# 零，数据分析简介

**什么叫数据分析，它要达到何种目的。**

数据分析是指的是通过统计学方法对采集来的数据进行整理分析，从中提取有用信息并最终得出结论的过程。

某些学者认为，数据分析分为三类，入门级的描述性数据分析，其方法主要有对比、平均、交叉分析法。高级的探索、和验证数据分析，分析方法有主要有相关分析，回归分析，因子分析。这样的提法有它自己的道理。在我看来，实际上就两类：描述性统计分析和计算性数据分析。

在日常生活和工作中，我们经常不自觉得进行着或者接触到数据分析，比如手机费连续几个月的陡升，我们往往会查看这几个月的电子账单，查找电话费上升的原因。再比如，比较北京市各区房价，在考虑自身经济和位置状况的前提下，我们选择价格与条件性价比最高的房子。工作中，数据分析的例子就更多了。比如销售额的增长状况，网络平台流量分析，推广效果，客户状况分布图等等。以上这些都是数据分析。

**数据分析的一般流程**

我们这里着重前五个步骤。

1. 明确分析目的

数据分析要根据目的选择分析方式。很多公司做分析的员工根本不知道分析的最终目的，或者告诉她，她也不是很在乎。只在乎你让她做什么，不在乎为什么这样做。或许这是我们独特企业文化造成的最不应该有的一个事实。然而，没有目的的数据分析往往会被数据本身淹没，深陷其中，且抓不住数据分析的重点，总是认为采用哪些所谓的数学建模等高大上的数学分析方法才算完美。实际上这种分析方式完全不可取的。分析前一定要目的明确，根据目的选择适合的分析方式。我们宗旨是用最节省脑细胞的方式分析数据。很多时候，一些一眼就能看出或者稍作分析就能得到结论的问题我们就不要做什么复杂的数据分析方法。根据我的经验，80%的数据问题都可以以简单的分析方式解决。如果分析没有目的，闷着头按照固定的分析套路走，很多时候用了半天劲，才发现一无所获。根据目的，选择刚刚好的分析方式是我们每位数据分析师应该把握的人生哲学。

分析的目的分为三类：

1. 对现状进行描述性分析，给决策者提供未来发展方向的依据。
2. 原因分析，弄清造成这种现状的原因。
3. 为事物将来的发展趋势做出预测。指导决策者做出相关应对措施。向有利方向发展的趋势采取加强或鼓励措施。反之，则是通过有效手段降低和弱化甚至消除不利的发展趋势。

明确分析目的后，确定详细的分析思路，也就是找到目的达成的方法。在这一步骤，我的习惯是参考现有的数据分析方法论画出解决问题的草图。先分析什么，后分析什么，怎么分析，都要详细的写在草图上，形成体系化的分析框架。

那么现有的分析理论都有哪些呢。稍后会有介绍。

1. 数据收集，有了分析框架，我们就要收集适合分析框架的数据。我以前做的数据分析，其数据大都来自自己互联网平台数据，数据采集可以说是信手拈来，想要什么数据就有什么数据，所以从来就没有数据收集的概念。其实，数据收集很多时候是一件十分辛苦的事？

关于如何收集数据，不是本课程关心的问题，这里不再做描述。

1. 数据的处理

数据处理常用的方法有四种：数据清洗，数据转化，数据提取，数据计算。

其目的是把杂乱无章的数据处理成可以分析的数据。

1. 数据分析

定义中已经讲的很清楚，我这里不再重复。

这里我想说下数据分析师的分级：

初级分析师应该能用excel做数据分析，对概率论和数理统计以及线性代数有稍许了解。

中级分析师应该能用Spss，Access，Pivot做数据分析，对概率论和数理统计以及线性代数有一定了解。

高级分析师除了能用各种软件，对Python, R，VBA等语言也十分精通，对概率论以及数理统计和线性代数也十分熟悉，并且对优选法也有一定了解。甚至对数据开发和挖掘也有一定经验。

我们教学的目的是通过我们的课程你能达到初级分析师的水平，同时掌握只有高级分析师具备的python编程能力。

下面我们谈一下数据挖掘，数据挖掘就是数据分析，但是数据分析不一定是数据挖掘。也就是他们之间是一种包含与被包含的关系。数据挖掘是一种高级数据分析方法。它和数学几乎没什么两样。属于数据分析难度的最高阶段。具体方法包括分类、聚类、关联和预测。

1. 数据展现

其实就是画图，根据数据的实际情况，画出最能展现数据的图表。

1. 数据报告

这个就看你的文学水平了。不仅要文笔好，而且要条理清晰。把数据分析的起因，过程，结论，建议都要写得清清楚楚，一目了然。

**数据分析之大忌**

**a.** 无目的的分析，为分析而分析

这方面前面已经讲过。

b. 不熟悉业务

如果一个分析师不熟悉他研究的数据所产生的环境，那么他的分析结果有多少可信度，真心无法估计。比如，一个从未干过销售业务的数据分析师去分析销售数据，纯粹是瞎子摸象，只识局部，不知全局。分析结果也就十分片面化，甚至是错误的。

C. 热衷高级分析法，一味建模

正如我前面提到的，要根据目的和实际情况来决定采用哪种分析方法。遵循的原则是，简单分析法能解决的问题，绝不用高级分析法。

**几个常见的容易搞混淆统计术语**

相对数与绝对数：

举个例子：一个城市总人口300万，这个300万是绝对数，一个人体重是另一个人的1.5倍，那么这个1.5就是相对数。

**绝对数描述客观事物总体在一定时间和地点条件下的总规模，总水平的综合性指标。**

**相对数是指两个相关事物的比值。**

百分数和百分点：

男生占班级总人数的90%，男生人数比上一年增长7个百分点，百分数表示个体占总体的程度。百分点表示相同事物不同时期的增幅。

频数频率：

频数是绝对数，频率是相对数。

频数指某种事物或现象在其所在总体出里出现的次数。比如班级有两个女生，我们就可以说女生出现的频数是2.

频率（注：不要和物理频率搞混），个体出现的总次数与总体出现的总次数的比值。

一个六口之家，有爷爷、奶奶、爸爸、妈妈、弟弟和我，老人出现的频率是33.33%，中年人出现的频率33.33%，小孩出现的频率33.33%.

比例和比率：

个体数值在总体数值中的占比。比如一个班级10个人，男生4个，女生6个，则男生的所占比例是4 ：10；女生所占比例是6 ：10。

比率是总体中各个体数值之间的对比。男女比率是：4 ：6

倍数与番数：

倍数是一个数除以另一个数所得的商。比如3是1.5的二倍。

番数指的是某种事物总数的二的n次方倍。

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program. 表示A的总数翻一番，现在A的总数是原来的2倍。

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program. 表示A的总数翻两番，现在A的总数是原来的4倍。

.

.

.

同比与环比：

同比是指与历史同时期进行比较得到的数值；环比是指与前一个统计期比较所得到的数值。

这个产品的销售额与上一年相比，同比增长40%；

这个商品的销售额与上个月相比，环比增长30%。

**数据分析方法论**

这个实在是太重要，他是数据分析两大基石之一。数据分析的两大基石是分析方法论和数学。如果方法论不对，你的分析结果基本不用看，看也是白看。如果方法论对了，后面的数学错了，也可以基本说分析失败。

但是，这里我有一点要提醒大家，我强调方法论的重要性，并不是说数据分析不根据实际情况死搬硬套方法论。方法论是死的，人是活的。我们要活学活用。很多时候一个问题解决是依靠一个或者多个方法。甚至有时候需要在已有方法论的基础上创新出新的适合问题解决的方法。

数据分析方法论（一个宏观概念，类似于三个代表，科学发展观等等之类的宏观思想）和数据分析是不一样的两个概念，但却相互依存。数据分析方法论是帮助我们建立一个分析框架，比如分析什么，先分析什么，哪些内容，哪些方面，各方面都有哪些指标。至于如何分析，用什么方法，那不是方法论的事情，那是数据分析的任务。说白了，方法论是数据分析的前期规划。

下面我列举一下这些分析方法，大家课下自己找一找。好好的看一下。如果大家愿意，改天我们上课时时每个人都可以在课堂上做一个简单的分享。

[PEST](file:///C:\Users\Administrator\Documents\tencent%20files\936233993\filerecv\补充资料课件\PEST.pdf)方法

[5W2H](file:///C:\Users\Administrator\Documents\tencent%20files\936233993\filerecv\补充资料课件\5W2H.pdf)方法

[逻辑树](file:///C:\Users\Administrator\Documents\tencent%20files\936233993\filerecv\补充资料课件\逻辑树方法.pdf)方法

[4P营销理论](file:///C:\Users\Administrator\Documents\tencent%20files\936233993\filerecv\补充资料课件\4P理论.pdf)方法

[用户行为理论](file:///C:\Users\Administrator\Documents\tencent%20files\936233993\filerecv\补充资料课件\客户行为理论.pdf)方法

