KARPOV.COURSES >>>
КОНСПЕКТ

> Конспект > 1 урок > Знакомство с библиотекой Pandas

> Оглавление

> Импорт библиотек

В прошлых уроках мы долго учились тому, как создавать собственный код на Python и использовать его в своих целях. Однако нам необязательно использовать исключительно самостоятельно написанный код. Всё величие анализа данных в Python как раз в том, что мы можем использовать

наработки других пользователей, которые лежат в открытом доступе! Обычно такие наработки называются библиотеками.

Библиотека — это в каком-то смысле папка с файлами. Если конкретнее, это набор файлов с кодом (модулей), которые вы можете использовать в своих проектах. Python распознаёт такие "папки" через наличие в них специального файла `__init__.py`. И мы можем импортировать эти библиотеки.

Что означает в таком случае импорт? Это механизм, позволяющий вашей текущей сессии распознать все те объекты, которые определены в то, что мы импортируем. Проще говоря, это способ сказать Python: "Вот тут лежит код, я хочу его использовать в своей работе".

Импорт библиотеки/модуля выглядит так:

```
import library_name
```

Ключевое слово здесь — `import`. `library_name` — это название той библиотеки, которую мы импортируем. В базовой библиотеке Python также есть ряд модулей, которые импортируются схожим образом. Например `math`, который, как можно понять из названия, содержит в себе математические функции:

```
import math
math.log10(10)
```

Output:

```
1.0
```

Только что мы импортировали модуль `math`, взяли из него функцию `log10()` и с помощью этой функции посчитали десятичный логарифм от числа 10!

Обратите внимание: чтобы использовать функцию из импортированной библиотеки, нужно сначала написать её название, поставить точку и уже потом написать название нужной функции. Такова философия Python — это делается для того, чтобы было однозначно видно, откуда взята функция.

Алиасы

Подобное поведение может сильно раздражать, если название библиотеки слишком длинное. Чтобы уменьшить раздражение, можно присвоить библиотеке какой-нибудь псевдоним или алиас. С этим вам поможет ключевое слово `as`. Формат: import <название библиотеки> as <любой алиас>.

```
import numpy as np
np.random.random()

#эквивалентноimport numpy
numpy.random.random()

#как видите, первый вариант короче
```

Внимание: у многих популярных библиотек есть конвенциональные алиасы, знакомые всем. Запомните их, и понимать код станет гораздо проще.

Импорт отдельного модуля или функции

Почему мы вдруг написали `random` дважды? Дело в том, что внутри библиотеки `numpy` находится модуль `random`, посвящённый генерации случайных чисел. А уже он, в свою очередь, содержит в себе функцию `random()`, генерирующую случайные числа от 0 до 1. В нём есть и другие функции — например `randint()`, генерирующая целые числа в заданном интервале:

```
import numpy as np
np.random.randint(10) #выдаст случайное целое число от 0 до 10
```

Можно ли импортировать в таком случае отдельный модуль или даже отдельную функцию из этого модуля? Можно, но с использованием ключевого слова `from`. Формат: from <название библиотеки> import <название модуля>.

```
from numpy import random #импортировали только модуль random
random.randint(10) #теперь не нужно писать название оригинальной

from numpy.random import randint #импортировали только функцию
randint(10) #не нужно писать ни названия библиотеки, ни названия
```

Естественно, это можно сочетать с использованием алиасов:

```
from scipy import stats as ss #из библиотеки scipy взяли модуль
ss.pearsonr(range(10), range(0, 20, 2)) #подсчитали коэффициент
```

Можно импортировать несколько элементов за раз:

```
from scipy import stats, linalg #импортирует модули stats и lina

#каждому элементу в отдельности можно давать алиасы - а можно не

from scipy import stats as ss, linalg #stats будет под алиасом s
```

Возможен также следующий вариант:

from <название библиотеки> import *. Такой код импортирует всё содержимое библиотеки без необходимости к ней обращаться. **Это считается очень дурным стилем в сообществе питонистов — никогда так не делайте.**

Правила оформления импортов из PEP8:

Правильно:

```
import library_name1
import library_name2
```

Неправильно:

```
import library_name1, library_name2
```

Импорты должны быть сгруппированы в следующем порядке:

1. Импорты из стандартной библиотеки
2. Импорты сторонних библиотек
3. Импорты модулей текущего проекта

Важно: необходима пустая строка между каждой группой.

Внутри каждой группы импорты идут в следующем порядке:

```
import library_name
import library_name as alias
from library_name import module
```

Важно: некоторые разработчики рекомендуют импорты модулей упорядочивать в алфавитном порядке для повышения читабельности.

Установка библиотек

Наш сервер, на котором работает JupyterHub, уже содержит необходимые для прохождения курса библиотеки. Не нужно пытаться установить туда другие библиотеки или обновить существующие - это может привести к ошибке в работе сервера. Информация из этого блока понадобится вам, если вы будете работать на своем сервере

Прежде чем импортировать библиотеку, её необходимо установить. Для этого применяется менеджер пакетов `pip`, который **используется в командной строке**. Начиная с Python 3.4, `pip` устанавливается вместе с интерпретатором python.

Список основных команд:

- `pip install library_name` — установка библиотеки (устанавливает последнюю версию этой библиотеки).

  library_name

- `pip install --upgrade library_name` — обновление библиотеки .

  library_name

- `pip uninstall library_name` — удаляет библиотеку .

  library_name

- `pip list` — список всех установленных библиотек.
- `pip list --outdated` — список всех библиотек с установленными последними версиями.

Важно: В случае если в системе есть python2.7, а библиотека нужна для python3, необходимо использовать команду pip3.

