

配套视频：[数据清洗](#)

PDF下载地址：[数据分析PDF](#)

一、Python数据分析简介

1. 常用Python数据分析开源库

- numpy
 - 一个强大的N维数组对象ndarray
 - 广播功能函数
 - 线性代数。傅里叶变换，随机数生成等
- pandas
 - 强大的分析结构化数据的工具集
 - 用于数据挖掘和数据分析，同时也提供数据清洗功能
 - pandas利器
 - series：一种类似于一维数组的对象
 - dataframe：是pandas中的应该表格型的数据结构
- matplotlib
 - 强大的数据可视化开源库
 - python中使用最多的图形绘图库
 - 可以创建静态，动态和交互式的图表
- seaborn
 - 建立在matplotlib之上，并集成了pandas的数据结构
 - seaborn通过更简洁的API来绘制信息更丰富，更具吸引力的图像
 - 面向数据集的API，与pandas配合使用起来比直接使用matplotlib更方便
- sklearn
 - scikit-learn是基于python语言的机器学习工具
 - 简单高效的数据挖掘和数据分析工具
 - 可供大家子啊各种环境中重复使用
 - 建立在numpy、scipy和matplotlib上
- jupyter notebook/jupyterlab
 - jupyter notebook是一个开源的web应用程序
 - 可以创建和共享代码、公式、可视化图表、笔记文档
 - 是数据分析学习和开发的首选开发环境
 - 用途：
 - 数据清理和转换
 - 数值模拟
 - 统计分析
 - 数据可视化
 - 机器学习等

2. Python数据分析环境搭建

Anaconda简介

Anaconda是最流行的数据分析平台，全球两千多万人在使用

Anaconda附带一大批常用数据科学包

Anaconda在conda(一个包管理器和环境管理器)上发展出来的

可以帮助你在计算机上安装和管理时间分析相关包

包含了虚拟环境管理工具

3.jupyter使用

打开jupyter

在要打开的文件目录下进入命令行，输入以下指令

```
shell
jupyter notebook
```

常用快捷键

编辑模式：按 **Enter** 进入

多光标操作：**Ctrl** 键 编辑鼠标

重做：**Ctrl** + **Y**

代码补全：变量、方法后跟 **Tab** 键

为一行代码或多行代码添加/取消注释：**Ctrl** + **/**

执行本单元代码，并跳转到下一单元：**shift** + **Enter**

执行本单元代码，并留在本单元：**Ctrl** + **Enter**

cell行号前的*，表示代码正在运行

二、pandas数据结构

1.series和dataframe

series和dataframe是pandas最基本的两种数据结构

dataframe用来处理结构化数据(sql数据表，excel表格)

series用来处理单列数据，也可以把dataframe看作是series对象组成的字典或集合

2.创建Series

```
python
import pandas as pd

# 创建一个series对象
a = pd.Series(['banana',42])

print(type(a))
print(a)

b = pd.Series(['张三',19],index=['name','age'])
print(b)
```

3.创建dataframe

可以使用字典来创建dataframe

python

```
import pandas as pd

students = {
    'name':['张三','李四'],
    'age':[22,19],
}
# 创建一个dataframe对象
a = pd.DataFrame(data=students)
print(a)

b = pd.DataFrame(data={'age':[22,19]},index=['张三','李四'])
print(b)
```

♥ 创建时没有指定行索引，会自动创建0,1作为行索引，字典中的key，自动作为列名

三、Series常用操作

1.Series常用属性

使用dataframe的loc属性获取数据集里的一行，就会得到一个series对象

加载数据

python

```
import pandas as pd

a = pd.read_csv('订单数据表.csv')
print(a)

b = pd.read_csv('订单数据表.csv', index_col='order_id')
print(b)
```

数据加载后会生成一个dataframe对象

读取数据

python

```
import pandas as pd

a = pd.read_csv('订单数据表.csv')
print(a)
print(a.head()) # 显示最前的5行数据
print(a.tail()) # 显示最后的5行数据
print(a.head(10)) # 显示最前的10行数据
print(a.tail(10)) # 显示最后的10行数据
```

使用索引标签选择一条记录

python

```
import pandas as pd

a = pd.read_csv('订单数据表.csv')
data = a.loc[1] # 读取指定的一行
print(data)
print(type(data))
```

使用loc取出数据后是一个series对象

可以通过index和values属性获取行索引和值

```
python
import pandas as pd

a = pd.read_csv('订单数据表.csv')
data = a.loc[1] # 读取指定的一行

print(data.index)
print(data.columns)
```

series对象有index属性和values属性

series的keys方法

作用和index属性一样

```
python
import pandas as pd

a = pd.read_csv('订单数据表.csv')
data = a.loc[1] # 读取指定的一行

print(data.keys())
```

series的一些属性

属性	说明
loc	使用索引值取子集
iloc	使用索引位置取子集
dtype或dtypes	series内容的类型
T	series的转置矩阵
shape	数据的维数
size	series中元素的数量
values	series的值

2.series常用方法

方法	作用
max()	最大值
min()	最小值
mean()	平均值
std()	标准差，反应的是一组数据中和平均值的差异程度
value_counts()	可以返回不同值的条目数量 会对这一列中的值进行分组 聚合，统计每个分组中的个数 排序，默认根据上一步的统计结果进行降序排序
count()	返回有多少非空值
describe()	打印描述信息

针对数值型的series，可以进行常见的计算

```
python
import pandas as pd

a = pd.read_csv('订单数据表.csv')

# 取出quantity这列
data = a.quantity
# 求这一列的平均值
print(data.mean())
# 最大值
print(data.max())
# 最小值
print(data.min())
# 计算标准差，反应的是一组数据中和平均值的差异程度
print(data.std())
```

通过value_counts()方法，可以返回不同值的条目数量

```
python
import pandas as pd

a = pd.read_csv('订单数据表.csv')
data = a['item_price']
# 会对这一列中的值进行分组
# 聚合，统计每个分组中的个数
# 排序，默认根据上一步的统计结果进行降序排序
print(data.value_counts())
# 设置升序排序
print(data.value_counts(ascending=True))
```

count方法

通过count()方法可以返回有多少非空值

```
python
import pandas as pd

a = pd.read_csv('订单数据表.csv')
data = a['item_price']
# 返回有多少非空值
print(data.count())
# 返回个数，不管是否为空
print(data.size)
```

describe()方法

打印描述信息

```
python
import pandas as pd

a = pd.read_csv('订单数据表.csv')
data = a['item_price']

# 打印描述信息
print(data.describe())
print('*'*80)
quantity = a['quantity']
print(quantity.describe())
```

其他的常用方法

方法	说明
append()	连接两个或多个series
corr()	计算与另一个series的相关系数
cov()	计算与另一个series的协方差
describe()	计算常见统计量
drop_duplicates()	返回去重后的两个series
equals()	判断两个series是否相同
get_values()	获取series的值，作用与values属性相同
hist()	绘制直方图
isin()	series中是否包含某些值
min()	最小值
max()	最大值
mean()	算数平均值
median()	返回中位数
mode()	返回众数
quantile()	返回指定位置的分位数
replace()	用指定值代替series中的值
sample()	返回series的随机采样值
sort_values()	对值进行排序
to_frame()	把series转化为dataframe
unique()	去重返回数组

3.series的布尔索引

从series中获取满足某些条件的数据，可以使用布尔索引

获取大于平均值的结果

python

```
import pandas as pd

a = pd.read_csv('订单数据表.csv')
quantity = a['quantity']
# 获取大于平均值的结果
print(quantity[quantity>quantity.mean()])
```

4.series的运算

series和数值变量计算时，变量会与series中的每一个元素逐一进行计算

python

```
import pandas as pd

a = pd.read_csv('订单数据表.csv')
quantity = a['quantity']
print(quantity+100)
print(quantity*2)
```

两个series之间计算，如果series元素个数相同，则两个series对应元素进行计算

python

```
import pandas as pd

a = pd.read_csv('订单数据表.csv')
quantity = a['quantity']
print(quantity+quantity)
```

元素的个数不同的series之间进行计算，会根据索引进行，索引不同的元素最终计算的结果会填充成缺失值，用NaN表示

python

```
import pandas as pd

a = pd.read_csv('订单数据表.csv')
quantity = a['quantity']
print(quantity+pd.Series([1,100]))
```

Series运算规则

- Series和常数做运算，Series中的每个值和这个常数进行运算
- Series和Series进行运算
 - 两个Series索引相同的值进行运算
 - 如果一个Series中对应位置没有值，也就是空值，生成结果中，对应位置也是空值
 - 对其中一个Series进行逆序操作，再进行运算，结果仍然符合以上规则

四、Dataframe常用操作

1.Dataframe的常用属性和方法

Dataframe是Pandas中最常见的对象，Series数据结构的许多属性和方法在Dataframe中也一样适用

python

```
import pandas as pd

a = pd.read_csv('订单数据表.csv')
# 打印行数和列数
print(a.shape)
# 打印数据的个数
print(a.size)
# 该数据集的维度
print(a.ndim)
# 该数据集的长度
print(len(a))
# 每列的非空值个数
print(a.count())
# 各列的最小值
print(a.min())
# 各列的最大值
print(a.max())
# 各列的平均值
# print(a.mean())
# 对数值列进行统计
print(a.describe())
# print(a.describe(include=object))
print('*'*80)
print(a.describe(include = 'all'))
```

2.Dataframe的布尔索引

同Series一样，Dataframe也可以使用布尔索引获取数据子集

python

```
import pandas as pd

a = pd.read_csv('订单数据表.csv')
# 计算每行quantity大于quantity平均值的行
print(a[a['quantity']>a['quantity'].mean()])
print(a.head()[[True,False,False,True,False]])
```

3.Dataframe的运算

当Dataframe和数值进行运算时，Dataframe中的每一个元素会分别和数值进行运算

python

```
import pandas as pd

a = pd.read_csv('订单数据表.csv')
print(a*2)
```

两个Dataframe之间进行计算，会根据索引进行对应计算

python

```
import pandas as pd

a = pd.read_csv('订单数据表.csv')
print(a+a)
# a[:4]取出前四行
b = a[:4]
print(b)
# 取出前四行三列数据
c = b[['order_id','quantity','item_name']]
print(c)

print('*'*80)
print(a.value_counts())
# 对结果取反
print('*'*80)
print(~a.value_counts())

# 取值的区别
# series
s = a['item_name']
print(type(s))
# dataframe
d = a[['item_name']]
print(type(d))
```



dataframe和常数运算，dataframe中的每个列(就是series对象)和常数运算，每个元素和这个常数运算
dataframe和dataframe运算，首先找行索引相同的行，再找列名相同的元素，再把这两个值进行运算

注：两个中括号得到的是dataframe类型的数据，一个中括号得到的是series类型的数据，两个中括号可以传多个列的列名(前提列名存在)，一个中括号只能传一个列名

4. 修改Series和Dataframe

给行索引命名

方法一set_index()

加载数据后通过set_index()修改索引

加载数据文件时，如果不指定行索引，pandas会自动加上从0开始的索引，可以通过set_index()方法重新设置行索引的名字

set_index()有一个inplace的参数，这个参数默认为False，不会在原始的Dataframe上进行修改，会吧修改后的Dataframe返回出来
如果设置为True，直接在原始的Dataframe上修改，没有返回值

```
python
import pandas as pd

# 无行索引
a = pd.read_csv('movie.csv')
print(a)
# 使用movie_title作为行索引
movie = a.set_index('movie_title')
print(movie)
```

