

Программирование на Python

АНАЛИЗ ДАННЫХ. ПРАКТИКА

Анализ данных – (не строго) процесс получения из данных выводов, на основе которых можно принимать решения.

Анализ данных – преобразование массивов данных в конкурентоспособные аналитические выводы, оказывающие влияние на деловые решения и последующие действия¹

Вид анализа	Цель	Основные понятия	Основной вопрос
Описательный (descriptive)	Количественное описание основных характеристик выборки.	Размер выборки, мода, медиана, среднее, размах, дисперсия, ...	Какие у нас данные?
Разведочный (exploratory, EDA)	Нахождение общих закономерностей, инсайтов, распределений, выбросов в данных.	Распределение, аномалия, тенденция, корреляция, гипотеза, ...	Какие есть закономерности в наших данных?
Индуктивный (inferential)	Оценка генеральной совокупности на основании выборки, выявление и оценка причинно-следственных связей между переменными.	Доверительный интервал, статистическая погрешность, математическое ожидание, ...	Что мы можем сказать о генеральной совокупности и насколько мы уверены в своем выводе?
Прогностический (predictive)	Предсказать поведение данных в будущем на основании их прошлых значений.	Прогноз, метрики качества, ...	Как будут вести себя данные в будущем?
Причинно-следственный (causal)	Объяснение с точки зрения данных причин возникновения события (следствия).	Причина, следствие, симптом, ...	Что является причиной такого поведения данных, а что следствием?

¹ Карл Андерсон. Аналитическая культура. М.: Манн, Иванов и Фербер, 2017.

Pandas – библиотека для обработки и анализа данных в Python.

```
import pandas as pd
```

Основные типы данных:

- **Series** – одномерный массив, `df = pd.Series()`
- **DataFrame** – таблица (двумерный массив), `df = pd.DataFrame()`

Примеры методов для DataFrame (для Series – аналогично):

- `df = pd.DataFrame(data, index=index, ...)` – создание датафрейма
- `df.shape` – размер датафрейма
- `df.info()` – получение информации о датафрейме
- `df.copy()` – копирование датафрейма
- `df = pd.read_csv(filename)` – создание датафрейма из csv файла
- `df.head(n), df.tail(n)` – возвращение n первых/последних строк
- `df.sort_values([columns])` – сортировка датафрейма
- `df.groupby([columns])[column].agg()` – группировка данных по columns
- `df.apply(function)` – применение функции к столбцам таблицы

```
df = pd.read_csv(filename)
df.head(5)
```

CPU times: total: 3.03 s

Wall time: 3.03 s



	Timestamp	FIT101	LIT101	MV101	P101	P102	AIT201	A
0	22/12/2015 4:30:00 PM	0.0	124.3135	1	1	1	251.9226	8.31
1	22/12/2015 4:30:01 PM	0.0	124.3920	1	1	1	251.9226	8.31
2	22/12/2015 4:30:02 PM	0.0	124.4705	1	1	1	251.9226	8.31
3	22/12/2015 4:30:03 PM	0.0	124.6668	1	1	1	251.9226	8.31
4	22/12/2015 4:30:04 PM	0.0	124.5098	1	1	1	251.9226	8.31

indexes

5 rows × 53 columns

```
df.info(verbose=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 495000 entries, 0 to 494999
Columns: 53 entries, Timestamp to Normal/Attack
dtypes: float64(25), int64(26), object(2)
memory usage: 200.2+ MB
```

```
df.describe().T
```

	count	mean	std	min	25%
FIT101	495000.0	1.850517	1.132519	0.000000	0.000000
LIT101	495000.0	587.532773	121.666482	120.623700	508.441000

- Из файла: `pd.read_<формат файла>`. Поддерживаются csv, excel-таблицы, URL-таблицы, pickle, json, html, sql, parquet и др. форматы:

```
df = pd.read_csv(<имя файла>, <параметры>)
```

- Из словаря: `df = pd.DataFrame(some_dict)`, `df = pd.DataFrame.from_dict(some_dict)`
- Из списка: `df = pd.DataFrame(some_list)`
- Пустой датафрейм: `df = pd.DataFrame()`
- Пустой датафрейм с индексами: `df = pd.DataFrame(index=some_list)`
- Пустой датафрейм с колонками: `df = pd.DataFrame(columns=some_list)`
- И другие способы...

С `Series` создание объекта идентично `DataFrame`, разница лишь в том, что `Series` – одномерный массив и данные на входе нужны соответствующие.

```
pd.Series(range(5),
          index=['id1', 'id2', 'id3', 'id4', 'id5'])
```

```
id1    0
id2    1
id3    2
id4    3
id5    4
dtype: int64
```

```
: pd.DataFrame({'column1': [11, 21, 31],
                 'column2': [12, 22, 32],
                 'column3': [13, 23, 33]})
```

	column1	column2	column3
0	11	12	13
1	21	22	23
2	31	32	33

```
: pd.DataFrame([[11, 12, 13],
                 [21, 22, 23],
                 [31, 32, 33]],
                 columns=['col1', 'col2', 'col3'])
```

	col1	col2	col3
0	11	12	13
1	21	22	23
2	31	32	33

Обращение к элементам датафрейма возможно:

- По целочисленному индексу: `df.iloc[<индекс строки>, <индекс колонки>]`
- По текстовой метке: `df.loc[<метка строки>, <метка колонки>]`
- Можно обращаться и без специального указания на тип запроса:

`df[<индекс строки>, <метка колонки>]`

Возможны манипуляции с отдельными строками или колонками:

```
df['column2']
```

```
row0    61
row1    17
row2    77
row3     5
row4    17
Name: column2, dtype: int64
```

```
df.loc['row1']
```

```
column0    64
column1    17
column2    17
column3     4
column4    71
Name: row1, dtype: int64
```

```
import random

dict1 = {
    f"column{i}": [random.randint(0,100) for i in range(5)] for i in range(5)
}

df = pd.DataFrame(dict1, index=[f"row{i}" for i in range(5)])
df
```

	0 column0	1 column1	2 column2	3 column3	4 column4
0 row0	1	61	61	19	0
1 row1	64	17	17	4	71
2 row2	43	35	77	70	11
3 row3	96	90	5	94	71
4 row4	26	94	17	57	59

```
df.iloc[1,1], df.loc['row1','column1']
```

```
(17, 17)
```

В Pandas реализована поддержка слайсов (как в списках или кортежах):

```
df.loc['row2':'row3', 'column1':'column3']
```

	column1	column2	column3
row2	35	77	70
row3	90	5	94

```
df[2:3]['column1']
```

```
row2    35
Name: column1, dtype: int64
```

Для выбора значений датафрейма, подходящих под условие, можно:

- Использовать **индексы**

```
df[(df['column1']>40) & (df['column3']<60)]
```

	column0	column1	column2	column3	column4
row0	1	61	61	19	0
row4	26	94	17	57	59

В этом случае при перечислении условий каждое условие заключается в (), а между скобками ставится операнд (&, |). Каждое условие возвращает на выходе Series с булевыми значениями, выражение внутри df[] должно иметь то же количество строк, что и фильтруемый датафрейм.

df['column3']<60	df['column1']>40
row0 True	row0 True
row1 True	row1 False
row2 False	row2 False
row3 False	row3 True
row4 True	row4 True

Name: column3, dtype: bool Name: column1, dtype: bool

- Использовать **query**

```
df.query("column1 > 40 and column3 < 60")
```

	column0	column1	column2	column3	column4
row0	1	61	61	19	0
row4	26	94	17	57	59

Исходный датафрейм

df	column0	column1	column2	column3	column4
row0	1	61	61	19	0
row1	64	17	17	4	71
row2	43	35	77	70	11
row3	96	90	5	94	71
row4	26	94	17	57	59

Для **изменения элемента** датафрейма следует обратиться к нему через **loc** или **iloc**:

```
df.loc['row1', 'column1'] = 10
df.iloc[1, 2] = 20
df.loc['row1', 'column1'], df.iloc[1, 2]
```

(10, 20)

Можно добавлять новую колонку/строку с нуля или на основании значений в другой колонке/строке:

Добавление колонки

```
df['column5'] = df['column4'] * 2
df['column6'] = range(df.shape[0])
df['column7'] = 99
df
```

	column0	column1	column2	column3	column4	column5	column6	column7
row0	0	10	20	30	40	80	0	99
row1	1	10	20	31	41	82	1	99
row2	2	12	22	32	42	84	2	99
row3	3	13	23	33	43	86	3	99
row4	4	14	24	34	44	88	4	99

Добавление строки

```
df.loc['row5'] = df.loc['row4'] + 10
df.loc['row6'] = range(df.shape[1])
df.loc['row6'] = 99
df
```

	column0	column1	column2	column3	column4	column5	column6	column7
row0	0	10	20	30	40	80	0	99
row1	1	10	20	31	41	82	1	99
row2	2	12	22	32	42	84	2	99
row3	3	13	23	33	43	86	3	99
row4	4	14	24	34	44	88	4	99
row5	14	24	34	44	54	98	14	109
row6	99	99	99	99	99	99	99	99

Исходный датафрейм

	column0	column1	column2	column3	column4
row0	0	10	20	30	40
row1	1	11	21	31	41
row2	2	12	22	32	42
row3	3	13	23	33	43
row4	4	14	24	34	44

Pandas. Удаление элементов, параметр axis

Для **удаления** строки или колонки можно воспользоваться методом `drop()`. Здесь и далее для методов DataFrame параметр `axis=1` подразумевает, что операция относится к **колонкам**, а `axis=0` – к **строкам**:

```
df.drop(['column1', 'column2'], axis=1)
df.drop(['row2', 'row3'])
```

	column0	column1	column2	column3	column4	column5	column6	column7
row0	0	10	20	30	40	80	0	99
row1	1	10	20	31	41	82	1	99
row4	4	14	24	34	44	88	4	99
row5	14	24	34	44	54	98	14	109
row6	99	99	99	99	99	99	99	99

Исходный датафрейм								
	column0	column1	column2	column3	column4	column5	column6	column7
row0	0	10	20	30	40	80	0	99
row1	1	10	20	31	41	82	1	99
row2	2	12	22	32	42	84	2	99
row3	3	13	23	33	43	86	3	99
row4	4	14	24	34	44	88	4	99
row5	14	24	34	44	54	98	14	109
row6	99	99	99	99	99	99	99	99

Для удаления несуществующих (NaN) значений следует использовать метод `dropna()`:

	column0	column1	column2	column3	column4	column5	column6	column7
row0	0.0	10	20	30	40	80.0	0.0	99.0
row1	1.0	10	20	31	41	82.0	NaN	99.0
row2	2.0	12	22	32	42	84.0	2.0	99.0
row3	3.0	13	23	33	43	86.0	3.0	99.0
row4	4.0	14	24	34	44	88.0	4.0	NaN
row5	14.0	24	34	44	54	98.0	NaN	109.0
row6	NaN	99	99	99	99	NaN	99.0	99.0



```
df.dropna()
```

	column0	column1	column2	column3	column4	column5	column6	column7
row0	0.0	10	20	30	40	80.0	0.0	99.0
row2	2.0	12	22	32	42	84.0	2.0	99.0
row3	3.0	13	23	33	43	86.0	3.0	99.0

В зависимости от установленных параметров метод `dropna` может удалять строки (`axis=0`) или колонки (`axis=1`) если все (`how='all'`) или хотя бы одно (`how='any'`) значение будет NaN. Также можно искать NaN только в конкретных колонках (`subset=[...]`)

Для объединения датафреймов (или Series и датафрейма) можно применять функцию

`pd.concat([df1, df2, ...], axis=...):`

The diagram illustrates two dataframes, `df_A` and `df_B`, and the operations `pd.concat()` and `df.merge()`.

df_A

	column0	column1	column2
row0	A0	A10	A20
row1	A1	A11	A21
row2	A2	A12	A22

df_B

	column0	column1	column2	column3
row0	B0	B10	B20	B30
row1	B1	B11	B21	B31
row2	B2	B12	B22	B32
row3	B3	B13	B23	B33

Arrows indicate the operations:

- `pd.concat()` is associated with `df_A`.
- `df.merge()` is associated with `df_B`.

pd.concat()

df.merge()

`pd.concat([df_A, df_B])`

	column0	column1	column2	column3
row0	A0	A10	A20	NaN
row1	A1	A11	A21	NaN
row2	A2	A12	A22	NaN
row0	B0	B10	B20	B30
row1	B1	B11	B21	B31
row2	B2	B12	B22	B32
row3	B3	B13	B23	B33

`pd.concat([df_A, df_B], axis=1)`

	column0	column1	column2	column0	column1	column2	column3
row0	A0	A10	A20	B0	B10	B20	B30
row1	A1	A11	A21	B1	B11	B21	B31
row2	A2	A12	A22	B2	B12	B22	B32
row3	NaN	NaN	NaN	B3	B13	B23	B33

`df_A.merge(df_B, left_index=True, right_index=True)`

	column0_x	column1_x	column2_x	column0_y	column1_y	column2_y	column3
row0	A0	A10	A20	B0	B10	B20	B30
row1	A1	A11	A21	B1	B11	B21	B31
row2	A2	A12	A22	B2	B12	B22	B32

Для объединения по заданным условиям лучше применять функцию `df1.merge(df2,...)`.

`merge` позволяет выбирать условия объединения (по индексам, по колонкам), тип объединения (`inner`, `outer`, `right`, `left`, ...) и другие опции.

