Введение в PANDAS

Камзолов Никита, МОЭВМ, 2023

Категориальный тип данных

Категориальный тип данных в Pandas соответствует категориальным переменным в статистике. Категориальная переменная принимает ограниченное и обычно фиксированное число возможных значений (пол, национальность и пр.)

Категориальный тип данных предоставляет возможность использовать ряд методов, удобных для работы с данными.

Создание категориального типа данных

- Явное указание/приведение к типу

```
pd.Series(list('abca'), dtype='category')
```

```
0    a
1    b
2    c
3    a
dtype: category
Categories (3, object): ['a', 'b', 'c']
```

```
pd.Series(list('abca')).astype('category')
```

```
0    a
1    b
2    c
3    a
dtype: category
Categories (3, object): ['a', 'b', 'c']
```

- Объект Categorical

```
category = pd.Categorical(list('abca'), categories=list('abc'))
pd.Series(category)
```

```
0    a
1    b
2    c
3    a
dtype: category
Categories (3, object): ['a', 'b', 'c']
```

- Использование специальных методов

```
series = pd.Series([i for i in range(100)])
pd.cut(series, bins=4)
```

```
0     (-0.099, 24.75]
1     (-0.099, 24.75]
2     (-0.099, 24.75]
3     (-0.099, 24.75]
4     (-0.099, 24.75]
           ...
95      (74.25, 99.0]
96      (74.25, 99.0]
97      (74.25, 99.0]
98      (74.25, 99.0]
99      (74.25, 99.0]
Length: 100, dtype: category
Categories (4, interval[float64, right]): [(-0.099, 24.75] < (24.75, 49.5] < (49.5, 74.25] < (74.25, 99.0]]
```

Основные параметры категорий

- .cat.categories – получить список категорий

```
series = pd.Series(list('abca')).astype('category')
series.cat.categories
```

```
Index(['a', 'b', 'c'], dtype='object')
```

- .cat.ordered – является ли категориальный тип упорядоченным (используется для сравнения)

```
series = pd.Series(list('abca')).astype('category')
series.cat.ordered
```

```
False
```

Основные методы категорий

- .cat.rename_categories()

```
category = pd.Categorical(list('abca'), categories=list('abc'))
series = pd.Series(category)
#значения тоже меняются
series.cat.rename_categories(['renamedA', 'renamedB', 'renamedC'])
```

```
0    renamedA
1    renamedB
2    renamedC
3    renamedA
dtype: category
Categories (3, object): ['renamedA', 'renamedB', 'renamedC']
```

```
category = pd.Categorical(list('abca'), categories=list('abc'))
series = pd.Series(category)
#с помощью словаря
series.cat.rename_categories({'a': 'renamedA', 'b': 'renamedB', 'c': 'renamedC'})
```

```
0    renamedA
1    renamedB
2    renamedC
3    renamedA
dtype: category
Categories (3, object): ['renamedA', 'renamedB', 'renamedC']
```

- .cat.add_categories()

```
category = pd.Categorical(list('abca'), categories=list('abc'))
series = pd.Series(category)
series = series.cat.add_categories(['d'])
series.cat.categories
```

```
Index(['a', 'b', 'c', 'd'], dtype='object')
```

- .cat.remove_categories()

```
series = series.cat.remove_categories(['d'])
series.cat.categories
```

```
Index(['a', 'b', 'c'], dtype='object')
```

- .cat.remove_unused_categories()

```
category = pd.Categorical(list('abca'), categories=list('abcd'))
series = pd.Series(category)
series = series.cat.remove_unused_categories()
series.cat.categories
```

```
Index(['a', 'b', 'c'], dtype='object')
```

- .cat.as_ordered()

```
category = pd.Categorical(list('abca'), categories=list('abcd'))
series = pd.Series(category)
#появились явные отношения
series = series.cat.as_ordered()
series
```

```
0    a
1    b
2    c
3    a
dtype: category
Categories (4, object): ['a' < 'b' < 'c' < 'd']
```

- .cat.as_unordered()

```
series = series.cat.as_unordered()
series
```

```
0    a
1    b
2    c
3    a
dtype: category
Categories (4, object): ['a', 'b', 'c', 'd']
```

- .cat.reorder_categories()

```
series = series.cat.as_ordered()
series = series.cat.reorder_categories(['b', 'c', 'a', 'd'])
series
```

```
0    a
1    b
2    c
3    a
dtype: category
Categories (4, object): ['b' < 'c' < 'a' < 'd']
```

Упорядоченные категории поддерживают операции сравнения, методы сортировки и методы min(), max().

