2.3 Введение в pandas

Pandas - расширение NumPy (структурированные массивы). Строки и столбцы индексируются метками, а не только числовыми значениями.

В pandas три основные структуры: Series, DataFrame, Index.

Series

```
import numpy as np
import pandas as pd

data = pd.Series([0.25, 0.5, 0.75, 1])
print(data)
# => 0    0.25
#    1    0.50
#    2    0.75
#    3    1.00
# dtype: float64

print(type(data))
# => <class 'pandas.core.series.Series'

print(data.values)
# => [0.25 0.5  0.75 1.  ]
print(type(data.values))
# => <class 'numpy.ndarray'>

print(data.index)
# => RangeIndex(start=0, stop=4, step=1)
print(type(data.index))
# => <class 'pandas.core.indexes.range.RangeIndex'>
```

Обращение к элементами массива

```
import numpy as np
import pandas as pd

data = pd.Series([0.25, 0.5, 0.75, 1])

print(data[0])
# => 0.25

print(data[1:3])
# => 1    0.50
#    2    0.75
# dtype: float64
```

Явное определение индексов как массив

```
import numpy as np
import pandas as pd

data = pd.Series([0.25, 0.5, 0.75, 1], index=['a', 'b', 'c', 'd

print(data)
# => a    0.25
#    b    0.50
#    c    0.75
#    d    1.00
# dtype: float64

print(data['a'])
# => 0.25

print(data['b':'d'])
# => b    0.50
```

```
#    c    0.75
#    d    1.00
# dtype: float64

print(type(data.index))
# => <class 'pandas.core.indexes.base.Index'>

data = pd.Series([0.25, 0.5, 0.75, 1], index=[1, 10, 7, 'd'])

print(data)
# => 1    0.25
#    10   0.50
#    7    0.75
#    d    1.00
# dtype: float64

print(data[1])
# => 0.25

print(data[10:'d')
# => 10   0.50
#    7    0.75
#    d    1.00
# dtype: float64
```

Определение индекса как словаря

```
import numpy as np
import pandas as pd

population_dict = {
    'city_1': 1001,
    'city_2': 1002,
    'city_3': 1003,
    'city_4': 1004,
```

```
    'city_5': 1005,
}

population = pd.Series(population_dict)
print(population)
# => city_1    1001
#    city_2    1002
#    city_3    1003
#    city_4    1004
#    city_5    1005
# dtype: int64

print(population['city_4'])
# => 1004

print(population['city_4':'city_5'])
# => city_4    1004
#    city_5    1005
# dtype: int64
```

Для создания Series можно использовать

1. списки Python или массивы NumPy
2. скалярные значение
3. словари

Задание для самостоятельной работы

1. Привести различные способы задания объектов типа Series

DataFrame

Двумерный массив с явно определенными индексами. Последовательность “согласованных” по индексам объектов Series.

```
import numpy as np
import pandas as pd

population_dict = {
    'city_1': 1001,
    'city_2': 1002,
    'city_3': 1003,
    'city_4': 1004,
    'city_5': 1005,
}

area_dict = {
    'city_1': 9991,
    'city_2': 9992,
    'city_3': 9993,
    'city_4': 9994,
    'city_5': 9995,
}

population = pd.Series(population_dict)
area = pd.Series(area_dict)

print(population)
# => city_1    1001
#    city_2    1002
#    city_3    1003
#    city_4    1004
#    city_5    1005
# dtype: int64

print(area)
# => city_1    9991
#    city_2    9992
#    city_3    9993
#    city_4    9994
```

```
#    city_5    9995
# dtype: int64

states = pd.DataFrame({
    'population1': population,
    'area1': area
})

print(states)
# =>       population1  area1
#    city_1        1001  9991
#    city_2        1002  9992
#    city_3        1003  9993
#    city_4        1004  9994
#    city_5        1005  9995

print(states.values)
# => [[1001 9991]
#     [1002 9992]
#     [1003 9993]
#     [1004 9994]
#     [1005 9995]]

print(states.index)
# => Index(['city_1', 'city_2', 'city_3', 'city_4', 'city_5'], c

print(states.columns)
# => Index(['population', 'area'], dtype='object')

print(type(states.values))
# => <class 'numpy.ndarray'>

print(type(states.index))
# => <class 'pandas.core.indexes.base.Index'>

print(type(states.columns))
```

```
# => <class 'pandas.core.indexes.base.Index'>

print(states['area1'])
# => city_1    9991
#    city_2    9992
#    city_3    9993
#    city_4    9994
#    city_5    9995
# Name: area1, dtype: int64
```

