β=1 это среднее гармоническое (с множителем 2, чтобы в случае precision = 1 и recall = 1 имеем F_1=1). F-мера достигает максимума при полноте и точности, равными единице, и близка к нулю, если один из аргументов близок к нулю. В sklearn есть удобная функция metrics.classification *report*, возвращающая recall, precision и F-меру для каждого из классов, а также количество экземпляров каждого класса.

AUC-ROC и AUC-PR

При конвертации вещественного ответа алгоритма (как правило, вероятности принадлежности к классу) в бинарную метку, мы должны выбрать какой-либо порог, при котором 0 становится 1. Естественным и близким кажется порог, равный 0.5, но он не всегда оказывается оптимальным, например, при вышеупомянутом отсутствии баланса классов.

Одним из способов оценить модель в целом, не привязываясь к конкретному порогу, является AUC-ROC (или ROC AUC) — площадь (Area Under Curve) под кривой ошибок (Receiver Operating Characteristic curve). Данная кривая представляет из себя линию от (0,0) до (1,1) в координатах True Positive Rate (TPR) и False Positive Rate (FPR):

$$\text{TPR}=\frac{\text{TP}}{\text{TP}+\text{FN}},\ \text{FPR}=\frac{\text{FP}}{\text{FP}+\text{TN}}.$$

TPR нам уже известна, это полнота (recall), а FPR показывает, какую долю из объектов negative класса алгоритм предсказал неверно. В идеальном случае, когда классификатор не делает ошибок (FPR = 0, TPR = 1), мы получим площадь под кривой, равную единице; в противном случае, когда классификатор случайно выдает вероятности классов, AUC-ROC будет стремиться к 0.5, так как классификатор будет выдавать одинаковое количество TP и FP. Каждая точка на графике соответствует выбору некоторого

порога. Площадь под кривой в данном случае показывает качество алгоритма (больше – лучше). Кроме этого, важной является крутизна самой кривой – мы хотим максимизировать TPR, минимизируя FPR, а значит, наша кривая в идеале должна стремиться к точке (0,1).

2.6. Практическое занятие №3. Построение моделей бинарной классификации и анализ их точности

Цель занятия:

Изучить основные элементы языка Python.

Научится использовать методы классификации: Дерево решений и Random forest.

Изучить метрики оценки качества получаемых решений.

Учебное задание

Постройте модель для прогнозирования вероятности спасения пассажиров Титаника. Для построения модели используйте метод машинного обучения *дерево решений* (DecisionTreeClassifier).

Технология выполнения учебного задания

1. Загружаем библиотеки pandas, numpy, matplotlib необходимые для решения задачи.

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

2. Загрузите данные, подготовленные на предыдущем практическом занятии.

```
tall = pd.read_csv("../Titanic/Data/Prak02.csv")
tall.head(8)
```

3. Как показывает предварительный анализ индекс пассажира, номер билета и имя пассажира нам никак не помогут, т.к. имя − это просто справочная информация, а по полю номер билета очень много пропущенной информации. Поэтому их нужно просто удалить. Для этого используем функцию drop() класса DataFrame.

```
tall = tall.drop([‘PassangerID’, 'Name','Ticket','Cabin'], axis=1)
tall.head(8)
```

4. В связи с тем, что мы используем методы машинного обучения, требуется сформировать массив независимых показателей и массив ответов для обучающей и тестовой выборок. Массив ответов хранится в поле Survived. Поэтому преобразуем поле Survived в массив ответов ytall, затем удалим это поле из таблицы tall и полученную таблицу преобразуем в массив независимых показателей. Для удаления поля используйте метод drop(). В качестве

аргументов используйте имя поля, axis (номер оси) = 1 (столбец), и inplace. Если inplace = True, то удаление столбца сохраняется в таблице tall.

```
clDrop='Survived'
ytall=tall[clDrop].values
tall.drop(clDrop, axis=1, inplace=True)
xtall=tall.values
```

5. Для проверки модели будем использовать прием «отложенной выборки». Поэтому полученные массивы разбиваем на обучающие и тестовые. Поступим следующим образом: 25% последних записей используем как тестовую выборку. Используем функцию len() для вычисления числа записей структуры tall.

```
Mtest=int(len(tall)/4)
Mtrain= int(len(tall)-Mtest)
print len(tall), Mtrain, Mtest
```

Создаем массив индексов для тестовой (ttest) и обучающей (ttrain) выборок с помощью функции arange библиотеки numpy и распечатываем их.

```
ttrain = np.arange(Mtrain)
ttest = np.arange(Mtest) + Mtrain
print ttrain
print ttest
```

Распечатайте массив ответов тестовой выборки. Для этого используйте следующий оператор:

```
print ytall[ttest]
```

Результат представлен на рис. 2.3.

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

Рис. 2.3. Индексы тестовой выборки

6. После обработки данных приступаем к построению модели, но для начала нужно определиться с тем, как мы будем проверять точность полученной модели. Обучать модель будем на обучающей выборке, после чего проверим ее на тестовой выборке.

Загрузим нужные нам библиотеки:

```
from sklearn.metrics import  roc_auc_score, accuracy_score, roc_curve
from sklearn import tree
```

7. Построим модель прогноза вероятности спасения пассажиров методом ближайших соседей. Класс метода «Дерево решений» имеет конструктор:

class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)

Наиболее важными аргументами являются:

- max_depth – максимальная глубина дерева;
- max_features – максимальное число признаков, по которым ищется лучшее разбиение в дереве (это нужно потому, что при большом количестве признаков будет «дорого» искать лучшее (по критерию типа прироста информации) разбиение среди *всех* признаков);
- min_samples_leaf – минимальное число объектов в листе. У этого параметра есть понятная интерпретация: скажем, если он равен 5, то дерево будет порождать только те классифицирующие правила, которые верны как минимум для 5 объектов.

При создании объекта укажем параметры max_depth =8, max_features =7. Для обучения будем использовать метод fit(). В качестве аргументов используем массив независимых показателей xtall[ttrain] и ответы ytall[ttrain] обучающей выборки.

```
dtc = tree.DecisionTreeClassifier(max_depth =8,max_features =7)
dtc=dtc.fit(xtall[ttrain], ytall[ttrain])
```

После обучение выполним прогнозирование как на тестовой так и на обучающей выборке, используя функцию predict_proba(). Функция predict_proba() возвращает вероятности принадлежности объектов к классу. После это оценим точность прогноза с помощью функции score(). Результаты распечатаем.

```
ytrainDtc = dtc.predict_proba(xtall[ttrain])
ytestDtc = dtc.predict_proba(xtall[ttest])
sc0= dtc.score(xtall[ttrain], ytall[ttrain])
sc1= dtc.score(xtall[ttest], ytall[ttest])
print("DesicionTree", "i_fold=", " Точность обучения", sc0, " Точность
      тестовая", sc1)
```

8. Распечатаем результаты прогноза, полученные на тестовой выборке. Для этого будем использовать цикл:

```
for i in np.arange(len(ttest)):
   it=ttest[i]
   print("Пассажир", it, "Вероятности", ytestDtc[i],"Правильный ответ ",
      ytall[it])
```

Сравним полученные результаты с ответами. Как видно, первая стока показывает вероятность спасения пассажира равна 0, и это соответствует

правильному ответу 1. Вторая строка аналогична первой. Третья строка показывает вероятность спасения пассажира 0.6595, а вероятность того что пассажир утонул 0.3404. Правильный ответ: пассажир не утонул. Четвертая строка дает правильный ответ, а пятая строка уже дает неправильный результат.

Распечатка результатов прогноза представлена на рис. 2.4.

```
Пассажир 669 Вероятности [0. 1.] Правильный ответ  1
Пассажир 670 Вероятности [0. 1.] Правильный ответ  1
Пассажир 671 Вероятности [0.65957447 0.34042553] Правильный ответ  0
Пассажир 672 Вероятности [0.8 0.2] Правильный ответ  0
Пассажир 673 Вероятности [0.90243902 0.09756098] Правильный ответ  1
Пассажир 674 Вероятности [1. 0.] Правильный ответ  0
Пассажир 675 Вероятности [1. 0.] Правильный ответ  0
Пассажир 676 Вероятности [0.90243902 0.09756098] Правильный ответ  0
Пассажир 677 Вероятности [1. 0.] Правильный ответ  1
Пассажир 678 Вероятности [1. 0.] Правильный ответ  0
```

Рис. 2.4. Вероятности принадлежности пассажиров к классам для тестовой выборки

9. Попробуем сами оценить точность решения. Для этого нам нужно посчитать число правильно спрогнозированных пассажиров, которые спаслись и утонули. Для оценки точности создадим функцию. В качестве аргументов будем передавать вероятности прогноза принадлежности пассажира к классу 1 («Утонул») и пороговое значение. Если вероятность прогноза превышает пороговое значение, значит пассажир принадлежит классу 1.

```
def estim(y_prob,y_exact, pr):
    kall=y_prob.shape[0]
    tp=0
    fp=0
    tn=0
    fn=0
    kr=0
    for i in np.arange(kall):
        if(y_prob[i] > pr):
            if(y_exact[i] == 1 ):
                tp+=1
            else:
                fp+=1
        else:
            if(y_exact[i] == 0 ):
                tn+=1
            else:
                fn+=1
```

```
kr=1.*(tp+tn)/kall
kr1=1.*tp/(tp+fp)
kr0=1.*tn/(tn+fn)
rr1=1.*tp/(tp+fn)
rr0=1.*tn/(tn+fp)
print("Precission: ",kr0, "класс 0 спаслись", tn+fn, " Из них правильно
   спрогнозировано =", tn, ", ошибочно=", fn)
print("Precission: ", kr1,"класс 1 утонуло", tp+tn, " Из них правильно
   спрогнозировано =", tp, ", ошибочно=", tn)
print("Recall: ", rr0,"класс 0 спаслись")
print("Recall: ", rr1,"класс 1 утонуло")
print("Точность распознавания ",kr, " precission", "recall")
return  kr
```

Здесь tn – число правильно идентифицированных пассажиров, как спасшихся, fn – число пассажиров, которые были неправильно идентифицированы, как спасшиеся, kall – общее число пассажиров, tp – число правильно идентифицированных пассажиров, как утонувшие, fp – число пассажиров, которые были неправильно идентифицированы, как утонувшие. Точность прогноза равна сумме tp и tn, деленной на общее число пассажиров.

10. Оцените точность прогноза с помощью функции estim(). Для этого используем оператор:

```
estim(ytestDtc[:,1], ytall[ttest], 0.5)
```

Обратите внимание, что в качестве прогнозных значений мы передаем второй столбец массива ytestDtc. Пороговое значение равно 0.5.

В результате получим следующие сообщения:

```
Precission:  0.8322147651006712 класс 0 спаслись 149  Из них правильно
спрогнозировано = 124 , ошибочно= 25
Precission:  0.7808219178082192 класс 1 утонуло 73  Из них правильно спро-
гнозировано = 57 , ошибочно= 16
Recall:  0.8857142857142857 класс 0 спаслись
Recall:  0.6951219512195121 класс 1 утонуло
Точность распознавания  0.8153153153153153
```

Как видно точность распознавания соответствует точности, полученной выше с помощью функции score().

11. Библиотеки машинного обучения содержат достаточно много функций позволяющих оценить точность. Одна из них accuracy_score(). Но в этой функции вместо вероятностной оценки нужно использовать бинарную оценку прогнозирования спасения пассажиров. Для этого будем использовать функцию argmax(), которая возвращает индекс элемента массива с максимальным значением.

```
pred=[]
for x in ytestDtc:
   pred.append(np.argmax(x))
```

Проверим правильно ли мы оценили точность прогноза.

```
acc= accuracy_score(ytall[ttest], pred)
print("Точность (accuracy)", acc)
```

Как можно видеть, полученный результат соответствует ранее полученным оценкам.

12. Используем для оценки точности функции classification_report() и confusion_matrix(). Для их использования необходимо сформировать метки классов (массив lab).

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
nclass=2
lab=["Спаслись", "Утонули"]
print(classification_report( ytall[ttest], pred, target_names=lab, digits=4))
print(confusion_matrix( ytall[ttest] ,  pred, labels=range(nclass)))
```

Результат представлен на рис. 2.5.

```
                precision    recall  f1-score   support

       Выжили     0.8322    0.8857    0.8581       140
      Утонули     0.7808    0.6951    0.7355        82

avg / total     0.8132    0.8153    0.8128       222

[[124  16]
 [ 25  57]]
```

Рис. 2.5 Отчет по оценке точности и матрица неточностей

Как видно, оценки precission и recall соответствует оценкам, полученным нами с помощью процедуры estim().

Матрица неточностей содержит результаты, полученные нами ранее. Диагональные элементы представляют число объектов, классифицированных правильно. Вне-диагональные элементы представляют число неправильно классифицированных элементов.

13. При конвертации вероятности принадлежности к классу в бинарную метку, мы должны выбрать какой-либо порог, при котором 0 становится 1. Естественным и близким кажется порог, равный 0.5, но он может оказаться не оптимальным.

Одним из способов оценить модель в целом, не привязываясь к конкретному порогу, является метрика AUC-ROC (или ROC AUC), которая представляет площадь под кривой ошибок. Данная кривая представляет из себя линию от (0;0) до (1;1) в координатах True Positive Rate (TPR) и False Positive Rate (FPR).

Для построения ROC AUC кривой используйте функцию roc_curve(). У этой функции три обязательных аргумента: массив правильных ответов, массив вероятностей ответов положительного класса и индекс класса положительных примеров. В качестве положительного класса используется 2 класс (индекс равен 1). Функция возвращает три массива:

1) массив роста положительных примеров, классифицированные как отрицательные;

2) массив роста верно классифицированных положительных примеров (так называемые истинно положительные случаи);

3) массив пороговых значений.

```
plt.figure(figsize=(12, 10))
fpr, tpr, thresholds = roc_curve(ytall[ttest], ytestDtc[:,1], pos_label=1)
plt.plot(fpr, tpr)
plt.show()
```

Результаты представлены на рис. 2.6.

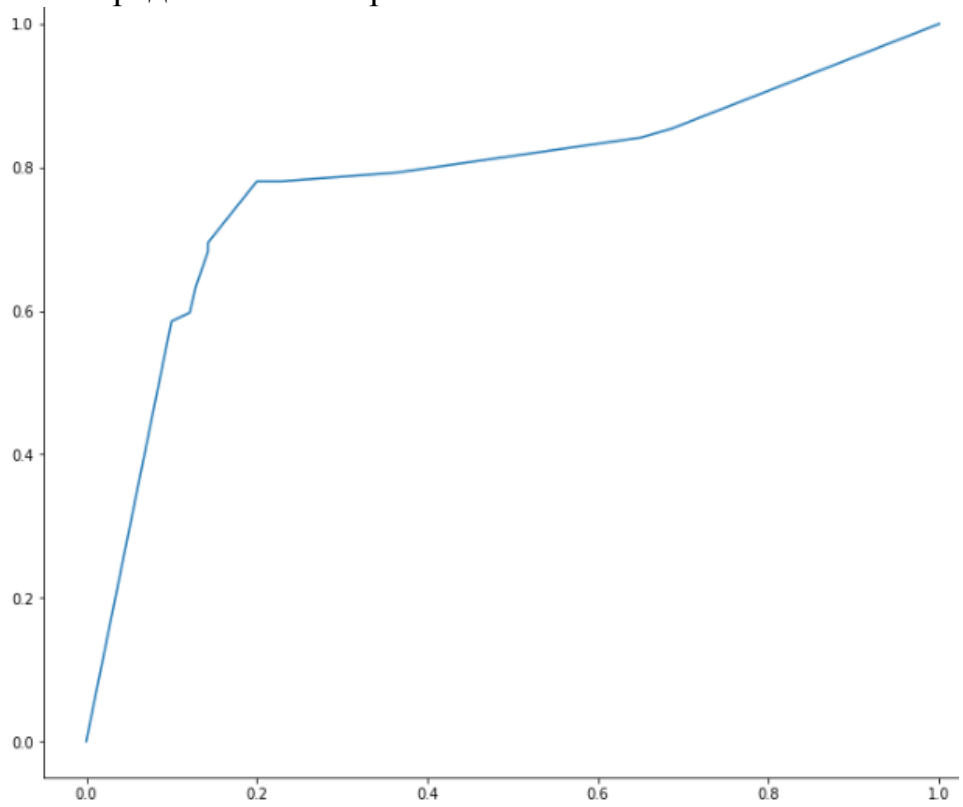

Рис. 2.6. ROC AUC кривая

Обычно для сравнения на рисунке также еще показывают ROC AUC кривую, площадь под кривой которой равна 0.5. Эта кривая представляет прямую линию проходящей по диагонали от точки (0;0) до точки (1;1). Величина ROC AUC метрики, равной 0.5, соответствует точности прогноза методом подбрасывания монетки.

Добавим такую прямую линию, кроме того откорректируем максимальные и минимальные значения осей x и y. Подпишем оси: ось x − False Positive Rate и ось y – True Positive Rate.

```
plt.figure(figsize=(12, 10))
fpr, tpr, thresholds = roc_curve(ytall[ttest], ytestDtc[:,1], pos_label=1)
lw = 2
plt.plot(fpr, tpr, lw=lw, label='ROC curve ')
plt.plot([0, 1], [0, 1])
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve')
plt.show()
```

В результате полуим график на рис. 2.7.

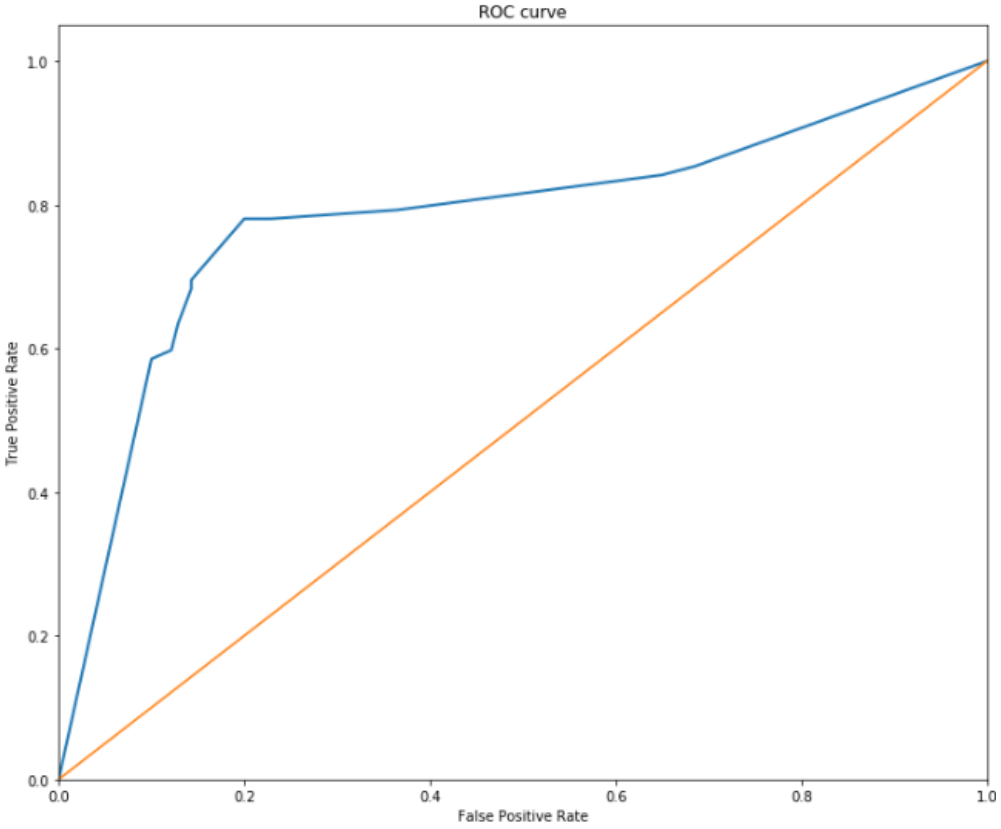

Рис. 2.7. ROC AUC кривая

14. Для вычисления площади под кривой будем использовать функцию roc_auc_score().