Pandas. Изменение датафреймов. Функция apply()

Функция `df.apply` позволяет применить произвольную функцию к отдельным строкам (`axis=0`)/колонкам (`axis=1`) датафрейма.

```
df[['column1', 'column2']].apply(lambda x: x%2)
```

	column1	column2
row0	0	0
row1	0	0
row2	0	0
row3	1	1
row4	0	0
row5	0	0
row6	1	1

Исходный датафрейм

	column0	column1	column2	column3	column4	column5	column6	column7
row0	0	10	20	30	40	80	0	99
row1	1	10	20	31	41	82	1	99
row2	2	12	22	32	42	84	2	99
row3	3	13	23	33	43	86	3	99
row4	4	14	24	34	44	88	4	99
row5	14	24	34	44	54	98	14	109
row6	99	99	99	99	99	99	99	99

Для применения функции ко всему датафрейму можно применить функцию `df.applymap()`

К колонкам можно применять собственные функции, аргументы передаются через параметр `args=`:

```
def some_function(element_df, a):
    return element_df - a

df.apply(some_function, args=(10,))
```

	column0	column1	column2	column3	column4	column5	column6	column7
row0	-10.0	0	10	20	30	70.0	-10.0	89.0
row1	-9.0	0	10	21	31	72.0	NaN	89.0
row2	-8.0	2	12	22	32	74.0	-8.0	89.0

Для сортировки по определенной колонке следует применять функцию `df.sort_values(<колонка>)`

```
df.sort_values('column6', ascending=False)
```

	column0	column1	column2	column3	column4	column5	column6	column7
row6	NaN	99	99	99	99	NaN	99.0	99.0
row4	4.0	14	24	34	44	88.0	4.0	NaN
row3	3.0	13	23	33	43	86.0	3.0	99.0
row2	2.0	12	22	32	42	84.0	2.0	99.0
row0	0.0	10	20	30	40	80.0	0.0	99.0
row1	1.0	10	20	31	41	82.0	NaN	99.0
row5	14.0	24	34	44	54	98.0	NaN	109.0

Исходный датафрейм

	column0	column1	column2	column3	column4	column5	column6
row0	1	61	61	19	0	0	0
row1	64	10	20	4	71	142	1
row2	43	35	77	70	11	22	2
row3	96	90	5	94	71	142	3
row4	26	94	17	57	59	118	4
row5	36	104	27	67	69	128	14
row6	0	1	2	3	4	5	6

Для сортировки по индексу следует применять функцию `df.sort_index()`

```
df.sort_index(ascending=False)
```

	column0	column1	column2	column3	column4	column5	column6	column7
row6	NaN	99	99	99	99	NaN	99.0	99.0
row5	14.0	24	34	44	54	98.0	NaN	109.0
row4	4.0	14	24	34	44	88.0	4.0	NaN
row3	3.0	13	23	33	43	86.0	3.0	99.0
row2	2.0	12	22	32	42	84.0	2.0	99.0
row1	1.0	10	20	31	41	82.0	NaN	99.0
row0	0.0	10	20	30	40	80.0	0.0	99.0

Pandas позволяет получить множество статистик из датафрейма и отлично подходит для описательной аналитики.

Для примера мы возьмем датасет с информацией о пассажирах Титаника.

```
df.head(5)
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

Сводная статистика

```
df.describe() # описание всего датасета
```

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Статистика по колонкам/строкам

```
df['age'].mean(), df['age'].min(), df['age'].max()
```

```
(29.69911764705882, 0.42, 80.0)
```

```
df['pclass'].unique()
```

```
array([3, 1, 2], dtype=int64)
```

```
df['pclass'].nunique()
```

```
3
```

```
df['pclass'].value_counts()
```

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

Для группировки данных в pandas используется функция `groupby()`:

`df.groupby([<колонки для агрегации>])[<колонки для подсчета>].<агрегирующая функция>()`

Стандартные агрегирующие функции: 'sum', 'mean', 'median', 'min', 'max', 'std', 'var', 'mad', 'prod'.

Можно применять к разным колонкам разные агрегирующие функции.

Можно написать собственную агрегирующую функцию.

```
df.groupby('pclass')[['fare', 'age']].mean()
```

	fare	age
pclass		
1	84.154687	38.233441
2	20.662183	29.877630
3	13.675550	25.140620

```
agg_func = {'fare': ['describe']}
df.groupby(['pclass']).agg(agg_func).round(2)
```

		count	mean	std	min	25%	50%	75%	max
pclass	fare								
1	describe	216.0	84.15	78.38	0.0	30.92	60.29	93.5	512.33
2		184.0	20.66	13.42	0.0	13.00	14.25	26.0	73.50
3		491.0	13.68	11.78	0.0	7.75	8.05	15.5	69.55

Исходный датафрейм										
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who
0	0	3	male	22.0	1	0	7.2500	S	Third	man
1	1	1	female	38.0	1	0	71.2833	C	First	woman
2	1	3	female	26.0	0	0	7.9250	S	Third	woman
3	1	1	female	35.0	1	0	53.1000	S	First	woman
4	0	3	male	35.0	0	0	8.0500	S	Third	man

```
(df
 .groupby(['pclass', 'who'])[['fare', 'age']]
 .agg(['min', 'mean', 'max'])
 .round(2)
 )
```

			fare			age		
			min	mean	max	min	mean	max
pclass	who							
1	child		81.86	139.38	211.34	0.92	7.82	15.0
	man		0.00	65.95	512.33	17.00	42.38	80.0
	woman		25.93	104.32	512.33	16.00	35.50	63.0
2	child		14.50	28.32	41.58	0.67	4.54	14.0
	man		0.00	19.05	73.50	16.00	33.59	70.0
	woman		10.50	20.87	65.00	17.00	32.18	57.0
3	child		7.22	23.22	46.90	0.42	6.82	15.0
	man		0.00	11.34	69.55	16.00	29.00	74.0
	woman		6.75	15.35	69.55	16.00	27.85	63.0

В Pandas существуют специальные форматы для времени и даты – `Timestamp`, `Timedelta` и `datetime64[ns]`. Преобразование к ним выполняется через функцию `pd.to_datetime(<строка>, <формат>)` или через `pd.to_timedelta(<строка>)`, если речь идет о разнице во времени. После преобразования в нужный формат становятся доступными операции с этими значениями как со временем (например, вычитание или ресэмплирование), а также получение дополнительных данных через специальный форматтер `dt`, например дня, даты или часа.

Исходный датафрейм									
	Timestamp	FIT101	LIT101	MV101	P101	AIT201	AIT202	AIT203	FIT201
0	2015-12-22 04:30:00	0.0	124.3135	1	1	251.9226	8.313446	312.7916	0.000000
1	2015-12-22 04:30:01	0.0	124.3920	1	1	251.9226	8.313446	312.7916	0.000000
2	2015-12-22 04:30:02	0.0	124.4705	1	1	251.9226	8.313446	312.7916	0.000000
3	2015-12-22 04:30:03	0.0	124.6668	1	1	251.9226	8.313446	312.7916	0.000000
4	2015-12-22 04:30:04	0.0	124.5098	1	1	251.9226	8.313446	312.7916	0.000000

Дата

