Операции в pandas

In [1]:

```
import numpy as np
import pandas as pd
import scipy.stats as sps
```

1. Простые операции

Сгенерируем DataFrame из случайных чисел

In [2]:

```
df = pd.DataFrame(sps.norm.rvs(size=(10, 4)),
                  columns=['A', 'B', 'C', 'D'])
df
```

Out[2]:

	A	B	C	D
0	0.098911	-1.311820	-0.498644	1.580585
1	0.664819	-0.003627	1.745190	-0.098430
2	0.879059	1.933319	-0.543434	0.269857
3	-0.073782	-0.050684	1.783527	2.475760
4	0.160128	0.251324	-0.472247	0.748638
5	0.933652	0.242252	0.766620	-1.051650
6	-0.332785	-0.145319	-0.168256	0.701769
7	-0.842948	-0.524782	-1.049934	0.177362
8	-0.319928	-1.231453	0.739019	2.052241
9	0.551910	1.470252	-0.063701	-2.340529

Выведем описательные статистики по столбцам -- количество значений, среднее, стандартное отклонение (корень из дисперсии), минимум, квантили, максимум.

In [3]:

```
1  df.describe()
```

Out[3]:

	A	B	C	D
count	10.000000	10.000000	10.000000	10.000000
mean	0.171904	0.062946	0.223814	0.451560
std	0.583101	1.028377	0.986650	1.435704
min	-0.842948	-1.311820	-1.049934	-2.340529
25%	-0.258392	-0.429916	-0.492045	-0.029482
50%	0.129520	-0.027156	-0.115978	0.485813
75%	0.636592	0.249056	0.759720	1.372598
max	0.933652	1.933319	1.783527	2.475760

Среднее по столбцам

In [4]:

```
1  df.mean()
```

Out[4]:

```
A    0.171904
B    0.062946
C    0.223814
D    0.451560
dtype: float64
```

Оценка матрицы корреляций значений в столбцах

In [5]:

```
1  df.corr()
```

Out[5]:

	A	B	C	D
A	1.000000	0.634713	0.27549	-0.507558
B	0.634713	1.000000	-0.10900	-0.616433
C	0.275490	-0.109000	1.00000	0.196100
D	-0.507558	-0.616433	0.19610	1.000000

Применение функции к данным. Для примера посчитаем разброс значений -- разница максимума и минимума.

In [6]:

```
df.apply(lambda x: x.max() - x.min())
```

Out[6]:

```
A    1.776600
B    3.245139
C    2.833462
D    4.816289
dtype: float64
```

2. Объединение таблиц

2.1 `df.append`

Добавление строк (в виде таблицы `other`) в таблицу `df`. При наличии у новых строк колонок, которых нет в таблице, они добавляются в таблицу.

```
df.append(other, ignore_index=False, verify_integrity=False, sort=None)
```

- `df` -- таблица;
- `other` -- добавляемые строки (в виде таблицы);
- `ignore_index` -- сохранить индексы или определить и как 0, ..., n-1;
- `verify_integrity` -- если `True`, то создает исключение в случае повторения индексов;
- `sort` -- сортировать ли колонки, если они (или их порядок) различаются.

Создадим новую таблицу из первых четырех строк таблицы `df`. В новую таблицу добавим колонку `flag`, в которую запишем условие, что число в столбце D положительно. Затем добавим строки из новой таблицы к старой. Полученная таблица содержит пропуски, которые отмечены как `NaN`.

In [7]:

```
other = df[:4].copy()  # Полное копирование
other['flag'] = other['D'] > 0
other['D'] = other['D'] ** 2

df.append(other, ignore_index=True, sort=False)
```

Out[7]:

	A	B	C	D	flag
0	0.098911	-1.311820	-0.498644	1.580585	NaN
1	0.664819	-0.003627	1.745190	-0.098430	NaN
2	0.879059	1.933319	-0.543434	0.269857	NaN
3	-0.073782	-0.050684	1.783527	2.475760	NaN
4	0.160128	0.251324	-0.472247	0.748638	NaN
5	0.933652	0.242252	0.766620	-1.051650	NaN
6	-0.332785	-0.145319	-0.168256	0.701769	NaN
7	-0.842948	-0.524782	-1.049934	0.177362	NaN
8	-0.319928	-1.231453	0.739019	2.052241	NaN
9	0.551910	1.470252	-0.063701	-2.340529	NaN
10	0.098911	-1.311820	-0.498644	2.498248	True
11	0.664819	-0.003627	1.745190	0.009688	False
12	0.879059	1.933319	-0.543434	0.072823	True
13	-0.073782	-0.050684	1.783527	6.129387	True