**数据清洗**

**数据加工**

数据转换

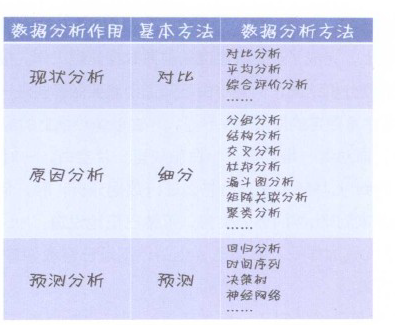
数据转换一般指的是数据表中的行列转换



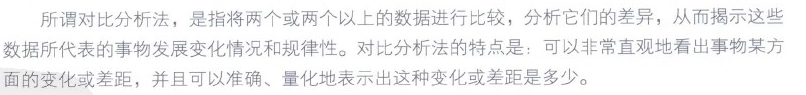


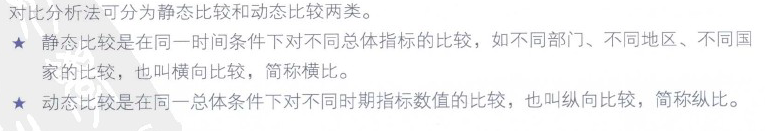
**数据抽样（随机抽样，随机函数抽样）**

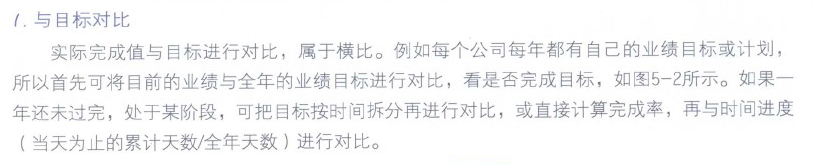
**数据分析方法**

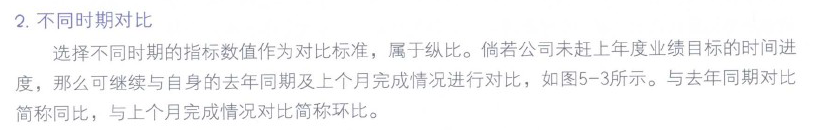


# 比较分析法：

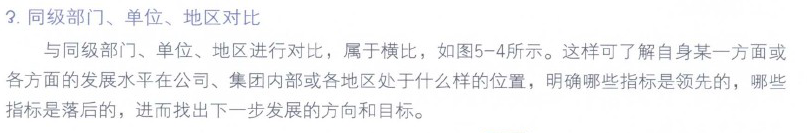






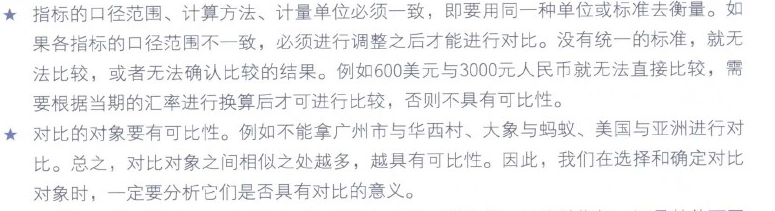










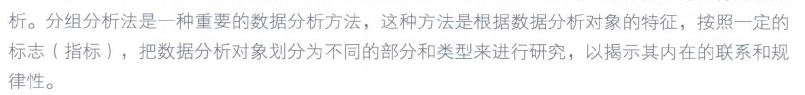






分组分析法：

分组分析，顾名思义就是分组，数据分析不仅要对总体的数量特征和数量关系进行分析，而且要深入到总体内部进行分组分析。



分组的目的为了对比，把总体中不同性质的对象区分开，把性质相同的对象合并在一起，保持组内对象性质一致，各组间对象性质有差异。（选品和潜在客户群确定）

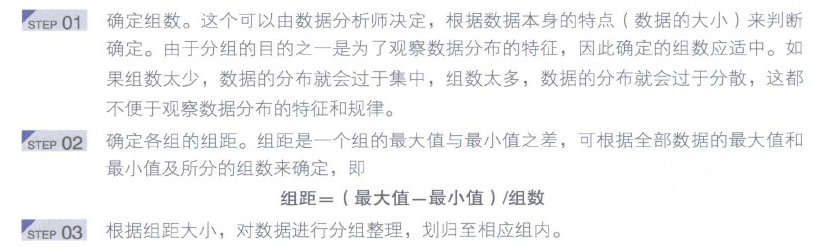








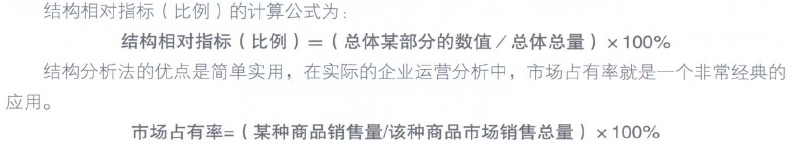
方法步骤：











# 

# 

就是一维表格转化为二维表格

[表格例子：](file:///C:\Users\Administrator\Documents\tencent%20files\936233993\filerecv\补充资料课件\交叉分析法（透视表工具）.pdf)

综合评价法

随着数据分析的广泛和深入的使用，我们遇到的问题越来越复杂，单靠对单一指标的分析越来越不能解决多指标问题。





一共分为5个步骤：



综合评价法的注意事项：

1. 综合指标体系里面的各个指标不是让我们一个接个去评定，而是要同时完成。这种要求是为了消除互为相关的指标带来的错误评价结果。

例如：某公司销售额是其他公司好几倍（销售额指标），但是他销售额与上一年同比下降了一半，（增减幅度指标）且推广投入与上一年相比也增加了好几倍（推广投入指标）。这种情况下，如果我们逐次评价这些指标，而不是同时结合其他关联指标进行同时评价，那么我们很可能会得出极其错误的结论。综合评定第一要义，弄清各指标关系和意义，同时评价。而不能评价那个指标就只盯着那个指标来看。

2. 重要指标不要忘记加权。

3. 结果不再是统计指标，仅仅是对评价对象的评分或排名。

4. 对各指标下的数据要根据实际情况采取数据标准化。

下面我们来谈一谈如何进行数据标准化和计算权重。

数据标准化三要素：统单位，同量级，归一化（把数据转换到0到1的一个数）。

这里着重介绍归一化。

ae

×

a1

×

×

a0

假设a0和ae是起点和终点，a1是线上任意一点，求a1一元表达式：

IMG_256

假定a1是给定待归一化的已知数据，那么t就是与数据a1相对应的归一化后的数据：

IMG_256

在数据分析中，a0和ae分别代表着一组原始数据中的最小值和最大值，a1 为原始数据中任意一个数值。

目的使数据一目了然，便于比较。另外更容易与高斯分布作比较（解释！！）

权重的确定：

方法很多，但很多很难操作，这里推荐一种目标优化矩阵表。（简单，而且十分准确）举例说明：

比如我们给某些人的综合能力做权重分析（以后打算做猎头分析师的兄弟姐妹要注意了！！这个方法对你们以后的工作很有用额！！）。那么，该如何做呢？我们这里给出一个例子，见下图：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| **权重评价** | **人品** | **团队能力** | **创新意思** | **专业水平** | **学历** | **合计** | **排序** |
| **人品** |  |  |  |  |  |  |  |
| **团队能力** |  |  |  |  |  |  |  |
| **创新意思** |  |  |  |  |  |  |  |
| **专业水平** |  |  |  |  |  |  |  |
| **学历** |  |  |  |  |  |  |  |
| 填表要求： | 请填表人把纵轴上的每一项指标依次与横轴比较，如果你认为纵轴指标比横轴指标重要， 请在其纵横交叉位置填1，否则填0.注：涂灰单元格不用填写。 | | | | | | |
| **表格的填写要找那些行业或公司大佬们去填写，因为他们工作经验告诉他们那些才是重要的。** | | | | | | | |
|
|  | | | | | | | |
|
|
|
|
| 下面是一个大佬的填写结果（仅仅是我自己编的例子，不要当真额！） |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| **权重评价** | **人品** | **团队能力** | **创新意思** | **专业水平** | **学历** | **合计** | **排序** |
| **人品** |  | 1 | 1 | 1 | 0 | 3 | 2 |
| **团队能力** | 0 |  | 1 | 1 | 0 | 2 | 3 |
| **创新意思** | 1 | 0 |  | 0 | 1 | 2 | 3 |
| **专业水平** | 0 | 1 | 0 |  | 0 | 0 | 4 |
| **学历** | 1 | 1 | 1 | 1 |  | 4 | 1 |
|  |  |  |  |  | sum： | 11 |  |
|  |  |  |  |  |  |  |  |
| 计算权重 | 因为专业水平得分为0，实际操作中我们往往会加1，一钱不值的东西在这世界上真的不多额！ 让合计中每一个数值分别除以所有指标总得分11，就可以得到每项指标的权重。 | | | | | | |
|  |  |  | **权重分布** | **大佬1** |  |  |  |
|  |  |  | **人品** | 0.272727273 |  |  |  |
|  |  |  | **团队能力** | 0.181818182 |  |  |  |
|  |  |  | **创新意思** | 0.181818182 |  |  |  |
|  |  |  | **专业水平** | 0.090909091 |  |  |  |
|  |  |  | **学历** | 0.363636364 |  |  |  |
|  | | | | | | | |
|
|
|
|
| 这仅仅是一个大佬的，把其它大佬的权重分布收集到一起，然后依次根据各项指标算平均值，得到最终权重分布。举例说明。下面是两位大佬的权重分布图 | | | | | | | |
|  |  |  |  |  |  |  |  |
|  |  |  | **权重分布** | **大佬1** | **大佬2** | **平均值** |  |
|  |  |  | **人品** | 0.272727273 | 0.230769231 | 0.251748252 |  |
|  |  |  | **团队能力** | 0.181818182 | 0.307692308 | 0.244755245 |  |
|  |  |  | **创新意思** | 0.181818182 | 0.153846154 | 0.167832168 |  |
|  |  |  | **专业水平** | 0.090909091 | 0.076923077 | 0.083916084 |  |
|  |  |  | **学历** | 0.363636364 | 0.307692308 | 0.335664336 |  |
|  | | | | | | | |
|
|
|
|
| 下面我们把权重转化为大于1的数，这样与我们的统计习惯比较统一。方法：找出最小值，每个最终值处于最小值 | | | | | | | |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  | **权重分布** | **最终值** | **习惯权重值** |  |  |
|  |  |  | **人品** | 0.251748252 | 3.00 |  |  |
|  |  |  | **团队能力** | 0.244755245 | 2.92 |  |  |
|  |  |  | **创新意思** | 0.167832168 | 2.00 |  |  |
|  |  |  | **专业水平** | 0.083916084 | 1.00 |  |  |
|  |  |  | **学历** | 0.335664336 | 4.00 |  |  |
|  |  |  | 最小值 | 0.083916084 |  |  |  |

杜邦分析法：

金字塔结构层层分析法，各层存在明确的因果关系，更多时候是严格的函数关系。

个人认为，杜邦分析法应用范围尽管比较单一，主要应用财务分析上。但是杜邦分析法的有现求源，层层分析的金字塔结构使数据分析变得层次清晰明了，思路简单工整（不乱发叉）并且整个分析过程十分严谨。

举例说明：查找A公司市场占有率下降的原因

+

+

+

+

+

÷

分析流程：

已知条件：某产品市场占有率下降，分析目的：为啥？？？

1. 首先我们要明白市场占有率如何计算的。它是(产品月销售额/产品在整个市场上月销售总额）。因此在市场占有率下面可分解成两个部分。
2. 然后产品月市场销售总额=市场上生产该产品的所有公司的月销售额之和。因此市场销售总额下可分解成三个部分，如果除了A公司，只有B公司和C公司生产该产品。
3. 把A公司和B公司的销售额继续分解，分解到各个平台。
4. 从中我们得出以下结论。

(1) 尽管A公司销售额增加，然而市场占有率却下降，说明分母市场销售

总额增长的幅度更大。

(2) 而市场销售总额又是三家公司共同带动的。然而造成A公司市场占有率直接下降的原因确实由于B公司和C公司共同的销售总额增幅大于A公司增幅。原因见下面等式（假定A公司月销售额为a、市场占有率为Va，B公司和C公司月销售额分别是b和c）：

gif

（b+c）增加，a也增加，但是b+c增加的更多，造成(b+c)/a的值增大，

从而导致A公司市场占有率下降。

(3) 到目前为止我们得出B和C公司销售额急剧上升造成了A公司市场占有率下降。

(4) 那么到底什么原因造成A公司销售额增速减缓呢。一眼就可看出是京东平台销售极其不给力和B公司天猫销售额的急剧增长造成的。

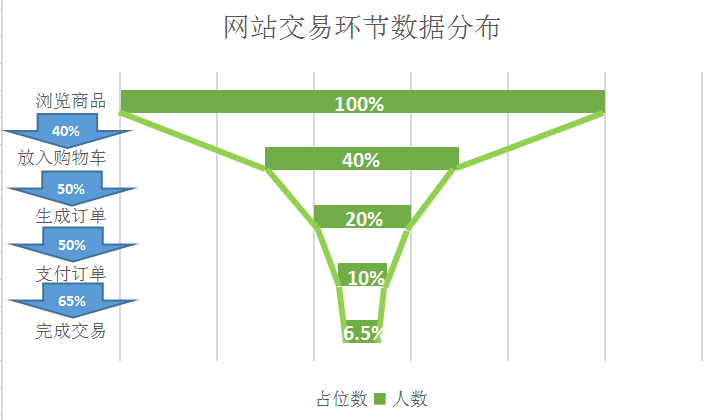
杜邦分析法就是这么牛，直接找到病根！

漏斗图分析法：

漏斗图适用于业务流程比较规范、周期长、环节多的流程分析，通过漏斗各环节业务数据的比较，能够直观地发现和说明问题所在。在网站分析中，通常用于转化率比较，它不仅能展示用户从进入网站到实现购买的最终转化率，还可以展示每个步骤的转化率。