Все про категориальный тип данных - тык

Строковый тип данных

В Pandas есть отдельные методы, которые характерны для типа данных StringDtype. Чтобы задать этот тип у Series достаточно явно указать dtype = “string”

```
series = pd.Series(['ab', 'cd', 'ef'], dtype='string')
series
```

```
0    ab
1    cd
2    ef
dtype: string
```

Основные методы строкового типа данных

- .str.lower()

```
series = pd.Series(['AA', 'BB', 'CC'], dtype='string')
series.str.lower()
```

```
0    aa
1    bb
2    cc
dtype: string
```

- .str.upper()

```
series = pd.Series(['aa', 'bb', 'cc'], dtype='string')
series.str.upper()
```

```
0    AA
1    BB
2    CC
dtype: string
```

- .str.len()

```
series = pd.Series(['a', 'bb', 'ccc'], dtype='string')
series.str.len()
```

```
0    1
1    2
2    3
dtype: Int64
```

- .str.strip(), .str.lstrip(), .str.rstrip()

```
series = pd.Series([' A ', ' B', 'C '], dtype='string')
series.str.strip().values
```

```
<StringArray>
['A', 'B', 'C']
Length: 3, dtype: string
```

```
series.str.rstrip().values
```

```
<StringArray>
[' A', ' B', 'C']
Length: 3, dtype: string
```

```
series.str.lstrip().values
```

```
<StringArray>
['A ', 'B', 'C ']
Length: 3, dtype: string
```

- .str.capitalize()

```
series = pd.Series(['aa', 'bb', 'cc'], dtype='string')
series.str.capitalize()
```

```
0    Aa
1    Bb
2    Cc
dtype: string
```

- .str.endswith()

```
series = pd.Series(['ab', 'bb', 'cc'], dtype='string')
series.str.endswith('b')
```

```
0     True
1     True
2    False
dtype: boolean
```

- .str.find()

```
series = pd.Series(['aba', 'bbb', 'ccb', 'xxx'], dtype='string')
#поиск индекса вхождения подстроки в строку
series.str.find('b')
```

```
0     1
1     0
2     2
3    -1
dtype: Int64
```

- .str.findall()

```
series = pd.Series(['abab', 'bbb', 'ccb', 'xxx'], dtype='string')
series.str.findall('b')
```

```
0       [b, b]
1    [b, b, b]
2          [b]
3           []
dtype: object
```

- .str.get()

```
series = pd.Series(['aa', 'b', 'cc'], dtype='string')
series.str.get(0)
```

```
0    a
1    b
2    c
dtype: string
```

```
series.str.get(1)
```

```
0       a
1    <NA>
2       c
dtype: string
```

- .str.removeprefix()

```
series = pd.Series(['aa', 'ab', 'cc'], dtype='string')
series.str.removeprefix('a')
```

```
0     a
1     b
2    cc
dtype: string
```

- .str.removesuffix()

```
series = pd.Series(['aab', 'ab', 'cc'], dtype='string')
series.str.removesuffix('b')
```

```
0    aa
1     a
2    cc
dtype: string
```

- .str.repeat()

```
series = pd.Series(['a', 'b', 'c'], dtype='string')
series.str.repeat(3)
```

```
0    aaa
1    bbb
2    ccc
dtype: string
```

- .str.replace()

```
series = pd.Series(['a', 'b', 'c'], dtype='string')
series.str.replace('a', 'A')
```

```
0    A
1    b
2    c
dtype: string
```

- .str.isdigit()

```
series = pd.Series(['a', '1', '2'], dtype='string')
series.str.isdigit()
```

```
0    False
1     True
2     True
dtype: boolean
```

- .str.split()

```
series = pd.Series(["a.b.c", "c.d.e", np.nan, "f.g.h"], dtype="string")
series.str.split('.', expand=True)
```

