# 机器学习在数据获取、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：**

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，多重索引则用嵌套列表，长度对齐。

类方法DataFrame.from\_dict() :

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():

参数是字典的items，也可以生成数据框。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。

**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

pandas.read\_excel（io，sheet\_name = 0，header = 0，names = None或者要使用的列名列表，index\_col = None，usecols = int或list（支持切片元素），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}, ...）

**1.2.1.1 Excel 自由读写两个模块**

**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，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 ;dataset.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）

**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生成一些公式推导列表推导[x\*x for x in range(10) if x%3 == 0] [(x,y) for x in range(3) for y in range(3) ]

array(list)

1.3.3 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。

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。

1.3.4利用np生成：

生成两个数据之间间距均匀的数据np.linspace(prices.min(), prices.max(), 7)

随机分布np.random模块

随机整数分布

np.random.randint(low，hight，n)生成随机整数

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]

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

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

**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）

...

**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：

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duplicated

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()#显示关键索引下的重复行keep{‘first’, ‘last’, False 显示所有}, default ‘first’。

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。

利用索引可以筛选空值，.isull叠加.any(axis=1)方法可以选择任意行元素返回为True的行。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。

只判断NULL是不够的。 np有丰富的方法判断np中的数据类型，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

np.isfinite：显示哪些元素是有限的（不是非数字，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。

.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

提取列名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

**2.2 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

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。

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

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)

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

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

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]))))

**2.1.2 对于时间的分类汇总**

to\_period、asfreq显示汇总索引不统计，groupby（pd.Grouper（））、resample显示并统计。

“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(‘spearman’)方法,pearson方法或kendell方法或spearman方法，默认使用pearson方法。

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。

matplotlib.pyplot.plot()参数详解 plt.plot([x], y, [fmt]= '[color][marker][line]', [x2], y2, [fmt2], ..., \*\*kwargs)

实例：plt.plot(x,y2,color='green', marker='o', linestyle='dashed', linewidth=1, markersize=6)

各个绘图类型参数不同函数名称不同，比如plt.scatter()增加点大小的s参数

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**

data2.sample(n=2) 从data2中随机抽取2条数据

**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]

**3.2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。

dataframe的行列选择:

3.2.1df[] 数据框索引操作，

只能填入一个参数：

填入单元素：df[“列名”]，选择在index是datetime时候填入’2016’,’2016-1’这种含有多天的日期范围，自动转化成行索引日期切片。进行行选择。

填入列表：df[列索引列表]，选择多列

填入切片：df[:] 用切片可以序号也可以行索引名，选择多行

填入布尔列表：选择多行，必须与行同长。

对于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

与上面规则不重复，选年，月，均可以看成多行切片。切片左右均包。无法只填一个索引选单列，只填一个值按照规则为选择单列，选择单行需要用切片方式。

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）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：

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

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

多重索引用元组选择

**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。

**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数据合并和匹配**

左右条件匹配合并pd.merge(,on='col1',how='inner')

左右连接pd.concat([data1,data2],axis=1, ignore\_index=False) 默认axis=0 上下连接

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,二位数组),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(x,y)，x行y列，是按照从第一行从左到右开始填充的，reshape（z,x,y）,z层，x行，y列，数据按照第一层，第一行从左到右开始排列。

**4.3 行排序**

**4.3.1data2.sort\_values(['col1'])按值排序，**

默认为正序，可通过ascending=False指定倒序排序

**4.3.2data2.sort\_index(ascending=False)按索引排序，**

默认为正序，可通过ascending=False指定倒序排序，指定新index set\_index()

**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")

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)

})

**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

**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()

**4.7 查找、替换内容**

序列操作：

data2.replace('a','A'),按列指定不同替换内容，基于行列。replace({‘score’:{999:np.nan},{‘name’:{‘bob’:np:nan}})，基于列。

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

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方法里面进入的每行或者每列。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()返回查早字符的索引 .lower() upper()

**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()去掉缺失值，可通过axis设置为0或 index、1或columns丢弃带有缺失值的行或列。how='any',any-只要有空值就删除（默认），all-全部为空值才删除 inplace=True

**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>3 \* diff.std(ddof=0)]；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，默认是1意思是除以总数减去1，求数据为样本估计的总体标准差

np.argwhere(np.abs(a-am)>3 \* np.std(a-am))

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。**

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

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

4.10.1.3于用字符串作为计算命令 eval(“”) assign（）

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

(少用)

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。顺序 1，每个特征，特征最高项，特征交叉项，特征最高项目

.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，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

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**

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)

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