Pandas

1 Pandas简介

Pandas 是 Python 语言的一个扩展程序库，用于数据分析。

Pandas 是一个强大的分析结构化数据的工具集，基础是 Numpy。

Pandas 可以从各种文件格式（如 CSV、JSON、SQL、Microsoft Excel）导入数据。

导入 Pandas 一般使用别名 `pd` 来代替。

2 Pandas数据结构

2.1 Series

Pandas Series 类似表格中的一个列，类似于一维数组，可以保存任何数据类型。

Series 由索引（index）和列组成。用 `pandas.Series()` 函数创建 `Series` 对象，参数为：

1. `data`：一组数据，可以是序列或数组。
2. `index`：数据索引标签。如果不指定，默认从 0 开始。
3. `dtype`：数据类型。默认会自己判断。
4. `name`：设置名称。
5. `copy`：布尔值，设置是否拷贝数据。默认为 `False`。

```
import pandas as pd

a = [1, 2, 3]
var = pd.Series(a)
print(var)
# 0    1
# 1    2
# 2    3
# dtype: int64
```

可以指定索引值。例如：

```
import pandas as pd

a = ["Google", "Runoob", "Wiki"]
var = pd.Series(a, index=["x", "y", "z"])
print(var)
# x    Google
# y    Runoob
# z      Wiki
# dtype: object
```

可以根据索引值读取数据。例如：

```
import pandas as pd

a = [1, 2, 3]
varA = pd.Series(a)
print(varA[1])  # 2

b = ["Google", "Runoob", "Wiki"]
varB = pd.Series(b, index=["x", "y", "z"])
print(varB["y"])  # Runoob
```

也可以使用字典来创建`Series`对象。例如：

```
import pandas as pd

sites = {1: "Google", 2: "Runoob", 3: "Wiki"}
var = pd.Series(sites)
print(var)
# 1    Google
# 2    Runoob
# 3      Wiki
# dtype: object
```

如果只需要字典中的一部分数据，可以指定需要数据的索引。例如：

```
import pandas as pd

sites = {1: "Google", 2: "Runoob", 3: "Wiki"}
var = pd.Series(sites, index=[1, 2])
print(var)
# 1    Google
# 2    Runoob
# dtype: object
```

设置`name`参数：

```
import pandas as pd

sites = {1: "Google", 2: "Runoob", 3: "Wiki"}
var = pd.Series(sites, index=[1, 2], name="RUNOOB-Series-TEST")
print(var)
# 1    Google
# 2    Runoob
# Name: RUNOOB-Series-TEST, dtype: object
```

2.2 DataFrame

DataFrame 是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔型值）。DataFrame 既有行索引也有列索引，它可以被看做由 Series 组成的字典（共同用一个索引）。

DataFrame 构造方法为`pandas.DataFrame(data, index, columns, dtype, copy)`，参数为：

1. `data`：一组数据，可以是 Ndarray、Series、map、list、dict 等类型。
2. `index`：行标签。
3. `columns`：列标签。默认为从 0 开始的整数。
4. `dtype`：数据类型。
5. `copy`：布尔值，设置是否拷贝数据。默认为`False`。

使用列表创建 DataFrame：

```
import pandas as pd

data = [['Google', 10], ['Runoob', 12], ['Wiki', 13]]
df = pd.DataFrame(data, columns=['Site', 'Age'])
print(df)
#      Site  Age
# 0  Google   10
# 1  Runoob   12
# 2    Wiki   13
```

使用字典创建 DataFrame 时，字典的 key 为列名，字典的 value 为列数据，每列的长度必须相同。如果传递了`index`参数，则索引的长度应等于数组的长度；如果没有传递索引，则默认情况下，索引将是`range(n)`，其中`n`是数组长度。

```
import pandas as pd

data = {'Site': ['Google', 'Runoob', 'Wiki'], 'Age': [10, 12, 13]}
df = pd.DataFrame(data)
print(df)
#      Site  Age
# 0  Google   10
# 1  Runoob   12
# 2    Wiki   13
```

还可以使用字典序列创建 DataFrame，其中每个字典为一行，字典的 key 为列名。例如：

```
import pandas as pd

data = [{'a': 1, 'b': 2}, {'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print(df)
#    a   b     c
# 0  1   2   NaN
# 1  5  10  20.0
```