索引可以有重复，也可以有空值

方法二index_col

加载数据时

加载数据的时候可以通过index_col参数，指定使用某一列数据作为行索引

```
python
import pandas as pd

a = pd.read_csv('movie.csv')
print(a)
print('*'*80)
a = pd.read_csv('movie.csv',index_col='movie_title')
print(a)
```

reset_index()重置索引

进行过滤后，或者合并数据表之后，可能需要进行重置索引

```
python
import pandas as pd

a = pd.read_csv('movie.csv')
print(a)
print('*'*80)
a2 = a.set_index('movie_title')
# 重置索引
c = a2.reset_index()
print(c)
```

有一个参数drop，默认是False，重置之前的索引仍然会保留下来
inplace默认是False，返回一个修改后的Dataframe，需要一个变量接收结果

Dataframe修改行名和列名

方法一：rename()

Dataframe创建之后可以通过，rename()方法对原有的索引名和列名进行修改

```
python
import pandas as pd

a = pd.read_csv('movie.csv',index_col = 'movie_title')
# 取出行索引的前五个
b = a.index[:5]
print(b)
# 取出列名的前三个
c = a.columns[:3]
```

```
print(c)

# 修改行索引
b_rename = {
    'The Shawshank Redemption':'肖生克的救赎',
    'The Godfather':'教父'
}
# 修改列名
c.rename = {
    'movie_year':'release_date',
}
d = a.rename(index=b_rename, columns=c.rename).head()
print(d)
```

rename() 方法的参数

需要修改某些特定的行索引或者列名时使用

- index：修改行索引
- columns：修改列名
- inplace：默认False，返回修改后的dataframe，设置为True是在原始的数据frame上修改
- 语法

```
python
line = {
    '原始值':'修改值',
    ...
}
row = {
    '原始值':'修改值',
    ...
}
a.rename(index=line,columns=row)
```

方法二：索引

如果不使用rename()，也可以将index和columns属性提取出来，修改之后再赋值回去

```
python
import pandas as pd

a = pd.read_csv('movie.csv',index_col = 'movie_title')

# 取出index
b = a.index.tolist()
# 取出columns
c = a.columns.tolist()

print(b)
print(c)

# 修改
b[0] = '肖生克的救赎'
c[0] = '电影评分'

l = a.index = b
r = a.columns = c

print(a)
```

修改列名/行索引，也可以直接生成一个list，存的是要修改的行索引/列名，覆盖掉原始的index/columns

添加、删除、插入列

添加列

语法

python

```
df['列名'] = 值      # 添加的列中值都是相同的
df['列名'] = df['列名1']+df['列名2']...    # 通过运算得到
```

通过dataframe[列名]添加新列

python

```
import pandas as pd
from datetime import datetime as dt

a = pd.read_csv('movie.csv',index_col = 'movie_title')

# 添加列
a['like'] = 'no'

now_year = dt.now().year
# 添加列并赋值
a['years_issuance'] = now_year - a['movie_year']
print(a)
```

插入列

语法

python

```
df.insert(loc=位置,column='新列名',value=值)
```

- loc: 插入的位置(0,1,2,...)
- column: 新插入的列名
- value: 新插入的列的是怎么计算的

使用insert()方法插入列loc新插入的列在所有列中的位置(0,1,2,3,...)column=列名 value=值

python

```
import pandas as pd
from datetime import datetime as dt

a = pd.read_csv('movie.csv',index_col = 'movie_title')

# 插入列
now_year = dt.now().year
# 插入列并赋值
a.insert(loc=2,column='years_issuance',value=now_year-a['movie_year'])
print(a[:4].to_string())
```

删除列

语法

python

```
# 删除列
df.drop(columns='列名')
# 删除行
df.drop(index='行名')
# 用法二
df.drop('列名/行名',axis='index/columns')
```

调用drop()方法删除列

python

```
import pandas as pd
from datetime import datetime as dt

a = pd.read_csv('movie.csv', index_col = 'movie_title')

# 插入列
now_year = dt.now().year
# 插入列并赋值
a.insert(loc=2, column='years_issuance', value=now_year-a['movie_year'])

# 删除列
a.drop(columns='years_issuance', inplace=True)
# 删除行
a.drop(index='The Shawshank      Redemption', inplace=True)
print(a[:4].to_string())
```

五、数据的导入和导出

1.pickle文件

保存成pickle文件

- 调用to_pickle()方法将以二进制格式保存数据
- 如果要保存的对象是计算的中间结果，或者保存的对象以后会在python中复用，可把对象保存为.pickle文件
- 如果保存成pickle文件，只能在python中使用
- 文件扩展名可以是.p/.pkl/.pickle

 python

```
import pandas as pd

a = pd.read_csv('movie.csv')
movies = a['movie_title']
# 保存为pickle文件
movies.to_pickle('movie_title.p')
a.to_pickle('a.p')
```

读取pickle文件

 python

```
import pandas as pd

a = pd.read_csv('movie.csv')
movies = a['movie_title']
# 保存为pickle文件
movies.to_pickle('movie_title.p')
a.to_pickle('a.p')

# 读取
s = pd.read_pickle('movie_title.p')
a = pd.read_pickle('a.p')
print(s)
print(a)
```

2.CSV文件

保存成CSV文件

- CSV(逗号分隔值)是很灵活的一种数据存储格式
- 在CSV文件中，对于每一行，各列采用逗号分隔
- 除了逗号，还可以使用其他类型的分隔符，比如TSV文件，使用制表符作为分隔符

- CSV是数据协作和共享的首选格式

```
python
import pandas as pd

a = pd.read_csv('movie.csv')
movie_title = a['movie_title']
# 保存为CSV文件
movie_title.to_csv('movie_title.csv')

# 指定分隔符
movie_title.to_csv('movie_title.csv',sep='/')
# 设置为/后也得使用/打开
# c = pd.read_csv('movie_title.csv',sep='/')

# 去除unnamed
movie_title.to_csv('movie_title.csv',index=False)
```

3.Excel文件

保存成Excel文件

- series这种结构不支持to_excel()方法，想要保存成Excel文件，需要把Series转换成Dataframe

```
python
import pandas as pd

a = pd.read_csv('movie.csv')
movie_title = a['movie_title']
# 保存为excel文件
movie_title.to_excel('movie_title.xlsx',index=False)
a.to_excel('a.xlsx',index=False,sheet_name='电影排行')
# sheet_name: 表名

# 加载文件
c = pd.read_excel('a.xlsx')
print(c)
```

4. 其他数据格式

feather文件

- feather是一种文件格式，用于存储二进制对象
- feather对象也可以加载到R语言中使用
- feather格式的主要优点是在python和R语言之间传递数据
- 一般不用做保存最终数据

导出方法	说明
to_clipboard()	把数据存到系统剪贴板，方便粘贴
to_dict()	把数据转换成python字典
to_hdf()	把数据保存为HDF格式
to_html()	把数据转换成HTML
to_json()	把数据转化成JSON字符串
to_sql()	把数据保存到SQL数据库

六、pandas Dataframe入门

1.加载数据集

```
python
import pandas as pd

a = pd.read_csv('movie.csv')
# 返回数据类型
print(type(a))
# 查看行列数
print(a.shape)
# 元素数量
print(a.size)
# 获取列名
print(a.columns)
# 查看属性
print(a.dtypes)
print(a.info())
```

pandas与python常用数据类型对照

pandas类型	python类型	说明
Object	string	字符串类型
int64	int	整型
float64	float	浮点型
datetime64	datetime	日期时间类型，python中需要加载

2.查看部分数据

根据列名加载部分数据

加载一列数据，通过df[‘列名’]方式获取

```
python
import pandas as pd

a = pd.read_csv('movie.csv')
# 获取movie_title这列
c = a['movie_title']
# 获取前5行
c = c.head()
print(c)
```

加载多列数据，通过df[['列名1','列名2',...]]

```
python
import pandas as pd

a = pd.read_csv('movie.csv')
# 获取多列
c = a[['movie_title','movie_rating']]
# 获取后5列
c = c.tail()
print(c)
```

按行加载部分数据

loc：通过行索引获取指定行数据

python

```
import pandas as pd

a = pd.read_csv('movie.csv',index_col='movie_title')
# 观察前5行数据
print(a.head())
# 取出指定行
c = a.loc['The Godfather']    # 索引名
d = a.iloc[0]    # 行号
print(c)
print('*'*80)
print(d)
```

获取指定行/列数据

loc和iloc属性可以用于获取列数据，也可以用于获取行数据

- df.loc[[行索引名],[列名]]
- df.iloc[[行],[列]]

使用loc获取数据中的1列/几列

- df.loc[[所有行],[列名]]
- 取出所有行，可以使用切片语法df.loc[:,[列名]]

使用iloc获取数据中的第1列/几列

- df.iloc[:,[列序号]]：列序号可以使用-1代表最后一列

python

```
import pandas as pd

a = pd.read_csv('movie.csv',index_col='movie_title')
# 获取第五行，第二列
b = a.iloc[[4],[0]]
c = a.loc[['12 Angry Men'],['movie_rating']]
# 获取指定列的所有行
d = a.loc[:,['movie_rating']]
e = a.iloc[:,[0,1,-1]]
print(a[:5])
print(b)
print(c)
print(d)
print(e)
```

获取多行多列

- 可以把获取单行单列的语法和获取多行多列的语法结合起来使用
- 获取第一列，第四六，第六列，数据中的第1行，第100行和第1000行

python

```
print(df.iloc[[0,100,1000],[0,3,5]])
```

- 在实际工作中，获取某几列的数据的时候，建议传入实际的列名，好处：
 - 增加代码的可读性
 - 避免因列顺序的变化导致取出错误，列的数据

3. 分组和聚合计算

gapminder文件下载地址：[gapminder.tsv](https://raw.githubusercontent.com/joaquinreyes/gapminder_tsv/master/gapminder.tsv)

♥️ 不要问我为啥知道文件在哪下的，我也是自己找到的🐶🐾

github不会下东西的话，那我没招了，自己问[豆包](#)吧

github有时候进不去很正常，因为不是国内的，没办法直连，有科学上网的用科学上网，没有科学上网的可以使用中国镜像站

在使用Excel或者SQL进行数据处理时，Excel和SQL都提供了基本的统计计算功能

python

```
import pandas as pd

df = pd.read_csv('gapminder.tsv', sep='\t')
print(df[:10])
```

结果：

python

	country	continent	year	lifeExp	pop	gdpPercap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
2	Afghanistan	Asia	1962	31.997	10267083	853.100710
3	Afghanistan	Asia	1967	34.020	11537966	836.197138
4	Afghanistan	Asia	1972	36.088	13079460	739.981106
5	Afghanistan	Asia	1977	38.438	14880372	786.113360
6	Afghanistan	Asia	1982	39.854	12881816	978.011439
7	Afghanistan	Asia	1987	40.822	13867957	852.395945
8	Afghanistan	Asia	1992	41.674	16317921	649.341395
9	Afghanistan	Asia	1997	41.763	22227415	635.341351

需求：

- 每一年的平均预期寿命的多少？每一年的平均人口和平均GDP是多少？
- 如果我们按照大洲计算，每年每个大洲的平均预期寿命，平均人口，平均GDP情况又如何？
- 在数据中，每个大洲列出了多少个国家和地区？

