

Pandas

xbZhong

2024-08-19

[本页PDF](#)

数据读取

常用类型的读取

```
import pandas as pd  
# 取csv、tsv、txt数据  
df = pd.read_csv('path')
```

```
# 取excel文件  
df = pd.read_excel('path')
```

```
# 取sql文件  
df = pd.read_sql('path')
```

```
# 取csv文件时index_col为0则可以忽略多余的索引列  
df = pd.read_csv('../input/wine-reviews/winemag-data_first150k.csv', index_col = 0)
```

读取txt文件，自己指定分隔符、列名

```
df = pd.read_csv(  
    'path',  
    sep = '\t',  
    header = None,  
    names = ['pdate', 'pv', 'uv'])
```

常用操作

看前几行数据

head()

看数据形状

shape

看列名列表

columns

看索引列

index

看每列的数据类型

types

导出文件

to_csv方法

animals 导出为 csv

```
ls = pd.DataFrame({'Cows': [12, 20], 'Goats': [22, 19]}, index=['Year 1', 'Year 2'])  
ls.to_csv('cows_and_goats.csv')
```

数据结构

- Series : 一维数据，一行一列

定义 Series 与 index

```
pd.Series([1, 'a', 5.2, 7], index=['d', 'b', 'a', 'c'])  
dients = pd.Series(['4 cups', '1 cup', '2 large', '1 can'], index = ['Flour', 'Milk', 'Eggs', 'Spam'], name = 'Dinner')
```

取索引

index

取值

values

据 index 访问 values

```
(s1['a'])  
(type(s1['a']))
```

- DataFrame : 二维数据，多行多列

看列类型

ypes

看行索引

dex

看列索引

lums

询一列，结果是Series

ear']

询多列，结果是DataFrame

year', 'pop']]

询第一行

c[1]

询多行(包含第三行，和python语法有差别)

cp[1:3]

创建DataFrame

```
t pandas as pd  
s = pd.DataFrame({'Apple':[30], 'Bananas':[21]})
```

数据类型

几种方法

看每一列数据类型

ns.dtypes

换数据类型

ns.points.astype('float64')

数据删除

- drop(columns=['列名'])

除主键id的列

```
= data.drop(columns=['主键id'])
```

- `drop_duplicates(subset = ['列1', '列2', ...], keep = 'first')`

- 根据subset里面的列去重
- keep：控制如何保留重复值的哪一行，常见的选项有：
 - first：保留第一次出现的重复值
 - last：保留第一次出现的重复值
 - False：删除所有重复项

```
df.drop_duplicates(subset=['监测点id', '监测时间', '企业DeptId'], keep='first')
```

数据查询

几种查询方法

`==.iloc用数字索引左闭右开==`

`==.loc用数字索引左闭右闭==`

根据行、列的标签值查询

`o`

根据行、列的数字位置查询

`oC`

.loc 的几种查询方法

用单个label的值来查询数据

```
['2018-01-03', 'bWendu']
```

用值列表批量查询

```
[['2018-01-03', '2018-01-04', '2018-01-05'], ['bWemdu', 'yWendu']]
```

用数值区间进行查询

```
['2018-01-03': '2018-01-05', 'bWendu']
```

用条件表达式查询

```
df['yWendu'] < -10, :]
```

用列名读取数据

性值查询列

vs.country

典方式查询列

vs['country']

询特定值

vs['country'][0]

补充方法

in方法查询country是Italy和France的行

vs.loc[reviews.country.isin(['Italy', 'France'])] # 返回的是一行

idxmax()返回具有最大值元素的索引

reviews.points / reviews.price).idxmax()

新增数据列

几种方法

接赋值

df[:, 'Wencha'] = df['bWendu'] - df['yWendu']

用apply方法

get_wendu_type(x):

if x['bWendu'] > 33:

 return '高温'

if x['bWendu'] < -10:

 return '低温'

return '常温'

c[:, 'wendu_type'] = df.apply(get_wendu_type, axis=1)

sign方法，不会修改原表，返回的是新的表

sign(

wendu_huashi = lambda x : x['yWendu'] * 9 / 5 + 32.

wendu_huashi = lambda x : x['bWendu'] * 9 / 5 + 32

新增行

```
增加title行  
df.set_index('title')
```

Pandas数据统计函数

1. 汇总类统计

取所有数字列统计结果，非数字列也可以统计

```
describe()  
均值  
df['wendu'].mean()  
大值  
df['wendu'].max()  
位数  
df['wendu'].median()
```

2. 唯一去重和按值计数

```
到不同类别(去重)  
df['engxiang'].unique()  
据不同类别计数  
df['engxiang'].value_counts()
```

3. 相关系数和协方差

方差矩阵：衡量同向反向程度
cov()
相关系数矩阵：衡量相似度程度
corr()
独查看两变量的相关系数
df['ji'].corr(df['bWendu'])

Pandas缺失值数据处理

检测是否是空值

询问整个DataFrame是否为空值

.isnull()

询问某个列是否为空值

[‘分数’].isnull()

isnull的相反

.notnull()

[‘分数’].notnull()

丢弃，删除缺失值

dropna : 丢弃、删除缺失值

- axis: 删除行还是列
- how: 为any则任何值为空都删除, 为all则所有值都为空才删除
- inplace: 为True则修改当前df, 否则返回新的df