DataFrame. Способы создания

1. через объекты Series
2. списки словарей
3. словари объектов Series
4. двумерный массив NumPy
5. структурированный массив NumPy

Задание для самостоятельной работы.

2. Привести различные способы создания объектов DataFrame

Index

Способ организации ссылки на данные объектов Series и DataFrame. Index неизменяем, упорядочен, является мультимножеством (могут быть повторяющиеся значения)

```
import numpy as np
import pandas as pd

ind = pd.Index([2, 3, 5, 7, 11])
```

```
print(ind)
# => Index([2, 3, 5, 7, 11], dtype='int64')

print(ind[1])
# => 3

print(ind[::2])
# => Index([2, 5, 11], dtype='int64')

ind[5] = 1
# => raise TypeError("Index does not support mutable operations'
```

Index следует соглашениям объекта set (pyhon)

Пересечение индексов

```
import numpy as np
import pandas as pd

indA = pd.Index([1, 2, 3, 4, 5])
indB = pd.Index([2, 3, 4, 5, 6])

print(indA.intersection(indB))
# => Index([2, 3, 4, 5], dtype='int64')
```

Выборка данных из Series

```
import numpy as np
import pandas as pd

data = pd.Series([0.25, 0.5, 0.75, 1], index=['a', 'b', 'c', 'd

print('a' in data)
# => True
```

```
print('z' in data)
# => False

print(data.keys())
# => Index(['a', 'b', 'c', 'd'], dtype='object')

print(list(data.items()))
# => [('a', 0.25), ('b', 0.5), ('c', 0.75), ('d', 1.0)]

data['a'] = 100
print(data)
# => a    100.00
#    b    0.50
#    c    0.75
#    d    1.00
# dtype: float64

data['z'] = 1000
print(data)
# => a    100.00
#    b    0.50
#    c    0.75
#    d    1.00
#    z    1000.00
# dtype: float64
```

Series как одномерный массив

```
import numpy as np
import pandas as pd

data = pd.Series([0.25, 0.5, 0.75, 1], index=['a', 'b', 'c', 'd

print(data['a':'c'])
```

```
# => a    0.25
#    b    0.50
#    c    0.75
# dtype: float64

print(data[0:2])
# => a    0.25
#    b    0.50
# dtype: float64

print(data[data > 0.5])
# => c    0.75
#    d    1.00
# dtype: float64

print(data[(data > 0.5) & (data < 1)])
# => c    0.75
# dtype: float64

print(data['a', 'd'])
# => a    0.25
#    d    1.00
# dtype: float64
```

Особенность (проблема)

```
import numpy as np
import pandas as pd

data = pd.Series([0.25, 0.5, 0.75, 1], index=[1, 3, 10, 15])

# Выбирается явное значение а не индекс
print(data[1])
# => 0.25
```

Атрибуты - индексаторы

```
import numpy as np
import pandas as pd

data = pd.Series([0.25, 0.5, 0.75, 1], index=[1, 3, 10, 15])

# Обращение по индексу
print(data.iloc[1])
# => 0.25

# Обращение к номеру индекса
print(data.iloc[1])
# => 0.5
```

