Машинное обучение

Библиотека Pandas

Материалы > Анализ и обработка данных

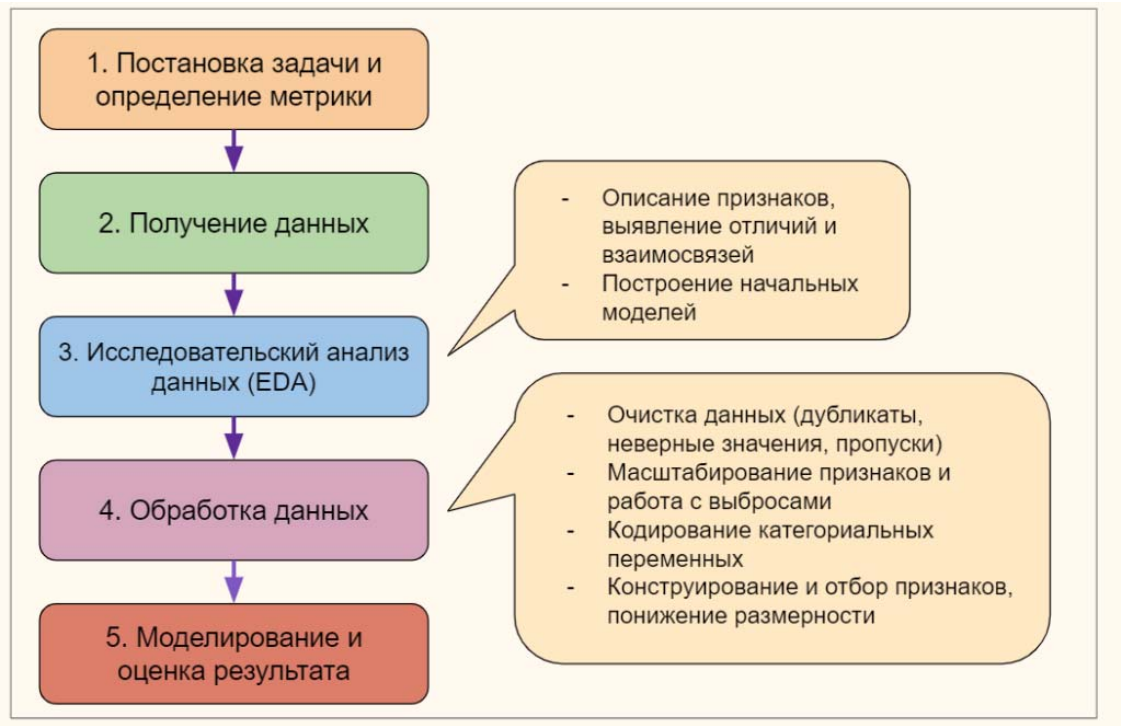
Мы начинаем курс анализа и обработки данных. На двух предыдущих курсах мы, во-первых, получили общее представление о том, как устроено машинное обучение, и, во-вторых, приобрели достаточно продвинутые навыки в программировании на Питоне.

Курс анализа данных — это первый курс, в рамках которого мы переходим непосредственно к работе над построением моделей.

Для того чтобы лучше понимать содержание этого курса, давайте вначале рассмотрим этапы построения модели.

Задача машинного обучения

Ниже представлена общая последовательность работы над задачей машинного обучения.



Описанный процесс принято называть **пайплайном**, то есть порядком действий (от англ. pipeline, трубопровод, конвейер), которые необходимо выполнить для построения модели. Подробнее рассмотрим некоторые из этих этапов.

Этап 1. Постановка задачи и определение метрики

Первый этап может показаться тривиальным, однако во многих случах, особенно в задачах классификации, **выбор правильной метрики** является ключевым для построения качественной модели. Про важность классификационной метрики мы начали говорить в рамках вводного курса и продолжим этот разговор в дальнейшем на гораздо более детальном уровне.

Этап 2. Получение данных

Важный этап. Хотя на этом курсе мы будем использовать уже готовые (зачастую учебные) датасеты, стоит помнить, что получение данных (data gathering) не происходит само собой и во многом от того как и какие данные собраны будет зависеть конечный результат (на который не смогут повлиять ни качественный EDA, ни сложный алгоритм машинного обучения).

Этап 3. Исследовательский анализ данных

В рамках EDA нам нужно решить три основные задачи: описать данные, найти отличия и выявить взаимосвязи. **Описание данных** позволяет понять, о каких данных идет речь, а также выявить недостатки в данных (с которыми мы справляемся на этапе обработки). **Отличия и взаимосвязи** в данных — основа для построения модели, это то, за что модель «цепляется», чтобы выдать верный числовой результат, правильно классифицировать или сформировать кластер.

Для решения этих задач наилучшим образом подходят средства визуализации и описательная статистика. Этим мы займемся во втором разделе.

Отдельно хочется сказать про **baseline models**, простые модели, которые мы строим в самом начале работы. Они позволяют понять, какой результат мы можем получить, не вкладывая дополнительных усилий в работу с данными, а затем отталкиваться от этого результата для обработки данных и построения более сложных моделей.

> Базовые модели мы начнем строить на курсе по обучению модели.

Этап 4. Обработка данных

Как уже было сказано, на этапе EDA зачастую становится очевидно, что в данных есть недостатки, которые сильно повляют на качество модели или в целом не позволят ее обучить.

Очистка данных: ошибки и пропуски

Во-первых, в данных могут встречаться **ошибки**: дубликаты, неверные значения или неподходящий формат данных. Кроме того, данные могут содержать **пропуски**, и с ними также нужно что-то делать. Этим вопросам посвящен третий раздел курса.

Преобразование данных

Во-вторых, зачастую *количественные данные* нужно привести к одному масштабу и/или нормальному распределению. Кроме того, числовые признаки могут содержать сильно отличающиеся от остальных данных значения или выбросы, которые также повляют на конечный результат. *Категориальные данные* необходимо закодировать с помощью чисел. Если категориальные данные выражены строками, это может воспрепятствовать обучению алгоритма.

Преобразование количественных и категориальных данных рассматривается в четвертом разделе.

Конструирование и отбор признаков, понижение размерности

Еще одним важным этапом является *конструирование признаков*, а также *отбор признаков* и *понижение размерности*. В рамках этого курса мы затронем лишь базовые способы конструирования признаков. Более сложные вопросы отбора признаков и понижения размерности мы отложим на потом.

Этап 5. Моделирование и оценка результата

Когда данные готовы, их можно загружать в модель, обучать эту модель и оценивать результат.

Здесь важно сказать, что это *итеративный* (iterative) или циклический процесс. Многие из описанных выше шагов могут повторяться. В частности, построение модели может привести к необходимости дополнительной обработки данных. Кроме того, разные алгоритмы требуют разной подготовки данных (например, линейные модели требуют масштабирования данных, а для деревьев решений этого не нужно).