分组方式

对于上面提出的问题，需要进行分组-聚合计算

- 先将数据分组(每一年的平均预期寿命问题，按照年份将相同年份的数据分成一组)
- 对魅族的数据再去进行统计计算如，求平均，求每条数据条目数(频数)等
- 再将每一组计算的结果合并起来
- 可以使用dataframe的groupby方法完成分组/聚合计算

python

```
import pandas as pd

df = pd.read_csv('gapminder.tsv', sep='\t')
# 每一年的平均预期寿命的多少？
print(df.groupby('year')['lifeExp'].mean())
# 每一年的平均人口和平均GDP是多少？
print(df.groupby('year')['pop'].mean())    # 平均人口
print(df.groupby('year')['gdpPercap'].mean())    # 平均GDP
# 每年每个大洲的平均预期寿命，平均人口，平均GDP情况又如何？
print(df.groupby(['continent', 'year'])[['lifeExp', 'pop', 'gdpPercap']].mean())
# 每个大洲列出了多少个国家和地区？
print(df.groupby('continent')['country'].nunique())
```

- df.groupby('year')：分组对象，可迭代对象，存的是已分组后的结果
- df.groupby('year')['lifeExp']：对每个分组取lifeExp列
- df.groupby('year')['lifeExp'].mean()：对每个分组中的lifeExp进行求平均值

分组频数计算

在数据分析中，一个常见的任务就是计算频数

- 可以使用nunique()方法计算，pandas series的唯一计数

- 可以使用value_counts()方法来获取pandas series的频数统计

python

```
import pandas as pd

df = pd.read_csv('gapminder.tsv', sep='\t')
# print(df.head())
# 每个大洲列出了多少个国家和地区?
# 方法一
print(df.groupby('continent')['country'].nunique())
# 方法二
print(df.groupby('continent')['country'].value_counts())
```

4. 简单绘图

可视化是在数据分析的每个步骤都非常重要，在理解或清理数据时，可视化有助于识别数据中的趋势

plot() 折线图

python

```
import pandas as pd

df = pd.read_csv('gapminder.tsv', sep='\t')
average_life_expectancy = df.groupby('year')['lifeExp'].mean() # 平均预期寿命
average_life_expectancy.plot()
```

hist() 直方图

python

```
average_life_expectancy.hist()
```

七、Pandas数据分析入门

1. 计算常用统计值

文件下载地址：[美国教育部公开大学数据平台](#)

♥ 进入网站后下载：All Data Files(全部数据文件)
下载好后进行解压，找到Most-Recent-Cohorts-Institution.csv这个文件
然后将要用的提取出来就行
提取代码

python

```
import pandas as pd

df = pd.read_csv('Most-Recent-Cohorts-Institution.csv', low_memory=False)
# 需要保留的列
columns_to_keep = [
    'INSTNM', 'CITY', 'STABBR', 'HBCU', 'MENONLY', 'WOMENONLY', 'RELAFFIL',
    'SATVRMID', 'SATMTMID', 'DISTANCEONLY',
    'UGDS', 'UGDS_WHITE', 'UGDS_BLACK', 'UGDS_HISP', 'UGDS_ASIAN', 'UGDS_AIAN',
    'UGDS_NHPI', 'UGDS_2MOR', 'UGDS_NRA', 'UGDS_UNKN', 'PPTUG_EF',
    'CURROPER', 'PCTPELL', 'PCTFLOAN', 'UG25ABV', 'MD_EARN_WNE_P10', 'GRAD_DEBT_MDN_SUPP'
]
df = df[columns_to_keep]
df.to_csv('colleges.csv', index=False)
```

加载数据后，可以通过计算最大值，最小值，平均值，分位数，方差等方式对数据的分布情况做基本了解

统计数值列，并进行转置

```
python
import pandas as pd

colleges = pd.read_csv('colleges.csv')

print(colleges.describe().T)
```

结果

```
python
```

	count	mean	std	min	25%	50%	75%	max
HBCU	5910.0	0.017259	0.130245	0.0000	0.000000	0.00000	0.000000	1.0000
MENONLY	5924.0	0.010297	0.100959	0.0000	0.000000	0.00000	0.000000	1.0000
WOMENONLY	5924.0	0.005064	0.070988	0.0000	0.000000	0.00000	0.000000	1.0000
RELAFFIL	878.0	56.037585	22.333529	22.0000	30.000000	54.00000	72.500000	110.0000
SATVRMID	965.0	581.603109	65.900615	395.0000	535.000000	575.00000	620.000000	760.0000
SATMTMID	965.0	575.289119	74.392549	395.0000	525.000000	564.00000	615.000000	785.0000
DISTANCEONLY	5924.0	0.010804	0.103386	0.0000	0.000000	0.00000	0.000000	1.0000
UGDS	5656.0	2488.466054	6157.338709	0.0000	116.000000	494.00000	2074.000000	156755.0000
UGDS_WHITE	5656.0	0.446514	0.279656	0.0000	0.200450	0.46430	0.675600	1.0000
UGDS_BLACK	5656.0	0.185987	0.221537	0.0000	0.038275	0.09900	0.245625	1.0000
UGDS_HISP	5656.0	0.207678	0.231515	0.0000	0.051700	0.12060	0.278975	1.0000
UGDS_ASIAN	5656.0	0.039212	0.076288	0.0000	0.003800	0.01575	0.039825	1.0000
UGDS_AIAN	5656.0	0.014218	0.074796	0.0000	0.000000	0.00220	0.007000	1.0000
UGDS_NHPI	5656.0	0.004394	0.031934	0.0000	0.000000	0.00050	0.002500	0.9983
UGDS_2MOR	5656.0	0.036967	0.043094	0.0000	0.008700	0.03190	0.050125	1.0000
UGDS_NRA	5656.0	0.022250	0.061891	0.0000	0.000000	0.00060	0.019600	1.0000
UGDS_UNKN	5656.0	0.039952	0.091475	0.0000	0.000000	0.01280	0.036100	1.0000
PPTUG_EF	5625.0	0.236278	0.265725	0.0000	0.000000	0.12320	0.420000	1.0000
CURROPER	6429.0	0.962669	0.189586	0.0000	1.000000	1.00000	1.000000	1.0000
PCTPELL	5612.0	0.424001	0.214319	0.0000	0.265150	0.39385	0.568250	1.0000
PCTFLOAN	5612.0	0.409069	0.274564	0.0000	0.152500	0.44265	0.631300	1.0000
UG25ABV	5558.0	0.351580	0.245947	0.0005	0.150325	0.31290	0.516875	1.0000
MD_EARN_WNE_P10	5280.0	43508.301136	17033.197929	8579.0000	31830.000000	40567.50000	51994.000000	143372.0000

统计对象和类型列

查看每个列的统计值

- pandas基于numpy，numpy支持的数据类型，pandas都支持
- np.object：字符串类型
- np.Categorical：类别类型

```
python
import numpy as np
colleges.describe(include=[np.object,np.Categorical]).T
```

```
python
import pandas as pd

colleges = pd.read_csv('colleges.csv')
print(colleges.describe(include='all').T)    # 统计所有的列，包括数值列和类别类型 字符串类型
# 老版本使用object，新版是str
print(colleges.describe(include='str').T)    # 类别类型，字符串类型
```

查看概况

通过info()方法了解不同字段的条目数量，数据类型，是否缺失及内存占用情况

python

```
import pandas as pd

colleges = pd.read_csv('colleges.csv')

print(colleges.info())
```

2. 常用排序方法

从最大的N个值中选取最小值→找到低成本高口碑电影

文件地址：[Kaggle/IMDB 5000 Movie Dataset](#)

nlargest() 方法

用nlargest() 方法，选出imdb_score分数最高的100个

python

```
import pandas as pd

movies = pd.read_csv('movie_metadata.csv')
print(movies.columns)
movies2 = movies[['movie_title', 'budget', 'imdb_score']]
movies2 = movies2.nlargest(100, 'imdb_score')
print(movies2)
```

nsmllest() 方法

使用nsmllest()方法再从中挑出预算最小的5部

python

```
import pandas as pd

movies = pd.read_csv('movie_metadata.csv')
print(movies.columns)
movies2 = movies[['movie_title', 'budget', 'imdb_score']]
movies2 = movies2.nlargest(100, 'imdb_score')
movies2 = movies2.nsmallest(5, 'budget')
print(movies2)
```

结果

python

	movie_title	budget	imdb_score
4924	Butterfly Girl	180000.0	8.7
4921	Children of Heaven	180000.0	8.5
4822	12 Angry Men	350000.0	8.9
4659	A Separation	500000.0	8.4
2242	Psycho	806947.0	8.5

通过排序选取每组的最大值→找到每年imdb评分最高的电影

sort_values() 方法

sort_values()按照年排序，ascending升序排列

python

```
import pandas as pd

movies = pd.read_csv('movie_metadata.csv')
print(movies.columns)
movies2 = movies[['movie_title', 'title_year', 'imdb_score']]
print(movies2.sort_values('title_year', ascending=False))
```

同时对'title_year', 'imdb_score'两列进行排序

python

```
import pandas as pd

movies = pd.read_csv('movie_metadata.csv')
print(movies.columns)
movies2 = movies[['movie_title', 'title_year', 'imdb_score']]
movies4 = movies2.sort_values(['title_year', 'imdb_score'], ascending=False)
print(movies4)
```

用drop_duplicates去重，只保留每年的第一条数据

python

```
import pandas as pd

movies = pd.read_csv('movie_metadata.csv')
print(movies.columns)
movies2 = movies[['movie_title', 'title_year', 'imdb_score']]
movies3 = movies2.sort_values(['title_year', 'imdb_score'], ascending=False)
movies4 = movies3.drop_duplicates('title_year')
print(movies4)
```

drop_duplicates()属性

- keep=值
 - first: 默认，保留第一个
 - last: 保留最后一个
 - False: 有重复的一个也不保留
- ignore_index: 默认False，保留原始的索引，设置为True相当于调用reset_index()

让'title_year'降序'imdb_score'升序

python

```
import pandas as pd

movies = pd.read_csv('movie_metadata.csv')
print(movies.columns)
movies2 = movies[['movie_title', 'title_year', 'imdb_score']]
movies3 = movies2.sort_values(['title_year', 'imdb_score'], ascending=[False, True])
print(movies3)
```

通过sort_values()排序可以给ascending赋值一个list，list每个True False控制对应的列升序降序

提取出每年，每种电影分级中预算少的电影→sort_values多列排序

多列排序时，ascending参数传入一个列表，列表一一对应

python

```
import pandas as pd

movies = pd.read_csv('movie_metadata.csv')
print(movies.columns)
movies = movies[['movie_title','title_year','content_rating','budget']]
# 提取出每年，每种电影分级中预算少的电影→sort_values多列排序
movies1 = movies.sort_values(['title_year','content_rating','budget'],ascending=[False,False,True])
# print(movies1.head(10))
# 去重，去重title_year，content_rating相同的数据
movies1.drop_duplicates(subset=['title_year','content_rating'])
print(movies1.head(10))
```

3.简单数据分析练习(租房数据)

1.加载数据，查看数据，数据准备

载入数据

python

```
import pandas as pd
house = pd.read_csv('house.csv')
```

把列名替换成英文

python

```
# 查看原始列名
print(house.columns)
# 将列名换成英文
house.columns =
['region','address','title','unit_type','area','price','floor','construction_time','orientation','update_time',
 'number_of_property_viewers','remark','url']
```