> Numpy

Для математических вычислений и других важных функций была создана бесплатная библиотека `numpy`.

1. Используется для научных вычислений
2. В её основе — язык C
3. Базовый объект — многомерный массив (`numpy.ndarray`), аналог векторов, матриц и тензоров
4. Арифметические операции над объектами массива быстрее и удобнее, чем при использовании стандартного функционала Python

Документация: https://numpy.org/

Важно: если у вас есть рабочая задача с какой-либо библиотекой, и вы не можете её решить, то весьма вероятно, что это уже кто-то делал и решение

проблемы можно найти в документации или в интернете, если в документации библиотеки вы его не нашли.

```
import numpy as np #np - конвенциональный алиас этой библиотеки
```

Массивы довольно легко сделать из списков. Давайте создадим вот такую матрицу, с которой будем работать:

	Вес	Рост
Наблюдение 1	56	156
Наблюдение 2	70	180
Наблюдение 3	45	160

```
#создаём матрицу - она будет как список со списками
matrix = [[56, 156],
          [70, 180],
          [45, 160]]

#конвертируем
matrix = np.array(matrix)

#смотрим, что вышло
matrix
```

Output:

```
array([[ 56, 156],
       [ 70, 180],
       [ 45, 160]])
```

Мы даже можем дополнительно проверить, что это массив. Для этого используем функцию `type()`:

```
type(matrix)
```

Output:

```
numpy.ndarray
```

Так как это матрица, мы можем посмотреть, сколько в ней строк и столбцов. Для этого нам потребуется атрибут `shape` — он выдаст кортеж, в котором первое число — это строки, а второе — столбцы. Для массивов с большим числом измерений чисел тоже будет больше.

```
# matrix.shape[0] - только количество строк
# matrix.shape[1] - только количество столбцов

matrix.shape #три строки, два столбца
```

Output:

(3, 2)

> Pandas: введение

Как аналитику данных, вам придётся очень много работать с таблицами. Вам потребуется читать эти данные, видоизменять их форму и содержание, подсчитывать разные метрики... Для этого в Python была создана специальная библиотека под названием `pandas`.

1. Основная библиотека для работы с данными в Python
2. Построена поверх `numpy`
3. Обширный функционал для чтения данных разных типов (csv, xlsx и т.д.)
4. Удобный синтаксис фильтрации + SQL-подобные возможности

Документация: https://pandas.pydata.org/ В ней можно посмотреть существующие в библиотеке методы и функции с примерами применения, а также описание их параметров.

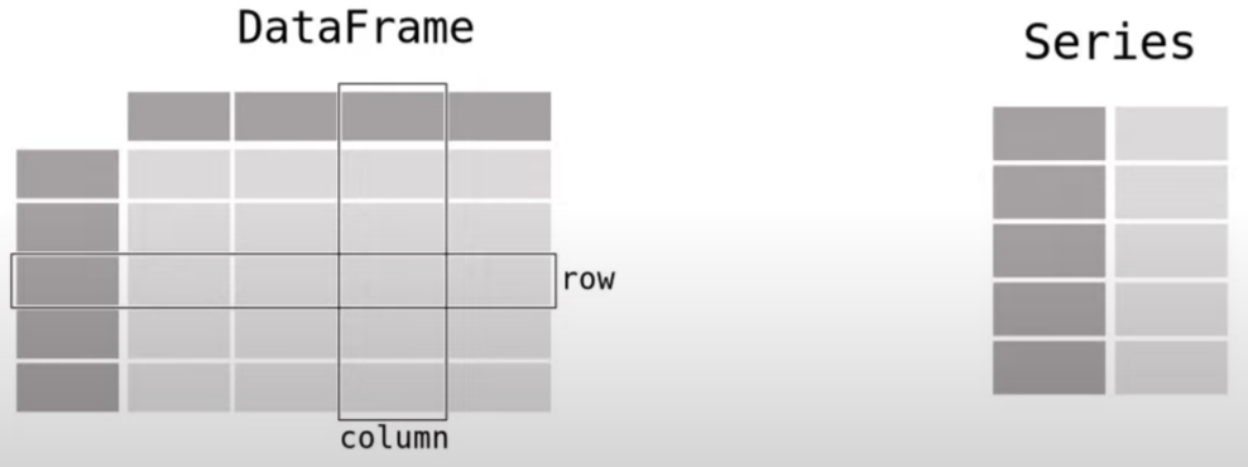
Важно: pandas в целом написан на смеси C и C++, т.е. эта библиотека весьма эффективно работает в памяти. Поэтому если ваш код на pandas работает медленно, то вполне возможно, что у вас есть какой-то неэффективный кусок кода, либо компьютеру не хватает мощностей.

```
import pandas as pd #pd - конвенциональный алиас этой библиотеки
```

Основные структуры:

- `pd.Series` — фактически `np.array`, но с именами и дополнительными особенностями
- `pd.DataFrame` (состоит из колонок, каждая из которых — `pd.Series`) — типичная табличка с данными, датафрейм

В них можно хранить самые разные типы данных — числа, строки, логические значения, даты и многое другое. При этом благодаря родству с `numpy` многие соответствующие этой библиотеке операции применимы и в `pandas`.



Как реализуется ввод и вывод структур данных, характерных для `pandas`?