При этом прежде чем приступить к анализу и обработке данных, важно освоить библиотеку Pandas. Именно этим мы и займемся в начале курса.

Про библиотеку Pandas

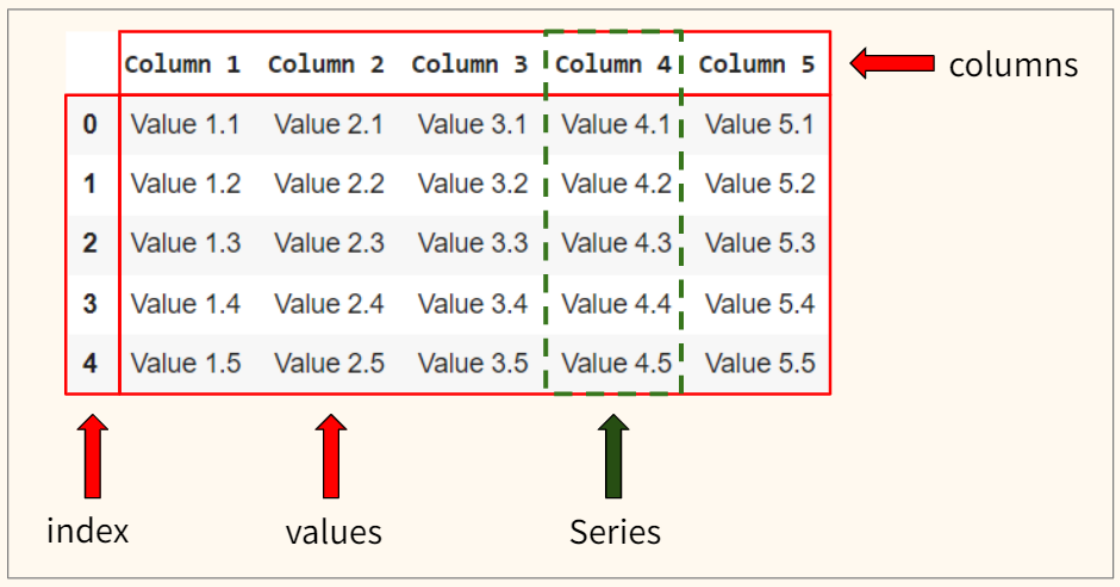
Библиотека Pandas — это ключевой инструмент для анализа данных в Питоне. Она позволяет работать с данными, представленными в табличной форме, а также временными рядами. Как вы уже видели, Pandas легко интегрируется с matplotlib, seaborn, sklearn и другими библиотеками.

Кроме того, структурно изучение Pandas можно разделить на две части: преобразование данных и статистический анализ. В этом разделе (первое и второе занятие) мы начнем знакомиться с первой частью, а в следующем (занятия три и четыре) перейдем ко второй.

Откроем блокнот к этому занятию⧉

Объекты DataFrame и Series

В Pandas есть два основных объекта: DataFrame и Series. Слегка упрощая, можно сказать, что DataFrame — это таблица (и соответственно двумерный массив), а Series — столбец этой таблицы (а значит одномерный массив).



Индекс (index), столбцы (columns) и значения (values) датафрейма мы разберем чуть позднее, а сейчас давайте посмотрим на способы создания датафрейма.

Создание датафрейма

Способ 1. Создание датафрейма из файла

Датафрейм можно создать, импортировав файлы различных форматов. Мы уже делали так с файлами в формате .csv с помощью **функции pd.read_csv()**. Аналогичным образом можно импортировать файлы и в других форматах, например MS Excel или html. Рассмотрим несколько примеров.

Файл .csv в zip-архиве. Если в zip-архиве содержится только один файл, его можно напрямую подгрузить в Pandas. Вначале скачаем необходимые данные.

train.zip **Скачать**

После скачивания не забудьте подгрузить данные в сессионное хранилище Google Colab.

```
# испортируем файл из папки content и выведем первые три строки
csv_zip = pd.read_csv('/content/train.zip')
csv_zip.head(3)
```

	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

Как вы наверное узнали, речь идет о датасете «Титаник».

MS Excel. Похожим образом мы можем импортировать файлы в формате Excel. Скачаем файл.

iris.xslx **Скачать**

Теперь используем функцию **pd.read_excel()** для импорта файла.

```
# импортируем данные в формате Excel, указав номер листа, который хотим использовать
excel_data = pd.read_excel('/content/iris.xlsx', sheet_name = 0)
excel_data.head(3)
```

	Petal_width	Petal_length	Sepal_width	Sepal_length	Species_name
0	0.2	1.4	3.5	5.1	Setosa
1	0.2	1.4	3.0	4.9	Setosa
2	0.2	1.3	3.2	4.7	Setosa

С датасетом про цветы ириса мы также уже знакомы.

> Дополнительно зачему, что файлы можно хранить локально на своем компьютере (если вы используете Jupyter Notebook), импортировать в сессионное хранилище Google Colab или передать в функцию импорта ссылку на файл в Интернете.

Файл в формате html. Данные можно загружать не только из файла, но и брать из Интернета. Для этого подойдет **функция read_html()**. По большому счету речь идет о примере несложного **веб-скрапинга** (web scraping), то есть получения информации с веб-страниц в автоматизированном режиме.

Возьмем страницу Википедии про население Земли. В частности предположим, что нас будет интересовать вот эта таблица.

World population milestones in billions[60] (Worldometers estimates)

Population	1	2	3	4	5	6	7	8	9	10
Year	1804	1927	1960	1974	1987	1999	2011	2022	*2037*	*2057*
Years elapsed	200,000+	123	33	14	13	12	12	11	*15*	*20*

```
# передадим соответствующую ссылку в функцию pd.read_html()
# в параметре match укажем ключевые слова, которые помогут найти нужную таблицу
html_data = pd.read_html('https://en.wikipedia.org/wiki/World_population',
                         match = 'World population')

# мы получили пять результатов
len(html_data)
```

```
5
```

Посмотрим на первый полученный результат.

