数据分析DAY03

数据加载

IO TOOLS (TEXT, CSV, HDF5, . . .)

The pandas I/O API is a set of top level `reader` functions accessed like *`pandas.read_csv()`* that generally return a pandas object. The corresponding `writer` functions are object methods that are accessed like *`DataFrame.to_csv()`*. Below is a table containing available `readers` and `writers`.

Format Type	Data Description	Reader	Writer
text	CSV	*read_csv*	*to_csv*
text	JSON	*read_json*	*to_json*
text	HTML	*read_html*	*to_html*
text	Local clipboard	*read_clipboard*	*to_clipboard*
binary	MS Excel	*read_excel*	*to_excel*
binary	HDF5 Format	*read_hdf*	*to_hdf*
binary	Feather Format	*read_feather*	*to_feather*
binary	Parquet Format	*read_parquet*	*to_parquet*
binary	Msgpack	*read_msgpack*	*to_msgpack*
binary	Stata	*read_stata*	*to_stata*
binary	SAS	*read_sas*	
binary	Python Pickle Format	*read_pickle*	*to_pickle*
SQL	SQL	*read_sql*	*to_sql*
SQL	Google Big Query	*read_gbq*	*to_gbq*

.

处理普通文本

读取文本：read_csv() read_table()

方法参数	参数解释
filepath_or_buffer	文件路径
sep	列之间的分隔符。read_csv()默认为为',', read_table()默认为'\t'
header	默认将首行设为列名。`header=None`时应手动给出列名。
names	`header=None`时设置此字段使用列表初始化列名。
index_col	将某一列作为行级索引。若使用列表，则设置复合索引。
usecols	选择读取文件中的某些列。设置为为相应列的索引列表。

方法参数	参数解释
skiprows	跳过行。可选择跳过前n行或给出跳过的行索引列表。
encoding	编码。

写入文本：dataFrame.to_csv()

方法参数	参数解释
filepath_or_buffer	文件路径
sep	列之间的分隔符。默认为','
na_rep	写入文件时dataFrame中缺失值的内容。默认空字符串。
columns	定义需要写入文件的列。
header	是否需要写入表头。默认为True。
index	会否需要写入行索引。默认为True。
encoding	编码。

案例：读取电信数据集。

```
pd.read_csv('../data/CustomerSurvival.csv', header=None, index_col=0)
```

处理JSON

读取json：read_json()

方法参数	参数解释
filepath_or_buffer	文件路径
encoding	编码。

案例：读取电影评分数据：

```
pd.read_json('../data/ratings.json')
```

写入json：to_json()

方法参数	参数解释
filepath_or_buffer	文件路径； 若设置为None，则返回json字符串
orient	设置面向输出格式：['records', 'index', 'columns', 'values']

案例：

```
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data, index=['s1','s2','s3','s4'])
df.to_json(orient='records')
```

其他文件读取方法参见：https://www.pypandas.cn/docs/user_guide/io.html

数值型描述统计（统计学）

算数平均值

$S = [s_1, s_2, ..., s_n] $

样本中的每个值都是真值与误差的和。

$mean = \frac{(s_1 + s_2 + ... + s_n) }{n}$

算数平均值表示对真值的无偏估计。

```
m = np.mean(array)
m = array.mean()
m = df.mean(axis=0)
```

案例：针对电影评分数据做均值分析：

```
mean = ratings['John Carson'].mean()
mean = np.mean(ratings['John Carson'])
means = ratings.mean(axis=1)
```

加权平均值

求平均值时，考虑不同样本的重要性，可以为不同的样本赋予不同的权重。

样本：$S = [s_1, s_2, s_3 ... s_n]$

权重：$W =[w_1, w_2, w_3 ... w_n]$

加权平均值：

$$a = \frac{s_1w_1 + s_2w_2 + ... + s_nw_n}{w_1 + w_2 + ... + w_n}$$

代码实现：

```
a = np.average(array, weights=volumes)
```

案例：自定义权重，求加权平均。

```
# 加权均值
w = np.array([3,1,1,1,1,1,1])
np.average(ratings.loc['Inception'], weights=w)

mask = ~pd.isna(ratings.loc['Inception'])
np.average(ratings.loc['Inception'][mask], weights=w[mask])
```

