# 数据获取、抽样 、探索、预处理，管道函数的方法汇总

**一、创建数据对象**

机器学学习数据形式一般来说array或者dataframe。他是由python基础数据结构对象——元素（字符串）、列表、元组、字典来构成的。所以遇到负责问题可以转化为python基础对象来解决。在编程中往往需要灵活运动基础数据结构。因此基础数据对象的方法也需要深入了解。

**1.1 pandas新建数据对象：**

Zip（） 可以把多个可迭代对象按照最短长度压制成一列元组。

不等长多个可迭代对象按照最长长度

>>> a = ['a1']

>>> b = ['b1', 'b2', 'b3']

>>> c = ['c1', 'c2']

list(itertools.zip\_longest(a, b, c))

[('a1', 'b1', 'c1'), (None, 'b2', 'c2'), (None, 'b3', None)]、

list(itertools.zip\_longest(a, b, c, fillvalue='foo'))

[('a1', 'b1', 'c1'), ('foo', 'b2', 'c2'), ('foo', 'b3', 'foo')]

enumerate() 把一个可迭代对象的index和数据压制成一列元组。

from pandas import \*

Series(list或者dict)

基本类函数是DataFrame():

参数是列表，列表里每个元素是一行，元素可以是嵌套列表。如果每个元素是字典元素，字典的key代表列名。

参数是数组array，同上，array可以用reshape()方法塑型。也可以用to\_list()方法变成嵌套列表。

对于数组或者嵌套列表可以用index=list('ABCD'),columns=list('ABCD')参数添加行索引和列名，

**多重（复合）索引则用嵌套列表，长度对齐。**

**data = pd.Series(np.random.randn(10),**

**index=[['a', 'a', 'a', 'b', 'b', 'b', 'c', 'c', 'd', 'd'],**

**[1, 2, 3, 1, 2, 3, 1, 2, 2, 3]])**

**或者pd.MultiIndex.from\_product生成笛卡尔积，names参数设置索引层级名称，注意：对列索引（列名）同样适用。如果上一种方法用嵌套列表也要用层级名称可以用pd.MultiIndex.from\_arrays(嵌套列表，names = [层级的名称]）**

df2 = pd.DataFrame(np.random.randint(80, 120, size=(4, 2)),

columns= ['girl', 'boy'],

index=pd.MultiIndex.from\_product([['English','Chinese'], ['like','dislike']],names=['class','like']))

类方法DataFrame.from\_dict() : 参数：{‘列名’:[数值列表1],‘列名’:[数值列表2]...}

参数是字典，字典的key是列名，字典的value是每列的数据。由此可以理解df['列名']选择一个series。

逆向操作 to\_dict

df.to\_dict(orient='dict')

{'colA': {0: 'A', 1: 'A', 2: 'B', 3: 'C', 4: 'A'},

……

'colD': {0: 90, 1: 60, 2: 60, 3: 80, 4: 50}}

df.to\_dict(orient='list')

{'colA': ['A', 'A', 'B', 'C', 'A'],

……

'colD': [90, 60, 60, 80, 50]}

df.to\_dict(orient='series')

Out[7]:

{'colA':

0 A

1 A

2 B

3 C

4 A

Name: colA, dtype: object,

……

'colD':

0 90

1 60

2 60

3 80

4 50

Name: colD, dtype: int64}

df.to\_dict(orient='split')

Out[8]:

{'columns': ['colA', 'colB', 'colC', 'colD'],

'data': [['A', 'X', 100, 90],

……

['A', 'Xa', 20, 50]],

'index': [0, 1, 2, 3, 4]}

In [9]: df.to\_dict(orient='records')

Out[9]:

[{'colA': 'A', 'colB': 'X', 'colC': 100, 'colD': 90},

……

{'colA': 'A', 'colB': 'Xa', 'colC': 20, 'colD': 50}]

In [10]: df.to\_dict(orient='index')

Out[10]:

{0: {'colA': 'A', 'colB': 'X', 'colC': 100, 'colD': 90},

……

4: {'colA': 'A', 'colB': 'Xa', 'colC': 20, 'colD': 50}}

DataFrame.from\_items():参数：[ (列名，[数值1，数值2,...]) ,(列名，[数值1，数值2,...]),... ]，

参数是字典的items,即一个元组组成的列表，生成数据框。用的较少。不如直接转化成字典。

DataFrame.from\_records()参数是列表元组，适合数据库查询结果。list(df.to\_records(index=False))可以把df转化成导入数据库的元组列表形式。适合excutmany操作。



**1.2从文件或者模块获取**

1.2.1 read\_table（sep=’,’,enconding=’utf-8’）、read\_csv(sep=’,’,enconding=’utf-8’)、read\_excel参数基本一致

pd.read\_excel（io#如果从windows直接复制地址。小心盘符前由一个隐藏字符要删掉。，sheet\_name = 0，header = int, list of int, default 0 ；如果是列表则是复合列名，

names = None或者在没有指定header的时候赋予的的列名列表，

index\_col = int, list of int, default None，如果使用usecols选择数据的子集，则索引列基于该子集。

usecols = int或list（支持切片元素,切片是切excel的列名），默认为None，

squeeze = 默认为False,如果解析的数据只包含一列，则返回一个Series。

,dtype = None或者{'a'：np.float64，'b'：np.int32}, #对于列日期转换最好用pd.to\_datetime处理

skiprows=None, nrows=None, na\_values=None ...）

在读取不规范的合并单元格时，会出现空值。可以使用df.fillna(method = "ffill")或者”bfill”的方式来前填充和后填充。



parse\_dates=[0],如上图所示，指定第一列作为日期解析列。但是如果第一列是中文的日期格式，比如2021年9月1日，那么还需要传入

date\_parser = lambda x : pd.to\_datetime(x,format=’%Y年%m月%d日’)作为解析函数

csv也一样

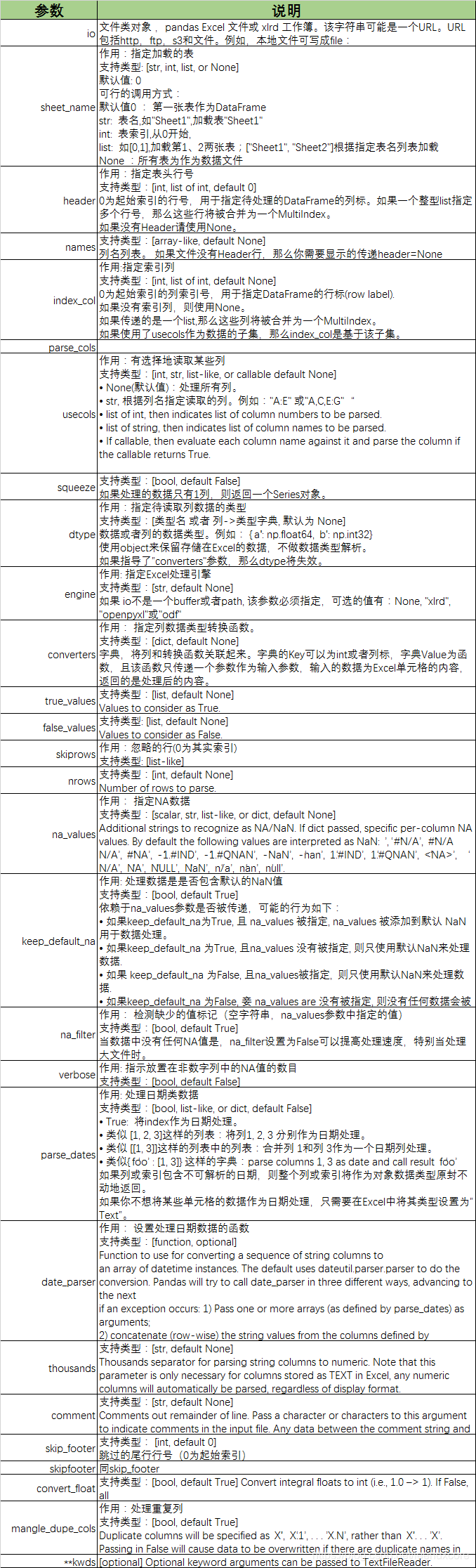
def parse(y,m,d,h):

return dt.datetime.strptime(' '.join([y,m,d,h]), '%Y %m %d %H')

dataset = pd.read\_csv(r'.\mypyworks\StatLedger\数据表\raw.csv',

parse\_dates = [['year', 'month', 'day', 'hour']], index\_col=0, date\_parser=parse)

Parse\_dates具体使用见下面的参数说明。



**1.2.1.1 Excel 自由读写**

Excel和Python之间数据互动工具比较多，有的工具是单向通道，有的工具是双向通道。以下罗列了一些工具，供大家参考：

Xlwings： 支持双向通道，支持和当前活动Excel数据互动，如果你Python是安装的anaconda版本的，默认安装了该分析包，后面重点介绍这个工具的用法；

PyXLL： 支持双向通道，能力比xlwings功能强大，可惜是商业工具，每个月几十美刀服务费，忍痛可以试试，而且该工具目前已经实现了在Excel内集成了jupyter notebook功能，有兴趣，可以去了解；

Xlrd： 从Excel之中读取数据，anaconda默认已经安装，不需要额外安装，不支持和当前Excel互动；

Xlwt： 向Excel之中写入数据，anaconda默认已经安装，也不需要再次安装，也不支持和当前Excel互动

Xlsxwriter： 也是默认安装包，功能比xlrd和xlwt强大，但不支持数据互动。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 能力 | xlwings | PyXLL | xlrd | xlwt | xlsxwriter |
| 读取当前excel | Y | Y | N | N | N |
| 调用VBA宏 | Y | Y | N | N | N |
| 写入数据 | Y | Y | N | Y | Y |
| 读取数据 | Y | Y | Y | N | Y |
| 操作Excel对象 | Y | Y | N | N | Y |

先要引入xlwings包，完整代码如下，数据获取逻辑为，先获得当前活动工作簿。获取工作簿后，也可以获取当前活动表。下案例直接通过工作表名称获取数据。

import xlwings as xw

import pandas as pd

wb = xw.books.active

sheet = xw.sheets.active

ranges = xw.sheets('iris数据集').range('$A$1:$E$151') #直接取活动的excel文件的某个表

data=ranges.value

pfD = pd.DataFrame(data[1:],columns=data[0])

成功获取数据后，可以将数据转为pandas DataFrame对象，后续就可以进行数据分析。

我们使用Python完成数据分析后，想将数据回传到Excel某个表之中，该怎么操作呢？例如pfD DataFrame数据，我们想回传到Excel Sheet5表格之中

import xlwings as xw

import pandas as pd

import numpy as np

wb = xw.books.active

sheet = xw.sheets('Sheet5')

if 'pfD' in locals().keys():

if isinstance(pfD, pd.DataFrame): # 对结果数据进行存储

sheet.range('$A$1').options(index=False).value = pfD

else:# 对变量进行处理

if isinstance(pfD, pd.Series):

sheet.range('$A$1').options(index=False).value = pfD

else:

try:

pfD = pd.DataFrame(pfD) # 数据转化

sheet.range('$A$1').options(index=False).value = pfD

except BaseException as Argument:

print( Argument)

else:

print('数据未初始化')

以下总结了xlwings和Excel工作簿互动使用频率较高的语法，详细内容，大家可以参见官网教程。在python基础文件夹中有教程pdf。

|  |  |
| --- | --- |
| 语法 | 作用 |
| wb = xw.Book() | 新建一个工作簿，同时启动Excel，xlwings会和该工作簿保持联动 |
| wb = xw.Book('FileName.xlsx') | 打开Excel当前进程某个工作簿，xlwings会和该工作簿保持联动 |
| wb = xw.Book(r'C:\path\to\file.xlsx') | 打开硬盘上某个工作簿，并建立互动联系 |
| wb = xw.books.active | 和Excel当前活动工作簿建立联系 |
| sheet = wb.sheets['sheetName'] | 通过名称获取sheet表格 |
| sheet = wb.sheets[index] | 通过序号，获取sheet表格 |
| range = sheet.range('$A$1:$E$151') | 获取单元格区域 |
| value = range.value | 获取单元格选区内的数据，数据可能为一个值，也可能为数组 |
| range.value=data | 给Excel内选区赋值，相当于将数据传到Excel之中 |
| sheet.pictures.add(fig, name='MyPlot', update=True) | 将Python绘制的图片，传到Excel之中 |
| …… | 还有其他很多命令，大家参见官方教程 |