DataFrame 可以使用`loc`属性返回指定行的数据，返回结果是一个`Series`对象。例如：

```python
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

df = pd.DataFrame(data)

# 返回第一行
print(df.loc[0])
# calories    420
# duration     50
# Name: 0, dtype: int64

# 返回第二行
print(df.loc[1])
# calories    380
# duration     40
# Name: 1, dtype: int64
```

也可以返回多行数据，使用`[[ ... ]]`格式，`...`为各行的索引，以逗号隔开，返回结果是一个`DataFrame`对象。例如：

```python
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

df = pd.DataFrame(data)

# 返回第一行和第二行
print(df.loc[[0, 1]])
#    calories  duration
# 0       420        50
# 1       380        40
```

可以使用`index`参数指定行索引。例如：

```python
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index=["day1", "day2", "day3"])
print(df)
#       calories  duration
# day1       420        50
# day2       380        40
# day3       390        45
```

如果数据过多，输出 DataFrame 时会省略部分行或列。可以使用`to_string()`方法将 DataFrame 呈现为控制台友好的表格输出。

`head(n)`方法用于读取前面的 n 行，如果不填参数 n，默认返回 5 行。

`tail(n)`方法用于读取尾部的 n 行，如果不填参数 n ，默认返回 5 行。

`info()`方法返回 DataFrame 的一些基本信息。例如：

```python
import pandas as pd

# 三个字段 name, site, age
nme = ["Google", "Runoob", "Taobao", "Wiki"]
st = ["www.google.com", "www.runoob.com", "www.taobao.com", "www.wikipedia.org"]
ag = [90, 40, 80, 98]

data = {'name': nme, 'site': st, 'age': ag}

df = pd.DataFrame(data)

print(df.info())
# <class 'pandas.core.frame.DataFrame'>
# RangeIndex: 4 entries, 0 to 3
# Data columns (total 3 columns):
#  #   Column  Non-Null Count  Dtype
# ---  ------  --------------  -----
#  0   name    4 non-null      object
#  1   site    4 non-null      object
#  2   age     4 non-null      int64
# dtypes: int64(1), object(2)
# memory usage: 224.0+ bytes
# None
```

3 读写CSV文件

CSV（Comma-Separated Values，逗号分隔值）文件以纯文本形式存储表格数据。

`pandas.read_csv()`函数用于读取 CSV 文件。例如：

```python
import pandas as pd

df = pd.read_csv("./data/nba.csv")
print(df)
#                Name            Team  ...            College     Salary
# 0    Avery Bradley  Boston Celtics  ...              Texas  7730337.0
# 1      Jae Crowder  Boston Celtics  ...          Marquette  6796117.0
# 2     John Holland  Boston Celtics  ...  Boston University        NaN
# 3      R.J. Hunter  Boston Celtics  ...      Georgia State  1148640.0
# 4    Jonas Jerebko  Boston Celtics  ...                NaN  5000000.0
# ..             ...             ...  ...                ...        ...
# 453   Shelvin Mack       Utah Jazz  ...             Butler  2433333.0
# 454      Raul Neto       Utah Jazz  ...                NaN   900000.0
# 455   Tibor Pleiss       Utah Jazz  ...                NaN  2900000.0
# 456    Jeff Withey       Utah Jazz  ...             Kansas   947276.0
# 457            NaN             NaN  ...                NaN        NaN
#
# [458 rows x 9 columns]
```

可以使用`to_csv()`方法将 DataFrame 存储为 CSV 文件。例如：

```python
import pandas as pd

# 三个字段 name, site, age
nme = ["Google", "Runoob", "Taobao", "Wiki"]
st = ["www.google.com", "www.runoob.com", "www.taobao.com", "www.wikipedia.org"]
ag = [90, 40, 80, 98]

data = {'name': nme, 'site': st, 'age': ag}

df = pd.DataFrame(data)

# 保存 DataFrame
df.to_csv('./data/site.csv')
```

4 读取JSON文件

`pandas.read_json()` 函数用于读取 JSON 文件。参数可以是本地 JSON 文件的路径，也可以是 JSON 文件的 URL。例如：

```json
// sites.json
[
    {
        "id": "A001",
        "name": "菜鸟教程",
        "url": "www.runoob.com",
        "likes": 61
    },
    {
        "id": "A002",
        "name": "Google",
        "url": "www.google.com",
        "likes": 124
    },
    {
        "id": "A003",
        "name": "淘宝",
        "url": "www.taobao.com",
        "likes": 45
    }
]
```

```python
import pandas as pd

df = pd.read_json('./data/sites.json')
print(df)
#      id    name             url  likes
# 0  A001    菜鸟教程  www.runoob.com     61
# 1  A002  Google  www.google.com    124
# 2  A003      淘宝  www.taobao.com     45
```