```
roc= roc_auc_score(ytall[ttest], ytestDtc[:,1])
print("roc-auc", roc)
```

Результат представлен ниже:

```
roc-auc 0.7819686411149824
```

Вместо вероятностной оценки можно использовать бинарную оценку прогнозирования спасения пассажиров. Для этого вычислите бинарную оценку прогнозирования на основе вероятностную оценки. Для этого будем

использовать функцию argmax(), которая возвращает индекс элемента массива с максимальным значением. Используйте код, представленный ниже.

```
print("Mean Score")
pred=[]
for x in ytestDtc:
   ind=np.argmax(x)
   print("index", ind, "x", x)
   pred.append(ind))
```

Результаты представлены на рис. 2.8.

```
Mean Score
index 1 x [0. 1.]
index 1 x [0. 1.]
index 0 x [0.65957447 0.34042553]
index 0 x [0.8 0.2]
index 0 x [0.90243902 0.09756098]
index 0 x [1. 0.]
index 0 x [1. 0.]
index 0 x [0.90243902 0.09756098]
index 0 x [1. 0.]
index 0 x [1. 0.]
index 0 x [1. 0.]
index 1 x [0.13043478 0.86956522]
index 0 x [0.65957447 0.34042553]
index 0 x [0.90243902 0.09756098]
index 0 x [1. 0.]
index 0 x [1. 0.]
```

Рис. 2.8. Результаты создания бинарного массива ответов

После этого снова вычислим снова кривую.

```
plt.figure(figsize=(12, 10))
fpr, tpr, thresholds = roc_curve(ytall[ttest], pred, pos_label=1)
plt.plot(fpr, tpr, lw=2, label='ROC curve ')
plt.plot([0, 1], [0, 1])
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve')
plt.show()
roc= roc_auc_score(ytall[ttest],pred)
print("roc-auc", roc
```

Результаты представлены на рис. 2.9.

Мы видим, что ROC AUC кривая отличается от кривой, полученной с помощью вероятностной оценке.

После этого определим метрику ROC AUC.

```
roc= roc_auc_score(ytall[ttest],pred)
```

```
print("roc-auc", roc)
```

В результате получим:

```
roc-auc 0.7761324041811847
```

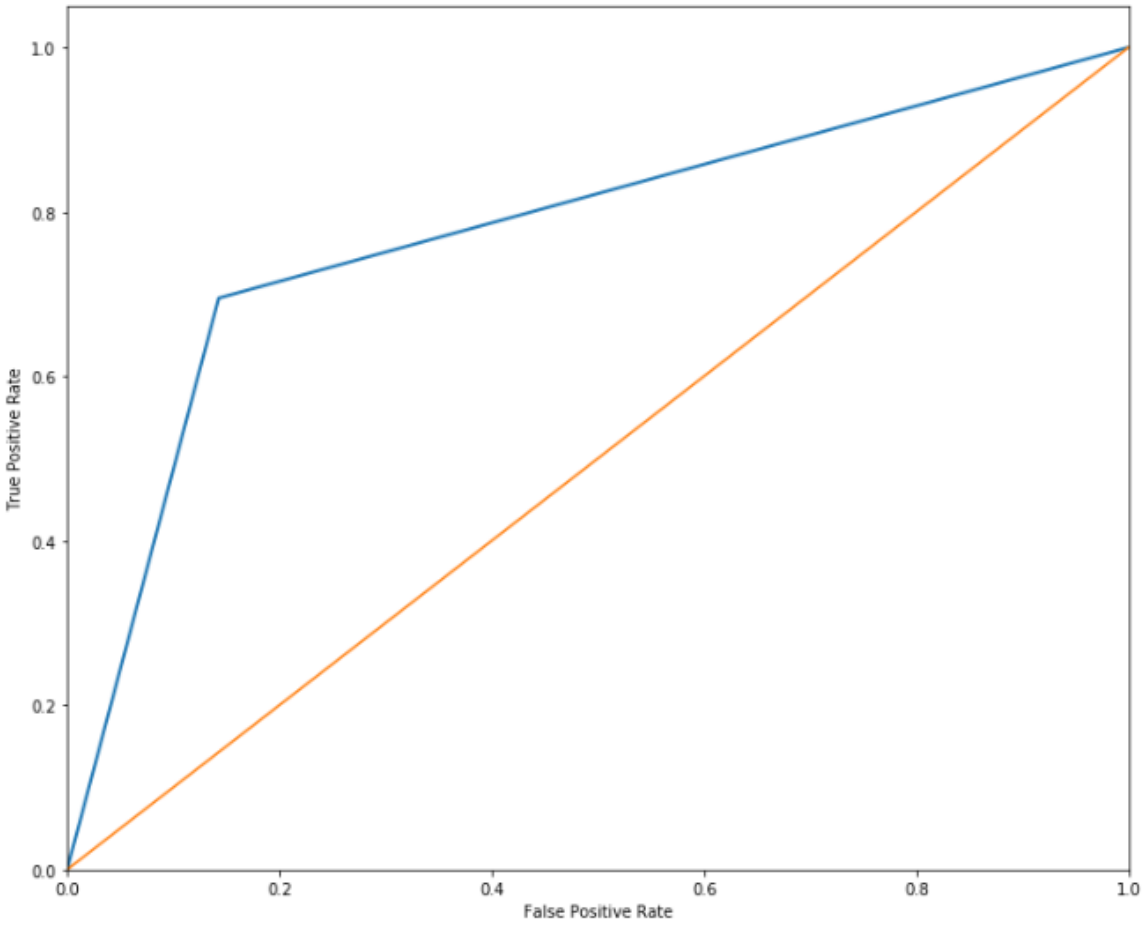

Рис. 2.9. ROC AUC кривая

Таким образом, вероятностная оценка дает более высокие значения метрики ROC AUC, нежели бинарная.

15. У классификатора DecisionTreeClassifier важными аргументами являются max_depth (максимальная глубина дерева) и max_features (максимальное число признаков).

Выберем оптимальные значения для нашей задачи. Для этого будем использовать циклы, внутри которых строить модель с разной глубиной и разным числом признаков. После обучения вычислим точность прогноза на обучающей выборке и точность прогноза на тестовой выборке 3.

```
gr=np.arange(1,8,1)
gr1=np.arange(1,12,1)
for i in gr:
    for j in gr1:
        dtc = tree.DecisionTreeClassifier(max_depth =j,max_features =i) #,
        min_samples_leaf =1)
        dtc=dtc.fit(xtall[ttrain], ytall[ttrain])
```

```
ytrainDtc = dtc.predict_proba(xtall[ttrain])
ytestDtc = dtc.predict_proba(xtall[ttest])
sc0= dtc.score(xtall[ttrain], ytall[ttrain])
sc1= dtc.score(xtall[ttest], ytall[ttest])
print("DesicionTree max_depth", j, "max_features", i, " Точность
обучения", sc0, " Точность тестовая", sc1)
```

Результаты представлены на рис. 2.10.

```
DesicionTree max_depth 5 max_features 6  Точность обучения 0.8460388639760837  Точность тестовая 0.8558558558558559
DesicionTree max_depth 6 max_features 6  Точность обучения 0.8445440956651719  Точность тестовая 0.8468468468468469
DesicionTree max_depth 7 max_features 6  Точность обучения 0.8834080717488789  Точность тестовая 0.8288288288288288
DesicionTree max_depth 8 max_features 6  Точность обучения 0.9013452914798207  Точность тестовая 0.8063063063063063
DesicionTree max_depth 9 max_features 6  Точность обучения 0.9267563527653214  Точность тестовая 0.7927927927927928
DesicionTree max_depth 10 max_features 6  Точность обучения 0.9267563527653214  Точность тестовая 0.779279279279279
DesicionTree max_depth 11 max_features 6  Точность обучения 0.9521674140508222  Точность тестовая 0.779279279279279
DesicionTree max_depth 1 max_features 7  Точность обучения 0.7862481315396114  Точность тестовая 0.7882882882882883
DesicionTree max_depth 2 max_features 7  Точность обучения 0.7892376681614349  Точность тестовая 0.8018018018018018
DesicionTree max_depth 3 max_features 7  Точность обучения 0.8295964125560538  Точность тестовая 0.8423423423423423
DesicionTree max_depth 4 max_features 7  Точность обучения 0.8340807174887892  Точность тестовая 0.8468468468468469
DesicionTree max_depth 5 max_features 7  Точность обучения 0.8460388639760837  Точность тестовая 0.8558558558558559
DesicionTree max_depth 6 max_features 7  Точность обучения 0.8714499252615845  Точность тестовая 0.8288288288288288
DesicionTree max_depth 7 max_features 7  Точность обучения 0.8953662182361734  Точность тестовая 0.7972972972972973
DesicionTree max_depth 8 max_features 7  Точность обучения 0.9147982062780269  Точность тестовая 0.7972972972972973
DesicionTree max_depth 9 max_features 7  Точность обучения 0.9237668161434978  Точность тестовая 0.7837837837837838
DesicionTree max_depth 10 max_features 7  Точность обучения 0.9327354260089686  Точность тестовая 0.779279279279279
DesicionTree max_depth 11 max_features 7  Точность обучения 0.9417040358744395  Точность тестовая 0.774774774774774
```

Рис. 2.10. Результаты обучения модели

Исходя из полученных результатов лучшими аргументами являются:

1) max_depth 5 и max_features 6;

2) max_depth 5 и max_features 7.

Точность на тестовой выборке для этих параметров максимальна и равна 0.85585.

16. Далее построим графики изменения точности на обучающей и тестовой выборке в зависимости от максимальной глубины дерева. снова используем цикл для перебора значений, max_features принимается равным 7.

```
scTs=[]
scTr=[]
gr=np.arange(1,12,1)
for i in gr:
    dtc = tree.DecisionTreeClassifier(max_depth =i,max_features = 7)
    dtc=dtc.fit(xtall[ttrain], ytall[ttrain])
    ytrainDtc = dtc.predict_proba(xtall[ttrain])
    ytestDtc = dtc.predict_proba(xtall[ttest])
    sc0= dtc.score(xtall[ttrain], ytall[ttrain])
    sc1= dtc.score(xtall[ttest], ytall[ttest])
    scTs.append(sc1)
```

```
scTr.append(sc0)
print("DesicionTree max_depth", i, " Точность обучения", sc0, "
    Точность тестовая", sc1)
```

Далее полученные зависимости scTs и scTr отображаются графически с помощью следующего кода:

```
plt.figure( figsize=(16,10))
plt.plot(gr, scTr, label="Точность на обучающей выборке")
plt.plot(gr, scTs,label="Точность на тестовой выборке")
plt.xlabel('Максимальная глубина',  fontsize=18)
plt.ylabel('Точность',  fontsize=18)
plt.legend(fontsize=18)
plt.rc('xtick', labelsize=18)
plt.rc('ytick', labelsize=18)
plt.savefig("DT_scor.jpg", dpi=400)
plt.show()
```

Результат представлен на рис. 2.11.

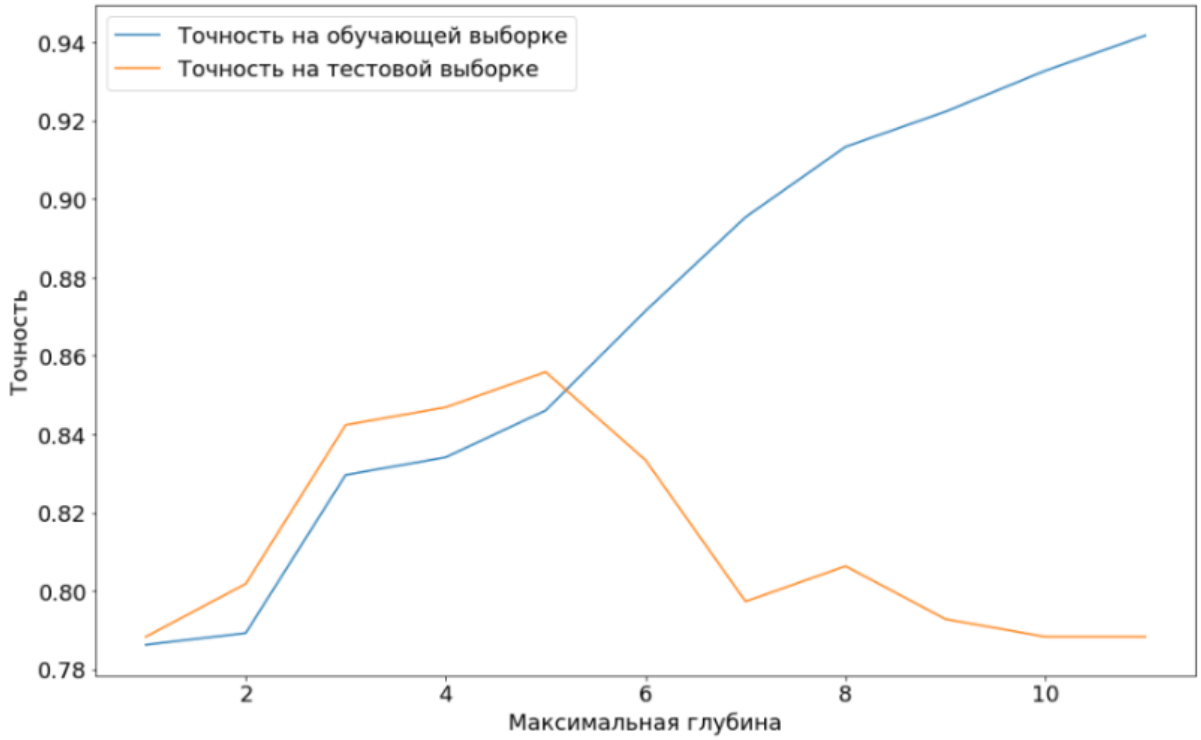

Рис. 2.11. Изменение точности на обучающей и тестовой выборке в зависимости от максимальной глубины дерева

Как видно из рисунка точность прогнозирования на тестовой выборке падает после того как максимальная глубина дерева превысило значение 5. По сути начиная с максимальной глубины равной 6 мы получаем переобученные модели.

17. Крос-валидация является более эффективным способом оценки качества моделей. Для создания фолдов (подвыборок) используется класс KFold.

В качестве аргументов задается длина исходной выборки, n_folds – количество фолдов, shuffle – логический параметр (если он равен True, то данные перед разбиением перемешиваются), random_state – псевдо случайное число, которое используется для сдвига в генераторе случайных чисел. Генератор случайных чисел используется, если shuffle равно True. Выполним разбиение нашей выборки на 5 фолдов.

```
from sklearn.cross_validation import KFold
ntrain = tall.shape[0]
SEED = 111123435 # for reproducibility
NFOLDS = 5 # set folds for out-of-fold prediction
kf = KFold(ntrain, n_folds= NFOLDS, random_state=SEED, shuffle
    =True)
```

KFold возвращает объект kf, которые содержит индексы фолдов (обучающих и тестовых).

17. Выполним поиска оптимальных параметров. Для этого используем циклы с перебором параметров max_depth (от 1до 11) и max_features (от 1 до 7). В теле этого двойного цикла вложим цикл перебора фолдов, нидексы которых берем из объекта kf. Результаты прогноза сохраняем в массиве pred_test, который создается перед циклом методом np.zeros(). Данный метод создает двумерный нулевой массив с указанными размерами. В результате в процессе работы для каждого обучающего фолда, создается модель, которая проверяется на тестовом фолде. Результаты прогноза сохраняются в массиве pred_test. После завершения цикла мы имеем массив с тестовыми прогнозами, поэтому мы можем вычислить метрику ROC AUC. Все эти вычисления повторяются ждя каждой комбинации i, j. Наилучшие (сточки зрения метрики ROC AUC) значения i, j сохраняются.

```
gr=np.arange(1,8,1)
gr1=np.arange(1,12,1)
acc=0
for i in gr:
    for j in gr1:
        sc=0
        pred_test = np.zeros((ntrain,2))
        for  (ttrain, ttest) in kf:
            dtc = tree.DecisionTreeClassifier(max_depth = j,max_features = i)
            dtc=dtc.fit(xtall[ttrain], ytall[ttrain])
            pred_test[ttest] = dtc.predict_proba(xtall[ttest])
        ac= roc_auc_score(ytall, pred_test[:,1])
        if ac > acc:
            acc=ac
            md=j
            mf=i
```

```
print("DesicionTree: max_depth", j, "max_features", i,  " Точность
ROC AUC", ac)
```

После выполнения поиска, распечатаем наилучшие значения параметров дерева решений.

```
print("best: max depth", md, "max_features", mf,  "Наилучшая метрика
    ROC AUC", acc  )
```

Выполним построение моделей с наилучшими параметрами настройки.

```
pred_test = np.zeros((ntrain,2))
for  (ttrain, ttest) in kf:
   dtc = tree.DecisionTreeClassifier(max_depth = 5,max_features = 3) #,
      min_samples_leaf =1)
   dtc=dtc.fit(xtall[ttrain], ytall[ttrain])
   pred_test[ttest] = dtc.predict_proba(xtall[ttest])
```

19. Построим кривую ROC AUC, используя уже знакомый нам код:

```
plt.figure(figsize=(12, 10))
fpr, tpr, thresholds = roc_curve(ytall, pred_test[:,1], pos_label=1)
plt.plot(fpr, tpr, lw=2, label='ROC curve ')
plt.plot([0, 1], [0, 1])
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve')
plt.show()
```

Результат представлен на рис. 2.12.

Вычислим метрику ROC AUC

```
roc= roc_auc_score(ytall, pred_test[:,1])
print("roc-auc", roc)
```

Результат (ROC AUC= 0.84014) соответствует метрике, полученный выше.

20. Вычислим другие характеристики точности. Для этого требуется вычислить категориальную оценку прогноза. Используем уже известный нам код:

```
print("Binary output")
pred=[]
for x in pred_test:
   pred.append(np.argmax(x))
```

Далее используем уже известные нам методы для построения отчета и матрицы неточностей.

```
print(classification_report( ytall, pred, target_names=lab, digits=4))
print(confusion_matrix( ytall ,  pred, labels=range(2)))
```

Результат представлен на рис. 2.13.

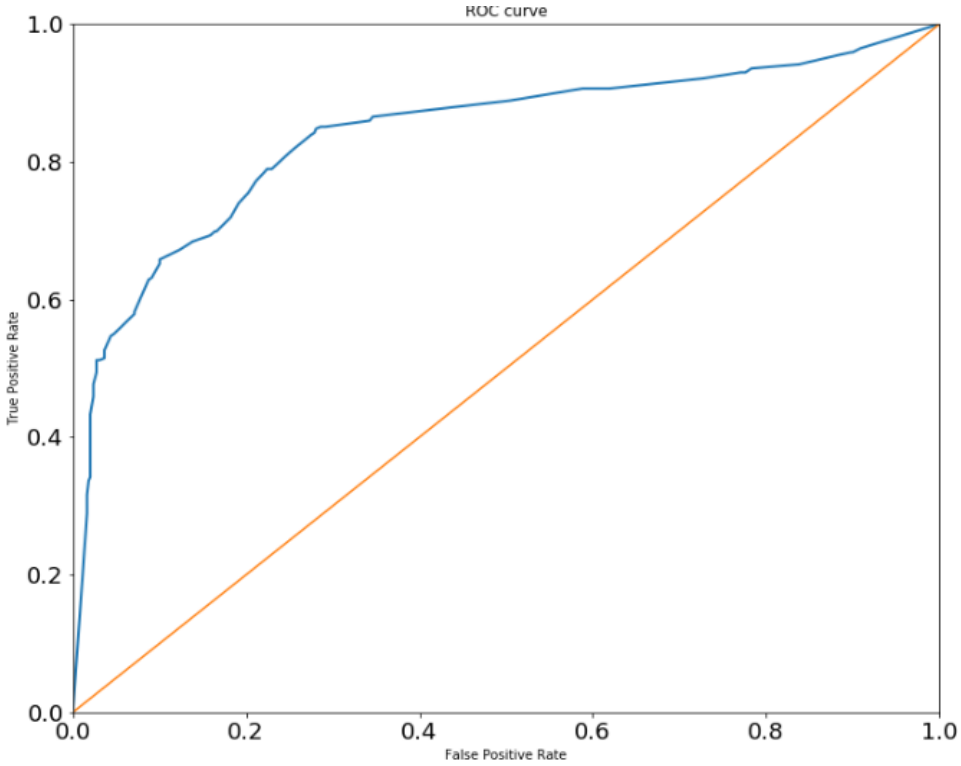

Рис. 2.12. Кривая ROC AUC

	precision	recall	f1-score	support
Спаслись	0.8111	0.8761	0.8424	549
Утонули	0.7718	0.6725	0.7188	342
avg / total	0.7960	0.7980	0.7949	891

```
[[481  68]
 [112 230]]
```

Рис. 2.13. Точность прогнозирования и матрица неточностей

21. Для улучшения модели и лучшей ее интерпретации требуется определить какие показатели наиболее значимыми и важными для прогнозирования вероятности спасения пассажиров. Основная цель определения важности переменных − это оценка того, какие переменные, когда, где и как влияют на решаемую задачу.

Для определения коэффициентов важности показателей используется атрибут feature_importances_.