**import xlrd xlwt 缺点不支持写入xlsx**

只读数据

rb = xlrd.open\_workbook('test\_cases\\read\_xlsx.xlsx')

xlrd.open\_workbook(r'原文件的路径+文件名') 打开原工作簿；formatting\_info = True表示保留工作簿中的格式。

获取工作表对象

①  sheet = wb.sheets()[0]      或者 wb.sheet\_names()[1] #通过索引顺序获取  
②  sheet = wb.sheet\_by\_index(0) #通过索引顺序获取  
③  sheet = wb.sheet\_by\_name(u'Sheet1')#通过名称获取

④ sheet = wb.get\_sheet(sheet编号)引用到目标sheet；

获取所有的合并单元单元格

sheet2.merged\_cells formatting\_info参数设置为True才有用

输出[(7, 8, 2, 5), (1, 3, 4, 5), (3, 6, 4, 5)]

merged\_cells返回的这四个参数的含义是：(row,row\_range,col,col\_range),其中[row,row\_range)包括row,不包括row\_range,col也是一样，即(1, 3, 4, 5)的含义是：第1到2行（不包括3）合并，(7, 8, 2, 5)的含义是：第2到4列合并。

sheet2.cell\_value(1,4) #(1, 3, 4, 5) 获取合并单元格值

获取整行和整列的值（返回数组）

sheet.row\_values(i)

sheet.col\_values(i)

获取sheet 名、行数和列数

sheet.name

sheet.nrows

sheet.ncols

获取单元格数据

sheet.cell(0,0).value

sheet.cell\_value(1,0)

sheet.row(1)[0].value

获取单元格数据类型

sheet2.cell(1,0).ctype

ctype : 0 empty,1 string, 2 number, 3 date, 4 boolean, 5 error

显示正确的日期数据

datemode 0: 1900-based, 1: 1904-based

datemode 有2个选项基本我们都会使用1900为基础的时间戳

①thedatetime = xlrd.xldate\_as\_datetime(sheet.cell(4,6).value, datemode=0)

quota\_date = thedatetime.strftime('%Y-%m-%d %H:%M:%S')

②date\_value = xlrd.xldate\_as\_tuple(sheet.cell\_value(2,2),wb.datemode)

#(1992, 2, 22, 0, 0, 0)

date(\*date\_value[:3])

#datetime.date(1992, 2, 22)

date(\*date\_value[:3]).strftime('%Y/%m/%d')

#'1992/02/22'

**已有文件修改**

**import xlwt**

from xlutils.copy import copy

rb = xlrd.open\_workbook('excel.xls')

wb = copy(rb) 不改动原文件

修改方式和保存方式和下面新建文件一样

wb.write(‘filename.xls’)

新建文件

book = xlwt.Workbook(encoding = 'ascii')可以不指定编码。建议用ascii编码（编码知识在下一节详细写），不然可能会有一些诡异的现象。

sheet = book.add\_sheet('My Worksheet',cell\_overwrite\_ok=True)

sheet.write(i, j, 65536)写入单格

sheet1.write\_merge(i,i+3,j,j+3,u’中文’,  sytle =set\_style('Arial',220,True))写入合并格子

设置格式技巧

def set\_style(name,height,bold=False):

style = xlwt.XFStyle() # 初始化样式

font = xlwt.Font() # 为样式创建字体

font.name = name # 'Times New Roman' excel字体

font.bold = bold

font.color\_index = 4

font.height = height

# borders= xlwt.Borders()

# borders.left= 6

# borders.right= 6

# borders.top= 6

# borders.bottom= 6

style.font = font

# style.borders = borders

return style

添加超链接

n= "HYPERLINK"

sheet2.write\_merge(9,9,2,8, xlwt.Formula(n +'("http://www.cnblogs.com/zhoujie";"jzhou\'s blog")'),set\_style('Arial',300,True))

sheet2.write\_merge(10,10, 2, 8, xlwt.Formula(n +'("mailto:zhoujie0111@126.com";"contact me")'),set\_style('Arial',300,True))

Excel表的保存

book.save('test\_cases\\write\_xls.xls')

**Import openpyxls 不支持xls 开启read\_only和write\_only模式后对其性能有大幅提升**

**只读数据**

wb= openpyxl.load\_workbook('test\_cases\\read\_xlsx.xlsx', read\_only=True)

wb.get\_sheet\_names()或者wb.sheetnames 查看并获得Excel文件中sheet的名字,返回列表

wb.worksheets返回列表 元素为工作表。

sheet = wb.worksheets[idx]或者mysheet = wb.get\_sheet\_by\_name('Sheet')通过sheet名字来获得该sheet工作表对象

获得工作表的名称 sheet.title

mysheet['F6'].value 获得单元格的值

**修改已有表**

wb= openpyxl.load\_workbook('test\_cases\\read\_xlsx.xlsx')

wb = copy(wb)

全新文件

wb= openpyxl.Workbook(write\_only=True) 创建了一个Excel表格，里面只有一个sheet 工作表

共同的方法

删除工作表wb.remove（sheet）

wb.create\_sheet(title='my\_second\_sheet',index=2) //index表示这是第几张工作表，从零开始；，title是工作表的名字

sheet=mywb.active 获取初始的sheet

mywb.get\_sheet\_names() 查看并获得Excel文件中sheet的名字,返回列表

mysheet = mywb.get\_sheet\_by\_name('Sheet')通过sheet名字来获得该sheet工作表对象

mysheet['F6'] = 'Writing new Value!' 操作单元格

mywb.save('要保存的路径.xlsx')

1.2.2 read\_sql 可以从数据库读取数据表

conn = cx\_Oracle.connect(user=user,password=password,dsn=dsn)

pd.read\_sql(sql,self.conn)

1.2.3从sklearn的datasets模块导入数据集：from sklearn import datasets ;datasets.load\_iris()

手写数字数据集load\_digits()：用于多分类任务的数据集

乳腺癌数据集load\_barest\_cancer（）：简单经典的用于二分类任务的数据集

糖尿病数据集：load\_diabetes（）：经典的用于回归认为的数据集，值得注意的是，这10个特征中的每个特征都已经被处理成0均值，方差归一化的特征值

波士顿房价数据集：load\_boston（）：经典的用于回归任务的数据集

体能训练数据集：load\_linnerud（）：经典的用于多变量回归任务的数据集，其内部包含两个小数据集：Excise是对3个训练变量的20次观测（体重，腰围，脉搏），physiological是对3个生理学变量的20次观测（引体向上，仰卧起坐，立定跳远）

**编码知识：encode是把一个字符按照指定的编码对应关系转换成序号二进制字节。decode是把二进制字节保存序号翻译成编码规则对应的字符。**

**ASCII编码：**因为计算机只能处理数字，如果要处理文本，就必须先把文本转换为数字才能处理。最早的计算机在设计时采用8个比特（bit）作为一个字节（byte），一个bit只能表示0，1,所以，一个字节能表示的最大的整数就是255,如果要表示更大的整数，就必须用更多的字节。比如两个字节可以表示的最大整数是255^2 =65535，4个字节可以表示的最大整数是4294967295。由于计算机是美国人发明的，因此，最早只有127个字符被编码到计算机里，也就是大小写英文字母、数字和一些符号，这个编码表被称为ASCII编码，比如大写字母A的编码是65，小写字母z的编码是122。

**GBK、GB2312（中英文编码）：**但是要处理中文显然一个字节是不够的，至少需要两个字节，所以，中国制定了GB2312编码，用来把中文编进去，并且编码序号与 ASCII编码已经有的兼容。

**Unicode字符集编码：**全世界有上百种语言，日本把日文编到Shift\_JIS里，韩国把韩文编到Euc-kr里，各国有各国的标准，编码冲突互相不认。Unicode字符集则把所有语言都统一到一套编码序号里并兼容ascII，这样就不会再有乱码问题了，它一般由两位，最多由四个字节来表示文字符号。比如“中”在unicode编码中序号是20013（口头说十进制），转成二进制存储是01001110 00101101（计算机认二进制）。不足的两个字节16位在前面补0. Python提供了ord()函数获取字符的整数表示，chr()函数把编码转换为对应的字符：ord(‘A’)返回Unicode码二进制对应的十进制编号, chr(66)返回Unicode码对应的字母B。

**UTF-8编码：**如果你基本上全部是英文的话，用Unicode编码2字节比ASCII编码1字节需要多一倍的存储空间，在存储和传输上就十分不划算。所以，本着节约的精神，又出现了把Unicode编码转化为“可变长编码”的UTF-8编码。UTF-8编码把一个Unicode字符根据不同的数字大小编码成1-6个字节，常用的英文字母被编码成1个字节，汉字通常是3个字节，只有很生僻的字符才会被编码成4-6个字节。如果你要传输的文本包含大量英文字符，用UTF-8编码就能节省空间。Utf-8与unicode的序号已经不一致了。

计算机一般情况下转换过程：在计算机内存中，统一使用Unicode编码，当需要保存到硬盘或者需要传输的时候，就转换为UTF-8编码。用记事本编辑的时候，从硬盘文件读取的UTF-8字符被转换为Unicode字符到内存里，编辑完成后，保存的时候再把Unicode转换为UTF-8保存到文件：浏览网页的时候，服务器会把动态生成的Unicode内容转换为UTF-8再传输到浏览器：

在最新的Python 3版本中，字符串是以Unicode编码的，也就是说，Python的字符串支持多语言。可以直接认16进制表示unicode编码序号的中文。

>>> '\u4e2d\u6587' #注意\u后面只是表示序号。并非表示字节。

'中文'

以Unicode表示的str通过encode()方法可以转化为指定编码字节，例如：

>>> 'ABC'.encode('ascii')

b'ABC' #此处不显示ascII编码，

>>> '中文'.encode('utf-8')

b'\xe4\xb8\xad\xe6\x96\x87'#显示utf-8编码。一个/x表示一个字节，一个中文三字节。

>>> '中文'.encode('ascii')

报错。AscII没有中文编码。

如果我们从网络或磁盘上读取了字节流，那么读到的数据就是bytes。要把bytes变为str，就需要用decode()方法：一下只是显性演示。实际中电脑后台进行。

>>> b'ABC'.decode('ascii')

'ABC'

>>> b'\xe4\xb8\xad\xe6\x96\x87'.decode('utf-8')

'中文'

如果bytes中包含无法解码的字节，decode()方法会报错：比如GBK编码是无法用utf-8解码

>>> b'\xe4\xb8\xad\xff'.decode('utf-8')

Traceback (most recent call last): ...

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xff in position 3: invalid start byte

如果bytes中只有一小部分无效的字节，可以传入errors='ignore'忽略错误的字节：能认多少翻译多少。

由于Python源代码也是一个文本文件，所以，当你的源代码中包含中文的时候，在保存源代码时，就需要务必指定保存为UTF-8编码。当Python解释器读取源代码时，为了让它按UTF-8编码读取，我们通常在文件开头写上这两行：

由于Python源代码也是一个文本文件，所以，当你的源代码中包含中文的时候，在保存源代码时，就需要务必指定保存为UTF-8编码。当Python解释器读取源代码时，为了让它按UTF-8编码读取，我们通常在文件开头写上这两行：

#!/usr/bin/env python3 用python解释编译代码

# -\*- coding: utf-8 -\*- 用utf-8 读取解码字符

申明了UTF-8编码并不意味着你的.py文件就是UTF-8编码的，必须并且要确保文本编辑器正在使用UTF-8 without BOM编码：spyder 默认

**1.3自行有规律的生成数据或者有意指定数据**

1.3.1 除了python基础的生成对象赋值外。还可以多变量赋值：元素的赋值有并行赋值 x，y，z = 1，2，3内存有三个值；链式赋值 x=y=1 内存里只有一个值；避免x=x+1这样的啰嗦写法可用增强赋值 x+=1 ,x\*=1；**条件赋值**用三元表达式a = x if x>y else y

1.3.2列表推导式 公式推导+条件推导[x\*x for x in range(10) if x%3 == 0] [(x,y) for x in range(3) for y in range(3) ] range包下不包上，只能是整数类型

array(list)

filter()函数用于过滤序列，过滤掉不符合条件的元素，返回由符合条件元素组成的新列表。其中接收两个参数，第一个为函数，第二个为序列，序列的每个元素作为参数传递给函数进行判断，然后返回 True 或 False，最后将返回 True 的元素放到新列表中。

用法：filter(function, iterable)

numbers = [1,3,4,6,7,8,10,12]

check\_even = lambda x:x%2==0

even\_numbers = list(filter(check\_even,numbers))

print(even\_numbers)

------------------------------------------------

[4, 6, 8, 10, 12]

**列表合并**

1.可以使用"+"号完成操作

2.使用extend方法

3.使用切片

List1[0:0]=list2#把list2放在开头

List1[len(list1):len(list1)]#把list2放在结尾

**创建元组：**

赋值法创建：

d={}

d['n']='name'