如果 JSON 文件中存在嵌套结构，则直接读取时无法解析内嵌的数据。例如：

```
// nested.json
{
    "school_name": "ABC primary school",
    "class": "Year 1",
    "students": [
    {
        "id": "A001",
        "name": "Tom",
        "math": 60,
        "physics": 66,
        "chemistry": 61
    },
    {
        "id": "A002",
        "name": "James",
        "math": 89,
        "physics": 76,
        "chemistry": 51
    },
    {
        "id": "A003",
        "name": "Jenny",
        "math": 79,
        "physics": 90,
        "chemistry": 78
    }]
}
```

```
import pandas as pd

df = pd.read_json('./data/nested.json')
print(df)
#           school_name  ...                                           students
# 0  ABC primary school  ...  {'id': 'A001', 'name': 'Tom', 'math': 60, 'phy...
# 1  ABC primary school  ...  {'id': 'A002', 'name': 'James', 'math': 89, 'p...
# 2  ABC primary school  ...  {'id': 'A003', 'name': 'Jenny', 'math': 79, 'p...
#
# [3 rows x 3 columns]
```

读取嵌套 JSON 文件的方法：

1. 使用 Python json 模块载入数据。
2. 使用`pandas.json_normalize()`函数将内嵌的数据解析出来。

```
import pandas as pd
import json

# 使用 Python json 模块载入数据
with open('./data/nested.json', 'r') as f:
    data = json.loads(f.read())

# 展开内嵌的 students 数据
df_nested_list = pd.json_normalize(data, record_path=['students'])
print(df_nested_list)
#      id   name  math  physics  chemistry
# 0  A001    Tom    60       66         61
# 1  A002  James    89       76         51
# 2  A003  Jenny    79       90         78
```

显示结果没有包含`school_name`和`class`元素，如果需要展示出来可以使用`meta`参数来显示这些元数据：

```python
import pandas as pd
import json

# 使用 Python json 模块载入数据
with open('./data/nested.json', 'r') as f:
    data = json.loads(f.read())

# 展开内嵌的 students 数据，并将 school_name 和 class 作为元数据
df_nested_list = pd.json_normalize(
    data,
    record_path=['students'],
    meta=['school_name', 'class']
)
print(df_nested_list)
#      id   name  math  physics  chemistry         school_name   class
# 0  A001    Tom    60       66         61  ABC primary school  Year 1
# 1  A002  James    89       76         51  ABC primary school  Year 1
# 2  A003  Jenny    79       90         78  ABC primary school  Year 1
```

5 数据清洗

数据清洗是对一些没有用的数据进行处理的过程。

5.1 清洗空值

测试文件 property-data.csv 的内容如下：

PID	ST_NUM	ST_NAME	OWN_OCCUPIED	NUM_BEDROOMS	NUM_BATH	SQ_FT
100001000	104	PUTNAM	Y	3	1	1000
100002000	197	LEXINGTON	N	3	1.5	--
100003000		LEXINGTON	N	n/a	1	850
100004000	201	BERKELEY	12	1	NaN	700
	203	BERKELEY	Y	3	2	1600
100006000	207	BERKELEY	Y	NA	1	800
100007000	NA	WASHINGTON		2	HURLEY	950
100008000	213	TREMONT	Y	1	1	
100009000	215	TREMONT	Y	na	2	1800

`DataFrame.isnull()` 方法用于判断各个单元格是否为空。例如：

```
import pandas as pd

df = pd.read_csv('./data/property-data.csv')

print(df['NUM_BEDROOMS'])
# 0      3
# 1      3
# 2    NaN
# 3      1
# 4      3
# 5    NaN
# 6      2
# 7      1
# 8     na
# Name: NUM_BEDROOMS, dtype: object

print(df['NUM_BEDROOMS'].isnull())
# 0    False
# 1    False
# 2     True
# 3    False
# 4    False
# 5     True
# 6    False
# 7    False
# 8    False
# Name: NUM_BEDROOMS, dtype: bool
```