1. Создать их вручную из других структур Python, например из словаря
2. Прочитать табулированные данные, вроде .csv и .xlsx, или какие-нибудь другие данные
3. Доступна и обратная операция — превратить датафрейм в .csv, .xlsx или другой тип данных

Первый взгляд

Для начала попробуем создать датафрейм самостоятельно!

```
# создадим датафрейм из словаря
df = pd.DataFrame(
    {
    "Name": ["Braund, Mr. Owen Harris",
             "Allen, Mr. William Henry",
             "Bonnell, Miss. Elizabeth"],
    "Age": [22, 35, 58],
    "Sex": ["male", "male", "female"]
    }
)

#отдельную pd.Series можно сделать так
ages = pd.Series([22, 35, 58], name="Age")
```

	Name	Age	Sex
0	Braund, Mr. Owen Harris	22	male
1	Allen, Mr. William Henry	35	male
2	Bonnell, Miss. Elizabeth	58	female

Как видите, у нас получилась вот такая аккуратная табличка. Три колонки с именами и с данными разных типов. Что мы можем с этим сделать?

1. Посмотреть на типы колонок:

```
df.dtypes

#две имеют тип object - так часто называются нечисловые типы дан
#чаще всего в этой категории находятся строковые данные

#одна имеет тип int32 - это целочисленные данные

#тип одной колонки можно глянуть так
#df.Age.dtype#это выведет тип одной переменной - Age
```

Output:

```
Name    object
Age      int64
Sex     object
dtype: object
```

2. Сделать сводную статистику по столбцам через метод `.describe()`:

```
df.describe()
```

	Age
count	3.000000
mean	38.333333
std	18.230012
min	22.000000
25%	28.500000
50%	35.000000
75%	46.500000
max	58.000000

Что всё это значит?

- count — количество значений, не считая пропуски
- mean — среднее значение по столбцу
- std — стандартное отклонение по столбцу
- min, max — минимальное и максимальное значения в столбце
- 25%, 50%, 75% — квартили распределения (числа, которые больше определённого процента значений в распределении). Например, 50%-й квартиль больше известен как медиана — число, которое больше 50% всех значений.

По умолчанию этот метод выведет информацию только для числовых колонок, если они есть в датафрейме. Если числовых нет - выведет для категориальных (для них набор показателей будет другим). Чтобы вывести информацию по всем колонкам, используйте параметр `include='all'`. Чтобы колонки с типом `datetime` воспринимались как числовые значения, а не категориальные, используйте параметр `datetime_is_numeric=True`

Документация

Отдельную pd.Series можно сделать так:

```
ages = pd.Series([22, 35, 58], name="Age")
ages
```

Output:

```
0    22
1    35
2    58
Name: Age, dtype: int64
```

Сравним размерность (количество строк и столбцов) датафрейма и серии:

```
df.shape, ages.shape
```

Output:

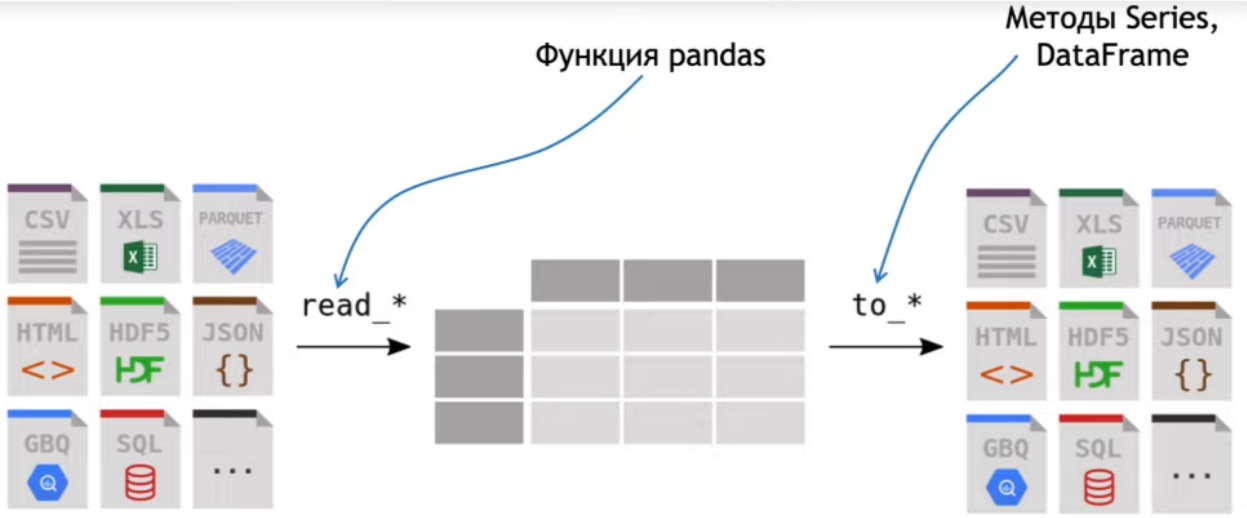
```
((3, 3), (3,))
```

Превратить серию в датафрейм можно с помощью метода to_frame()

```
df_ages = ages.to_frame()
df_ages
```

	Age
0	22
1	35
2	58

Чтение/запись реальных данных



Но с такими фейковыми данными работать неинтересно. Давайте заберёмся в реальный набор данных, а именно в данные пассажиров Титаника. Оригинальный набор данных можно достать вот тут.