	0	1	2
0	a	b	c
1	c	d	e
2	<NA>	<NA>	<NA>
3	f	g	h

В большинстве представленных методов также можно пользоваться регулярными выражениями. Еще больше методов и примеров можно найти здесь - тык

DataFrame

Индексированная таблица

Создание DataFrame

- Из списка Python/numpy (построчно)

```
pd.DataFrame([[1, 2, 3], [4, 5, 6]], index=list('ab'), columns=list('abc'))
```

	a	b	c
a	1	2	3
b	4	5	6

```
pd.DataFrame([np.array([1, 2, 3]), np.array([4, 5, 6])], index=list('ab'))
```

	0	1	2
a	1	2	3
b	4	5	6

- Из словаря

```
#значение ключа становится наименованием столбца
pd.DataFrame({'a': [1, 4], 'b': [2, 5], 'c': [3, 6]}, index=list('ab'))
```

	a	b	c
a	1	2	3
b	4	5	6

- Из Series

```
#Аналогично спискам
pd.DataFrame([pd.Series([1, 2, 3]), pd.Series(list('ab'))])
```

	0	1	2
0	1	2	3.0
1	a	b	NaN

Основные параметры DataFrame

- .index – аналогично Series

```
df = pd.DataFrame([[1, 2, 3], [4, 5, 6]], index=list('ab'))
df.index
```

```
Index(['a', 'b'], dtype='object')
```

- .values – все значения в виде двумерного numpy array.

```
df = pd.DataFrame([[1, 2, 3], [4, 5, 6]], index=list('ab'))
df.values
```

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

- .dtypes – возвращает Series, где индексы – наименования столбцов, а значения – тип столбца

```
df = pd.DataFrame({'a': [1, 2], 'b': ['c', 'd']})
df.dtypes
```

```
a     int64
b    object
dtype: object
```

- Размер и форма

```
df = pd.DataFrame([[1, 2], [3, 4], [5, 6]])
df
```

	0	1
0	1	2
1	3	4
2	5	6

```
len(df)#размер столбца
```

```
3
```

```
df.size#полное количество элементов
```

```
6
```

```
df.shape#кортеж: количество строк; столбцов
```

```
(3, 2)
```

Работа с индексами

- .set_index() – переносит столбец в индекс

```
df = pd.DataFrame({'kcal': [130, 120, 80, 70],
                   'price': [100, 200, 300, 111],
                   'type': ['fruit', 'vegetable', 'fruit', 'vegetable']})
df = df.set_index('type')
df
```

type	kcal	price
fruit	130	100
vegetable	120	200
fruit	80	300
vegetable	70	111

- .reset_index() – сбросить индекс и перенести в столбец

```
df.reset_index()
```

	type	kcal	price
0	fruit	130	100
1	vegetable	120	200
2	fruit	80	300
3	vegetable	70	111

Ограничение размера

Методы аналогичные Series

```
df = pd.DataFrame({'a': [1, 2, 3, 4, 5, 6, 7, 8, 9],
                   'b': [1, 2, 3, 4, 5, 6, 7, 8, 9]})
df.head(4)
```

	a	b
0	1	1
1	2	2
2	3	3
3	4	4

```
df.take((1, 4, 5))
```

	a	b
1	2	2
4	5	5
5	6	6

```
df.tail(4)
```

	a	b
5	6	6
6	7	7
7	8	8
8	9	9

Полезные методы DataFrame

- Почти все основные методы Series применимы и к DataFrame (применяется ко всем столбцам и выводится информация относительно столбцов)

```
df = pd.DataFrame({'a': [1, 2, 3, 4, 5, 6, 7, 8, 9],
                   'b': [10, 11, 12, 13, 14, 15, 16, 17, 18]})
df.describe()
```

	a	b
count	9.000000	9.000000
mean	5.000000	14.000000
std	2.738613	2.738613
min	1.000000	10.000000
25%	3.000000	12.000000
50%	5.000000	14.000000
75%	7.000000	16.000000
max	9.000000	18.000000

```
df.sum()
```

```
a     45
b    126
dtype: int64
```

```
df.mean()
```

```
a     5.0
b    14.0
dtype: float64
```

- .value_counts() – считает число уникальных сочетаний всех столбцов

```
df = pd.DataFrame({'a': [1, 2, 3, 3, 4],
                   'b': [1, 2, 3, 3, 4]})
df.value_counts()
```

```
a  b
3  3    2
1  1    1
2  2    1
4  4    1
dtype: int64
```