除全是空值的列

.dropna(axis = 1, how = ‘all’, inplace = True)

除掉只要有空值的行

.dropna(axis = 0, how = ‘any’, inplace = True)

填充空值

fillna : 填充空值

- value: 用于填充的值, 可以是字典或单个值
- method: 填充方式, ffill为使用前一个不为空的值填充, bfill为使用后一个不为空的值填充
- axis: 按行还是列填充
- inplace: 为True修改当前df, 否则返回新的df

分数列为空的填充为0分

.fillna({‘分数’: 0})

同于

.loc[:, ‘姓名’] = studf.fillna(0)

姓名的缺失值填充

.loc[:, ‘姓名’] = studf[‘姓名’].fillna(method = ‘ffill’)

替换

```
replace方法把taster_twitter_handle列的@kerinokeeife替换为@kerino  
ws.taster_twitter_handle.replace("@kerinokeeife", "@kerino")
```

重命名和合并

重命名

rename

- index或columns：重命名行或列
- 用字典来实现
- inplace：为True表示在原DataFrame上进行修改

列points改成score

```
ws.rename(columns={'points': 'score'})
```

rename_axis

- 给行或列增加name属性
- name：name属性名字
- rows或columns：属性是在行索引还是列索引

行索引添加name属性'wines'，给列索引添加name属性'fields'

```
ws.rename_axis('wines', axis = 'rows').rename_axis('fields', axis = 'columns')
```

合并

concat 语法: pandas.concat(objs, axis = 0, join = 'outer', ignore_index = False)

- objs：一个列表，内容可以是 DataFrame 或者 Series
- axis：0为按行合并，1为按列合并
- join：合并时索引对齐方式，默认为outer,inner会过滤掉不匹配的列
- ignore_index：是否忽略掉原来的位置索引
- concat会自动匹配索引的值，但是没有定义的会填充NaN

```
#  
concat([df1,df2])  
列合并  
concat([df1,df2],axis = 1)
```

join方法

- 根据索引自动匹配并合并值，未匹配成功的会丢弃
- other**: 要连接的另一个 DataFrame。
- how**: 连接方式，可以选择以下几种：
 - 'left' : 使用左边的 DataFrame 的索引，右边的 DataFrame 会填充 NaN。
 - 'right' : 使用右边的 DataFrame 的索引，左边的 DataFrame 会填充 NaN。
 - 'outer' : 保留两个 DataFrame 的所有索引，未匹配的地方填充 NaN。
 - 'inner' : 只保留索引匹配的行。
- on**: 指定用于连接的列（如果未设置为索引）。
- lsuffix**: 当两个 DataFrame 中存在重名列时，为左边 DataFrame 的列添加后缀。
- rsuffix**: 当两个 DataFrame 中存在重名列时，为右边 DataFrame 的列添加后缀。
- sort**: 是否根据连接后的索引排序，默认为 False。

```
frame.join(other, how='left', on=None, lsuffix='', rsuffix='', sort=False)
```

据两个DataFrame的MeetID进行合并

```
lifting_combined = powerlifting_meets.set_index('MeetID').join(powerlifting_competitors.set_index('MeetID'))
```

映射

map

得points列的绝对值，用map定义lambda函数自减，p指的是reviews自己

```
review_points_mean = reviews.points.mean()  
reviews.points.map(lambda p: p - review_points_mean)
```

算description列中tropical和fruity出现的次数

```
tropical_counts = pd.Series([reviews.description.map(lambda x: 'tropical' in x).sum(), reviews.description.map(lambda y: 'fruity' in y).sum()], index = ['tropical', 'fruity'])
```

apply

- axis为columns表示

以用于行

```
def remean_points(row):
    row.points = row.points - review_points_mean
    return row
```

```
reviews.apply(remean_points, axis='columns')
```

义函数，将row中的country为Canada的星级设为3颗星，然后根据分数来评星级

```
def method(row):
    if row.country == 'Canada':
        return 3
    if row.points >= 95:
        return 3
    if row.points >= 85:
        return 2
    return 1
```

```
ratings = reviews.apply(method, axis = 'columns')
```

check your answer

```
check()
```

数据分组及排序

分组

groupby分组

照points进行分组并对points进行数量计算，具有相同points的行会被分到同一组

```
reviews.groupby('points').points.sum()
```

以接受多个参数并使用apply()方法

```
reviews.groupby(['country', 'province']).apply(lambda df: df.loc[df.points.idxmax()])
```

照price来分组，选取最大的points并按照points来排序

```
rating_per_price = reviews.groupby('price')['points'].max().sort_index()
```

agg方法

- 参数为一个列表，里面的元素是要调用的函数

以接收多个参数，运行一系列不同的函数

```
ms.groupby(['country']).price.agg([len, min, max])
```

多索引问题

换回单索引，但前面的多索引会变成新的列

```
ries_reviewed.reset_index()
```

排序

- ascending: 默认为True升序排序，为False降序排序
- inplace: 是否替换原始Series
- by: 按照哪一列进行排序，可以接收一个列表

值进行排序

```
ianqi'].sort_values()
```

照len列进行排序

```
ries_reviewed.sort_values(by='Len')
```

受一个列表进行排序

```
ries_reviewed.sort_values(by=['country', 'len'])
```

索引进行排序

```
ries_reviewed.sort_index()
```