Допустим, мы скачали этот набор данных. Как нам его прочитать? Так как набор данных имеет формат .csv, то нам поможет функция `pd.read_csv()`.

```
titanic = pd.read_csv("data/titanic.csv")
#в скобках должен быть полный путь к файлу в виде строки
#сойдёт и URL на открытый источник в интернете

#в Windows значок слэша повернут не в ту сторону: '\'
#не забудьте его повернуть, иначе не прочитается: '/'
```

У этой функции и подобных ей есть ряд дополнительных аргументов:

- `sep` — разделитель в файле, дефолт — запятые
- `header` — указание на заголовки, дефолт `header=0`. Можно передавать массив чисел (несколько уровней заголовков), если заголовков нет — `header=None`
- `names` — массив имён колонок, работает независимо от `header=None` или `header=0`
- `index_col` — номер колонки с индексом строк

- `usecols` — список колонок, которые нужно использовать
- `dtype` — словарь с явным указанием типов колонок
- `skiprows` — номера строк, которые нужно пропустить (можно функцией)
- `nrows` — количество строк, которое нужно прочитать
- `skip_blank_lines` — пропускать пустые строки, "да" по умолчанию
- `parse_dates` — `bool` или список колонок, распознаёт даты в наборе данных
- `thousands`, `decimal` — разделители разрядов
- `encoding` — кодировка в файле

Прочесть мы его прочли, а как сделать обратную операцию? Очень просто — через метод `.to_csv()`:

```
titanic.to_csv('second_titanic.csv', index=False)

#первым аргументом нужно указать полный путь до места, где сохра
#если написать только название, то сохранится в текущую рабочую
#не забудьте написать расширение файла!

#index=False - это чтобы индексы строк не сохранялись
#можете убрать этот аргумент и сравнить результат
```

Что внутри?

Попробуем изучить этот набор данных. Самый быстрый способ это сделать — с помощью метода `.info()`:

```
titanic.info()

#количество наблюдений и разброс индексов
#число столбцов
#индекс, названия столбцов, число ненулевых значений и тип данны
```

```
#общий подсчёт всех встречающихся типов данных
#место, которое занимает датафрейм
```

Output:

```
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Давайте посмотрим на само содержимое датафрейма! Конечно, выводить 801 наблюдение на экран — дело страшное. Но делать это совершенно не обязательно: с этим нам помогут методы `.head()` и `.tail()`:

```
titanic.head()
#выводит первые несколько строк сверху
```

	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

```
#если в скобках указать число, то столько строк он и выведет
titanic.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

```
titanic.tail()
#работает так же, как и .head(), но с другого конца
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	Q

Что ещё мы можем сделать?

1. `.columns` — вывести все названия столбцов

2. `.index` — вывести индексы датафрейма
3. `.to_numpy()` — превратить датафрейм в массив `numpy`

> Pandas: фильтрация и присвоение

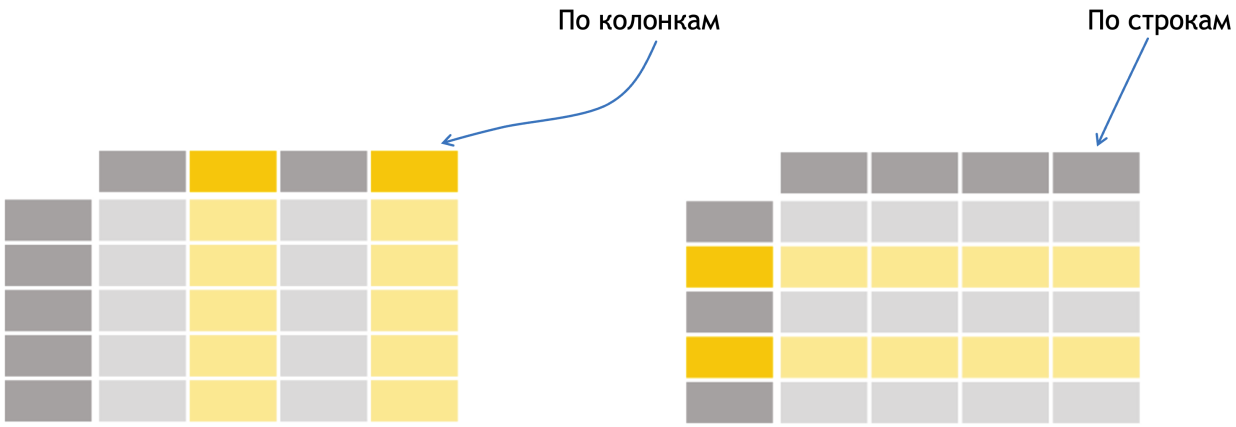
В большинстве случаев нам не нужен весь набор данных — нам достаточно лишь какого-то определённого кусочка. С этим нам поможет фильтрация данных.

Отбирать данные можно по именам. Собственно, это наиболее частый вариант фильтрации — взять какие-то столбцы, а остальные убрать. Например, возьмём столбцы Age и Sex из нашего датафрейма:

```
#подаём список с именами в квадратные скобки
titanic[["Age", "Sex"]]
```

	Age	Sex
0	22.0	male
1	38.0	female
2	26.0	female
3	35.0	female
4	35.0	male
...	...	...
886	27.0	male
887	19.0	female
888	NaN	female
889	26.0	male
890	32.0	male

Но фильтровать можно и иначе — как по столбцам, так и по строкам! Давайте посмотрим, как это делается.



Фильтрация по индексам

Как вы могли заметить, у каждой из строк есть свой порядковый номер, начиная с нуля. Это по сути и есть индексы — по этим порядковым номерам можно отбирать куски датафрейма. Стоит учесть, что индексы могут повторяться и вместо числовых индексов также можно ставить строковые.

Визуально индекс можно определить так:

1. Он всегда находится слева от столбцов
2. Его название (если оно есть) написано ниже, чем названия столбцов

Фильтрация работает не только для строк — столбцы также можно отбирать, используя индексы. С этим нам поможет индексатор `.iloc[]` (да, именно квадратные скобки). На первую позицию туда идут индексы строк, на вторую — индексы столбцов.

Попробуем взять первую и вторую строки нашего датафрейма:

```
#опять подаём в виде списка
titanic.iloc[[0, 1]]