最值

np.max() / np.min() / np.ptp()： 返回一个数组中最大值/最小值/极差（最大值减最小值）

```
import numpy as np
# 产生9个介于[10, 100)区间的随机数
a = np.random.randint(10, 100, 9)
print(a)
print(np.max(a), np.min(a), np.ptp(a))
```

np.argmax() np.argmin()： 返回一个数组中最大/最小元素的下标

```
print(np.argmax(a), np.argmin(a))
print(series.idxmax(), series.idxmin())
print(dataframe.idxmax(), dataframe.idxmin())
```

np.maximum() np.minimum()： 将两个同维数组中对应元素中最大/最小元素构成一个新的数组

```
print(np.maximum(a, b), np.minimum(a, b), sep='\n')
```

中位数

将多个样本按照大小排序，取中间位置的元素。

若样本数量为奇数，中位数为最中间的元素

$[1, 2000, 3000, 4000, 10000000]$

若样本数量为偶数，中位数为最中间的两个元素的平均值

$[1,2000,3000,4000,5000,10000000]$

案例：分析中位数的算法，测试numpy提供的中位数API：

```
import numpy as np
closing_prices = np.loadtxt('../../data/aapl.csv',
    delimiter=',', usecols=(6), unpack=True)
size = closing_prices.size
sorted_prices = np.msort(closing_prices)
median = (sorted_prices[int((size - 1) / 2)] +
          sorted_prices[int(size / 2)]) / 2
print(median)
median = np.median(closing_prices)
print(median)
```

频数与众数

频数指一组数据中各离散值出现的次数，而众数则是指一组数据中出现次数最多的值。

```
cars = np.array(['bmw', 'bmw', 'bz', 'audi', 'bz', 'bmw'])
cars = pd.Series(cars)
cars.value_counts()
cars.mode()
```

四分位数

所谓四分位数，即把数值由小到大排列并分成四等份，处于三个分割点位置的数值就是四分位数。

- 第1四分位数 (Q1)，又称“较小四分位数”，等于该样本中所有数值由小到大排列后第25%的数字。
- 第2四分位数 (Q2)，又称“中位数”，等于该样本中所有数值由小到大排列后第50%的数字。
- 第3四分位数 (Q3)，又称“较大四分位数”，等于该样本中所有数值由小到大排列后第75%的数字。

第3四分位数与第1四分位数的差距又称四分位距（InterQuartile Range,IQR）

```
ary = np.array([1,1,1,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3,3,4,4,4,4,5,5,5])
s = pd.Series(ary)
s.quantile([.0, .25, .5, .75, 1.])
```

标准差

样本（sample）：

$$S = [s_1, s_2, s_3, \ldots, s_n]$$

平均值：

$$m = \frac{s_1 + s_2 + s_3 + \ldots + s_n}{n}$$

离差（deviation）：表示某组数据距离某个中心点的偏离程度

$$D = [d_1, d_2, d_3, \ldots, d_n]$$
$$d_i = S_i - m$$

离差方：

$$Q = [q_1, q_2, q_3, \ldots, q_n]$$
$$q_i = d_i^2$$

总体方差（variance）：

$$v = \frac{(q_1 + q_2 + q_3 + \ldots + q_n)}{n}$$

总体标准差（standard deviation）：

$$s = \sqrt{v}$$

样本方差：

$$v' = \frac{(q_1 + q_2 + q_3 + \ldots + q_n)}{n - 1}$$

其中，n-1称之为“贝塞尔校正”，这是因为抽取样本时候，采集的样本主要是落在中心值附近，那么通过这些样本计算的方差会小于等于对总体数据集方差的无偏估计值。为了能弥补这方面的缺陷，那么我们把公式的n改为n-1,以此来提高方差的数值。称为贝塞尔校正系数。

样本标准差：

$$s' = \sqrt{v'}$$

案例： 根据标准差理论，针对评分数据进行方差分析：

```
ratings.std(axis=0)
```

宏观数值统计

```
ratings.describe()
```

协方差、相关矩阵、相关系数

通过两组统计数据计算而得的协方差可以评估这两组统计数据的相似程度（相关性）。

样本：

```
A = [a1, a2, ..., an]
B = [b1, b2, ..., bn]
```

平均值：

```
ave_a = (a1 + a2 +...+ an)/n
ave_b = (b1 + b2 +...+ bn)/n
```

离差（用样本中的每一个元素减去平均数，求得数据的误差程度）：

```
dev_a = [a1, a2, ..., an] - ave_a
dev_b = [b1, b2, ..., bn] - ave_b
```