查看数据基本情况

python

```
print(house.head())
print(house.info())
```

2.找到租金最低和租金最高的房子

方法一

对价格进行排序，取第一个/最后一个,就是租金最低和最高的

python

```
# 租金最低
house_price_min = house['price'].sort_values().head(1)
print(f'租金最低的房子：{house_price_min}')

# 租金最高
house_price_max = house['price'].sort_values().tail(1)
print(f'租金最高的房子：{house_price_max}')
```

方法二

通过describe()方法知道最低/最高的价格

python

租金最低

```
house_price_min = house[house['price'] == 1500]
print(f'租金最低的房子: {house_price_min}')
```

租金最高

```
house_price_max = house[house['price'] == 30000]
print(f'租金最高的房子: {house_price_max}')
```

或者根据min()/max()获取最小值和最大值

python

租金最低

```
# house_price_min = house[house['price'] == 1500]
# house_price_min = house[house['price'] == house['price'].min()]
house_price_min = house.loc[house['price'] == house['price'].min()]
print(f'租金最低的房子: {house_price_min}')
```

租金最高

```
# house_price_max = house[house['price'] == 30000]
# house_price_max = house[house['price'] == house['price'].max()]
house_price_max = house.loc[house['price'] == house['price'].max()]
print(f'租金最高的房子: {house_price_max}')
```

建议使用loc，更加规范

3. 找到最近新上的10套房源

找到最近新上的10套房源

python

找到最近新上的10套房源

```
# house = house['update_time'].sort_values(ascending=False).head(10)
house = house.sort_values('update_time',ascending=False).head(10)
print(house)
```

查看所有更新时间

python

查看所有更新时间

```
house2 = house['update_time'].unique() # 统计去重之后的结果
# house2 = house['update_time'].nunique() # 统计去重之后的数量
print(house2)
```

4. 看房人数

python

平均值

```
house_mean = house['number_of_property_viewers'].mean()
print(f'看房人数平均值: {house_mean}')
```

中位数

```
house_median = house['number_of_property_viewers'].median()
print(f'看房人数中位数{house_median}')
```

不同看房人数的房源数量，as_index=False分组字段不作为行索引(默认为True)

python

不同看房人数的房源数量，as_index=False分组字段不作为行索引(默认为True)

```
# house3 = house.groupby('number_of_property_viewers')['title'].count() # 统计每个看房人数对应的房源数量
house3 = house.groupby('number_of_property_viewers',as_index=False)['title'].count()
```

as_index: 作为索引

```
print(house3)
```

画图

画图%matplotlib inline功能就是在jupyter notebook中内嵌绘图，并可以省略掉plt.show

```
python
# 画图
import matplotlib.pyplot as plt
house3.columns = ['title', 'count']
tmp_df = house3['count'].plot(kind='bar', figsize=(20, 10))
plt.show()
```

5. 房租价格分布

```
python
print(house['price'].describe())
print(house['price'].mean())      # 平均值
print(house['price'].median())    # 中位数
print(house['price'].min())        # 最小值
print(house['price'].max())        # 最大值
print(house['price'].std())        # 标准差
```

6. 看房人数最多的朝向

```
python
# 计算出每个朝向看房的人数
house4 = house.groupby('orientation', as_index=False)['number_of_property_viewers'].sum()
# 找出看房最多的朝向
house4 = house4[house4['number_of_property_viewers'] == house4['number_of_property_viewers'].max()]
print(f'看房最多的朝向: {house4}')
```

7. 房型分布情况

```
python
house5 = house.groupby('unit_type', as_index=False)['title'].count()
house5.columns = ['unit_type', 'count']
house5.set_index('unit_type', inplace=True)
house5['count'].plot(kind='bar', figsize=(20, 10))
plt.show()
```

💕 字体显示报错设置

```
python
# 字体设置
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
```

8. 最受欢迎的房型

```
python
tmp = house.groupby('unit_type', as_index=False).agg({'number_of_property_viewers': 'sum'})
tmp = tmp[tmp['number_of_property_viewers'] == tmp['number_of_property_viewers'].max()]
print(f'最受欢迎的房型: {tmp}')
```

9. 房子的平均租房价格(元/平米)

```
python

house.loc[:, 'yuan_square_meter'] = house['price']/house['area']      # 添加一个新列
yuan_square_meter = house['yuan_square_meter'].mean()
print(f'房子的平均租房价格(元/平米): {yuan_square_meter}')
```

10. 热门小区

```
python

# 先取出['address', 'number_of_property_viewers']这两列
# 按照address进行分组
# 然后将number_of_property_viewers进行求和统计
popular = house[['address', 'number_of_property_viewers']].groupby(by = ['address'], as_index = False).sum()
popular.sort_values('number_of_property_viewers' , ascending=False, inplace = True)      # 排序
print(popular)
```

11. 出租房源最多的小区

```
python

# 按照小区名进行分组，统计每个小区的数量，再按这个数量找到房源最多的小区
rent_out_the_most = house[['address', 'number_of_property_viewers']].groupby('address', as_index=False).count()
# 修改列名
rent_out_the_most.columns = ['address', 'count']
rent_out_the_most = rent_out_the_most.nlargest(10, 'count')
print(rent_out_the_most)
```

八、数据组合

1. 连接数据

组合数据的一种方式是使用“连接”

- 连接是把某行或某列追加到数据中
- 数据被分成了多份可以使用连接把数据拼接起来
- 把计算的结果追加到现有的数据集，可以使用连接

添加行

加载多份数据，并连接起来

```
python

import pandas as pd

# 加载数据
df1 = pd.read_csv('a_1.csv')
df2 = pd.read_csv('a_2.csv')
df3 = pd.read_csv('a_3.csv')

# 查看三条数据
print(df1)
print(df2)
print(df3)
```

结果



```
python

A    B    C    D
0  a1  b1  c1  d1
1  a2  b2  c2  d2
```

```

2  a3  b3  c3  d3
3  a4  b4  c4  d4
4  a5  b5  c5  d5
   A    B    C    D
0  a6  b6  c6  d6
1  a7  b7  c7  d7
2  a8  b8  c8  d8
3  a9  b9  c9  d9
4  a10 b10 c10 d10
   A    B    C    D
0  a11 b11 c11 d11
1  a12 b12 c12 d12
2  a13 b13 c13 d13
3  a14 b14 c14 d14
4  a15 b15 c15 d15

```

可以使用concat函数将上面3个dataframe连接起来，需将3个dataframe放到同一个列表中

```

python
# 拼接数据
df = pd.concat([df1, df2, df3])    # 将三个dataframe堆叠起来了
print(df)

```

♥♥ 结果

```

python
   A    B    C    D
0  a1  b1  c1  d1
1  a2  b2  c2  d2
2  a3  b3  c3  d3
3  a4  b4  c4  d4
4  a5  b5  c5  d5
0  a6  b6  c6  d6
1  a7  b7  c7  d7
2  a8  b8  c8  d8
3  a9  b9  c9  d9
4  a10 b10 c10 d10
0  a11 b11 c11 d11
1  a12 b12 c12 d12
2  a13 b13 c13 d13
3  a14 b14 c14 d14
4  a15 b15 c15 d15

```

上面的结果中可以看到，concat函数把三个dataframe连接在了一起(简单堆叠)，之后可以使用iloc，loc等方法取出连接后的数据子集

```

python
# 取出数据
print(df.iloc[0])    # 取出第0行
print(df.loc[0])     # 取出索引为0的每一行

```

使用concat连接dataframe和series

```

python
# 使用concat连接dataframe和series
# 生成新的series
new_series = pd.Series(['n1','n2','n3','n4'])
# 连接
df4 = pd.concat([df1,new_series])
print(df4)

```

💖 结果

```
python
      A    B    C    D    0
0  a1  b1  c1  d1  NaN
1  a2  b2  c2  d2  NaN
2  a3  b3  c3  d3  NaN
3  a4  b4  c4  d4  NaN
4  a5  b5  c5  d5  NaN
0  NaN NaN NaN NaN  n1
1  NaN NaN NaN NaN  n2
2  NaN NaN NaN NaN  n3
3  NaN NaN NaN NaN  n4
```

上面的结果中包含了NaN值，NaN是python用于表示‘缺失值’的方法，由于series是列数据，concat方法默认是添加行，由于series数据没有索引，所以添加了一个新列，缺失的部分用NaN填充

如果想要将['n1','n2','n3','n4']作为行连接到df1后，可以创建dataframe并指定列名

```
python
# 生成新的dataframe
new_dataframe = pd.DataFrame([['n1','n2','n3','n4']],columns=['A','B','C','D'])
df5 = pd.concat([df1,new_dataframe])
print(df5)
```

💖 结果

```
python
      A    B    C    D
0  a1  b1  c1  d1
1  a2  b2  c2  d2
2  a3  b3  c3  d3
3  a4  b4  c4  d4
4  a5  b5  c5  d5
0  n1  n2  n3  n4
```

concat连接多个对象

💖 **注:**append方法：Pandas 2.0及以上版本已弃用，后续版本都使用concat方法

concat可以连接多个对象，如果只需要向现有的dataframe追加一个对象，可以通过append实现

```
python
print(df1.append(df2)) # 废弃
print(pd.concat([df1, df2]))
```

忽略索引

如果是两个或者多个dataframe连接，可以通过ignore_index=True参数，忽略后面的dataframe的索引

```
python
# 忽略索引
print(pd.concat([df1, df2], ignore_index=True))
```

💖 结果

python

	A	B	C	D
0	a1	b1	c1	d1
1	a2	b2	c2	d2
2	a3	b3	c3	d3
3	a4	b4	c4	d4
4	a5	b5	c5	d5
5	a6	b6	c6	d6
6	a7	b7	c7	d7
7	a8	b8	c8	d8
8	a9	b9	c9	d9
9	a10	b10	c10	d10

将字典连接dataframe

向dataframe中添加一个字典的时候， 必须将字典转换为单行dataframe

python

```
# 将字典连接dataframe
data_dict = {'A':'n1','B':'n2','C':'n3','D':'n4'}
# 将字典转换为单行dataframe
data_dict = pd.DataFrame([data_dict])
df6 = pd.concat([df1,data_dict],ignore_index=True)
print(df6)
```

结果

python

	A	B	C	D
0	a1	b1	c1	d1
1	a2	b2	c2	d2
2	a3	b3	c3	d3
3	a4	b4	c4	d4
4	a5	b5	c5	d5
5	n1	n2	n3	n4

添加列

使用concat函数添加列，与添加行的方法类似，需要多传一个axis参数axis的默认值是index行添加，传入参数axis=“columns”即可按列添加

axis参数

- 0/index
- 1/columns

python

```
import pandas as pd

df1 = pd.read_csv('a_1.csv')
df2 = pd.read_csv('a_2.csv')
df3 = pd.read_csv('a_3.csv')

# 合并列
df = pd.concat([df1,df2,df3],axis=1)
print(df)
```

结果

```
python
A  B  C  D  A  B  C  D  A  B  C  D
0  a1 b1 c1 d1 a6 b6 c6 d6 a11 b11 c11 d11
1  a2 b2 c2 d2 a7 b7 c7 d7 a12 b12 c12 d12
2  a3 b3 c3 d3 a8 b8 c8 d8 a13 b13 c13 d13
3  a4 b4 c4 d4 a9 b9 c9 d9 a14 b14 c14 d14
4  a5 b5 c5 d5 a10 b10 c10 d10 a15 b15 c15 d15
```