#если мы хотим взять не строки, а столбцы, то на первой позиции
#такой код возьмёт первый и второй столбец
#titanic.iloc[:, [0, 1]]
```

	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

А если мы хотим взять строки/столбцы в каком-то диапазоне? Тогда нужно использовать `:`, как со списками:

```
#возьмём строки с 10 по 25
#и столбцы с 3 по 5

titanic.iloc[9:25, 2:5]
```

	Pclass	Name	Sex
9	2	Nasser, Mrs. Nicholas (Adele Achem)	female
10	3	Sandstrom, Miss. Marguerite Rut	female
11	1	Bonnell, Miss. Elizabeth	female
12	3	Saundercock, Mr. William Henry	male
13	3	Andersson, Mr. Anders Johan	male
14	3	Vestrom, Miss. Hulda Amanda Adolfina	female
15	2	Hewlett, Mrs. (Mary D Kingcome)	female
16	3	Rice, Master. Eugene	male
17	2	Williams, Mr. Charles Eugene	male
18	3	Vander Planke, Mrs. Julius (Emelia Maria Vande...	female
19	3	Masselmani, Mrs. Fatima	female
20	2	Fynney, Mr. Joseph J	male
21	2	Beesley, Mr. Lawrence	male
22	3	McGowan, Miss. Anna "Annie"	female
23	1	Sloper, Mr. William Thompson	male
24	3	Palsson, Miss. Torborg Danira	female

Фильтрация по условию

Это всё хорошо, но в большинстве случаев нам нужна фильтрация не по номерам. Обычно нам нужны строки, которые удовлетворяют какому-либо условию.

Для этого есть индексатор `.loc[]` — обратите внимание на отсутствие буквы i

Возьмём, например, только тех пассажиров, чей возраст больше 35 лет:

```
#titanic["Age"] > 35 - получится набор логических значений True
above_35 = titanic.loc[titanic["Age"] > 35] #строки, для которы

above_35.head()

#можно отфильтровать строки и выбрать столбец
```

```
#titanic.loc[titanic["Age"] > 35, "Name"]
#вернёт имена людей с возрастом больше 35
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S
13	14	0	3	Andersson, Mr. Anders Johan	male	39.0	1	5	347082	31.2750	NaN	S
15	16	1	2	Hewlett, Mrs. (Mary D Kingcome)	female	55.0	0	0	248706	16.0000	NaN	S

Если нам нужно взять только те строки, где **не** выполняется это условие, то перед условием нужно поставить оператор `~` (называется тильда, волнистая линия)

```
#titanic["Age"] > 35 - получится набор логических значений True
not_above_35 = titanic.loc[~titanic["Age"] > 35] #строки, для к

#вернёт имена людей с возрастом меньше или равно 35
```

А что если вариантов, по которым мы хотим отобрать, несколько? Тогда нам поможет метод `.isin()`:

```
#выберем пассажиров, которые плыли либо 2, либо 3 классом
#.isin() принимает на вход список с нужными нам значениями
class_23 = titanic.loc[titanic["Pclass"].isin([2, 3])]

#иначе можно записать череду условий через '|'
#обратите внимание на скобки
#class_23 = titanic.loc[(titanic["Pclass"] == 2) | (titanic["Pc

class_23.head()
```

	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
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	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
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S

Фильтрация пропущенных значений

Видите эти странные символы NaN? Так в `numpy` и `pandas` отображаются пропущенные значения — их настоящее имя `np.nan`.

Работа с NaN — это тема для отдельного обсуждения, но в большинстве случаев мы хотим избавиться от этих значений. Можем ли мы это сделать по условию через `.loc[]`?

Как ни странно, нет. По конвенции `np.nan != np.nan`, поэтому попытка их фильтрации через условие либо никак не изменит датафрейм, либо вернёт пустую таблицу.

Что же делать? Нам поможет метод `.notna()`, который возвращает False для всех NaN и True для всего остального. Есть и обратный метод — `.isna()`:

```
#возьмём только те строки, где нет пропущенного возраста
age_no_na = titanic.loc[titanic["Age"].notna()]
age_no_na.head()

#количество строк можно сравнить через .shape
```

	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

Присвоение значений

Вместо того чтобы отбирать куски данных, мы можем заменять эти куски на что-то другое:

```
#в третьей колонке с первую по третью строку будет слово "anonyr

titanic.iloc[0:3, 3] = "anonymous"
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	anonymous	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	anonymous	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	anonymous	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

```
#заменим у человека с таким именем возраст на 25 лет

titanic.loc[titanic.Name == 'Allen, Mr. William Henry', 'Age'] =
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	anonymous	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	anonymous	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	anonymous	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	25.0	0	0	373450	8.0500	NaN	S

Помимо этого, мы можем создавать новые колонки! Рассмотрим эту ситуацию на примере датасета по содержанию NO_2 в воздухе — airquality_no2:

datetime	station_antwerp	station_paris	station_london
2019-05-07 02:00:00	NaN	NaN	23.0
2019-05-07 03:00:00	50.5	25.0	19.0
2019-05-07 04:00:00	45.0	27.7	19.0
2019-05-07 05:00:00	NaN	50.4	16.0
2019-05-07 06:00:00	NaN	61.9	NaN