例子：网站各环节监控数据分析：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 网站交易环节 | 占位数 | 人数 | 环节转化率 | 总体转化率=累积环节转化率=第一行环节转化率\*...\*当前环节转化率 |  |
| 浏览商品 | 0 | 1000 | 100% | 100.00% |  |
| 放入购物车 | 300 | 400 | 40% | 40.00% |  |
| 生成订单 | 400 | 200 | 50% | 20.00% |  |
| 支付订单 | 450 | 100 | 50% | 10.00% |  |
| 完成交易 | 467.5 | 65 | 65% | 6.50% |  |
|  |  |  |  |  |  |
|  | 占位数=（第一环节人数-当前环节人数）/2 |  |  |  |  |



从漏斗图我们一眼就看出各个环节的转化率以及整体转化率。然而我们仅仅是对整个网站的运营状况有了了解。至于他到底运营的如何我们根本无法做出决断，这就需要用到我们前面用到的对比法，比如对同一环节优化前后效果的对比。或同行业各环节转化率的对比等等。

矩阵关联分析法（作业）

高级分析法

回归分析，时间序列，聚类分析，神经网络等

高级分析通常是计算性数据分析（我们前面讲的是描述性数据分析），需要比较

多的数理统计和概率论知识。假如我们时间足够的话，可以挑一些比较有用的分

析方法给大家讲一下。即使我们没有时间讲，我也会把与此相关的数学知和python语法尽量告诉大家。

# 一，Numpy基础：数组

**Numerical Python**

**数组的定义**

数组是Numpy中最基本的数据对象，也是一种大容量数据容器。他的强大之处在于人们能够像操作标量那样操作数组。这样编出的代码不仅简单易于理解，而且基本告别了for语句，大大提高了运算速度。数组（Ndarray）可以是一维的，也可以是多维的，我们平常遇到的都是一维和二维数组。

例：

In [**1**]: import numpy as np

In [**2**]: data\_array=np.array([0.243,0.246,0.32,0.345,0.231])

In [**3**]: data\_array

Out[**3**]: array([ 0.243, 0.246, 0.32 , 0.345, 0.231])

In [**4**]: #array可以由列表直接产生

In [**5**]: data\_array\*3

Out[**5**]: array([ 0.729, 0.738, 0.96 , 1.035, 0.693])

In [**7**]: #可以与标量直接相乘，其计算方法与线性代数中矢量和标量相乘没有任何区别、

Ndarray必须是同构架的，也就是说，其中元素必须有相同结构。例如数值性数组每个元素都必须是数值。每个数组都有一个shape（表示各维度大小的元组）属性和一个dtype属性（一个说明数组数据类型的对象）.假如你的数值型数组出现其他数据类型，系统不会报错，但数组的数据类型不在是数值，因此不再能进行数组的数学运算。

例：（数组中突然有字符串会出现什么结果？能运算吗？）

In [**8**]: dara\_str=np.array([0.2,0.34,'am'])

In [**9**]: dara\_str.dtype

Out[**9**]: dtype('<U32')

In [**10**]: dara\_str\*2

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

TypeError Traceback (most recent call last)

<ipython-input-10-4683742c97bc> in <module>()

----> 1 dara\_str\*2

TypeError: ufunc 'multiply' did not contain a loop with signature matching types dtype('<U32') dtype('<U32') dtype('<U32')#ufunc 'multiply’不包含使用与类型dtype('<U32')相匹配的标签的循环，（也就是这个函数不含有操作dtype('<U32')类型的循环）

**数组的创建**

除了上节讲到的用简单列表创建一维数组外，还可以嵌套列表创造多维列表：

In [**1**]: building\_array=[[1,2,3],[4,2,1.2],[2.3,3.2,1.6]]

In [**3**]: array\_build=np.array(building\_array)

In [**4**]: array\_build

Out[**4**]:

array([[ 1. , 2. , 3. ],

[ 4. , 2. , 1.2],

[ 2.3, 3.2, 1.6]])

In [**5**]: #上面创建的是二维数组

In [**6**]: #下面我们创建一个3维数组

In [**9**]: buildarray\_1=[[[2.4,1.2,9],[5.6,32,7],[2,1,2]],[[7.6,3.2,6.7],[1,0,0],[1,1,2]],[[3,3,3],[2,1,2],[4,7,65]]]

In [**12**]: arrat\_out=np.array(buildarray\_1)

In [**13**]: arrat\_out

Out[**13**]:

array([[[ 2.4, 1.2, 9. ],

[ 5.6, 32. , 7. ],

[ 2. , 1. , 2. ]],

[[ 7.6, 3.2, 6.7],

[ 1. , 0. , 0. ],

[ 1. , 1. , 2. ]],

[[ 3. , 3. , 3. ],

[ 2. , 1. , 2. ],

[ 4. , 7. , 65. ]]])

In [**14**]: arrat\_out.dtype

Out[**14**]: dtype('float64')

In [**16**]: arrat\_out.shape

Out[**16**]: (3, 3, 3)

**特殊二维数组**

0矩阵（二维数组）

In [**17**]: np.zeros(5)

Out[**17**]: array([ 0., 0., 0., 0., 0.])

In [**19**]: np.zeros((3,3))

Out[**19**]:

array([[ 0., 0., 0.],

[ 0., 0., 0.],

[ 0., 0., 0.]])

空矩阵（二维数组）

In [**21**]: np.empty((2,3,2))

Out[**21**]:

array([[[ 4.02472974e-312, 3.16202013e-322],

[ 0.00000000e+000, 0.00000000e+000],

[ 0.00000000e+000, 2.87822582e+180]],

[[ 1.11667979e+165, 2.00668786e-052],

[ 4.46803262e-062, 2.95834365e-032],

[ 7.69120705e+169, 4.35610475e-061]]])

空矩阵并不为空，认为空矩阵全为空值或者零值是不对的，空矩阵里面的元素都是返回的垃圾值。

Arange asarray ones one\_like identity函数：

In [**22**]: np.arange(15)

Out[**22**]: array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])

In [**23**]: #arange函数产生一维矩阵

In [**24**]: np.arange(16).reshape(2,2,2,2)

Out[**24**]:

array([[[[ 0, 1],

[ 2, 3]],

[[ 4, 5],

[ 6, 7]]],

[[[ 8, 9],

[10, 11]],

[[12, 13],

[14, 15]]]])

In [**25**]: #arange函数产生四维数组

#借助asarray把列表转化为数组

In [**26**]: list1=[3,21,2,1]

In [**27**]: build\_1=np.asarray(list1)

In [**28**]: build\_1

Out[**28**]: array([ 3, 21, 2, 1])In [**32**]: build\_2=np.asarray(tuple\_1)

In [**33**]: build\_2

Out[**33**]: array([1, 2, 3])

In [**34**]: build3=np.array(tuple\_1)

In [**35**]: build3

Out[**35**]: array([1, 2, 3])

#借助于asarray把元组转化为数组

In [**29**]: tuple\_1=1,2,3

In [**32**]: build\_2=np.asarray(tuple\_1)

In [**33**]: build\_2

Out[**33**]: array([1, 2, 3])

#借助于array把元组转化为数组

In [**34**]: build3=np.array(tuple\_1)

In [**35**]: build3

Out[**35**]: array([1, 2, 3])

In [**36**]: np.ones((3,4))

Out[**36**]:

array([[ 1., 1., 1., 1.],

[ 1., 1., 1., 1.],

[ 1., 1., 1., 1.]])

In [**37**]: np.ones\_like(build3)

Out[**37**]: array([1, 1, 1])

In [**41**]: np.zeros\_like(build\_2)

Out[**41**]: array([0, 0, 0])

In [**44**]: np.identity(4)

Out[**44**]:

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

[ 0., 1., 0., 0.],

[ 0., 0., 1., 0.],

[ 0., 0., 0., 1.]])

**Empty\_like函数**

In [**3**]: import numpy as np

In [**4**]: arrayrr=np.array([1,3,2])

In [**5**]: test\_array1=np.empty\_like(arrayrr)

In [**6**]: test\_array1

Out[**6**]: array([6488174, 7077985, 7340146])

**多维数组的数据类型**

Dtype是一种特殊的对象，通过它的帮助ndarray能把一块内存解释为自己所需的数据类型来存放自己承载的数据，这个数据的类型就是数组解释的类型

In [8]: array123=np.array([1.4,6.7,6.6],dtype=np.float64)#指定数据类型的数组

In [10]: array123

Out[10]: array([ 1.4, 6.7, 6.6])

In [11]: array\_321=np.array([1,6,**6.7**],dtype=np.int32)#如果这里设定数据类型为整形，那么，结果只能是整数数组，注意绿色标注的数组数据变化

In [12]: array\_321

Out[12]: array([1, 6, **6**])

**数据类型表**

Int8和uint8（有符号和无符号的8位整形，i1 和 u1）；

Int16和uint16（同上，i2 和 u2）

Int32和uint32（同上，i4 和 u4）

Int64和uint64（同上，i8 和 u8）

Float16（半精度浮点数，f2）

Float32（单精度浮点数，f4）

Float64（双精度浮点数，f8）

Float128（扩展精度浮点数，f16）

Complex64（c8），Complex128（c16），

Complex256（c32）

Bool（？），object（O），string\_（S），Unicode（U）

**数组数据类型转换**

**Astype**

In [**13**]: test\_array=np.array([1,2,3,6,1])

In [**14**]: test\_array

Out[**14**]: array([1, 2, 3, 6, 1])

In [**15**]: test\_array.dtype

Out[**15**]: dtype('int32')

In [**16**]: test\_array\_1=test\_array.astype(np.float64)

In [**18**]: test\_array\_1.dtype

Out[**18**]: dtype('float64')

Astype方法可以将数值型字符串数组直接转化数值型数组。

In [**19**]: numeric\_strings=np.array(['12','21','33'],dtype=np.string\_)

In [**21**]: numeric\_strings.astype(np.float32)

Out[**21**]: array([ 12., 21., 33.], dtype=float32)

**数组的简单运算**

数组很重要，它能够使你不用编写循环语句就可以对数据执行批量操作，这一优点很重要。因此，在数据分析方面，python和R语言地位是其它语言无法替代的。

不用循环语句，直接对数据处理成可以进行批量操作的过程叫做数据矢量化。

In [**23**]: arrt\_1=np.array([[1,6,3,10],[11,9,2,4]])

In [**24**]: arrt\_1\*\*2

Out[**24**]:

array([[ 1, 36, 9, 100],

[121, 81, 4, 16]], dtype=int32)

In [**25**]: arrt\_1\*arrt\_1

Out[**25**]:

array([[ 1, 36, 9, 100],

[121, 81, 4, 16]])

In [**26**]: arrt\_2=np.array([[1,2,3,4],[2,1,4,3]])

In [**27**]: arrt\_1+arrt\_2

Out[**27**]:

array([[ 2, 8, 6, 14],

[13, 10, 6, 7]])

In [**28**]: 1/arrt\_2

Out[**28**]:

array([[ 1. , 0.5 , 0.33333333, 0.25 ],

[ 0.5 , 1. , 0.25 , 0.33333333]])

**数组的索引和切片**

**基本索引和切片**

In [**29**]: test\_array\_1=np.arange(10)

In [**30**]: test\_array\_1

Out[**30**]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [**31**]: test\_array\_1[5]

Out[**31**]: 5

In [**32**]: test\_array\_1[1:5]

Out[**32**]: array([1, 2, 3, 4])

In [**33**]: #上面的数组切片可以被赋值

In [**34**]: #列表赋值：

In [**35**]: test\_array\_1[1:5]=[6,6,6,6]