```
df1['Timestamp'].dt.date
```

0	2015-12-22
1	2015-12-22
2	2015-12-22
3	2015-12-22
4	2015-12-22
...	...
494995	2015-12-28
494996	2015-12-28
494997	2015-12-28
494998	2015-12-28
494999	2015-12-28

Name: Timestamp, Length: 495000,

День

```
df1['Timestamp'].dt.day
```

0	22
1	22
2	22
3	22
4	22
...	...
494995	28
494996	28
494997	28
494998	28
494999	28

Name: Timestamp, Length: 495000,

Разница по времени

```
df1['Timestamp'].diff()
```

0	NaT
1	0 days 00:00:01
2	0 days 00:00:01
3	0 days 00:00:01
4	0 days 00:00:01
...	...
494995	0 days 00:00:01
494996	0 days 00:00:01
494997	0 days 00:00:01
494998	0 days 00:00:01
494999	0 days 00:00:01

Name: Timestamp, Length: 495000,

Увеличение даты на 1 день

```
df1['Timestamp'] + pd.to_timedelta('1D')
```

0	2015-12-23 04:30:00
1	2015-12-23 04:30:01
2	2015-12-23 04:30:02
3	2015-12-23 04:30:03
4	2015-12-23 04:30:04
...	...
494995	2015-12-29 12:59:55
494996	2015-12-29 12:59:56
494997	2015-12-29 12:59:57
494998	2015-12-29 12:59:58
494999	2015-12-29 12:59:59

Name: Timestamp, Length: 495000, dtype:

В Pandas существует специальный форматтер для удобной работы со строками: **str**.

При его использовании становятся доступными все основные функции работы со строками.

contains (содержит значение)

```
df1[df1['sex'].str.contains('fem')]
```

	sex	class	who	embark_town
1	female	First	woman	Cherbourg
2	female	Third	woman	Southampton
3	female	First	woman	Southampton
8	female	Third	woman	Southampton
9	female	Second	child	Cherbourg

upper

(преобразование регистров)

```
df1['sex'].str.upper()
```

```
0      MALE
1     FEMALE
2     FEMALE
3     FEMALE
4      MALE
...
```

count

(подсчет символов)

```
df1['sex'].str.count('e')
```

```
0      1
1      2
2      2
3      2
4      1
```

Исходный датафрейм

	sex	class	who	embark_town
0	male	Third	man	Southampton
1	female	First	woman	Cherbourg
2	female	Third	woman	Southampton
3	female	First	woman	Southampton
4	male	Third	man	Southampton

Можно выполнять быстрое преобразование из одного текста в другой:

```
df1['new_feature'] = df['sex'].str.replace('e', '*') + "_" + df['class'].str.lower()
df1
```

	sex	class	who	embark_town	pclass	new_feature
0	male	Third	man	Southampton	3	mal*_third
1	female	First	woman	Cherbourg	1	f*mal*_first
2	female	Third	woman	Southampton	3	f*mal*_third
3	female	First	woman	Southampton	1	f*mal*_first
4	male	Third	man	Southampton	3	mal*_third

Сохранять данные в файл в pandas очень просто: `df.to_<нужный формат>(<имя файла>, <опции>)`.

Поддерживаются все те же форматы, что и при загрузке данных: csv, excel, json, parquet, pickle и т.п.

```
df.to_csv(filename, index=False)
```

```
%%time
df = pd.read_csv(filename)
df.head(5)
```

CPU times: total: 2.31 s

Wall time: 2.33 s

	Timestamp	FIT101	LIT101	MV101	P101	P102	AIT201	AIT202
0	22/12/2015 4:30:00 PM	0.0	124.3135	1	1	1	251.9226	8.313446
1	22/12/2015 4:30:01 PM	0.0	124.3920	1	1	1	251.9226	8.313446
2	22/12/2015 4:30:02 PM	0.0	124.4705	1	1	1	251.9226	8.313446
3	22/12/2015 4:30:03 PM	0.0	124.6668	1	1	1	251.9226	8.313446
4	22/12/2015 4:30:04 PM	0.0	124.5098	1	1	1	251.9226	8.313446



SciPy – библиотека для научных и инженерных расчетов в Python.

```
from scipy import <имя пакета>
```

Основные пакеты библиотеки:

- *cluster* – кластерный анализ
- *constants* – различные константы (физические и математические)
- *fftpack* – быстрое преобразование Фурье
- *integrate* – интегральные уравнения и диффуры
- *interpolate* – интерполяция и сглаживание
- *io* – ввод/вывод
- *linalg* – линейная алгебра
- *ndimage* – обработка изображений
- *odr* – метод ортогональных расстояний
- *optimize* – оптимизация и численное решение уравнений
- *signal* – обработка сигналов
- *sparse* – разреженные матрицы
- *spatial* – разреженные структуры данных и алгоритмы
- *special* – специальные функции
- *stats* – статистические распределения и функции

Тригонометрические и экспоненциальные функции

```
from scipy import special
a = special.exp10(3)
print(a)
b = special.exp2(3)
print(b)
c = special.sindg(90)
print(c)
d = special.cosdg(45)
print(f"{d:.4f}")
```

```
1000.0
8.0
1.0
0.7071
```

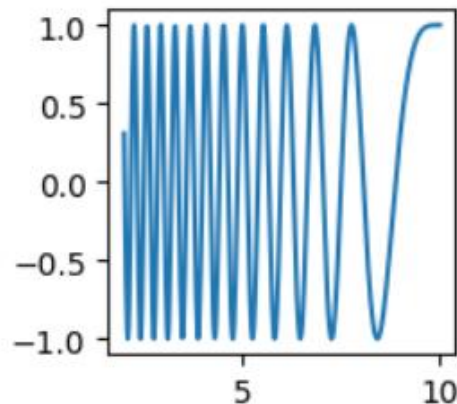
Операции с матрицами

```
from scipy import linalg
A = np.array([[10,11], [21,30]])
print(linalg.det(A)) # определитель
linalg.inv(A) # обратная матрица
```