2.2 `pd.concat`

Соединение таблиц вдоль выбранной оси

```
pd.concat(objs, axis=0, join='outer', ignore_index=False, copy=True, ...)
```

- `objs` -- объединяемые таблицы;
- `axis` : { `0` или `'index'` , `1` или `'columns'` } -- ось;
- `join` : { `'inner'` , `'outer'` } -- тип объединения (пересечение или объединение);
- `ignore_index` -- сохранить индексы или определить и как 0, ..., n-1;
- `copy` -- копировать данные или нет.

Простой пример соединения таблиц:

In [8]:

```
1  pd.concat([df[:5], df[5:]])
```

Out[8]:

	A	B	C	D
0	0.098911	-1.311820	-0.498644	1.580585
1	0.664819	-0.003627	1.745190	-0.098430
2	0.879059	1.933319	-0.543434	0.269857
3	-0.073782	-0.050684	1.783527	2.475760
4	0.160128	0.251324	-0.472247	0.748638
5	0.933652	0.242252	0.766620	-1.051650
6	-0.332785	-0.145319	-0.168256	0.701769
7	-0.842948	-0.524782	-1.049934	0.177362
8	-0.319928	-1.231453	0.739019	2.052241
9	0.551910	1.470252	-0.063701	-2.340529

2.3 `pd.merge` и `df.join`

Слияние таблиц путем выполнения операций слияния баз данных в стиле SQL

```
pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None,
left_index=False, right_index=False, suffixes=('_x', '_y'), ...)
```

- `left` и `right` -- объединяемые таблицы;
- `how` -- тип объединения:
  - `left` -- только по ключам из левой таблицы == *SQL left outer join*;
  - `right` -- только по ключам из правой таблицы == *SQL right outer join*;
  - `outer` -- по объединению ключей == *SQL full outer join*;
  - `inner` -- по пересечению ключей == *SQL inner join*;
- `on` -- имя (или имена) колонок, по которым будет производиться объединение (т.е. ключи). Если их несколько, то нужно передать список имен. Имена колонок в таблице должны совпадать;
- `left_on` и `right_on` -- аналогично `on` для случая, когда в таблицах различаются имена колонок, соответствующие ключам;
- `left_index` и `right_index` -- использовать ли индексы в качестве ключей;
- `suffixes` -- суффиксы, которые будут добавлены к тем колонкам, имена которых повторяются.

```
df.join(other, on=None, how='left', lsuffix='', rsuffix='', sort=False)
```

- `df` -- основная таблица. В качестве ключей используется индекс;
- `other` -- другая таблица;
- `on` -- колонка(-и) в `other`, соответствующая ключам, по ним происходит объединение. Если `None`, то используется индекс;
- `how` -- тип объединения (см. `pd.merge`)
- `lsuffix` и `rsuffix` -- суффиксы, которые будут добавлены к тем колонкам, имена которых повторяются.

Пример 1.

В обеих таблицах ключи повторяются

In [9]:

```
left = pd.DataFrame({'key': ['A', 'A'],
                     'lval': [1, 2]})
right = pd.DataFrame({'key': ['A', 'A'],
                      'rval': [4, 5]})
```

In [10]:

```
left
```

Out[10]:

	key	lval
0	A	1
1	A	2

In [11]:

```
right
```

Out[11]:

	key	rval
0	A	4
1	A	5

В результате объединения получаем 4 строки -- для каждой строки из левой таблице есть две строки из правой таблицы с таким же ключом.

In [12]:

```
pd.merge(left, right, on='key')
```

Out[12]:

	key	lval	rval
0	A	1	4
1	A	1	5
2	A	2	4
3	A	2	5

Пример 2.