Выборка данных из DataFrame

```
import numpy as np
import pandas as pd

population_dict = {
    'city_1': 1001,
    'city_2': 1002,
    'city_3': 1003,
    'city_4': 1004,
    'city_5': 1005,
}

area_dict = {
    'city_1': 9991,
    'city_2': 9992,
    'city_3': 9993,
    'city_4': 9994,
    'city_5': 9995,
```

```
}

population = pd.Series(population_dict)
area = pd.Series(area_dict)

data = pd.DataFrame({
    'area1': area,
    'population1': population
})

print(data)
# =>        area1  population1
#    city_1  9991        1001
#    city_2  9992        1002
#    city_3  9993        1003
#    city_4  9994        1004
#    city_5  9995        1005

print(data['area1'])
print(data.area1)
# => city_1  9991
#    city_2  9992
#    city_3  9993
#    city_4  9994
#    city_5  9995

print(data.population1 is data['population1'])
# => True

print(data.population is data['population'])
# => False

data['new'] = data['area1']
print(data)
# =>         area1  population1   new
#    city_1   9991         1001  9991
```

```
#    city_2   9992         1002  9992
#    city_3   9993         1003  9993
#    city_4   9994         1004  9994
#    city_5   9995         1005  9995

data['new1'] = data['area1'] / data['population1']
print(data)
# =>         area1  population1   new      new1
#    city_1   9991         1001  9991  9.981019
#    city_2   9992         1002  9992  9.972056
#    city_3   9993         1003  9993  9.963111
#    city_4   9994         1004  9994  9.954183
#    city_5   9995         1005  9995  9.945274
```

Двумерный NumPy массив

```
import numpy as np
import pandas as pd

population_dict = {
    'city_1': 1001,
    'city_2': 1002,
    'city_3': 1003,
    'city_4': 1004,
    'city_5': 1005,
}

area_dict = {
    'city_1': 9991,
    'city_2': 9992,
    'city_3': 9993,
    'city_4': 9994,
    'city_5': 9995,
}
```

```
population = pd.Series(population_dict)
area = pd.Series(area_dict)

data = pd.DataFrame({
    'area1': area,
    'population1': population
})

print(data)
# =>        area1  population1
#    city_1  9991        1001
#    city_2  9992        1002
#    city_3  9993        1003
#    city_4  9994        1004
#    city_5  9995        1005

print(data.values)
# => [[9991 1001]
#     [9992 1002]
#     [9993 1003]
#     [9994 1004]
#     [9995 1005]]

# Транспонирование
print(data.T)
# =>              city_1  city_2  city_3  city_4  city_5
#    area1          9991    9992    9993    9994    9995
#    population1    1001    1002    1003    1004    1005

print(data['area1'])
# => city_1    9991
#    city_2    9992
#    city_3    9993
#    city_4    9994
#    city_5    9995
# Name: area1, dtype: int64
```

```
# Обращение к строке
print(data.values[0])
# => [9991 1001]

print(data.values[0:3])
# => [[9991 1001]
#     [9992 1002]
#     [9993 1003]]
```

Атрибуты - индексаторы

```
import numpy as np
import pandas as pd

population_dict = {
    'city_1': 1001,
    'city_2': 1002,
    'city_3': 1003,
    'city_4': 1004,
    'city_5': 1005,
}

area_dict = {
    'city_1': 9991,
    'city_2': 9992,
    'city_3': 9993,
    'city_4': 9994,
    'city_5': 9995,
}

population = pd.Series(population_dict)
area = pd.Series(area_dict)

data = pd.DataFrame({
```

```
    'area1': area,
    'population1': population,
    'population': population
})

print(data)
# =>         area1  population1  population
#    city_1   9991         1001        1001
#    city_2   9992         1002        1002
#    city_3   9993         1003        1003
#    city_4   9994         1004        1004
#    city_5   9995         1005        1005

#              [столбцец, строка]
print(data.iloc[:3, 1:2])
# =>         population1
#    city_1         1001
#    city_2         1002
#    city_3         1003

print(data.loc[:'city_4', 'population1':'population'])
# =>         population1  population
#    city_1         1001        1001
#    city_2         1002        1002
#    city_3         1003        1003
#    city_4         1004        1004

print(data.loc[data['population'] > 1002, ['area1', 'population
# =>         area1  population
#    city_3   9993        1003
#    city_4   9994        1004
#    city_5   9995        1005

data.iloc[0,2] = 999999
print(data)
# =>         area1  population1  population
```

```
#    city_1   9991         1001      999999
#    city_2   9992         1002        1002
#    city_3   9993         1003        1003
#    city_4   9994         1004        1004
#    city_5   9995         1005        1005
```