```
imp= dtc.feature_importances_
imp
```

Результат представлен на рис. 2.14.

Для того чтобы этот массив был более понятным оформим его в виде таблиц rfImp, в которую добавим имена показателей на русском языке и

отсортируем строки таблицы по убыванию с помощью функции sort_values().

```
array([0.20269018, 0.4483921 , 0.10043018, 0.07438183, 0.00297763,
       0.16456656, 0.00656152])
```

Рис. 2.14. Коэффициенты важности показателей

В качестве аргументов задаем имя столбца, по которому нужно отсортировать, и порядок сортировки.

```
imp= dtc.feature_importances_
rfImp = pd.DataFrame(imp, columns=['imp'])
title1=tall.columns
rfImp['title']=title1
rfImp = rfImp.sort_values('imp', ascending = False)
rfIm.head(7)
```

Результат представлен на рис. 2.15.

	imp	title
1	0.448392	Пол
0	0.202690	Социальный статус
5	0.164567	Цена билета
2	0.100430	Возраст
3	0.074382	Число родственников 2-го порядка
6	0.006562	Порт посадки
4	0.002978	Число родственников 1-го порядка

Рис. 2.15. Коэффициенты значимости показателей для модели «Дерево решений»

Визуализация модели (только если установлены библиотеки pyplotplus)

22. Для визуализации модели будем использовать функцию export_graphviz() класса Дерево решений. Этот метод формирует текстовый массив, который можно преобразовать в графический с помощью метода graph_from_dot_data() библиотеки pydotplus.

Функция export_graphviz() имеет следующие аргументы:

1) ссылка на объект обученной модели;

2) feature_names – массив имен показателей;

3) class_names –массив имен целевых классов;

4) out_file – имя выходного файла;

5) filled – если параметр имеет значение True, то для каждого узла указывается приоритетный класс;

6) rounded – если параметр имеет значение True, то узлы рисуются с закругленными углами и используются шрифт Helvetica вместо Times-Roman.

Функция .graph_from_dot_data() имеет всего один аргумент, а именно, тот массив данных, который генерирует функция export_graphviz().

Для сохранения рисунка в файл используется метод write_png().

```
import pydotplus
# Visualize data
dot_data = tree.export_graphviz(dtc, feature_names=ti-
    tle1,class_names=["Выжили", "Утонули"] , out_file=None,
    filled=True,   rounded=True)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_png("../Titanic/Data/DC.jpg")
```

Для того чтобы посмотреть дерево решений, не используя графический редактор, прочитаем графический файл и отобразим его на картинке с помощью библиотеки matplotlib. Сначала используя функцию imread() прочитаем наш файл. При этом функция возвращает числовой массив типа numpy.array. Для изображений в оттенках серого возвращается массив MxN. Для изображений RGB возвращаемое значение равно MxNx3. Для изображений RGBA возвращаемое значение − MxNx4. Формат RGBA похож по синтаксису на RGB, но включает в себя дополнительный канал, задающий прозрачность элемента.

```
import matplotlib.image as mpimg
img=mpimg.imread("../Titanic/Data/DC.jpg")
img
```

Результат представлен на рис.2.16.

```
array([[[255, 255, 255, 255],
        [255, 255, 255, 255],
        [255, 255, 255, 255],
        ...,
        [255, 255, 255, 255],
        [255, 255, 255, 255],
        [255, 255, 255, 107]],

       [[255, 255, 255, 255],
        [255, 255, 255, 255],
        [255, 255, 255, 255],
        ...,
        [255, 255, 255, 255],
        [255, 255, 255, 255],
        [255, 255, 255, 107]],
```

Рис. 2.16 Числовой массив графического файла

Далее используем класс figure и метод imshow() для отображения нашего графического файла.

```
import matplotlib.image as mpimg
img=mpimg.imread("../Titanic/Data/DC.jpg")
```

```
fig = plt.figure(figsize=(100,100))
imgplot = plt.imshow(img)
```

Метод Random Forest

23. При использовании кросс-валидации мы разбили выборку на 5 фолдов и выполнили построение 5 моделей. Поэтому при появлении новых данных для их классификации у нас имеется 5 моделей. Встатет вопрос как их использовать. Самый простой путь выполнить классификацию с помощью 5 моделей, а затем провести осреднение. Таким образом мы используем самый простой вариант ансамбля моделей.

Random forest представляет метод, который строит не одно дерево решений, а много деревьев (лес деревьев). Таким образом, наша задача − построить модель прогноза вероятности спасения пассажиров методом случайных деревьев (Random forest). Описание класса метода случайных деревьев имеет конструктор:

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=10,
criterion='gini', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features='auto', max_leaf_nodes=None, min_impurity_split=1e-07,
bootstrap=True, oob_score=False, n_jobs=1,
random_state=None, verbose=0, warm_start=False,
class_weight=None)
```

Описание некоторых наиболее важных аргументов представлено ниже.

1) n_estimators – число деревьев. Чем больше деревьев, тем лучше качество, но время настройки и работы RF также пропорционально увеличиваются.

2) max_features – число признаков для выбора расщепления. При увеличении max_features увеличивается время построения леса, а деревья становятся «более однообразными». По умолчанию он равен sqrt(n) в задачах классификации и n/3 в задачах регрессии. Это самый важный параметр! Его настраивают в первую очередь (при достаточном числе деревьев в лесе). Он не может быть больше числа независимых показателей. В нашем случае не больше 7.

3) min_samples_split – минимальное число объектов, при котором выполняется расщепление. Этот параметр, как правило, не очень важный и можно оставить значение по умолчанию (2). График качества на контроле может быть похожим на «расчёску» (нет явного оптимума). При увеличении параметра качество на обучении падает, а время построения RF сокращается.

4) min_samples_leaf – ограничение на число объектов в листьях. Всё, что было описано про min_samples_split, годится и для описания этого параметра. Часто можно оставить значение по умолчанию (1).

5) max_depth – максимальная глубина деревьев. Ясно, что чем меньше глубина, тем быстрее строится и работает метод. При увеличении глубины резко возрастает качество обучения, но и на тестовой выборке оно, как правило, увеличивается. Рекомендуется использовать максимальную глубину (кроме случаев, когда объектов слишком много и получаются очень глубокие деревья, построение которых занимает значительное время). При использовании неглубоких деревьев изменение параметров, связанных с ограничением числа объектов в листе и для деления, не приводит к значимому эффекту (листья и так получаются «большими»). Неглубокие деревья рекомендуют использовать в задачах с большим числом шумовых объектов (выбросов).

6) criterion – критерий расщепления. По смыслу это очень важный параметр, но по факту здесь нет вариантов выбора. Для классификации реализованы критерии gini и entropy, которые соответствуют классическим критериям расщепления: Джини и энтропийному.

Из всех параметров для нас наиболее интересны два: число деревьев и максимальная глубина дерева. Поэтому на первом шаге выполним поиск наилучших параметров.

```
gr=np.arange(1,45,1)
gr1=np.arange(1,12,1)
acc=0
for i in gr:
    for j in gr1:
        scc=0
        pred_test = np.zeros((ntrain,2))
        for  (ttrain, ttest) in kf:
            rfc = RandomForestClassifier(n_estimators = i, max_depth =j)
            rfc.fit(xtall[ttrain], ytall[ttrain])
            pred_test[ttest] = rfc.predict_proba(xtall[ttest])
            scc+= rfc.score(xtall[ttest], ytall[ttest])
        ac= roc_auc_score(ytall, pred_test[:,1])
        scc=scc/NFOLDS
        if ac > acc:
            acc=ac
            md=j
            mf=i
            print("Random Forest: max_depth", j, "n_estimators", i,  " ROC
        AUC", ac, " Точность", scc)
```

При поиске наилучших параметров вычисляется точность прогноза, как среднеарифметическое точность прогнозирования отдельных фолдов. Результаты поиска представлены на рис. 2.18.

```
Random Forest: max_depth 1 n_estimators 1  ROC AUC 0.7090217194473738  Точность 0.67791726821919
Random Forest: max_depth 2 n_estimators 1  ROC AUC 0.732320327229732  Точность 0.695894796309082
Random Forest: max_depth 3 n_estimators 1  ROC AUC 0.7736661021101631  Точность 0.74193082669010
Random Forest: max_depth 4 n_estimators 1  ROC AUC 0.7858173819491048  Точность 0.74076956876530
Random Forest: max_depth 5 n_estimators 1  ROC AUC 0.7951698462915029  Точность 0.77099993722930
Random Forest: max_depth 6 n_estimators 1  ROC AUC 0.8043359004676233  Точность 0.78784759274370
Random Forest: max_depth 3 n_estimators 2  ROC AUC 0.8214882987675625  Точность 0.75988324650053
Random Forest: max_depth 4 n_estimators 2  ROC AUC 0.8296743680695364  Точность 0.77780428096164
Random Forest: max_depth 5 n_estimators 2  ROC AUC 0.8367180093524644  Точность 0.79457033456782
Random Forest: max_depth 6 n_estimators 2  ROC AUC 0.8512713173340151  Точность 0.80918335321072
Random Forest: max_depth 5 n_estimators 3  ROC AUC 0.8575666549494564  Точность 0.81369028937292
Random Forest: max_depth 6 n_estimators 4  ROC AUC 0.8598941190255542  Точность 0.81927060448182
Random Forest: max_depth 6 n_estimators 5  ROC AUC 0.8628447256574954  Точность 0.81254158558784
Random Forest: max_depth 9 n_estimators 7  ROC AUC 0.863518465258471  Точность 0.821511518423199
Random Forest: max_depth 4 n_estimators 10  ROC AUC 0.8647514353582803  Точность 0.8181909484652
Random Forest: max_depth 8 n_estimators 10  ROC AUC 0.8676541079474643  Точность 0.8271169418115
Random Forest: max_depth 7 n_estimators 13  ROC AUC 0.869464949562735  Точность 0.81702969054045
Random Forest: max_depth 9 n_estimators 13  ROC AUC 0.8745432950926192  Точность 0.82487602787019
```

Рис. 2.18. Результаты выбора наилучших параметров обучения модели

Внимание!!! При выполнении учебного задания у Вас могут быть немного другие результаты.

Распечатаем наилучшие показатели:

```
print("best: max depth", md, "n_estimators", mf,  "Наилучшая метрика
    ROC AUC", acc  )
```

В результате получим best: max depth 9 n_estimators 13. Наилучшая метрика ROC AUC 0.87454.

Внимание!!! При выполнении учебного задания у Вас могут быть немного другие результаты.

Попробуем повторить результат.

```
scc=0
pred_test = np.zeros((ntrain,2))
for  (ttrain, ttest) in kf:
    rfc = RandomForestClassifier(n_estimators = 13, max_depth = 9)
    rfc.fit(xtall[ttrain], ytall[ttrain])
    pred_test[ttest] = rfc.predict_proba(xtall[ttest])
    scc+= rfc.score(xtall[ttest], ytall[ttest])
roc= roc_auc_score(ytall, pred_test[:,1])
scc=scc/NFOLDS
print("roc_auc", roc. "Accuracy", acc)
```

В результате получим следующую точность прогноза:

roc_auc 0.8543 Accuracy 0.8293

Как мы видим полученный результат не совпадает с тем, что получен нами при поиске наилучших параметров. Это связано с тем, что при построении модели присутствует случайный фактор. Если мы начнем повторять выше представленный код, то для каждого повторения мы будем получать разный результат.

24. Чтобы избежать этого недостатка, нам требуется при выборе наилучших параметров сохранять наилучшие модели. Напишем две функции, одна из которых сохраняет модель в файл, а другая загружает модель из файла. Для этого используем библиотеку _pickle.

Модуль _pickle реализует алгоритм сериализации и десериализации объектов Python. Pickling − процесс преобразования объекта Python в поток байтов, а unpickling − обратная операция, в результате которой поток байтов преобразуется обратно в Python-объект. Так как поток байтов легко можно записать в файл, модуль pickle широко применяется для сохранения и загрузки сложных объектов в Python.

Ниже представлен код функций для сохранения и загрузки моделей.

```
import _pickle as cPickle
def SaveModel(name, rfc):
    with open(name, 'wb') as f:
        cPickle.dump(rfc, f)
def LoadModel(name):
    with open(name, 'rb') as f:
        rf = cPickle.load(f)
    return rf
```

Дополните код по поиску наилучших параметров обучения операторами сохранения модели в файл. Так как мы строим одну модель на каждый фолд, нам нужно сохранить пять моделей. Поэтому в рамках двойного цикла перебора параметров создайте массив modl, в который для каждого сочетания параметров будут сохраняться модели с помощью функции append().

```
gr=np.arange(1,45,1)
gr1=np.arange(1,12,1)
acc=0
for i in gr:
    for j in gr1:
        scc=0
        modl=[]
        pred_test = np.zeros((ntrain,2))
        for  (ttrain, ttest) in kf:
            rfc = RandomForestClassifier(n_estimators = i, max_depth =j)
            rfc.fit(xtall[ttrain], ytall[ttrain])
            pred_test[ttest] = rfc.predict_proba(xtall[ttest])
            modl.append(rfc)
```

```
        scc+= rfc.score(xtall[ttest], ytall[ttest])
    ac= roc_auc_score(ytall, pred_test[:,1])
    scc=scc/NFOLDS
    if ac > acc:
        acc=ac
        for kk in np.arange(0,NFOLDS,1):
            s="../Titanic/Data/RF"+str(kk)+".model"
            SaveModel(s, modl[kk])
        md=j
        mf=i
        print("Random Forest: max_depth", j, "n_estimators", i,  " ROC
    AUC", ac, " Точность", scc)
```

Результат представлен на рис. 2.19.

Внимание!!! При выполнении учебного задания у Вас могут быть немного другие результаты.

```
Random Forest: max_depth 1 n_estimators 1  ROC AUC 0.7045478754567048  Точность 0.699140041428661
Random Forest: max_depth 2 n_estimators 1  ROC AUC 0.7155860203027301  Точность 0.6936162199485281
Random Forest: max_depth 3 n_estimators 1  ROC AUC 0.7974333983105912  Точность 0.7665306634862846
Random Forest: max_depth 4 n_estimators 1  ROC AUC 0.7997954814175694  Точность 0.7553825874081979
Random Forest: max_depth 2 n_estimators 2  ROC AUC 0.8170597258172754  Точность 0.7822421693553449
Random Forest: max_depth 3 n_estimators 2  ROC AUC 0.8202446766582515  Точность 0.7788902140480823
Random Forest: max_depth 6 n_estimators 2  ROC AUC 0.8255413883829181  Точность 0.7822358922854812
Random Forest: max_depth 7 n_estimators 2  ROC AUC 0.8342653841647226  Точность 0.7834348126294646
Random Forest: max_depth 4 n_estimators 3  ROC AUC 0.8484671758327209  Точность 0.7968300797187873
Random Forest: max_depth 5 n_estimators 3  ROC AUC 0.8535614993768575  Точность 0.7990961019396146
Random Forest: max_depth 7 n_estimators 3  ROC AUC 0.8636888974104965  Точность 0.8080911430544223
Random Forest: max_depth 6 n_estimators 7  ROC AUC 0.8645437211729994  Точность 0.809189630280585
Random Forest: max_depth 7 n_estimators 9  ROC AUC 0.8668578702372202  Точность 0.8214989642834725
Random Forest: max_depth 9 n_estimators 10  ROC AUC 0.8673771557004228  Точность 0.836124537066097
Random Forest: max_depth 5 n_estimators 14  ROC AUC 0.8674037857241769  Точность 0.823752432364572
Random Forest: max_depth 11 n_estimators 18  ROC AUC 0.8702318942468497  Точность 0.83498211035088
Random Forest: max_depth 6 n_estimators 19  ROC AUC 0.870394337391749  Точность 0.81814700897621
Random Forest: max_depth 8 n_estimators 36  ROC AUC 0.8722025160046442  Точность 0.838334065658150
```

Рис. 2.19. Результаты поиска наилучших моделей

Для прогнозирования используйте функцию загрузки модели.

```
    scc=0
    pred_test = np.zeros((ntrain,2))
    for kk, (ttrain, ttest) in enumerate(kf):
        rfc = RandomForestClassifier(n_estimators = 26, max_depth =6)
        s="../Titanic/Data/RF"+str(kk)+".model"
        print(s)
        rfc=LoadModel(s)
        pred_test[ttest] = rfc.predict_proba(xtall[ttest])
        scc+= rfc.score(xtall[ttest], ytall[ttest])
```

```
roc= roc_auc_score(ytall, pred_test[:,1])
scc=scc/NFOLDS
print("roc_auc", roc, "Accuracy", scc)
```

Результат представлен на рис.2.20.

```
../Titanic/Data/RF0.model
../Titanic/Data/RF1.model
../Titanic/Data/RF2.model
../Titanic/Data/RF3.model
../Titanic/Data/RF4.model
roc_auc 0.8722025160046442 Accuracy 0.8383340656581508
```

Рис. 2.20. Результаты прогнозирования с помощью сохраненных моделей

Отметим, что теперь результаты, полученные при поиске наилучших параметров обучения, и результаты прогнозирования с помощью сохраненных моделей одинаковые.

25. Используйте функцию estim() для оценки полученных результатов.

```
estim(pred_test[:,1], ytall, 0.5)
```

В результате получим:

Precission: 0.8282828282828283 класс 0 спаслись 594 Из них правильно спрогнозировано = 492 , ошибочно= 102
Precission: 0.8080808080808081 класс 1 утонуло 732 Из них правильно спрогнозировано = 240 , ошибочно= 492
Recall: 0.8961748633879781 класс 0 спаслись
Recall: 0.7017543859649122 класс 1 утонуло
Точность распознавания 0.8215488215488216

Используйте методы classification_report() и confusion_matrix() для вычисление различных метрик, оценивающих точность прогнозирования, и вычисления матрицы неточностей.

```
pred=[]
for x in pred_test:
    pred.append(np.argmax(x))
print(classification_report( ytall, pred, target_names=lab, digits=4))
print(confusion_matrix( ytall ,  pred, labels=range(2)))
```

Результаты представлены на рис. 2.21. Сравните полученные результаты с метрикой точности, вычисленной с использованием модели «дерева решений» (см. рис. 2.16).

Как видно из сравнения метод Random Forest позволяет на несколько процентов повысить точность прогнозирования.

