Библиотека pandas : часть 1

Урок 5.1: основные структуры

Библиотека `pandas` используется для удобной и более эффективной работы с таблицами. Её функционал достаточно разнообразен, но давайте начнем с каких-то базовых функций и методов. Для начала импортируем саму библиотеку (она была установлена вместе с Anaconda):

In [1]:

```
import pandas as pd
```

Здесь мы опять использовали тот же трюк, что и с библиотекой `NumPy` – импортировали её с сокращённым названием. Основная структура данных в `pandas` – это датафрейм (`DataFrame`), который можно рассматривать как совокупность массивов `NumPy` , а точнее как таблицу, столбцами которой являются массивы. Библиотека `pandas` позволяет загружать данные из файлов разных форматов (*csv*, *xls*, *json*), но так как у вас будет отдельный модуль, посвящённый работе с файлами, давайте для примера создадим маленький датафрейм с нуля, из списка списков. В каждом списке указано имя респондента, его возраст и число лет опыта работы, каждый список можно рассматривать как строку в таблице:

In [2]:

```
df = pd.DataFrame([['Anna', 23, 3],
                   ['Sam', 36, 12],
                   ['Bill', 33, 10],
                   ['Moica', 25, 7],
                   ['Lisa', 27, 7],
                   ['Peter', 32, None]])
```

Посмотрим на датафрейм:

In [3]:

```
df
```

Out[3]:

	0	1	2
0	Anna	23	3.0
1	Sam	36	12.0
2	Bill	33	10.0
3	Moica	25	7.0
4	Lisa	27	7.0
5	Peter	32	NaN

На датафреймы смотреть гораздо приятнее, чем на массивы. Проверим, какой тип у полученного объекта:

In [4]:

```
type(df)  # pandas DataFrame
```

Out[4]:

```
pandas.core.frame.DataFrame
```

В начале урока я сказала, что на датафрейм можно смотреть как на совокупность массивов. На самом деле, если говорить совсем точно, датафрейм – это совокупность особых объектов `pandas Series`, последовательностей `pandas`.

In [5]:

```
df[0]
```

Out[5]:

```
0     Anna
1      Sam
2     Bill
3    Moica
4     Lisa
5    Peter
Name: 0, dtype: object
```

Столбец датафрейма `df` имеет особый тип *Series*. Внешне *Series* отличается от обычного списка значений, потому что, во-первых, при вызове столбца на экран выводятся не только сами элементы, но их номер (номер строки), а во-вторых, на экран выводится строка с названием столбца (`Name: 0`) и его тип (`dtype: object`, текстовый). Первая особенность роднит *Series* со словарями: он представляет собой пары ключ-значение, то есть номер-значение. Вторая особенность роднит *Series* с массивами `NumPy`: элементы должны быть одного типа.

Урок 5.2: индексы и метод `.iloc`

In [1]:

```
import pandas as pd
df = pd.DataFrame([['Anna', 23, 3],
                   ['Sam', 36, 12],
                   ['Bill', 33, 10],
                   ['Moica', 25, 7],
                   ['Lisa', 27, 7],
                   ['Peter', 32, None]])
```

Чтобы наш датафрейм был больше похож на настоящие данные из файла, давайте назовём столбцы в таблице. Для этого нам необходимо изменить атрибут `.columns`, присвоить ему значение в виде списка.

In [2]:

```
df.columns = ['name', 'age', 'expr']
```

In [3]:

```
df
```

Out[3]:

	name	age	expr
0	Anna	23	3.0
1	Sam	36	12.0
2	Bill	33	10.0
3	Moica	25	7.0
4	Lisa	27	7.0
5	Peter	32	NaN

Для выбора строк и столбцов в `Pandas` есть два основных метода: `.iloc` и `.loc`. Первый используется для выбора строк и столбцов по их индексу, второй – по их названию. Внутри каждого из этих методов в квадратных скобках указывается сначала идентификатор (индекс или название) строки, а затем – идентификатор столбца. Попробуем выбрать элемент, который находится в строке с индексом 1 и в столбце с индексом 2:

In [4]:

```
df.iloc[1, 2]
```

Out[4]:

```
12.0
```

В качестве индексов можно указывать числовые срезы (как обычно, правый конец отрезка не включается):

In [11]:

```
df.iloc[1:3, 1]
```

Out[11]:

```
1    36
2    33
Name: age, dtype: int64
```

И полные срезы тоже:

In [12]:

```
df.iloc[:, 0]
```

Out[12]:

```
0     Anna
1      Sam
2     Bill
3    Moica
4     Lisa
5    Peter
Name: name, dtype: object
```

In [13]:

```
df.iloc[1:3, :]
```

Out[13]:

	name	age	expr
1	Sam	36	12.0
2	Bill	33	10.0

Можно попробовать ввести индекс без `.loc` и посмотреть, что получится:

In [9]:

```
df[0]
```

```
---------------------------------------------------------------------------
-
KeyError                                  Traceback (most recent call las
t)
/anaconda3/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_
loc(self, key, method, tolerance)
   3062             try:
-> 3063                 return self._engine.get_loc(key)
   3064             except KeyError:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObject
HashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObject
HashTable.get_item()

KeyError: 0

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call las
t)
<ipython-input-9-ad11118bc8f3> in <module>()
----> 1 df[0]

/anaconda3/lib/python3.6/site-packages/pandas/core/frame.py in __getitem__
(self, key)
   2683             return self._getitem_multilevel(key)
   2684         else:
-> 2685             return self._getitem_column(key)
   2686
   2687     def _getitem_column(self, key):

/anaconda3/lib/python3.6/site-packages/pandas/core/frame.py in _getitem_co
lumn(self, key)
   2690         # get column
   2691         if self.columns.is_unique:
-> 2692             return self._get_item_cache(key)
   2693
   2694         # duplicate columns & possible reduce dimensionality

/anaconda3/lib/python3.6/site-packages/pandas/core/generic.py in _get_item
_cache(self, item)
   2484         res = cache.get(item)
   2485         if res is None:
-> 2486             values = self._data.get(item)
   2487             res = self._box_item_values(item, values)
   2488             cache[item] = res

/anaconda3/lib/python3.6/site-packages/pandas/core/internals.py in get(sel
f, item, fastpath)
   4113
   4114             if not isna(item):
-> 4115                 loc = self.items.get_loc(item)
   4116             else:
   4117                 indexer = np.arange(len(self.items))[isna(self.ite
ms)]
```

```
/anaconda3/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_
loc(self, key, method, tolerance)
   3063                 return self._engine.get_loc(key)
   3064             except KeyError:
-> 3065                 return self._engine.get_loc(self._maybe_cast_index
er(key))
   3066
   3067         indexer = self.get_indexer([key], method=method, tolerance
=tolerance)

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObject
HashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObject
HashTable.get_item()

KeyError: 0
```

Ничего не получится: Python думает, что в квадратных скобках указано название столбца и возвращает `KeyError`, что означает, что столбца с таким названием в таблице нет. Самое интересное: если мы укажем в квадратных скобках числовой срез, всё сработает! Только Python будет воспринимать эти числа как индексы строк (да `pandas` коварна, к ней нужно привыкнуть):

In [10]:

```
df[0:2]
```

Out[10]:

	name	age	expr
0	Anna	23	3.0
1	Sam	36	12.0

Урок 5.3: индексы и метод .loc

In [3]:

```
import pandas as pd
df = pd.DataFrame([['Anna', 23, 3],
                   ['Sam', 36, 12],
                   ['Bill', 33, 10],
                   ['Moica', 25, 7],
                   ['Lisa', 27, 7],
                   ['Peter', 32, None]])
df.columns = ['name', 'age', 'expr']
```

Теперь посмотрим, как выбирать столбцы и строки по названию. Выберем столбец `name`, вписав его название в квадратных скобках:

In [4]:

```
df['name']
```

Out[4]:

```
0     Anna
1      Sam
2     Bill
3    Moica
4     Lisa
5    Peter
Name: name, dtype: object
```

Если нужно выбрать более одного столбца сразу, то названия столбцов нужно оформить в список и поместить его в квадратные скобки:

In [5]:

```
df[['name', 'expr']]
```

Out[5]:

	name	expr
0	Anna	3.0
1	Sam	12.0
2	Bill	10.0
3	Moica	7.0
4	Lisa	7.0
5	Peter	NaN

Хорошая новость: помимо числовых срезов в `Pandas` можно использовать текстовые срезы, то есть, сейчас я могу, например, выбрать все столбцы с `name` до `age` подряд. Только тогда этот срез нужно будет указать в методе `.loc`:

In [6]:

```
df.loc[:, 'name':'age']
```

Out[6]:

	name	age
0	Anna	23
1	Sam	36
2	Bill	33
3	Moica	25
4	Lisa	27
5	Peter	32

Обратите внимание: текстовый срез включает оба конца отрезка, правый конец не исключается.

Метод `.loc` будет работать так же, как и `.iloc`, только здесь в качестве идентификаторов будут использоваться названия. Для удобства назовём строки по столбцу `name`, чтобы к ним можно было обращаться по названию:

In [7]:

```
df.index = df.name
df
```

Out[7]:

	name	age	expr
name			
Anna	Anna	23	3.0
Sam	Sam	36	12.0
Bill	Bill	33	10.0
Moica	Moica	25	7.0
Lisa	Lisa	27	7.0
Peter	Peter	32	NaN

Теперь у строк есть текстовые названия. Выберем значение, соответствующее возрасту Билла:

In [10]:

```
df.loc['Bill', 'age']
```

Out[10]:

```
33
```

А теперь опыт работы для нескольких человек (строк) подряд:

In [11]:

```
df.loc['Sam':'Lisa', 'expr']
```

Out[11]:

```
name
Sam      12.0
Bill     10.0
Moica     7.0
Lisa      7.0
Name: expr, dtype: float64
```

А теперь два столбца одновременно (представим, что они идут не подряд, более общий способ):

In [12]:

```
df.loc['Sam':'Lisa', ['age', 'expr']]
```

Out[12]:

name	age	expr
Sam	36	12.0
Bill	33	10.0
Moica	25	7.0
Lisa	27	7.0

Урок 5.4: характеристики датафрейма pandas

In [1]:

```
import pandas as pd
df = pd.DataFrame([['Anna', 23, 3],
                   ['Sam', 36, 12],
                   ['Bill', 33, 10],
                   ['Moica', 25, 7],
                   ['Lisa', 27, 7],
                   ['Peter', 32, None]])
df.columns = ['name', 'age', 'expr']
```

Какую сводную информацию по таблице можно получить? Например, число переменных (столбцов) и наблюдений (строк), а также число заполненных значений.

In [2]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 3 columns):
name    6 non-null object
age     6 non-null int64
expr    5 non-null float64
dtypes: float64(1), int64(1), object(1)
memory usage: 224.0+ bytes
```

Какую информацию выдал метод `.info()` ? Во-первых, он сообщил нам, что `df` является объектом `DataFrame` . Во-вторых, он вывел число строк (`6 entries`) и показал их индексы. В-третьих, он вывел число столбцов (`total 3 columns`). Наконец, он выдал информацию по каждому столбцу. Остановимся на этом поподробнее.

В выдаче выше представлено, сколько непустых элементов содержится в каждом столбце. Непустые элементы `non-null` – это всё, кроме пропущенных значений, которые кодируются особым образом (`NaN` – от *Not A Number*). В нашей таблице почти все столбцы заполнены полностью: 6 ненулевых элементов из 6.

Далее указан тип каждого столбца, целочисленный `int64` и с плавающей точкой `float64` . Что означают числа в конце? Это объем памяти, который требуется для хранения.

При желании можно запросить число строк и столбцов отдельно:

In [3]:

```
df.shape  # как в массивах numpy
```

Out[3]:

```
(6, 3)
```

In [6]:

```
df.shape[0] # отдельно строкиё
```

Out[6]:

6

In [7]:

```
df.shape[1]  # отдельно столбцы
```

Out[7]:

3

Можем запросить описательные статистики по столбцам данного датафрейма:

In [12]:

```
df.describe()
```

Out[12]:

	age	expr
count	6.000000	5.000000
mean	29.333333	7.800000
std	5.085928	3.420526
min	23.000000	3.000000
25%	25.500000	7.000000
50%	29.500000	7.000000
75%	32.750000	10.000000
max	36.000000	12.000000

В случае количественных показателей этот метод возвращает таблицу с основными описательными статистиками:

- `count` – число непустых (заполненных) значений;
- `mean` – среднее арифметическое;
- `std` – стандартное отклонение (показатель разброса данных относительно среднего значения);
- `min` – миниммальное значение;
- `max` – максимальное значение;
- `25%` – нижний квартиль (значение, которое 25% значений не превышают);
- `50%` – медиана (значение, которое 50% значений не превышают);
- `75%` – верхний квартиль (значение, которое 75% значений не превышают).

Можем вывести названия столбцов:

In [8]:

```
df.columns
```

Out[8]:

```
Index(['name', 'age', 'expr'], dtype='object')
```

Обратите внимание: полученный объект не является обычным списком:

In [9]:

```
type(df.columns)
```

Out[9]:

```
pandas.core.indexes.base.Index
```

Если мы попробуем обратиться к элементу как обычно, всё получится:

In [13]:

```
df.columns[2]
```

Out[13]:

```
'expr'
```

А вот изменить значение уже нет:

In [14]:

```
df.columns[2] = 'experience'
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-14-03f886445795> in <module>()
----> 1 df.columns[2] = 'experience'

/anaconda3/lib/python3.6/site-packages/pandas/core/indexes/base.py in __se
titem__(self, key, value)
   2048
   2049     def __setitem__(self, key, value):
-> 2050         raise TypeError("Index does not support mutable operation
s")
   2051
   2052     def __getitem__(self, key):

TypeError: Index does not support mutable operations
```

Чтобы получить список названий, достаточно сконвертировать тип с помощью привычного `list()` :

In [10]:

```
list(df.columns)
```

Out[10]:

```
['name', 'age', 'expr']
```

Аналогично для строк:

In [11]:

```
df.index
```

Out[11]:

```
RangeIndex(start=0, stop=6, step=1)
```

Урок 5.5: операции над датафреймами – часть 1

In [1]:

```
import pandas as pd
df = pd.DataFrame([['Anna', 23, 3],
                   ['Sam', 36, 12],
                   ['Bill', 33, 10],
                   ['Moica', 25, 7],
                   ['Lisa', 27, 7],
                   ['Peter', 32, None]])
df.columns = ['name', 'age', 'expr']
```

Если датафрейм достаточно объёмный, иногда удобно вывести из него только первые несколько строк:

In [2]:

```
df.head()
```

Out[2]:

	name	age	expr
0	Anna	23	3.0
1	Sam	36	12.0
2	Bill	33	10.0
3	Moica	25	7.0
4	Lisa	27	7.0

По умолчанию выводятся первые 5, но это можно изменить:

In [3]:

```
df.head(2)
```

Out[3]:

	name	age	expr
0	Anna	23	3.0
1	Sam	36	12.0

Или вывести последние несколько строк:

In [4]:

```
df.tail()
```

Out[4]:

	name	age	expr
1	Sam	36	12.0
2	Bill	33	10.0
3	Moica	25	7.0
4	Lisa	27	7.0
5	Peter	32	NaN

Если в датафрейме присутствуют строки с пропущенными значениями (`NaN` , *Not a number*), то их можно удалить:

In [5]:

```
df = df.dropna()
df  # последней строки уже нет
```

Out[5]:

	name	age	expr
0	Anna	23	3.0
1	Sam	36	12.0
2	Bill	33	10.0
3	Moica	25	7.0
4	Lisa	27	7.0

Выбор строк по условиям

Если вы помните, как происходил выбор элементов массива по условиям, то похожая логика будет использоваться и в датафреймах `pandas` . Попробуем выбрать строки, соответствующие респондентам старше 30 лет:

In [6]:

```
df[df['age'] > 30]
```

Out[6]:

	name	age	expr
1	Sam	36	12.0
2	Bill	33	10.0

Теперь в квадратных скобках мы будем указывать целое условие (вспомните про отбор элементов из массива). Выберем респондентов не моложе 25 лет с опытом работы более 7 лет:

In [7]:

```
df[(df['age'] > 30) & (df['expr'] > 7)]  # не забываем круглые скобки
```

Out[7]:

	name	age	expr
1	Sam	36	12.0
2	Bill	33	10.0

Или респондентов старше 35 или моложе 25:

In [8]:

```
df[(df['age'] > 35) | (df['age'] < 25)]
```

Out[8]:

	name	age	expr
0	Anna	23	3.0
1	Sam	36	12.0

Урок 5.6: операции над датафреймами – часть 2

In [1]:

```
import pandas as pd
df = pd.DataFrame([['Anna', 23, 3],
                   ['Sam', 36, 12],
                   ['Bill', 33, 10],
                   ['Moica', 25, 7],
                   ['Lisa', 27, 7],
                   ['Peter', 32, None]])
df.columns = ['name', 'age', 'expr']
df = df.dropna()
```

Создание новых столбцов

Добавим столбец `age_sq`, содержащий значения возраста респондентов, возведенные в квадрат:

In [2]:

```
df['age_sq'] = df['age'] ** 2
df
```

Out[2]:

	name	age	expr	age_sq
0	Anna	23	3.0	529
1	Sam	36	12.0	1296
2	Bill	33	10.0	1089
3	Moica	25	7.0	625
4	Lisa	27	7.0	729

Сначала указывается название нового столбца в квадратных скобках, затем описывам те операции, которые необходимо выполнить со старым столбцом. Можно создать новый столбец на основе двух (и более) старых. Создадим столбец `no_work`, в котором будет сохранено число лет, которое люди не работали (возраст за вычетом лет опыта работы).

In [3]:

```
df['no_work'] = df['age'] - df['expr']
df
```

Out[3]:

	name	age	expr	age_sq	no_work
0	Anna	23	3.0	529	20.0
1	Sam	36	12.0	1296	24.0
2	Bill	33	10.0	1089	23.0
3	Moica	25	7.0	625	18.0
4	Lisa	27	7.0	729	20.0

Можно создать столбец с нуля, например, столбец, состоящий из одного значения (возьмем `W` как статус человека – в трудоспособном возрасте):

In [4]:

```
df['status'] = 'W'  # из одного значения
df
```

Out[4]:

	name	age	expr	age_sq	no_work	status
0	Anna	23	3.0	529	20.0	W
1	Sam	36	12.0	1296	24.0	W
2	Bill	33	10.0	1089	23.0	W
3	Moica	25	7.0	625	18.0	W
4	Lisa	27	7.0	729	20.0	W

Или столбец из списка значений (столбец `gender`, пол):

In [5]:

```
df['gender'] = [0, 1, 1, 0, 0]  # из списка значений
df
```

Out[5]:

	name	age	expr	age_sq	no_work	status	gender
0	Anna	23	3.0	529	20.0	W	0
1	Sam	36	12.0	1296	24.0	W	1
2	Bill	33	10.0	1089	23.0	W	1
3	Moica	25	7.0	625	18.0	W	0
4	Lisa	27	7.0	729	20.0	W	0