Сделаем пару новых колонок и присвоим их датафрейму:

```
#в квадратных скобках пишем имя новой колонки
#через оператор присвоения добавляем колонку нового содержания

#домножим одну из колонок на число
air_quality["london_mg_per_cubic"] = air_quality["station_londo

#поделим одну колонку на другую
air_quality["ratio_paris_antwerp"] = air_quality["station_paris
```

datetime	station_antwerp	station_paris	station_london	london_mg_per_cubic	ratio_paris_antwerp
2019-05-07 02:00:00	NaN	NaN	23.0	43.286	NaN
2019-05-07 03:00:00	50.5	25.0	19.0	35.758	0.495050
2019-05-07 04:00:00	45.0	27.7	19.0	35.758	0.615556
2019-05-07 05:00:00	NaN	50.4	16.0	30.112	NaN
2019-05-07 06:00:00	NaN	61.9	NaN	NaN	NaN

Наконец, мы можем переименовать интересующие нас столбцы с помощью `.rename()`:

```
#подаём в аргумент columns словарь
#ключ - старое название
#значение - новое название

air_quality_renamed = air_quality.rename(
    columns={"station_antwerp": "BETR801",
             "station_paris": "FR04014",
             "station_london": "London Westminster"})
```

datetime	BETR801	FR04014	London Westminster	london_mg_per_cubic	ratio_paris_antwerp
2019-05-07 02:00:00	NaN	NaN	23.0	43.286	NaN
2019-05-07 03:00:00	50.5	25.0	19.0	35.758	0.495050
2019-05-07 04:00:00	45.0	27.7	19.0	35.758	0.615556
2019-05-07 05:00:00	NaN	50.4	16.0	30.112	NaN
2019-05-07 06:00:00	NaN	61.9	NaN	NaN	NaN

> Pandas: агрегация и сортировка

Очень частая рабочая ситуация — необходимость что-то подсчитать по набору наших данных. Это можно делать по столбцам (и редко по строкам), но особенно часто возникает ситуация, когда подсчитать какое-то значение нужно по конкретным группам. Собственно, это и называется агрегацией.

Статистические методы `pandas` без группировки

Берём один или несколько столбцов и для каждого отдельно что-то считаем.

Когда мы с вами использовали метод `.describe()`, мы уже подсчитали много статистических индексов — это такой вариант "несколько за раз". Но можно делать это более точечно и для конкретных столбцов!
Например, `.mean()` считает арифметическое среднее:

```
#подсчитаем средний возраст
titanic["Age"].mean()
```

Output:

```
29.685112044817924
```

А функция `.median()`, как и следует из названия, медиану:

```
#подсчитаем медианный возраст и стоимость билета
titanic[["Age", "Fare"]].median()
```

Output:

```
Age     28.0000
Fare    14.4542
dtype: float64
```

Для подсчета суммы используется функция `.sum()` :

```
# если бы мы хотели посчитать сумму лет всех пассажиров
titanic["Age"].sum()
```

Output:

```
21195.17
```

Чтобы суммировать значения не в колонке, а в строке, используйте параметр `axis=1` и применяйте `sum()` ко всему датафрейму:

```
# если бы мы зачем-то хотели посчитать сумму значений во всех к
titanic.sum(axis=1)
```

Количество строк считает метод `.count()` Если значение в строке пустое, `count()` его не посчитает. Если нужно учесть и пустые строки тоже, есть метод `.size()` - но его стоит применять после группировки (`groupby`).

Особый случай — метод `.agg()`: он позволяет подсчитать любую статистическую метрику для столбца, если дать ему соответствующую функцию. Более того, можно не давать ему саму функцию, а написать ключевое слово: это работает для методов, встроенных в сам `pandas`. При этом никто не мешает рассчитать сразу несколько метрик:

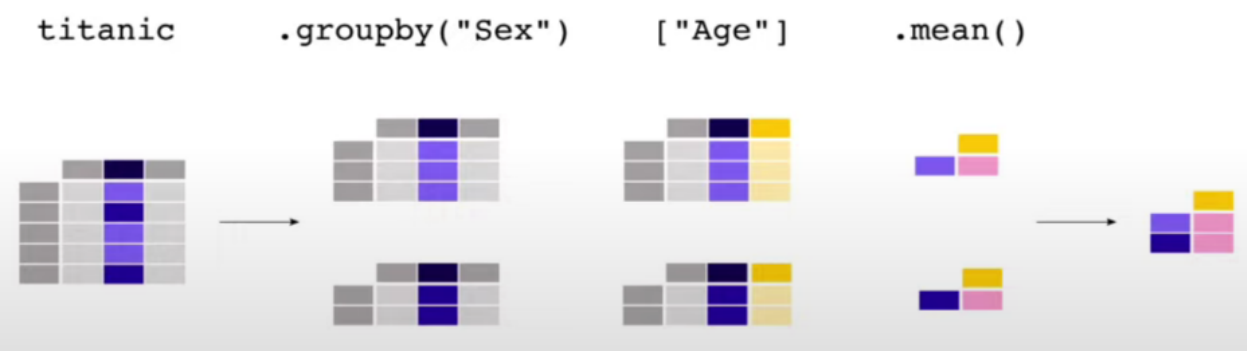
```
#на вход принимает словарь
#ключ - название столбца
#значение - функция или имя метода, встроенного в pandas
#можно подать сразу несколько через список

#подсчитаем минимум, максимум и медиану для обоих столбцов
#для возраста ещё подсчитаем коэффициент асимметрии
#для цены билета - среднее
```

```
titanic.agg({'Age': ['min', 'max', 'median', 'skew'],
             'Fare': ['min', 'max', 'median', 'mean']})
```

	Age	Fare
min	0.420000	0.000000
max	80.000000	512.329200
median	28.000000	14.454200
skew	0.391916	NaN
mean	NaN	32.204208

Статистические методы `pandas` с группировкой

Теперь каждой группе соответствует своё значение расчёта по каждому столбцу.