26. Для модели, построенной методом случайных деревьев наиболее важные показатели определяются с помощью функции feature_importances_,

которая возвращает imp массив весовых коэффициентов, характеризующих важность показателей.

```
                precision    recall  f1-score   support

      Спаслись     0.8369    0.9162    0.8748       549
       Утонули     0.8414    0.7135    0.7722       342

   avg / total     0.8386    0.8384    0.8354       891

[[503  46]
 [ 98 244]]
```

Рис. 2.21. Точность прогнозирования и матрица неточностей

Создайте структуру rfImp со столбцами 'title' и 'imp'

```
imp =rfc.feature_importances_
rfImp = pd.DataFrame(imp, columns=['imp'])
rfImp['title']=title1
```

Отсортируйте коэффициенты важности в порядке их убывания.

```
rfImp = rfImp.sort_values('imp', ascending = False)
rfImp
```

Результаты показываются на рис. 2.22.

	imp	title
1	0.329472	Пол
5	0.232855	Цена билета
2	0.193283	Возраст
0	0.104827	Социальный статус
3	0.066598	Число родственников 2-го порядка
4	0.041462	Число родственников 1-го порядка
6	0.031503	Порт посадки

Рис. 2.22. Коэффициенты важности (значимости) показателей

Самостоятельное задание 1

По 50 водителям были зарегистрированы:

X1 – возраст,

X2 – состояние зрения (имеются проблемы со зрением − значение переменной равно единице, нет проблем − нулю),

X3 – уровень подготовки водителя (если прошел курсы для водителей, X3 = 1; если нет, X3 = 0),

Y – наличие дорожно-транспортных происшествий (ДТП) в последний год (0 – нет, 1 – да).

Все данные хранятся в файле DTP.csv. Файл подготовлен в российском формате (разделители точка с запятой).

1) Загрузите данные в структуру DataFrame.

2) Постройте модель для прогнозирования вероятности ДТП. Для построения модели используйте методы машинного обучения: дерево решений и Random forest.

2.7. Практическое занятие №4. Построение моделей классификации и анализ их точности

Цель занятия:

Изучить основные элементы языка Python.

Закрепить навыки и умения использования методов классификации.

Изучить метрики оценки качества получаемых решений.

Учебное задание

В своей работе Р. Фишер использовал данные, собранные американским ботаником Э. Андерсоном, который измерил следующие характеристики цветков каждого из 150 образцов:

- длина чашелистика (англ. sepal length),
- ширина чашелистика (англ. sepal width),
- длина лепестка (англ. petal length),
- ширина лепестка (англ. petal width).

Соцветия ириса имеют форму веера и содержат один или более симметричных шестидольных цветков. Растут они на коротком стебельке. Три чашелистика направлены вниз. Они расширяются из узкого основания в обширное окончание, украшенное прожилками, линиями или точками. Три лепестка, которые иногда могут быть редуцированными, находятся в вертикальной позиции и частично скрыты основанием чашелистика. У более мелких ирисов вверх направлены все шесть доль. Чашелистики и лепестки отличаются друг от друга. Они объединены у основания в цветочный цилиндр, который лежит над завязью. Виды ириса представлены на рис. 2.23.

Рассматривается задача кластеризация цветов, которая получила название задача ирисов Фишера. Эта задача стала популярной после работы Роберта Фишера. Набор данных включает три класса по 50 образцов в каждом. Каждый класс соответствует виду ириса:

- Iris Setosa (класс 1),
- Iris Versicolour (класс 2),
- Iris Virginica (класс 3).

Рис. 2.23. Ирисы Фишера (слева направо): Setosa, Versicolour и Virginica

Технология выполнения учебного задания

1. На первом шаге загрузим все необходимые нам библиотеки. Наряду с известными уже нам библиотекам pandas, numpy, matplotlib загрузите из библиотеки sklearn модуль datasets. Модуль datasets содержит достаточно много различных наборов данных.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
```

2. Читаем данные из datasets, которые сохраняем в объекте iris. Этот объект имеет несколько атрибутов data, target, target_names. Эти атрибыты содержат массиыв данных, которые мы сохраняем в массивах X, y и target_names.

```
iris = datasets.load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names
```

Распечатаем эти массивы

```
print(X)
```

Результат показан на рис. 2.24.

Как видно, каждая строка массива X содержит 4 числа, которые представляют длину чашелистика, ширину чашелистика, длину лепестка, ширину лепестка.

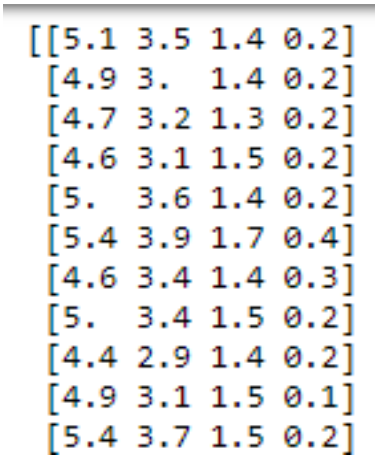

Рис. 2.24. Массив X

Распечатайте массив y, результат показывается на рис. 2.25.

```
print(y)
```

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

Рис. 2.25. Массив значений целевой переменной y

Распечатайте список имен классов (рис. 2.26).

```
print(target_names)
```

```
['setosa' 'versicolor' 'virginica']
```

Рис. 2.26. Название классов целевой переменной

Распечатайте список имен показателей (рис. 2.27).

```
print(feature_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

Рис. 2.27. Название показателей

Создайте список имен показателей на русском языке.

```
title=["Длина чашелистика (см)", "Ширина чашелистика (см)", "Длина
    лепестка (см)", "Ширина лепестка (см)"]
```

3. Определите размеры массива X. Для этого используйте атрибут shape.

```
m,n = X.shape
print("Число показателей", m, "число строк (объектов)", m)
```

4. Разбейте набор данных на 6 фолдов и задайте значение переменной nclass = 3. Для этого используйте код:

```
from sklearn.cross_validation import KFold
nclass=3
```

```
SEED = 23435
NFOLDS = 6
kf = KFold(m, n_folds= NFOLDS, random_state=SEED, shuffle =True)
```

5. Загрузите библиотеки машинного обучения и метрики для оценки точности. Для задачи с большим числом классов (больше 2) функция для вычисления метрики ROC AUC не подходит, так как она работает только с двумя классами. Для нашей задачи с мультиклассами подходят функции accuracy_score и precision_score.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score
from sklearn import tree
```

6. Проведите классификацию ирисов используя «дерево решений». Сначала выполним поиск наилучших показателей обучения max_depth и max_features. При переборе наилучшую модель будем сохранять с помощью функции SaveModel() и загружать функцией LoadModel().

```
import _pickle as cPickle
def SaveModel(name, rfc):
    with open(name, 'wb') as f:
        cPickle.dump(rfc, f)
# in your prediction file
def LoadModel(name):
    with open(name, 'rb') as f:
        rf = cPickle.load(f)
    return rf
```

Значения параметра max_features будем перебирать в диапазоне от 1 до 4, т.к. 4 – это число показателей. А наилучшие значения max_depth будем искать в диапазоне от 1 до 11.

```
gr=np.arange(1,5,1)
gr1=np.arange(1,12,1)
acc=0
for i in gr:
    for j in gr1:
        modl=[]
        pred_test = np.zeros((m,nclass))
        for  (ttrain, ttest) in kf:
            dtc = tree.DecisionTreeClassifier(max_depth = j,max_features = i)
            dtc=dtc.fit(X[ttrain], y[ttrain])
            pred_test[ttest] = dtc.predict_proba(X[ttest])
            modl.append(dtc)
        pred=[]
        for x in pred_test:
            pred.append(np.argmax(x))
```

```
            ac= accuracy_score(y, pred)
            if ac > acc:
                acc=ac
                for kk in np.arange(0,NFOLDS,1):
                    s="../Titanic/Data/DT-iris"+str(kk)+".model"
                    SaveModel(s, modl[kk])
                md=j
                mf=i
                print("DesicionTree: max_depth", j, "max_features", i,  "
            Точность", ac)
```

Результат представлен на рис. 2.28.

```
DesicionTree: max_depth 1 max_features 1  Точность 0.5733333333333334
DesicionTree: max_depth 2 max_features 1  Точность 0.7666666666666667
DesicionTree: max_depth 3 max_features 1  Точность 0.9266666666666666
DesicionTree: max_depth 3 max_features 2  Точность 0.9733333333333334
```

Рис. 2.28. Результаты поиска наилучших моделей

Если повторять исполнение кода, то можно получать другие результаты.

Создадим внешний цикл повторения, чтобы повысить шансы нахождения наилучшей модели.

```
gr=np.arange(1,5,1)
gr1=np.arange(1,12,1)
acc=0
for rep in np.arange(100):
    for i in gr:
        for j in gr1:
            modl=[]
            pred_test = np.zeros((m,nclass))
            for  (ttrain, ttest) in kf:
                dtc = tree.DecisionTreeClassifier(max_depth = j,max_features
            = i)
                dtc=dtc.fit(X[ttrain], y[ttrain])
                pred_test[ttest] = dtc.predict_proba(X[ttest])
                modl.append(dtc)
            pred=[]
            for x in pred_test:
                pred.append(np.argmax(x))
            ac= accuracy_score(y, pred)
            if ac > acc:
                acc=ac
                for kk in np.arange(0,NFOLDS,1):
                    s="../Titanic/Data/DT-iris"+str(kk)+".model"
                    SaveModel(s, modl[kk])
```

```
            md=j
            mf=i
            print("DesicionTree: max_depth", j, "max_features", i,  "
        Точность", ac)
```

Результаты представлены на рис. 2.29.

```
DesicionTree: max_depth 1 max_features 1  Точность 0.5933333333333334
DesicionTree: max_depth 2 max_features 1  Точность 0.7533333333333333
DesicionTree: max_depth 3 max_features 1  Точность 0.9
DesicionTree: max_depth 5 max_features 1  Точность 0.9266666666666666
DesicionTree: max_depth 8 max_features 2  Точность 0.9333333333333333
DesicionTree: max_depth 2 max_features 3  Точность 0.9666666666666667
DesicionTree: max_depth 3 max_features 2  Точность 0.98
```

Рис. 2.29. Результаты поиска

7. Используйте полученные модели, чтобы вычислить прогнозные значения для исходного набора. Результаты представлены на рис. 2.30.

```
pred_test = np.zeros((m,nclass))
for  (ttrain, ttest) in kf:
    dtc = tree.DecisionTreeClassifier(max_depth = 3,max_features = 2)
    s="../Titanic/Data/DT-iris"+str(kk)+".model"
    print(s)
    dtc=LoadModel(s)
    pred_test[ttest] = dtc.predict_proba(X[ttest])
pred=[]
for x in pred_test:
    pred.append(np.argmax(x))
ac= accuracy_score(y, pred)
print("Точность", ac)
```

```
../Titanic/Data/DT-iris0.model
../Titanic/Data/DT-iris1.model
../Titanic/Data/DT-iris2.model
../Titanic/Data/DT-iris3.model
../Titanic/Data/DT-iris4.model
../Titanic/Data/DT-iris5.model
Точность 0.98
```

Рис. 2.30. Результаты обучения моделей

8. Вычислите метрики точности и матрицу неточностей. Результат показан на рис. 2.31.

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
print(classification_report( y, pred, target_names=target_names, digits=4))
print(confusion_matrix( y,  pred, labels=range(3)))
```

```
             precision    recall  f1-score   support

     setosa     1.0000    1.0000    1.0000        50
 versicolor     0.9608    0.9800    0.9703        50
  virginica     0.9796    0.9600    0.9697        50

avg / total     0.9801    0.9800    0.9800       150

[[50  0  0]
 [ 0 49  1]
 [ 0  2 48]]
```

Рис. 2.31. Метрики точности и матрица неточностей

9. Вычислите коэффициенты важности модели. Результат показан на рис. 2.32.

```
imp= dtc.feature_importances_
rfImp = pd.DataFrame(imp, columns=['imp'])
rfImp['title']=title
rfImp = rfImp.sort_values('imp', ascending = False)
rfImp.head(7)
```

	imp	title
2	0.593927	Длина лепестка (см)
3	0.406073	Ширина лепестка (см)
0	0.000000	Длина чашелистика (см)
1	0.000000	Ширина чашелистика (см)

Рис. 2.32. Коэффициенты важности

10. Нарисуйте одну из сохраненных моделей, например, последнюю модель. Результат представлен на рис. 2.33.

```
import pydotplus
dot_data = tree.export_graphviz(dtc, feature_names=title,class_names=
      target_names , out_file=None,  filled=True,   rounded=True)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_png("../Titanic/Data/DC-iris.jpg")
import matplotlib.image as mpimg
img=mpimg.imread("../Titanic/Data/DC-iris.jpg")
fig = plt.figure(figsize=(100,100))
imgplot = plt.imshow(img)
```

Как видно из рисунка при построении модели используется только два показателя.

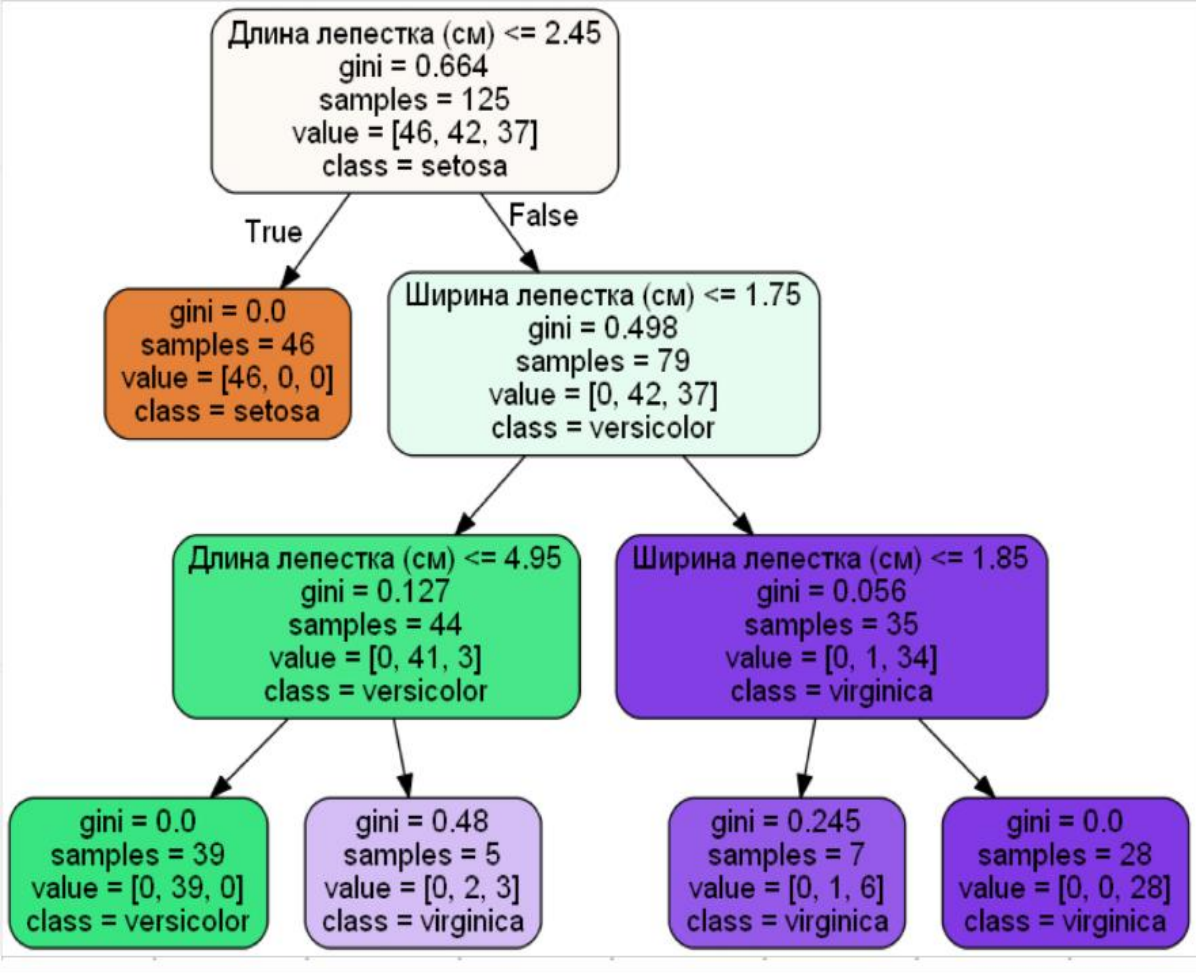

Рис. 2.33. Дерево решений

RandomForestClassifier

11. Для поиска наилучших моделей, построенных методом Random Forest, используем перебор параметров. Также применяем такой перебор несколько раз.

```
gr=np.arange(2,15,1)
gr1=np.arange(1,12,1)
acc=0
for rep in np.arange(10):
    for i in gr:
        for j in gr1:
            modl=[]
            pred_test = np.zeros((m,nclass))
            for  (ttrain, ttest) in kf:
                rfc = RandomForestClassifier(n_estimators = i, max_depth =j)
                rfc.fit(X[ttrain], y[ttrain])
                pred_test[ttest] = rfc.predict_proba(X[ttest])
                modl.append(rfc)
            pred=[]
```

```
            for x in pred_test:
                pred.append(np.argmax(x))
            ac= accuracy_score(y, pred)
            if ac > acc:
                acc=ac
                for kk in np.arange(0,NFOLDS,1):
                    s="../Titanic/Data/RF-iris"+str(kk)+".model"
                    SaveModel(s, modl[kk])
                md=j
                mf=i
                print("Random Forest: max_depth", j, "n_estimators", i,   "
        Точность", ac)
```

Результаты представлены на рис. 2.34.

```
Random Forest: max_depth 1 n_estimators 2  Точность 0.7133333333333334
Random Forest: max_depth 2 n_estimators 2  Точность 0.9333333333333333
Random Forest: max_depth 5 n_estimators 2  Точность 0.94
Random Forest: max_depth 10 n_estimators 2  Точность 0.96
Random Forest: max_depth 2 n_estimators 2  Точность 0.9733333333333334
```

Рис. 2.34. Результаты поиска наилучшей модели

12. Проверьте эффективность сохраненных моделей. Результат представляется на рис. 2.35.

```
scc=0
pred_test = np.zeros((m,nclass))
for kk, (ttrain, ttest) in enumerate(kf):
    rfc = RandomForestClassifier()
    s="../Titanic/Data/RF-iris"+str(kk)+".model"
    print(s)
    rfc=LoadModel(s)
    pred_test[ttest] = rfc.predict_proba(X[ttest])
pred=[]
for x in pred_test:
    pred.append(np.argmax(x))
ac= accuracy_score(y, pred)
print( "Accuracy", ac)
```

```
../Titanic/Data/RF-iris0.model
../Titanic/Data/RF-iris1.model
../Titanic/Data/RF-iris2.model
../Titanic/Data/RF-iris3.model
../Titanic/Data/RF-iris4.model
../Titanic/Data/RF-iris5.model
Accuracy 0.9733333333333333
```

Рис. 2.35 Использование моделей

13. Для оценки точности используйте функции classification_report() и confusion_matrix(). Результат показан на рис.2.36.

```
print(classification_report( y, pred, target_names=target_names, digits=4))
print(confusion_matrix( y ,  pred, labels=range(nclass)))
```

```
             precision    recall  f1-score   support

     setosa     1.0000    1.0000    1.0000        50
 versicolor     0.9600    0.9600    0.9600        50
  virginica     0.9600    0.9600    0.9600        50

avg / total     0.9733    0.9733    0.9733       150

[[50  0  0]
 [ 0 48  2]
 [ 0  2 48]]
```

Рис. 2.36. Оценка точности полученной модели и матриц неточности

Самостоятельное задание 1

Рассматривается задача анализа кредитоспособности заемщика. Для описания финансового и социального состояния заемщика используются следующие показатели:

ZAEM – брался ли кредит ранее (да, нет);

DOXOD – среднемесячный доход семьи заемщика, тыс руб.;

TIME – период погашения кредита;

VOL – размер кредита;

FAMIL – состав семьи заемщика, чел.;

OLD – возраст заемщика, лет;

RETURN – вероятность погашения кредита: 3 – высокая, 2 – средняя, 1–низкая.

Все данные хранятся в файле credit.csv. Файл подготовлен в российском формате (разделители точка с запятой).

1) Загрузите данные в структуру DataFrame.

2) Постройте модель для оценки вероятности погашения кредита методом «Дерево решений» и Random Forest.