В таблицах ключи не повторяются

In [13]:

```
left = pd.DataFrame({'key': ['A', 'B'],
                     'lval': [1, 2]})
right = pd.DataFrame({'key': ['A', 'B'],
                      'rval': [4, 5]})
```

In [14]:

```
left
```

Out[14]:

	key	lval
0	A	1
1	B	2

In [15]:

```
right
```

Out[15]:

	key	rval
0	A	4
1	B	5

В результате объединения получаем 2 строки -- для каждой строки из левой таблице есть только одна строка из правой таблицы с таким же ключом.

In [16]:

```
pd.merge(left, right, on='key')
```

Out[16]:

	key	lval	rval
0	A	1	4
1	B	2	5

Пример 3.

Посмотрим на различные типы объединения. Сооздадим и напечатаем две таблицы.

In [17]:

```
left = pd.DataFrame({'lkey': ['A', 'B', 'C', 'A'],
                     'value': range(4)})
right = pd.DataFrame({'rkey': ['A', 'B', 'D', 'B'],
                      'value': range(4, 8)})
```

In [18]:

```
1  left
```

Out[18]:

	lkey	value
0	A	0
1	B	1
2	C	2
3	A	3

In [19]:

```
1  right
```

Out[19]:

	rkey	value
0	A	4
1	B	5
2	D	6
3	B	7

Внешнее слияние -- используются ключи из объединения списков ключей. Иначе говоря, используются ключи, которые есть хотя бы в одной из таблиц.

In [20]:

```
1 ▾ pd.merge(left, right,
2            left_on='lkey', right_on='rkey', how='outer')
```

Out[20]:

	lkey	value_x	rkey	value_y
0	A	0.0	A	4.0
1	A	3.0	A	4.0
2	B	1.0	B	5.0
3	B	1.0	B	7.0
4	C	2.0	NaN	NaN
5	NaN	NaN	D	6.0

Внутреннее слияние -- используются ключи из пересечения списков ключей. Иначе говоря, используются ключи, которые присутствуют в обеих таблицах.

In [21]:

```
pd.merge(left, right,
         left_on='lkey', right_on='rkey', how='inner')
```

Out[21]:

	lkey	value_x	rkey	value_y
0	A	0	A	4
1	A	3	A	4
2	B	1	B	5
3	B	1	B	7

Объединение по ключам левой таблицы. Не используются ключи, которые есть в правой таблицы, но которых нет в левой

In [22]:

```
pd.merge(left, right,
         left_on='lkey', right_on='rkey', how='left')
```

Out[22]:

	lkey	value_x	rkey	value_y
0	A	0	A	4.0
1	B	1	B	5.0
2	B	1	B	7.0
3	C	2	NaN	NaN
4	A	3	A	4.0

Объединение по ключам правой таблицы. Не используются ключи, которые есть в левой таблицы, но которых нет в правой

In [23]:

```
pd.merge(left, right,
         left_on='lkey', right_on='rkey', how='right')
```

Out[23]:

	lkey	value_x	rkey	value_y
0	A	0.0	A	4
1	A	3.0	A	4
2	B	1.0	B	5
3	B	1.0	B	7
4	NaN	NaN	D	6

Выполним внтуреннее объединение и установим ключ качестве индекса

In [24]:

```
pd.merge(left, right,
         left_on='lkey', right_on='rkey', how='inner') \
        .set_index('lkey')[['value_x', 'value_y']]
```

Out[24]:

lkey	value_x	value_y
A	0	4
A	3	4
B	1	5
B	1	7

Ту же операцию можно выполнить с помощью `join`

In [25]:

```
left.set_index('lkey') \
    .join(right.set_index('rkey'), rsuffix='_r')
```

Out[25]:

	value	value_r
A	0	4.0
A	3	4.0
B	1	5.0
B	1	7.0
C	2	NaN

3. Группировка

Этапы группировки данных:

- разбиение данных на группы по некоторым критериям;
- применение функции отдельно к каждой группе;
- комбинирование результата в структуру данных.

Группировка выполняется функцией

```
df.groupby(by=None, axis=0, level=None, sort=True, ...)
```

- `df` -- таблица, данные которой должны быть сгруппированы;
- `by` -- задает принцип группировки. Чаще всего это имя столбца, по которому нужно сгруппировать. Может так же быть функцией;
- `axis` -- ось (0 = группировать строки, 1 = группировать столбцы);
- `level` -- если ось представлена мультииндексом, то указывает на уровень мультииндекса;
- `sort` -- сортировка результата по индексу.