Здесь нам очень поможет метод `.groupby()` — в нём мы указываем тот столбец или те столбцы, по которым делается агрегация. Например, рассчитаем средний возраст для каждого пола:

```
#берём только пол и возраст
#группируем по полу
#считаем среднее по тем столбцам, которые не в .groupby()
#в данном случае это Age

titanic[["Sex", "Age"]].groupby("Sex").mean()
```

	Age
Sex	
female	27.915709
male	30.704570

Если бы мы не взяли только эти два столбца, то среднее по полу посчиталось бы по всем столбцам:

```
titanic.groupby("Sex").mean()
```

Sex	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
female	431.028662	0.742038	2.159236	27.915709	0.694268	0.649682	44.479818
male	454.147314	0.188908	2.389948	30.704570	0.429809	0.235702	25.523893

Можно группировать сразу по нескольким столбцам:

```
#рассчитаем среднее для каждого сочетания пола и пассажирского к
#оба названия надо подать внутри списка
#если поставить as_index=True, то группирующие переменные встану
#обратите внимание, что в прошлых двух случаях так оно и было

titanic.groupby(["Sex", "Pclass"], as_index=False)["Fare"].mean(
```

	Sex	Pclass	Fare
0	female	1	106.125798
1	female	2	21.970121
2	female	3	16.118810
3	male	1	67.226127
4	male	2	19.741782
5	male	3	12.661633

Отдельный вариант агрегации — подсчёт количества значений. Для такого был создан метод `.value_counts()`:

```
#подсчитаем, сколько раз встречается каждый пассажирский класс
```

```
titanic["Pclass"].value_counts()

#больше всего третьего, меньше всего второго
```

```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

Метод `value_counts` принимает на вход несколько параметров:

- `normalize` – показать относительные частоты уникальных значений (по умолчанию равен False).
- `dropna` – не включать количество NaN (по умолчанию равен True)
- `bins` – сгруппировать количественную переменную (например, разбить возраст на возрастные группы); для использования данного параметра нужно указать, на сколько групп разбить переменную

Несколько примеров:

1) Получаем частоту встречаемости (напр. Persik – в 40% наблюдений), также не удаляем из результата NaN:

```
df['name'].value_counts(normalize=True, dropna=False)
```

```
Persik    0.4
Tolya     0.2
Barsik    0.2
NaN       0.2
Name: name, dtype: float64
```

2) Разбиваем `year` на 2 промежутка:

```
df['year'].value_counts(bins=2)
```

```
(2017.5, 2020.0]      3
(2014.994, 2017.5]    2
Name: year, dtype: int64
```

Сортировка данных

Когда-нибудь доводилось сортировать в Excel значения по убыванию или возрастанию? Возможно, в алфавитном порядке? Вот это про то же самое, только не в Excel. Нам для этого обычно нужен всего один метод — `.sort_values()`.

Через аргумент `by` мы задаём, по каким столбцам идёт сортировка; если их несколько, то подаём их в виде списка:

```
#отсортируем пассажиров по возрасту
titanic.sort_values(by="Age").head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
803	804	1	3	Thomas, Master. Assad Alexander	male	0.42	0	1	2625	8.5167	NaN	C
755	756	1	2	Hamalainen, Master. Viljo	male	0.67	1	1	250649	14.5000	NaN	S
644	645	1	3	Baclini, Miss. Eugenie	female	0.75	2	1	2666	19.2583	NaN	C
469	470	1	3	Baclini, Miss. Helene Barbara	female	0.75	2	1	2666	19.2583	NaN	C
831	832	1	2	Richards, Master. George Sibley	male	0.83	1	1	29106	18.7500	NaN	S

По умолчанию сортировка идёт по возрастанию, но это можно поменять через аргумент `ascending=False`:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
851	852	0	3	Svensson, Mr. Johan	male	74.0	0	0	347060	7.7750	NaN	S
116	117	0	3	Connors, Mr. Patrick	male	70.5	0	0	370369	7.7500	NaN	Q
280	281	0	3	Duane, Mr. Frank	male	65.0	0	0	336439	7.7500	NaN	Q
483	484	1	3	Turkula, Mrs. (Hedwig)	female	63.0	0	0	4134	9.5875	NaN	S
326	327	0	3	Nysveen, Mr. Johan Hansen	male	61.0	0	0	345364	6.2375	NaN	S

При этом если мы сортируем по нескольким столбцам, то для каждого из них мы можем указать направление сортировки:

```
#возрастание по пассажирскому классу, убывание по возрасту
titanic.sort_values(by=['Pclass', 'Age'], ascending=[True, Fals
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
630	631	1	1	Barkworth, Mr. Algernon Henry Wilson	male	80.0	0	0	27042	30.0000	A23	S
96	97	0	1	Goldschmidt, Mr. George B	male	71.0	0	0	PC 17754	34.6542	A5	C
493	494	0	1	Artagaveytia, Mr. Ramon	male	71.0	0	0	PC 17609	49.5042	NaN	C
745	746	0	1	Crosby, Capt. Edward Gifford	male	70.0	1	1	WE/P 5735	71.0000	B22	S
54	55	0	1	Ostby, Mr. Engelhart Cornelius	male	65.0	0	1	113509	61.9792	B30	C

Иногда необходимо сортировать не по столбцам, а по индексам. Тут нам поможет метод `.sort_index()`. Попробуем это на примере airquality:

```
air_quality = pd.read_csv("data/air_quality_long.csv",
                          index_col="date.utc",
                          parse_dates=True)