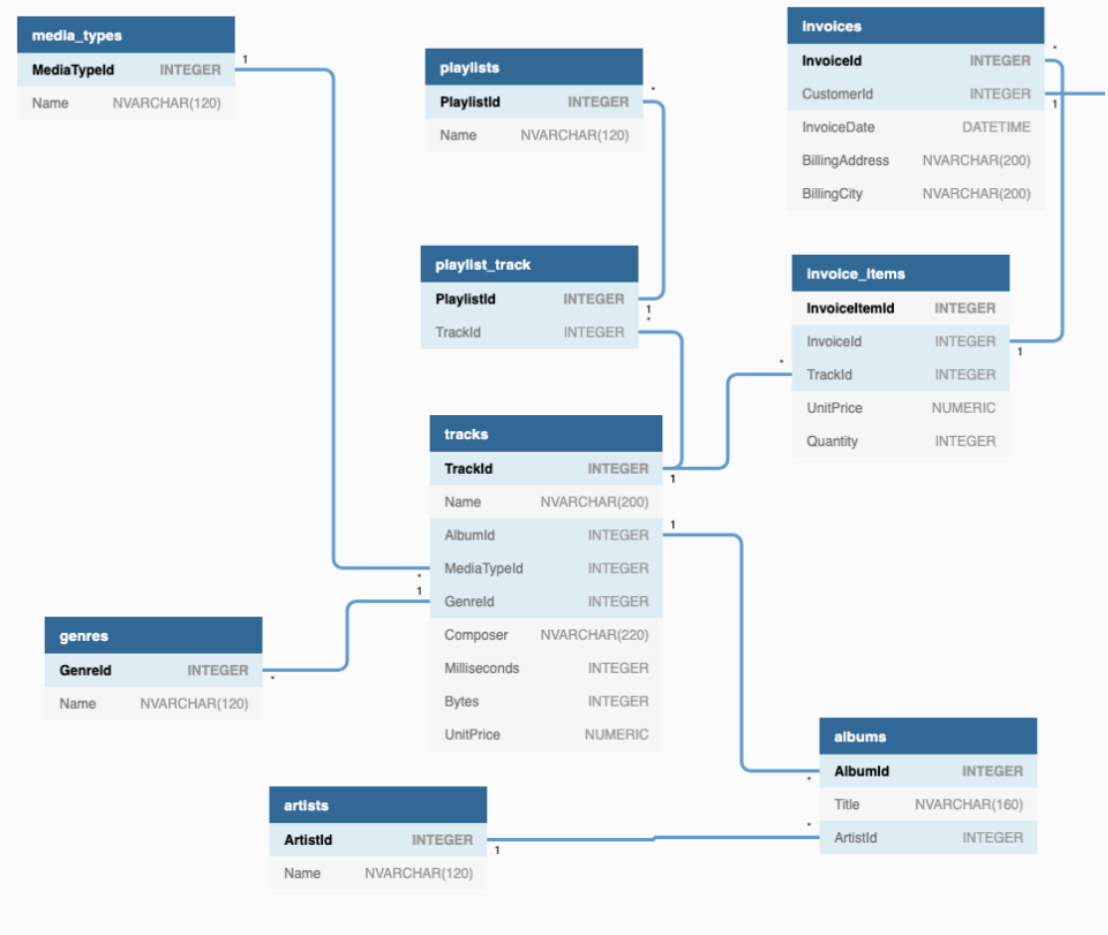
```
1 html_data[0]
```

	Population	1	2	3	4	5	6	7	8	9	10
0	Year	1804	1927	1960	1974	1987	1999	2011	2022	2037	2057
1	Years elapsed	200,000+	123	33	14	13	12	12	11	15	20

Способ 2. Подключение к базе данных SQL

Библиотека Pandas позволяет получать информацию напрямую из базы данных SQL. В качестве примера возьмем популярную учебную базу данных Chinook, в которой содержится информация о магазине, торгующем цифровыми записями (digital media store).

Ниже представлена схема этой **реляционной базы данных** (relational database schema).



> Тема реляционных баз данных и SQL выходит за рамки сегодняшнего занятия, однако для полноты изложения скажу, что такая база данных называется **реляционной** (от англ. relation — «отношение, зависимость»), потому что ее таблицы взаимосвязаны.

Например, в таблице *tracks* (записи) в поле GenreId содержится идентификатор жанра. По этому идентификатору мы можем определить, к какому жанру принадлежит та или иная композиция, заглянув в таблицу *genres*, где каждому GenreId соответствует название определенного жанра.

SQL же расшифровывается как structured query language (**язык структурированных запросов**) и позволяет достаточно гибко создавать, извлекать и изменять данные из реляционных баз данных.

Попробуем подключить базу данных к нашему блокноту и написать очень простой запрос на SQL внутри кода на Питоне. Вначале скачаем базу данных chinook (архив нужно распаковать и подгрузить файл в формате .db).

chinook.zip **Скачать**

```
# импортируем модуль sqlite3 для работы с базой данных SQL
import sqlite3 as sql

# создадим соединение с базой данных chinook
conn = sql.connect('/content/chinook.db')

# выберем все строки из таблицы tracks
sql_data = pd.read_sql('SELECT * FROM tracks', conn) # vs. read_sql_query

# посмотрим на результат
sql_data.head(3)
```

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
0	1	For Those About To Rock (We Salute You)	1	1	1	Angus Young, Malcolm Young, Brian Johnson	343719	11170334	0.99
1	2	Balls to the Wall	2	2	1	None	342562	5510424	0.99
2	3	Fast As a Shark	3	2	1	F. Baltes, S. Kaufman, U. Dirkscneider & W. Ho...	230619	3990994	0.99

Позднее в рамках курса мы поговорим еще про два формата: json и pickle.

Способ 3. Создание датафрейма из словаря

Кроме этого, и мы уже много раз так делали, библиотека Pandas позволяет создать датафрейм из словаря с помощью **функции pd.DataFrame()**. В этом случае ключи словаря будут названиями столбцов, а значения — элементами строк. Последние можно передать с помощью списков или массивов Numpy. Рассмотрим на примере.

Вначале подготовим данные.

```
# создадим несколько списков и массивов Numpy с информацией о семи странах мира
country = np.array(['China', 'Vietnam', 'United Kingdom', 'Russia', 'Argentina', 'Boliv
capital = ['Beijing', 'Hanoi', 'London', 'Moscow', 'Buenos Aires', 'Sucre', 'Pretoria']
population = [1400, 97, 67, 144, 45, 12, 59] # млн. человек
area = [9.6, 0.3, 0.2, 17.1, 2.8, 1.1, 1.2] # млн. кв. км.
sea = [1] * 5 + [0, 1] # выход к морю (в этом списке его нет только у Боливии)
```

Теперь создадим пустой словарь и поместим в него наши данные.

```
# создадим пустой словарь
countries_dict = {}

# превратим эти списки в значения словаря,
```

```
# одновременно снабдив необходимыми ключами
countries_dict['country'] = country
countries_dict['capital'] = capital
countries_dict['population'] = population
countries_dict['area'] = area
countries_dict['sea'] = sea
```

Наконец, создадим датафрейм с помощью функции **pd.DataFrame()**.

```
countries = pd.DataFrame(countries_dict)
countries
```

	country	capital	population	area	sea
0	China	Beijing	1400	9.6	1
1	Vietnam	Hanoi	97	0.3	1
2	United Kingdom	London	67	0.2	1
3	Russia	Moscow	144	17.1	1
4	Argentina	Buenos Aires	45	2.8	1
5	Bolivia	Sucre	12	1.1	0
6	South Africa	Pretoria	59	1.2	1

Способ 4. Создание датафрейма из 2D массива Numpy

Помимо этого, датафрейм можно создать из двумерного массива Numpy.

```
# внешнее измерение будет столбцами, внутренее - строками
arr = np.array([[1, 1, 1],
                [2, 2, 2],
                [3, 3, 3]])

pd.DataFrame(arr)
```

	0	1	2
0	1	1	1
1	2	2	2
2	3	3	3

Перейдем к структуре и свойствам датафрейма.