Глава 3. ПАРСИНГ WEB САЙТОВ

3.1. Парсинг WEB сайтов с использованием библиотеки Beautiful Soup

Если открыть любую веб-страницу, то она будет содержать в себе типичные элементы, которые называются тегами. Рассмотрим HTML теги, определяющие структуру страницы:

`<html> </html>` – обязательные, определяют HTML документ;

`<head> </head>` – определяют секцию со служебной информацией, содержат инструкции для поисковиков, для браузеров, скрипты;

`<title> </title>` – определяют основной заголовок web-страницы;

`<body> </body>` – обязательные, определяют видимую часть документа;

`<h3> </h3>` – определяют заголовок 3-го уровня;

`<p> </p>` – определяют параграф.

Начинается документ с элемента `<!DOCTYPE>`, котрый предназначен для указания типа текущего документа – DTD (document type definition, описание типа документа). Это необходимо, чтобы браузер понимал, как следует интерпретировать текущую веб-страницу, ведь HTML существует в нескольких версиях, кроме того, имеется XHTML (EXtensible HyperText Markup Language, расширенный язык разметки гипертекста), похожий на HTML, но различающийся с ним по синтаксису. Чтобы браузер «не путался» и понимал, согласно какому стандарту отображать веб-страницу и необходимо в первой строке кода задавать `<!DOCTYPE>`. Существует несколько видов `<!DOCTYPE>`, они различаются в зависимости от версии HTML, на которую ориентированы. В табл. 3.1. приведены основные типы документов с их описанием.

Таблица 3.1.

DOCTYPE	Описание
HTML 4.01	
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">	Строгий синтаксис HTML
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">	Переходный синтаксис HTML
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">	В HTML-документе применяются фреймы

Окончание таблицы 3.1

HTML 5	
<!DOCTYPE html>	В этой версии HTML только один доктайп
XHTML 1.0	
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">	Строгий синтаксис XHTML
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">	Переходный синтаксис XHTML
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">	Документ написан на XHTML и содержит фреймы
XHTML 1.1	
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">	Разработчики XHTML 1.1 предполагают, что он постепенно вытеснит HTML. Как видите, никакого деления на виды это определение не имеет, поскольку синтаксис один и подчиняется четким правилам

Разница между строгим и переходным описанием документа состоит в различном подходе к написанию кода документа. Строгий HTML требует жесткого соблюдения спецификации HTML и не прощает ошибок. Переходный HTML более «спокойно» относится к некоторым огрехам кода, поэтому этот тип в определенных случаях использовать предпочтительнее.

Например, в строгом HTML и XHTML непременно требуется наличие тега <title>, а в переходном HTML его можно опустить и не указывать. При этом помним, что браузер в любом случае покажет документ, независимо от того, соответствует он синтаксису или нет. Подобная проверка осуществляется при помощи валидатора и предназначена в первую очередь для разработчиков, чтобы отслеживать ошибки в документе.

Тег <html> определяет начало HTML-файла, внутри него хранится заголовок (<head>) и тело документа (<body>).

Заголовок документа, как еще называют блок <head>, может содержать текст и теги, но содержимое этого раздела не показывается напрямую на странице, за исключением контейнера <title>.

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

Тег <meta> является универсальным и добавляет целый класс возможностей, в частности, с помощью метатегов, как обобщенно называют этот тег, можно изменять кодировку страницы, добавлять ключевые слова, описание документа и многое другое. Чтобы браузер понимал, что имеет дело с кодировкой UTF-8 (Unicode transformation format, формат преобразования Юникод) и добавляется данная строка.

<title>Пример веб-страницы</title>

Тег <title> определяет заголовок веб-страницы, это один из важных элементов предназначенный для решения множества задач.

Обязательно следует добавлять закрывающий тег </head>, чтобы показать, что блок заголовка документа завершен.

Тело документа <body> предназначено для размещения тегов и содержательной части веб-страницы.

<h1>Заголовок</h1>

HTML предлагает шесть текстовых заголовков разного уровня, которые показывают относительную важность секции, расположенной после заголовка. Так, тег <h1> представляет собой наиболее важный заголовок первого уровня, а тег <h6> служит для обозначения заголовка шестого уровня и является наименее значительным. По умолчанию, заголовок первого уровня отображается самым крупным шрифтом жирного начертания, заголовки последующего уровня по размеру меньше. Теги <h1>...<h6> относятся к блочным элементам, они всегда начинаются с новой строки, а после них другие элементы отображаются на следующей строке. Кроме того, перед заголовком и после него добавляется пустое пространство.

Некоторый текст можно спрятать от показа в браузере, сделав его комментарием. Хотя такой текст пользователь не увидит, он все равно будет передаваться в документе, так что, посмотрев исходный код, можно обнаружить скрытые заметки.

Комментарии нужны для внесения в код своих записей, не влияющих на вид страницы. Начинаются они тегом <! и заканчиваются тегом >. Все, что находится между этими тегами, отображаться на веб-странице не будет.

Первый абзац.

Тег определяет абзац (параграф) текста. Если закрывающего тега нет, считается, что конец абзаца совпадает с началом следующего блочного элемента.

Второй абзац.

Тег является блочным элементом, поэтому текст всегда начинается с новой строки, абзацы идущие друг за другом разделяются между собой отбивкой (так называется пустое пространство между ними).

Следует добавить закрывающий тег </body>, чтобы показать, что тело документа завершено.

Последним элементом в коде всегда идет закрывающий тег </html>.

Библиотека requests для работы с веб-страницами

Стандартная библиотека Python имеет ряд готовых модулей по работе с HTTP: urllib и httplib. Библиотека requests ничто иное как обёртка над urllib3, а последняя является надстройкой над стандартными средствами Python.

Библиотека requests является Python-библиотекой для выполнения запросов к серверу и обработки ответов. Пользуясь данной библиотекой, мы получаем содержимое страницы в виде html для дальнейшего парсинга.

В библиотеке requests имеется:

- множество методов http аутентификации;
- сессии с куками;
- полноценная поддержка SSL;
- различные методы типа .json(), которые вернут данные в нужном формате;
- проксирование;
- грамотная и логичная работа с исключениями.

Если библиотека requests не установлена на вашем компьютере, то инсталлируйте ее в командной строке pip install requests.

Библиотека Beautiful Soup

Beautifulsoup4 – это библиотека для парсинга html и xml документов. Позволяет получить доступ напрямую к содержимому любых тегов в html. Подробная документация:

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Если библиотека Beautiful Soup не установлена, то нужно в командной строке запустить команду:

```
pip install BeautifulSoup4
```

Библиотека Beautiful Soup создает дерево синтаксического разбора из проанализированных HTML и XML-документов (включая документы с открытыми тегами, tag soup и неправильной разметкой). Эта библиотека сделает текст веб-страницы, извлеченный с помощью Requests, более удобочитаемым. Импортируйте Beautiful Soup.

3.2. Практическое занятие №5. Парсинг WEB сайтов с использованием библиотеки Beautiful Soup

Учебное задание

Выполнить парсинг учебных сайтов с помощью функции библиотек requests и Beautiful Soup.

Технология выполнения учебного заданий

Загрузка web страниц: Библиотека requests является де-факто стандартом для выполнения HTTP-запросов в Python. Загрузите библиотеку requests:

```
import requests
```

Присвойте URL-адрес тестовой страницы (в данном случае это https://en.wikipedia.org/wiki/List_of_S%26P_500_companies) переменной url. Для добавления HTTP заголовка в запрос, сформируем переменную hdr типа словаря.

```
url="https://en.wikipedia.org/wiki/List_of_S%26P_500_companies"
hdr = {"User-Agent": "Google"}
```

Результат запроса этой страницы можно получить с помощью метода request.get(). В качестве аргументов запроса используются URL-адрес страницы (переменной url), заголовок (переменная hdr).

```
reqv = requests.get(url, headers = hdr)
```

Переменная reqv представляет объект типа Response и используется для проверки результатов запроса и извлечения из нее нужной информации. Объект reqv имеет свойство status_code, который возвращается в квадратных скобках (в данном случае это 200). Этот атрибут можно вызвать явно:

```
reqv.status_code
```

Возвращаемый код 200 сообщает, что страница загружена успешно. Коды, начинающиеся с номера 2, обычно указывают на успешное выполнение операции, а коды, начинающиеся с 4 или 5, сообщают об ошибке. Вы можете узнать больше о кодах состояния HTTP по этой ссылке:

```
if reqv.status_code == 200:
    print('Success!')
elif reqv.status_code == 404:
    print('Not Found.')
```

Если при выполнении программы возвращается код 200, то программа напечатает Success!. Если программа возвращается код 404, то программа напечатает Not Found.

Можно упростить процесс проверки. Если вы используете экземпляр Response в условном операторе if, то он оценивается как True, если код состояния был между 200 и 400, и False в противном случае.

Таким образом, вы можете упростить последний скрипт, переписав оператор if:

```
if reqv:
    print('Success!')
else:
    print('An error has occurred.')
```

Нужно понимать, что этот способ не проверяет ситуацию, когда код состояния равен 200. Причина этого заключается в том, что другими значениями кода состояния в диапазоне от 200 до 400 являются такие как 204 НЕТ КОНТЕНТА и 304 НЕ ИЗМЕНЕНО, которые также считаются успешными, так как они дают какой-то работоспособный ответ.

Например, 204 сообщает вам, что ответ был успешным, но в теле сообщения нет содержимого для возврата.

Поэтому убедитесь, что вы используете правильный способ проверки кода состояния, только если вы хотите узнать, был ли запрос в целом успешным, а затем, при необходимости, обработать ответ соответствующим образом на основе кода состояния.

Если вам требуется проверять код состояния ответа не в операторе if, а использовать исключение, если запрос был неудачным, то можно использовать метод raise_for_status ().

```
try:
    # If the response was successful, no Exception will be raised
    reqv.raise_for_status()
except HTTPError as http_err:
    print(f'HTTP error occurred: {http_err}')  # Python 3.6
except Exception as err:
    print(f'Other error occurred: {err}')  # Python 3.6
else:
    print('Success!')
```

Ответ на запрос часто содержит некоторую ценную информацию в теле сообщения. Чтобы работать с веб-данными, нужно получить доступ к текстовому содержимому веб-файлов. Прочитать содержимое ответа сервера можно с помощью reqv.text (или reqv.content, чтобы получить значение в байтах).

Чтобы увидеть содержание запроса в байтах используйте атрибут content:

```
reqv.content
```

Фрагмент результата представлен ниже:

```
b'<!DOCTYPE html>\n<html class="client-nojs" lang="en"
dir="ltr">\n<head>\n<meta charset="UTF-8"/>\n<title>List of
S&amp;P 500 companies - Wikipedia</title>\n<script>document.docu-
mentElement.className=document.documentElement.className.re-
place(/(^|\\s)client-nojs(\\s|$)/,"$1client-js$2");RLCONF={"wgCanonical-
Namespace":"","wgCanonicalSpecialPageName":!1,"wgNamespace-
Number":0,"wgPageName":"List_of_S\\u0026P_500_compa-
nies","wgTitle":"List of S\\u0026P 500 companies","wgCurRevi-
sionId":906900594,"wgRevisionId":906900594,"wgArti-
cleId":2676045,"wgIsArticle":!0,"wgIsRedirect":!1,"wgAc-
tion":"view","wgUserName":null,"wgUserGroups":["*"],"wgCatego-
ries":["CS1 maint: Archived copy as title","Webarchive template wayback
```

links","Articles with short description","S\\u0026P Dow Jones Indices","Lists of companies","Standard \\u0026 Poor\'s"],"wgBreakFrames":!1,"wgPageContentLanguage":"e

Страница начинается с <!DOCTYPE html> – это значит, что страница написана на HTML 5.

При этом вы получаете доступ к необработанным байтам полезной нагрузки ответа на запрос, хотя часто требуется полученные данные преобразовать в строку с использованием кодировки символов, такой как UTF-8. Для этого используйте атрибут text:

```
reqv.text
```

Полный текст страницы был отображен со всеми тегами HTML. Однако его трудно прочитать, поскольку между ними не так много пробелов.

'<!DOCTYPE html>\n<html class="client-nojs" lang="en" dir="ltr">\n<head>\n<meta charset="UTF-8"/>\n<title>List of S&P 500 companies - Wikipedia</title>\n<script>document.documentElement.className=document.documentElement.className.replace(/(^|\\s)client-nojs(\\s|$)/,"$1client-js$2");RLCONF={"wgCanonicalNamespace":"","wgCanonicalSpecialPageName":!1,"wgNamespaceNumber":0,"wgPageName":"List_of_S\\u0026P_500_companies","wgTitle":"List of S\\u0026P 500 companies","wgCurRevisionId":906900594,"wgRevisionId":906900594,"wgArticleId":2676045,"wgIsArticle":!0,"wgIsRedirect":!1,"wgAction":"view","wgUserName":null,"wgUserGroups":["*"],"wgCategories"

Заголовки ответа могут дать вам полезную информацию, такую как тип содержимого полезной информации ответа, ограничение по времени, в течение которого необходимо кэшировать ответ. Для просмотра этих заголовков откройте .headers:

```
reqv.headers
```

Результат представлен ниже:

{'Date': 'Mon, 29 Jul 2019 09:01:53 GMT', 'Content-Type': 'text/html; charset=UTF-8', 'Content-Length': '79052', 'Connection': 'keep-alive', 'Server': 'mw1246.eqiad.wmnet', 'X-Content-Type-Options': 'nosniff', 'P3P': 'CP="This is not a P3P policy! See https://en.wikipedia.org/wiki/Special:CentralAutoLogin/P3P for more info."', 'X-Powered-By': 'HHVM/3.18.6-dev', 'Content-language': 'en', 'Last-Modified': 'Thu, 25 Jul 2019 21:59:08 GMT', 'Backend-Timing': 'D=128720 t=1564091959728657', 'Content-Encoding': 'gzip', 'Vary': 'Accept-Encoding,Cookie,Authorization,X-Seven', 'X-Varnish': '670327713 494070418, 167540380 76192322, 1061636976 710992254', 'Age': '128404', 'X-Cache': 'cp1087 hit/6, cp3041 hit/2,

```
cp3032 hit/90', 'X-Cache-Status': 'hit-front', 'Server-Timing':
'cache;desc="hit-front"', 'Strict-Transport-Security': 'max-age=106384710;
includeSubDomains; preload', 'Set-Cookie': 'WMF-Last-Access=29-Jul-
2019;Path=/;HttpOnly;secure;Expires=Fri, 30 Aug 2019 00:00:00 GMT,
WMF-Last-Access-Global=29-Jul-2019;Path=/;Domain=.wikipe-
dia.org;HttpOnly;secure;Expires=Fri, 30 Aug 2019 00:00:00 GMT,
GeoIP=RU:CHE:Chelyabinsk:55.16:61.43:v4; Path=/; secure; Do-
main=.wikipedia.org', 'X-Analytics': 'ns=0;page_id=2676045;https=1;no-
cookies=1', 'X-Client-IP': '77.222.116.58', 'Cache-Control': 'private, s-max-
age=0, max-age=0, must-revalidate', 'Accept-Ranges': 'bytes'}
```

Атрибут headers возвращает словарный объект, позволяющий получить доступ к значениям заголовка по ключу. Например, чтобы увидеть тип содержимого полезной нагрузки ответа, вы можете получить доступ по ключу «Content-Type»:

```
reqv.headers['Content-Type']
```

В результате получим:

```
'text/html; charset=UTF-8'
```

Теперь вы узнали основы о Response. Вы видели его наиболее полезные атрибуты и методы в действии.

Парсинг веб-данных с помощью Beautiful Soup

Beautifulsoup4 – это библиотека для парсинга html и xml документов. Позволяет получить доступ напрямую к содержимому любых тегов в html.

Библиотека Beautiful Soup создает дерево синтаксического разбора из проанализированных HTML и XML-документов. Эта библиотека сделает текст веб-страницы, извлеченный с помощью Requests, более удобочитаемым.

Импортируйте Beautiful Soup с помощью команды:

```
from bs4 import BeautifulSoup
```

Затем нужно запустить обработку документа reqv.text, чтобы получить объект BeautifulSoup, то есть дерево синтаксического разбора этой страницы, полученной с помощью встроенного html.parser через HTML. Построенный объект представляет документ mockturtle.html как вложенную структуру данных, которая присваивается переменной soup.

```
soup = BeautifulSoup(reqv.text, 'html.parser')
```

Навигация по документу

Чтобы отобразить содержимое страницы в терминале, используйте метод prettify(), который превращает дерево разбора Beautiful Soup в красиво отформатированную строку Unicode.

```
print(soup.prettify())
```

Результат частично представлен ниже:

```
<!DOCTYPE html>
<html class="client-nojs" dir="ltr" lang="en">
 <head>
  <meta charset="utf-8"/>
  <title>
   List of S&amp;amp;P 500 companies - Wikipedia
  </title>
  <script>
   document.documentElement.className=document.documentEle-
ment.className.replace(/(^|\s)client-nojs(\s|$)/,"$1client-
js$2");RLCONF={"wgCanonicalNamespace":"","wgCanonicalSpecialPage-
Name":!1,"wgNamespaceNumber":0,"wgPage-
Name":"List_of_S\u0026P_500_companies","wgTitle":"List of S\u0026P 500
companies","wgCurRevisionId":906900594,"wgRevi-
sionId":906900594,"wgArticleId":2676045,"wgIsArticle":!0,"wgIsRedi-
rect":!1,"wgAction":"view","wgUserName":null,"wgUser-
Groups":["*"],"wgCategories":["CS1 maint: Archived copy as title","Webar-
chive template wayback links","Articles with short description","S\u0026P
Dow Jones Indices","Lists of companies","Standard \u0026
Poor's"],"wgBreakFrames":!1,"wgPageContentLanguage":"en","wgPageCon-
tentModel":"wikitext","wgSeparatorTransformTable":["",""],"wgDigitTrans-
formTable":["",""],"wgDefaultDateFormat":"dmy","wgMonth-
Names":["","January","February","March","April","May","June","July","Au-
gust","September","October","November","December"],"wgMonth-
NamesShort":
["","Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"],
"wgRelevantPageName":"List_of_S\u0026P_500_companies","wgRele-
vantArticleId":2676045,"wgRequestId":"XTomNwpAIC4AADvYOKE-
AAACQ","wgCSPNonce":!1,"wgIsProbablyEditable":!0,"wgRelevantPageIs-
ProbablyEditable":!0,"wgRestrictionEdit":[],"wgRestrictionMove":[],"wgMe-
diaViewerOnClick":!0,"wgMediaViewerEnabledByDe-
fault":!0,"wgPopupsReferencePreviews":!1,"wgPopupsConflictsWith-
NavPopupGadget":!1,"wgVisualEditor":{"pageLanguageCode":"en","page-
LanguageDir":"ltr","pageVariantFallbacks":"en"},
```

Объект парсера (экземпляр класса BeautifulSoup или BeautifulStoneSoup) обладает большой глубиной вложенности связанных структур данных, соответствующих структуре документа XML или HTML.

Объект парсера состоит из объектов двух других типов:

1) объект Tag, который соответствуют тегам, к примеру, тегу <TITLE> и тегу <B>;

2) объект NavigableString, соответствующие таким строкам как Page title или This is paragraph.

Класс NavigableString имеет несколько подклассов (CData, Comment, Declaration и ProcessingInstruction), которые соответствуют специальным конструкциям в XML. Они работают также как NavigableString, и, за исключением того, что когда приходит время выводить их на экран, они содержат некоторые дополнительные данные. Вот документ с включенным в него комментарием:

```
doc = ['<html><head><title>Page title</title></head>',
      '<body><p id="firstpara" align="center">This is paragraph
       <b>one</b>.',
      '<p id="secondpara" align="blah">This is paragraph <b>two</b>.',
      '</html>']
soup2 = BeautifulSoup(''.join(doc))
print(soup2.prettify())
```

Результат:

```
</html>
 <head>
  <title>
   Page title
  </title>
 </head>
<body>
  <p align="center" id="firstpara">
  This is paragraph
  <b>
   one
  </b>
  .
  </p>
  <p align="blah" id="secondpara">
  This is paragraph
  <b>
   two
  </b>
  .
 </p>
</body>
</html>
```

Объекты Tag и NavigableString имеют множество полезных элементов. Теги SGML имеют атрибуты. Например, в приведенном выше примере HTML документа каждый тег <p> имеет атрибуты id и align. К атрибутам тегов можно обращаться таким же образом, как если бы объект Tag был словарем:

```
firstPTag, secondPTag = soup.findAll('p')
print(firstPTag['id'])
print(secondPTag['id'])
```

Результат:

```
firstpara
secondpara
```

Объекты NavigableString не имеют атрибутов, только объекты Tag имеют их.

