Категориальные данные

```
# создаем категориальную переменную непосредственно из списка
>>> lmh_values = ["low", "high", "medium", "medium", "high"] # здесь три уникальных значения
>>> lmh_cat = pd.Categorical(lmh_values)
>>> lmh_cat
['low', 'high', 'medium', 'medium', 'high']
Categories (3, object): ['high', 'low', 'medium']

 # смотрим категории
>>> lmh_cat.categories
Index(['high', 'low', 'medium'], dtype='object')

# свойство .codes показывает коды (целочисленные значения) для каждого значения категориальной переменной
>>> lmh_cat.codes
array([1, 0, 2, 2, 0], dtype=int8)
```

```
# создаем категориальную переменную с помощью объекта Series и dtype
>>> cat_series = pd.Series(lmh_values, dtype="category")
>>> cat_series
0       low
1      high
2    medium
3    medium
4      high
dtype: category
Categories (3, object): ['high', 'low', 'medium']

# создаем категориальную переменную с помощью метода .astype()
>>> s = pd.Series(lmh_values)
>>> as_cat = s.astype('category')
>>> as_cat
0       low
1      high
2    medium
3    medium
4      high
dtype: category
Categories (3, object): ['high', 'low', 'medium']
```

```
# получаем индекс категориальной переменной
>>> cat_series.cat.categories
Index(['high', 'low', 'medium'], dtype='object')
```

```
# создаем датафрейм из 100 значений
>>> np.random.seed(123456)
>>> values = np.random.randint(0, 100, 5)
>>> bins = pd.DataFrame({"Values": values})
>>> bins
```

	Values
0	65
1	49
2	56
3	43
4	43

```
# разбиваем значения на 10 групп
>>> bins['Group'] = pd.cut(values, range(0, 101, 10))
>>> bins
```

	Values	Group
0	65	(60, 70]
1	49	(40, 50]
2	56	(50, 60]
3	43	(40, 50]
4	43	(40, 50]

```
# проверяем, является ли созданная переменная категориальной
>>> bins.Group

0    (60, 70]
1    (40, 50]
2    (50, 60]
3    (40, 50]
4    (40, 50]
Name: Group, dtype: category
Categories (10, interval[int64]): [(0, 10] < (10, 20] < (20, 30] < (30, 40] ... (60, 70] < (70, 80] < (80, 90] < (90,
100]]
```

Ранжирование заключается в присваивании рангов – от единицы до числа присутствующих в массиве элементов. Для ранжирования применяется метод rank объектов Series и DataFrame; по умолчанию rank обрабатывает связанные ранги, присваивая каждой группе средний ранг:

```
>>> obj = pd.Series([7, -5, 7, 4, 2, 0, 4])
>>> obj
0    7
1   -5
2    7
3    4
4    2
5    0
6    4
dtype: int64
>>> obj.sort_values()
>>> obj.rank()
0    6.5
1    1.0
2    6.5
3    4.5
4    3.0
5    2.0
6    4.5
dtype: float64
>>> bj.rank().sort_values()
```

Способы обработки связанных рангов

Способ	Описание
`'average'`	По умолчанию: одинаковым значениям присвоить средний ранг
`'min'`	Всем элементам группы присвоить минимальный ранг
`'max'`	Всем элементам группы присвоить максимальный ранг
`'first'`	Присваивать ранги в порядке появления значений в наборе данных
`'dense'`	Как `method='min'`, но при переходе к следующей группе элементов с одинаковым рангом ранг всегда увеличивается на 1, а не на количество элементов в группе

rank блокнот

Создание случайной выборки данных

```
# создаем датафрейм, состоящий из 50 строк и 4 столбцов
>>> df = pd.DataFrame(np.random.randn(50, 4))
>>> df.head()

# отбираем три случайные строки
>>> df.sample(n=3)

# отбираем 10% строк
>>> df.sample(frac=0.1)
```

Еще один класс методов служит для извлечения информации о значениях, хранящихся в одномерном объекте Series.

```
>>> obj = pd.Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])
>>> uniques = obj.unique()
>>> uniques
array(['c', 'a', 'd', 'b'], dtype=object)
>>> obj.value_counts()
c    3
a    3
b    2
d    1
dtype: int64
>>> pd.value_counts(obj.values, sort=False)
b    2
a    3
c    3
d    1
dtype: int64
>>> mask = obj.isin(['b', 'c'])
>>> obj[mask]
0    c
5    b
6    b
7    c
8    c
dtype: object
```

Метод unique возвращает массив уникальных значений в Series

Метод value_counts вычисляет объект Series, содержащий частоты встречаемости значений

Метод isin вычисляет булев вектор членства в множестве и может быть полезен для фильтрации набора данных относительно подмножества значений в объекте Series или столбце DataFrame

Выравнивание индексов в объектах DataFrame

При выполнении операций над объектами DataFrame происходит аналогичное выравнивание как для столбцов, так и для индексов:

```
>>> A = pd.DataFrame(rng.randint(0, 20, (2, 2)),
                     columns=list('AB'))
>>> A
```

	A	B
0	1	11
1	5	1

```
>>> B = pd.DataFrame(rng.randint(0, 10, (3, 3)),
                     columns=list('BAC'))
>>> B
```

	B	A	C
0	4	0	9
1	5	8	0
2	9	2	6

```
>>> A + B
```

	A	B	C
0	1.0	15.0	NaN
1	13.0	6.0	NaN
2	NaN	NaN	NaN

```
>>> A.add(B, fill_value=0)
```

	A	B	C
0	1.0	15.0	9.0
1	13.0	6.0	0.0
2	2.0	9.0	6.0

В следующем примере мы заполним отсутствующие значения средним значением всех элементов объекта A (которое вычислим, выстроив сначала значения объекта A в один столбец с помощью функции **stack**):

```
>>> fill = A.stack().mean()

>>> A.add(B, fill_value=fill)
```

	A	B	C
0	1.0	15.0	13.5
1	13.0	6.0	4.5
2	6.5	13.5	10.5

Универсальные функции: выполнение операции между объектами DataFrame и Series

Операции между объектами DataFrame и Series подобны операциям между двумерным и одномерным массивами библиотеки NumPy. Рассмотрим одну из часто встречающихся операций — вычисление разности двумерного массива и одной из его строк:

```
>>> A = rng.randint(10, size=(3, 4))
>>> A
```

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

```
>>> A - A[0]
```

```
array([[ 0,  0,  0,  0],
       [-1, -2,  2,  4],
       [ 3, -7,  1,  4]])
```

Вычитание из двумерного массива одной из его строк выполняется построчно.

В библиотеке Pandas вычитание по умолчанию также происходит построчно:

```
>>> df = pd.DataFrame(A, columns=list('QRST'))
>>> df - df.iloc[0]
```

	Q	R	S	T
0	0	0	0	0
1	-1	-2	2	4
2	3	-7	1	4

```
>>> df.subtract(df['R'], axis=0)
```

	Q	R	S	T
0	-5	0	-6	-4
1	-4	0	-2	2
2	5	0	2	7

```
>>> df
```

	Q	R	S	T
0	3	8	2	4
1	2	6	4	8
2	6	1	3	8

```
>>> halfrow = df.iloc[0, ::2]
>>> halfrow

Q    3
S    2
Name: 0, dtype: int64
```

```
>>> df - halfrow
```

	Q	R	S	T
0	0.0	NaN	0.0	NaN
1	-1.0	NaN	2.0	NaN
2	3.0	NaN	1.0	NaN