Структура и свойства датафрейма

Датафрейм библиотеки Pandas состоит из трех основных компонентов: строк (index), столбцов (columns) и значений (values).

Получить доступ к названиям столбцов можно с помощью **атрибута columns**.

```
# выведем столбцы датафрейма countries
countries.columns
```

```
Index(['country', 'capital', 'population', 'area', 'sea'], dtype='object')
```

Атрибут index описывает, каким образом идентифицируются строки.

```
# в нашем случае индекс это числовая последовательность
countries.index
```

```
RangeIndex(start=0, stop=7, step=1)
```

С помощью **атрибута values** мы получаем доступ к значениям датафрейма.

```
# значения выводятся в формате массива Numpy
countries.values
```

```
array([['China', 'Beijing', 1400, 9.6, 1],
       ['Vietnam', 'Hanoi', 97, 0.3, 1],
       ['United Kingdom', 'London', 67, 0.2, 1],
       ['Russia', 'Moscow', 144, 17.1, 1],
       ['Argentina', 'Buenos Aires', 45, 2.8, 1],
       ['Bolivia', 'Sucre', 12, 1.1, 0],
       ['South Africa', 'Pretoria', 59, 1.2, 1]], dtype=object)
```

Описание индекса и названия столбцов мы также можем получить с помощью **атрибута axes** (множественное от axis, ось) с индексами [0] и [1] соответственно.

```
# выведем описание индекса датафрейма через атрибут axes[0]
countries.axes[0]
```

```
RangeIndex(start=0, stop=7, step=1)
```

```
# axes[1] выводит названия столбцов
countries.axes[1]
```

```
Index(['country', 'capital', 'population', 'area', 'sea'], dtype='object')
```

> В целом, полезно запомнить, что axis = 0 позволяет применить какой-либо метод к строкам датафрейма, а axis = 1, соответственно к столбцам.

Мы также можем посмотреть количество измерений, размерность и общее количество элементов датафрейма с помощью **атрибутов ndim, shape и size** соответственно.

```
countries.ndim, countries.shape, countries.size
```

```
(2, (7, 5), 35)
```

Атрибут dtypes выдает тип данных каждого столбца.

```
countries.dtypes
```

```
country        object
capital        object
population      int64
area          float64
sea             int64
dtype: object
```

Кроме того, с помощью **метода .memory_usage()** мы можем посмотреть объем занимаемой памяти по столбцам в байтах.

```
countries.memory_usage()
```

```
Index         128
country        56
capital        56
population     56
```

```
area           56
sea            56
dtype: int64
```

Индекс датафрейма

По умолчанию при создании датафрейма библиотека Pandas снабжает его числовым индексом, который начинается с нуля. При этом мы можем присвоить датафрейму собственный индекс.

Присвоение индекса

Более того, этот индекс не обязательно должен быть числовым.

```
# создадим список с кодами стран
custom_index = ['CN', 'VN', 'GB', 'RU', 'AR', 'BO', 'ZA']

# создадим датафрейм из словаря и передадим список через параметр index
countries = pd.DataFrame(countries_dict,
                         index = custom_index)

# посмотрим на результат
countries
```

	country	capital	population	area	sea
CN	China	Beijing	1400	9.6	1
VN	Vietnam	Hanoi	97	0.3	1
GB	United Kingdom	London	67	0.2	1
RU	Russia	Moscow	144	17.1	1
AR	Argentina	Buenos Aires	45	2.8	1
BO	Bolivia	Sucre	12	1.1	0
ZA	South Africa	Pretoria	59	1.2	1

Этот индекс можно сбросить с помощью **метода .reset_index()**.

```
# при этом параметр inplace = True сделает изменения постоянными
countries.reset_index(inplace = True)
countries
```

	index	country	capital	population	area	sea
0	CN	China	Beijing	1400	9.6	1
1	VN	Vietnam	Hanoi	97	0.3	1
2	GB	United Kingdom	London	67	0.2	1
3	RU	Russia	Moscow	144	17.1	1
4	AR	Argentina	Buenos Aires	45	2.8	1
5	BO	Bolivia	Sucre	12	1.1	0
6	ZA	South Africa	Pretoria	59	1.2	1

Как мы видим, прошлый индекс стал отдельным столбцом. Снова сделаем этот столбец индексом с помощью **метода .set_index()**.

```
# передадим методу название столбца, который хотим сделать индексом
countries.set_index('index', inplace = True)
countries
```

index	country	capital	population	area	sea
CN	China	Beijing	1400	9.6	1
VN	Vietnam	Hanoi	97	0.3	1
GB	United Kingdom	London	67	0.2	1
RU	Russia	Moscow	144	17.1	1
AR	Argentina	Buenos Aires	45	2.8	1
BO	Bolivia	Sucre	12	1.1	0
ZA	South Africa	Pretoria	59	1.2	1

Еще раз сбросим индекс, но на этот раз не будем делать его отдельным столбцом. Для этого передадим методу **.reset_index()** параметр drop = True.

```
countries.reset_index(drop = True, inplace = True)
countries
```

	country	capital	population	area	sea
0	China	Beijing	1400	9.6	1
1	Vietnam	Hanoi	97	0.3	1
2	United Kingdom	London	67	0.2	1
3	Russia	Moscow	144	17.1	1
4	Argentina	Buenos Aires	45	2.8	1
5	Bolivia	Sucre	12	1.1	0
6	South Africa	Pretoria	59	1.2	1

Собственный индекс можно создать, просто поместив новые значения в атрибут **index**.

```
countries.index = custom_index
countries
```

	country	capital	population	area	sea
CN	China	Beijing	1400	9.6	1
VN	Vietnam	Hanoi	97	0.3	1
GB	United Kingdom	London	67	0.2	1
RU	Russia	Moscow	144	17.1	1
AR	Argentina	Buenos Aires	45	2.8	1
BO	Bolivia	Sucre	12	1.1	0
ZA	South Africa	Pretoria	59	1.2	1

Многоуровневый индекс

Многоуровневый (MultiIndex) или **иерархический** (hierarchical) индекс позволяет задать несколько уровней (levels) для индексации строк или столбцов. «Внешний» индекс

объединяет несколько элементов «внутреннего» индекса.