元组推导式：

students = ['x', 'y', 'z', 'a', 'b', 'c', 'g', 'k', 'j']

Dict = {s: randint(60, 100) for s in students}

resultdict = {i:diffquotadict.get(i,0) for i in quotalist}

从元组列表创建：

Items = [(‘name’,’bob’),(‘age’,’42’)]

d = dict(items) 与.items()方法互逆

.fromkeys的方法创建：

a = dict.fromkeys(['name','age'],’unkown’)默认value为none，也可以在第二个参数中指定

字典显示值的常用方法除了dict[key],来实现以外还有.get(‘name’ , ’N/A’),即使不存在也不会报错，此方法对dataframe同样适用。

还有显示值和赋值合二为一的方法.setdefault(‘’name,’N/A’), 值存在显示值，不存在更新值，并显示更新值。比直接赋值有个好处是不会刷新已有数据，

合并两个字典：dict(dic1,\*\*dic2) ；dict1.update(dict2)

提取字典的keys/values

Dict.keys()/values()

1.3.3.1 利用datetime、 dateutil构建基础时间时间数据

**通过阳历获取查询农历日期信息**

import sxtwl

lunar = sxtwl.Lunar()

solar\_day = lunar.getDayBySolar(2018, 10, 20) # 查询2018年10月20日

# 日历中文索引

ymc = [u"十一", u"十二", u"正", u"二", u"三", u"四", u"五", u"六", u"七", u"八", u"九", u"十"]

rmc = [u"初一", u"初二", u"初三", u"初四", u"初五", u"初六", u"初七", u"初八", u"初九", u"初十", u"十一", u"十二", u"十三", u"十四", u"十五", u"十六", u"十七", u"十八", u"十九", u"二十", u"廿一", u"廿二", u"廿三", u"廿四", u"廿五", u"廿六", u"廿七", u"廿八", u"廿九", u"三十", u"卅一"]

if day.Lleap:

print(u"阴历:润", ymc[day.Lmc], u"月", rmc[day.Ldi], u"日")

else:

print(u"阴历:", ymc[day.Lmc], u"月", rmc[day.Ldi], u"日")

**通过阴历获取查询日期信息**

solar\_day = lunar.getDayByLunar(2018, 10, 20 , False) #查询阴历2018年10月20日的信息，最后一个False表示是否是润

solar\_date = dt.date(solar\_day.y, solar\_day.m, solar\_day.d)

dt.datetime.strftime/strptime

datetime.date：表示日期的类。常用的属性有year, month, day；date.today()

datetime.time：表示时间的类。常用的属性有hour, minute, second, microsecond；

datetime.datetime：表示日期时间。上面综合datetime.now()

datetime.timedelta：表示时间间隔，即两个时间点之间的长度。

D+-dt.timedelta(days=2, hours=6)可以计算日期移动,不能按照月、年为单位移动

类属性和方法：

datetime.year、month、day、hour、minute、second、microsecond、tzinfo：

datetime.date()：获取date对象；

datetime.time()：获取time对象；

datetime. replace ([ year[ , month[ , day[ , hour[ , minute[ , second[ , microsecond[ , tzinfo] ] ] ] ] ] ] ])：生成一个新的日期对象，用参数指定的年，月，日代替原有对象中的属性。（原有对象仍保持不变）

datetime. timetuple ()

datetime. utctimetuple ()

datetime. weekday ()返回weekday，如果是星期一，返回0；如果是星期2，返回1，以此类推；

datetime. isocalendar ()返回格式如(year，month，day)的元组；

data.isoweekday()：返回weekday，如果是星期一，返回1；如果是星期2，返回2，以此类推；

datetime. isoformat ([ sep] )返回格式如'YYYY-MM-DD’的字符串；

datetime. strftime (format)

在数据框中可以用.dt.strfime()来使用单个时间类型的属性。

1.3.3.2需要执行更加复杂的日期操作，比如月、年移动、处理时区，模糊时间范围，节假日计算等等， 可以考虑使用 dateutil模块

from dateutil.relativedelta import relativedelta

print(f+relativedelta(months=+1)) # 2019-02-23 00:00:00

cc=relativedelta(bb,aa)#计算两个日期的差

date+relativedelta(weekday=FR)#下一个周五

date+relativedelta(weekday=FR(-1))#上一个周五

relativedelta(arg1=x,arg2=y,arg3=z...)参与依次year, month, day, hour, minute, second, microsecond:单数参数是替换相当于replace。

years, months, weeks, days, hours, minutes, seconds, microseconds:复数单位是参与计算时间差。

常用格式代码

%a 星期的简写。如 星期三为Web

%A 星期的全写。如 星期三为Wednesday

%b 月份的简写。如4月份为Apr

%B月份的全写。如4月份为April

%c: 日期时间的字符串表示。（如： 04/07/10 10:43:39）

%d: 日在这个月中的天数（是这个月的第几天）

%f: 微秒（范围[0,999999]）

%H: 小时（24小时制，[0, 23]）

%I: 小时（12小时制，[0, 11]）

%j: 日在年中的天数 [001,366]（是当年的第几天）

%m: 月份（[01,12]）

%M: 分钟（[00,59]）

%p: AM或者PM

%S: 秒（范围为[00,61]，为什么不是[00, 59]，参考python手册~\_~）

%U: 周在当年的周数当年的第几周），星期天作为周的第一天

%w: 今天在这周的天数，范围为[0, 6]，6表示星期天

%W: 周在当年的周数（是当年的第几周），星期一作为周的第一天

%x: 日期字符串（如：04/07/10）

%X: 时间字符串（如：10:43:39）

%y: 2个数字表示的年份

%Y: 4个数字表示的年份

%z: 与utc时间的间隔 （如果是本地时间，返回空字符串）

%Z: 时区名称（如果是本地时间，返回空字符串）

%%: %% => %

**如果需要日期前不带0的格式有两种方法：**

1. 替换法：mydatetime.strftime('%m/%d/%Y %I:%M%p').lstrip("0").replace(" 0", " ")批量可以转成series用str 属性解决
2. 利用datetime格式不含0解决，如果个别需要添加0de ,添加“：”定义格式（红色）

'{d.month:02}/{d.day}/{d.year}'.format(d=datetime.datetime.now())

1.3.3.2利用pandas时间序列函数来生成datetime

pandas.date\_range(start=None, end=None, periods=None, freq=字符含义见下5小时:“5H”'2h20min', tz=None, normalize=False, name=None, closed=None, \*\*kwargs)

我们传入的是一个带有时间戳的日期 在生成日期序列之前，将start/end标准化为午夜。，可以传入 normalize Ture选项

pd.date\_range('5/2/2012 12:56:31', periods=5, normalize=True)

DatetimeIndex(['2012-05-02', '2012-05-03', '2012-05-04', '2012-05-05',

'2012-05-06'],

dtype='datetime64[ns]', freq='D')

closed：左右日期区间是否闭合可取。‘left’：左闭右开，‘right’：左开右闭。默认：None， 左右都是闭区间。

name : string, default None 生成的DatetimeIndex的名称

Datetime 没有获得季节的字符。可以运用(int(datetime.date(2014,8,15).strftime(“%m”))-1)//3+1 来获得季节。

//表示取商的整数部分 %表示取余数

1.3.4利用np生成：

生成固定范围固定步长的整数和小数

np.arange(0 ,1 , 0.1)

随机抽样列表或者数组元素

np.random.choice(a, size=None, replace=True, p=None)一个一个的的抽取

a是数组或者列表。

Size 抽多少次，replace是否放回来被在抽到，也就是结果中有无重复数据

P是对应a数组元素的数量的一组概率，是数组每个值的被抽取出来的概率

np.random.seed(0)

p = np.array([0.1, 0.0, 0.7, 0.2])

index = np.random.choice([0, 1, 2, 3], p = p.ravel()) .ravel是把多维数组变成一维数组。

(或者使用random模块

import random

slicelist = random.sample(list,10))

生成两个数据之间间距均匀的数据

np.linspace(prices.min(), prices.max(), 7) 默认endpoint =True 所以上下都包，改成false则不包上。

np.arange(0 ,1 , 0.1) 包下不包上

随机整数分布

np.random.randint(low，hight，n)生成随机整数,np.random开头函数取值范围左闭右开。

任意随机数分布

np.random.uniform(n1,n2,size) 产生size个随机数，服从平均分布[n1,n2]

np.random.rand(size) 通过本函数可以返回一个或一组服从“0~1”均匀分布的随机样本值。随机样本取值范围是[0,1)，不包括1。范围不可调，一个参数是一维数组，两个数字参数则是二维数组，依次下去。

二项式分布np.random.binomial(n,p,size) n 抛硬币次数 p正面朝上概率 size 重复实验多少次

正态分布

np.random.normal(μ,σ,size)-->随机生成一组服从正态分布的随机数，期望为μ，标准差为σ。

np.random.randn(size) 所谓标准正态分布（μ=0, σ=1），对应于np.random.normal(loc=0, scale=1, size)，多个参数可以生成服从分布的多维数组。

对于正态分布在-1.96～+1.96范围内曲线下的面积等于0.9500（即取值在这个范围的概率为95%），在-2.58～+2.58范围内曲线下面积为0.9900（即取值在这个范围的概率为99%）。

超几何分布

np.random.hypergeometric(7, 3, 3, 10),7个好的3个坏的，摸3个，重复10次，返回好球的个数组成的数组

利用np生成重复数组

np.repeat() np.repeat([1,2,3],2)每个元素重复多次，二维数组通用

np.tile() 所有元素整体重复多次，二维数组通用。

round 只能对一个数据保留小数位置

np.around 数组保留两位小数

dataframe.round() pandas保留两位

**生成笛卡尔数据**

Import itertools

Itertools.product([1,2,3],[100,200])

for item in Itertools.product([1,2,3],[100,200]):

Print item

输出结果：

（1，100）

（1，200）

...

**集合计算**

s = {1,2,3,4,5}

s2 = {3,4,5,6,7}

# & 交集运算

result = s & s2 # {3, 4, 5}

s1.intersection(s2)

# | 并集运算

result = s | s2 # {1,2,3,4,5,6,7}

s1.union(s2)

# - 差集

result = s - s2 # {1, 2}

s1.difference(s2)

# ^ 异或集或者叫交叉并集 获取只在一个集合中出现的元素

result = s ^ s2 # {1, 2, 6, 7}

s1.symmetric\_difference(s2)

# <= 检查一个集合是否是另一个集合的子集

# 如果a集合中的元素全部都在b集合中出现，那么a集合就是b集合的子集，b集合是a集合超集

a = {1,2,3}

b = {1,2,3,4,5}

result = a <= b # True

a.issubset(b)

result = {1,2,3} <= {1,2,3} # True

result = {1,2,3,4,5} <= {1,2,3} # False

# < 检查一个集合是否是另一个集合的真子集

# 如果超集b中含有子集a中所有元素，并且b中还有a中没有的元素，则b就是a的真超集，a是b的真子集

result = {1,2,3} < {1,2,3} # False

result = {1,2,3} < {1,2,3,4,5} # True

# >= 检查一个集合是否是另一个的超集

# > 检查一个集合是否是另一个的真超集

s3.issuperset(s1)

合并集合

set1.update(set2)

字典合并，在原来字典上合并

dic1.update（dic2）

合并方式会产生新的字典对象

dict(dic1,\*\*dic2)

集合变成列表

sorted(set(zhibiao\_list),key=zhibiao\_list.index)

**1.4数据的保存**

1.4.1保存txt 、csv、excel 参数差不多

DataFrame.to\_excel(excel\_writer, sheet\_name='Sheet1', na\_rep='',

float\_format=None, columns=None, header=True, index=True,

index\_label=None, startrow=0, startcol=0, engine=None,

merge\_cells=True, encoding=None, inf\_rep='inf', verbose=True,

freeze\_panes=None)

可以建立操作对象 excel\_writer= pd.ExcelWriter(file\_abs, engine='openpyxl')

writer.save()

writer.close()

不需要保存，关闭对象的方式

with pd.ExcelWriter('E:/每日快报营运周报/广州自来水每日快报模板.xlsx') as writer:

shuju\_df.to\_excel(writer, sheet\_name='Sheet\_name\_1')

保存进入数据库,利用元组格式：

cursor.executemany("INSERT INTO TJ\_DATA\_"+"tmp3"+" \

VALUES(:1,:2,:3,:4,:5,:6,:7,:8,:9)",mylist)