```
69.0
array([[ 0.43478261, -0.15942029],
       [-0.30434783,  0.14492754]])
```

Частотно-модулированный сигнал

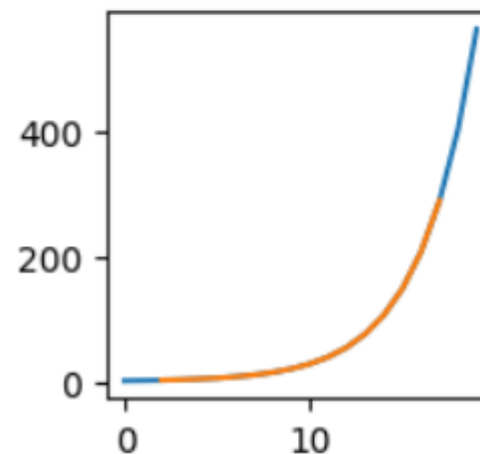
```
from scipy.signal import chirp
t = np.linspace(2, 10, 500)
w = chirp(t, f0=4, f1=2, t1=5, method='linear')
plt.figure(figsize=(2,2))
plt.plot(t, w)
plt.show()
```



Интерполяция

```
import matplotlib.pyplot as plt
from scipy import interpolate

x = np.arange(0, 20)
y = np.exp(x/3.0)
f = interpolate.interp1d(x, y)
x1 = np.arange(2, 18)
y1 = f(x1)
plt.figure(figsize=(2,2))
plt.plot(x, y, x1, y1)
plt.show()
```



NumPy – библиотека для работы с многомерными массивами и матрицами.

```
import numpy as np
```

Основной тип данных: **numpy.ndarray**

Примеры методов работы с массивом:

- `arr = np.array([1,2],[3,4])` – создание двумерного массива
- `arr.ndim` – число измерений массива
- `arr.shape` – размеры массива
- `arr.size` – количество элементов в массиве

NumPy позволяет автоматически создавать различные массивы.

```
import numpy as np

a = np.array([[1,2], [3,4]])
print(type(a))
print(a)
print('ndim: \t', a.ndim)
print('shape: \t', a.shape)
print('size: \t', a.size)
```

```
<class 'numpy.ndarray'>
[[1 2]
 [3 4]]
ndim:      2
shape:     (2, 2)
size:      4
```

Матрица с нулевыми эл-тами

```
np.zeros((5, 4))
```

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

Матрица с единичными эл-тами

```
np.ones((5, 4))
```

```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

Единичная матрица

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

```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

Многомерный массив

```
np.ones((2,2,2))
```

```
array([[[1., 1.],
        [1., 1.]],
       [[1., 1.],
        [1., 1.]])
```

Также можно воспользоваться функцией `np.empty()` для создания пустого массива, заполненного остатками данных в памяти

Генерация последовательностей при помощи `np.arange()`

(возможно использование `np.random.random()`, `np.linspace()`):

```
arr = np.arange(1, 10)
arr
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Изменение размерности массива при помощи `arr.reshape()`:

```
arr = arr.reshape(3,3)
arr
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Приведение массива к плоскому виду при помощи `arr.flat`:

```
np.array(arr.flat)
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Доступ к элементам массива осуществляется

по индексам, поддерживаются слайсы.

```
print("1 элемент:", arr[1,2])
print("Слайс:\n", arr[:2, :2])
print("Другой слайс:\n", arr[:2, :])
```

1 элемент: 6

Слайс:

```
[[1 2]
```

```
[4 5]]
```

Другой слайс:

```
[[1 2 3]
```

```
[4 5 6]]
```

Для массивов поддерживаются все стандартные операции:

сложение, вычитание, умножение и деление, скалярное

произведение.

Сложение и вычитание

```
arr1 + 10
```

```
array([[11, 12, 13],
       [14, 15, 16],
       [17, 18, 19]])
```

```
arr1 - 10
```

```
array([[ -9, -8, -7],
       [-6, -5, -4],
       [-3, -2, -1]])
```

```
arr1 + arr2[:, :3]
```

```
array([[ 1,  3,  5],
       [ 8, 10, 12],
       [15, 17, 19]])
```

Умножение и деление

```
arr1 / 10
```

```
array([[0.1, 0.2, 0.3],
       [0.4, 0.5, 0.6],
       [0.7, 0.8, 0.9]])
```

```
arr2 * 2
```

```
array([[ 0,  2,  4,  6],
       [ 8, 10, 12, 14],
       [16, 18, 20, 22]])
```

Скалярное произведение

```
arr1.dot(arr2)
```

```
array([[ 32,  38,  44,  50],
       [ 68,  83,  98, 113],
       [104, 128, 152, 176]])
```

Исходный массив

```
arr1 = np.arange(1,10).reshape(3,3)
arr2 = np.arange(0,12).reshape(3,4)
print(arr1)
print(arr2)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

Другие преобразования

```
np.cos(arr1)
```

```
array([[ 0.54030231, -0.41614684, -0.9899925 ],
       [-0.65364362,  0.28366219,  0.96017029],
       [ 0.75390225, -0.14550003, -0.91113026]])
```

Для массивов поддерживается стандартный набор статистик:

- максимальный (`np.max`) и минимальный (`np.min`) элементы
- дисперсия (`np.var`) и стандартное отклонение (`np.std`)
- среднее значение (`np.mean`) и медиана (`np.median`)
- суммы (`np.sum`) и т.п.

Операции могут производиться как над всем массивом, так и отдельно по его осям (`axis=0` или `axis=1`)

```
arr2
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
np.sum(arr2)
```

```
66
```

```
np.sum(arr2, axis=0)
```

```
array([12, 15, 18, 21])
```

```
np.sum(arr2, axis=1)
```

```
array([ 6, 22, 38])
```

```
np.mean(arr2)
```

```
5.5
```

```
np.var(arr2) # дисперсия
```

```
11.916666666666666
```

```
np.std(arr2) # ст.отклонение
```

```
3.452052529534663
```

Для **объединения** массивов существуют методы:

- `np.hstack()` – объединение по горизонтальной оси
- `np.vstack()` – объединение по вертикальной оси
- `np.column_stack()` – объединение одномерных массивов как столбцов
- `np.row_stack()` – объединение одномерных массивов как строк

Для **разбиения** массивов аналогично:

- `np.hsplit()` – разбиение вдоль горизонтальной оси
- `np.vsplit()` – разбиение вдоль вертикальной оси

```
arr1
```

```
array([[1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9]])
```

```
arr2
```

```
array([[ 0,  1,  2,  3],  
       [ 4,  5,  6,  7],  
       [ 8,  9, 10, 11]])
```

```
np.hstack((arr1, arr2))
```

```
array([[ 1,  2,  3,  0,  1,  2,  3],  
       [ 4,  5,  6,  4,  5,  6,  7],  
       [ 7,  8,  9,  8,  9, 10, 11]])
```

```
np.vstack((arr1, arr2[:3]))
```

```
array([[ 1,  2,  3],  
       [ 4,  5,  6],  
       [ 7,  8,  9],  
       [ 0,  1,  2],  
       [ 4,  5,  6],  
       [ 8,  9, 10]])
```