In [**36**]: test\_array\_1

Out[**36**]: array([0, 6, 6, 6, 6, 5, 6, 7, 8, 9])

In [**37**]: #数组赋值

In [**38**]: test\_array\_1[1:5]=np.array([9,9,9,9])

In [**39**]: test\_array\_1

Out[**39**]: array([0, 9, 9, 9, 9, 5, 6, 7, 8, 9])

In [**42**]: #单元素赋值

In [**43**]: test\_array\_1[1:5]=12

In [**44**]: test\_array\_1

Out[**44**]: array([ 0, 12, 12, 12, 12, 5, 6, 7, 8, 9])

In [**45**]: #一个标量的赋值或者叫输入瞬间实现整个切片选区的赋值，这就是numpy的广播功能。

In [**46**]: #比较数组切片和列表切片,我们发现列表切片没有广播功能，数组切片有。

In [**58**]: list2=[1,8,9,10,5,1,8,7.2]

In [**59**]: list2[1:6]=3

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

TypeError Traceback (most recent call last)

<ipython-input-59-51e4dda31e91> in <module>()

----> 1 list2[1:6]=3

TypeError: can only assign an iterable

In [**60**]: array2=np.arange(8)

In [**61**]: array2

Out[**61**]: array([0, 1, 2, 3, 4, 5, 6, 7])

In [**62**]: array2[0:6]=6

In [**63**]: array2

Out[**63**]: array([6, 6, 6, 6, 6, 6, 6, 7])

列表切片和数组切片还有另外一个区别:

跟列表切片最大的区别是数组切片是原数组的的视图，这就意味着原数组数据不会被复制保留，视图上任何改变都会影响的原数组。（注：切片只有真正的切下来才叫切片，也就是得有这有这样的语句：Transport\_array=A[a:b]。更确切的讲把选区A[a:b] 赋给一个变量Transport\_array时，切片才产生，Transport\_array就是一个切片。）

**原数组**

**原数组视图=原数组切片**

例子：我们先看数组切片如何影响原数组的：

In [**64**]: array\_11=np.arange(2,10,2)

In [**65**]: array\_11

Out[**65**]: array([2, 4, 6, 8])

In [**66**]: transport\_1=array\_11[0:3]

In [**69**]: transport\_1

Out[**69**]: array([2, 4, 6])

In [**70**]: transport\_1[0:2]=100

In [**71**]: array\_11

Out[**71**]: array([100, 100, 6, 8])

然后再对比列表切片：

In [**78**]: list123=[0,1,2,3,4,5]

In [**85**]: transport\_2=list123[0:3]

In [**86**]: transport\_2=[10,10,10]

In [**87**]: list123

Out[**87**]: [0, 1, 2, 3, 4, 5]

我们发现原列表没有发生任何改变。要想改变列表，需要直接赋值即可：

In [**88**]: list123[1:3]=[0,0]

In [**89**]: list123

Out[**89**]: [0, 0, 0, 3, 4, 5]

结论，改变列表切片不会影响到原列表。

**数组的这个特性大家务必注意，熟记于胸！！**

要想改变数组这个特性，我们用下列技巧。

例子：

In [**90**]: list132=[1,3,5,7,9]

In [**91**]: array\_123=np.array(list132)

In [**92**]: array\_123

Out[**92**]: array([1, 3, 5, 7, 9])

In [**93**]: ransport\_12=array\_123[0:2].copy()

In [**94**]: ransport\_12=[100,100]

In [**95**]: array\_123

Out[**95**]: array([1, 3, 5, 7, 9])

**高维数组**

如何获取高维数组的行列和元素

1. 二维数组

二维数组的列行以及值得选取：

In [**5**]: array\_12=np.arange(9).reshape(3,3)

In [**6**]: array\_12[0]

Out[**6**]: array([0, 1, 2])

In [**7**]: array\_12[:,0]

Out[**7**]: array([0, 3, 6])

In [**9**]: array\_12[2,0]

Out[**9**]: 6

In [**10**]: array\_12[2,2]

Out[**10**]: 8

1. 三维数组

三维数组的列行以及值得选取：

In [**11**]: array\_12=np.arange(18).reshape(2,3,3)

In [**12**]: array\_12

Out[**12**]:

array([[[ 0, 1, 2],

[ 3, 4, 5],

[ 6, 7, 8]],

[[ 9, 10, 11],

[12, 13, 14],

[15, 16, 17]]])

In [**13**]: array\_12[1][0]

Out[**13**]: array([ 9, 10, 11])

In [**14**]: array\_12[1,0]

Out[**14**]: array([ 9, 10, 11])

In [**17**]: array\_12[1,:,1]

Out[**17**]: array([10, 13, 16])

In [**18**]: array\_12[1][:,1]

Out[**18**]: array([10, 13, 16])

In [**19**]: array\_12[0,1,2]

Out[**19**]: 5

在前面我们说了改变切片会影响到原数据，同样改变源数据也可以影响到切片。

见下例：

In [**20**]: array\_12[1]

Out[**20**]:

array([[ 9, 10, 11],

[12, 13, 14],

[15, 16, 17]])

In [**21**]: values\_1=array\_12[1]

In [**22**]: array\_12[1]=7

In [**23**]: array\_12

Out[**23**]:

array([[[0, 1, 2],

[3, 4, 5],

[6, 7, 8]],

[[7, 7, 7],

[7, 7, 7],

[7, 7, 7]]])

In [**24**]: array\_12[1]=values\_1

In [**25**]: array\_12

Out[**25**]:

array([[[0, 1, 2],

[3, 4, 5],

[6, 7, 8]],

[[7, 7, 7],

[7, 7, 7],

[7, 7, 7]]])

In [**26**]: values\_1

Out[**26**]:

array([[7, 7, 7],

[7, 7, 7],

[7, 7, 7]])

通过.copy() 可以消除这些影响

In [**28**]: array\_12=np.arange(18).reshape(2,3,3)

In [**29**]: values\_1=array\_12[1].copy()

In [**30**]: array\_12[1]=7

In [**31**]: array\_12

Out[**31**]:

array([[[0, 1, 2],

[3, 4, 5],

[6, 7, 8]],

[[7, 7, 7],

[7, 7, 7],

[7, 7, 7]]])

In [**32**]: array\_12[1]=values\_1

In [**33**]: array\_12

Out[**33**]:

array([[[ 0, 1, 2],

[ 3, 4, 5],

[ 6, 7, 8]],

[[ 9, 10, 11],

[12, 13, 14],

[15, 16, 17]]])

In [**34**]: values\_1

Out[**34**]:

array([[ 9, 10, 11],

[12, 13, 14],

[15, 16, 17]])

**二维数组的复杂切片**

In [**37**]: ary1=np.arange(16).reshape(4,4)

In [**39**]: ary1

Out[**39**]:

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11],

[12, 13, 14, 15]])

In [**40**]: ary1[:2]

Out[**40**]:

array([[0, 1, 2, 3],

[4, 5, 6, 7]])

In [**41**]: ary1[:2,1:]

Out[**41**]:

array([[1, 2, 3],

[5, 6, 7]])

In [**44**]: ary1[1,:2]

Out[**44**]: array([4, 5])

In [**45**]: ary1[2,:1]

Out[**45**]: array([8])

In [**46**]: ary1[:,1]

Out[**46**]: array([ 1, 5, 9, 13])

In [**47**]: ary1[:,:1]

Out[**47**]:

array([[ 0],

[ 4],

[ 8],

[12]])

In [**48**]: ary1[:2,1:]=0

In [**49**]: ary1

Out[**49**]:

array([[ 0, 0, 0, 0],

[ 4, 0, 0, 0],

[ 8, 9, 10, 11],

[12, 13, 14, 15]])

例子（练习）：

要求：必须一次性选出。不能分两次选

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | 1 | 2 | 3 | | 4 | 5 | 6 | | 7 | 8 | 9 | | In [**1**]: import numpy as np  In [**3**]: array\_test=np.arange(1,10,1).reshape(3,3)  In [**4**]: array\_test  Out[**4**]:  array([[1, 2, 3],  [4, 5, 6],  [7, 8, 9]])  In [**5**]: array\_test[:,:2]  Out[**5**]:  array([[1, 2],  [4, 5],  [7, 8]]) |
| |  |  |  | | --- | --- | --- | | 1 | 2 | 3 | | 4 | 5 | 6 | | 7 | 8 | 9 | | In [**18**]: arrayselection\_1=[]  In [**19**]: for i,j in ((0,0),(2,2)):      ...: arrayselection\_1.append(array\_test[i,j])      ...:  In [**20**]: arrayselection\_1  Out[**20**]: [1, 9]  In [**22**]: arrte=np.array(arrayselection\_1)  In [**23**]: arrte  Out[**23**]: array([1, 9]) |
| |  |  |  | | --- | --- | --- | | 1 | 2 | 3 | | 4 | 5 | 6 | | 7 | 8 | 9 | | In [21]: array\_test[1]  Out[21]: array([4, 5, 6]) |
| |  |  |  | | --- | --- | --- | | 1 | 2 | 3 | | 4 | 5 | 6 | | 7 | 8 | 9 | | In [24]: array\_test[2,1:]  Out[24]: array([8, 9]) |
| |  |  |  | | --- | --- | --- | | 1 | 2 | 3 | | 4 | 5 | 6 | | 7 | 8 | 9 | | In [**25**]: arrayselection\_2=[]  In [**26**]: for i,j in ((2,0),(1,1)):      ...: arrayselection\_2.append(array\_test[i,j])      ...:  In [**27**]: arrayselection\_2  Out[**27**]: [7, 5]  In [**28**]: array\_3=np.array(arrayselection\_2)  In [**29**]: array\_3  Out[**29**]: array([7, 5]) |
| |  |  |  | | --- | --- | --- | | 1 | 2 | 3 | | 4 | 5 | 6 | | 7 | 8 | 9 | | In [30]: array\_test[0:2,1]  Out[30]: array([2, 5]) |
| |  |  |  | | --- | --- | --- | | 1 | 2 | 3 | | 4 | 5 | 6 | | 7 | 8 | 9 | | In [**31**]: arrayselection\_3=[]  In [**32**]: for i,j in ((0,0),(1,1),(2,2)):      ...: arrayselection\_3.append(array\_test[i,j])  In [**33**]: arrayselection\_3  Out[**33**]: [1, 5, 9]  In [**34**]: arry\_22=np.array(arrayselection\_3)  In [**35**]: arry\_22  Out[**35**]: array([1, 5, 9]) |

**布尔型索引**

In [**36**]: #下面我们通过一个例子来讲解布尔索引

In [**38**]: Score=np.array([[60,65,30,76],[56,78,98,34],[55,63,76,34],[78,90,45,66],[55,89,99,70],[55,45,89,77]])

In [**39**]: Name=np.array(['Mr.Lin','Miss Fang','Mr.Tian','Miss Fang','Mr.Lin','Mr.Tian'])

In [**40**]: Name

Out[**40**]:

array(['Mr.Lin', 'Miss Fang', 'Mr.Tian', 'Miss Fang', 'Mr.Lin', 'Mr.Tian'],

dtype='<U9')

In [**41**]: Score

Out[**41**]:

array([[60, 65, 30, 76],

[56, 78, 98, 34],

[55, 63, 76, 34],

[78, 90, 45, 66],

[55, 89, 99, 70],

[55, 45, 89, 77]])

In [**42**]: Name=='Mr.Lin'