#parse_dates у нас распознаёт даты и переводит их в нужный форма
#через index_col мы указываем, какой столбец мы хотим сделать ин
```

date.utc	city	country	location	parameter	value	unit
2019-06-18 06:00:00+00:00	Antwerpen	BE	BETR801	pm25	18.0	µg/m³
2019-06-17 08:00:00+00:00	Antwerpen	BE	BETR801	pm25	6.5	µg/m³
2019-06-17 07:00:00+00:00	Antwerpen	BE	BETR801	pm25	18.5	µg/m³
2019-06-17 06:00:00+00:00	Antwerpen	BE	BETR801	pm25	16.0	µg/m³
2019-06-17 05:00:00+00:00	Antwerpen	BE	BETR801	pm25	7.5	µg/m³

Отфильтруем только те значения, где измерялся NO_2, отсортируем по индексу и сгруппируем по месту измерения:

```
#оставляем только оксид азота
no2 = air_quality.loc[air_quality["parameter"] == "no2"]

#сортируем индекс, группируем по месту измерения, выводим по два
no2.sort_index().groupby(["location"]).head(2)
```

date.utc	city	country	location	parameter	value	unit
2019-04-09 01:00:00+00:00	Antwerpen	BE	BETR801	no2	22.5	µg/m³
2019-04-09 01:00:00+00:00	Paris	FR	FR04014	no2	24.4	µg/m³
2019-04-09 02:00:00+00:00	London	GB	London Westminster	no2	67.0	µg/m³
2019-04-09 02:00:00+00:00	Antwerpen	BE	BETR801	no2	53.5	µg/m³
2019-04-09 02:00:00+00:00	Paris	FR	FR04014	no2	27.4	µg/m³
2019-04-09 03:00:00+00:00	London	GB	London Westminster	no2	67.0	µg/m³

> Pandas: время, даты, строки

Мы уже успели немного посмотреть на типы данных, связанные с датами и временем. Заранее вас предупреждаем: с ними в реальной работе сплошные мучения. Именно поэтому стоит хотя бы на базовом уровне познакомиться с датами и временем в `pandas`.

Что стоит знать о датах и времени?

1. Если не определить даты однозначно как даты (например, через `parse_dates=True`), то `pandas` будет считать их типом `object`
2. Даты со временем в базовом Python выражаются через объект `datetime.datetime`, в `pandas` же объект называется `pd.Timestamp`. Они взаимозаменяемы, но не эквивалентны

3. Под такие данные в `numpy` есть специальный объект — он называется `np.datetime64`
4. А ещё в `numpy` есть объект `np.timedelta64`. Это уже не дата и не время - это различие во времени. Чаще всего такое получается, когда одну дату или время вычитают из другой.
5. `pd.DatetimeIndex` — массив `np.timedelta64`, может стать `pd.Timestamp`

Как ещё можно делать конвертацию в даты и время?

Для этого есть специальная функция `pd.to_datetime()`, которой можно скормить разные объекты, и она интерпретирует их как даты со временем.

Например, так можно конвертировать Unix-время в нормальное человеческое:

```
#unit задаёт единицу измерения - в данном случае это секунды
pd.to_datetime(1490195805, unit='s')

#а так - наносекунды
pd.to_datetime(1490195805433502912, unit='ns')
```

Output:

```
Timestamp('2017-03-22 15:16:45')

Timestamp('2017-03-22 15:16:45.433502912')
```

Можно отдать ей датафрейм, в которой разные столбцы соотносятся с разными элементами дат, и она это преобразует в нужный формат:

```
df = pd.DataFrame({'year': [2015, 2016],
                   'month': [2, 3],
                   'day': [4, 5]})
df
```

	year	month	day
0	2015	2	4
1	2016	3	5

```
pd.to_datetime(df)

#три столбца стали одним с датами
```

Output:

```
0   2015-02-04
1   2016-03-05
dtype: datetime64[ns]
```

Заметим, что даты могут записываться по-разному — с разной степенью подробности и разной позицией элементов (например, день-месяц-год или год-день-месяц). Обычно `pd.to_datetime()` старается это угадывать при аргументе `infer_datetime_format=True`. Однако порой лучше бывает мануально указать формат подаваемого для преобразования в дату и время объекта (обычно это строка или целое число) через аргумент `format`. Все возможные варианты элементов формата можно посмотреть вот тут.

Работа со строками

Обычно в `pandas` они читаются как тип данных `object`. Чтобы можно было применять на них строковые методы и функции, необходимо дописать аксессор `.str`. Подробнее о методах строк тут.

Попробуем поработать с игрушечными данными:

```
s = pd.DataFrame({'test_str':
                  ['A', 'B', 'C', 'Aaba', 'Baca', np.nan, 'CABA
```

	test_str
0	A
1	B
2	C
3	Aaba
4	Baca
5	NaN
6	CABA
7	dog
8	cat

Приведём все строки к нижнему регистру:

```
#берём столбец test_str
#используем аксессор .str
#и метод .upper()

s.test_str.str.lower()
```

Output:

```
0       a
1       b
2       c
3    aaba
4    baca
5     NaN
6    caba
```

```
7     dog
8     cat
Name: test_str, dtype: object
```

А теперь верхнему:

```
s.test_str.str.upper()
```

Output:

```
0       A
1       B
2       C
3    AABA
4    BACA
5     NaN
6    CABA
7     DOG
8     CAT
Name: test_str, dtype: object
```

И подсчитаем длину каждой строки:

```
s.test_str.str.len()
```

Output:

```
0    1.0
1    1.0
2    1.0
3    4.0
4    4.0
5    NaN
6    4.0
7    3.0
```

```
8    3.0
Name: test_str, dtype: float64
```

> Дополнительные материалы

- об индексах датафреймов
- работа с файлами больше чем оперативная память
- иерархия типов numpy
- библиотека для EDA в pandas
- сравнение эффективности операций в python и numpy