```
np.hsplit(arr2, 2)
```

```
[array([[0, 1],  
       [4, 5],  
       [8, 9]]),  
 array([[ 2,  3],  
       [ 6,  7],  
       [10, 11]])]
```

Matplotlib – пакет для визуализации данных в Python, **pyplot** – модуль matplotlib, который предоставляет интерфейс к созданным при помощи matplotlib объектам.

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

Основные методы:

- `plt.plot(x, y)` – построить линейный график по точкам из списков `x` и `y`
- `plt.show()` – вывод визуализированных данных
- `plt.xlabel('Ось x')`, `plt.ylabel('Ось y')` – подписи к осям графика
- `plt.title('Название графика')` – название графика
- `plt.legend()` – добавление к графику легенды

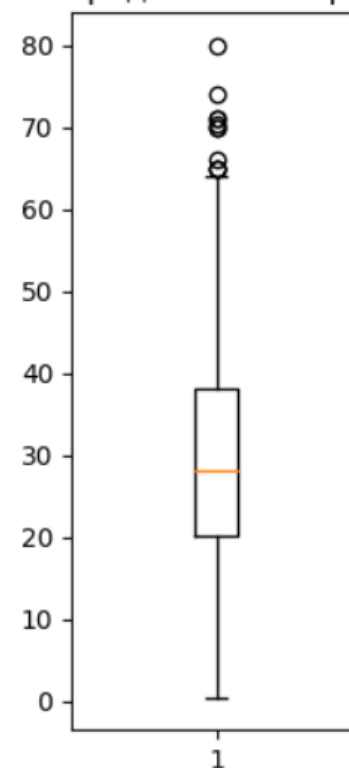
Кроме основного метода `plot`, который отображает данные в виде линейного графика, доступны:

- `plt.scatter(x, y)` – диаграмма рассеяния
- `plt.bar(x, y)` – столбчатая диаграмма
- `plt.pie(values)` – круговая диаграмма
- `plt.boxplot(x)` – диаграмма распределения
- И множество других...

Диаграммы обычно можно комбинировать на одном графике.

```
x = np.array(df['age'].dropna())
plt.figure(figsize=(2,5))
plt.boxplot(x)
plt.title('Распределение возраста')
plt.show()
```

Распределение возраста



Для демонстрации работы matplotlib будем использовать датасет от Титаника.

```
size = 200
age = np.array(df['age'][:size])
fare = np.array(df['fare'][:size])

plt.figure(figsize=(20,5)) # Изменение размера области построения графика
plt.title('Пример названия графика') # изменение заголовка
plt.plot(range(size),age, label='age', color='green') # линейный график
plt.scatter(range(size), fare, label='fare', color='blue') # точечный график
plt.xlabel('Номер пассажира') # подпись оси X
plt.ylabel('возраст/стоимость билета') # подпись оси Y
plt.legend() # отобразить легенду
plt.grid() # отобразить сетку
plt.show() # скрыть служебные сообщения
```

df.head(5)

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True



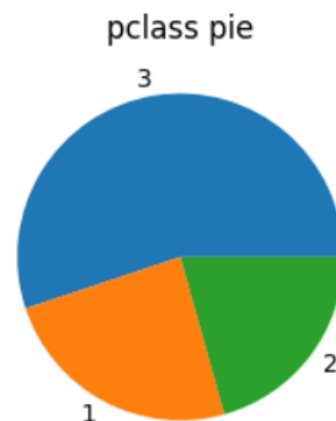
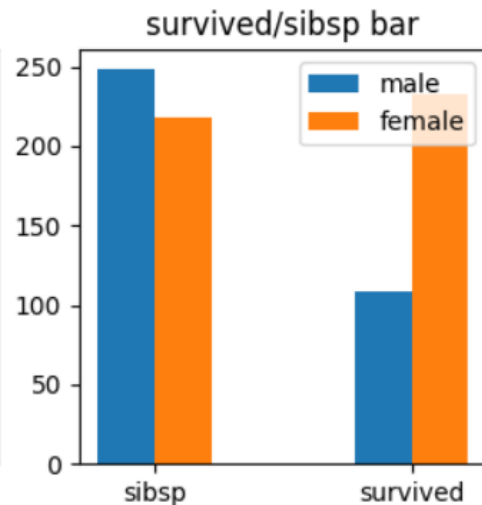
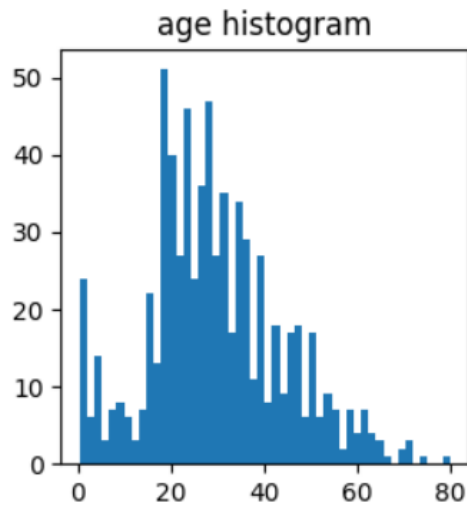
```
plt.figure(figsize=(10,3))

plt.subplot(1, 3, 1) # (кол-во строк, кол-во столбцов, ячейка)
plt.hist(df['age'], bins=50)
plt.title('age histogram')

plt.subplot(1, 3, 2)
sex = df.groupby('sex')[['sibsp', 'survived']].sum()
plt.bar([0.1, 1], sex.loc['male'], 0.2, label='male')
plt.bar([0.3, 1.2], sex.loc['female'], 0.2, label='female')
plt.xticks([0.2, 1.1], ['sibsp', 'survived'])
plt.title('survived/sibsp bar')
plt.legend()

plt.subplot(1, 3, 3)
pclass = df['pclass'].value_counts()
plt.pie(pclass.values, labels=pclass.index)
plt.title('pclass pie')