Все объекты Tag содержат элементы, перечисленные ниже (тем не менее, фактическое значение элемента может равняться None). Объекты NavigableString имеют все из них за исключением contents и string.

В примере выше, родителем объекта <HEAD> Tag является объект <HTML> Tag. Родителем объекта <HTML> Tag является сам объект парсера BeautifulSoup. Родитель объекта парсера равен None. Передвигаясь по объектам parent, можно перемещаться по дереву синтаксического разбора:

```
print(soup2.head.parent.name)
print(soup2.head.parent.parent.__class__.__name__)
print(soup2.parent)
```

Результат:

```
html
BeautifulSoup
None
```

С помощью parent вы перемещаетесь вверх по дереву синтаксического разбора. С помощью contents вы перемещаетесь вниз по дереву синтаксического разбора. contents является упорядоченным списком объектов Tag и NavigableString, содержащихся в элементе страницы (page element). Только объект парсера самого высокого уровня и объекты Tag содержат contents. Объекты NavigableString являются простыми строками и не могут содержать подэлементов, поэтому они не содержат contents.

```
pTag = soup2.p
print(pTag.contents)
```

Результат:

```
['This is paragraph ', <b>one</b>, '.']
```

Элемент contents объекта pTag является списком, содержащим объект NavigableString ("This is paragraph"), объекта <B> Tag ("one") и еще одного объекта NavigableString (".").

Таким образом, нулевой и второй элемент списка contents объекта pTag имеют тип NavigableString и поэтому не имеют атрибута contents.

Второй элемент списка contents объекта pTag (объекта <B> Tag:) представляет список, состоящий из одного объекта NavigableString ("one"). Поэтому он имеет атрибут contents:

```
print(pTag.contents[1].contents)
print(pTag.contents[0].contents)
```

Результат:

```
['one']
<ipython-input-102-15e8a0f0ab15> in <module>
    1 print(pTag.contents[1].contents)
----> 2 print(pTag.contents[0].contents)

C:\ProgramData\Anaconda3\lib\site-packages\bs4\element.py in
__getattr__(self, attr)
   735         raise AttributeError(
   736             "'%s' object has no attribute '%s'" % (
--> 737                 self.__class__.__name__, attr))
   738
   739     def output_ready(self, formatter="minimal"):

AttributeError: 'NavigableString' object has no attribute 'contents'
```

Элементы nextSibling и previousSibling позволяют пропускать следующий или предыдущий элемент на этом же уровне дерева синтаксического разбора.

```
print(soup2.head.nextSibling.name)
print(soup2.html.nextSibling)
```

Результат:

```
body
None
```

В представленном коде элемент nextSibling объекта <HEAD>Tag равен объекту <BODY>Tag, поскольку объект <BODY>Tag является следующим вложенным элементом по отношению к объекту <html>Tag. Элемент nextSibling объекта <BODY>Tag равен None, поскольку в нем больше нет вложенных по отношению к объекту <HTML>Tag элементов.

И наоборот, элемент previousSibling объекта <BODY>Tag равен объекту <HEAD>Tag, а элемент previousSibling объекта <HEAD>Tag равен None:

```
print(soup2.body.previousSibling.name)
```

```
print(soup2.head.previousSibling)
```

Результат:

```
head
None
```

Элементы next и previous позволяют передвигаться по элементам документа в том порядке, в котором они были обработаны парсером, а не в порядке появления в дереве. Например, элемент next для объекта <HEAD>Tag равен объекту <TITLE>Tag, а не объекту <BODY>Tag. Это потому, что в исходном документе, тег <TITLE> идет сразу после тега <HEAD>.

```
print(soup2.head.next)
print(soup2.head.nextSibling.name)
print(soup2.head.previous.name)
```

Результат:

```
<title>Page title</title>
body
html
```

Элементы contents объекта Tag можно перебирать, рассматривая его как список. Подобным образом можно узнать, сколько дочерних узлов имеет объект Tag, вызвав функцию len(tag) вместо len(tag.contents).

```
for el in soup2.body:
    print(el)
print(len(soup2.body))
print(len(soup2.body.contents))
```

Результат:

```
<p align="center" id="firstpara">This is paragraph <b>one</b>.</p>
<p align="blah" id="secondpara">This is paragraph <b>two</b>.</p>
2
2
```

Гораздо легче перемещаться по дереву синтаксического разбора, если в качестве имен тегов выступали элементы парсера или объекта Tag. Оператор soup2.head возвращает нам первый (как и следовало ожидать, единственный) объекта <HEAD> Tag документа.

Используя оператор soup2.p, вы перейдете к первому тегу <P> внутри документа, где бы тот ни был. Оператор soup2.table.tr.td перейдет к первому столбцу первой строки первой же таблицы документа.

Поиск в дереве синтаксического разбора

Beautiful Soup предоставляет множество методов для обхода дерева синтаксического разбора, отбирая по заданным критериям объекты Tag и NavigableString. Для определения критериев отбора объектов Beautiful Soup

есть несколько способов. Продемонстрируем доскональное исследование наиболее общего из всех методов поиска Beautiful Soup, findAll.

Методы findAll и find доступны только для объектов Tag и объектов парсера самого высокого уровня, но не для объектов NavigableString.

Метод findAll

Метод findAll является основным методом поиска.

Метод обхода дерева findAll начинает с заданной точки и ищет все объекты Tag и NavigableString, соответствующие заданным критериям. Сигнатура метода findall следующая:

findAll(name=None, attrs={}, recursive=True, text=None, limit=None, **kwargs)'"

Наиболее важными являются аргументы name и именованные аргументы.

Аргумент name

Аргумент name ограничивает набор имен тегов. Имеется несколько способов ограничить имена.

Самый простой способ – передать имя тега. Приведем код, который ищет все объекты <B> Tag в документе:

```
soup2.findAll('b')
```

Результат:

```
[<b>one</b>, <b>two</b>]
```

Также можно передать регулярное выражение. Код, который ищет все теги, имена которых начинаются на английскую букву "B":

```
import re
tagsStartingWithB = soup2.findAll(re.compile('^b'))
[tag.name for tag in tagsStartingWithB]
```

Результат:

```
['body', 'b', 'b']
```

Можно передать список или словарь. Эти два вызова ищут все теги <TITLE> и <P>. Принцип работы у них одинаков, но второй вызов отработает быстрее:

```
print(soup2.findAll(['title', 'p']))
print(soup2.findAll({'title' : True, 'p' : True}))
```

Результат:

```
[<title>Page title</title>, <p align="center" id="firstpara">This is paragraph
    <b>one</b>.</p>, <p align="blah" id="secondpara">This is paragraph
    <b>two</b>.</p>]
```

[<title>Page title</title>, This is paragraph
one. , This is paragraph
two.]

Можно передать специальное значение True, которому соответствуют все теги с любыми именами.

```
allTags = soup2.findAll(True)
[tag.name for tag in allTags]
```

Результат:

```
['html', 'head', 'title', 'body', 'p', 'b', 'p', 'b']
```

Можно передать вызываемый объект, который принимает объект Tag как единственный аргумент и возвращает логическое значение. Каждый объект Tag, который находит findAll, будет передан в этот объект, и если его вызов возвращает True, то необходимый тег найден.

Вот код, который ищет теги с двумя и только двумя атрибутами:

```
soup2.findAll(lambda tag: len(tag.attrs) == 2)
```

Результат:

```
[<p align="center" id="firstpara">This is paragraph <b>one</b>.</p>,
 <p align="blah" id="secondpara">This is paragraph <b>two</b>.</p>]
```

Данный код ищет теги, имена которых состоят из одной буквы и которые не имеют атрибутов:

```
soup2.findAll(lambda tag: len(tag.name) == 1 and not tag.attrs)
```

Результат:

```
[<b>one</b>, <b>two</b>]
```

Аргументы ключевых слов (keyword arguments) налагают ограничения на атрибуты тега. Вот простой пример поиска всех тегов, атрибут align которых имеет значение center.

Как и в случае с аргументом name вы можете передать именованный аргумент различными видами объектов для наложения разных ограничений на соответствующие атрибуты. Можно передать строку, как показано выше, чтобы ограничить значение атрибута единственным значением. Можно также передать регулярное выражение, список, хэш, специальные значения True или None, или вызываемый объект, который получает значение атрибута в качестве аргумента (обратите внимание на то, что значение может быть и None). Несколько примеров:

```
print(soup2.findAll(id=re.compile("para$")))
print(soup2.findAll(align=["center", "blah"]))
print(soup2.findAll(align=lambda value : value and len(value) < 5))
```

Результат:

```
[<p align="center" id="firstpara">This is paragraph <b>one</b>.</p>,
<p align="blah" id="secondpara">This is paragraph <b>two</b>.</p>]
[<p align="center" id="firstpara">This is paragraph <b>one</b>.</p>,
<p align="blah" id="secondpara">This is paragraph <b>two</b>.</p>]
[<p align="blah" id="secondpara">This is paragraph <b>two</b>.</p>]
```

Специальные значения True и None особо интересны. Значению True соответствует тег, заданный атрибут которого имеет любое значение, а None соответствует тег, у которого заданный атрибут не содержит значения. Несколько примеров:

```
print(soup2.findAll(align=True))
[tag.name for tag in soup2.findAll(align=None)]
```

```
[<p align="center" id="firstpara">This is paragraph <b>one</b>.</p>,
<p align="blah" id="secondpara">This is paragraph <b>two</b>.</p>]
 ['html', 'head', 'title', 'body', 'b', 'b']
```

Если необходимо наложить сложные или взаимосвязанные ограничения на атрибуты тегов, передавайте вызываемый объект для name, как показано выше, и работайте с объектом Tag.

Здесь вы должны обратить внимание на одну проблему. Что делать, если в вашем документе есть тег, который определяет атрибут с именем name? Поскольку методы поиска Beautiful Soup всегда определяют аргумент name, вы не можете использовать именованный аргумент с именем name. В качестве именованного аргумента также нельзя использовать зарезервированные слова Python, такие как for.

Beautiful Soup предоставляет специальный аргумент с именем attrs, который можно использовать в таких ситуациях; attrs представляет собой словарь, который работает также как именованные аргументы.

```
print(soup2.findAll(id=re.compile("para$")))
print(soup2.findAll(attrs={'id' : re.compile("para$")}))
```

Результат:

```
[<p align="center" id="firstpara">This is paragraph <b>one</b>.</p>,
<p align="blah" id="secondpara">This is paragraph <b>two</b>.</p>]
[<p align="center" id="firstpara">This is paragraph <b>one</b>.</p>,
<p align="blah" id="secondpara">This is paragraph <b>two</b>.</p>]
```

Аргумент recursive

Аргумент recursive – логический аргумент (по умолчанию равен True), который сообщает Beautiful Soup о том, нужно ли обходить все поддерево или искать лишь среди непосредственных потомков объекта Tag или объекта парсера. Вот в чем различие:

```
print([tag.name for tag in soup2.html.findAll()])
print([tag.name for tag in soup2.html.findAll(recursive=False)])
```

```
['head', 'title', 'body', 'p', 'b', 'p', 'b']
['head', 'body']
```

Когда аргумент recursive равен True, то просматривается все дерево и находятся все теги.

Когда аргумент recursive ложен, ищутся только непосредственные потомки тега <HTML>. А ими являются head, body.

Аргумент text

Аргумент text – аргумент, позволяющий находить вместо объектов NavigableString объекты Tag. Его значением может быть строка, регулярное выражение, список или словарь, True или None, или вызываемый объект, который получает объект NavigableString в качестве аргумента:

```
s1=soup2.findAll(text="one")
s2=soup2.findAll(text=u'one')
s3=soup2.findAll(text=["one", "two"])
s4=soup2.findAll(text=re.compile("paragraph"))
s5=soup2.findAll(text=True)
s6=soup2.findAll(text=lambda x: len(x) < 12)
print(' s1=', s1, '\n', 's2= ',s2,'\n',  's3= ',s3)
print(' s4=', s4, '\n', 's5= ',s5,'\n',  's6= ',s6)
```

Результат:

```
s1= ['one']
s2=  ['one']
s3=  ['one', 'two']
s4= ['This is paragraph ', 'This is paragraph ']
s5=  ['Page title', 'This is paragraph ', 'one', '.', 'This is paragraph ', 'two', '.']
s6=  ['Page title', 'one', '.', 'two', '.']
```

Если вы используете text, то любые значения передаются в name и именованные аргументы игнорируются.

Аргумент limit

Установка аргумента limit позволяет останавливать поиск после того, как будет найдено заданное число совпадений. Если в документе тысячи таблиц, а нужно только четыре, то передайте в аргументе limit значение 4 и сэкономьте время. По умолчанию ограничения нет.

```
l1=soup2.findAll('p', limit=1)
l2= soup2.findAll('p', limit=100)
print(' l1=', l1, '\n', 'l2= ',l2)
```

Результат:

```
l1= [<p align="center" id="firstpara">This is paragraph <b>one</b>.</p>]
l2= [<p align="center" id="firstpara">This is paragraph <b>one</b>.</p>, <p align="blah" id="secondpara">This is paragraph <b>two</b>.</p>]
```

Метод find

Метод find почти в точности совпадает с findAll, за исключением того, что он ищет первое вхождение искомого объекта, а не все. Это похоже на установку для результирующего множества limit равным 1 и затем извлечения единственного результата из массива. Ниже представлены примеры:

```
l1=soup2.findAll('p', limit=1)
l2=soup2.find('p')
l3=soup2.find('nosuchtag')
print(' l1=', l1, '\n', 'l2= ',l2,'\n',  'l3= ',l3)
```

Результат:

```
l1= [<p align="center" id="firstpara">This is paragraph <b>one</b>.</p>]
l2= <p align="center" id="firstpara">This is paragraph <b>one</b>.</p>
l3= None
```

В общем, когда используете метод поиска с множественными именами такой, как findAll, то этот метод получает аргумент limit и возвращает список результатов. Когда вы используете метод поиска без множественных имен такой, как find, вы должны понимать, что этот метод не получает limit и возвращает единственный результат.

Контрольные вопросы

1. Для чего нужна функция get() библиотеки requests?
2. Из каких объектов состоит экземпляр класса BeautifulSoup?
3. У экземпляра класса BeautifulSoup есть такие элементы как parent, nextSibling, previousSibling, next, previous?
4. Для каких объектов доступны методы findAll() и find()?
5. Чем отличается метод findAll() от метода find()?

3.3. Использование библиотеки Parsel для парсинга WEB сайтов

Извлечение данных из html связано с обходом дерева, который может осуществляться с применением различных техник и технологий. Получили широкое распространение три «языка» обхода дерева: CSS-селекторы, XPath и DSL. Первые два состоят в довольно тесном родстве и выигрывают за счет своей универсальности и широкой сфере применения.

Преимущество Parsel заключается в его широкой применимости. Это полезно для ряда ситуаций, включая:

1. Обработку данных XML / HTML Python;
2. Написание сквозных тестов для вашего сайта или приложения;
3. Простые веб-проекты с библиотекой Python Requests;
4. Простые задачи автоматизации в командной строке.

Использовать Parsel просто: нужно создать объект Selector для текста HTML или XML, который вы хотите проанализировать и использовать доступные методы для выбора частей из текста и извлечения данных из результата.

Если библиотека Parsel не установлена, то установите ее используя команду:

```
pip install parsel
```

или

```
conda install -c anaconda parsel
```

Согласно стандартам W3C, селекторы CSS не поддерживают выбор текстовых узлов или значений атрибутов. Но их выбор настолько важен в контексте очистки веб-страниц, что Parsel реализует несколько нестандартных псевдоэлементов:

- для выбора текстовых узлов, используется :: текст;
- для выбора значений атрибута, используется :: attr (name), где name − это имя атрибута, значение которого вы хотите.

Описание класса Selector:

class parsel.selector.Selector(text=None, type=None, namespaces=None, root=None, base_url=None,_expr=None)

Selector позволяет выбирать части текста XML или HTML с помощью выражений CSS или XPath и извлекать из него данные.

Аргументы:

- text – это объект типа str;
- type – определяет тип селектора, это может быть html, xml или None (по умолчанию). Если type − None, селектор по умолчанию имеет значение html.

Методы:

1. attrib – возвращает словарь атрибутов для базового элемента.
2. css(query) – возвращает экземпляр SelectorList, где query представляет собой строку, содержащую селектор CSS для применения. В фоновом режиме запросы CSS переводятся в запросы XPath с использованием библиотеки cssselect и запускают метод .xpath().extract() – возвращает совпавшие узлы в одной строке Unicode. Процент закодированного контента без кавычек.

3. get() – возвращает совпавшие узлы в одной строке Unicode. Процент закодированного контента без кавычек.

4. getall() – возврат соответствующего узла в списке строк Unicode.

5. re(regex, replace_entities = True) , возвращает список строк Unicode с совпадениями. Здесь regex – может быть либо скомпилированным регулярным выражением, либо строкой, которая будет скомпилирована в регулярное выражение с использованием re.compile (regex). По умолчанию ссылки на символьные объекты заменяются соответствующими символами (кроме & amp; и & lt;). Передача replace_entities как False отключает эти замены.

6. re_first (regex, default = None, replace_entities = True) – возвращает первую строку Unicode с совпадением. Если совпадений нет, вернуть значение по умолчанию (Нет, если аргумент не указан). По умолчанию ссылки на символьные объекты заменяются соответствующими символами (кроме & amp; и & lt;). Передача replace_entities как False отключает эти замены.

7. register_namespace(prefix, uri) – регистрирует данное пространство имен для использования в этом селекторе. Без регистрации пространств имен вы не можете выбирать или извлекать данные из нестандартных пространств имен.

8. remove_namespaces() – удаляет все пространства имен, позволяя обойти документ, используя xpath без пространства имен.

9. xpath(query, namespaces=None, **kwargs) – находит узлы, соответствующие запросу xpath, и возвращает результат в виде экземпляра SelectorList. Здесь query – это строка, содержащая запрос xpath для применения, namespaces является необязательным префиксом: namespace-uri mapping (dict) для дополнительных префиксов к тем, которые зарегистрированы в register_namespace (prefix, uri). В отличие от register_namespace () эти префиксы не сохраняются для будущих вызовов.

3.4. Практическое занятие № 6. Основы парсинга WEB сайтов с помощью библиотеки Parsel

Цель

Изучить функции библиотеки Parsel.

Научится извлекать информацию из данных XML / HTML.

Учебное задание

Выполните парсинг сайта с использованием функций библиотеки Parsel.

Технология выполнения учебного задания

Одним из полезных объектов библиотеки Parsel является класс Selector. Создание объекта Selector выполняется следующим образом.

```
htm=u"""<html>
    <head>
```

```
        <base href='http://example.com/' />
        <title>Example website</title>
    </head>
    <body>
        <div id='images'>
         <a href='image1.html'>Name: My image 1 <br /><img src='im-
    age1_thumb.jpg' /></a>
         <a href='image2.html'>Name: My image 2 <br /><img src='im-
    age2_thumb.jpg' /></a>
         <a href='image3.html'>Name: My image 3 <br /><img src='im-
    age3_thumb.jpg' /></a>
         <a href='image4.html'>Name: My image 4 <br /><img src='im-
    age4_thumb.jpg' /></a>
         <a href='image5.html'>Name: My image 5 <br /><img src='im-
    age5_thumb.jpg' /></a>
        </div>
    </body>
    </html>"""
sel = Selector(text=htm)
```

Здесь переменная htm представляет контент html-документа и передается в качестве аргумента в конструктор Selector. В результате создается объект sel. После создания объекта Selector (переменная sel), можно использовать выражения CSS или XPath для выбора элементов.