Вначале подготовим данные для многоуровневого индекса строк.

```
# создадим список из кортежей с названием континента и кодом страны
rows = [('Asia', 'CN'),
        ('Asia', 'VN'),
        ('Europe', 'GB'),
        ('Europe', 'RU'),
        ('S. America', 'AR'),
        ('S. America', 'BO'),
        ('Africa', 'ZA')]
```

Проделаем аналогичную работу для столбцов.

```
# в столбцах название страны и столицы мы объединим в категорию names
# а размер населения, площадь и выход к морю в data
cols = [('names', 'country'),
        ('names', 'capital'),
        ('data', 'population'),
        ('data', 'area'),
        ('data', 'sea')]
```

Теперь создадим иерархический индекс для строк и столбцов с помощью **функции pd.MultiIndex.from_tuples()**.

```
# создадим многоуровневый индекс для строк
# индексам присвоим названия через names = ['region', 'code']
custom_multindex = pd.MultiIndex.from_tuples(rows, names = ['region', 'code'])

# сделаем то же самое для столбцов
custom_multicols = pd.MultiIndex.from_tuples(cols)
```

Наконец остается передать эти индексы в атрибуты **index** и **columns** и вывести результат.

```
countries.index = custom_multindex
countries.columns = custom_multicols

countries
```

		names		data		
		country	capital	population	area	sea
region	code					
Asia	CN	China	Beijing	1400	9.6	1
	VN	Vietnam	Hanoi	97	0.3	1
Europe	GB	United Kingdom	London	67	0.2	1
	RU	Russia	Moscow	144	17.1	1
S. America	AR	Argentina	Buenos Aires	45	2.8	1
	BO	Bolivia	Sucre	12	1.1	0
Africa	ZA	South Africa	Pretoria	59	1.2	1

Как мы видим, страны (code) теперь разбиты на континенты (region), а информация о них — на текстовые (names) и числовые (data) данные.

```
# вернемся к обычному индексу и названиям столбцов
custom_cols = ['country', 'capital', 'population', 'area', 'sea']

countries.index = custom_index
```

```
countries.columns = custom_cols

countries
```

	country	capital	population	area	sea
CN	China	Beijing	1400	9.6	1
VN	Vietnam	Hanoi	97	0.3	1
GB	United Kingdom	London	67	0.2	1
RU	Russia	Moscow	144	17.1	1
AR	Argentina	Buenos Aires	45	2.8	1
BO	Bolivia	Sucre	12	1.1	0
ZA	South Africa	Pretoria	59	1.2	1

Преобразование в другие форматы

Получившийся датафрейм можно преобразовать в словарь с помощью **метода .to_dict()**.

```
print(countries.to_dict())
```

```
{'country': {'CN': 'China', 'VN': 'Vietnam', 'GB': 'United Kingdom', 'RU': 'Russia', 'A
```

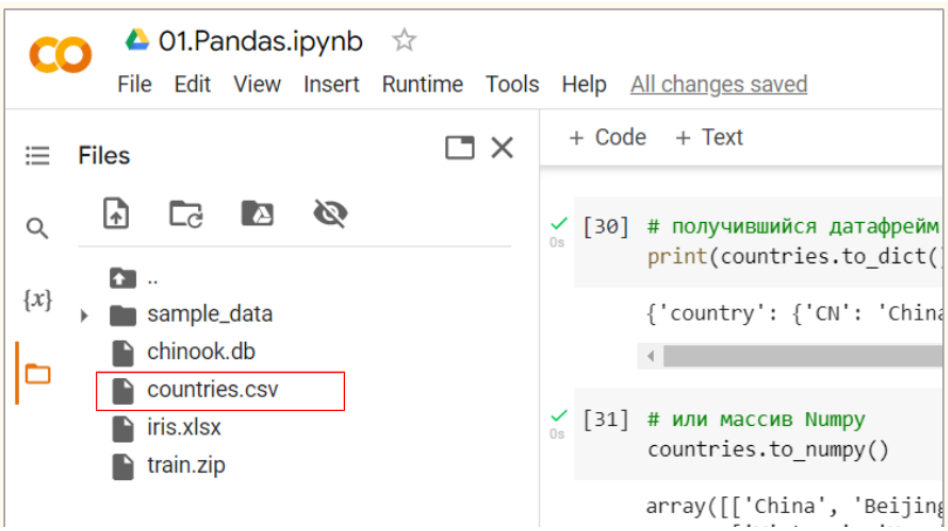
Аналогично, **метод .to_numpy()** преобразовывает данные в массив Numpy.

```
countries.to_numpy()
```

```
array([['China', 'Beijing', 1400, 9.6, 1],
       ['Vietnam', 'Hanoi', 97, 0.3, 1],
       ['United Kingdom', 'London', 67, 0.2, 1],
       ['Russia', 'Moscow', 144, 17.1, 1],
       ['Argentina', 'Buenos Aires', 45, 2.8, 1],
       ['Bolivia', 'Sucre', 12, 1.1, 0],
       ['South Africa', 'Pretoria', 59, 1.2, 1]], dtype=object)
```

Кроме того, мы можем сохранить датафрейм в файл (в Google Colab он появится в сессионном хранилище). Например, это может быть файл в формате .csv и тогда необходимо использовать **функцию .to_csv()**.

```
# по умолчанию, индекс также станет частью .csv файла
# параметр index = False позволит этого избежать
countries.to_csv('countries.csv', index = False)
```



Метод .to_list() позволяет преобразовать объект Series (столбец датафрейма) в список.

```
print(countries.country.to_list())
```

```
['China', 'Vietnam', 'United Kingdom', 'Russia', 'Argentina', 'Bolivia', 'South Africa'
```

Применить метод **.to_list()** к датафрейму не получится.

Создание Series

Создание Series из списка

Объект Series можно создать из списка. Возьмем следующий список с названиями стран.

```
country_list = ['China', 'South Africa', 'United Kingdom', 'Russia', 'Argentina', 'Viet
```

Передадим его в **функцию pd.Series()**.

```
country_Series = pd.Series(country_list)
country_Series
```

```
0             China
1      South Africa
2    United Kingdom
3            Russia
4         Argentina
5           Vietnam
6         Australia
dtype: object
```

Обратите внимание, что для объекта Series был автоматически создан числовой индекс. С его помощью мы можем получить доступ к элементам этого объекта.

```
# например, выведем первый элемент
country_Series[0]
```

```
'China'
```

Создание Series из словаря

Помимо списка, объект Series можно создать из словаря. Создадим словарь, в котором коды стран будут ключами, а их названия — значениями.

```
country_dict = {'CN' : 'China',
                'ZA' : 'South Africa',
                'GB' : 'United Kingdom',
                'RU' : 'Russia',
                'AR' : 'Argentina',
                'VN' : 'Vietnam',
                'AU' : 'Australia'
                }
```

Передадим этот словарь в функцию **pd.Series()**. Ключи станут индексами, а значения — элементами объекта.

```
country_Series = pd.Series(country_dict)
country_Series
```

```
CN             China
ZA      South Africa
GB    United Kingdom
RU            Russia
AR         Argentina
VN           Vietnam
AU         Australia
dtype: object
```

Доступ к элементам будет осуществляться теперь уже по коду отдельной страны.

```
country_Series['AU']
```

```
'Australia'
```

Доступ к строкам, столбцам и элементам

Поговорим про способы обращения к строкам, столбцам и элементам датафрейма.