1.4.3保存pkl文件

1.4.3.1 保存列表 、字典 、元组 、对象等python对象到硬盘用

import pickle

with open('staytim\_li.pkl', 'wb') as f: #不用f.close()

pickle.dump(staytim\_li, f)

with open("./1.pkl", 'rb') as f:

# 反序列化解析成列表a

a = pickle.load(f)

1.4.3.2保存模型

from sklearn.externals import joblib

joblib.dump(my\_model, "my\_model.pkl") # DIFF

#and later...

my\_model\_loaded = joblib.load("my\_model.pkl") # DIFF

**二、数据探索**

数据探索是指原数据不修改，只是观察数据的结构和指标。

**2.1 查看数据信息**

2.1.1 方法：

列表排序：list.sort()

数组列表通用排序：

sorted([5, 2, 3, 1, 4])

数组 a[a.argsort()]排序

a[np.argsort(x)] #升序

a[np.argsort(-x)]#降序

max(list)

min,mean(list)…

series.count() #非空元素计算

series.value\_counts(dropna=False) #各类数值计数，默认不含空值，可以指定dropna参数。指定ascending=True按照正序排列，normalize=True，计算各类比例。bins = integer对于数值数据，可以用这个指定切分的的数量。并统计每个数量的多少。

Series.unique()#唯一值 只对series ，df用.drop\_duplicates（）

Series.nunique()#唯一值的个数

series.min() #最小值

series.max() #最大值

series.idxmin() #最小值的位置，类似于R中的which.min函数

series.idxmax() #最大值的位置，类似于R中的which.max函数

series.quantile(0.1) #10%分位数

Df/series.sum() #求和 通过axis=1指定按行求和

series.mean() #均值

series.median() #中位数

series.mode() #众数

series.var() #方差

series.std() #标准差

series.mad() #平均绝对偏差

series.skew() #偏度

series.kurt() #峰度

cumsum()依次给出前1、2、… 、n个数的和

cumprod() 依次给出前1、2、… 、n个数的积

cummax() 依次给出前1、2、… 、n个数的最大值

cummin() 依次给出前1、2、… 、n个数的最小值

**DataFrame.duplicated(subset,keep)#显示关键索引下的重复行**

keep{‘first’,所有重复项目keep住第一个行，判断为false，其他判断为ture被选择出来

‘last’, 所有重复keep住最后一项目判断为false，其他判断为ture被选择出来

False 所有的重复值都变成ture被选择出来}, default ‘first’，这个比较常用。

df.info(),dtype查看数据框的索引和列的类型、费控设置和内存用量信息。

describe显示描述性统计数据，包括集中趋势、分散趋势、形状等。

head;tail查看前后N条结果

Isnull/notnull()元素级判断，查看每个值是否为空值返回布尔值。

meterdata[meterdata['表码'].isnull()]可以筛选某一列的空值。

df.isnull().any(axis=1)

df.isnull()判断df所有元素是否未空的布尔值矩阵。叠加.any(axis=1)任何元素有缺失的列则显示TRUE。默认是行。.all()则是行全部元素未空缺才显示为TURE。housing[housing.isnull().any(axis=1)].head()。.isnull()叠加.all(axis=1)的方法必须所有元素返回为True才能被选择。

注意：“空”在python原生数据中空为None，float('inf')表示正无穷，float('-inf')，表示负无穷。在pandas中空值为pd.NA,显示为NULL，pd.NaT表示非时间类型，在np中为np.NaN(nan)是not a number非数值型。numpy.inf 表示正无穷大,-np.inf表示负无穷大,numpy.pi 表示圆周率numpy.e 表示自然数

Pyton 所有数据类型



列表用map方法将空字段转换成0或者空值。

def FloatOrZero(value):

try:

return float(value)

except:

return 0.0

print map(FloatOrZero, ['3', '1', '', '0', '0', '8.4583', '0'])

数组用下面方法转换空值

s = np.array(['1', '0', ''])

s[s==''] = '0'

s.astype(float)

#array([ 1., 0., 0.])

sup\_water[sup\_water.isnull().any(axis=1)]

只判断NULL是不够的。 np有丰富的方法判断np中的元素的数据类型，同时可以包含判断null（np中理解为np.nan）:

np.isinf：显示哪些元素为正或负无穷大

np.isposinf：显示哪些元素为正无穷大

np.isneginf：显示哪些元素为负无穷大

np.isnan：显示哪些元素不是数字，包含无穷大。

np.isfinite：显示哪些元素是有限的（不含非数字，正无穷大和负无穷大中的一个）

以上为元素级别判断。可以用于dataframe。和.any().all()结合功能更强大。

sup\_water[~(np.isfinite(sup\_water).all(axis=1))]

sup\_water[~np.isfinite(sup\_water['最高温度'])==1] #判断某一列不是的值不是空，来行删除

~按位取反1100 0011，也可以用于逻辑取反，适用于array。

not 用于逻辑取反，并只适用于一个逻辑值

.unique查看特定列的唯一值，只对series有用

2.1.2 属性：

提取索引index

提取列标签Columns

提取行列数shape

提取df的值变成数组 以方便在输入模型或者np进行处理。Df.values

2.1.3 查看列表，数据列是否是否含有重复数据

列表可以用语句推导

A = list(1,2,3,3,4)

Set1 = {}

Dupli\_list = []

For i in a:

If i not in set1:

Set1.append(i)

Else:

Dupli\_list.append(i)

Set(dupli\_list)

转换为pd.series

a\_seri = pd.Series(a)

set(a\_seri[a\_seri.duplicated(keep=False)])

查看元素重复多少次

Series方法

a\_seri = pd.Series(a)

a\_seri.value\_counts()

列表方式

result = {}

for i in set(a):

result[i] = a.count(i)

**2.2 常规性统计数据分类汇总指标**

2.1.1 按照指定索引分类，指定多个汇总函数或者不同列不同函数

对于index是日期

df.set\_index('date').resample('M')['ext price'].sum()

df.set\_index('date').groupby('name')['ext price'].resample("M").sum()#resample与groupby联合

全部用groupby搞定日期。

df.groupby(['name', pd.Grouper(key='date', freq='M')])[['ext price']].sum()#注意双括号

df[["ext price", "quantity", "unit price"]].groupby([]).agg(['sum', 'mean'])

df.agg({'ext price': ['sum', 'mean'], 'quantity': ['sum', 'mean'], 'unit price': ['mean']})

统计唯一值的个数这这种pandas的函数可以通过pd.Series.nunique调用

meter\_2019\_2020.groupby('月份').agg({'用户编号': pd.Series.nunique,'水量':sum})

自造函数

get\_max = lambda x: x.value\_counts(dropna=False).index[0]

get\_max.\_\_name\_\_ = "most frequent"

import collections

agg\_dict = {

'ext price': ['sum', 'mean'],

'quantity': ['sum', 'mean'],

'unit price': ['mean'],

'sku': [get\_max]}

# 按照列名的长度排序。 OrderedDict的顺序是跟插入顺序一致的

df.agg(collections.OrderedDict(sorted(agg\_dict.items(), key = lambda x: len(x[0]))))

如果需要一个一个小df应用构造函数

lawsuit2[['EID','LAWAMOUNT','LAWDATE']].groupby(['EID']).apply(lambda df:print(df))

apply 是一个更一般化的方法，将当前分组后的数据一起传入，可以返回多个df。对分组后的单个df应用函数。不同于agg或者.sum()方法一列一列应用用函数

在groupby后使用apply，小df会出现有多余的groupby索引问题，可以使用group\_keys=False去掉groupby("reviewerID",group\_keys=False)

#长表转宽表

Pivot\_table在不指定展开列的情况下相当于goupby

pd.pivot\_table(df,index=["Manager","Status"],~~columns=["Product"],~~values=["Quantity","Price"],

aggfunc={"Quantity":len,"Price":[np.sum,np.mean]},fill\_value=0,margins=True)

aggfunc 等同于agg 不同的是前者参数必须是函数，后者可以是函数名或函数。aggfunc参数是单个的时候必须加方括号。aggfunc=[np.sum]

透视表依然支持query方法查询table.query('Manager == ["Debra Henley"]')

#宽表转长表

pandas.melt 使用参数：

pandas.melt(frame, id\_vars=None, value\_vars=None, var\_name=None, value\_name='value', col\_level=None,ignore\_index=True)

参数解释：

frame:要处理的数据集。

id\_vars:不需要被转换的列名。

value\_vars:需要转换的列名，如果剩下的列全部都要转换，就不用写了。

var\_name和value\_name是自定义设置对应的列名。

ignore\_index=True -- 是否忽略原始索引

col\_level :如果列是MultiIndex，则使用此级别。col\_level=1选择内层索引

# 多重索引

df.columns = [list('ABC'), list('DEF')]

df

'''

A B C

D E F

0 a 1 2

1 b 3 4

2 c 5 6

'''

# 选择复合索引

pd.melt(df, id\_vars=[('A', 'D')], value\_vars=[('B', 'E')])

'''

(A, D) variable\_0 variable\_1 value

0 a B E 1

1 b B E 3

2 c B E 5

'''

**2.1.2 对于时间的分类汇总**

四个时间处理函数

1.df.to\_period用法：data\_h.set\_index('QUOTA\_DATE').to\_period('y') 将df或者series的时间索引改变成制定的时间格式。数据不变。不统计。

2.df.asfreq(freq, method=None, how=None, normalize=False, fill\_value=None) 按照制定频率抽取一个数据，其他扔掉。不统计，

freq : DateOffset对象，或字符串

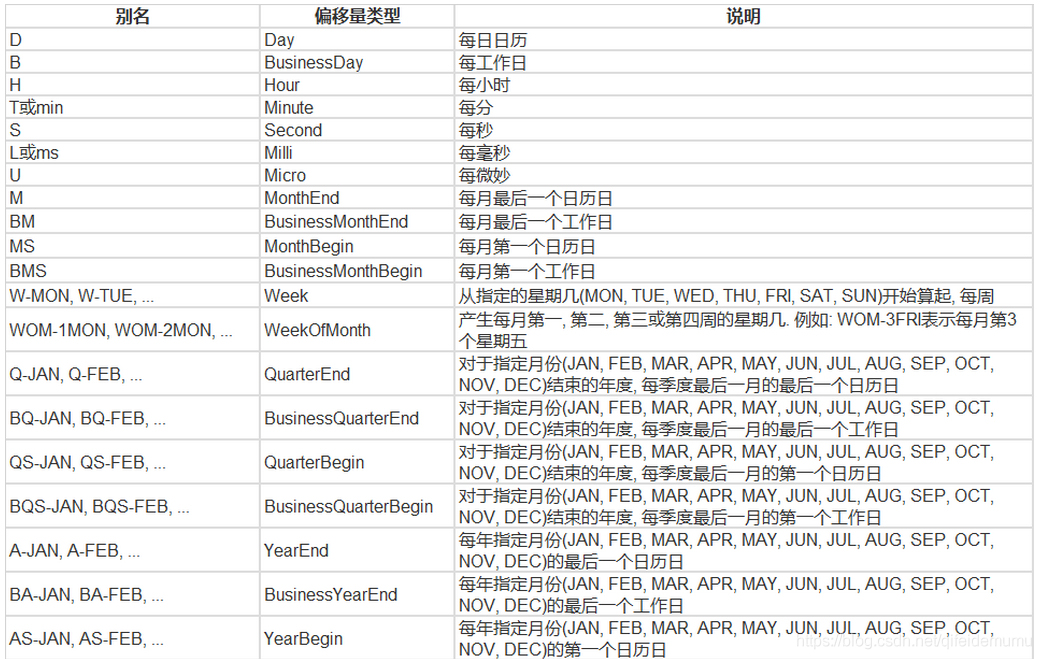
method:用于填补重新索引系列的漏洞的方法

how: 仅对于PeriodIndex，见PeriodIndex.asfreq

normalize : 是否将输出指数重置为午夜时分

fill\_value :用于缺失值的值，在上升采样过程中应用（注意这并不填补已经存在的NaN）。

3.groupby（pd.Grouper（））、4.resample显示并统计。（后面有详细用法）



“B” 间隔工作日，索引均为date格式

groupby 休息日属于前一工作日汇总

resample 同上

to\_period 休息日索引显示为后一工作日，显示全部数据

asfreq 删除休息日

“M” 间隔为日历月， “MS”原理一样，取第一日

groupby 索引为当月最后一日datetime。

resample 同上

to\_period 索引为年月，全数据显示

asfreq 索引为datetime，留下当月最后日数据。

“BM” 。“BMS”同理，取当月第一个工作日

groupby 索引为当月最后工作日datetime。范围为上月最后工作日后到本月最后工作日

resample 同上

to\_period 无效

asfreq 索引为datetime，留下当月最后工作日数据。

“W-MON，W-TUE” 自定义周间隔

groupby 索引为自定义周最后一日datetime。范围为自定义周

resample 同上

to\_period 索引为date/date标记周起止日期，显示全数据

asfreq 索引为datetime，留下自定义周最后日数据。

“WOM-1MON，WOM-2MON” 自定义月间隔

groupby 索引为每月的第一个周一datetime。范围为每月第一个周一到下月第一个周一前。

resample 同上

to\_period 无效

asfreq 索引为datetime，留下每月第一个周一数据。

“Q-JAN,Q-FEB” 自定义季度间隔

groupby 索引为自定义月结尾年的当季最后一天datetime。范围为自定义月结尾年的每季度。

resample 同上

to\_period 索引为年季度，过了自定义结尾月即为下一年。所有数据显示。

asfreq 索引为datetime，留下自定义月结尾年的当季最后一天数据。

“BQ-”

groupby 索引为自定义月结尾年的当季最后一个工作日datetime。范围为自定义月结尾年的上一个季度最后一个工作日到本季度最后一个工作日。

to\_period 无效

asfreq 索引为datetime，留下自定义月结尾年的当季最后一工作日数据。

“QS-JAN”