Поскольку мы имеем дело с HTML, т.е. с типом, который по умолчанию задан в конструкторе Selector, нам не нужно указывать аргумент типа.

Итак, посмотрев HTML-код этой страницы, давайте создадим XPath для выделения текста внутри тега title:

```
sel.xpath('//title/text()')
```

Результат:

```
[<Selector xpath='//title/text()' data='Example website'>]
```

Формат аргумента //title/text() достаточно запутанный, но мы можем получить аналогичный результат, используя CSS:

```
sel.css('title::text')
```

Результат:

```
[<Selector xpath='descendant-or-self::title/text()' data='Example website'>]
```

Чтобы фактически извлечь текстовые данные, вы должны вызвать методы селектора .get () или .getall () следующим образом:

```
sel.xpath('//title/text()').getall()
```

или

```
sel.xpath('//title/text()').get()
```

Результат:

```
'Example website'
```

Обратите внимание, что селекторы CSS могут выбирать узлы текста или атрибута, используя псевдоэлементы CSS3:

```
sel.css('title::text').get()
```

Метод get() всегда возвращает один результат. Если найдено несколько совпадений, возвращается содержимое первого совпадения. Если совпадений нет, возвращается None. Метод getall() возвращает список со всеми результатами.

Как видите, методы xpath() и css() возвращают экземпляр SelectorList, который представляет собой список новых селекторов. Этот API можно использовать для быстрого выбора вложенных данных:

```
sel.css('img').xpath('@src').getall()
```

Результат:

```
['image1_thumb.jpg',
 'image2_thumb.jpg',
 'image3_thumb.jpg',
 'image4_thumb.jpg',
 'image5_thumb.jpg']
```

Если вы хотите извлечь только первый соответствующий элемент, вы можете вызвать селектор get() (или его псевдоним extract_first (), который обычно использовался в предыдущих версиях parsel):

```
sel.xpath('//div[@id="images"]/a/text()').get()
'Name: My image 1 '
```

Возвращает None, если элемент не был найден:

```
sel.xpath('//div[@id="not-exists"]/text()').get()
```

Вместо использования, например, '@src' XPath может запрашивать атрибуты, используя свойство .attrib селектора:

```
image=[]
for img in sel.css('img'):
    print(img)
    image.append(img.attrib['src'])
image
 ['image1_thumb.jpg',
 'image2_thumb.jpg',
 'image3_thumb.jpg',
 'image4_thumb.jpg',
 'image5_thumb.jpg']
```

В качестве ярлыка .attrib также доступен непосредственно в SelectorList. Он возвращает атрибуты для первого соответствующего элемента:

```
sel.css('img').attrib['src']
'image1_thumb.jpg'
```

Это полезно, когда ожидается только один результат, например, при выборе по идентификатору или при выборе уникальных элементов на вебстранице. Теперь подводим итоги, как получить несколько ссылок на изображения. Чтобы получить ссылку по тегу base можно использовать операторы:

```
sel.xpath('//base/@href').get()
```

или

```
sel.css('base').attrib['href']
```

Результат:

```
'http://example.com/'
```

Чтобы получить ссылки на изображения, то используйте операторы:

```
sel.xpath('//a[contains(@href, "image")]/@href').getall()
```

или

```
[img.attrib['href'] for img in sel.css('a')]
```

или в развернутом виде

```
image=[]
for img in sel.css('a'):
   image.append(img.attrib['href'])
image
```

В результате получим список ссылок:

```
['image1.html', 'image2.html', 'image3.html', 'image4.html', 'image5.html']
```

Parsel реализует несколько нестандартных псевдоэлементов:

- для выбора текстовых узлов, используется :: текст;
- для выбора значений атрибута, используется :: attr (name), где name – это имя атрибута, значение которого вы хотите.

Используйте псевдоэлемент title::text для выбора дочерних текстовых узлов элемента-потомка <title>:

```
sel.css('title::text').get()
```

Результат:

```
'Example website'
```

Конструкция *::text выделяет все текстовые узлы-потомки контекста текущего селектора:

```
sel.css('#images *::text').getall()
```

Результат:

```
'\n            ',
 'Name: My image 1 ',
```

```
'\n              ',
'Name: My image 2 ',
'\n              ',
'Name: My image 3 ',
'\n              ',
'Name: My image 4 ',
'\n              ',
'Name: My image 5 ',
'\n            ']
```

Конструкция a::attr (href) выбирает значение атрибута href для потомков:

```
sel.css('a::attr(href)').getall()
```

Результат:

```
['image1.html', 'image2.html', 'image3.html', 'image4.html', 'image5.html']
```

У селектора также есть метод re() для извлечения данных с использованием регулярных выражений. Однако, в отличие от использования методов xpath() или css(), .re () возвращает список строк Unicode. Таким образом, вы не можете создавать вложенные вызовы re().

Есть дополнительный метод, похожий на метод get() (и его псевдоним extract_firs ()) для re(), с именем re_first(). Используйте его для извлечения только первой подходящей строки.

Помните, что если вы вкладываете селекторы и используете XPath, начинающийся с /, этот XPath будет абсолютным для документа, из которого вы его вызываете.

Например, предположим, что вы хотите извлечь все элементы <p> внутри элементов . Во-первых, вы получите все элементы :

```
divs = sel.xpath('//div')
```

Этот оператор извлекает все элементы , следующий оператор должен извлекать элементы <p>. При этом мы можем сделать ошибку. Например, следующий оператор является некорректным, так как он фактически извлекает из документа все элементы <p>, а не только те, которые находятся внутри элементов :

```
for p in divs.xpath('//p'):
   print(p.get())
```

А это правильный оператор, который делает выборку внутри элементов (обратите внимание на точку с префиксом .//p XPath):

```
for p in divs.xpath('.//p'):
    print(p.get())
```

Контрольные вопросы

1. Какие методы имеет класс Selector?
2. Для чего предназначена функция extract()?

3. Для чего предназначена функция get() и getall()?
4. Что возвращает функция attrib?
5. Для чего предназначена функция css?

3.5. Практическое занятие№7. Использование библиотеки Parsel для поиска и загрузки списка книг с сайта

Цель

Закрепить навыки парсинга веб-сайтов с использованием библиотеки Parsel.

Учебное задание

Выполните поиск и загрузку списка книг с сайта http://books.toscrape.com/. Требуется создать скрипт на Python для сбора данных и создания файла данных книг и их цен. Алгоритм должен работать следующим образом:

1) найдите на сайте ссылки на книги;
2) загрузите содержимое описания книг;
3) найдите на сайте книги информацию об ее наименовании, цене и ссылки на ее изображение;
4) сохраните полный набор данных в Excel файле.

Технология выполнения учебного заданий

1. Загрузите библиотеки requests и parsel.

```
import requests
from parsel import Selector
```

2. Присвоить результат запроса страницы (в данном случае это http://books.toscrape.com/) переменной index с помощью метода request.get(). Сохраните URL-адрес страницы в переменной url_base.

```
index = requests.get('http://books.toscrape.com/')
url_base=index.url
index.status_code
```

Переменна index представляет объект Response() и имеет свойство status_code (в данном случае это 200).

3. Прочитать содержимое запроса можно с помощью index.text (или index.content, чтобы получить значение в байтах).

```
index.text
```

Результат:

```
'<!DOCTYPE html>\n<!--[if lt IE 7]>    <html lang="en-us" class="no-js lt-ie9 lt-ie8 lt-ie7"> <![endif]-->\n<!--[if IE 7]>   <html lang="en-us" class="no-js lt-ie9 lt-ie8"> <![endif]-->\n<!--[if IE 8]>   <html lang="en-us" class="no-js lt-ie9"> <![endif]-->\n<!--[if gt IE 8]><!--> <html lang="en-us" class="no-js"> <!--
```

```
<![endif]-->\n  <head>\n      <title>\n  All products | Books to Scrape - Sand-
box\n</title>\n\n      <meta
```

Полный текст страницы отображается со всеми тегами HTML. Однако его трудно прочитать, поскольку между ними не так много пробелов. HTML страницу можно представить в виде дерева, узлы которого могут быть:

- узлы с элементами This is a paragraph ;
- узлы с атрибутами (href="page.html"внутри <a>тега);
- текстовые узлы ( ),"I have something to say";
- комментарий узлов ( ),<!-- a comment -->;
- корневые узлы;
- узлы пространства имен;
- узлы инструкций по обработке.

Согласно стандартам W3C, селекторы CSS не поддерживают выбор текстовых узлов или значений атрибутов. Но их выбор настолько важен в контексте очистки веб-страниц, что Parsel реализует несколько нестандартных псевдоэлементов:

- чтобы выбрать текстовые узлы, используйте :: текст;
- чтобы выбрать значения атрибута, используйте :: attr (name), где name − это имя атрибута, значение которого вы хотите.

Примеры:

- title :: text − выбирает дочерние текстовые узлы элемента-потомка <title>;
- :: text − выбирает все нисходящие текстовые узлы текущего селектора;
- a :: attr (href) − выбирает значение атрибута href для потомков.

4. Прежде всего создадим объект:

```
s=Selector(index.text)
```

5. Далее нам нужно найти ссылку на страницу, которая содержит описание книги. Для этого нам нужно найти эту ссылку на странице, которую мы загрузили в переменную index. Ниже представлена часть странице, которая содержит эту ссылку:

```
<article class="product_pod">
        <div class="image_container">
            <a       href="catalogue/a-light-in-the-attic_1000/index.html"><img
src="media/cache/2c/da/2cdad67c44b002e7ead0cc35693c0e8b.jpg" alt="A Light in
the Attic" class="thumbnail"></a>
        </div>
```

Требуемая ссылка содержится в качестве атрибута в узле с тегом a. Родительский узел этого узла имеет имя .

Если в качестве родительского узла выбрать product_pod, то мы получим список из пар одинаковых ссылок.

Таким образом, нам нужно выбрать узел a с атрибутом href, у которого родителем является узел image_container. Точка перед image_container делает наш поиск относительным, а не абсолютным.

Ниже представлен код:

```
ref=s.css('.image_container a::attr(href)').extract()
ref
```

Результат:

```
['catalogue/a-light-in-the-attic_1000/index.html',
 'catalogue/tipping-the-velvet_999/index.html',
 'catalogue/soumission_998/index.html',
 'catalogue/sharp-objects_997/index.html',
 'catalogue/sapiens-a-brief-history-of-humankind_996/index.html',
 'catalogue/the-requiem-red_995/index.html',
 'catalogue/the-dirty-little-secrets-of-getting-your-dream-job_994/index.html',
 'catalogue/the-coming-woman-a-novel-based-on-the-life-of-the-infamous-feminist-victoria-woodhull_993/index.html',
 'catalogue/the-boys-in-the-boat-nine-americans-and-their-epic-quest-for-gold-at-the-1936-berlin-olympics_992/index.html',
 'catalogue/the-black-maria_991/index.html',
 'catalogue/starving-hearts-triangular-trade-trilogy-1_990/index.html',
 'catalogue/shakespeares-sonnets_989/index.html',
 'catalogue/set-me-free_988/index.html',
 'catalogue/scott-pilgrims-precious-little-life-scott-pilgrim-1_987/index.html',
 'catalogue/rip-it-up-and-start-again_986/index.html',
 'catalogue/our-band-could-be-your-life-scenes-from-the-american-indie-underground-1981-1991_985/index.html',
 'catalogue/olio_984/index.html',
 'catalogue/mesaerion-the-best-science-fiction-stories-1800-1849_983/index.html',
 'catalogue/libertarianism-for-beginners_982/index.html',
 'catalogue/its-only-the-himalayas_981/index.html']
```

6. Так как полученные ссылки даны относительно базового URL, нам нужно получить полные адреса страниц и загрузить их с помощью известной функции.

```
href=ref[0]
url=url_base + href
print(url)
book_page = requests.get(url)
book_page.text
```

Результат:

```
http://books.toscrape.com/catalogue/a-light-in-the-attic_1000/index.html
```

```
'\n\n<!DOCTYPE html>\n<!--[if lt IE 7]>  <html lang="en-us" class="no-
js lt-ie9 lt-ie8 lt-ie7"> <![endif]-->\n<!--[if IE 7]>  <html lang="en-us"
class="no-js lt-ie9 lt-ie8"> <![endif]-->\n<!--[if IE 8]>  <html lang="en-
us" class="no-js lt-ie9"> <![endif]-->\n<!--[if gt IE 8]><!--> <html
lang="en-us" class="no-js"> <!--<![endif]-->\n <head>\n  <title>\n
A Light in the Attic | Books to Scrape - Sandbox\n</title>\n\n
```

7. Следующим шагом нам нужно создать объект класса Selector для полученного запроса book_page.

```
sel = Selector(book_page.text)
```

8. Ниже представлен фрагмент загруженной страницы:

```
<article class="product_page"><!-- Start of product page -->
   <div class="row">
      <div class="col-sm-6">
        <div id="product_gallery" class="carousel">
         <div class="thumbnail">
            <div class="carousel-inner">
               <div class="item active">
                     <img src= "../../me-
dia/cache/fe/72/fe72f0532301ec28892ae79a629a293c.jpg" alt="A
Light in the Attic" />
               </div>
            </div>
         </div>
       </div>
      </div>
      <div class="col-sm-6 product_main">
         <h1>A Light in the Attic</h1>
      <p class="price_color">£51.77</p>
```

Название книги содержится в текстовом узле <h1>A Light in the Attic</h1>. Поэтому получить название книги можно с помощью оператора:

```
name=sel.css('h1::text').extract_first()
name
```

Результат:

```
'A Light in the Attic'
```

Цена книги содержится в текстовом узле:

```
<p class="price_color">£51.77</p>
```

В принципе в качестве аргумента мы можем указать 'p::text', как и выше, но более правильно сделать запрос более конкретным (чтобы не было совпадений) '.price_color::text'.

```
pr=    sel.css('.price_color::text').extract_first()
pr
```

Результат:

```
'Â£51.77'
```

Ну и наконец, ссылка на изображение книги содержится в следующем фрагменте:

```
<<article class="product_page"><!-- Start of product page -->
   <div class="row">
      <div class="col-sm-6">
        <div id="product_gallery" class="carousel">
         <div class="thumbnail">
            <div class="carousel-inner">
          <div class="item active">
                <img                                                    src="../../me-
dia/cache/fe/72/fe72f0532301ec28892ae79a629a293c.jpg" alt="A Light in the Attic"
/>
```

По сути это атрибутный узел, но в качестве родительского узла укажем #product_page.

```
image_ref=sel.css('#product_gallery img::attr(src)').extract_first()
image_ref
```

Результат:

```
'../../media/cache/fe/72/fe72f0532301ec28892ae79a629a293c.jpg'
```

9. Теперь организуем сбор о всех книгах на этой странице.

```
title=[]
price=[]
image=[]
print(url_base)
for href in Selector(index.text).css('.image_container a::attr(href)').ex-
      tract():
   url=url_base + href
   print(url)
   book_page = requests.get(url)
   sel = Selector(book_page.text)
   title.append( sel.css('h1::text').extract_first())
   price.append(sel.css('.price_color::text').extract_first())
   image.append(sel.css('#product_gallery img::attr(src)').extract_first())
```

Результат:

```
http://books.toscrape.com/
http://books.toscrape.com/catalogue/a-light-in-the-attic_1000/index.html
```

```
http://books.toscrape.com/catalogue/tipping-the-velvet_999/index.html
http://books.toscrape.com/catalogue/soumission_998/index.html
http://books.toscrape.com/catalogue/sharp-objects_997/index.html
http://books.toscrape.com/catalogue/sapiens-a-brief-history-of-human-kind_996/index.html
http://books.toscrape.com/catalogue/the-requiem-red_995/index.html
http://books.toscrape.com/catalogue/the-dirty-little-secrets-of-getting-your-dream-job_994/index.html
http://books.toscrape.com/catalogue/the-coming-woman-a-novel-based-on-the-life-of-the-infamous-feminist-victoria-woodhull_993/index.html
```

10. Список price представляет сочетание букв и цифр, в то время как для обработки требуются числа. Для того чтобы преобразовать в число, нужно отбросить первые два символа.

```
price1=[]
print("price", price)
for s in price:
    f=float(s[2:])
    price1.append(f)
print("price1", price1)
```

Результат:

```
price ['Â£51.77', 'Â£53.74', 'Â£50.10', 'Â£47.82', 'Â£54.23', 'Â£22.65', 'Â£33.34', 'Â£17.93', 'Â£22.60', 'Â£52.15', 'Â£13.99', 'Â£20.66', 'Â£17.46', 'Â£52.29', 'Â£35.02', 'Â£57.25', 'Â£23.88', 'Â£37.59', 'Â£51.33', 'Â£45.17']
price1 [51.77, 53.74, 50.1, 47.82, 54.23, 22.65, 33.34, 17.93, 22.6, 52.15, 13.99, 20.66, 17.46, 52.29, 35.02, 57.25, 23.88, 37.59, 51.33, 45.17]
```

Есть другие более универсальные способы удаления ненужных символов из строки. Для этого нужно воспользоваться библиотекой re.

Библиотека re предоставляет операции сопоставления регулярных выражений. Регулярное выражение определяет набор строк, а функции в этом библиотеке позволяют проверить, соответствует ли конкретная строка заданному регулярному выражению.

При обработке строки необходимо в ней оставить только числа и точку. Поэтому можно использовать функцию re.sub().

```
price1=[]
print("price",price)
for s in price:
    f=re.sub("[^1234567890\.]" ,'', s)
    price1.append(f)
print("price1",price1)
```

Результат будет тот же.

Регулярные выражения могут содержать как специальные, так и обычные символы. В таблице 3.2 представлены наиболее часто используемые специальные символы.

Таблица 3.2

Оператор	Значение
.	Один любой символ, кроме новой строки \n.
?	0 или 1 вхождение шаблона слева
+	1 и более вхождений шаблона слева
*	0 и более вхождений шаблона слева
\w	Любая цифра или буква (\W – все, кроме буквы или цифры)
\d	Любая цифра [0-9] (\D – все, кроме цифры)
\s	Любой пробельный символ (\S – любой непробельный символ)
\b	Граница слова
[..]	Один из символов в скобках ([^..] – любой символ, кроме тех, что в скобках)
\	Экранирование специальных символов (\. означает точку или \+ означает знак «плюс»)
^ и $	Начало и конец строки соответственно
{n,m}	От n до m вхождений ({,m} – от 0 до m)
a|b	Соответствует a или b
()	Группирует выражение и возвращает найденный текст
\t, \n, \r	Символ табуляции, новой строки и возврата каретки соответственно

11. Далее создаем пустую таблицу типа DataFrame.

```
import pandas as pd
n=len(title)
print("Число книг",n)
zero_data = np.zeros(shape=(n,3))
books= pd.DataFrame(zero_data, axis=1), columns=['title', 'price', 'image'])
books.head()
```

Результат: число книг 20.

После этого мы заполняем таблицу:

```
books['title']=title
books['price']=price1
books['image']=image
books.head()
```

Результат представлен в таблице 4.1.

И наконец, сохраняем эту таблицу в Excel в файле books.csv. Обратите внимание, что путь до файла нужно откорректировать.

```
books.to_csv('../BD/books.csv',sep=';', decimal=',')
```

Давайте поставим задачу собрать информацию о книгах не только на первой странице, но и на других страницах. Для этого создадим массив адресов страниц, представленный в таблице 3.3.