协方差

协方差可以简单反映两组统计样本的相关性，值为正，则为正相关；值为负，则为负相关，绝对值越大相关性越强。

```
cov_ab = ave(dev_a x dev_b)
cov_ba = ave(dev_b x dev_a)
```

案例：计算两组股票数据的协方差，得出分析结论。

```
bhp = pd.read_csv('../data/bhp.csv', header=None, usecols=[6], names=['closing'])
vale = pd.read_csv('../data/vale.csv', header=None, usecols=[6], names=['closing'])

#平均值
ave_bhp = np.mean(bhp)
ave_vale = np.mean(vale)
#离差
dev_bhp = bhp - ave_bhp
dev_vale = vale - ave_vale
#协方差
cov_ab = np.mean(dev_bhp*dev_vale)
cov_ab
```

相关系数

协方差除去两组统计样本标准差的乘积是一个[-1, 1]之间的数。该结果称为统计样本的相关系数。

```
# a组样本 与 b组样本做对照后的相关系数
cov_ab/(std_a x std_b)
# b组样本 与 a组样本做对照后的相关系数
cov_ba/(std_b x std_a)
# a样本与a样本作对照    b样本与b样本做对照    二者必然相等
cov_ab/(std_a x std_b)=cov_ba/(std_b x std_a)
```

通过相关系数可以分析两组数据的相关性：

```
若相关系数越接近于0，越表示两组样本越不相关。
若相关系数越接近于1，越表示两组样本正相关。
若相关系数越接近于-1，越表示两组样本负相关。
```

案例：输出案例中两组数据的相关系数。

```
print('相关系数：', cov_ab/(np.std(a)*np.std(b)), cov_ba/(np.std(a)*np.std(b)))
```

相关矩阵

$$\begin{bmatrix} \frac{var\_a}{std\_a \times std\_a} & \frac{cov\_ab}{std\_a \times std\_b} \\ \frac{cov\_ba}{std\_b \times std\_a} & \frac{var\_b}{std\_b \times std\_b} \end{bmatrix}$$

矩阵正对角线上的值都为1。（同组样本自己相比绝对正相关）

$$\begin{bmatrix} 1 & \frac{cov\_ab}{std\_a \times std\_b} \\ \frac{cov\_ba}{std\_b \times std\_a} & 1 \end{bmatrix}$$

numpy与pandas提供了求得相关矩阵与协方差矩阵的API：

```
# 相关矩阵
np.corrcoef(a, b)
dataFrame.corr()

# 协方差矩阵
# [[aa的协方差，ab的协方差], [ba的协方差，bb的协方差]]
np.cov(a, b)
dataFrame.cov()
```

项目：保健品消费情况特征描述性统计分析

```
...
...
...
...
```

apply函数

pandas提供了apply函数方便的处理Series与DataFrame；apply函数支持逐一处理数据集中的每个元素都会执行一次目标函数，把返回值存入结果集中。：

```
# series.apply()
ary = np.array(['80公斤','83公斤','78公斤','74公斤','84公斤'])
s = pd.Series(ary)
def func(x):
    return x[:2]
s.apply(func)

# dataframe.apply()
def func(x):
    x[pd.isna(x)] = x.mean()
    return x
ratings.apply(func, axis=1)
```

排序

*Pandas*有两种排序方式，它们分别是按标签与按实际值排序。

```
import numpy as np

d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Minsu','Jack','Lee','David','G
asper','Betina','Andres']),
 'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
 'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}
unsorted_df = pd.DataFrame(d)
```

按标签（行）排序

使用sort_index()方法，通过传递axis参数和排序顺序，可以对DataFrame进行排序。 默认情况下，按照升序对行标签进行排序。

```
# 按照行标进行排序
sorted_df=unsorted_df.sort_index()
print (sorted_df)
# 控制排序顺序
sorted_df = unsorted_df.sort_index(ascending=False)
print (sorted_df)
```