Универсальные функции

```
import numpy as np
import pandas as pd

rng = np.random.default_rng()
s = pd.Series(rng.integers(0, 10, 4))

print(s)
# => 0    8
#    1    6
#    2    3
#    3    8
# dtype: int64

print(np.exp(s))
# => 0      54.598150
#    1     148.413159
#    2    1096.633158
#    3    2980.957987
# dtype: float64
```

Данные, у которых ключи не совпадают

```
import numpy as np
import pandas as pd

population_dict = {
```

```
    'city_1': 1001,
    'city_2': 1002,
    'city_3': 1003,
    'city_41': 1004,
    'city_51': 1005,
}

area_dict = {
    'city_1': 9991,
    'city_2': 9992,
    'city_3': 9993,
    'city_42': 9994,
    'city_52': 9995,
}

population = pd.Series(population_dict)
area = pd.Series(area_dict)

data = pd.DataFrame({
    'area1': area,
    'population1': population,
})

# NaN - not a number
print(data)
# =>           area1  population1
#    city_1   9991.0       1001.0
#    city_2   9992.0       1002.0
#    city_3   9993.0       1003.0
#    city_41     NaN       1004.0
#    city_42  9994.0          NaN
#    city_51     NaN       1005.0
#    city_52  9995.0          NaN
```

Задание для самостоятельной работы

3. Объединить два объекта Series с неодинаковыми множествами ключей (индексами) так чтобы вместо NaN было установлено значение 1.

Объединение DataFrame

```
import numpy as np
import pandas as pd

rng = np.random.default_rng()

dfA = pd.DataFrame(rng.integers(0, 10, (2, 2)), columns=['a', 'b
dfB = pd.DataFrame(rng.integers(0, 10, (3, 3)), columns=['a', 'b

print(dfA)
# =>    a  b
#    0  0  3
#    1  9  2

print(dfB)
# =>    a  b  c
#    0  3  1  7
#    1  5  4  8
#    2  5  7  3

print(dfA + dfB)
# =>      a    b   c
#   0  12.0  2.0 NaN
#   1   9.0  3.0 NaN
#   2   NaN  NaN NaN
```

Транслирование DataFrame

```
import numpy as np
import pandas as pd
```

```
rng = np.random.default_rng(1)

A = rng.integers(0, 10, (3, 4))
print(A)
# => [[4 5 7 9]
#     [0 1 8 9]
#     [2 3 8 4]]

print(A[0])
# => [4 5 7 9]

print(A - A[0])
# => [[ 0  0  0  0]
#     [-4 -4  1  0]
#     [-2 -2  1 -5]]

df = pd.DataFrame(A, columns=['a', 'b', 'c', 'd'])
print(df)
# =>    a  b  c  d
#    0  4  5  7  9
#    1  0  1  8  9
#    2  2  3  8  4

print(df.iloc[0])
# => a    4
#    b    5
#    c    7
#    d    9
#Name: 0, dtype: int64

print(df - df.iloc[0])
# =>    a  b  c  d
#    0  0  0  0  0
#    1 -4 -4  1  0
#    2 -2 -2  1 -5
```

```
print(df.iloc[0, ::2])
# => a    4
#    c    7
# Name: 0, dtype: int64

print(df - df.iloc[0, ::2])
# =>      a   b    c   d
#    0  0.0 NaN  0.0 NaN
#    1 -4.0 NaN  1.0 NaN
#    2 -2.0 NaN  1.0 NaN
```

Задание для самостоятельной работы

4. Переписать пример с транслированием для DataFrame так, чтобы вычитание происходило не по строкам, а по столбцам

NaN - not a number

NA-значения (not available value) : NaN, null

Pandas. Два способа хранения отсутствующих значения

1. Способ: индикаторы NaN, None

None - объект. А если это объект, то его использование может привести к накладным расходам. Не работает с sum, min (агрегирующие операторы)

Через NumPy

```
import numpy as np
import pandas as pd

val1 = np.array([1, 2, 3,])
print(val1.sum())
# => 6
```

```
val2 = np.array([1, None, 2, 3])
print(val2.sum())
# => TypeError: unsupported operand type(s) for +: 'int' and 'No

val3 = np.array([1, np.nan, 2, 3])
print(val3.sum())
print(np.sum(val3))
# => nan

print(np.nansum(val3))
# => 6.0
```