Получение данных из DataFrame

- Оператор []

```
df = pd.DataFrame({'a': [1, 2, 3, 4, 5], 'b': list('cdefg')})
df['b'] #получение столбца
```

```
0    c
1    d
2    e
3    f
4    g
Name: b, dtype: object
```

```
df[['a', 'b']]
```

	a	b
0	1	c
1	2	d
2	3	e
3	4	f
4	5	g

- .loc() – позволяет отбирать не только по индексам, но и по столбцам

```
df.loc[0, 'b']
```

```
'c'
```

```
df.loc[0, ['a', 'b']]
```

```
a    1
b    c
Name: 0, dtype: object
```

- .iloc() – аналогично, может отбирать по позиции столбца.

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

```
'e'
```

```
df.iloc[2, [0, 1]]
```

```
a    3
b    e
Name: 2, dtype: object
```

Срезы DataFrame

- Срезы по позиции
  - Оператор []

```
df = pd.DataFrame({'numbers': [10, 2, 4, 6, 7],
                   'words': ['ab', 'cd', 'ef', 'gh', 'ij']},
                  index = list('abcde'))
df[0:2] #Правая граница не включается
```

	numbers	words
a	10	ab
b	2	cd

  - .iloc() – позволяет делать срезы сразу по 2-ум измерениям

```
df = pd.DataFrame({'a': [10, 2, 4, 6, 7],
                   'b': ['ab', 'cd', 'ef', 'gh', 'ij'],
                   'c': [0.1, 0.6, 0.8, 0.3, 0.5]},
                  index = list('abcde'))
df.iloc[0:2, 0:2]
```

	a	b
a	10	ab
b	2	cd

- Срезы по индексам/столбцам

```
df = pd.DataFrame({'a': [10, 2, 4, 6, 7],
                   'b': ['ab', 'cd', 'ef', 'gh', 'ij'],
                   'c': [0.1, 0.6, 0.8, 0.3, 0.5]},
                  index = list('abcde'))
df.loc['a':'b', 'b':'c'] #границы включаются
```

	b	c
a	ab	0.1
b	cd	0.6

Добавление столбцов и строк

- Добавление столбца
  - Список значений

```
df = pd.DataFrame({'a': [1, 2], 'b': ['c', 'd']})
df['c'] = [0.1, 0.2]
df
```

	a	b	c
0	1	c	0.1
1	2	d	0.2

  - Series

```
df = pd.DataFrame({'a': [1, 2], 'b': ['c', 'd']})
df['c'] = pd.Series([0.1, 0.2], index=[0,2])
df#требуется совпадение индексов
```

	a	b	c
0	1	c	0.1
1	2	d	NaN

- Добавление строки

```
df.loc[2] = [3, 'e']
df
```

	a	b
0	1	c
1	2	d
2	3	e

```
df.loc[2] = [3, 'e', 4] #ошибка
df
```

Удаление столбцов и строк

- Удаление столбцов
  - .pop() – удаляет in-place и возвращает удаленный столбец

```
df = pd.DataFrame({'a': [10, 2], 'b': ['ab', 'cd'], 'c': [0.1, 0.6]})
series = df.pop('b') #в series записан столбец b
df
```

	a	c
0	10	0.1
1	2	0.6

  - .drop(axis=1) – удаляет столбец или список столбцов.

```
df = pd.DataFrame({'a': [10, 2], 'b': ['ab', 'cd'], 'c': [0.1, 0.6]})
df.drop(
    ['a', 'b', 'd'],
    axis = 1,#удаление столбца
    inplace = True,
    errors = 'ignore'#игнорировать ошибку, если нет столбца с указанным именем
)
df
```

	c
0	0.1
1	0.6

- Удаление строк
  - .drop(axis=0)

```
df = pd.DataFrame({'a': [10, 2], 'b': ['ab', 'cd'], 'c': [0.1, 0.6]})
df = df.drop(0, axis = 0)#удаление строки
df
```

	a	b	c
1	2	cd	0.6

Фильтрация данных DataFrame

Фильтрация строк производится аналогично Series.

```
df = pd.DataFrame({'a': [10, 2, 4, 6, 8], 'c': [0.1, 0.6, 0.8, 0.13, 0.5]})
df[(df['a'] > 4) & (df['c'] <= 0.5)]
```