按标签（列）排序

```
# 按照列标签进行排序
sorted_df=unsorted_df.sort_index(axis=1)
print (sorted_df)
```

按某列值排序

像索引排序一样，sort_values()是按值排序的方法。它接受一个by参数，它将使用要与其排序值的DataFrame的列名称。

```
sorted_df = unsorted_df.sort_values(by='Age')
print (sorted_df)
# 先按Age进行升序排序，然后按Rating降序排序
sorted_df = unsorted_df.sort_values(by=['Age', 'Rating'], ascending=[True, False])
print (sorted_df)
```

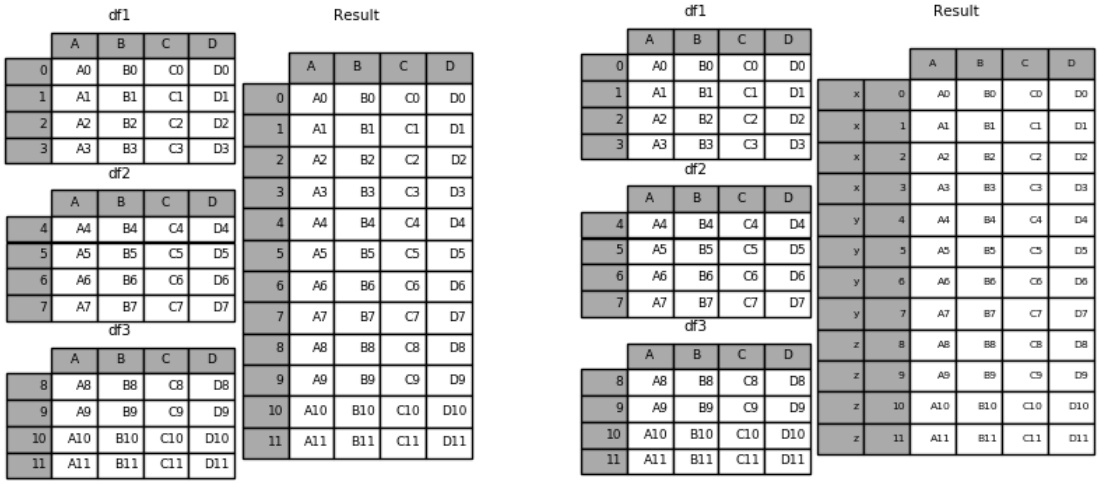
数据合并

concat

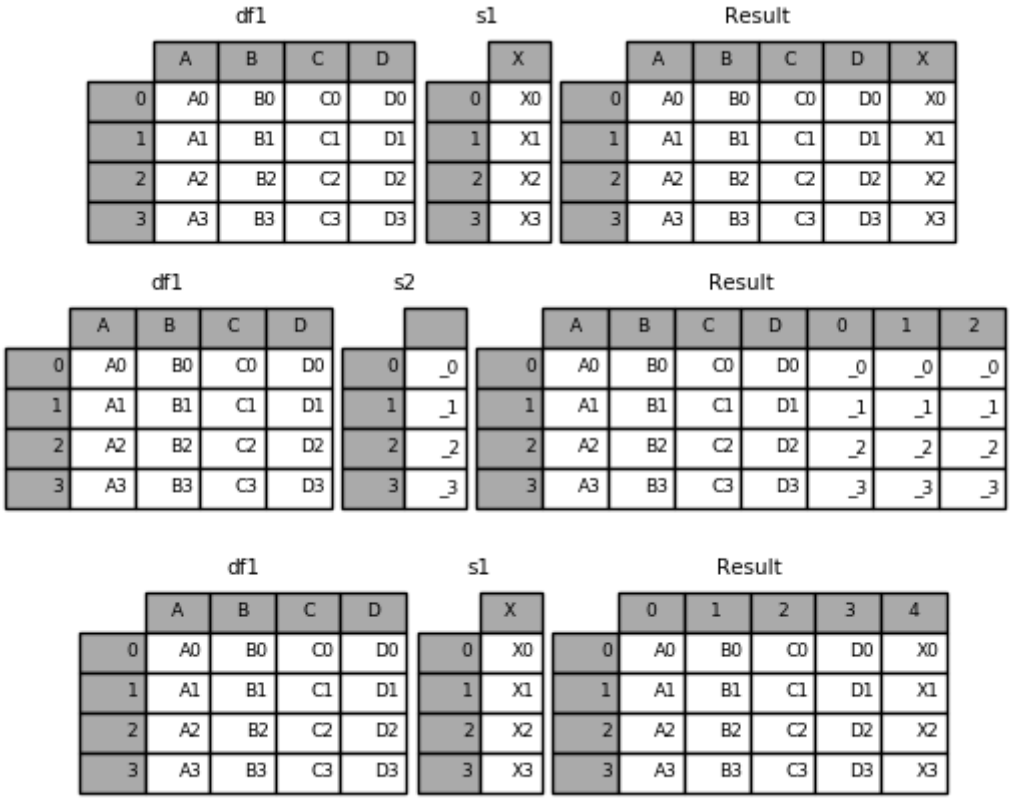
concat函数是在pandas的方法，可以根据不同的轴合并数据集。

```
r = pd.concat(datas, axis=0, join='outer', ignore_index=False,
              keys=['x', 'y', 'z'])
```