Out[**42**]: array([ True, False, False, False, True, False], dtype=bool)

In [**43**]: #瞬间转化为bool型数值

In [**44**]: Score[Name=='Mr.Lin'] #这里的角码是一个布尔型列表，程序自动检索里面的bool值，自动选取“Ture”布尔值。然后把“True”布尔值转化为行号值，按行号选取数组行。

Out[**44**]:

array([[60, 65, 30, 76],

[55, 89, 99, 70]])

In [**45**]: #这就是一个简单Python数据库查询程序，可以说比任何程序的代码都简单。

注：要想用bool型数组从数值性矩阵选取数据，Python对这个做角码（索引序列）的bool型数组有以下要求：

1. 必须是bool型数组，不能是其它任何数据类型，比如列表就不行
2. 做索引序列的数组中的每个元素都要与数值数组中的行相对应。比如索引数组有六个元素，被索引数组应该有六行。
3. Bool型数组每一次做索引序列索引数值数组，必须把整个序列作为索引角码。不能只选部分。也就是说bool型索引数组不可拆。
4. Python按照bool值索引数组中的“True”分布来索引行：

例如

索引数组:[False, False, True, False, True, False],python会按照“True”分布选取3行和5行数据：array([55, 63, 76, 34],[55, 89, 99, 70], dtype=bool)

**活用Bool型索引数组**

例如：

In [**52**]: #Name！='Miss Fang'等价于~（Name=='Miss Fang）

In [**55**]: Score[~(Name=='Miss Fang')]

Out[**55**]:

array([[60, 65, 30, 76],

[55, 63, 76, 34],

[55, 89, 99, 70],

[55, 45, 89, 77]])

In [**56**]: Score[Name!='Miss Fang']

Out[**56**]:

array([[60, 65, 30, 76],

[55, 63, 76, 34],

[55, 89, 99, 70],

[55, 45, 89, 77]])

In [**57**]: #利用布尔操作符操作角码bool数组

In [**58**]: suoyin=(Name=='Miss Fang')|(Name=='Mr.Lin')

In [**59**]: suoyin

Out[**59**]: array([ True, True, False, True, True, False], dtype=bool)

In [**60**]: Score[suoyin]

Out[**60**]:

array([[60, 65, 30, 76],

[56, 78, 98, 34],

[78, 90, 45, 66],

[55, 89, 99, 70]])

In [**62**]: suoyin\_1=(Name=='Miss Fang')&(Name=='Mr.Lin')

In [**63**]: suoyin\_1

Out[**63**]: array([False, False, False, False, False, False], dtype=bool)

In [**64**]: Score[suoyin\_1]

Out[**64**]: array([], shape=(0, 4), dtype=int32)

In [**65**]: suoyin\_2=(Name=='Miss Fang')^(Name=='Mr.Lin')

In [**66**]: suoyin\_2

Out[**66**]: array([ True, True, False, True, True, False], dtype=bool)

In [**67**]: Score[suoyin\_2]

Out[**67**]:

array([[60, 65, 30, 76],

[56, 78, 98, 34],

[78, 90, 45, 66],

[55, 89, 99, 70]])

Bool索引数组与切片与整数的混合使用：

In [**68**]: Score[Name=='Mr.Lin',:3]

Out[**68**]:

array([[60, 65, 30],

[55, 89, 99]])

In [**69**]: #这个bool值不能出现在列（axis=1）

In [**71**]: Score[Name=='Mr.Lin',2]

Out[**71**]: array([30, 99])

**伟大的Python，神秘的（借助bool索引数组）赋值**

In [**72**]: #通过bool型索引给数组赋值最彰显Python优势

In [**73**]: operator\_111=(Name=='Mr.Lin')|(Name=='Miss Fang')|(Name=='Mr.Tian')

In [**74**]: Score[operator\_111,0]

Out[**74**]: array([60, 56, 55, 78, 55, 55])

In [**75**]: Score[operator\_111,0]=0

In [**76**]: Score

Out[**76**]:

array([[ 0, 65, 30, 76],

[ 0, 78, 98, 34],

[ 0, 63, 76, 34],

[ 0, 90, 45, 66],

[ 0, 89, 99, 70],

[ 0, 45, 89, 77]])

In [**77**]: array\_231=np.arange(9).reshape(3,3)

In [**79**]: array\_231

Out[**79**]:

array([[0, 1, 2],

[3, 4, 5],

[6, 7, 8]])

In [**92**]: (array\_231<=7)&(array\_231>=2)

Out[**92**]:

array([[False, False, True],

[ True, True, True],

[ True, True, False]], dtype=bool)

In [**93**]: array\_231[(array\_231<=7)&(array\_231>=2)]

Out[**93**]: array([2, 3, 4, 5, 6, 7])

In [**94**]: array\_231[(array\_231<=7)&(array\_231>=2)]=0

In [**95**]: array\_231

Out[**95**]:

array([[0, 1, 0],

[0, 0, 0],

[0, 0, 8]])

**花式索引**

In [**98**]: array\_168=np.arange(16).reshape(4,4)

In [**99**]: array\_168

Out[**99**]:

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11],

[12, 13, 14, 15]])

In [**100**]: array\_168[[1,3]]

Out[**100**]:

array([[ 4, 5, 6, 7],

[12, 13, 14, 15]])

In [**101**]: array\_168[[-1,-2]]

Out[**101**]:

array([[12, 13, 14, 15],

[ 8, 9, 10, 11]])

**花式索引可以任选多行，行与行之间的行号不用连贯。**

asss=np.arange(200).reshape(20,10)

asss

Out[34]:

array([[ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9],

[ 10, 11, 12, 13, 14, 15, 16, 17, 18, 19],

[ 20, 21, 22, 23, 24, 25, 26, 27, 28, 29],

[ 30, 31, 32, 33, 34, 35, 36, 37, 38, 39],

[ 40, 41, 42, 43, 44, 45, 46, 47, 48, 49],

[ 50, 51, 52, 53, 54, 55, 56, 57, 58, 59],

[ 60, 61, 62, 63, 64, 65, 66, 67, 68, 69],

[ 70, 71, 72, 73, 74, 75, 76, 77, 78, 79],

[ 80, 81, 82, 83, 84, 85, 86, 87, 88, 89],

[ 90, 91, 92, 93, 94, 95, 96, 97, 98, 99],

[100, 101, 102, 103, 104, 105, 106, 107, 108, 109],

[110, 111, 112, 113, 114, 115, 116, 117, 118, 119],

[120, 121, 122, 123, 124, 125, 126, 127, 128, 129],

[130, 131, 132, 133, 134, 135, 136, 137, 138, 139],

[140, 141, 142, 143, 144, 145, 146, 147, 148, 149],

[150, 151, 152, 153, 154, 155, 156, 157, 158, 159],

[160, 161, 162, 163, 164, 165, 166, 167, 168, 169],

[170, 171, 172, 173, 174, 175, 176, 177, 178, 179],

[180, 181, 182, 183, 184, 185, 186, 187, 188, 189],

[190, 191, 192, 193, 194, 195, 196, 197, 198, 199]])

asss[[0,5,15,17,19,3]]

Out[35]:

array([[ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9],

[ 50, 51, 52, 53, 54, 55, 56, 57, 58, 59],

[150, 151, 152, 153, 154, 155, 156, 157, 158, 159],

[170, 171, 172, 173, 174, 175, 176, 177, 178, 179],

[190, 191, 192, 193, 194, 195, 196, 197, 198, 199],

[ 30, 31, 32, 33, 34, 35, 36, 37, 38, 39]])

In [**104**]: array\_168[[0,1],[2,3]]

Out[**104**]: array([2, 7])

In [**105**]: #我们看到上面这个结果很可爱，也很奇怪。原因是：[[0,1],[2,3]]=》[(0,2),(1,3）]

下面这个例子需要动点脑筋，各位看官要注意额！

In [**106**]: oparray\_1=np.arange(32).reshape(8,4)

In [**107**]: oparray\_1

Out[**107**]:

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11],

[12, 13, 14, 15],

[16, 17, 18, 19],

[20, 21, 22, 23],

[24, 25, 26, 27],

[28, 29, 30, 31]])

In [**108**]: oparray\_1[[1,3,5,7]][:,[0,2,3,1]]

Out[**108**]:

array([[ 4, 6, 7, 5],

[12, 14, 15, 13],

[20, 22, 23, 21],

[28, 30, 31, 29]])

In [**109**]: #上面那个操作的索引角码可以分两个部分，前面[[1,3,5,7]]确定选取，后面部分[:,[0,2,3,1]]中的：号表示整个数组的所有行，里面的列表[0,2,3,1]表示四个列的分布

In [**110**]: oparray\_1[:,[0,2,3,1]]

Out[**110**]:

array([[ 0, 2, 3, 1],

[ 4, 6, 7, 5],

[ 8, 10, 11, 9],

[12, 14, 15, 13],

[16, 18, 19, 17],

[20, 22, 23, 21],

[24, 26, 27, 25],

[28, 30, 31, 29]])

In [**111**]: oparray\_1[:,0]

Out[**111**]: array([ 0, 4, 8, 12, 16, 20, 24, 28])

In [**112**]: #上面两个花式取列的方法

最后为大家再介绍一种比较快捷的区域索引法：

In [**113**]: oparray\_1[np.ix\_([1,3,5,7],[0,2,3,1])]

Out[**113**]:

array([[ 4, 6, 7, 5],

[12, 14, 15, 13],

[20, 22, 23, 21],

[28, 30, 31, 29]])

In [**114**]: #上面2个中括号标识的是前行后列

**列表与数组之间的转换：（练习）**

In [**11**]: list\_1=[1,2,4,5,1,9]

In [**17**]: list1=np.array(list\_1)

In [**19**]: list1.reshape(3,2)

Out[**19**]:

array([[1, 2],

[4, 5],

[1, 9]])

例子：

In [**20**]: list2=range(12)

In [**21**]: list\_array=np.array(list2)

In [**22**]: list\_array.reshape(2,3,2)

Out[**22**]:

array([[[ 0, 1],

[ 2, 3],

[ 4, 5]],

[[ 6, 7],

[ 8, 9],

[10, 11]]])

**数组转置和轴对换**

与线性代数的定义一样：

转置矩阵定义为：

CodeCogsEqn (1)

计算内积时会用到

gif（1-1）

考虑当A为实数矩阵时，CodeCogsEqn



内积的Python实现：

import numpy as np

arra=np.arange(4).reshape((2,2))

np.dot(arra.T,arra)

Out[4]:

array([[ 4, 6],

[ 6, 10]])

np.trace(np.dot(arra.T,arra))

Out[5]: 14

**高维数组转置：transpose**

例子：

arr1=np.arange(16).reshape(2,2,4)

arr1

array([[[ 0,  1,  2,  3],

        [ 4,  5,  6,  7]],  
  
       [[ 8,  9, 10, 11],  
        [12, 13, 14, 15]]])

arr2 = arr1.transpose((1,0,2))

arr2  
array([[[ 0,  1,  2,  3],  
        [ 8,  9, 10, 11]],  
  
       [[ 4,  5,  6,  7],  
        [12, 13, 14, 15]]])

为什么会这样？

这里设计到shape编号问题，这里shape是一个元组，他是有编号的，（2，2，4）的编号是（0，1，2）（也叫轴编号），现在我们要求（1，0，2）这就是书0位置和1位置互换。我们要按照新的shape 元组进行对每个矩阵元素重新排位。