Результатом группировки является объект, состоящий из пар (имя группы, подтаблица). Имя группы соответствует значению, по которому произведена группировка. К объекту-результату группировки применимы, например, следующие операции:

- `for name, group in groupped: ...` -- цикл по группам;
- `get_group(name)` -- получить таблицу, соответствующую группе с именем `name` ;
- `groups` -- получить все группы в виде словаря имя-подтаблица;
- `count()` -- количество значений в группах, исключая пропуски;
- `size()` -- размер групп;
- `sum()` , `max()` , `min()` ;
- `mean()` , `median()` , `var()` , `std()` , `corr()` , `quantile(q)` ;
- `describe()` -- вывод описательных статистик;
- `aggregate(func)` -- применение функции (или списка функций) `func` к группам.

Создадим таблицу для примера (ура, котики!)

In [26]:

```
df = pd.DataFrame({
    'Животное' : ['Котик', 'Песик', 'Котик', 'Песик',
                  'Котик', 'Песик', 'Котик', 'Песик'],
    'Цвет шерсти' : ['белый', 'белый', 'коричневый', 'черный',
                     'коричневый', 'коричневый', 'белый', 'черный'],
    'Рост' : sps.gamma(a=12, scale=3).rvs(size=8),
    'Длина хвостика' : sps.gamma(a=10).rvs(size=8)
})

df
```

Out[26]:

	Животное	Цвет шерсти	Рост	Длина хвостика
0	Котик	белый	27.380391	7.488426
1	Песик	белый	24.989956	7.387730
2	Котик	коричневый	23.819317	9.193795
3	Песик	черный	35.805990	11.180022
4	Котик	коричневый	34.044305	14.427289
5	Песик	коричневый	32.002396	12.568725
6	Котик	белый	54.752458	9.349173
7	Песик	черный	30.974410	8.463583

Пример 1.

Если все котики встанут друг на друга, то какой их суммарный рост? А у песиков? А какова суммарная длинна хвостиков у котиков и у песиков?

Группировка по одной колонке и последующее применение операции суммирования:

In [27]:

```
df.groupby('Животное').sum()
```

Out[27]:

Животное	Рост	Длина хвостика
Котик	139.996471	40.458683
Песик	123.772752	39.600060

Посчитаем описательные статистики для каждого животного

In [28]:

```
df.groupby('Животное').describe()
```

Out[28]:

	Рост							
	count	mean	std	min	25%	50%	75%	max
Животное								
Котик	4.0	34.999118	13.834005	23.819317	26.490123	30.712348	39.221343	54.752458
Песик	4.0	30.943188	4.479982	24.989956	29.478296	31.488403	32.953294	35.805990

Пример 2.

Теперь предположим, что котики и песики встают только на представителей своего вида и своего цвета шерсти. Что тогда будет?

Группировка по двум колонкам и последующее применение операции суммирования

In [29]:

```
df.groupby(['Животное', 'Цвет шерсти']).sum()
```

Out[29]:

Животное	Цвет шерсти	Рост	Длина хвостика
Котик	**белый**	82.132850	16.837599
	коричневый	57.863622	23.621084
Песик	**белый**	24.989956	7.387730
	коричневый	32.002396	12.568725
	черный	66.780400	19.643605

Полученная таблица имеет *мультииндекс*

In [30]:

```
1  df.groupby(['Животное', 'Цвет шерсти']).sum().index
```

Out[30]:

```
MultiIndex([('Котик',      'белый'),
            ('Котик', 'коричневый'),
            ('Песик',      'белый'),
            ('Песик', 'коричневый'),
            ('Песик',     'черный')],
           names=['Животное', 'Цвет шерсти'])
```

4. Таблицы сопряженности (Crosstab) и сводные таблицы (Pivot table)

Задача. В медицинской клинике информацию о приемах записывают в таблицу со следующими полями:

- время приема
- врач
- пациент
- поставленный диагноз
- назначение
- другие поля

Требуется посчитать, сколько раз за предыдущий месяц каждый врач ставил какой-либо диагноз. Результаты представить в виде таблицы, в которой посчитать также суммы по строкам и столбцам, т.е. сколько врач сделал приемов за месяц и сколько раз конкретный диагноз поставлен всеми врачами.