自定义月开始年的当季度第一天datetime，范围为自定义月开始年的每季度。

“BQS-”，“A-”年，“BA-”，“AS-”，“BAS-”同理

**2.3探索分析**

**2.3.1分析相关性**

Df生成相关系数矩阵用.corr(method='pearson', min\_periods=1)

method：可选值为 {'pearson', 'kendall', 'spearman'}默认使用pearson方法。

pearson：Pearson相关系数来衡量两个数据集合是否在一条线上面，即针对线性数据的相关系数计算，针对非线性数据便会有误差。

kendall：用于反映分类变量相关性的指标，即针对无序序列的相关系数，非正太分布的数据

spearman：非线性的，非正太分析的数据的相关系数

只想关注某一个变量与其余变量的相关系数的话df.corrwith(df['x1'])

生成相关矩阵图from pandas.plotting import scatter\_matrix

生成协方差矩阵df.cov()

from pandas.plotting import scatter\_matrix

scatter\_matrix(df, figsize=(12, 8))

df或者series用plot()方法直接画图。默认线图，通过参数kind='scatter'画散点图，通过hist()方法话单列数据分布直方图。Hist与bar的区别是hist自动会汇总各段频次。Bar需要先计算好。

Array数组则通过通用函数plt.plot(x,y)来画图。

housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,

s=housing["population"]/100, label="population", figsize=(10,7),

c="median\_house\_value", cmap=plt.get\_cmap("jet"), colorbar=True,

sharex=False)

对于数值型变量可以用来直接映射颜色、大小、透明度等。分类变量则只能多次作图来区别。

数据框内置绘图函数

DataFrame.plot(x=只需要写列名, y=None, kind='line', ax=None, subplots=False,

sharex=None, sharey=False, layout=None, figsize=None,

use\_index=True, title=None, grid=None, legend=True,

style=None, logx=False, logy=False, loglog=False,

xticks=None, yticks=None, xlim=None, ylim=None, rot=None,

fontsize=None, colormap=None, position=0.5, table=False, yerr=None,

xerr=None, stacked=True/False, sort\_columns=False,

secondary\_y=False, mark\_right=True, \*\*kwds)

x : label or position, default None#指数据列的标签或位置参数

y : label, position or list of label, positions, default None

kind : str#绘图类型

‘line’ : line plot (default)#折线图

‘bar’ : vertical bar plot#条形图。stacked为True时为堆叠的柱状图

‘barh’ : horizontal bar plot#横向条形图

‘hist’ : histogram#直方图（数值频率分布）

‘box’ : boxplot#箱型图

‘kde’ : Kernel Density Estimation plot#密度图，主要对柱状图添加Kernel 概率密度线

‘density’ : same as ‘kde’

‘area’ : area plot#与x轴所围区域图（面积图）。Stacked=True时，每列必须全部为正或负值，stacked=False时，对数据没有要求

‘pie’ : pie plot#饼图。数值必须为正值，需指定Y轴或者subplots=True

‘scatter’ : scatter plot#散点图。需指定X轴Y轴

‘hexbin’ : hexbin plot#蜂巢图。需指定X轴Y轴

**2.3.2变量重要性判断**

主要还是相关系数 决策树模型用 .feature\_importances\_

1. **数据选择或者集成**

数据选择是指选取原数据的一部分，或者对原数据进行划分，或者多个数据表集成新数据表，但不改动数据内容。

**3.1数据随机抽样**

**3.1.1随机抽样**

1、 按照个数抽样

data.sample(n=10)从data中随机抽取10条数据

2、按照百分比进行抽样

data.sample(frac=0.02)

3、有放回的抽样，有可能两次抽到同样的数据

data.sample(n=10,replace=True)

**3.1.2分层抽样**

train\_test\_split 函数 StratifiedShuffleSplit 类 均可。输出的内容不同，前者直接数据集，后者输出index。需要用.loc[]来选择。

train\_set, test\_set = train\_test\_split(housing, test\_size=0.2,

stratify=housing[‘income\_cat‘], random\_state=42)

from sklearn.model\_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n\_splits=1, test\_size=0.2, random\_state=42)

for train\_index, test\_index in split.split(housing, housing["income\_cat"]):

strat\_train\_set = housing.loc[train\_index]

strat\_test\_set = housing.loc[test\_index]

**3.1.3 关于研究样本如何选择的问题**

样本的选择应该注意均衡的问题。可能所选样本各属性类别比例并不符合总体分布比例，比如男比比例，总体时自然人口分布比如男51% 、49%的话。如果要样本符合总体分布需要按照样本容量分别制定各层抽样的样本数。以达到模拟总体的效果。

但是如果所能获得总体本身的分布，目标变量不均衡，会造成模型判断偏向某一个结果，那么选取样本是要按照目标分类变量各个值均匀分布。

在实际应用中涉及卡方检验的问题。应该取app所有用户，计算用户的性别，地区，年龄比例，然后再计算评论歌曲用户中关于的的各维度的比例人数，进行卡方检验，比明显比总体多才能判断地区影响显著。

一个指标STG

**3.2人为指定数据切片、切块、提取元素**

二维数组array可以ararry[，]中填入两个索引进行选择。其中一个是单数字的话会降维。

也可以s[1][4]

扩展一下，若想同时取多个元素，则把行号，列号对应的元素封装成list，或者range,也可以用切片

test3[:,2:]

>>> s[[0,1,2],[1,8,3]]

array([ 1, 18, 23])

>>> s[range(3),range(4,7)]

array([ 4, 15, 26])

**dataframe的行列选择:**

3.2.1df[] 数据框索引操作，

**填入一个参数：**

选择列情况：

填入单元素：df[“列名”]，填入列表：df[列名列表]，选择多列

无法只填一个列序号选单列，因为只填一个非切片值按照规当成单列名，

选择行情况：

填入切片：df[:] 用切片可以序号也可以行索引名，选择多行

选择在index是datetime格式

df['2013']# 获取2013年的数据

df['2016':'2017'] #获取2016至2017年的数据

df['2013-11'] # 获取某月的数据

df['2013-11-06':'2013-11-06']#选择某一天

df.truncate(after = '2013-11')，df[:'2013-11']某个日期之前，用切片更通俗。

df.truncate(before='2017-02')，df['2017-02':]某个日期之后

与上面规则不矛盾，选年，月，均可以看成多行切片。切片左右均包。

选择单行则必须用切片方式。

填入布尔列表：选择多行，必须与行同长。

**填入多个参数不能交叉选择**：相当与填入元组，是复合索引和复合列的选择方法。后面会详细介绍。

**如果需要行列交叉选择用以下方法：**

3.2.2df.iloc[] 只能填单个位置序号，位置序号切片，位置序号列表（数组）方便隔行选取和布尔值列表。

一个参数：

单位置序号，选择单行。结果为series

数字序号切片，选择多行。结果数据框。

数字序号列表，选择多行，结果为数据框。

布尔值列表，必须和行一样长，结果为数据框。

两个参数：

选择列必须有”，”分隔两个参数，行不能空需要填“：”：

原理同上，第一个选择行，第二个选择列

3.2.3loc[]只能填单个索引名，索引名列表（数组），索引名切片，布尔值列表。

一个参数：

单索引名，选择单行。结果为series

索引名切片，选择多行。结果数据框。

索引名列表，选择多行，结果为数据框。

布尔值列表，必须和行一样长。结果为数据框

两个参数：

原理同上，第一个选择行，第二个选择列

无论几个参数，只要选择单行或者单列，必降维成series。Series也可以有复合索引。

3.2.4iat[]位置序号提取一个数值

3.2.5df.drop()不能填位置序号，只能填索引名或者列名，axis参数控制行或列。索引名或者列名可以为单值也可以是列表。因为是方法不能用切片。

**3.2.6多重索引**

设定多层索引

import pandas

np.arrays = [[‘one’,‘one’,‘one’,‘two’,‘two’,‘two’],[1,2,3,1,2,3]]

df = pd.DataFrame(np.random.randn(6,2),index=pd.MultiIndex.from\_tuples(list(zip(\*np.arrays))),columns=[‘A’,‘B’])

df = pandas.DataFrame({'group1': ['a','a','a','b','b','b'],

'group2': ['c','c','d','d','d','e'],

'value1': [1.1,2,3,4,5,6],

'value2': [7.1,8,9,10,11,12]

})

df = df.set\_index(['group1', 'group2'])

多重列索引用元组选择（列表不行）

行列同时选择用loc和iloc

test5.loc['2018':'2019',[('广州自来水公司','售水量'),('广州自来水公司','供水总量')]]

多重索引如何简化列索引

df[‘0级列标签’]，sup\_water.columns=sup\_water.columns.get\_level\_values(1)来将复合列标签降维。复合列一般每一层级列标签会具有名称(大多是由于pivot的列标签转变而来)，在降维的时候会保存，可以通过sup\_water.columns = sup\_water.columns.get\_level\_values(1).values 来去除列标签维度名称，简化数据框。

或者

---------------------------------------------------------------------------------

多重行列索引灵活选择用（**更推荐**）

DataFrame.xs(key, axis=0, level=None, drop\_level=True)

举例：

num\_legs num\_wings

class animal locomotion

mammal cat walks 4 0

dog walks 4 0

bat flies 2 2

bird penguin walks 2 2

>>> df.xs('mammal')

num\_legs num\_wings

animal locomotion

cat walks 4 0

dog walks 4 0

bat flies 2 2

>>> df.xs(('mammal', 'dog'))

num\_legs num\_wings

locomotion

walks 4 0

获取多个索引和级别的值

>>> df.xs(('bird', 'walks'),

... level=[0, 'locomotion'])

num\_legs num\_wings

animal

penguin 2 2

获取指定列和轴上的值

>>> df.xs('num\_wings', axis=1)

class animal locomotion

mammal cat walks 0

dog walks 0

bat flies 2

bird penguin walks 2

Name: num\_wings, dtype: int64

--------------------------------------------------------------------------

df.columns.names = ['Language', 'Pass'] # 设置列索引层级的名

df.index.names = ['Class', 'Six'] # 设置行索引层级的名

# 修改索引层的名字

df\_ex.rename\_axis(index={'Extra':'Extra\_change'},

columns={'animal':'animal\_change'})

# ranme修改索引值，index/columns均可

df\_ex.rename(index = {'alpha': 'alpha\_change'}, level = 2)

#对于rename,传入参数也可以是函数，其输入值就是索引元素：

df\_ex.rename(columns = lambda x: str.upper(x), level=2)