通过列名获取子集

```
python
print(df['A'])
```

结果

```
python
A  A  A
0  a1  a6  a11
1  a2  a7  a12
2  a3  a8  a13
3  a4  a9  a14
4  a5  a10 a15
```

添加列

向dataframe添加一列，不需要调用函数，通过dataframe['列名']=[值]即可

```
python
# 向dataframe添加一列
df['new_col'] = ['n1','n2','n3','n4','n5']
# 添加Series
df['new_col2'] = pd.Series(['n1','n2','n3','n4','n5'])
print(df)
```

结果

```
python
A  B  C  D  A  B  C  D  A  B  C  D  new_col  new_col2
0  a1 b1 c1 d1 a6 b6 c6 d6 a11 b11 c11 d11      n1      n1
1  a2 b2 c2 d2 a7 b7 c7 d7 a12 b12 c12 d12      n2      n2
2  a3 b3 c3 d3 a8 b8 c8 d8 a13 b13 c13 d13      n3      n3
3  a4 b4 c4 d4 a9 b9 c9 d9 a14 b14 c14 d14      n4      n4
4  a5 b5 c5 d5 a10 b10 c10 d10 a15 b15 c15 d15      n5      n5
```

合并后可以重置索引

```
python
# 合并后可以重置索引
df_2 = pd.concat([df1,df2,df3],axis="columns",ignore_index=True)
print(df_2)
```

结果

```
python
0  1  2  3  4  5  6  7  8  9  10  11
0  a1  b1  c1  d1  a6  b6  c6  d6  a11 b11  c11  d11
1  a2  b2  c2  d2  a7  b7  c7  d7  a12 b12  c12  d12
2  a3  b3  c3  d3  a8  b8  c8  d8  a13 b13  c13  d13
3  a4  b4  c4  d4  a9  b9  c9  d9  a14 b14  c14  d14
4  a5  b5  c5  d5  a10 b10 c10 d10 a15 b15  c15  d15
```

concat连接具有不同行列索引的数据

不同列索引

将上面例子中的数据集做调整，修改列名

```
python
# 修改列名
df1.columns = ['A','B','C','D']
df2.columns = ['E','F','G','H']
df3.columns = ['A','C','F','H']
```

使用concat直接连接，数据会堆叠在一起，列名相同的数据会合并到一列，合并后不存在的数据会用NaN填充

```
python
# 合并数据
df_3 = pd.concat([df1,df2,df3])
print(df_3)
```

结果

```
python
A  B  C  D  E  F  G  H
0  a1  b1  c1  d1  NaN  NaN  NaN  NaN
1  a2  b2  c2  d2  NaN  NaN  NaN  NaN
2  a3  b3  c3  d3  NaN  NaN  NaN  NaN
3  a4  b4  c4  d4  NaN  NaN  NaN  NaN
4  a5  b5  c5  d5  NaN  NaN  NaN  NaN
0  NaN  NaN  NaN  NaN  a6  b6  c6  d6
1  NaN  NaN  NaN  NaN  a7  b7  c7  d7
2  NaN  NaN  NaN  NaN  a8  b8  c8  d8
3  NaN  NaN  NaN  NaN  a9  b9  c9  d9
4  NaN  NaN  NaN  NaN  a10 b10 c10 d10
0  a11  NaN  b11  NaN  NaN  c11  NaN  d11
1  a12  NaN  b12  NaN  NaN  c12  NaN  d12
2  a13  NaN  b13  NaN  NaN  c13  NaN  d13
3  a14  NaN  b14  NaN  NaN  c14  NaN  d14
4  a15  NaN  b15  NaN  NaN  c15  NaN  d15
```

如果在连接的时候指向保留所有数据集中都有的数据，可以数据join参数，默认是‘outer’所有数据，如果设置为‘inner’只保留数据中共有的部分

```
python
# join 需列名一致
df_4 = pd.concat([df1,df2,df3],join='inner')
print(df_4)
```

结果

```
python
Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4]
```

join参数

- outer：把df中的数据都放到合并的结果中
- inner：必须在两个表中都存在的行/列才会被合并到结果中

不同行索引

连接具有不同行索引的数据

```
python
# 修改行索引
df1.index = [0,1,2,3,4]
df2.index = [4,5,6,7,8]
df3.index = [0,2,5,7,9]
```

传入axis='columns'，连接后的dataframe按列添加，并匹配各自行索引，缺失值用NaN

```
python
# 合并数据
df_5 = pd.concat([df1,df2,df3],axis='columns')
print(df_5)
```

结果

```
python
      A   B   C   D   E   F   G   H   A   C   F   H
0   a1  b1  c1  d1  NaN  NaN  NaN  NaN  a11  b11  c11  d11
1   a2  b2  c2  d2  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
2   a3  b3  c3  d3  NaN  NaN  NaN  NaN  a12  b12  c12  d12
3   a4  b4  c4  d4  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
4   a5  b5  c5  d5   a6  b6  c6  d6  NaN  NaN  NaN  NaN
5  NaN  NaN  NaN  NaN   a7  b7  c7  d7  a13  b13  c13  d13
6  NaN  NaN  NaN  NaN   a8  b8  c8  d8  NaN  NaN  NaN  NaN
7  NaN  NaN  NaN  NaN   a9  b9  c9  d9  a14  b14  c14  d14
8  NaN  NaN  NaN  NaN  a10 b10 c10 d10  NaN  NaN  NaN  NaN
9  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  a15  b15  c15  d15
```

join

```
python
# join
df_6 = pd.concat([df1,df3],axis='columns',join='inner')
print(df_6)
```

结果

```
python
      A   B   C   D   A   C   F   H
0   a1  b1  c1  d1  a11  b11  c11  d11
2   a3  b3  c3  d3  a12  b12  c12  d12
```

2. 合并多个数据集

在使用concat连接数据时，涉及到了参数join(join='inner',join='outer')

数据库中可以依据共有数据把两个或者多个数据表组合起来，即join操作

dataframe也可以实现类似数据库的join操作

pandas可以通过pd.join命令组合数据，也可以通过pd.merge命令组合数据

- merge更灵活
- 如果想依据行索引来合并dataframe可以考虑使用join函数

加载数据

安装sqlalchemy库

```
● ● ● shell
    pip install sqlalchemy
```

read_sql_table函数可以从数据库中读取表

```
● ● ● python
import pandas as pd
from sqlalchemy import create_engine

# 连接数据库
engine = create_engine('mysql+pymysql://root:huihuia24@127.0.0.1:3306/databases_demo?charset=utf8mb4')

# 读取数据
df1 = pd.read_sql_table('auth_permission',engine,index_col='id')
df2 = pd.read_sql_table('django_content_type',engine,index_col='id')

print(df1)
print(df2)
```

一对一合并

最简单的合并只涉及两个dataframe——把一列与另一列连接，且要连接的列不含任何重复的值

先从数据表中提取部分数据，使其不含重复的值

```
● ● ● python
# 取值
auth_permission = df1.loc[[1,5,9,13,17,22,25,30,35,40]]
print(auth_permission)
```

♥♥ 结果

● ● ● python

	name	content_type_id		codename
id				
1	Can add log entry	1		add_logentry
5	Can add permission	3		add_permission
9	Can add group	2		add_group
13	Can add user	4		add_user
17	Can add content type	5		add_contenttype
22	Can change session	6		change_session
25	Can add book	7		add_book
30	Can change user	9		change_user
35	Can delete article	8		delete_article
40	Can view comment	10		view_comment

通过content_type_id列合并数据，how参数指定连接方式

- how='left' 对应SQL中的left outer保留左侧表中的所有key

- how='right' 对应SQL中的right outer保留右侧表中的所有key
- how='outer' 对应SQL中的full outer保留左右两侧表中的所有key
- how='inner' 对应SQL中的inner只保留左右两侧表中的都有key

💖 注：
pd.merge需要传两个表
df.merge时df本身就是左表，所以只要传右表

左连接

```
python
# 左连接
df_1 =
df2.merge(auth_permission[['name','content_type_id','codename']],left_on='id',right_on='content_type_id',how='left')
print(df_1)
```

💖 结果

	id	app_label	model	name	content_type_id	codename
0	1	admin	logentry	Can add log entry	1	add_logentry
1	8	article	article	Can delete article	8	delete_article
2	10	article	comment	Can view comment	10	view_comment
3	9	article	user	Can change user	9	change_user
4	2	auth	group	Can add group	2	add_group
5	3	auth	permission	Can add permission	3	add_permission
6	4	auth	user	Can add user	4	add_user
7	7	book	book	Can add book	7	add_book
8	5	contenttypes	contenttype	Can add content type	5	add_contenttype
9	6	sessions	session	Can change session	6	change_session

右连接

```
python
# 右连接
df_2 =
df2.merge(auth_permission[['name','content_type_id','codename']],left_on='id',right_on='content_type_id',how='right')
print(df_2)
```

💖 结果

	id	app_label	model	name	content_type_id	codename
0	1	admin	logentry	Can add log entry	1	add_logentry
1	3	auth	permission	Can add permission	3	add_permission
2	2	auth	group	Can add group	2	add_group
3	4	auth	user	Can add user	4	add_user
4	5	contenttypes	contenttype	Can add content type	5	add_contenttype
5	6	sessions	session	Can change session	6	change_session
6	7	book	book	Can add book	7	add_book
7	9	article	user	Can change user	9	change_user
8	8	article	article	Can delete article	8	delete_article
9	10	article	comment	Can view comment	10	view_comment

多对一合并

python

多对一合并

```
df_all = music.merge(music_type,left_on='type_id',right_on='id',how='left')
df_all = df_all[['song_name','singer','album','duration_ms','release_date','type_name']]
print(df_all.to_string())
```