比如：5在这个三维数组的位置是（0，1，1）转置后的位置是（1，0，1），也就是现在“9”的位置。

其它例子：

1. In [1]: **import** numpy as np
3. In [2]: arr=np.arange(16).reshape(2,2,4)
5. In [3]: arr
6. Out[3]:
7. array([[[ 0,  1,  2,  3],
8. [ 4,  5,  6,  7]],
10. [[ 8,  9, 10, 11],
11. [12, 13, 14, 15]]])
12. In [20]: arr.T
13. Out[20]:
14. array([[[ 0,  8],
15. [ 4, 12]],
17. [[ 1,  9],
18. [ 5, 13]],
20. [[ 2, 10],
21. [ 6, 14]],
23. [[ 3, 11],
24. [ 7, 15]]])
26. In [21]: arr.transpose(2,1,0)
27. Out[21]:
28. array([[[ 0,  8],
29. [ 4, 12]],
31. [[ 1,  9],
32. [ 5, 13]],
34. [[ 2, 10],
35. [ 6, 14]],
37. [[ 3, 11],
38. [ 7, 15]]])

这就是说高纬度数组T=（2，1，0）

Transpose 和 T Process：shape（2，2，4）--->（4，2，2）

Axes （0，1，2）-->（2，1，0）

例2：

1. In [22]: arr3=np.arange(16).reshape(2,2,2,2)
3. In [23]: arr3
4. Out[23]:
5. array([[[[ 0,  1],
6. [ 2,  3]],
8. [[ 4,  5],
9. [ 6,  7]]],

12. [[[ 8,  9],
13. [10, 11]],
15. [[12, 13],
16. [14, 15]]]])
18. In [24]: arr3.T
19. Out[24]:
20. array([[[[ 0,  8],
21. [ 4, 12]],
23. [[ 2, 10],
24. [ 6, 14]]],

27. [[[ 1,  9],
28. [ 5, 13]],
30. [[ 3, 11],
31. [ 7, 15]]]])
33. In [25]: arr3.transpose(3,2,1,0)
34. Out[25]:
35. array([[[[ 0,  8],
36. [ 4, 12]],
38. [[ 2, 10],
39. [ 6, 14]]],

42. [[[ 1,  9],
43. [ 5, 13]],
45. [[ 3, 11],
46. [ 7, 15]]]])

**两轴对换 Swapaxes**

1. In [27]: arr.swapaxes(1,2)
2. Out[27]:
3. array([[[ 0,  4],
4. [ 1,  5],
5. [ 2,  6],
6. [ 3,  7]],
8. [[ 8, 12],
9. [ 9, 13],
10. [10, 14],
11. [11, 15]]])
13. In [28]: arr.transpose(0,2,1)
14. Out[28]:
15. array([[[ 0,  4],
16. [ 1,  5],
17. [ 2,  6],
18. [ 3,  7]],
20. [[ 8, 12],
21. [ 9, 13],
22. [10, 14],
23. [11, 15]]])

Swapaxes实际上纵横两轴对换

再给大家一个比较难的例子：

1. In [29]: arr3
2. Out[29]:
3. array([[[[ 0,  1],
4. [ 2,  3]],
6. [[ 4,  5],
7. [ 6,  7]]],

10. [[[ 8,  9],
11. [10, 11]],
13. [[12, 13],
14. [14, 15]]]])
16. In [30]: arr3.swapaxes(1,3)
17. Out[30]:
18. array([[[[ 0,  4],
19. [ 2,  6]],
21. [[ 1,  5],
22. [ 3,  7]]],

25. [[[ 8, 12],
26. [10, 14]],
28. [[ 9, 13],
29. [11, 15]]]])
31. In [31]: arr3.transpose(0,3,2,1)
32. Out[31]:
33. array([[[[ 0,  4],
34. [ 2,  6]],
36. [[ 1,  5],
37. [ 3,  7]]],

40. [[[ 8, 12],
41. [10, 14]],
43. [[ 9, 13],
44. [11, 15]]]])

**这个例子的解释留给有兴趣的同学！！！费点脑子！！！**

**通用函数sqrt、exp、maximum**

**特点：对整个数组逐元素进行操作，这一点很好！！**

**我们先看下一段代码：**

import numpy as np

arr = np.arange(10)

arr

输出：

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

np.sqrt(arr)

输出：

array([ 0. , 1. , 1.41421356, 1.73205081, 2. ,

2.23606798, 2.44948974, 2.64575131, 2.82842712, 3. ])

np.exp(arr)

输出：

array([ 1.00000000e+00, 2.71828183e+00, 7.38905610e+00,

2.00855369e+01, 5.45981500e+01, 1.48413159e+02,

4.03428793e+02, 1.09663316e+03, 2.98095799e+03,

8.10308393e+03])

x = np.random.randn(8)

x

输出：

array([-0.24726724, 0.69709717, 0.9658356 , 1.89019088, -0.28912795,

-0.09235779, 0.37690775, 0.9102138 ])

y = np.random.randn(8)

y

输出：

array([-0.05048326, -0.02207697, -0.59940773, -1.32029941, 0.30894105,

-0.05807405, -1.5019804 , 0.12918562])

np.maximum(x,y)

输出：

array([-0.05048326, 0.69709717, 0.9658356 , 1.89019088, 0.30894105,

-0.05807405, 0.37690775, 0.9102138 ])

arr = np.random.randn(7)\*5

arr

输出：

array([ -1.53462646, 6.15168006, 4.32588912, -0.05408803,

-2.98953481, -10.83013834, 1.13673478])

np.modf(arr)

输出：

(array([-0.53462646, 0.15168006, 0.32588912, -0.05408803, -0.98953481,

-0.83013834, 0.13673478]),

array([ -1., 6., 4., -0., -2., -10., 1.]))

modf函数可以把数组分别提取出整数部分和小数部分

**常用1元函数**

**Abs, fabs, sqrt,square,exp,log,ceil,sign,floor,sin,sinh,cos,cosh,tan,tanh...**

**常用二元函数**

**Add，subtract,multiply,devide,floor\_devide,power,maximum,minimum,mod...**

例如：

x1 = range(6)

x1

输出：[0, 1, 2, 3, 4, 5]

np.power(x1, 3)

输出：array([ 0, 1, 8, 27, 64, 125])

x2 = [1.0, 2.0, 3.0, 3.0, 2.0, 1.0]

np.power(x1, x2)

输出：array([ 0., 1., 8., 27., 16., 5.])

x2 = np.array([[1, 2, 3, 3, 2, 1], [1, 2, 3, 3, 2, 1]])

x2

输出：array([[1, 2, 3, 3, 2, 1],

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

np.power(x1, x2)

array([[ 0, 1, 8, 27, 16, 5],

[ 0, 1, 8, 27, 16, 5]])

数组的元素分别求n次方。x1,x2可以是数字，列表，也可以是数组，但是x1和x2的列数要相同。

**利用数组处理进行数据处理的好处？**

从上面一章我们已经发现，用数组数据形式处理数据不仅形势简单，而且运算快捷。所以强大的数组运算能力正是python魅力所在。这一点上R语言与其相比略显不足。这里我们先简单了解下python是如何利用数组这种数据类型来处理数据的。

这里我再强调一下。只要所处理的数据能够矢量化（用数组表达数据并处理数据代替循环处理数据）就尽量矢量化。

例子，一个函数 的热像图。

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

In [**3**]: import numpy as np

In [**4**]: arr\_dim1=np.arange(-10,10,1)

In [**5**]: arr\_dim2=np.arange(-8,8,1)

In [**6**]: X,Y=np.meshgrid(arr\_dim1,arr\_dim2)

In [**7**]: z=np.sqrt(X\*\*2+Y\*\*2)

In [**8**]: import matplotlib.pyplot as plt

In [**9**]: X,Y,z

Out[**9**]:

(array([[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9],

[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9],

[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9],

[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9],

[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9],

[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9],

[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9],

[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9],

[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9],

[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9],

[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9],

[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9],

[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9],

[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9],

[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9],

[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2,

3, 4, 5, 6, 7, 8, 9]]),

array([[-8, -8, -8, -8, -8, -8, -8, -8, -8, -8, -8, -8, -8, -8, -8, -8, -8,

-8, -8, -8],

[-7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7,

-7, -7, -7],

[-6, -6, -6, -6, -6, -6, -6, -6, -6, -6, -6, -6, -6, -6, -6, -6, -6,

-6, -6, -6],

[-5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5,

-5, -5, -5],

[-4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,

-4, -4, -4],

[-3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3,

-3, -3, -3],

[-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2,

-2, -2, -2],

[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,

-1, -1, -1],

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0],

[ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1],

[ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2],

[ 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

3, 3, 3],

[ 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,

4, 4, 4],

[ 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,

5, 5, 5],

[ 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,

6, 6, 6],

[ 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,

7, 7, 7]]),

array([[ 12.80624847, 12.04159458, 11.3137085 , 10.63014581,

10. , 9.43398113, 8.94427191, 8.54400375,

8.24621125, 8.06225775, 8. , 8.06225775,

8.24621125, 8.54400375, 8.94427191, 9.43398113,

10. , 10.63014581, 11.3137085 , 12.04159458],

[ 12.20655562, 11.40175425, 10.63014581, 9.89949494,

9.21954446, 8.60232527, 8.06225775, 7.61577311,

7.28010989, 7.07106781, 7. , 7.07106781,

7.28010989, 7.61577311, 8.06225775, 8.60232527,

9.21954446, 9.89949494, 10.63014581, 11.40175425],

[ 11.66190379, 10.81665383, 10. , 9.21954446,

8.48528137, 7.81024968, 7.21110255, 6.70820393,

6.32455532, 6.08276253, 6. , 6.08276253,

6.32455532, 6.70820393, 7.21110255, 7.81024968,

8.48528137, 9.21954446, 10. , 10.81665383],

[ 11.18033989, 10.29563014, 9.43398113, 8.60232527,

7.81024968, 7.07106781, 6.40312424, 5.83095189,

5.38516481, 5.09901951, 5. , 5.09901951,

5.38516481, 5.83095189, 6.40312424, 7.07106781,

7.81024968, 8.60232527, 9.43398113, 10.29563014],

[ 10.77032961, 9.8488578 , 8.94427191, 8.06225775,

7.21110255, 6.40312424, 5.65685425, 5. ,

4.47213595, 4.12310563, 4. , 4.12310563,

4.47213595, 5. , 5.65685425, 6.40312424,

7.21110255, 8.06225775, 8.94427191, 9.8488578 ],

[ 10.44030651, 9.48683298, 8.54400375, 7.61577311,

6.70820393, 5.83095189, 5. , 4.24264069,

3.60555128, 3.16227766, 3. , 3.16227766,

3.60555128, 4.24264069, 5. , 5.83095189,

6.70820393, 7.61577311, 8.54400375, 9.48683298],

[ 10.19803903, 9.21954446, 8.24621125, 7.28010989,

6.32455532, 5.38516481, 4.47213595, 3.60555128,

2.82842712, 2.23606798, 2. , 2.23606798,

2.82842712, 3.60555128, 4.47213595, 5.38516481,

6.32455532, 7.28010989, 8.24621125, 9.21954446],

[ 10.04987562, 9.05538514, 8.06225775, 7.07106781,

6.08276253, 5.09901951, 4.12310563, 3.16227766,

2.23606798, 1.41421356, 1. , 1.41421356,

2.23606798, 3.16227766, 4.12310563, 5.09901951,

6.08276253, 7.07106781, 8.06225775, 9.05538514],

[ 10. , 9. , 8. , 7. ,

6. , 5. , 4. , 3. ,

2. , 1. , 0. , 1. ,

2. , 3. , 4. , 5. ,

6. , 7. , 8. , 9. ],

[ 10.04987562, 9.05538514, 8.06225775, 7.07106781,

6.08276253, 5.09901951, 4.12310563, 3.16227766,

2.23606798, 1.41421356, 1. , 1.41421356,

2.23606798, 3.16227766, 4.12310563, 5.09901951,

6.08276253, 7.07106781, 8.06225775, 9.05538514],

[ 10.19803903, 9.21954446, 8.24621125, 7.28010989,

6.32455532, 5.38516481, 4.47213595, 3.60555128,

2.82842712, 2.23606798, 2. , 2.23606798,

2.82842712, 3.60555128, 4.47213595, 5.38516481,

6.32455532, 7.28010989, 8.24621125, 9.21954446],

[ 10.44030651, 9.48683298, 8.54400375, 7.61577311,

6.70820393, 5.83095189, 5. , 4.24264069,

3.60555128, 3.16227766, 3. , 3.16227766,

3.60555128, 4.24264069, 5. , 5.83095189,

6.70820393, 7.61577311, 8.54400375, 9.48683298],

[ 10.77032961, 9.8488578 , 8.94427191, 8.06225775,

7.21110255, 6.40312424, 5.65685425, 5. ,

4.47213595, 4.12310563, 4. , 4.12310563,

4.47213595, 5. , 5.65685425, 6.40312424,

7.21110255, 8.06225775, 8.94427191, 9.8488578 ],

[ 11.18033989, 10.29563014, 9.43398113, 8.60232527,

7.81024968, 7.07106781, 6.40312424, 5.83095189,

5.38516481, 5.09901951, 5. , 5.09901951,

5.38516481, 5.83095189, 6.40312424, 7.07106781,

7.81024968, 8.60232527, 9.43398113, 10.29563014],

[ 11.66190379, 10.81665383, 10. , 9.21954446,

8.48528137, 7.81024968, 7.21110255, 6.70820393,

6.32455532, 6.08276253, 6. , 6.08276253,

6.32455532, 6.70820393, 7.21110255, 7.81024968,

8.48528137, 9.21954446, 10. , 10.81665383],

[ 12.20655562, 11.40175425, 10.63014581, 9.89949494,

9.21954446, 8.60232527, 8.06225775, 7.61577311,

7.28010989, 7.07106781, 7. , 7.07106781,

7.28010989, 7.61577311, 8.06225775, 8.60232527,

9.21954446, 9.89949494, 10.63014581, 11.40175425]]))

In [**10**]: fig = plt.figure()

In [**13**]: ax = fig.add\_subplot(221)

C:\Users\dongfeng\Anaconda3\lib\site-packages\matplotlib\cbook\deprecation.py:106: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.

warnings.warn(message, mplDeprecation, stacklevel=1)

In [**14**]: ax.imshow(z)

Out[**14**]: <matplotlib.image.AxesImage at 0x6829961ba8>

In [**15**]: ax = fig.add\_subplot(222)

C:\Users\dongfeng\Anaconda3\lib\site-packages\matplotlib\cbook\deprecation.py:106: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.

warnings.warn(message, mplDeprecation, stacklevel=1)

In [**16**]: ax.imshow(z,cmap = plt.cm.gray)

Out[**16**]: <matplotlib.image.AxesImage at 0x6829961f60>

In [**17**]: ax = fig.add\_subplot(223)

C:\Users\dongfeng\Anaconda3\lib\site-packages\matplotlib\cbook\deprecation.py:106: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.

warnings.warn(message, mplDeprecation, stacklevel=1)

In [**18**]: ax.imshow(z,cmap=plt.cm.cool)

Out[**18**]: <matplotlib.image.AxesImage at 0x682998e320>

In [**19**]: ax = fig.add\_subplot(224)

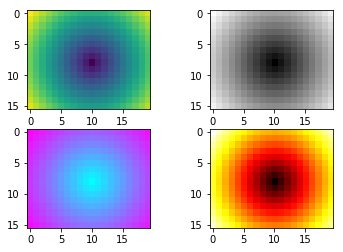
C:\Users\dongfeng\Anaconda3\lib\site-packages\matplotlib\cbook\deprecation.py:106: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.

warnings.warn(message, mplDeprecation, stacklevel=1)

In [**20**]: ax.imshow(z,cmap=plt.cm.hot)

Out[**20**]: <matplotlib.image.AxesImage at 0x682998e6a0>

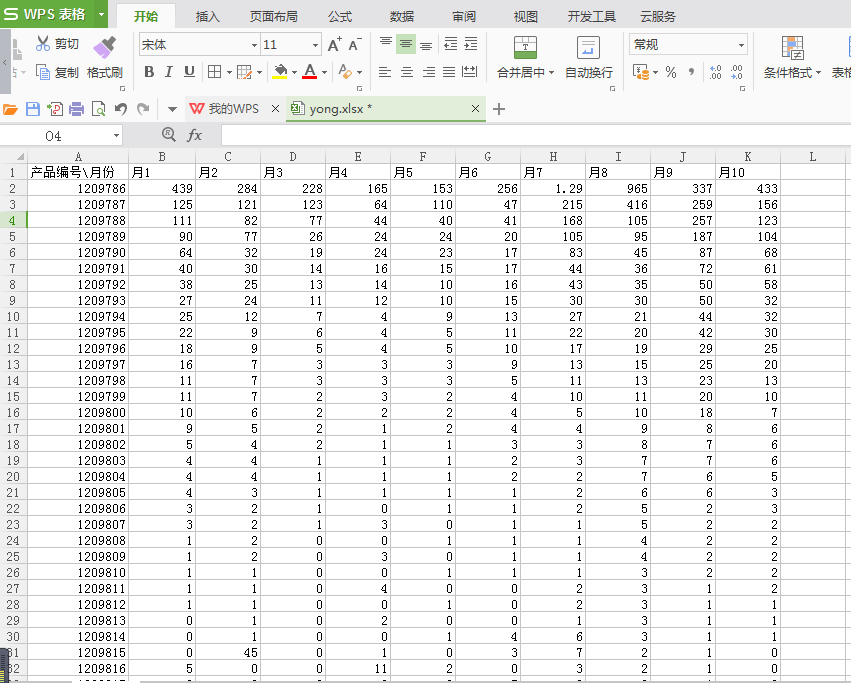
In [**23**]: plt.show()



**快速挑选数据**

**利用 np.where 函数让你快速找到你所需数据。**

**例：下面是C公司从一月到10月份个产品销售状况，出于保护该公司商业机密目的，这里隐去产品中文名称，只列举产品编号。现在我们根据np.where函数来寻找某些我们需要的销量区间，以便更好的进行分组。**



In [89]: import numpy as np

In [9**0**]: import xlrd

In [**95**]: name=r'C:\Users\dongfeng\Desktop\yong.xls'

In [**96**]: test1=xlrd.open\_workbook(name)

In [**99**]: table = test1.sheet\_by\_name(u'Sheet1')

In [**101**]: daten1=[]

In [**102**]: nrows = table.nrows

In [**104**]: for j in range(1,nrows,1):

     ...: daten1.append(list(table.row\_values(j)))

In [**105**]: daten1

Out[**105**]:

[[439.0, 284.0, 228.0, 165.0, 153.0, 256.0, 1.29, 965.0, 337.0, 433.0],

[125.0, 121.0, 123.0, 64.0, 110.0, 47.0, 215.0, 416.0, 259.0, 156.0],

[111.0, 82.0, 77.0, 44.0, 40.0, 41.0, 168.0, 105.0, 257.0, 123.0],

[90.0, 77.0, 26.0, 24.0, 24.0, 20.0, 105.0, 95.0, 187.0, 104.0],

[64.0, 32.0, 19.0, 24.0, 23.0, 17.0, 83.0, 45.0, 87.0, 68.0],

[40.0, 30.0, 14.0, 16.0, 15.0, 17.0, 44.0, 36.0, 72.0, 61.0],

[38.0, 25.0, 13.0, 14.0, 10.0, 16.0, 43.0, 35.0, 50.0, 58.0],

[27.0, 24.0, 11.0, 12.0, 10.0, 15.0, 30.0, 30.0, 50.0, 32.0],

[25.0, 12.0, 7.0, 4.0, 9.0, 13.0, 27.0, 21.0, 44.0, 32.0],

[22.0, 9.0, 6.0, 4.0, 5.0, 11.0, 22.0, 20.0, 42.0, 30.0],

[18.0, 9.0, 5.0, 4.0, 5.0, 10.0, 17.0, 19.0, 29.0, 25.0],

[16.0, 7.0, 3.0, 3.0, 3.0, 9.0, 13.0, 15.0, 25.0, 20.0],

[11.0, 7.0, 3.0, 3.0, 3.0, 5.0, 11.0, 13.0, 23.0, 13.0],

[11.0, 7.0, 2.0, 3.0, 2.0, 4.0, 10.0, 11.0, 20.0, 10.0],

[10.0, 6.0, 2.0, 2.0, 2.0, 4.0, 5.0, 10.0, 18.0, 7.0],

[9.0, 5.0, 2.0, 1.0, 2.0, 4.0, 4.0, 9.0, 8.0, 6.0],

[5.0, 4.0, 2.0, 1.0, 1.0, 3.0, 3.0, 8.0, 7.0, 6.0],

[4.0, 4.0, 1.0, 1.0, 1.0, 2.0, 3.0, 7.0, 7.0, 6.0],

[4.0, 4.0, 1.0, 1.0, 1.0, 2.0, 2.0, 7.0, 6.0, 5.0],

[4.0, 3.0, 1.0, 1.0, 1.0, 1.0, 2.0, 6.0, 6.0, 3.0],

[3.0, 2.0, 1.0, 0.0, 1.0, 1.0, 2.0, 5.0, 2.0, 3.0],

[3.0, 2.0, 1.0, 3.0, 0.0, 1.0, 1.0, 5.0, 2.0, 2.0],

[1.0, 2.0, 0.0, 0.0, 1.0, 1.0, 1.0, 4.0, 2.0, 2.0],

[1.0, 2.0, 0.0, 3.0, 0.0, 1.0, 1.0, 4.0, 2.0, 2.0],

[1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 3.0, 2.0, 2.0],

[1.0, 1.0, 0.0, 4.0, 0.0, 0.0, 2.0, 3.0, 1.0, 2.0],

[1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 2.0, 3.0, 1.0, 1.0],

[0.0, 1.0, 0.0, 2.0, 0.0, 0.0, 1.0, 3.0, 1.0, 1.0],

[0.0, 1.0, 0.0, 0.0, 1.0, 4.0, 6.0, 3.0, 1.0, 1.0],

[0.0, 45.0, 0.0, 1.0, 0.0, 3.0, 7.0, 2.0, 1.0, 0.0],

[5.0, 0.0, 0.0, 11.0, 2.0, 0.0, 3.0, 2.0, 1.0, 0.0],

[0.0, 0.0, 0.0, 0.0, 0.0, 5.0, 3.0, 2.0, 1.0, 0.0],

[0.0, 6.0, 0.0, 67.0, 6.0, 3.0, 3.0, 3.0, 1.0, 0.0],

[0.0, 0.0, 33.0, 0.0, 8.0, 0.0, 2.0, 9.0, 1.0, 0.0],

[45.0, 0.0, 0.0, 0.0, 5.0, 1.0, 2.0, 9.0, 0.0, 0.0],

[0.0, 0.0, 56.0, 54.0, 0.0, 0.0, 23.0, 3.0, 0.0, 0.0],

[0.0, 32.0, 0.0, 0.0, 5.0, 4.0, 2.0, 1.0, 0.0, 3.0],

[0.0, 0.0, 0.0, 0.0, 4.0, 6.0, 2.0, 2.0, 0.0, 0.0],

[0.0, 6.0, 0.0, 22.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0],

[0.0, 0.0, 21.0, 0.0, 0.0, 5.0, 1.0, 0.0, 0.0, 0.0],

[32.0, 0.0, 0.0, 33.0, 0.0, 0.0, 1.0, 56.0, 0.0, 0.0],

[5.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0],

[0.0, 0.0, 2.0, 1.0, 0.0, 0.0, 6.0, 0.0, 0.0, 54.0],

[3.0, 30.0, 4.0, 2.0, 0.0, 0.0, 7.0, 8.0, 1.0, 0.0],

[0.0, 0.0, 3.0, 1.0, 8.0, 4.0, 0.0, 0.0, 7.0, 0.0],

[0.0, 3.0, 0.0, 1.0, 81.0, 0.0, 0.0, 0.0, 0.0, 0.0],

[4.0, 0.0, 0.0, 0.0, 9.0, 0.0, 55.0, 2.0, 0.0, 1.0],

[0.0, 78.0, 0.0, 3.0, 3.0, 9.0, 8.0, 0.0, 0.0, 1.0],

[0.0, 0.0, 34.0, 1.0, 2.0, 0.0, 47.0, 0.0, 1.0, 0.0]]