Циклы в датафрейме

Цикл **for** позволяет получить доступ к названиям столбцов датафрейма.

```
for column in countries:
  print(column)
```

```
country
capital
population
area
sea
```

Метод .iterrows() возвращает индекс строки и ее содержимое в формате Series.

```
# прервем цикл после первой итерации с помощью break
for index, row in countries.iterrows():
  print(index)
  print(row)
  print('...')
  print(type(row))
  break
```

```
CN
country         China
capital       Beijing
population       1400
area              9.6
sea                 1
Name: CN, dtype: object
...
<class 'pandas.core.series.Series'>
```

Получить доступ к элементам одной строки можно по индексу объекта Series (то есть по

названиям столбцов исходного датафрейма).

```
for _, row in countries.iterrows():
  # например, сформируем вот такое предложение
  print(row['capital'] + ' is the capital of ' + row['country'])
  break
```

```
Beijing is the capital of China
```

Доступ к столбцам

В отличие от объекта Series в датафрейме через квадратные скобки мы получаем доступ к его столбцам.

```
# выведем столбец capital датафрейма countries
countries['capital']
```

```
CN         Beijing
VN           Hanoi
GB          London
RU          Moscow
AR    Buenos Aires
BO           Sucre
ZA        Pretoria
Name: capital, dtype: object
```

Доступ к столбцу можно также получить, указав название датафрейма и, через точку, название искомого столбца.

```
# однако в этом случае название не должно содержать пробелов
countries.capital
```

```
CN         Beijing
VN           Hanoi
GB          London
RU          Moscow
AR    Buenos Aires
BO           Sucre
ZA        Pretoria
Name: capital, dtype: object
```

Как уже было сказано, отдельные столбцы в датафрейме имеют тип данных Series.

```
type(countries.capital)
```

```
pandas.core.series.Series
```

Для того чтобы получить доступ к столбцам и при этом на выходе сформировать датафрейм, необходимо использовать двойные скобки.

```
# логика здесь в том, что внутрениие скобки - это список,
# внешние - оператор индексации
countries[['capital']]
```

	capital
CN	Beijing
VN	Hanoi
GB	London
RU	Moscow

AR	Buenos Aires
BO	Sucre
ZA	Pretoria

Так мы можем получить доступ к нескольким столбцам.

```
countries[['capital', 'area']]
```

	capital	area
CN	Beijing	9.6
VN	Hanoi	0.3
GB	London	0.2
RU	Moscow	17.1
AR	Buenos Aires	2.8
BO	Sucre	1.1
ZA	Pretoria	1.2

Доступ к столбцам можно также получить с помощью **метода .filter()**, передав параметру *items* список с необходимыми нам столбцами.

```
countries.filter(items = ['capital', 'population'])
```

	capital	population
CN	Beijing	1400
VN	Hanoi	97
GB	London	67
RU	Moscow	144
AR	Buenos Aires	45
BO	Sucre	12
ZA	Pretoria	59

Доступ к строкам

Получить доступ к строкам можно через срез индекса.

```
# выведем строки со второй по пятую (не включительно)
countries[1:5]
```

	country	capital	population	area	sea
VN	Vietnam	Hanoi	97	0.3	1
GB	United Kingdom	London	67	0.2	1
RU	Russia	Moscow	144	17.1	1
AR	Argentina	Buenos Aires	45	2.8	1

Методы .loc[] и .iloc[]

Метод .loc[]

Метод .loc[] позволяет получить доступ к строкам и столбцам через их названия (label-based location). Например, выведем первые три строки и первые три столбца датафрейма про страны.

```
# для этого передадим методу .loc[] два списка:
# с индексами строк и названиями столбцов
countries.loc[['CN', 'RU', 'VN'], ['capital', 'population', 'area']]
```

	capital	population	area
CN	Beijing	1400	9.6
RU	Moscow	144	17.1
VN	Hanoi	97	0.3

Через двоеточие, как и в Numpy, мы можем вывести все строки или все столбцы датафрейма.

```
# например, выведем все строки датафрейма
countries.loc[:, ['capital', 'population', 'area']]
```

	capital	population	area
CN	Beijing	1400	9.6
VN	Hanoi	97	0.3
GB	London	67	0.2
RU	Moscow	144	17.1
AR	Buenos Aires	45	2.8
BO	Sucre	12	1.1
ZA	Pretoria	59	1.2

Метод **.loc[]** также поддерживает значения Boolean.

```
# например, выведем все строки и только последний столбец,
# передав список соответствующих логических значений
countries.loc[:, [False, False, False, False, True]]
```

	sea
CN	1
VN	1
GB	1
RU	1
AR	1
BO	0
ZA	1

Метод .get_loc() позволяет узнать порядковый номер (начиная с нуля) строки или столбца по

их индексу и названию соответственно. Для того чтобы узнать порядковый номер строки, метод **.get_loc()** нужно использовать совместно с атрибутом **index**.

```
# выведем номер строки с индексом RU
countries.index.get_loc('RU')
```

```
'3'
```

Теперь выведем номер столбца country. Для этого нам понадобится атрибут **columns**.

```
countries.columns.get_loc('country')
```

```
'0'
```

Метод .iloc[]

Метод .iloc[] действует примерно также, как и **.loc[]**, с тем отличием, что он основывается на числовом индексе (integer-based location).

```
# теперь в списки мы передаем номера строк и столбцов,
# нумерация начинается с нуля
countries.iloc[[0, 3, 5], [0, 1, 2]]
```

	country	capital	population
CN	China	Beijing	1400
RU	Russia	Moscow	144
BO	Bolivia	Sucre	12