结果

python

	song_name	singer	album	duration_ms	release_date	type_name
0	晚风告白	小阿七	晚风告白	218000	2023-05-12	流行
1	山海	草东没有派对	如常	267000	2020-07-25	摇滚
2	成都	赵雷	无法长大	328000	2016-12-21	民谣
3	月光奏鸣曲	贝多芬	古典精选	289000	1801-03-15	古典
4	Fly Me to the Moon	Frank Sinatra	爵士经典	234000	1964-08-01	爵士
5	飘向北方	薛之谦/邓紫棋	渡	298000	2017-11-02	嘻哈
6	花海	周杰伦	魔杰座	256000	2008-10-15	R&B
7	Faded	Alan Walker	Faded	212000	2015-12-04	电子
8	Take Me Home	John Denver	Country Roads	245000	1971-04-12	乡村
9	Enter Sandman	Metallica	Metallica	238000	1991-08-12	金属
10	菊次郎的夏天	久石让	菊次郎的夏天	189000	1999-05-26	轻音乐
11	牵丝戏	银临/Aki阿杰	腐草为萤	242000	2013-12-01	古风
12	At Last	Etta James	Etta James	225000	1960-11-01	蓝调
13	加州旅馆	老鹰乐队	加州旅馆	386000	1976-12-08	朋克
14	野狼Disco	宝石Gem	野狼Disco	220000	2019-09-02	说唱
15	字字句句	张碧晨	字字句句	248000	2022-07-18	流行
16	理想	赵雷	吉姆餐厅	312000	2014-10-19	民谣
17	卡农	帕赫贝尔	古典合集	265000	1680-01-01	古典
18	青花瓷	周杰伦	我很忙	259000	2007-11-02	流行
19	光年之外	邓紫棋	光年之外	231000	2016-12-30	R&B
20	Wake Me Up	Avicii	True	247000	2013-06-17	电子
21	南山南	马頔	孤岛	287000	2014-09-26	民谣
22	晴天	周杰伦	叶惠美	262000	2003-07-31	流行
23	海阔天空	Beyond	乐与怒	334000	1993-05-14	摇滚
24	梦中的婚礼	理查德·克莱德曼	钢琴精选	215000	1979-01-01	古典
25	水星记	郭顶	飞行器的执行周期	296000	2016-11-25	流行
26	稻香	周杰伦	魔杰座	243000	2008-10-15	流行
27	鼓楼	赵雷	无法长大	305000	2016-12-21	民谣
28	起风了	买辣椒也用券	起风了	278000	2017-02-12	流行
29	春风十里	鹿先森乐队	所有的酒	290000	2016-11-09	民谣
30	告白气球	周杰伦	床边故事	237000	2016-06-24	流行
31	七里香	周杰伦	七里香	254000	2004-08-03	流行
32	小幸运	田馥甄	我的少女时代	249000	2015-10-21	流行
33	平凡之路	朴树	猎户星座	275000	2014-07-16	民谣
34	往后余生	马良	往后余生	233000	2018-05-16	民谣
35	纸短情长	烟把儿乐队	纸短情长	217000	2018-03-05	民谣
36	病变	Cubi/Fi9江澈	病变	226000	2017-06-23	嘻哈
37	全部都是你	Dragon Pig	全部都是你	208000	2017-03-16	嘻哈
38	学不会	林俊杰	学不会	268000	2011-12-31	流行
39	江南	林俊杰	第二天堂	251000	2004-06-04	流行
40	演员	薛之谦	绅士	246000	2015-05-20	流行
41	认真的雪	薛之谦	薛之谦	253000	2006-06-09	流行
42	天外来物	薛之谦	天外来物	261000	2020-12-31	流行
43	大鱼	周深	大鱼海棠	272000	2016-05-20	古风
44	不染	毛不易	香蜜沉沉烬如霜	269000	2018-08-13	流行
45	消愁	毛不易	平凡的一天	292000	2017-09-01	民谣
46	小酒馆	陈粒	如也	229000	2016-07-26	民谣
47	追光者	岑宁儿	夏至未至	235000	2017-06-16	流行
48	星辰大海	黄霄雲	星辰大海	216000	2021-01-15	流行

计算每种类型音乐的平均时长

- to_timedelta将duration_ms列转变为timedelta数据类型
- 参数unit='ms' 时间单位
- dt.floor('s') dt.floor() 时间类型数据，按指定单位截断数据

```
python
# 计算每种类型音乐的平均时长
# 分组
df_1 = df_all.groupby('type_name')['duration_ms'].mean()
print(df_1)
```

💖 结果

```
python
type_name
R&B      243500.000000
乡村      245000.000000
古典      256333.333333
古风      257000.000000
嘻哈      244000.000000
摇滚      300500.000000
朋克      386000.000000
民谣      276800.000000
流行      252388.888889
爵士      234000.000000
电子      229500.000000
蓝调      225000.000000
说唱      220000.000000
轻音乐      189000.000000
金属      238000.000000
Name: duration_ms, dtype: float64
```

```
python
# 转换成时间
df_1 = pd.to_timedelta(df_1,unit='ms').dt.floor('s').sort_values()
print(df_1)
```

💖 结果

```
python
type_name
轻音乐    0 days 00:03:09
说唱      0 days 00:03:40
蓝调      0 days 00:03:45
电子      0 days 00:03:49
爵士      0 days 00:03:54
金属      0 days 00:03:58
R&B      0 days 00:04:03
嘻哈      0 days 00:04:04
乡村      0 days 00:04:05
流行      0 days 00:04:12
古典      0 days 00:04:16
古风      0 days 00:04:17
民谣      0 days 00:04:36
摇滚      0 days 00:05:00
朋克      0 days 00:06:26
Name: duration_ms, dtype: timedelta64[ns]
```

join合并

- 只能水平连接两个或多个pandas对象
- 对齐是靠被调用的dataframe的列索引或行索引和另一个对象的行索引(不能是列索引)
- 默认是左连接(也可以设为内连接，外连接，右连接)

```
python
# 合并数据
# outer外连接
df_all = df1.join(df2, lsuffix='_1', rsuffix='_2', how='outer')
print(df_all)
```

结果

```
python
```

	Symbol_1	Shares_1	Low_1	High_1	Symbol_2	Shares_2	Low_2	High_2
0	AAPL	50	120	140	AAPL	80	95	110
1	GE	100	30	40	TSLA	50	80	130
2	IBM	87	75	95	WMT	40	55	70
3	SLB	20	55	85	MSFT	60	320	380
4	TXN	500	15	23	NVDA	35	880	1200

将两个dataframe的Symbol设置为行索引，再次join数据

```
python
# 将两个dataframe的Symbol设置为行索引，再次join数据
df_all = df1.set_index('Symbol').join(df2.set_index('Symbol'), lsuffix='_1', rsuffix='_2', how='outer')
print(df_all)
```

结果

```
python
```

	Shares_1	Low_1	High_1	Shares_2	Low_2	High_2
Symbol						
AAPL	50.0	120.0	140.0	80.0	95.0	110.0
GE	100.0	30.0	40.0	NaN	NaN	NaN
IBM	87.0	75.0	95.0	NaN	NaN	NaN
MSFT	NaN	NaN	NaN	60.0	320.0	380.0
NVDA	NaN	NaN	NaN	35.0	880.0	1200.0
SLB	20.0	55.0	85.0	NaN	NaN	NaN
TSLA	NaN	NaN	NaN	50.0	80.0	130.0
TXN	500.0	15.0	23.0	NaN	NaN	NaN
WMT	NaN	NaN	NaN	40.0	55.0	70.0

九、缺失数据处理

1.NaN简介

pandas中的NaN值来自numpy库，numpy中缺失值有几种表示形式，NaN,NAN,nan,他们都一样
缺失值和其他类型的数据不同，他毫无意义，NaN不等于0,也不等于空串

```
python
from numpy import nan

print(nan==True)
print(nan==False)
print(nan==0)
print(nan=='')
```

显示结果

```
python
```

```
False
False
False
False
```

pandas提供了

isnull/isna方法，用于测试某个值是否为缺失值

notnull/notna方法也可以用于判断某个值是否为缺失值

```
python
```

```
import pandas as pd

print(pd.isnull(nan))
print(pd.notnull(nan))
print(pd.isna(nan))
print(pd.notna(nan))
```

显示结果

```
python
```

```
True
False
True
False
```

2. 缺失值从何而来

缺失值的来源有两个

- 原始数据包含缺失值
- 数据整理过程中产生缺失值

加载包含缺失的数据

```
csv
```

```
ident,site,dated
0,619,1927-02-08
1,622,1927-02-10
2,734,1939-01-07
3,735,1930-01-12
4,751,1930-02-26
5,752,1945-05-18
6,802,
7,815,?
8,830,nan
9,856,1968-11-22
```

加载数据时可以通过keep_default_na与na_values指定加载数据时的缺失值

默认

```
python
```

```
import pandas as pd

df = pd.read_csv('a.csv', na_values=[], keep_default_na=True)
print(df)
```

💖 结果

python

	ident	site	dated
0	0	619	1927-02-08
1	1	622	1927-02-10
2	2	734	1939-01-07
3	3	735	1930-01-12
4	4	751	1930-02-26
5	5	752	1945-05-18
6	6	802	NaN
7	7	815	?
8	8	830	NaN
9	9	856	1968-11-22

情况一

python

```
# 将问号设置也为缺失值
df1 = pd.read_csv('a.csv', na_values='?',keep_default_na=True)
print(df1)
```

💖 结果

python

	ident	site	dated
0	0	619	1927-02-08
1	1	622	1927-02-10
2	2	734	1939-01-07
3	3	735	1930-01-12
4	4	751	1930-02-26
5	5	752	1945-05-18
6	6	802	NaN
7	7	815	NaN
8	8	830	NaN
9	9	856	1968-11-22

情况二

python

```
df2 = pd.read_csv('a.csv', na_values='?',keep_default_na=False)
print(df2)
```

💖 结果

python

	ident	site	dated
0	0	619	1927-02-08
1	1	622	1927-02-10
2	2	734	1939-01-07
3	3	735	1930-01-12
4	4	751	1930-02-26
5	5	752	1945-05-18
6	6	802	
7	7	815	NaN
8	8	830	nan
9	9	856	1968-11-22

情况三

python

df3 = pd.read_csv('a.csv', na_values=[], keep_default_na=False)
print(df3)

结果

python

ident site dated
0 0 619 1927-02-08
1 1 622 1927-02-10
2 2 734 1939-01-07
3 3 735 1930-01-12
4 4 751 1930-02-26
5 5 752 1945-05-18
6 6 802
7 7 815 ?
8 8 830 nan
9 9 856 1968-11-22

3. 处理缺失值

文件下载地址：

- [train.csv](#)
- [test.csv](#)

计算缺失比例

加载数据

python

import pandas as pd

df1 = pd.read_csv('train.csv')
df2 = pd.read_csv('test.csv')
print(df1.info())

结果

python

<class 'pandas.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
Column Non-Null Count Dtype
--- ---
0 PassengerId 891 non-null int64
1 Survived 891 non-null int64
2 Pclass 891 non-null int64
3 Name 891 non-null str
4 Sex 891 non-null str
5 Age 714 non-null float64
6 SibSp 891 non-null int64
7 Parch 891 non-null int64
8 Ticket 891 non-null str
9 Fare 891 non-null float64
10 Cabin 204 non-null str
11 Embarked 889 non-null str
dtypes: float64(2), int64(5), str(5)
memory usage: 83.7 KB
None

此数据为泰坦尼克号生存预测数据，Survived字段，代表该名乘客是否获救

python

```
print(df1['Survived'].value_counts())
```

结果

python

```
Survived
0      549
1      342
Name: count, dtype: int64
```

百分比

python

```
print(df1['Survived'].value_counts(normalize=True))
```

结果

python

```
Survived
0      0.616162
1      0.383838
Name: proportion, dtype: float64
```

检测数据集中每一列中缺失值的百分比

python

```
# 计算所有的缺失值
null_all = df1.isnull().sum()
print(null_all)
```

结果

python

```
PassengerId      0
Survived          0
Pclass           0
Name             0
Sex              0
Age             177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin           687
Embarked         2
dtype: int64
```

python

```
# 计算缺失值比例
proportion = 100 * null_all / len(df1)
print(proportion)
```

结果

python

```
PassengerId      0.000000
Survived          0.000000
Pclass           0.000000
Name             0.000000
Sex              0.000000
Age             19.865320
SibSp            0.000000
Parch            0.000000
Ticket           0.000000
Fare             0.000000
Cabin           77.104377
Embarked         0.224467
dtype: float64
```

python

```
# 将结果拼成dataframe
null_1 = pd.concat([null_all, proportion], axis=1)
print(null_1)
```