Как решать?

Способ 1

1. Группировка по врачам.
2. Для каждого врача группировка по диагнозам.
3. В каждой группе вычисление суммы.
4. Соединение в одну таблицу.
5. Вычисление суммы по столбцам и по строкам.

Оцените количество строк кода и время работы.

Способ 2

1. Создать пустую таблицу.
2. Циклом [**:)))**] по всем записям исходной таблицы считать суммы.
3. Вычисление суммы по столбцам и по строкам.

Оцените количество строк кода и время работы.

Способ 3

Применить умную функцию из pandas, которая сделает все сама!

4.1 `pd.crosstab`

Эксель-подобные таблицы сопряженности

```
pd.crosstab(index, columns, values=None, rownames=None, colnames=None, aggfunc=None, margins=False, margins_name='All', dropna=True, normalize=False)
```

- `index` -- значения для группировки по строкам;
- `columns` -- значения для группировки по столбцам;
- `values` -- аггригируемый столбец (или столбцы), его значения непосредственно определяют значения таблицы сопряженности;
- `aggfunc` -- функция, которая будет применена к каждой группе значений `values`, сгруппированным по значениям `index` и `columns`. Значения этой функции и есть значения сводной таблицы;
- `rownames` и `colnames` -- имена строк и столбцов таблицы сопряженности;
- `margins` -- добавляет результирующий столбец/строку;
- `margins_name` -- имя результирующего столбец/строку;
- `dropna` -- не включать столбцы, которые состоят только из `NaN`;
- `normalize`: `boolean`, `{'all', 'index', 'columns'}` -- нормировка всей таблицы (или только по строкам/столбцам).

В примере выше:

```
pd.crosstab(df['Врач'], df['Диагноз'], margins=True)
```

4.2 `pd.pivot_table`

Эксель-подобные сводные таблицы

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

- `data` -- исходная таблица;
- `values` -- аггригируемый столбец, его значения непосредственно определяют значения сводной таблицы;
- `index` -- ключи для группировки, относятся к индексам сводной таблицы;
- `columns` -- ключи для группировки, относятся к столбцам сводной таблицы;
- `aggfunc` -- функция, которая будет применена к каждой группе значений `values`, сгруппированным по значениям `index` и `columns`. Значения этой функции и есть значения сводной таблицы. Если передается список функций, то сводная таблица имеет иерархические имена колонок, верхние значения которых -- имена функций;
- `fill_value` -- значения для замены пропусков;
- `dropna` -- не включать столбцы, которые состоят только из `NaN`;
- `margins` -- добавляет результирующий столбец/строку;
- `margins_name` -- имя результирующего столбец/строку.

В примере выше:

```
pd.pivot_table(df, index='врач', columns='диагноз', margins=True)
```

Создадим таблицу для примера

In [31]:

```
df = pd.DataFrame({
    'Специальность' : ['Ветеринар', 'Ветеринар',
                       'Психолог', 'Психолог'] * 6,
    'Врач' : ['Андрей', 'Сергей', 'Ирина'] * 8,
    'Диагноз' : ['Простуда', 'Простуда', 'Простуда',
                 'Волнения', 'Волнения', 'Простуда'] * 4,
    'Доза' : sps.randint(low=1, high=6).rvs(size=24),
    'Продолжительность' : sps.randint(low=1, high=6).rvs(size=24)
})

df
```

Out[31]:

	Специальность	Врач	Диагноз	Доза	Продолжительность
0	Ветеринар	Андрей	Простуда	1	5
1	Ветеринар	Сергей	Простуда	5	5
2	Психолог	Ирина	Простуда	2	2
3	Психолог	Андрей	Волнения	4	4
4	Ветеринар	Сергей	Волнения	3	1
5	Ветеринар	Ирина	Простуда	3	4
6	Психолог	Андрей	Простуда	5	3
7	Психолог	Сергей	Простуда	4	5
8	Ветеринар	Ирина	Простуда	3	3
9	Ветеринар	Андрей	Волнения	2	3
10	Психолог	Сергей	Волнения	1	4
11	Психолог	Ирина	Простуда	1	3
12	Ветеринар	Андрей	Простуда	5	1
13	Ветеринар	Сергей	Простуда	3	1
14	Психолог	Ирина	Простуда	3	3
15	Психолог	Андрей	Волнения	5	2
16	Ветеринар	Сергей	Волнения	2	2
17	Ветеринар	Ирина	Простуда	4	4
18	Психолог	Андрей	Простуда	4	5
19	Психолог	Сергей	Простуда	3	3
20	Ветеринар	Ирина	Простуда	4	2
21	Ветеринар	Андрей	Волнения	5	4
22	Психолог	Сергей	Волнения	4	2
23	Психолог	Ирина	Простуда	5	1