从上面的例子中可以看出 Pandas 把 n/a 和 NA 当作空数据，na 不是空数据，不符合要求。我们可以指定空数据类型，例如：

```
import pandas as pd

missing_values = ["n/a", "na", "--"]
df = pd.read_csv('./data/property-data.csv', na_values=missing_values)

print(df['NUM_BEDROOMS'])
# 0    3.0
# 1    3.0
# 2    NaN
# 3    1.0
# 4    3.0
# 5    NaN
# 6    2.0
# 7    1.0
# 8    NaN
# Name: NUM_BEDROOMS, dtype: float64

print(df['NUM_BEDROOMS'].isnull())
# 0    False
# 1    False
# 2     True
# 3    False
# 4    False
# 5     True
# 6    False
# 7    False
# 8     True
# Name: NUM_BEDROOMS, dtype: bool
```

`DataFrame.dropna()` 方法用于删除空值。参数为：

1. `axis`：默认值为 0，表示逢空值剔除整行。如果设置为 1 表示逢空值去掉整列。

2. `how`：默认值为`'any'`，表示一行（或一列）里任何一个数据为空就去掉整行（或列）。如果设置为`'all'`，则只有一行（或列）都是空才去掉这行（或列）。
3. `thresh`：设置需要多少非空值的数据才可以保留下来。
4. `subset`：设置想要检查的列。如果是多个列，可以使用列名的 list 作为参数。
5. `inplace`：如果设置为`True`，将计算得到的值直接覆盖之前的值并返回`None`。默认值为`False`，返回一个新的`DataFrame`，不会修改原数据。

删除所有包含空值的行：

```
import pandas as pd

df = pd.read_csv('./data/property-data.csv')

new_df = df.dropna()
print(new_df)
#            PID  ST_NUM    ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH SQ_FT
# 0  100001000.0   104.0     PUTNAM            Y            3        1  1000
# 1  100002000.0   197.0  LEXINGTON            N            3      1.5    --
# 8  100009000.0   215.0    TREMONT            Y           na        2  1800
```

删除指定列有空值的行：

```
import pandas as pd

df = pd.read_csv('./data/property-data.csv')

df.dropna(subset=['ST_NUM'], inplace=True)
print(df)
#            PID  ST_NUM    ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH SQ_FT
# 0  100001000.0   104.0     PUTNAM            Y            3        1  1000
# 1  100002000.0   197.0  LEXINGTON            N            3      1.5    --
# 3  100004000.0   201.0   BERKELEY           12            1      NaN   700
# 4          NaN   203.0   BERKELEY            Y            3        2  1600
# 5  100006000.0   207.0   BERKELEY            Y          NaN        1   800
# 7  100008000.0   213.0    TREMONT            Y            1        1   NaN
# 8  100009000.0   215.0    TREMONT            Y           na        2  1800
```

`DataFrame.fillna()`方法用给定值替换空值。例如：

```
import pandas as pd

df = pd.read_csv('./data/property-data.csv')

df.fillna(12345, inplace=True)
print(df)
#            PID   ST_NUM     ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH  SQ_FT
# 0  100001000.0    104.0      PUTNAM            Y            3        1   1000
# 1  100002000.0    197.0   LEXINGTON            N            3      1.5     --
# 2  100003000.0  12345.0   LEXINGTON            N        12345        1    850
# 3  100004000.0    201.0    BERKELEY           12            1    12345    700
# 4      12345.0    203.0    BERKELEY            Y            3        2   1600
# 5  100006000.0    207.0    BERKELEY            Y        12345        1    800
# 6  100007000.0  12345.0  WASHINGTON        12345            2   HURLEY    950
# 7  100008000.0    213.0     TREMONT            Y            1        1  12345
# 8  100009000.0    215.0     TREMONT            Y           na        2   1800
```

也可以指定某一列来替换数据。例如：

```
import pandas as pd

df = pd.read_csv('./data/property-data.csv')

df['PID'].fillna(12345, inplace=True)
print(df)
#            PID  ST_NUM     ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH SQ_FT
# 0  100001000.0   104.0      PUTNAM            Y            3        1  1000
# 1  100002000.0   197.0   LEXINGTON            N            3      1.5    --
# 2  100003000.0     NaN   LEXINGTON            N          NaN        1   850
# 3  100004000.0   201.0    BERKELEY           12            1      NaN   700
# 4      12345.0   203.0    BERKELEY            Y            3        2  1600
# 5  100006000.0   207.0    BERKELEY            Y          NaN        1   800
# 6  100007000.0     NaN  WASHINGTON          NaN            2   HURLEY   950
# 7  100008000.0   213.0     TREMONT            Y            1        1   NaN
# 8  100009000.0   215.0     TREMONT            Y           na        2  1800
```