Через Pandas

```
import numpy as np
import pandas as pd

x = pd.Series(range(10), dtype=int)
print(x)
# => 0    0
#    1    1
#    2    2
#    3    3
#    4    4
#    5    5
#    6    6
#    7    7
#    8    8
#    9    9

x[0] = None
x[1] = np.nan

print(x)
```

```
# => 0    NaN
#    1    NaN
#    2    2
#    3    3
#    4    4
#    5    5
#    6    6
#    7    7
#    8    8
#    9    9

x1 = pd.Series(['a', 'b', 'c'])
print(x1)
# => 0    a
#    1    b
#    2    c

x1[0] = None
x1[1] = np.nan

print(x1)
# => 0    None
#    1    NaN
#    2    c

x2 = pd.Series([1, 2, 3, np.nan, None, pd.NA])
print(x2)
# => 0       1
#    1       2
#    2       3
#    3     NaN
#    4    None
#    5    <NA>
# dtype: object

x3 = pd.Series([1, 2, 3, np.nan, None, pd.NA], dtype='Int32')
```

```
print(x3)
# => 0       1
#    1       2
#    2       3
#    3    <NA>
#    4    <NA>
#    5    <NA>
# dtype: Int32
```

Работа с пустыми значениями

```
import numpy as np
import pandas as pd

x3 = pd.Series([1, 2, 3, np.nan, None, pd.NA], dtype='Int32')
print(x3.isnull())
# => 0    False
#    1    False
#    2    False
#    3     True
#    4     True
#    5     True
# dtype: bool

print(x3[x3.isnull()])
# => 3    <NA>
#    4    <NA>
#    5    <NA>
# dtype: Int32

print(x3[x3.isnull()])
# => 0    1
#    1    2
#    2    3
# dtype: Int32
```

```
print(x3.dropna())
# => 0    1
#    1    2
#    2    3
# dtype: Int32

df = pd.DataFrame([
    [1, 2, 3, np.nan, None, pd.NA],
    [1, 2, 3, 4, 5, 6, ]
])

print(df)
# =>    0  1  2    3    4     5
#       0  1  2  3  NaN  NaN  <NA>
#       1  1  2  3  4.0  5.0     6

print(df.dropna())
# =>    0  1  2    3    4  5
#       1  1  2  3  4.0  5.0  6

print(df.dropna(axis=0)) # по строкам ( по умолчанию )
# =>    0  1  2    3    4  5
#       1  1  2  3  4.0  5.0  6

print(df.dropna(axis=1)) # по столбцам
# =>    0  1  2
#       0  1  2  3
#       1  1  2  3
```

Критерий выбрасывания строки или столбца (how)

- all - все значения NA
- any - хотя бы одно значение NA
- thresh = x, остается, если присутствует МИНИМУМx’ НЕПУСТЫХ значений

```
import numpy as np
import pandas as pd

df = pd.DataFrame([
    [1, 2, 3, np.nan, None, pd.NA],
    [1, 2, 3, None, 5, 6, ],
    [1, np.nan, 3, None, np.nan, 6],
])

print(df)
# =>    0    1  2    3    4     5
#       0  1  2.0  3  NaN  NaN  <NA>
#       1  1  2.0  3  4.0  5.0     6
#       2  1  NaN  3  NaN  NaN     6

print(df.dropna(axis=1, how='all')) # выкидывается только те сто
# =>   0    1  2    4     5
#      0  1  2.0  3  NaN  <NA>
#      1  1  2.0  3  5.0     6
#      2  1  NaN  3  NaN     6

print(df.dropna(axis=1, how='any')) # выкидывается только те сто
# =>   0  2
#      0  1  3
#      1  1  3
#      2  1  3

print(df.dropna(axis=1, thresh=2))
# =>      0    1  2     5
#         0  1  2.0  3  <NA>
#         1  1  2.0  3     6
#         2  1  NaN  3     6
```

Задание для самостоятельной работы

5. На примере объектов DataFrame продемонстрировать использование методов ffill() и bfill()

2. Способ: обозначение через null