纵向合并：



横向合并：



merge & join

panda具有全功能、高性能的内存连接操作，与SQL之类的关系数据库非常相似。与其他开源实现相比，这些方法的性能要好得多(在某些情况下要好一个数量级以上)

pandas提供了merge函数实现高效的内存链接操作：

```
pd.merge(left, right, how='inner', on=None, left_on=None,
right_on=None,left_index=False, right_index=False)
```

参数名称	说明
left	接收DataFrame或Series。表示要添加的新数据。无默认。
right	接收DataFrame或Series。表示要添加的新数据。无默认。。
how	接收inner，outer，left，right。表示数据的连接方式。默认为inner。
on	接收string或sequence。表示外键字段名。默认为None。
left_on	接收string或sequence。关联操作时左表中的关联字段名。
right_on	接收string或sequence。关联操作时右表中的关联字段名。
left_index	接收boolean。表示是否将left参数接收数据的index作为连接主键。默认为False。
right_index	接收boolean。表示是否将right参数接收数据的index作为连接主键。默认为False。
sort	接收boolean。表示是否根据连接键对合并后的数据进行排序。默认为False。
suffixes	接收接收tuple。表示用于追加到left和right参数接收数据重叠列名的尾缀默认为('*x*', 'y')。

合并两个DataFrame：

```
import pandas as pd
left = pd.DataFrame({
         'student_id':[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20],
         'student_name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung', 'Billy', 'Brian',
'Bran', 'Bryce', 'Betty', 'Emma', 'Marry', 'Allen', 'Jean', 'Rose', 'David', 'Tom',
'Jack', 'Daniel', 'Andrew'],
         'class_id':[1,1,1,2,2,2,3,3,3,4,1,1,1,2,2,2,3,3,3,2],
         'gender':['M', 'M', 'F', 'F', 'M', 'M', 'F', 'F', 'M', 'M', 'F', 'F', 'M',
'M', 'F', 'F', 'M', 'M', 'F', 'F'],
         'age':[20,21,22,20,21,22,23,20,21,22,20,21,22,23,20,21,22,20,21,22],
         'score':[98,74,67,38,65,29,32,34,85,64,52,38,26,89,68,46,32,78,79,87]})
right = pd.DataFrame(
         {'class_id':[1,2,3,5],
         'class_name': ['ClassA', 'ClassB', 'ClassC', 'ClassE']})
# 合并两个DataFrame
data = pd.merge(left,right)
print(data)
```

其他合并方法同数据库相同：

合并方法	SQL等效	描述
left	LEFT OUTER JOIN	使用左侧对象的键
right	RIGHT OUTER JOIN	使用右侧对象的键
outer	FULL OUTER JOIN	使用键的联合
inner	INNER JOIN	使用键的交集

实验：

```
# 合并两个DataFrame (左连接)
rs = pd.merge(left,right,on='subject_id', how='right')
print(rs)
# 合并两个DataFrame (左连接)
rs = pd.merge(left,right,on='subject_id', how='outer')
print(rs)
# 合并两个DataFrame (左连接)
rs = pd.merge(left,right,on='subject_id', how='inner')
print(rs)
```

###