Pandas

xbZhong

2024-08-19

本页 PDF

数据读取

常用类型的读取

```
import panda as pd
## 读取 csv、tsv、txt 数据
csv = pd.read_csv('path')

## 读取 excel 文件
excel = pd.read_excel('path')

## 读取 sql 文件
sql = pd.read_sql('path')

## 读取 csv 文件时 index_col 为 0 额可以忽略多余的索引列
reviews = 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

```
txt = pd.read_csv(
    'path'
    # 指定分割字符
    sep = '\t'
    # 无标题行
    header = None
    # 自定义列名
    names = ['pdate','pv','uv']
)
```

常用操作

```
## 查看前几行数据
csv.head()
## 查看数据形状
csv.shape
## 查看列名列表
csv.colums
## 查看索引列
csv.index
## 查看每列的数据类型
csv.dtypes
```

导出文件

to_csv 方法

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

数据结构

- Series: 一维数据，一行一列

```
## 自定义 Series 与 index
s1 = pd.Series([1,'a',5.2,7],index=['d','b','a','c'])
ingredients = pd.Series(['4 cups','1 cup','2 large','1 can'],index = ['Flour','Milk','Eggs','Spam'],name
## 获取索引
s1.index

## 获取值
s1.values

## 根据 index 访问 values
print(s1['a'])
print(type(s1['a']))
```

- DataFrame: 二维数据，多行多列

```
## 查看列类型
df.dtypes
## 查看行索引
df.index
## 查看列索引
df.colums

## 查询一列，结果是 Series
df['year']
## 查询多列，结果是 DataFrame
df[['year','pop']]

## 查询第一行
df.loc[1]
## 查询多行（包含第三行，和 python 语法有差别）
df.locp[1:3]
```

创建 DataFrame

```
import pandas as pd
fruits = pd.DataFrame({'Apple':[30],'Bananas':[21]})
```

数据类型

几种方法

```
## 查看每一列数据类型
reviews.dtypes
```

```
## 转换数据类型
reviews.points.astype('float64')
```

数据删除

- drop(colums=['列名'])

```
## 删除主键 id 的列
data = data.drop(colums=['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 = 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
df.loc
## 根据行、列的数字位置查询
df.iloc
```

.loc 的几种查询方法

```
## 使用单个 label 的值来查询数据
df.loc['2018-01-03','bWendu']
## 使用值列表批量查询
df.loc[['2018-01-03','2018-01-04','2018-01-05'],['bWemdu','yWendu']]
## 使用数值区间进行查询
df.loc['2018-01-03':'2018-01-05','bWendu']
## 使用条件表达式查询
df.loc[df['yWendu']<-10,:]
```

用列名读取数据

```
## 属性值查询列
reviews.country
## 字典方式查询列
reviews['country']
## 查询特定值
reviews['country'][0]
```

补充方法

```
## isin 方法查询 country 是 Italy 和 France 的行
reviews.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

```
## 直接赋值
df.loc[:,'Wencha'] = df['bWendu'] - df['yWendu']
## 使用 apply 方法
def get_wendu_type(x):
    if x['bWendu'] > 33:
        return '高温'
    if x['bWendu'] < -10:
        return '低温'
    return '常温'
df.loc[:,'wendu_type'] = df.apply(get_wendu_type,axis = 1)
## assign 方法，不会修改原表，返回的是新的表
df.assign(
    yWendu_huashi = lambda x : x['yWendu'] * 9 / 5 + 32.
    bWendu_huashi = lambda x : x['bWendu'] * 9 / 5 + 32
)
```

新增行

```
## 新增 title 行
reviews.set_index('title')
```

Pandas 数据统计函数

1. 汇总类统计

```
## 提取所有数字列统计结果，非数字列也可以统计
df.describe()
## 平均值
df['bWendu'].mean()
## 最大值
df['bWendu'].max()
## 中位数
df['bWendu'].median()
```

2. 唯一去重和按值计数

```
## 得到不同类别（去重）
df['fengxiang'].unique()
## 根据不同类别计数
df['fengxiang'].value_counts()
```

3. 相关系数和协方差

```
## 协方差矩阵：衡量同向反向程度
df.cov()
## 相关系数矩阵：衡量相似度程度
df.corr()
```

```
## 单独查看两变量的相关系数
df['api'].corr(df['bWendu'])
```

Pandas 缺失值数据处理

检测是否是空值

```
## 查询整个 DataFrame 是否为空值
studf.isnull()
## 查询某个列是否为空值
studf['分数'].isnull()
## 与 isnulll 相反
studf.notnull()
studf['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

```
## 删除全是空值的列
studf.dropna(axis = 1,how = 'all',inplace = True)
## 删除掉只要有空值的行
studf.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

```
## 将分数列为空的填充为 0 分
studf.fillna({'分数':0})
## 等同于
studf.loc[:,'姓名'] = studf.fillna(0)
## 将姓名的缺失值填充
studf.loc[:,'姓名'] = studf['姓名'].fillna(method = 'ffill')
```

替换

```
## 用 replace 方法把 taster_twitter_handle 列的 @kerinokeefe 替换为 @kerino
reviews.taster_twitter_handle.replace("@kerinokeefe", "@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

```
## 将列 points 改成 score
reviews.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

```
## 给行索引添加 name 属性'wines'，给列索引添加 name 属性'fields'
reviews.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

```
## 合并
pd.concat([df1,df2])
## 按列合并
pd.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。

```
DataFrame.join(other, how='left', on=None, lsuffix='', rsuffix='', sort=False)
## 根据两个 DataFrame 的 MeetID 进行合并
powerlifting_combined = owerlifting_meets.set_index('MeetID').join(powerlifting_competitors.set_index('M
```

映射

map

```
## 求得 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)
## 计算 description 列中 tropical 和 fruity 出现的次数
descriptor_counts = pd.Series([reviews.description.map(lambda x : 'tropical' in x).sum(),reviews.descrip
```

apply

- axis 为 colums 表示

```
## 可以用于行
def remean_points(row):
    row.points = row.points - review_points_mean
    return row

reviews.apply(remean_points, axis='columns')
## 定义函数，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
    elif row.points >= 95:
        return 3
    elif row.points >= 85:
        return 2
    return 1

star_ratings = reviews.apply(method,axis = 'columns')

## Check your answer
q7.check()
```

数据分组及排序

分组

groupby 分组

```
## 按照 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()
## 可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()])


## 按照 price 来分组，选取最大的 points 并按照 points 来排序
best_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

```
## 可以接收多个参数，运行一系列不同的函数
reviews.groupby(['country']).price.agg([len, min, max])
```

多索引问题

```
## 转换回单索引，但前面的多索引会变成新的列
countries_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

```
## 按值进行排序
df['tianqi'].sort_values()
## 按照 len 列进行排序
countries_reviewed.sort_values(by='len')
## 接受一个列表进行排序
countries_reviewed.sort_values(by=['country', 'len'])
## 按索引进行排序
countries_reviewed.sort_index()
```