plt.show()
```



Для отображения независимых графиков в одном окне можно воспользоваться методом:

`plt.subplot(<кол-во строк>, <кол-во столбцов>, <ячейка в которой будет график>)`

Объекты Pandas (`DataFrame`, `Series`) имеют встроенные методы для рисования графика (на базе matplotlib): `plot()`, `hist()`, `boxplot()`.

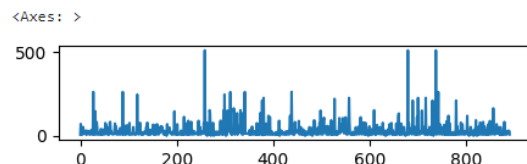
Метод `plot()` в свою очередь позволять построить различные виды графиков (через параметр `kind=`):

- `'area'` – график с накоплением
- `'bar'` – вертикальная гистограмма
- `'barh'` – горизонтальная гистограмма
- `'box'` – столбчатая диаграмма
- `'hexbin'` – шестнадцатеричный график
- `'hist'` – гистограмма
- `'kde'` – оценка плотности ядра
- `'density'` = `'kde'`
- `'line'` – линейный график
- `'pie'` – круговая диаграмма
- `'scatter'` – график рассеяния

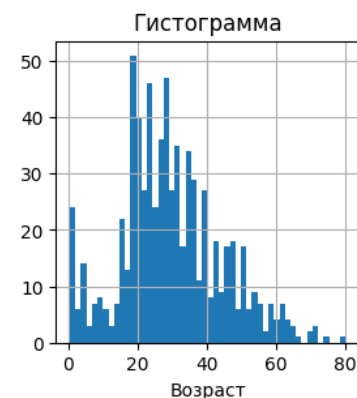
Набор параметров функции `plot()` зависит от типа графика.

Чаще всего это удобнее, чем использовать исходный синтаксис matplotlib, особенно для быстрого взгляда на данные, т.к. визуализация может быть настроена в одну строчку:

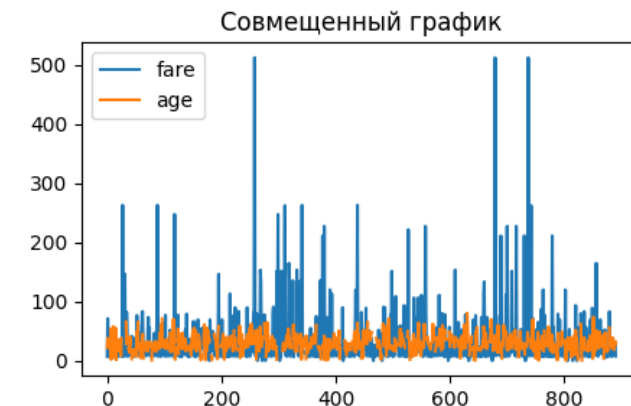
```
df.fare.plot(figsize=(5,1))
```



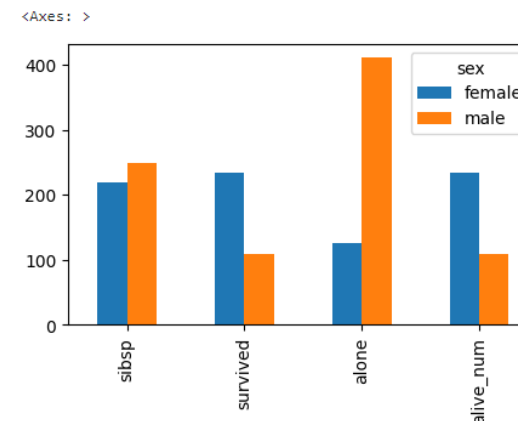
```
df['age'].hist(figsize=(3,3), bins=50)
plt.xlabel('Возраст')
plt.title('Гистограмма')
plt.show()
```



```
df[['fare', 'age']].plot(figsize=(5,3))
plt.title('Совмещенный график')
plt.show()
```



```
df['alive_num'] = df['alive']=='yes'
df.groupby('sex')[['sibsp', 'survived', 'alone', 'alive_num']].sum().T.plot.bar(figsize=(5,3))
```



Seaborn – пакет для визуализации данных в Python (на базе matplotlib), красивее и синтаксически проще, чем matplotlib.

```
import seaborn as sns
```

В отличие от matplotlib здесь не нужно прописывать отдельные конструкции, все атрибуты задаются в параметрах функции.

Основные методы:

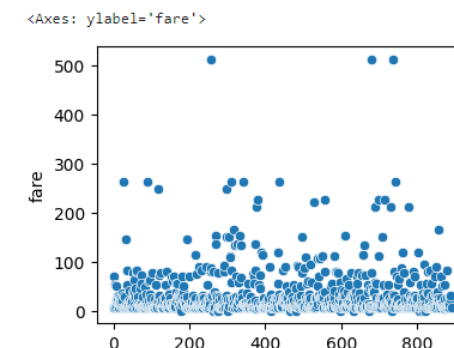
- `sns.barplot(x)` – столбчатая диаграмма
- `sns.heatmap(x)` – тепловая карта
- `sns.scatterplot(x, y)` – диаграмма рассеяния
- `sns.FacedGrid(df)` – связанные графики
- `sns.boxplot(x)` – диаграмма размаха
- `sns.violinplot(x)` – скрипичная диаграмма
- `sns.pairplot(x)` – парный график
- `sns.histplot(x)` – гистограмма
- `sns.displot(x)` – оценка распределения вероятностей

Seaborn хорошо использовать там, где нужны более красивые и информативные графики, например, для презентации или для более глубокого анализа.

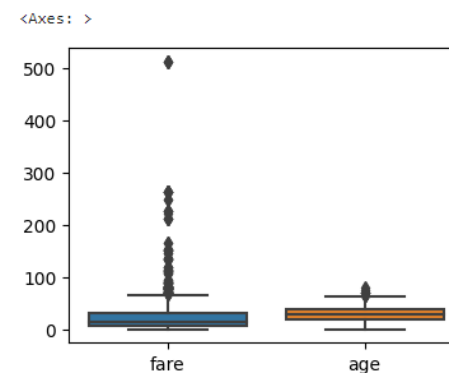
```
plt.figure(figsize=(4, 3))
sns.barplot(x="class", y="survived", hue="sex", data=df)
```



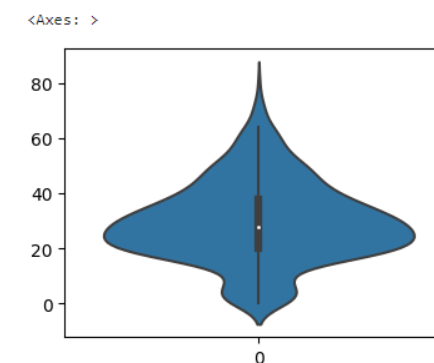
```
plt.figure(figsize=(4, 3))
sns.scatterplot(df['fare'])
```