#同时修改多层索引名字可以用map例如我们将上面的索引中的小写转化为大写，且在每个索引后加上 “\_change':

df\_temp = df\_ex.copy()

new\_idx = df\_temp.index.map(lambda x: (x[0]+'\_change',str.upper(x[1])+'\_change', str.upper(x[2])+'\_change'))

df\_temp.index = new\_idx

#map的另外一个常用的用法是用于多层索引的压缩，如下：

df\_temp = df\_ex.copy()

new\_idx = df\_temp.index.map(lambda x: (x[0]+'-'+x[1]+'-'+x[2]))

df\_temp.index = new\_idx

df\_temp.head() # 将原来的三层索引压缩为一层

#也可以反向展开：

new\_idx = df\_temp.index.map(lambda x:tuple(x.split('-')))

df\_temp.index = new\_idx

df\_temp# 三层索引

df.swaplevel('Six','Class', axis=0) # 按照输入顺序更改行索引的层级, *a*x*is=1*改变列标签层级

df\_ex.reorder\_levels([2,0,1], axis = 0) # 行索引改变顺序

df\_ex.droplevel([0,2], axis = 0)可以使用droplevel方法。同时删除多层，可以传入一个列表

df.sort\_index(axis=0, level=None或者list, ascending=True, inplace=False, kind='quicksort', na\_position='last', sort\_remaining=True, by=None)

na\_position：{‘first’, ‘last’}, default ‘last’，默认缺失值排在最后面。

(level= [0,1,2], axis=0, ascending=False) # 对行索引第一、二、三层的值进行降序排列

ascending=[True, False]参数可以是列表，分别指定各层索引的排序方式。

相同的：

df.sortlevel(level : int ,

axis : {0 or ‘index’, 1 or ‘columns’}, default 0,

ascending : boolean, default True,

inplace : boolean, default False,Sort the DataFrame without creating a new instance

sort\_remaining : boolean, default True Sort by the other levels too.)先对外层索引进行排序，再对内层索引进行排序，默认是升序。

df1.sum(level='OS',axis=1)；axis=1是按照列标签，默认0是按照行索引。Level是索引的层级可以是序号也可以是索引名称。

df.sum(level=1) 或df.sum(level='Six') # 按二级行索引层级或者Six进行求和

df.sum(level=0, axis=1) 或 df.sum(level='Language', axis=1) #按一级列索引层级或者Language列索引层级进行求和

df.stack(level=-1, dropna=True) # 默认将最内层的列标签(Pass)转换为了索引，表变长了

df.unstack(level=0) # 指定将第一层索引(Class)转化成列标签

**3.3数据指定条件筛选和过滤**

data2[data2['col3']==True]：选择符合条件的行返回df，以单独列为基础选择符合条件的数据

data2[data2=='a']：对df的元素进行操作，符合条件的元素返回原值，不符合条件的元素返回NaN。可以对符合条件的元素赋值。

data2[(data2['col2']=='a') & (data2['col3']==True)]：多行多条件选择

data2[data2['col1'].isin([1,2])]：使用isin查找范围

data2.query('col2=="b"')：使用字符串作为查询条件。

**Pandas 按数据类型选择列**

pandas 的 df 提供了 select\_dtypes 函数，可以按数据类型选择 df 的列。该函数包含 include 与 exclude 参数：

df = pd.DataFrame({'a': [1, 2] \* 3,

'b': [True, False] \* 3,

'c': [1.0, 2.0] \* 3})

# 输出包含 bool 数据类型的列

df.select\_dtypes(include='bool')

# 输出包含小数数据类型的列

df.select\_dtypes(include=['float64'])

# 输出排除整数的列

df.select\_dtypes(exclude=['int64'])

选择所有数字类型的列，用 np.number 或 'number'

选择字符串类型的列，必须用 object，注意，这将返回所有数据类型为 object 的列

选择日期时间类型的列，用np.datetime64、'datetime' 或 'datetime64'

选择 timedelta 类型的列，用np.timedelta64、'timedelta' 或 'timedelta64'

选择 category 类型类别，用 'category'

选择 datetimetz 类型的列，用'datetimetz'或 'datetime64[ns, tz]'



**3.4数据按照指定的比例的随机分离成训练和测试集：就是二维数据随机抽样过程**

3.4.1随机分离：train\_set,test\_set=sklearn.model\_selection.train\_test\_split(housing,test\_size=0.2, andom\_state=42)

3.4.2按照重要属性进行分层划分

参考3.1.2分层抽样

**3.5数据框合并和匹配**

左右条件匹配合并pd.merge(left, right, how='inner', on=None, left\_on=None, right\_on=None,

left\_index=False, right\_index=False, sort=True,

suffixes=('\_x', '\_y'), copy=True, indicator=False,

validate=None)

how: One of ‘left’, ‘right’, ‘outer’, ‘inner’. 默认inner。inner是取交集，outer取并集

注意：当被merge被并入一方有多条记录符合被并入一方的合并条件时候，不会报错，而是变成相应条数的合并记录。

left\_index 是否使用左表的行索引作为连接键，默认False

right\_index 是否使用右表的行索引作为连接键，默认False

sort: 按字典顺序通过连接键对结果DataFrame进行排序。 默认为True，设置为False将在很多情况下显着提高性能。

copy: 始终从传递的DataFrame对象复制数据（默认为True），即使不需要重建索引也是如此。

indicator:将一列添加到名为\_merge的输出DataFrame，其中包含有关每行源的信息。 \_merge是分类类型，并且对于其合并键仅出现在“左”DataFrame中的观察值，取得值为left\_only，对于其合并键仅出现在“右”DataFrame中的观察值为right\_only，并且如果在两者中都找到观察点的合并键，则为left\_only。

左右连接pd.concat([data1,data2],axis=1,

Join=”outer”|”inner” 默认取索引并集，可以改取索引的交集

ignore\_index=False,keys=[,]) 默认axis=0 上下连接,keys给每个子data赋予第一层列索引

上下连接 data1.append(data2)等同等价于pd.concat((data1,data2), axis=0)

左右连接data1.join(data2,lsuffix='\_series', rsuffix='\_d2')

对于数组可以使用 np.concatenate((二维数组,二维数组),axis=0)，0是行链接，1是列链接

但是注意必须是二维数组，不是二维可以用reshape方法变换。

如果要1，2维数据通用可以用np.c\_[数组1,数组2]列连接两个数组，也可以将一个一维series转化成n\*1的二维数组（将一个一维度数组变成二维数组可以用reshape(-1，1)方法）。用np.r\_将连个二维数组按照行连接

np.vstack((a,b,c,d))，把一维数组按照行组成二维数组。

还可以np.column\_stack((x\_vals\_column, ones\_column)) 两个数组按照列合并。

np.row\_stack((x\_vals\_column, ones\_column))按照行合并。

1. **数据处理**

数据处理也叫数据准备，在深入了解数据后，对数据进行清洗和修改。

**4.1data2.T转置数据框，行和列转换**

**对于二维数组使用np.transpose（）**

**4.2数组用reshape（）重塑维度.**

一维数组转化成二维数组，reshape(行数,列数)，是按照从第一行从左到右开始填充的，reshape（层数,行数,列数），数据按照第一层，第一行从左到右开始排列，第一层填完再填第二层。

二维数组转化成一维数组

a.flatten（）会将一个多维数组按照行顺序返回一个一维度数组拷贝。不改变原来数组。加参数“F”，可以控制按照列顺序

b=a.ravel（）作用一样，区别是例如对b[0]=10修改，会作用到a。也就是说不是拷贝。

**4.3 行排序**

**4.3.1DataFrame.sort\_values(by, axis=0, ascending=True, inplace=False, kind='quicksort', na\_position='last')按值排序**

axis：{0 or ‘index’, 1 or ‘columns’}, default 0，默认按照列排序，即纵向排序；如果为1，则是横向排序。

by：str or list of str；如果axis=0，那么by="列标签"；如果axis=1，那么by="行名"。

ascending：布尔型，True则升序，如果by=['列标签1','列标签2']，则该参数可以是[True, False]，即第一字段升序，第二个降序。

inplace：布尔型，是否用排序后的数据框替换现有的数据框。

kind：排序方法，{‘quicksort’, ‘mergesort’, ‘heapsort’}, default ‘quicksort’。似乎不用太关心。

na\_position：{‘first’, ‘last’}, default ‘last’，默认缺失值排在最后面。

**4.3.2sort\_index(axis=0, level=None, ascending=True, inplace=False, kind='quicksort', na\_position='last', sort\_remaining=True, by=None)按索引排序**

axis：0按照行名排序；1按照列标签排序

level：默认None，否则按照给定的level顺序排列

ascending：默认True升序排列；False降序排列

inplace：默认False，否则排序之后的数据直接替换原来的数据框

kind：排序方法，{‘quicksort’, ‘mergesort’, ‘heapsort’}, default ‘quicksort’。似乎不用太关心。

na\_position：缺失值默认排在最后{"first","last"}

by：按照某一列或几列数据进行排序，类似sortvalues但是by参数貌似不建议使用

df.sort\_index(level=0, axis=0, ascending=False) # 对行索引第一层的值进行降序排列

**二维数组排列**

1.np.argsort(x, axis=0) #按列排序，1是按行排列

2. lexsort()

lexsort支持对数组按指定行或列的顺序排序；是间接排序，lexsort不修改原数组，返回索引。

a[np.lexsort(a.T)] 按最后一列顺序排序,a.T，是对数组的行排序

a[np.lexsort(-a.T)] 按最后一列逆序排序

a[np.lexsort(a[:,::-1].T)] 按第一列顺序排序

a.T[np.lexsort(a)].T按最后一行顺序排序

a.T[np.lexsort(a[::-1,:])].T按第一行顺序排序

#最好用的方法如下

idex=np.lexsort([-1\*data[:,2], data[:,1], data[:,0]])

#先按第一列升序，再按第二列升序，再按第三列降序....

#注意先按后边的关键词排序

sorted\_data = data[idex, :]

**列排序**

**4.3.3调整列的顺序并且添加新的手输入列**

feature.reindex(columns=[‘b’,’d’,’c’,’a’], fill\_value=1)列存在调整位置，列不存在增加列，并指定数值。对于原列名比较长和复杂可以用以下处理

col\_name = df.columns.tolist()

col\_name.insert(col\_name.index('B'),'D')# 在 B 列前面插入D。

**4.4 转换类型基于列**

4.4.1data2['col3'] = data2['col3'].astype(int)转换特定列的类型， 也可以用于np.array

4.4.2利用# pandas中pd.to\_numeric()，to\_datetime

处理pd.to\_numeric(df["Jan Units"],errors='coerce').fillna(0)

4.4.3通过.apply()。df["2016"].apply(lambda x: x.replace(",","").replace("$","")).astype("float64")

**Lambda 使用if结构**

f = lambda x: 'big' if x > 100 else 'small'

#或者

f = lambda x: ['small', 'big'][x>100]

4.4.3读取数据时指定。在read\_csv中 converters 参数 设置每一行的转换函数，dtype是简单转换。

df\_2 = pd.read\_csv("sales\_data\_types.csv",dtype={"Customer\_Number":"int"},converters={

"2016":convert\_currency,

"2017":convert\_currency,

"Percent Growth":convert\_percent,

"Jan Units":lambda x:pd.to\_numeric(x,errors="coerce"),

"Active":lambda x: np.where(x=="Y",True,False)

})

处理日期型数据

def parse(y,m,d,h):

return dt.datetime.strptime(' '.join([y,m,d,h]), '%Y %m %d %H')

dataset = pd.read\_csv(r'.\mypyworks\StatLedger\数据表\raw.csv',

parse\_dates = [['year', 'month', 'day', 'hour']], index\_col=0, date\_parser=parse)

**4.5行操作：基于列data2.drop\_duplicates(['col3'])去重重复行，**通过指定列设置去重的参照

keep： {‘first’, ‘last’, False}, 默认值 ‘first’

first： 保留重复项目的第一行，删除后面的重复行。

last： 保留重复项目的最后一行，删除前面所有重复行。

False： 删除整个重复项目，一个都不留。慎重使用。

**4.6重命名 行列操作**

4.6.1 重命列名 data2.rename(columns= {'col1':'A','col2':'B','col3':'C'})更新列名，

df2.rename(index={'beijing':'bj'}, columns = {'A':'aa'})同时更新列名行名。

或者data.columns=list df1.index =list

**4.6.2 自动行名称。.set\_index(),reset\_index()**

DataFrame.set\_index(keys, drop=True, append=False, inplace=False, verify\_integrity=False)

keys：列标签或列标签/数组列表，需要设置为索引的列,也就是说只能把列换成行索引。

drop：默认为True，删除用作新索引的列。

append：是否将列附加到现有索引，默认为False。已有索引继续保留。

inplace：输入布尔值，表示当前操作是否对原数据生效，默认为False。

verify\_integrity：检查新索引的副本。否则，请将检查推迟到必要时进行。将其设置为false将提高该方法的性能，默认为false。

DataFrame.reset\_index(level=None, drop=False, inplace=False, col\_level=0, col\_fill='')

level：数值类型可以为：int、str、tuple或list，默认无，仅从索引中删除给定行层级。默认情况下移除所有级别。控制了具体要还原的那个层级的索引 。

drop：当指定drop=False时，则索引列会被还原为普通列；否则，经设置后的新索引值被会丢弃。默认为False。

inplace：输入布尔值，表示当前操作是否对原数据生效，默认为False。

col\_level：数值类型为int或str，默认值为0，如果列索引有多个层级，则确定将标签插入到哪个层级。默认情况下，它将插入到第一级。

col\_fill：对象，默认‘’，如果列有多个层级，则确定其他级别的命名方式。如果没有，则重复索引名。

**4.7 查找、替换内容**

序列操作：

data2.replace('a','A',inplace = True),整体替换一个值。

df.replace(['A','29.54'],['B',100])