替换空单元格的常用方法是计算列的均值、中位数或众数。`mean()`方法用于计算均值，`median()`方法用于计算中位数，`mode()`方法用于计算众数。例如：

```
import pandas as pd

df = pd.read_csv('./data/property-data.csv')

x = df["ST_NUM"].mean()
df["ST_NUM"].fillna(x, inplace=True)

print(df)
#            PID      ST_NUM     ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH SQ_FT
# 0  100001000.0  104.000000      PUTNAM            Y            3        1  1000
# 1  100002000.0  197.000000   LEXINGTON            N            3      1.5    --
# 2  100003000.0  191.428571   LEXINGTON            N          NaN        1   850
# 3  100004000.0  201.000000    BERKELEY           12            1      NaN   700
# 4          NaN  203.000000    BERKELEY            Y            3        2  1600
# 5  100006000.0  207.000000    BERKELEY            Y          NaN        1   800
# 6  100007000.0  191.428571  WASHINGTON          NaN            2   HURLEY   950
# 7  100008000.0  213.000000     TREMONT            Y            1        1   NaN
# 8  100009000.0  215.000000     TREMONT            Y           na        2  1800
```

5.2 清洗格式错误数据

格式化日期示例：

```
import pandas as pd

data = {
  "Date": ['2020/12/01', '2020/12/02' , '20201226'],
  "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index=["day1", "day2", "day3"])
print(df)
#             Date  duration
# day1  2020/12/01        50
# day2  2020/12/02        40
# day3    20201226        45

df['Date'] = pd.to_datetime(df['Date'])
print(df)
#            Date  duration
# day1 2020-12-01        50
# day2 2020-12-02        40
# day3 2020-12-26        45
```

5.3 清洗错误数据

可以通过赋值替换错误数据。例如：

```
import pandas as pd

person = {
  "name": ['Google', 'Runoob', 'Taobao'],
  "age": [50, 40, 12345]    # 12345 年龄数据是错误的
}

df = pd.DataFrame(person)
print(df)
#      name    age
# 0  Google     50
# 1  Runoob     40
# 2  Taobao  12345

df.loc[2, 'age'] = 30  # 修改数据
print(df)
#      name  age
# 0  Google   50
# 1  Runoob   40
# 2  Taobao   30
```

也可以用条件语句，为满足一定条件的单元格重新赋值。例如：

```python
import pandas as pd

person = {
  "name": ['Google', 'Runoob', 'Taobao'],
  "age": [50, 200, 12345]
}

df = pd.DataFrame(person)

for x in df.index:
    if df.loc[x, "age"] > 120:
        df.loc[x, "age"] = 120

print(df.to_string())
#      name  age
# 0  Google   50
# 1  Runoob  120
# 2  Taobao  120
```

也可以将错误数据的行删除。例如：

```python
import pandas as pd

person = {
  "name": ['Google', 'Runoob', 'Taobao'],
  "age": [50, 40, 12345]    # 12345 年龄数据是错误的
}

df = pd.DataFrame(person)

for x in df.index:
    if df.loc[x, "age"] > 120:
        df.drop(x, inplace=True)

print(df)
#      name  age
# 0  Google   50
# 1  Runoob   40
```

5.4 清洗重复数据

`DataFrame.duplicated()` 方法用于检测每一行是否重复，若重复则为 `True`，不重复则为 `False`。例如：

```python
import pandas as pd

person = {
  "name": ['Google', 'Runoob', 'Runoob', 'Taobao'],
  "age": [50, 40, 40, 23]
}

df = pd.DataFrame(person)

print(df.duplicated())
# 0    False
# 1    False
# 2     True
# 3    False
# dtype: bool
```

`DataFrame.drop_duplicates()` 方法用于去除重复行。例如：

```python
import pandas as pd

person = {
  "name": ['Google', 'Runoob', 'Runoob', 'Taobao'],
  "age": [50, 40, 40, 23]
}

df = pd.DataFrame(person)

df.drop_duplicates(inplace=True)
print(df)
#      name  age
# 0  Google   50
# 1  Runoob   40
# 3  Taobao   23
```