```
plt.figure(figsize=(4, 3))
sns.boxplot(df[['fare', 'age']])
```



```
plt.figure(figsize=(4, 3))
sns.violinplot(df['age'])
```



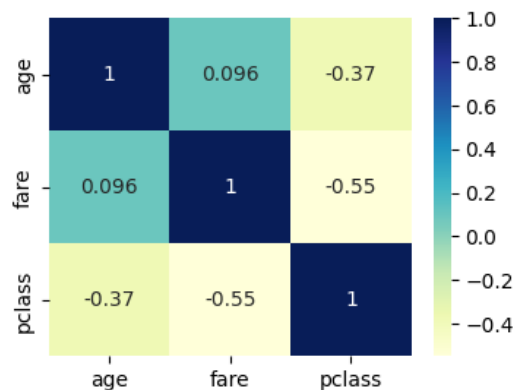
Матрица корреляций хорошо помогает при оценке взаимосвязей между переменными.

Для вычисления самой матрицы обычно применяется метод `corr()` /pandas, а для отображения – встроенные функции pandas или тепловая карта (`heatmap()`) из sns: `sns.heatmap(df.corr())`

На пересечении соответствующего столбца и строки находится значение коэффициента корреляции этих двух переменных.

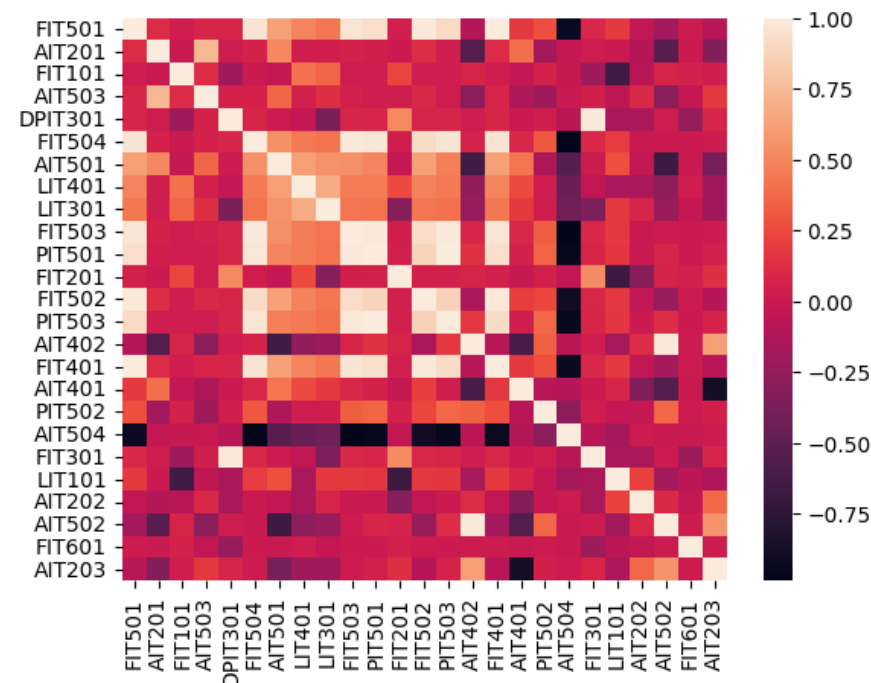
```
plt.figure(figsize=(4, 3))
sns.heatmap(df[['age', 'fare', 'pclass']].sort_values('age').corr(),
            cmap="YlGnBu",
            annot=True)
```

<Axes: >



```
sns.heatmap(df[cont_columns].corr())
```

<Axes: >



plotly – библиотека для визуализации данных, в т.ч. [интерактивной](#).

`import plotly`

Plotly очень функциональная, но из-за этого не очень легкая библиотека, поэтому для нее существуют различные модули.

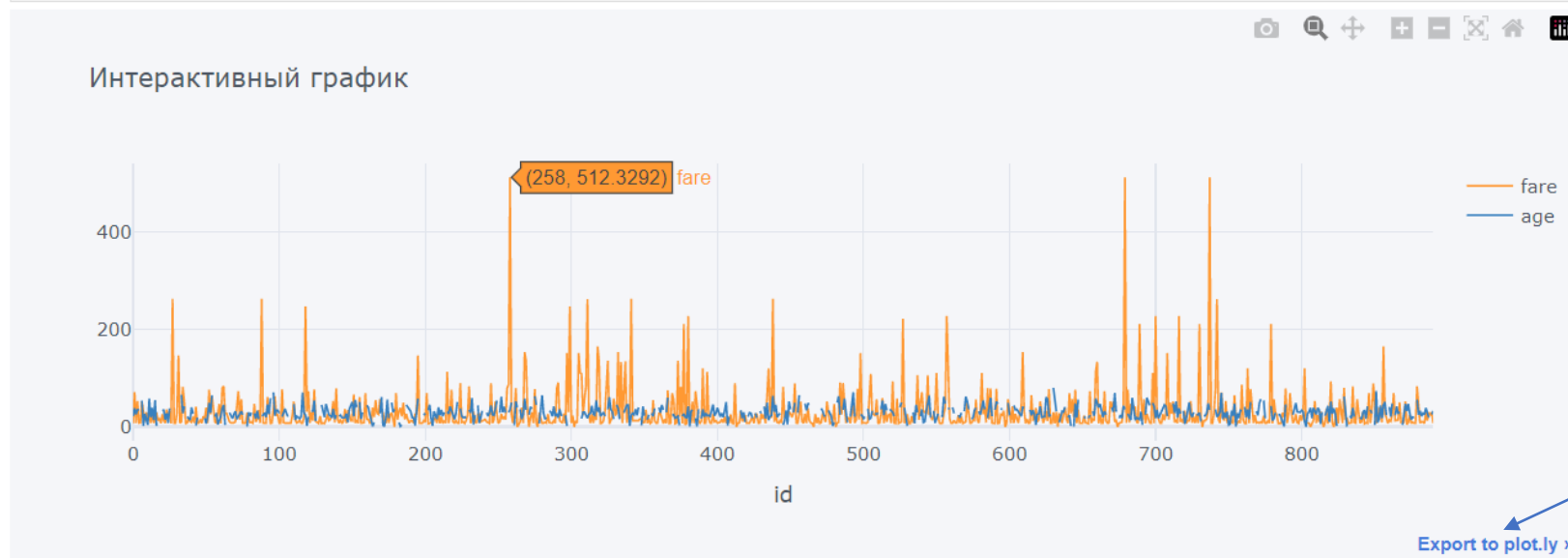
Например, модуль [plotly.express](#) или [cufflinks](#) для визуализации данных Pandas.

cufflinks – модуль для упрощенной визуализации данных Pandas с использованием Plotly.

`import cufflinks`

Для отображения графика вместо стандартного `df.plot()` для датафрейма нужно использовать `df.iplot()`.

```
import cufflinks
df[['fare', 'age']].iplot(title='Интерактивный график', xTitle='id')
```



Также можно выгрузить данные в облако и очень удобно поиграть с ними там, выбирая различные представления, настройки и т.п.

Вид анализа	Цель	Инструменты
Описательный (descriptive)	Количественное описание основных характеристик выборки.	Pandas в большинстве случаев достаточно
Разведочный (exploratory, EDA)	Нахождение общих закономерностей, инсайтов, распределений, выбросов в данных.	Pandas для основных задач, SciPy (Numpy) для математических вычислений, для интерактивной визуализации – Plotly , для красивой – Seaborn
Индуктивный (inferential)	Оценка генеральной совокупности на основании выборки, выявление и оценка причинно-следственных связей между переменными.	Pandas или Numpy
Прогностический (predictive)	Предсказать поведение данных в будущем на основании их прошлых значений.	Библиотеки машинного обучения (про это в следующем курсе)
Причинно-следственный (causal)	Объяснение с точки зрения данных причин возникновения события (следствия).	Библиотеки машинного обучения (про это в следующем курсе)



Пока все.