或者

df.replace({'A':'B',29.54:100}) 整体替换多个值

df.replace(['A','29.54'],'B') #全部替换变成一个值

data[‘score’].replace(999,0,inplace = True),基于某一列修改

data.replace({‘score’:{999:np.nan},{‘name’:{‘bob’:np:nan}})，基于多列修改。

df.replace(‘[A-Z]’,'字母',regex=True) #运用正则表达式

把列中每个元素单个拿出来替换其中一部分文本使用.str.replace()

Df[‘列’].str.replace(‘ab’,’AB’),因为它是str的方法，不能使用inplace参数。

Df[‘col’].str.contains(‘[M]+’)匹配M开头的记录，返回布尔值序列

df2[df2=='a']='B' dataframe 条件替换全局内容

df.loc[df['First Season'] > 1990, 'First Season'] = 1 条件替换某一列内容

(df['First Season'] > 1990).astype(int)#布尔值转数字

类似于上面返回序列的属性应用方法的还有riqibiao['日期'].dt.strftime("%Y")

非series操作：

自由的apply方法逐行行或者逐列运用函数。用str的方法

X.str.replace(old, new[, max])old -- 将被替换的子字符串。new -- 新字符串，用于替换old子字符串。max -- 可选字符串, 替换不超过 max 次

import re

re.sub(pattern, repl, string, count=0, flags=0)

pattern：表示正则表达式中的模式字符串；

repl：被替换的字符串（既可以是字符串，也可以是函数）；

string：要被处理的，要被替换的字符串；

count：匹配的次数, 默认是全部替换

flags：具体用处不详

gongshiku.apply(lambda x: re.findall(r'\b[a-z]\_\d+\_\d+\b',x['setformula']),axis=1)

（返回string中所有与pattern相匹配的全部字串，返回形式为数组）

.strip()删除字符串首尾字符

1. 默认删除行首或者行尾的空白符（包括'\n', '\r', '\t', ' ')

2. 能使用‘，’分隔去除多个值

‘good’.strip(‘G,b’)

1. lstrip()表示删除左边的，rstrip()表示删除右边的

其他方法 ：.split() 拆分成列表 .find()返回查询字符的索引，多个时返回第一个；不存在的字符返回-1 。 .lower() upper()

字符串的测试，返回布尔值。

S.startswith(prefix[,start[,end]])

#是否以指定’prefix’字符开头

S.endswith(suffix[,start[,end]])

#以指定的‘suffix’字符结尾

S.isalnum()

#是否全是字母和数字，并至少有一个字符

S.isalpha() #是否全是字母，并至少有一个字符

S.isdigit() #是否全是数字，并至少有一个字符

S.isspace() #是否全是空白字符，并至少有一个字符

S.islower() #S中的字母是否全是小写

S.isupper() #S中的字母是否便是大写

S.istitle() #S是否是首字母大写的

**字符串的格式化**

%[(name)][flags][width宽度][.precision精度]typecode，-左对齐，+正负号，0补零

test='%d...%-6d...%06d'%(x,x,x) #-号左对齐。0不足位数补零

Typecode：

s，获取传入对象的\_\_str\_\_方法的返回值，并将其格式化到指定位置

r，获取传入对象的\_\_repr\_\_方法的返回值，并将其格式化到指定位置

c，整数：将数字转换成其unicode对应的值，10进制范围为 0 <= i <= 1114111（py27则只支持0-255）；字符：将字符添加到指定位置

o，将整数转换成 八 进制表示，并将其格式化到指定位置

x，将整数转换成十六进制表示，并将其格式化到指定位置

d，将整数、浮点数转换成 十 进制表示，并将其格式化到指定位置

e，将整数、浮点数转换成科学计数法，并将其格式化到指定位置（小写e）

E，将整数、浮点数转换成科学计数法，并将其格式化到指定位置（大写E）

f， 将整数、浮点数转换成浮点数表示，并将其格式化到指定位置（默认保留小数点后6位）

F，同上

g，自动调整将整数、浮点数转换成 浮点型或科学计数法表示（超过6位数用科学计数法），并将其格式化到指定位置（如果是科学计数则是e；）

G，自动调整将整数、浮点数转换成 浮点型或科学计数法表示（超过6位数用科学计数法），并将其格式化到指定位置（如果是科学计数则是E；）

%，当字符串中存在格式化标志时，需要用 %%表示一个百分号 %.2f%%

format格式结构｛fieldname!conversionflag:formatspec｝，fieldname表示参数的一个数字位置或关键字，conversionflag可以是r,s,a对应repr/str/ascii内置函数的一次调用，formatspec指定了如何表示该值：字段宽度、对齐方式、补零、小数精度等。冒号后的formatspec组成形式有：[[fill]align对齐方式][sign][#][0][width宽度][.precision精度][typecode]

'{0:>10}={1:<10}'.format('test',12.62424)

fill 【可选】空白处填充的字符

align 【可选】对齐方式（需配合width使用）

<，内容左对齐

>，内容右对齐(默认)

＝，内容右对齐，将符号放置在填充字符的左侧，且只对数字类型有效。 即使：符号+填充物+数字

^，内容居中

sign 【可选】有无符号数字

+，正号加正，负号加负；

-，正号不变，负号加负；

空格 ，正号空格，负号加负；

# 【可选】对于二进制、八进制、十六进制，如果加上#，会显示 0b/0o/0x，否则不显示

， 【可选】为数字添加分隔符，如：1,000,000

width 【可选】格式化位所占宽度

.precision 【可选】小数位保留精度

type 【可选】格式化类型

传入” 字符串类型 “的参数

s，格式化字符串类型数据

空白，未指定类型，则默认是None，同s

传入“ 整数类型 ”的参数

b，将10进制整数自动转换成2进制表示然后格式化

c，将10进制整数自动转换为其对应的unicode字符

d，十进制整数

o，将10进制整数自动转换成8进制表示然后格式化；

x，将10进制整数自动转换成16进制表示然后格式化（小写x）

X，将10进制整数自动转换成16进制表示然后格式化（大写X）

传入“ 浮点型或小数类型 ”的参数

e， 转换为科学计数法（小写e）表示，然后格式化；

E， 转换为科学计数法（大写E）表示，然后格式化;

f ， 转换为浮点型（默认小数点后保留6位）表示，然后格式化；

F， 转换为浮点型（默认小数点后保留6位）表示，然后格式化；

g， 自动在e和f中切换

G， 自动在E和F中切换

%，显示百分比（默认显示小数点后6位）

**删除中文字符和标点的函数**

def removeChnAndCharacter(str1):

#将中文标点符号转换为英文标点符号

def C\_trans\_to\_E(string):

E\_pun = u',.!?[]()<>"\''

C\_pun = u'，。！？【】（）《》“‘'

#ord返回ASCII码对应的int

#zip将合并为列表，元素为元祖，元祖为对应位置所有元素依次的集合，如这种形式[(',','，')...]

#s生成对应字典

table= {ord(f):ord(t) for f,t in zip(C\_pun,E\_pun)}

#将字符传对应转换

return string.translate(table)

C\_pun = u'，。！？【】（）《》“‘'

strTmp = ''

if not isinstance(str1,str):

return strTmp

for i in range(len(str1)):

#中文字符范围

#https://blog.csdn.net/qq\_22520587/article/details/62454354

if str1[i] >= u'\u4e00' and str1[i] <= u'\u9fa5' \

or str1[i] >= u'\u3300' and str1[i] <= u'\u33FF' \

or str1[i] >= u'\u3200' and str1[i] <= u'\u32FF' \

or str1[i] >= u'\u2700' and str1[i] <= u'\u27BF' \

or str1[i] >= u'\u2600' and str1[i] <= u'\u26FF' \

or str1[i] >= u'\uFE10' and str1[i] <= u'\uFE1F' \

or str1[i] >= u'\u2E80' and str1[i] <= u'\u2EFF' \

or str1[i] >= u'\u3000' and str1[i] <= u'\u303F' \

or str1[i] >= u'\u31C0' and str1[i] <= u'\u31EF' \

or str1[i] >= u'\u2FF0' and str1[i] <= u'\u2FFF' \

or str1[i] >= u'\u3100' and str1[i] <= u'\u312F' \

or str1[i] >= u'\u21A0' and str1[i] <= u'\u31BF' \

:

pass#中文字符不处理

else:

if str1[i] in C\_pun:

# st = C\_trans\_to\_E(str1[i])#中文标点不处理

pass

else:

st = str1[i]

strTmp += st

return strTmp

#使用函数

formuladetail['QUOTA\_DEPT\_CODE'].apply(removeChnAndCharacter)

**4.7缺失值填充**

data2.fillna(method='bfill')填充缺失值，整个表均可操作。可设置为固定值以及不同的填充方法

或者用 sklearn.imputer 模块 只对列操作

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")

imputer.fit\_transform(array)

**4.8删除缺失值：行列均可操作**

data2.dropna()去dataframe掉缺失值，

axis :0, or 'index'：删除包含丢失值的行

1, or 'columns'：删除包含丢失值的列

默认为0

how : {'any', 'all'}, default 'any'

'any': 如果存在NA值，则删除该行或列

'all': 如果所有值都是NA，则删除该行或列

thresh: int,保留含有int个非空值的行

subset: 对特定的列进行缺失值删除处理

inplace=True

data.dropna(axis=0,subset = ["Age", "Sex"])

**对于基础的list的删除方法**

a.pop(index):删除列表a中index处的值,并且返回这个值.

del(a[index]):删除列表a中index处的值,无返回值. del中的index可以是切片,所以可以实现批量删除.

a.remove(value):删除列表a中第一个等于value的值,无返回.

**4.9离群值判断与修正**

**4.9.1人工指定**

Df[df>999]=999

**4.9.2算法判断**

4.9.2.1对于时间序列的特殊性可以用移动平均线的距离的把个标准差来判断

def moving\_average(a, n=3): #构造一个移动平均函数

....ret = np.cumsum(np.pad(a, n, 'edge'), dtype=float) pad是填充函数

....ret[n:] = ret[n:] - ret[:-n]

....return ret[n:-n] / n

am = moving\_average(a, n=15) #计算移动平均数

或者变成series或者df 用rolling（）函数

am = Pd.Series(a).rolling(window=3, min\_periods=1缩小前面的NAN, ,center=False 以中心为基点移动平均,on=None对于多列DataFrame，用on来指定使用哪列, axis=0, closed=None)

也可以指数移动平均

ewm = Pd.Series(a).ewm(span=10).mean() span是根据范围指定衰减， α=2/(span+1), for span≥1。alpha ：直接指定平滑系数α， 0<α≤1。

.plot()方法画图

最后以三个标准差来找出离群

Diff = am-a

diff[diff.abs()>3 \* diff.std(ddof=0)]；diff.std()为pandas计算的为样本标准偏差，ddof默认是1意思是除以总数减去1，求数据为较小样本估计的总体标准差。ddof=0，的意思是离差的平方除以总数，即数据即总体，总体标准差，和np.std()一致。

np.argwhere(np.abs(a-am)>3 \* np.std(a-am)),返回array的索引，数组形式。

4.9.2.1 对于期望值固定的一维数据，最长用的是大于上或者小于下四分位数 1.5个IQR（四分位距）或者超过均值的正负3个标准差。3个IQR或者5个标准差为极端值

前者直接用箱线图来获得

df.boxplot(column=["合计"], by="meter\_calibre",figsize=(12,8)，如果是数据框必须指定列

sym = 'o', #异常点形状

vert = True, # 是否垂直

whis=1.5, # IQR

patch\_artist = True, # 上下四分位框是否填充

meanline = False,showmeans = True, # 是否有均值线及其形状

showbox = True, # 是否显示箱线

showfliers = True, #是否显示异常值

notch = False, # 中间箱体是否缺口

return\_type='dict') # 返回类型为字典

p = data.boxplot(return\_type='dict') #画箱线图，直接使用DataFrame的方法

x = p['fliers'][0].get\_xdata() # 'fliers'即为离群点的标签 获得图上x坐标位置。

y = p['fliers'][0].get\_ydata()

y.sort() #从小到大排序，该方法直接改变原对象

y就是离群点的数值

4.9.2.3 多维数据聚类算法，线性回归

**4.10计算添加新的列**

**4.10.1基础做法 ：apply 加函数做法。**

**4.10.1.1赋值计算法（最自由可用于数组）**

**4.10.1.2 用已有列的str属性中的方法处理整个序列**

Series(['a', 'b', 'c']).str.cat(['A', 'B', 'C'], sep=',')

Series(['a', 'b', 'c']).str.cat(sep=',')

        'a,b,c'

Series(['a', 'b']).str.cat([['x', 'y'], ['1', '2']], sep=',')

        0    a,x,1

        1    b,y,2

s.str.split('\_'，3) 按字符切开3次

get() 获取指定位置的字符串

join(‘！’) 对每个字符都用给定的字符串拼接起来

contains() 是否包含表达式

replace() 替换

repeat(3) 重复三次并连接

左补齐s.str.pad(10, fillchar="?")右补齐s.str.pad(10, side="right", fillchar="?")两边补齐

s.str.center(10, fillchar="?")