结果

python

	0	1
PassengerId	0	0.000000
Survived	0	0.000000
Pclass	0	0.000000
Name	0	0.000000
Sex	0	0.000000
Age	177	19.865320
SibSp	0	0.000000
Parch	0	0.000000
Ticket	0	0.000000
Fare	0	0.000000
Cabin	687	77.104377
Embarked	2	0.224467

python

```
# 将列重命名
null_1.columns = ['缺失值', '占比(%)']
print(null_1)
```

结果

python

	缺失值	占比(%)
PassengerId	0	0.000000
Survived	0	0.000000
Pclass	0	0.000000
Name	0	0.000000
Sex	0	0.000000
Age	177	19.865320
SibSp	0	0.000000
Parch	0	0.000000
Ticket	0	0.000000
Fare	0	0.000000
Cabin	687	77.104377
Embarked	2	0.224467

```
python
# 按照缺失值降序排序，把缺失值为0的数据排除
null_1 = null_1[null_1.iloc[:,1] != 0].sort_values('占比(%)',ascending=False).round(1) # round(1)保留一位小数
print(null_1)
```

💖 结果

```
python
      缺失值  占比(%)
Cabin      687    77.1
Age        177    19.9
Embarked     2     0.2
```

完整代码

```
python
import pandas as pd

df1 = pd.read_csv('train.csv')
df2 = pd.read_csv('test.csv')
print(df1.info())

# 该名乘客是否获救
print(df1['Survived'].value_counts())
print(df1['Survived'].value_counts(normalize=True))

# 检测数据集中每一列中缺失值的百分比
# 计算所有的缺失值
null_all = df1.isnull().sum()
# 计算缺失值比例
proportion = 100 * null_all / len(df1)
# 将结果拼成dataframe
null_1 = pd.concat([null_all, proportion], axis=1)
# 将列重命名
null_1.columns = ['缺失值', '占比(%)']
# 按照缺失值降序排序，把缺失值为0的数据排除
null_1 = null_1[null_1.iloc[:,1] != 0].sort_values('占比(%)',ascending=False).round(1) # round(1)保留一位小数
print(null_1)
```

缺失值可视化

使用missingno库对缺失值进行可视化

- 我们可以使用missingno对缺失值进行可视化
- 使用missingno很简单

```
shell
pip install missingno
```

方法一

```
python
import missingno as msno
import pandas as pd
df1 = pd.read_csv('train.csv')
msno.bar(df1)
```

方法二

```
python
msno.matrix(df1)
```

数据缺失原因

查看缺失值之间是否具有相关性

```
python
msno.heatmap(df1)
```

4. 缺失值处理

删除缺失值

删除缺失值：删除缺失值会损失信息，并不推荐删除，当缺失数据占比较低的时候，可以尝试使用删除缺失值

```
python
import pandas as pd

df = pd.read_csv('train.csv')
# 复制一份
df1 = df.copy()
```

按行删除

.dropna() 属性值

- axis: 指定行/列
 - 0/index: 行
 - 1/columns: 列
- subset: 指定列名
- how
 - any: 每列其中有一行为空，就删除那一行
 - all: 一般用在多列，每列的同一行都为缺失值，就删除那行
- thresh: 满足指定缺失值数量，对应的行才被删掉

```
python
# 按行删除
df_1 = df1.dropna(axis=0,subset='Age',how='any')
print(df.info())
print(df_1.info())
```

对比

df1	df_1
-----	------

df1	df_1
<pre><class 'pandas.DataFrame'> RangeIndex: 891 entries, 0 to 890 Data columns (total 12 columns): # Column Non-Null Count Dtype --- --- 0 PassengerId 891 non-null int64 1 Survived 891 non-null int64 2 Pclass 891 non-null int64 3 Name 891 non-null str 4 Sex 891 non-null str 5 Age 714 non-null float64 6 SibSp 891 non-null int64 7 Parch 891 non-null int64 8 Ticket 891 non-null str 9 Fare 891 non-null float64 10 Cabin 204 non-null str 11 Embarked 889 non-null str dtypes: float64(2), int64(5), str(5) memory usage: 83.7 KB None</pre>	<pre><class 'pandas.DataFrame'> Index: 714 entries, 0 to 890 Data columns (total 12 columns): # Column Non-Null Count Dtype --- --- 0 PassengerId 714 non-null int64 1 Survived 714 non-null int64 2 Pclass 714 non-null int64 3 Name 714 non-null str 4 Sex 714 non-null str 5 Age 714 non-null float64 6 SibSp 714 non-null int64 7 Parch 714 non-null int64 8 Ticket 714 non-null str 9 Fare 714 non-null float64 10 Cabin 185 non-null str 11 Embarked 712 non-null str dtypes: float64(2), int64(5), str(5) memory usage: 72.5 KB None</pre>

填充缺失值

填充缺失值是指用一个估算的值来代替缺失数

df.fillna()

- value
 - 常数
 - 使用计算后的结果填充，例：

```
python
df['xx'].mean()
```

```
python
# 填充缺失值
df_2 = df1.fillna(0)
print(df_2)
```

时间序列缺失值处理

使用pandas的fillna来处理这类情况

- 用时间序列中空值的上一个非空值填充
- 用时间序列中空值的下一个非空值填充
- 线性插值方法

加载数据，数据集为印度城市空气质量数据(2015-2020)

💖 印度城市空气质量数据文件下载地址：[city_day.csv](#)

方法一

python

```
import pandas as pd

df = pd.read_csv('city_day.csv', index_col='Date')
print(df.info())
df_1 = pd.read_csv('city_day.csv', index_col='Date', parse_dates=True)
print(df_1.info())
```

对比

df	df_1
<class 'pandas.DataFrame'> Index: 29531 entries, 2015-01-01 to 2020-07-01 Data columns (total 15 columns): # Column Non-Null Count Dtype --- --- 0 City 29531 non-null str 1 PM2.5 24933 non-null float64 2 PM10 18391 non-null float64 3 NO 25949 non-null float64 4 NO2 25946 non-null float64 5 NOx 25346 non-null float64 6 NH3 19203 non-null float64 7 CO 27472 non-null float64 8 SO2 25677 non-null float64 9 O3 25509 non-null float64 10 Benzene 23908 non-null float64 11 Toluene 21490 non-null float64 12 Xylene 11422 non-null float64 13 AQI 24850 non-null float64 14 AQI_Bucket 24850 non-null str dtypes: float64(13), str(2) memory usage: 3.6+ MB None	<class 'pandas.DataFrame'> DatetimeIndex: 29531 entries, 2015-01-01 to 2020-07-01 Data columns (total 15 columns): # Column Non-Null Count Dtype --- --- 0 City 29531 non-null str 1 PM2.5 24933 non-null float64 2 PM10 18391 non-null float64 3 NO 25949 non-null float64 4 NO2 25946 non-null float64 5 NOx 25346 non-null float64 6 NH3 19203 non-null float64 7 CO 27472 non-null float64 8 SO2 25677 non-null float64 9 O3 25509 non-null float64 10 Benzene 23908 non-null float64 11 Toluene 21490 non-null float64 12 Xylene 11422 non-null float64 13 AQI 24850 non-null float64 14 AQI_Bucket 24850 non-null str dtypes: float64(13), str(2) memory usage: 3.6 MB None

方法二

python

```
df_2 = pd.read_csv('city_day.csv',parse_dates=['Date'])
print(df.info())
print(df_2.info())
```

对比

df	df_2
----	------

df	df_2
<pre><class 'pandas.DataFrame'> Index: 29531 entries, 2015-01-01 to 2020-07-01 Data columns (total 15 columns): # Column Non-Null Count Dtype --- --- --- 0 City 29531 non-null str 1 PM2.5 24933 non-null float64 2 PM10 18391 non-null float64 3 NO 25949 non-null float64 4 NO2 25946 non-null float64 5 NOx 25346 non-null float64 6 NH3 19203 non-null float64 7 CO 27472 non-null float64 8 SO2 25677 non-null float64 9 O3 25509 non-null float64 10 Benzene 23908 non-null float64 11 Toluene 21490 non-null float64 12 Xylene 11422 non-null float64 13 AQI 24850 non-null float64 14 AQI_Bucket 24850 non-null str dtypes: float64(13), str(2) memory usage: 3.6+ MB None</pre>	<pre><class 'pandas.DataFrame'> RangeIndex: 29531 entries, 0 to 29530 Data columns (total 16 columns): # Column Non-Null Count Dtype --- --- --- 0 City 29531 non-null str 1 Date 29531 non-null datetime64[us] 2 PM2.5 24933 non-null float64 3 PM10 18391 non-null float64 4 NO 25949 non-null float64 5 NO2 25946 non-null float64 6 NOx 25346 non-null float64 7 NH3 19203 non-null float64 8 CO 27472 non-null float64 9 SO2 25677 non-null float64 10 O3 25509 non-null float64 11 Benzene 23908 non-null float64 12 Toluene 21490 non-null float64 13 Xylene 11422 non-null float64 14 AQI 24850 non-null float64 15 AQI_Bucket 24850 non-null str dtypes: datetime64[us](1), float64(13), str(2) memory usage: 3.6 MB None</pre>

parse_dates如果设置为True，解析索引作为时间日期，如果直接传人列名，按照列名进行解析

统计缺失值

```
python
import pandas as pd

def missing_values(df):
    # 计算所有缺失值
    null_all = df.isnull().sum()
    # 计算缺失值比例
    proportion = 100 * null_all / len(df)
    # 将结果拼成dataframe
    null_df = pd.concat([null_all,proportion],axis=1)
    # 将列重命名
    null_df.columns = ['缺失值','占比(%)']
    # 将结果为0的去除，并排序
    null_df = null_df[null_df.iloc[:,1] != 0].sort_values('占比(%)',ascending=False).round(1)
    return null_df

city_day = pd.read_csv('city_day.csv', index_col='Date',parse_dates=True)
city_day_1 = city_day.copy()

# 统计缺失值
null_value = missing_values(city_day_1)
print(null_value)
```

结果

	缺失值	占比(%)
Xylene	18109	61.3
PM10	11140	37.7
NH3	10328	35.0
Toluene	8041	27.2
Benzene	5623	19.0

AQI	4681	15.9
AQI_Bucket	4681	15.9
PM2.5	4598	15.6
NOx	4185	14.2
O3	4022	13.6
SO2	3854	13.1
NO2	3585	12.1
NO	3582	12.1
CO	2059	7.0

数据中有很多缺失值比如Xylene(二甲苯)和PM10有超过50%的缺失值

python

```
# 查看包含缺失数据部分
print(city_day_1['Xylene'][50:64])
```

结果

python

```
Date
2015-02-20    7.48
2015-02-21   15.44
2015-02-22    8.47
2015-02-23   28.46
2015-02-24    6.05
2015-02-25    0.81
2015-02-26    NaN
2015-02-27    NaN
2015-02-28    NaN
2015-03-01    1.32
2015-03-02    0.22
2015-03-03    2.25
2015-03-04    1.55
2015-03-05    4.13
Name: Xylene, dtype: float64
```

使用ffill填充

用时间序列中空值的上一个非空值填充

注：pandas 2.0+版本废弃并移除了method、limit 等关键字参数，同时提供了更直观的替代方法

python

```
# ffill：用时间序列中空值的上一个非空值填充
# 老版本
# city_day_1 = city_day_ffill.fillna(method='ffill')
city_day_ffill = city_day_1.ffill()
print(city_day_ffill['Xylene'][50:64])
```