Таблица 3.3

№	title	price	image
0	A Light in the Attic	51.77	../../media/cache/fe/72/fe72f0532301ec28892ae7...
1	Tipping the Velvet	53.74	../../media/cache/08/e9/08e94f3731d7d6b760dfbf...
2	Soumission	50.10	../../media/cache/ee/cf/eecfe998905e455df12064...
3	Sharp Objects	47.82	../../media/cache/c0/59/c05972805aa7201171b8fc...
4	Sapiens: A Brief History of Humankind	54.23	../../media/cache/ce/5f/ce5f052c65cc963cf4422b.

```
url_base_page="http://books.toscrape.com/catalogue/"
title=[]
price=[]
image=[]
url_lst=['http://books.toscrape.com/catalogue/page-
      1.html','http://books.toscrape.com/catalogue/page-2.html',
      'http://books.toscrape.com/catalogue/page-
      3.html','http://books.toscrape.com/catalogue/page-4.html',
      'http://books.toscrape.com/catalogue/page-
      5.html','http://books.toscrape.com/catalogue/page-6.html',
      'http://books.toscrape.com/catalogue/page-
      7.html','http://books.toscrape.com/catalogue/page-8.html']
for url_page in url_lst:
   print(url_page)
   index = requests.get(url_page)
   for href in Selector(index.text).css('.image_container a::attr(href)').ex-
      tract():
      url = url_base_page + href
      print(url)
      book_page = requests.get(url)
      if book_page.status_code == 200:
         sel = Selector(book_page.text)
         title.append( sel.css('h1::text').extract_first())
         price.append(sel.css('.price_color::text').extract_first())
         image.append(sel.css('#product_gallery img::attr(src)').ex-
      tract_first())
```

Попробуйте разобраться в выше представленном коде, чем он отличается от предыдущего кода загрузки одной страницы.

Далее мы проводим точно такие же операции, как и в случаи обработки одной страницы.

Сначала создаем пустую таблицу:

```
n=len(title)
print('Число книг',n)
zero_data = np.zeros(shape=(n,3))
books_all= pd.DataFrame(zero_data, columns=['title', 'price', 'image'])
```

Далее преобразуем список price:

```
price1=[]
print("price",price)
for s in price:
   f=re.sub("[^1234567890\.]" ,'', s)
   price1.append(f)
print("price1",price1
```

Заполняем таблицу и сохраняем ее. Не забывайте корректировать путь до файла.

```
books_all['title']=title
books_all['price']=price1
books_all['image']=image
books_all.to_csv('../BD/books_all.csv',sep=';', decimal=',')
books_all.head()
```

Глава 4. ИЗВЛЕЧЕНИЕ ДАННЫХ ИЗ ИНТЕРНЕТ-РЕСУРСА

4.1. Источники данных в интеренете

Функции из pandas_datareader.data и pandas_datareader.wb извлекают данные из различных интернет-источников в DataFrame pandas. В настоящее время поддерживаются следующие источники:

- Google Finance,
- Morningstar,
- IEX,
- Robinhood,
- Enigma,
- Quandl,
- St.Louis FED (FRED),
- Kenneth French’s data library,
- World Bank,
- OECD,
- Eurostat,
- Thrift Savings Plan,
- Nasdaq Trader symbol definitions,
- Stooq,
- MOEX.

Следует отметить, что различные источники поддерживают разные виды данных, поэтому не все источники реализуют одинаковые методы, и возвращаемые элементы данных также могут отличаться.

API Google стал менее надежным в 2017 году. Несмотря на то, что Google Data Reader часто работает, как и ожидалось, нередко возникают ошибки при попытке чтения особенно массовых данных.

Tiingo является платформой отслеживания, которая предоставляет данные с историческими ценами на конец дня на акции, инвестиционные фонды и ETF. Для получения ключа API необходима бесплатная регистрация. Бесплатные аккаунты ограничены по скорости и могут получать доступ к ограниченному количеству бумаг (500 на момент написания).

4.2. Практическое занятие №8. Поиск и загрузка данных из интернет ресурса Google Finance

Цель

Получить навыки чтения данных из данные из различных баз данных расположенных в интернет.

Учебное задание

Поиск и загрузка списка компаний S&P500 и соответствующих курсов ценных бумаг компаний является трудоемким процессом. Требуется создать скрипт на Python для создания базы ежедневных данных ценных бумаг компаний из списка S&P500.

Алгоритм должен работать следующим образом:

1) сформируйте список секторов промышленности из списка компаний S&P500 в Википедии и список тиккеров S&P500 для этих секторов;

2) сформируйте таблицу ежедневных данных стоимости ценных бумаг для каждого сектора промышленности из финансов Yahoo, используя pandas DataReader;

3) отрегулируйте данные открытия, максимума и минимума, используя соотношение скорректированного закрытия к закрытию;

4) сохраните полный набор данных в локальном файле Excel, проиндексированном по отраслям.

Вы можете изменить даты начала и окончания, используя переменные START и END в верхней части таблицы.

Технология выполнения учебного заданий

Будет использоваться словарная структура, где каждый сектор соответствует ключу, а в каждом секторе нужное нам поле данных соответствует другому ключу. Итак, наша структура: Сектор, Поле (Open, High, Low, Adj, Close Volume), Временной ряд тиккера.

Загрузите библиотеку requests:

```
import requests
```

Присвойте URL-адрес тестовой страницы (в данном случае это https://en.wikipedia.org/wiki/List_of_S%26P_500_companies) переменной url.

```
url="https://en.wikipedia.org/wiki/List_of_S%26P_500_companies"
```

Затем можно присвоить результат запроса этой страницы переменной page с помощью метода request.get(). Передайте URL-адрес страницы, который был присвоен переменной url, этому методу.

```
page = requests.get(url)
```

Объект Response сообщает свойство status_code в квадратных скобках (в данном случае это 200). Этот атрибут можно вызвать явно:

```
page.status_code
```

Возвращаемый код 200 сообщает, что страница загружена успешно.

Проверить загрузку сайта можно с помощью кода:

```
try:
    # If the response was successful, no Exception will be raised
    page.raise_for_status()
except HTTPError as http_err:
    print(f'HTTP error occurred: {http_err}')  # Python 3.6
except Exception as err:
```

```
        print(f'Other error occurred: {err}')  # Python 3.6
    else:
        print('Success!')
```

Чтобы работать с веб-данными, нужно получить доступ к текстовому содержимому веб-файлов. Прочитать содержимое ответа сервера можно с помощью page.text (или page.content, чтобы получить значение в байтах).

Полный текст страницы был отображен со всеми тегами HTML. Однако, его трудно прочитать, поскольку между ними не так много пробелов.

```
page.content
```

Результат:

```
b'<!DOCTYPE html>\n<html class="client-nojs" lang="en" dir="ltr">\n<head>\n<meta charset="UTF-8"/>\n<title>List of S&amp;P 500 companies - Wikipedia</title>\n<script>document.documentElement.className=document.documentElement.className.replace(/(^|\\s)client-nojs(\\s|$)/,"$1client-js .....
```

Анализ веб-данных с помощью Beautiful Soup

Импортируйте Beautiful Soup:

```
from bs4 import BeautifulSoup
```

Затем нужно получить объект BeautifulSoup, т.е. дерево синтаксического разбора этой страницы, полученной с помощью встроенного html.parser через HTML. Построенный объект представляет документ в виде вложенной структуры данных, которая присваивается переменной soup.

```
soup = BeautifulSoup(page.text, 'html.parser')
```

Чтобы отобразить содержимое страницы в терминале, используйте метод prettify(), который превратит дерево разбора Beautiful Soup в красиво отформатированную строку Unicode.

Извлечь один тег со страницы можно с помощью метода findAll() или find(). Нужная нам таблица имеет тег table.

```
<table class="wikitable sortable" id="constituents">
<tbody><tr>
<th><a href="/wiki/Symbol" title="Symbol">Symbol</a>
```

Мы можем извлечь нужный нам текст с помощью оператора:

```
table = soup.findAll("table")
type(table)
```

Элементы HTML, относящиеся к селекторам CSS, такие как класс и ID, могут быть полезны при работе с веб-данными и Beautiful Soup. Можно указать, что класс wikitable sortable нужно искать в тегах.

```
table = soup.findAll(class_ ="wikitable sortable")
type(table)
```

Эти операторы вернут все экземпляры тега table в документе. Результат будет:

```
bs4.element.ResultSet
```

Так как на странице у нас не одна таблица (а две), поэтому мы имеем список ResultSet. Мы можем с этим результатом работать как с обычным списком:

```
print("Число таблиц", len(table))
print(type(table[0]))
```

Результат:

```
Число таблиц 2
<class 'bs4.element.Tag'>
```

Мы можем использовать метод find(), который ищет только одно вхождение.

```
table0 = soup.find("table")
type(table)
```

Можно указать в качестве дополнительного элемента при поиске class: wikitable sortable.

```
table0 = soup.find("table", {"class": "wikitable sortable"})
type(table0)
```

Результат будет одинаковым:

```
bs4.element.Tag
```

Извлечь строки таблицы можно, если искать в найденном нами фрагменте экземпляры тега td. Но экземпляр ResultSet не имеет методов findAll() и find(), а вот экземпляр element.Tag может вызывать эти методы.

```
tbl=table0.findAll('tr')
type(tbl)
```

Результат:

```
bs4.element.ResultSet
```

Вместо элемента table0 мы можем использовать первый элемент списка table[0]. Результаты поиска тега td возвращаются в виде списка элементов element.ResultSet. Каждый элемент описывает одну строку таблицы. Определим число строк и длину элемента, описывающего строку.

```
import numpy as np
print("Число строк",len(tbl))
for i in np.arange(len(tbl)):
   print("number", i, "Lenght",len(tbl[i]))
```

Результат:

```
Число строк 506
number 0 Lenght 18
number 1 Lenght 18
number 2 Lenght 18
number 3 Lenght 18
```

```
number 4 Lenght 18
number 5 Lenght 18
number 6 Lenght 18
number 7 Lenght 18
```

Первый элемент списка должен описывать строку с именами столбцов таблицы.

```
print(tbl[0])
```

Результат:

```
<tr>
<th><a href="/wiki/Symbol" title="Symbol">Symbol</a>
</th>
<th>Security</th>
<th><a href="/wiki/SEC_filing" title="SEC filing">SEC filings</a></th>
<th><a href="/wiki/Global_Industry_Classification_Standard" title="Global Industry Classification Standard">GICS</a> Sector</th>
<th>GICS Sub Industry</th>
<th>Headquarters Location</th>
<th>Date first added</th>
<th><a href="/wiki/Central_Index_Key" title="Central Index Key">CIK</a></th>
<th>Founded
</th></tr>
```

Элементы строки выделяются тегами th. Таким образом, используем метод findAll() для формирования списка элементов строки-заголовка.

```
name=tbl[0].findAll("th")
for i in np.arange(len(name)):
    print(name[i].get_text())
```

Результат:

```
Symbol
Security
SEC filings
GICS Sector
GICS Sub Industry
Headquarters Location
Date first added
CIK
Founded
```

Нас интересует 0-й и 3-й элемент списка. 0-й элемент содержит имя (тикер) ценной бумаги, а 3-й элемент − название сектора промышленности. Распечатаем 1-й элемент списка tbl:

```
print(tbl[1])
```

Результат:

```
<tr>
<td><a class="external text" href="https://www.nyse.com/quote/XNYS:MMM" rel="nofollow">MMM</a>
</td>
<td><a href="/wiki/3M" title="3M">3M Company</a></td>
<td><a class="external text" href="https://www.sec.gov/cgi-bin/browse-edgar?CIK=MMM&amp;action=getcompany" rel="nofollow">reports</a></td>
<td>Industrials</td>
<td>Industrial Conglomerates</td>
<td><a class="mw-redirect" href="/wiki/St._Paul,_Minnesota" title="St. Paul, Minnesota">St. Paul, Minnesota</a></td>
<td></td>
<td>0000066740</td>
<td>1902
</td></tr>
```

Это первая строка таблицы, содержащая значения столбцов. Элементы строки имеют теги <td>.

```
row=tbl[1].findAll("td")
for i in np.arange(len(row)):
    print(row[i].get_text())
```

Результат:

```
MMM
3M Company
reports
Industrials
Industrial Conglomerates
St. Paul, Minnesota
0000066740
1902
```

Нулевой элемент списка tbl[1].findAll("td") содержит имя бумаги MMM, а третий элемент – название сектора промышленности Industrials.

Соберем всю информацию из таблицы:

```
sector_tickers = dict()
tbl=table0.findAll("tr")
for row in tbl:
```

```
            col = row.findAll("td")
            if len(col) > 0:
                sector = str(col[3].string.strip()).lower().replace(" ", "_")
                print("sector", sector)
                ticker = str(col[0].get_text())
                print('ticker',ticker)
                if sector not in sector_tickers:
                    sector_tickers[sector] = list()
                sector_tickers[sector].append(ticker)
```

Результат:

```
sector industrials
ticker MMM
sector health_care
ticker ABT
sector health_care
ticker ABBV
sector health_care
ticker ABMD
sector information_technology
ticker ACN
sector communication_services
ticker ATVI
.....
```

Распечатайте полученный словарь:

```
        for sector, ticker in sector_tickers.items():
            print("Downloading sector: " + sector )
            print(ticker)
```

Результат:

```
Downloading sector: industrials
['MMM\n', 'ALK\n', 'ALLE\n', 'AAL\n', 'AME\n', 'AOS\n', 'ARNC\n', 'BA\n',
'CHRW\n', 'CAT\n', 'CTAS\n', 'CPRT\n', 'CSX\n', 'CMI\n', 'DE\n', 'DAL\n',
'DOV\n', 'ETN\n', 'EMR\n', 'EFX\n', 'EXPD\n', 'FAST\n', 'FDX\n', 'FLS\n',
'FTV\n', 'FBHS\n', 'GD\n', 'GE\n', 'GWW\n', 'HON\n', 'HII\n', 'IEX\n',
'INFO\n', 'ITW\n', 'IR\n', 'JEC\n', 'JBHT\n', 'JCI\n', 'KSU\n', 'LHX\n', 'LMT\n',
'MAS\n', 'NLSN\n', 'NSC\n', 'NOC\n', 'PCAR\n', 'PH\n', 'PNR\n', 'PWR\n',
'RTN\n', 'RSG\n', 'RHI\n', 'ROK\n', 'ROL\n', 'ROP\n', 'SNA\n', 'LUV\n',
'SWK\n', 'TXT\n', 'TDG\n', 'UNP\n', 'UAL\n', 'UPS\n', 'URI\n', 'UTX\n',
'VRSK\n', 'WAB\n', 'WM\n', 'XYL\n']
Downloading sector: health_care
['MMM\n', 'ALK\n', 'ALLE\n', 'AAL\n', 'AME\n', 'AOS\n', 'ARNC\n', 'BA\n',
'CHRW\n', 'CAT\n', 'CTAS\n', 'CPRT\n', 'CSX\n', 'CMI\n', 'DE\n', 'DAL\n',
```

```
'DOV\n', 'ETN\n', 'EMR\n', 'EFX\n', 'EXPD\n', 'FAST\n', 'FDX\n', 'FLS\n',
'FTV\n', 'FBHS\n', 'GD\n', 'GE\n', 'GWW\n', 'HON\n', 'HII\n', 'IEX\n',
'INFO\n', 'ITW\n', 'IR\n', 'JEC\n', 'JBHT\n', 'JCI\n', 'KSU\n', 'LHX\n',
.....
```

```
tickers= sector_tickers['industrials']
print(tickers)
tic=[]
for t in tickers:
    tt=t.strip()
    if tt.find('.') == -1:
        tic.append(t.strip())
print(tic)
```

Результат:

```
['MMM\n', 'ALK\n', 'ALLE\n', 'AAL\n', 'AME\n', 'AOS\n', 'ARNC\n', 'BA\n',
'CHRW\n', 'CAT\n', 'CTAS\n', 'CPRT\n', 'CSX\n', 'CMI\n', 'DE\n', 'DAL\n',
'DOV\n', 'ETN\n', 'EMR\n', 'EFX\n', 'EXPD\n', 'FAST\n', 'FDX\n', 'FLS\n',
'FTV\n', 'FBHS\n', 'GD\n', 'GE\n', 'GWW\n', 'HON\n', 'HII\n', 'IEX\n',
'INFO\n', 'ITW\n', 'IR\n', 'JEC\n', 'JBHT\n', 'JCI\n', 'KSU\n', 'LHX\n', 'LMT\n',
'MAS\n', 'NLSN\n', 'NSC\n', 'NOC\n', 'PCAR\n', 'PH\n', 'PNR\n', 'PWR\n',
'RTN\n', 'RSG\n', 'RHI\n', 'ROK\n', 'ROL\n', 'ROP\n', 'SNA\n', 'LUV\n',
'SWK\n', 'TXT\n', 'TDG\n', 'UNP\n', 'UAL\n', 'UPS\n', 'URI\n', 'UTX\n',
'VRSK\n', 'WAB\n', 'WM\n', 'XYL\n']
['MMM', 'ALK', 'ALLE', 'AAL', 'AME', 'AOS', 'ARNC', 'BA', 'CHRW', 'CAT', 'CTAS',
'CPRT', 'CSX', 'CMI', 'DE', 'DAL', 'DOV', 'ETN', 'EMR', 'EFX', 'EXPD', 'FAST',
'FDX', 'FLS', 'FTV', 'FBHS', 'GD', 'GE', 'GWW', 'HON', 'HII', 'IEX', 'INFO', 'ITW',
'IR', 'JEC', 'JBHT', 'JCI', 'KSU', 'LHX', 'LMT', 'MAS', 'NLSN', 'NSC', 'NOC', 'PCAR',
'PH', 'PNR', 'PWR', 'RTN', 'RSG', 'RHI', 'ROK', 'ROL', 'ROP', 'SNA', 'LUV', 'SWK',
'TXT', 'TDG', 'UNP', 'UAL', 'UPS', 'URI', 'UTX', 'VRSK', 'WAB', 'WM', 'XYL']
```

Используя DataReader скачиваем из баз данных Yahoo finance ежедневные данные о стоимости бумаг из списка tic. Например, MMM.

```
from datetime import datetime
from pandas_datareader import data
start = datetime(2019,5,1)
start=start.strftime("%Y-%m-%d")
end = datetime.utcnow()
end=end.strftime("%Y-%m-%d")
print(start, end)
tmp = data.DataReader('MMM', "yahoo", start, end)
tmp.head()
```

Результат представлен на рис. 4.1.

```
2019-05-01 2019-08-21
```

Date	High	Low	Open	Close	Volume	Adj Close
2019-04-30	190.860001	186.550003	190.860001	189.509995	4541000.0	186.198486
2019-05-01	189.710007	185.919998	189.490005	186.070007	3818500.0	182.818604
2019-05-02	186.500000	183.380005	184.500000	184.750000	5654900.0	181.521667
2019-05-03	186.690002	184.089996	185.809998	185.220001	4747700.0	181.983444
2019-05-06	183.110001	180.130005	182.039993	183.039993	6517600.0	179.841537

Рис. 4.1. Ежедневные данные MMM

Переменные start и end определяют период сбора данных. Таблица содержит столбцы максимальных и минимальных значений стомости, цены открытия, закрытия, объемы продаж и цены скорректированного закрытия.

Для того чтобы собрать ежедневные данные по всем бумагам можно использовать следующий скрипт:

```
dataAll=pd.DataFrame()
print(start, end)
for t in tic:
    tmp = data.DataReader(t, "yahoo", start, end)
    tmp.columns=t+tmp.columns
    dataAll=pd.concat([dataAll, tmp], axis=1)
dataAll.head()
```

Результат представлен на рис. 4.2.

```
2019-05-01 2019-08-01
```

Date	MMMHigh	MMMLow	MMMOpen	MMMClose	MMMVolume	MMMAdj Close	ALKHigh	ALKLow
2019-04-30	190.860001	186.550003	190.860001	189.509995	4541000.0	186.198486	62.380001	61.180000
2019-05-01	189.710007	185.919998	189.490005	186.070007	3818500.0	182.818604	62.459999	61.720001
2019-05-02	186.500000	183.380005	184.500000	184.750000	5654900.0	181.521667	63.200001	61.980000
2019-05-03	186.690002	184.089996	185.809998	185.220001	4747700.0	181.983444	63.259998	62.349998
2019-05-06	183.110001	180.130005	182.039993	183.039993	6517600.0	179.841537	62.290001	61.150002

5 rows x 414 columns

Рис. 4.2. Ежедневные данные по нескольким бумагам

Сохраните полученную таблицу в Excel файле.

```
dataAll.to_csv("../BD/dataAll.csv", decimal=',', sep=';')
```

Путь до файла необходимо скорректировать в соответствии со своим рабочим местом.