截取字符s.str.slice(1,3)

指定范围字符给替换s.str.slice\_replace(1, 3, "?") 将第二到第三个字符替换成问号

计算某个字母出现的频率s.str.count("a")

。。。。几乎所有字符串方法都适用

**巧妙运用字符串.split()方法拆分dataframe列，原理是运用list生成新数据框。**

df['A'], df['B'] = df['AB'].str.split('-', 1).str

grade\_split = pd.DataFrame((x.split('-') for x in

loandata.grade),index=loandata.index,columns=['grade','sub\_grade'])

**合并列（字符型）的方法**

1. 如果列全为文本型，则直接相加

psi['cc'] = psi['symbol'] + psi['as\_id']

1. 如果列不为字符型，则通过.map()方法转字符型

df2["period"] = df["Year"].map(str) + df["quarter"]

1. 万能apply方法

df1['period'] = df[['Year', 'quarter']].apply(lambda x: ''.join(x), axis=1)

4.10.1.3于用字符串作为计算命令 eval(“”)

df.eval(

... '''

... 净追收水量 = 追收水量合计 – 剔除水量合计

... D = A - B

... '''

... )

assign（）

同时生成多列 .assign(new\_col = sample[‘a’)-sample[‘b’],new\_col2 = sample[‘a’)+sample[‘b’])

Assign 使用iris.assign(Sepalration2 = lambda x:(x['SepalWidth']/x['SepalLength']))

Lambda函数才能允许依赖赋值，\*\*kwargs 后的表达式，可以引用同一个 assign() 函数里之前创建的列 。

可以写成函数，用FunctionTransformer包装成一个类,统一 fit transform方法

from sklearn.preprocessing import FunctionTransformer

def add\_extra\_features(X, add\_bedrooms\_per\_room=True):

rooms\_per\_household = X[:, rooms\_ix] / X[:, household\_ix]

population\_per\_household = X[:, population\_ix] / X[:, household\_ix]

if add\_bedrooms\_per\_room:

bedrooms\_per\_room = X[:, bedrooms\_ix] / X[:, rooms\_ix]

return np.c\_[X, rooms\_per\_household, population\_per\_household,

bedrooms\_per\_room]

else:

return np.c\_[X, rooms\_per\_household, population\_per\_household]

attr\_adder = FunctionTransformer(add\_extra\_features, validate=False,

kw\_args={"add\_bedrooms\_per\_room": False})

housing\_extra\_attribs = attr\_adder.fit\_transform(housing.values)

4.10.2多项式拓展特征属性列

preprocessing.PolynomialFeatures(degree=2)

.fit()方法用于数组输出数组

Degree表示最高项次数

interaction\_only：默认为 False，如果指定为 True，那么就不会有特征自己和自己结合的项，组合的特征中没有 a2 和 b2

include\_bias 设置 0 次（1）幂那一列是否要。也就是常数项。

多项式顺序 常数，每个特征的一次项，第一个特征的二次项，第一个特征的一次项分别与其他特征两两组合，第二个特征二次项，第二个特征的一次项分别与其他特征剩下的两两组合，。。。。，第一个特征的三次项目，第一个特征的二次项与其他特征的一次项两两组合，第一个特征的一次项目与其他特征的二次项组合。最好用powers属性查看。

.powers\_会返回特征参数，[0,0]a^0b^0, [1,0]a^1b^0, [0,1]a^0b^1, [2,0]a^2b^0 , [1,1]a^1b^1 , [02]a^0b^2

.n\_input\_featrues\_ 输入变量个数 .n\_output\_featrues\_ 输出变量个数

.get\_feature\_names()会返回列名和数据

4.10.3数据的无量纲化 sklearn preprocessing模块 输入数组

标准化preprocessing.StandardScaler()实用类StandarScaler，计算Z分位数。作用如上，不同在于可以估算器的参数转化验证数据集的数据。保持标准一致性。

归一化preprocessing.MinMaxScaler()规范在0-1之间

MaxAbsScaler（）规范在-1到1之间

正则化sklearn.preprocessing.Normalizer(norm=’l2’, copy=True)正则化 每行的元素除以该行的范数

||X||p=(|x1|^p+|x2|^p+...+|xn|^p)^1/p,l1的P=1，l2的p=2

preprocessing.Binarizer()特征的二值化是指将数值型的特征数据转换成布尔类型的值。默认是根据0来二值化，大于0的都标记为1，小于等于0的都标记为0，参数threshold 来设置阀值

RobustScaler 减去中位数，然后除以四分位差。

如果你的数据包含许多异常值，使用均值和方差缩放可能并不是一个很好的选择。这种情况下，你可以使用 robust\_scale 以及 RobustScaler 作为替代品。它们对你的数据的中心和范围使用更有鲁棒性的估计。

4.10.4类别数据数字化编码 .inverse\_transform() 反转

简单preprocessing.LabelEncoder .classes\_

有序f rom sklearn.preprocessing import OrdinalEncoder 返回数组ordinal\_encoder.categories\_

部分顺序 from sklearn.preprocessing import OneHotEncoder

preprocessing.OneHotEncoder().transform(array)返回稀疏矩阵 .toarray() 转化为数组

4.10.5在现有数据列指定划分点进行离散化

pd.cut(housing["median\_income"], bins=[0., 2,2.7, 4.5, 6., np.inf],labels=[1, 2, 3, 4, 5])

pandas.cut(x, bins, right=True, labels=None, retbins=False, precision=3, include\_lowest=False)

参数：

x : 输入待cut的一维数组

bins : cut的段数，一般为整型，但也可以为序列向量(若不在该序列中，则是NaN)。

right : 布尔值，确定右区间是否开闭，取True时右区间闭合

labels : 数组或布尔值，默认为None，用来标识分后的bins，长度必须与结果bins相等，返回值为整数或者对bins的标识

retbins : 布尔值，可选。是否返回数值所在分组，Ture则返回

precision : 整型，bins小数精度，也就是数据以几位小数显示

include\_lowest : 布尔类型，是否包含左区间

**4.11 指定位置添加、计算新的行。apply 加函数做法也适用。Axis参数**

**最好用的reindex**

**df1.reindex(index=['a','b','c','d'],columns=['one','two','three','four'],fill\_value=100,method=)，可以在任意位置增加空行或者填充值，最后用赋值法。method=’ffill’或’pad 前向填充method=’bfill’或’backfill 后向填充**

df.loc['c'或者6]=['test','test'] ###指定位置插入一行

构造一个单行或者多行的df，通过append方法插入

df = df1.append(df3, ignore\_index = True).append(df2, ignore\_index = True)

也可以是append字典

row={'a':9,'b':10,'c':11,'d':12}

df.append(row,ignore\_index=True)

**4.12 dataframe或者series整体变化同比环比**

4.12.1数据框整体上移动一行 data1.shift(periods=-1, axis=0) 通常和自己计算计算变化比率。

4.12.2或者直接计算比例函数：DataFrame（Series）.pct\_change(periods=1, fill\_method=‘pad’, limit=None, freq=None, \*\*kwargs)

表示当前元素与先前元素的相差百分比，当然指定periods=n,表示当前元素与先前n 个元素的相差百分比。df.pct\_change(axis='columns'))#可以指定列进行计算的

4.12.3 也可以计算整体差额：df.diff(periods=1, axis=0)

以上函数可以dataframe整体运算也可以分单个series计算。

4.12.3 applymap或者dataframe.tansform(lambda x : x+1) .tansform([np.sqrt,np.exp])

**正常情况下，pandas 会给数据列自动设置默认的数据类型，其中最令人讨厌并且最消耗内存的数据类型就是object(O)，这也恰好限制了 pandas 的一些功能。下面是 pandas 、Python、Numpy的数据类型列表，对比你就发现pandas的数据类型是有很大优化空间的。**

| **Pandas dtype** | **Python type** | **NumPy type** | **Usage** |
| --- | --- | --- | --- |
| object | str | string\_,unicode | Text |
| int64 | int | int,int8,intl6,int32,int64,uint8,uint16,uint32,uint64 | Integer numbers |
| float64 | float | float,float16,float32,float64 | Floating point numbers |
| bool | bool | bool\_ | True/False values |
| datetime64 | NA | datetime64[ns] | Date and time values |
| timedelta[ns] | NA | NA | Differences between two datetimes |
| category | NA | NA | Finite list of text values |

很多默认的数据类型占用很多内存空间，其实根据没有必要，我们完全可以压缩到可能小的子类型。

| **Data type** | **Description** |
| --- | --- |
| bool\_ | Boolean(True or False) stored as a byte |
| int\_ | Default integer type(same as C 1ong ; normally either int64or int32) |
| intc | ldentical to C int(normally int32 or int64) |
| intp | Integer used for indexing(same as C ssize\_t; normally either int32 or int64) |
| int | 8Byte(-128 to 127) |
| int16 | Integer(-32768 to 32767) |
| int32 | Integer(-2147483648 to 2147483647) |
| int64 | Integer(-9223372036854775808 to 9223372036854775807) |
| uint8 | Unsigned integer(0 to 255) |
| uint16 | Unsigned integer(0 to 65535) |
| uint32 | Unsigned integer(0 to 4294967295) |
| uint64 | Unsigned integer(0 to 18446744073709551615) |
| float\_ | Shorthand for float64. |
| float16 | Half precision float: sign bit,5 bits exponent,10 bits mantissa |
| float32 | Single precision float: sign bit,8 bits exponent,23 bits mantissa |
| float64 | Double precision float: sign bit,11 bits exponent,52 bits mantissa |
| complex\_ | Shorthand for complex128. |
| complex64 | Complex number, represented by two 32-bit floats(real and imaginary components) |
| complex128 | Complex number, represented by two 64-bit floats(real and imaginary components) |

上面是scipy文档中列出的所有数据类型，从简单到复杂。我们希望将类型简单化，以此节省内存，比如将浮点数转换为float16/32，或者将具有正整数和负整数的列转为int8/16/32，还可以将布尔值转换为uint8，甚至仅使用正整数来进一步减少内存消耗。

基于上面所说的变量类型简化的思考，写出一个自动转化的函数，它可以根据上表将浮点数和整数转换为它们的最小子类型：

def reduce\_memory\_usage(df, verbose=True):

numerics = ["int8", "int16", "int32", "int64", "float16", "float32", "float64"]

start\_mem = df.memory\_usage().sum() / 1024 \*\* 2

for col in df.columns:

col\_type = df[col].dtypes

if col\_type in numerics:

c\_min = df[col].min()

c\_max = df[col].max()

if str(col\_type)[:3] == "int":

if c\_min > np.iinfo(np.int8).min and c\_max < np.iinfo(np.int8).max:

df[col] = df[col].astype(np.int8)

elif c\_min > np.iinfo(np.int16).min and c\_max < np.iinfo(np.int16).max:

df[col] = df[col].astype(np.int16)

elif c\_min > np.iinfo(np.int32).min and c\_max < np.iinfo(np.int32).max:

df[col] = df[col].astype(np.int32)

elif c\_min > np.iinfo(np.int64).min and c\_max < np.iinfo(np.int64).max:

df[col] = df[col].astype(np.int64)

else:

if (

c\_min > np.finfo(np.float16).min

and c\_max < np.finfo(np.float16).max

):

df[col] = df[col].astype(np.float16)

elif (

c\_min > np.finfo(np.float32).min

and c\_max < np.finfo(np.float32).max

):

df[col] = df[col].astype(np.float32)

else:

df[col] = df[col].astype(np.float64)

end\_mem = df.memory\_usage().sum() / 1024 \*\* 2

if verbose:

print(

"Mem. usage decreased to {:.2f} Mb ({:.1f}% reduction)".format(

end\_mem, 100 \* (start\_mem - end\_mem) / start\_mem

)

)

return df

当然，这个函数不是固定，只是提供个模板，大家可以直接复制拿过去改成自己习惯的方式。