В методе **.iloc[]** можно использовать срезы.

```
# выведем первые три строки и последние два столбца
countries.iloc[:3, -2:]
```

	area	sea
CN	9.6	1
VN	0.3	1
GB	0.2	1

К столбцам удобно обращаться по их названиям, к строкам — по порядковому номеру (числовому индексу). Для того чтобы это реализовать, мы можем объединить двойные квадратные скобки и метод **.iloc[]**.

```
# вначале передадим названия столбцов в двойных скобках,
# затем номера строк через метод .iloc[]
countries[['population', 'area']].iloc[[0, 3]]
```

	population	area
CN	1400	9.6
RU	144	17.1

Многоуровневый индекс и методы .loc[] и .iloc[]

Поговорим про применение **методов .loc[]** и **.iloc[]** к датафрейму с многоуровневым индексом.

```
# вновь создадим датафрейм с многоуровневым индексом по строкам и столбцам
countries.index = custom_multindex
countries.columns = custom_multicols

countries
```

		names		data		
		country	capital	population	area	sea
region	code					
Asia	CN	China	Beijing	1400	9.6	1
	VN	Vietnam	Hanoi	97	0.3	1
Europe	GB	United Kingdom	London	67	0.2	1
	RU	Russia	Moscow	144	17.1	1
S. America	AR	Argentina	Buenos Aires	45	2.8	1
	BO	Bolivia	Sucre	12	1.1	0
Africa	ZA	South Africa	Pretoria	59	1.2	1

Для доступа к первой строке передадим **методу .loc[]** соответствующий двойной индекс.

```
countries.loc['Asia', 'CN']
```

```
names  country          China
       capital        Beijing
data   population        1400
       area               9.6
       sea                  1
Name: (Asia, CN), dtype: object
```

Мы также можем передать значения в форме кортежей как для строк, так и для столбцов.

```
# выведем первую строку и столбцы с числовыми данными
countries.loc[('Asia', 'CN'), [('data', 'population'), ('data', 'area'), ('data', 'sea'
```

```
data  population    1400.0
      area             9.6
      sea              1.0
Name: (Asia, CN), dtype: float64
```

Доступ к строкам можно получить, указав внутри кортежа название региона и список с кодами стран.

```
# например, выведем только азиатские страны
countries.loc[('Asia', ['CN', 'VN']), :]
```

		names		data		
		country	capital	population	area	sea
region	code					
Asia	CN	China	Beijing	1400	9.6	1
	VN	Vietnam	Hanoi	97	0.3	1

Внутри кортежа можно указать только регион. В этом случае мы получим все находящиеся в нем страны.

```
countries.loc[('Asia'), :]
```

	names		data		
	country	capital	population	area	sea
code					
CN	China	Beijing	1400	9.6	1
VN	Vietnam	Hanoi	97	0.3	1

Аналогичным образом мы можем получить доступ к столбцам.

```
countries.loc[:, [('names', 'country'), ('data', 'population')]]
```

		names	data
		country	population
region	code		
Asia	CN	China	1400
	VN	Vietnam	97
Europe	GB	United Kingdom	67
	RU	Russia	144
S. America	AR	Argentina	45
	BO	Bolivia	12
Africa	ZA	South Africa	59

Метод .iloc[], при этом, игнорирует структуру иерархического индекса и использует простой числовой индекс.

```
# получим доступ к четвертой строке и третьему, четвертому и пятому столбцам
countries.iloc[3, [2, 3, 4]]
```

```
data  population    144.0
      area           17.1
      sea             1.0
Name: (Europe, RU), dtype: float64
```

Метод .xs()

Метод .xs() (от англ. cross-section, срез) позволяет получить доступ к определенному уровню иерархического индекса. Начнем со строк.

```
# выберем Европу из уровня region
# axis = 0 указывает, что мы берем строки
countries.xs('Europe', level = 'region', axis = 0)
```

	names		data		
country	capital	population	area	sea	

code	country	capital	population	area	sea
GB	United Kingdom	London	67	0.2	1
RU	Russia	Moscow	144	17.1	1

Перейдем к столбцам и выберем names и country (по сути мы выбираем столбец country) на первом и втором уровнях соответственно.

```
# levels указывает, на каких уровнях искать названия столбцов
# параметр axis = 1 говорит о том, что мы имеем дело со столбцами
countries.xs(('names', 'country'), level = [0, 1], axis = 1)
```

		names
		country
region	code	
Asia	CN	China
	VN	Vietnam
Europe	GB	United Kingdom
	RU	Russia
S. America	AR	Argentina
	BO	Bolivia
Africa	ZA	South Africa

Кроме того, мы можем соединить два метода **.xs()** и одновременно получить доступ и к строкам, и к столбцам.

```
# в данном случае мы можем не указывать level, потому что
# Europe и names находятся во внешних индексах, которые в level указаны по умолчанию
countries.xs('Europe', axis = 0).xs(('names'), axis = 1)
```

	country	capital
code		
GB	United Kingdom	London
RU	Russia	Moscow

Вернем датафрейму одноуровневый индекс.

```
# обновим атрибуты index и columns
countries.index = custom_index
countries.columns = custom_cols

# посмотрим на исходный датафрейм
countries
```

	country	capital	population	area	sea
CN	China	Beijing	1400	9.6	1
VN	Vietnam	Hanoi	97	0.3	1
GB	United Kingdom	London	67	0.2	1

RU	Russia	Moscow	144	17.1	1
AR	Argentina	Buenos Aires	45	2.8	1
BO	Bolivia	Sucre	12	1.1	0
ZA	South Africa	Pretoria	59	1.2	1

Метод .at[]

Метод .at[] подходит для извлечения или записи *одного* значения датафрейма.

```
countries.at['CN', 'capital']
```

```
'Beijing'
```

Фильтры

Логическая маска

Как и в случае с массивом Numpy, мы можем создать логическую маску (Boolean mask) датафрейма, в которой нежелательные значения будут помечены как False.

```
# создадим логическую маску для стран с населением больше миллиарда человек
countries.population > 1000
```

```
CN     True
VN    False
GB    False
RU    False
AR    False
BO    False
ZA    False
Name: population, dtype: bool
```

Применив эту маску к исходному датафрейму, мы получим только те значения, которые помечены как True.

```
# применим логическую маску к исходному датафрейму
countries[countries.population > 1000]
```

	country	capital	population	area	sea
CN	China	Beijing	1400	9.6	1

Мы можем применить к датафрейму две маски, объединенные, например, логическим оператором И.

```
# отфильтруем датафрейм по критериям численности населения и площади
countries[(countries.population > 50) & (countries.area < 2)]
```

	country	capital	population	area	sea
VN	Vietnam	Hanoi	97	0.3	1
GB	United Kingdom	London	67	0.2	1
ZA	South Africa	Pretoria	59	1.2	1