Посчитаем, сколько раз какой врач ставил каждый из диагнозов, а также суммы по строкам и столбцам

In [32]:

```
1  pd.crosstab(df['Врач'], df['Диагноз'], margins=True)
```

Out[32]:

Диагноз	Волнения	Простуда	All
Врач			
Андрей	4	4	8
Ирина	0	8	8
Сергей	4	4	8
All	8	16	24

Посчитаем, какую среднюю дозу какой врач назначал по каждому из диагнозов

In [33]:

```
1 ▾ pd.crosstab(df['Врач'], df['Диагноз'],
2               values=df['Доза'], aggfunc=np.mean)
```

Out[33]:

Диагноз	Волнения	Простуда
Врач		
Андрей	4.0	3.750
Ирина	NaN	3.125
Сергей	2.5	3.750

Простейший вариант сводной таблицы -- среднее в группах, определяемых столбцом. Посчитаем средние по каждому врачу

In [34]:

```
1  pd.pivot_table(df, index=['Врач'])
```

Out[34]:

	Доза	Продолжительность
Врач		
Андрей	3.875	3.375
Ирина	3.125	2.750
Сергей	3.125	2.875

Посчитаем, сколько раз врач и в какой специальности ставил тот или иной диагноз

In [35]:

```
pd.pivot_table(df,
               values='Доза',
               index=['Специальность', 'Врач'],
               columns=['Диагноз'],
               aggfunc=np.sum)
```

Out[35]:

	Диагноз	Волнения	Простуда
Специальность	**Врач**		
Ветеринар	**Андрей**	7.0	6.0
	Ирина	NaN	14.0
	Сергей	5.0	8.0
Психолог	**Андрей**	9.0	9.0
	Ирина	NaN	11.0
	Сергей	5.0	7.0

Добавим строчку, являющейся суммой столбцов, и столбец, являющийся суммой строк

In [36]:

```
pd.pivot_table(df,
               values='Доза',
               index=['Специальность', 'Врач'],
               columns=['Диагноз'],
               aggfunc=np.sum,
               margins=True)
```

Out[36]:

	Диагноз	Волнения	Простуда	All
Специальность	**Врач**			
Ветеринар	**Андрей**	7.0	6.0	13
	Ирина	NaN	14.0	14
	Сергей	5.0	8.0	13
Психолог	**Андрей**	9.0	9.0	18
	Ирина	NaN	11.0	11
	Сергей	5.0	7.0	12
All		26.0	55.0	81

Применим несколько функций и несколько столбцов со значениями

In [37]:

```
pd.pivot_table(df,
               values=['Доза', 'Продолжительность'],
               index=['Специальность', 'Врач'],
               columns=['Диагноз'],
               aggfunc=[np.min, np.mean, np.max],
               margins=True)
```

Out[37]:

		amin						mean	
		Доза			Продолжительность			Доза	
	Диагноз	Волнения	Простуда	All	Волнения	Простуда	All	Волнения	Пр
Специальность	Врач								
Ветеринар	Андрей	2.0	1.0	1	3.0	1.0	1	3.50	
	Ирина	NaN	3.0	3	NaN	2.0	2	NaN	
	Сергей	2.0	3.0	2	1.0	1.0	1	2.50	
Психолог	Андрей	4.0	4.0	4	2.0	3.0	2	4.50	
	Ирина	NaN	1.0	1	NaN	1.0	1	NaN	
	Сергей	1.0	3.0	1	2.0	3.0	2	2.50	
All		1.0	1.0	1	1.0	1.0	1	3.25	

Введение в анализ данных, 2020

Никита Волков

https://mipt-stats.gitlab.io/ (https://mipt-stats.gitlab.io/)