	a	c
0	10	0.10
3	6	0.13
4	8	0.50

- Оператор ~.

```
df = pd.DataFrame({'a': [10, 2, 4, 6, 8],
                   'c': [0.1, 0.6, 0.8, 0.13, 0.5]},
                  index=list('abcde'))
#отфильтровать индексы так, чтобы в итоговой структуре
#НЕ БЫЛО индексов из списка недопустимых индексов
df[~df.index.isin(list('abc'))]
```

	a	c
d	6	0.13
e	8	0.50

Импортирование/экспортирование данных

Формат	Чтение	Запись
csv	pd.read_csv	df.to_csv
json	pd.read_json	df.to_json
Fixed-Width Text File	d.read_fwf	-
html	pd.read_html	df.to_html
latex	-	df.to_latex
xml	pd.read_xml	df.to_xml
excel	pd.read_excel	df.to_excel
hdf5	pd.read_hdf	df.to_hdf
sql	pd.read_sql	df.to_sql
Parquet Format	pd.read_parquet	df.to_parquet
ORC	pd.read_orc	df.to_orc
Stata	pd.read_stata	df.to_stata

Аргументы функций импорта

- filepath_or_buffer – путь к файлу, URL или любой другой объект с методом read()
- sep – строка-разделитель для чтения из текстовых файлов
- header – номер строки заголовка (по умолчанию 0)
- names – список имен столбцов (тогда header = None)
- index_col – номера или названия столбцов, которые будут использоваться, как индекс DataFrame’a
- usecols – список строк или функция для отбора конкретных столбцов
- dtype – словарь с соответствием столбца и типа данных
- skiprows – список строк, которые необходимо пропустить при чтении
- skipfooter – количество строк, которые отбрасываем из конца файла
- nrows – количество строк для считывания
- na_values – какие строки необходимо считать, как NaN
- И др.

Сортировка DataFrame

- sort_values()
  - axis – ось подлежащая сортировке (‘index’/0 (сортировка строк) или ‘columns’/1 (сортировка столбцов)) (по умолчанию 0)
  - by – имя или список имен для сортировки (если axis=‘index’, то наименование столбца или имя индекса, если axis=‘columns’, то индекс)
  - Остальные параметры аналогичны Series.

```
df = pd.DataFrame({'a': [10, 2, 4, 6, 8],
                   'c': [0.1, 0.6, 0.8, 0.13, 0.5]},
                  index=list('abcde'))
#сортировка столбца 'c' по строкам
df.sort_values(by='c', axis='index')
```

	a	c
a	10	0.10
d	6	0.13
e	8	0.50
b	2	0.60
c	4	0.80

```
df = pd.DataFrame([[10, 2, 4, 6, 8],
                   [0.1, 0.6, 0.8, 0.13, 0.5]],
                  index=list('ab'),
                  columns=list('abcde'))
#сортировка строки с индексом 'b' по столбцам
df.sort_values(by='b', axis='columns')
```

	a	d	e	b	c
a	10.0	6.00	8.0	2.0	4.0
b	0.1	0.13	0.5	0.6	0.8

- .sort_index()
  - axis – ось, по которой производится сортировка (‘index’ или ‘columns’)

```
df = pd.DataFrame([[10, 2, 4, 6, 8],
                   [0.1, 0.6, 0.8, 0.13, 0.5]],
                  index=list('ab'),
                  columns=list('cdbea'))
#сортировка столбцов в лексикографическом порядке
df.sort_index(axis='columns')
```

	a	b	c	d	e
a	8.0	4.0	10.0	2.0	6.00
b	0.5	0.8	0.1	0.6	0.13

```
df = pd.DataFrame({'a': [10, 2, 4, 6, 8],
                   'c': [0.1, 0.6, 0.8, 0.13, 0.5]},
                  index=list('cdeba'))
#сортировка строк в лексикографическом порядке
df.sort_index(axis='index')
```