Приведу еще один вариант синтаксиса. Его удобно использовать, если у вас есть множество условий, по которым вы хотите отфильтровать датафрейм.

```
# вначале создаем нужные нам маски
population_mask = countries.population > 70
area_mask = countries.population < 50

# затем объединяем их по необходимым условиям (в данном случае ИЛИ)
mask = population_mask | area_mask
# и применяем маску к исходному датафрейму
countries[mask]
```

	country	capital	population	area	sea
CN	China	Beijing	1400	9.6	1
VN	Vietnam	Hanoi	97	0.3	1
RU	Russia	Moscow	144	17.1	1
AR	Argentina	Buenos Aires	45	2.8	1
BO	Bolivia	Sucre	12	1.1	0

Метод .query()

Метод .query() позволяет задавать условие фильтрации «своими словами».

```
# например, выберем страны с населением более 50 млн. человек И
# площадью менее двух млн. кв. километров
countries.query('population > 50 and area < 2')
```

	country	capital	population	area	sea
VN	Vietnam	Hanoi	97	0.3	1
GB	United Kingdom	London	67	0.2	1
ZA	South Africa	Pretoria	59	1.2	1

При фильтрации по строковым значениям (которые записываются в кавычках), необходимо использовать различающиеся кавычки: двойные для метода **.query()** и одинарные для строкового значения или наоборот.

```
# выведем данные по Великобритании
countries.query("country == 'United Kingdom'")
```

	country	capital	population	area	sea
GB	United Kingdom	London	67	0.2	1

Другие способы фильтрации

С помощью **метода .isin()** мы можем проверить наличие нескольких значений в определенном столбце, а затем использовать результат в качестве логической маски.

```
# найдем строки, в которых в столбце capital присутствуют следующие значения
keyword_list = ['Beijing', 'Moscow', 'Hanoi']

countries[countries.capital.isin(keyword_list)]
```

	country	capital	population	area	sea
CN	China	Beijing	1400	9.6	1
VN	Vietnam	Hanoi	97	0.3	1
RU	Russia	Moscow	144	17.1	1

Похожим образом можно использовать **метод .startswith()**.

```
# например, для нахождения стран, НЕ начинающихся с буквы "A"
countries[ ~ countries.country.str.startswith('A')]
```

	country	capital	population	area	sea
CN	China	Beijing	1400	9.6	1
VN	Vietnam	Hanoi	97	0.3	1
GB	United Kingdom	London	67	0.2	1
RU	Russia	Moscow	144	17.1	1
BO	Bolivia	Sucre	12	1.1	0
ZA	South Africa	Pretoria	59	1.2	1

Метод .nlargest() позволяет найти строки с наибольшим значением в определенном столбце.

```
# например, возьмем три строки с наибольшими значением по столбцу population
countries.nlargest(3, 'population')
```

	country	capital	population	area	sea
CN	China	Beijing	1400	9.6	1
RU	Russia	Moscow	144	17.1	1
VN	Vietnam	Hanoi	97	0.3	1

Метод .nsmallest() аналогичным образом находит наименьшие значения.

Метод .argmax() выводит индекс строки, в которой в определенном столбце содержится наибольшее значение.

```
# например, предположим, что мы хотим найти индекс страны с наибольшей площадью
countries.area.argmax()
```

```
3
```

Посмотрим, какой стране соответствует этот индекс.

```
# напомню, что двойные скобки позволяют вывести DataFrame, а не Series
countries.iloc[[countries.area.argmax()]]
```

	country	capital	population	area	sea

RU	Russia	Moscow	144	17.1	1

Метод .argmin() выводит, соответственно, индекс строки, в которой содержится наименьшее для заданного столбца значение.

Вспомним, что в метод **.loc[]** можно передать тип данных Boolean. Используем это свойство для создания фильтра.

```
# выведем страны с населением более 90 млн. человек
countries.loc[countries.population > 90, :]
```

	country	capital	population	area	sea
CN	China	Beijing	1400	9.6	1
VN	Vietnam	Hanoi	97	0.3	1
RU	Russia	Moscow	144	17.1	1

Метод .filter(), если использовать параметр like, ищет совпадения с искомой фразой в индексе (если axis = 0) или названии столбцов (если axis = 1).

```
# найдем строки, в которых в индексе есть буквосочетание "ZA"
countries.filter(like = 'ZA', axis = 0)
```

	country	capital	population	area	sea
ZA	South Africa	Pretoria	59	1.2	1

Очевидно, если указать параметр axis = 1, выведется индекс, потому что именно в нем есть буквосочетание ZA.

Сортировка

Для сортировки значений в датафрейме библиотеки Pandas используется **метод .sort_values()**. Этому методу мы можем передать, среди прочих, следующие параметры:

- **by** — по какому столбцу или столбцам вести сортировку;
- **inplace** — сохранять ли результат;
- **ascending** — в восходящем или нисходящем порядке сортировать.

Рассмотрим два примера.

```
# выполним сортировку по столбцу population, не сохраняя изменений,
# в возрастающем порядке (значение по умолчанию)
countries.sort_values(by = 'population', inplace = False, ascending = True)
```

	country	capital	population	area	sea
BO	Bolivia	Sucre	12	1.1	0
AR	Argentina	Buenos Aires	45	2.8	1
ZA	South Africa	Pretoria	59	1.2	1

GB	United Kingdom	London	67	0.2	1
VN	Vietnam	Hanoi	97	0.3	1
RU	Russia	Moscow	144	17.1	1
CN	China	Beijing	1400	9.6	1

```
# теперь отсортируем по двум столбцам в нисходящем порядке
countries.sort_values(by = ['area', 'population'], inplace = False, ascending = False)
```

	country	capital	population	area	sea
RU	Russia	Moscow	144	17.1	1
CN	China	Beijing	1400	9.6	1
AR	Argentina	Buenos Aires	45	2.8	1
ZA	South Africa	Pretoria	59	1.2	1
BO	Bolivia	Sucre	12	1.1	0
VN	Vietnam	Hanoi	97	0.3	1
GB	United Kingdom	London	67	0.2	1

Кроме того, можно отсортировать строки по индексу с помощью **метода .sort_index()**.

```
countries.sort_index()
```

	country	capital	population	area	sea
AR	Argentina	Buenos Aires	45	2.8	1
BO	Bolivia	Sucre	12	1.1	0
CN	China	Beijing	1400	9.6	1
GB	United Kingdom	London	67	0.2	1
RU	Russia	Moscow	144	17.1	1
VN	Vietnam	Hanoi	97	0.3	1
ZA	South Africa	Pretoria	59	1.2	1

На этом завершим сегодняшнее занятие.

Подведем итог

На первом занятии по анализу и обработке данных мы рассмотрели, из каких этапов состоит решение задачи машинного обучения и начали углубленное изучение библиотеки Pandas.

В частности, мы узнали про типы объектов библиотеки Pandas (датафрейм и Series), способы их создания, структуру и свойства. Кроме того, мы рассмотрели возможности доступа к строкам, столбцам и элементам датафрейма.