结果

python

```
Date
2015-02-20    7.48
2015-02-21   15.44
2015-02-22    8.47
2015-02-23   28.46
2015-02-24    6.05
2015-02-25    0.81
2015-02-26    0.81
2015-02-27    0.81
```

```
2015-02-28      0.81
2015-03-01      1.32
2015-03-02      0.22
2015-03-03      2.25
2015-03-04      1.55
2015-03-05      4.13
Name: Xylene, dtype: float64
```

使用bfill填充

用时间序列中空值的下一个非空值填充

python

```
# bfill: 用时间序列中空值的下一个非空值填充
city_day_bfill = city_day_1.bfill()
print(city_day_bfill['Xylene'][50:64])
```

结果

python

```
Date
2015-02-20      7.48
2015-02-21     15.44
2015-02-22      8.47
2015-02-23     28.46
2015-02-24      6.05
2015-02-25      0.81
2015-02-26      1.32
2015-02-27      1.32
2015-02-28      1.32
2015-03-01      1.32
2015-03-02      0.22
2015-03-03      2.25
2015-03-04      1.55
2015-03-05      4.13
Name: Xylene, dtype: float64
```

线性插值interpolate()

注:interpolate()是数值型数据专属的插值方法，仅能对int/float类型列进行线性推算

python

```
# 线性插值interpolate()
city_day_interpolate = city_day_1[['Xylene']].interpolate(numeric_only=True)
print(city_day_interpolate['Xylene'][50:64])
```

结果

python

```
Date
2015-02-20      7.4800
2015-02-21     15.4400
2015-02-22      8.4700
2015-02-23     28.4600
2015-02-24      6.0500
2015-02-25      0.8100
2015-02-26      0.9375
2015-02-27      1.0650
2015-02-28      1.1925
2015-03-01      1.3200
2015-03-02      0.2200
2015-03-03      2.2500
2015-03-04      1.5500
```

```
2015-03-05    4.1300
Name: Xylene, dtype: float64
```

对比

💖 对比

ffill	bfill	interpolate
Date	Date	Date
2015-02-20 7.48	2015-02-20 7.48	2015-02-20 7.4800
2015-02-21 15.44	2015-02-21 15.44	2015-02-21 15.4400
2015-02-22 8.47	2015-02-22 8.47	2015-02-22 8.4700
2015-02-23 28.46	2015-02-23 28.46	2015-02-23 28.4600
2015-02-24 6.05	2015-02-24 6.05	2015-02-24 6.0500
2015-02-25 0.81	2015-02-25 0.81	2015-02-25 0.8100
2015-02-26 0.81	2015-02-26 1.32	2015-02-26 0.9375
2015-02-27 0.81	2015-02-27 1.32	2015-02-27 1.0650
2015-02-28 0.81	2015-02-28 1.32	2015-02-28 1.1925
2015-03-01 1.32	2015-03-01 1.32	2015-03-01 1.3200
2015-03-02 0.22	2015-03-02 0.22	2015-03-02 0.2200
2015-03-03 2.25	2015-03-03 2.25	2015-03-03 2.2500
2015-03-04 1.55	2015-03-04 1.55	2015-03-04 1.5500
2015-03-05 4.13	2015-03-05 4.13	2015-03-05 4.1300
Name: Xylene, dtype: float64	Name: Xylene, dtype: float64	Name: Xylene, dtype: float64

十、整理数据

1.melt整理数据

💖 文件下载地址：[github-pew.csv](#)

加载美国收入与宗教信仰数据，这种数据称为“宽”数据

```
python
import pandas as pd

pew = pd.read_csv('pew.csv')
```

pandas的melt函数可以把宽数据集，转换为长数据集

melt即是类函数也是实例函数，也就是说可以pd.melt也可以使用df.melt()

参数	类型	说明
frame	dataframe	被melt的数据集名称在pd.melt()中使用
id_vars	tuple/list/ndarray	可选项不需要被转换的列名，在转换后作为标识符列(不是索引列)
value_vars	tuple/list/ndarray	可选项需要被转换的现有列如果未指明，除id_vars之外的其他列都被转换
var_name	string	variable默认值自定义列名名称设置由value_vars组成新的column name
value_name	string	value默认值自定义列名名称设置由value_vars的数据组成新的column name
col_level	int/string	可选项如果是MultiIndex，则使用此级别

使用melt对上面的pew数据集进行处理

```
python
# 使用melt对上面的pew数据集进行处理
pew_long = pd.melt(pew, id_vars='religion')
print(pew_long)
```

结果

```
python

   religion variable value
0    Agnostic    <$10k    27
1    Atheist    <$10k    12
2    Buddhist    <$10k    27
3    Catholic    <$10k   418
4  Don't know/refused    <$10k    15
..      ...      ...      ...
175  Orthodox Don't know/refused    73
176  Other Christian Don't know/refused    18
177    Other Faiths Don't know/refused    71
178  Other World Religions Don't know/refused     8
179    Unaffiliated Don't know/refused   597

[180 rows x 3 columns]
```

```
python
# 指定列名
pew_long = pd.melt(pew, id_vars='religion', var_name='a', value_name='b')
print(pew_long)
```

结果

```
python

   religion      a      b
0    Agnostic    <$10k    27
1    Atheist    <$10k    12
2    Buddhist    <$10k    27
3    Catholic    <$10k   418
4  Don't know/refused    <$10k    15
..      ...      ...      ...
175  Orthodox Don't know/refused    73
176  Other Christian Don't know/refused    18
177    Other Faiths Don't know/refused    71
178  Other World Religions Don't know/refused     8
179    Unaffiliated Don't know/refused   597

[180 rows x 3 columns]
```

转换少数列

在使用melt函数转换数据的时候，也可以固定多列数据，只转换少数列

文件下载地址：[billboard.csv](#)

```
python
import pandas as pd

df = pd.read_csv('billboard.csv')
```

使用melt对上面数据的week进行处理，转换成成长数据

```
python
import pandas as pd

df = pd.read_csv('billboard.csv')

billboard_long = pd.melt(df, id_vars=
['year','artist','track','time','date.entered'],var_name='week',value_name='rating')
print(billboard_long)
```

结果

```
python
year      artist  ... week rating
0      2000      2 Pac  ...  wk1  87.0
1      2000      2Ge+her  ...  wk1  91.0
2      2000      3 Doors Down  ...  wk1  81.0
3      2000      3 Doors Down  ...  wk1  76.0
4      2000      504 Boyz  ...  wk1  57.0
...      ...      ...  ...  ...  ...
24087  2000      Yankee Grey  ...  wk76  NaN
24088  2000  Yearwood, Trisha  ...  wk76  NaN
24089  2000      Ying Yang Twins  ...  wk76  NaN
24090  2000      Zombie Nation  ...  wk76  NaN
24091  2000  matchbox twenty  ...  wk76  NaN

[24092 rows x 7 columns]
```

可以将上述数据进一步处理，当我们查询任意一首歌曲信息时，会发现数据的存储有冗余情况

```
python
# 查找歌曲为Loser的所有行
print(billboard_long[billboard_long['track'] == 'Loser'])
```

结果

```
python
year      artist track  time date.entered  week  rating
3      2000  3 Doors Down  Loser  4:24  2000-10-21  wk1  76.0
320    2000  3 Doors Down  Loser  4:24  2000-10-21  wk2  76.0
637    2000  3 Doors Down  Loser  4:24  2000-10-21  wk3  72.0
954    2000  3 Doors Down  Loser  4:24  2000-10-21  wk4  69.0
1271   2000  3 Doors Down  Loser  4:24  2000-10-21  wk5  67.0
...      ...      ...  ...  ...  ...  ...  ...
22510  2000  3 Doors Down  Loser  4:24  2000-10-21  wk72  NaN
22827  2000  3 Doors Down  Loser  4:24  2000-10-21  wk73  NaN
23144  2000  3 Doors Down  Loser  4:24  2000-10-21  wk74  NaN
23461  2000  3 Doors Down  Loser  4:24  2000-10-21  wk75  NaN
23778  2000  3 Doors Down  Loser  4:24  2000-10-21  wk76  NaN

[76 rows x 7 columns]
```

实际上，上面的数据包含了两类数据，歌曲信息，周排行信息

- 减少上表中保存的歌曲信息，可以节省存储空间，需要完整信息的时候，可以通过merge拼接数据
- 我们可以把year,artist,track,time,date.entered放入一个新的dataframe中
- 相当于数据库中的左/右表

```
python
# 减少重复
billboard_songs = billboard_long[['year','artist','track','time','date.entered']].drop_duplicates()
print(billboard_songs)
```

添加id列

```
python
# 为上述表添加列
billboard_songs['id'] = round(len(billboard_songs))
```

将id列关联到原始数据，得到包含id的完整数据,并从完整数据中，取出每周评分部分，去掉冗余信息

```
python
billboard_ratings = billboard_long.merge(billboard_songs, on=['year','artist','track','time','date.entered'])
print(billboard_ratings)
```

结果

```
python
year      artist      track  ...  week rating  id
0      2000      2 Pac  Baby Don't Cry (Keep...  ...  wk1    87.0  317
1      2000      2Ge+her  The Hardest Part Of ...  ...  wk1    91.0  317
2      2000      3 Doors Down      Kryptonite  ...  wk1    81.0  317
3      2000      3 Doors Down      Loser  ...  wk1    76.0  317
4      2000      504 Boyz      Wobble Wobble  ...  wk1    57.0  317
...      ...      ...      ...  ...  ...      ...      ...
24087  2000      Yankee Grey      Another Nine Minutes  ...  wk76     NaN  317
24088  2000  Yearwood, Trisha      Real Live Woman  ...  wk76     NaN  317
24089  2000      Ying Yang Twins  Whistle While You Tw...  ...  wk76     NaN  317
24090  2000      Zombie Nation      Kernkraft 400  ...  wk76     NaN  317
24091  2000      matchbox twenty      Bent  ...  wk76     NaN  317

[24092 rows x 8 columns]
```

```
python
billboard_ratings = billboard_ratings[['id','week','rating']]
print(billboard_ratings)
```

结果

```
python
id  week  rating
0    317  wk1    87.0
1    317  wk1    91.0
2    317  wk1    81.0
3    317  wk1    76.0
4    317  wk1    57.0
...    ...    ...    ...
24087  317  wk76     NaN
24088  317  wk76     NaN
24089  317  wk76     NaN
24090  317  wk76     NaN
24091  317  wk76     NaN

[24092 rows x 3 columns]
```

完整代码

```
python
import pandas as pd

df = pd.read_csv('billboard.csv')

billboard_long= pd.melt(df, id_vars=
['year','artist','track','time','date.entered'],var_name='week',value_name='rating')
```

```
print billboard_long)

# 查找歌曲为Loser的所有行
print billboard_long[ billboard_long['track'] == 'Loser' ])

# 减少重复
billboard_songs = billboard_long[['year','artist','track','time','date.entered']].drop_duplicates()
# 为上述表添加列
billboard_songs['id'] = range(len(billboard_songs))
print billboard_songs)

# 将id列关联到原始数据，得到包含id的完整数据, 并从完整数据中，取出每周评分部分，去掉冗余信息
billboard_ratings = billboard_long.merge(billboard_songs, on=['year','artist','track','time','date.entered'])
print billboard_ratings)
billboard_ratings = billboard_ratings[['id','week','rating']]
print billboard_ratings)

# 合并表
billboard = billboard_songs.merge(billboard_ratings, on='id', how='left')
print billboard)
```