	a	c
a	8	0.50
b	6	0.13
c	10	0.10
d	2	0.60
e	4	0.80

Очистка данных DataFrame

- .dropna() – удаление столбца/строки, содержащей NaN значение
  - axis – ‘index’/0(удаление строк) или ‘columns’/1(удаление столбцов) (по умолчанию 0)
  - how – ‘any’(удалить, если хотя бы одно значение NaN) или ‘all’ (удалить, если все значения NaN) (по умолчанию ‘any’)
  - thresh – сколько не NaN значений требуется, чтобы избежать удаления (не комбинируется с how)
  - subset – метки вдоль другой оси, которые следует учитывать, например, если вы удаляете строки, это будет список столбцов, которые стоит проверять на наличие NaN.
  - inplace – ‘in-place’ очистка от NaN
  - ignore_index – заменить в результирующей таблице индекс на последовательность 0..n

```
df = pd.DataFrame({'a': [10, 2, 4, 6, np.nan],
                   'c': [0.1, np.nan, 0.8, np.nan, 0.5]},
                  index=list('cdeba'))
#удаляем столбцы, где больше одного значения NaN
df.dropna(axis=1, thresh=df.shape[0] - 1)
```

	a
c	10.0
d	2.0
e	4.0
b	6.0
a	NaN

```
df = pd.DataFrame([[np.nan, 1, 2],
                   [1, 2, np.nan],
                   [1, np.nan, 2]],
                  index=list('abc'),
                  columns=list('abc'))
#удаляем все строки, где есть np.nan, но не учитываем столбец 'c'
df.dropna(axis=0, subset=['a', 'b'])
```

	a	b	c
b	1.0	2.0	NaN

- .fillna() – заполнение NaN другим значением
  - value – значение, которым заполняются NaN. Можно также передавать словарь, Series или DataFrame, указывая какое значение использовать для каждого индекса (для Series) или столбца (для DataFrame).
  - method – альтернативный метод заполнения. Если ‘bfil’ – используется следующее достоверное наблюдение, чтобы заполнить NaN, если ‘ffil’ – распространяется последнее достоверное наблюдение (по умолчанию None). Не комбинируется с value.
  - axis – ось вдоль которой заполняем значения (используется при заполнении с параметром method != None), ‘index’/0 или ‘columns’/1 (по умолчанию 0).
  - inplace – ‘in-place’ заполнение NaN
  - limit – если указан method, то это максимальное количество NaN значений, на которое может распространиться достоверное наблюдение. Если указан value, то это максимальное число значений, которое может заполнить метод вдоль указанной оси.

```
df = pd.DataFrame([[np.nan, 1, np.nan],
                   [np.nan, 1, 2],
                   [1, np.nan, 2]])
#заполняем NaN следующим валидным значением в СТОЛБЦЕ
df.fillna(method="bfill", axis=1)
```

	0	1	2
0	1.0	1.0	NaN
1	1.0	1.0	2.0
2	1.0	2.0	2.0

```
df = pd.DataFrame([[np.nan, 1, np.nan],
                   [np.nan, 1, 2],
                   [1, np.nan, 2]])
#заполняем NaN своим значением для каждого столбца
#но не более 1-го раза за столбец
df.fillna({0:2, 1:3, 2:4}, limit=1)
```

	0	1	2
0	2.0	1.0	4.0
1	NaN	1.0	2.0
2	1.0	3.0	2.0

- .drop_duplicates() – удаление дублирующихся строк.
  - subset – столбцы которые следует учитывать при поиске дубликатов (по умолчанию – все столбцы)
  - keep – какое из дублирующихся вхождений необходимо сохранить. ‘first’ – первое, ‘last’ – последнее, False – удалить все.
  - inplace – ‘in-place’ очистка от дубликатов
  - ignore_index – заменить в результирующей таблице индекс на последовательность 0..n

```
df = pd.DataFrame([[1, 2, 3], [1, 2, 3], [1, 2, 4], [1, 3, 5]])
#удалить дубликаты (дубликаты рассчитываются по первым двум столбцам)
df.drop_duplicates(subset=[0, 1])
```

	0	1	2
0	1	2	3
3	1	3	5

```
df = pd.DataFrame([[1, 2, 3], [1, 2, 3], [1, 2, 4], [1, 3, 5]])
#берем последнее вхождение
df.drop_duplicates(subset=[0, 1], keep='last')
```

	0	1	2
2	1	2	4
3	1	3	5