In [**106**]: array\_ur=np.array(daten1)

In [**111**]: daten\_pro1=np.where(array\_ur>400,-5,array\_ur)

In [**113**]: daten\_pro2=np.where(daten\_pro1>300,-4,daten\_pro1)

In [**114**]: daten\_pro3=np.where(daten\_pro2>200,-3,daten\_pro2)

In [**115**]: daten\_pro4=np.where(daten\_pro3>100,-2,daten\_pro3)

In [**116**]: daten\_pro5=np.where(daten\_pro4>0,-1,daten\_pro4)

In [**117**]: daten\_pro5

Out[**117**]:

array([[-5., -3., -3., -2., -2., -3., -1., -5., -4., -5.],

[-2., -2., -2., -1., -2., -1., -3., -5., -3., -2.],

[-2., -1., -1., -1., -1., -1., -2., -2., -3., -2.],

[-1., -1., -1., -1., -1., -1., -2., -1., -2., -2.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., 0., -1., -1., -1., -1., -1., -1.],

[-1., -1., -1., -1., 0., -1., -1., -1., -1., -1.],

[-1., -1., 0., 0., -1., -1., -1., -1., -1., -1.],

[-1., -1., 0., -1., 0., -1., -1., -1., -1., -1.],

[-1., -1., 0., 0., -1., -1., -1., -1., -1., -1.],

[-1., -1., 0., -1., 0., 0., -1., -1., -1., -1.],

[-1., -1., 0., 0., -1., 0., -1., -1., -1., -1.],

[ 0., -1., 0., -1., 0., 0., -1., -1., -1., -1.],

[ 0., -1., 0., 0., -1., -1., -1., -1., -1., -1.],

[ 0., -1., 0., -1., 0., -1., -1., -1., -1., 0.],

[-1., 0., 0., -1., -1., 0., -1., -1., -1., 0.],

[ 0., 0., 0., 0., 0., -1., -1., -1., -1., 0.],

[ 0., -1., 0., -1., -1., -1., -1., -1., -1., 0.],

[ 0., 0., -1., 0., -1., 0., -1., -1., -1., 0.],

[-1., 0., 0., 0., -1., -1., -1., -1., 0., 0.],

[ 0., 0., -1., -1., 0., 0., -1., -1., 0., 0.],

[ 0., -1., 0., 0., -1., -1., -1., -1., 0., -1.],

[ 0., 0., 0., 0., -1., -1., -1., -1., 0., 0.],

[ 0., -1., 0., -1., 0., 0., -1., 0., 0., 0.],

[ 0., 0., -1., 0., 0., -1., -1., 0., 0., 0.],

[-1., 0., 0., -1., 0., 0., -1., -1., 0., 0.],

[-1., 0., 0., 0., 0., 0., -1., 0., 0., -1.],

[ 0., 0., -1., -1., 0., 0., -1., 0., 0., -1.],

[-1., -1., -1., -1., 0., 0., -1., -1., -1., 0.],

[ 0., 0., -1., -1., -1., -1., 0., 0., -1., 0.],

[ 0., -1., 0., -1., -1., 0., 0., 0., 0., 0.],

[-1., 0., 0., 0., -1., 0., -1., -1., 0., -1.],

[ 0., -1., 0., -1., -1., -1., -1., 0., 0., -1.],

[ 0., 0., -1., -1., -1., 0., -1., 0., -1., 0.]])

In [**22**]: datasou5=[]

    ...: def index\_2dim\_array\_2(arrayx,searchvalue):

    ...: indext=arrayx.shape

    ...: r=indext[0]

    ...: cl=indext[1]

    ...: for i in range(r):

    ...: for j in range(cl):

    ...: if arrayx[i,j]==searchvalue:

    ...: indexel=i,j

    ...: datasou5.append(indexel)

    ...: return datasou5

    ...:

In [**23**]: index\_2dim\_array\_2(daten\_pro5,-5)

Out[**23**]: [(0, 0), (0, 7), (0, 9), (1, 7)]

**小练习：分组的练习**

**def abc\_functional(rownr,colnr,max\_distribution,min\_distribution):**

**array\_test=np.random.randint(min\_distribution,max\_distribution,rownr\*colnr).reshape(rownr,colnr)**

**i=0**

**indirekt\_array=array\_test**

**for j in range(min\_distribution,max\_distribution+1,1):**

**indirekt\_array=np.where(indirekt\_array>=max\_distribution-i,min\_distribution-i-1,indirekt\_array)**

**i=i+1**

**return indirekt\_array**

**基本数学统计方法**

**方差和标准差（var，std）**

方差（var）：假如随机变量是离散的，则有：

（1-2）

其中



Pn是与随机变量n相关的概率。

上面的公式是方差的理论公式，实际应用中根据情况的不同会有不同的公式变种。

在统计描述中，方差用来计算每一个变量（观察值）与总体均数之间的差异。他的变量出现概率是一个定值（某些书称这种概率分布为Laplace分布）1/N，N是总体随机变量数，或叫总体例数。在此情况下，（1-2）可转化为：

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

实际工作中，由于数据量庞大，我们往往要就行抽样研究，应用样本统计This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.量代替总体参数，经校正后，样本方差计算公式：

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program. 为样本方差，X为变量,IMG_256为样本均值，N为样本例数。

方差实际意义：数学意义是变量或者说样本与样本平均值的差异，更直观点讲实际值与期望值之差平方的平均值。方差越大整体数据越离散，越不易做量化的统计分析。反之，越紧凑，利于数据的量化分析。

我们这里仅仅谈了与离散型Laplace分布相关的方差，实际上，典型离散分布还有二项式分布超几何分布、几何分布、复合二项式分布、两点分布，泊松分布等等。除了离散分布还有连续分布，典型的连续分布有均匀分布，正态分布，标正态分布，指数分布等等。这些知识与我们现在的课程关联不大。若将来大家想了解这方面的知识，我们会考虑开设慨率论与数理统计这门课程。

标差：也叫标准差。是方差的算术平方根

标差的量纲（或者说单位）与样本变量相同，因此更多设计到物理学化学问题的统计学问他往往用标差来描述样本与均值的总体差异。

其它数组统计方法：

Sum,mean,min,max,argmin,argmax(分别为最大和最小元素的索引）cumsum（所有元素的累积和）cumprod（所有元素的累积积）

例1：

In [**1**]: import numpy as np

In [**3**]: testarr1=np.random.randn(3,3)

In [**4**]: testarr2=testarr1\*7

In [**5**]: testarr2

Out[**5**]:

array([[ 0.14798325, -4.95454558, -3.70303278],

[ -2.27484428, 10.02286341, -12.82076134],

[ -0.30437921, 15.59126043, -3.80802942]])

例2：

In [**6**]: np.cosh(testarr2)

Out[**6**]:

array([[ 1.01096952e+00, 7.09125991e+01, 2.02974036e+01],

[ 4.91460863e+00, 1.12679335e+04, 1.84908088e+05],

[ 1.04668210e+00, 2.95235362e+06, 2.25418729e+01]])

In [**7**]: testarr3=np.cosh(testarr2)\*3+np.sinh(testarr2)\*6

In [**8**]: testarr3

Out[**8**]:

array([[ 3.92405233e+00, -2.12695490e+02, -6.07443188e+01],

[ -1.41269484e+01, 1.01411401e+05, -5.54724265e+05],

[ 1.28544039e+00, 2.65711826e+07, -6.74924674e+01]])

In [**9**]: testarr3=np.floor(testarr3)

In [**10**]: testarr3

Out[**10**]:

array([[ 3.00000000e+00, -2.13000000e+02, -6.10000000e+01],

[ -1.50000000e+01, 1.01411000e+05, -5.54725000e+05],

[ 1.00000000e+00, 2.65711820e+07, -6.80000000e+01]])

例3：

In [**11**]: testarr3.mean()

Out[**11**]: 2901946.111111111

In [**35**]: np.mean(testarr3)

Out[**35**]: 2901946.111111111

In [**12**]: np.sum(testarr3)

Out[**12**]: 26117515.0

例4：

In [**13**]: jar=np.arange(9).reshape(3,3)

In [**14**]: jar

Out[**14**]:

array([[0, 1, 2],

[3, 4, 5],

[6, 7, 8]])

In [**15**]: sum(jar)

Out[**15**]: array([ 9, 12, 15])

In [**16**]: np.sum(jar)

Out[**16**]: 36

In [**17**]: np.mean(jar)

Out[**17**]: 4.0

例5

In [**18**]: jar.mean(axis=1)

Out[**18**]: array([ 1., 4., 7.])

In [**19**]: jar.mean(1,)

Out[**19**]: array([ 1., 4., 7.])

In [**29**]: jar.mean(1)

Out[**29**]: array([ 1., 4., 7.])

#横向相加求均值

In [**28**]: jar.mean(0)

Out[**28**]: array([ 3., 4., 5.])

#纵向相加求均值

例6

In [**32**]: jar.sum(0)

Out[**32**]: array([ 9, 12, 15])

#纵向相加

In [**33**]: jar.sum(1)

Out[**33**]: array([ 3, 12, 21])

#横向相加

例7：

In [**36**]: jar.cumsum(0)#纵向累计加

Out[**36**]:

array([[ 0, 1, 2],

[ 3, 5, 7],

[ 9, 12, 15]], dtype=int32)

In [**37**]: jar.cumsum(1)#横向累计加

Out[**37**]:

array([[ 0, 1, 3],

[ 3, 7, 12],

[ 6, 13, 21]], dtype=int32)

In [**38**]: np.cumsum(jar)

Out[**38**]: array([ 0, 1, 3, 6, 10, 15, 21, 28, 36], dtype=int32)

In [**39**]: np.cumprod(jar)

Out[**39**]: array([0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)

例8

In [**40**]: jar.cumprod(0)#纵向累计乘

Out[**40**]:

array([[ 0, 1, 2],

[ 0, 4, 10],

[ 0, 28, 80]], dtype=int32)

In [**42**]: jar.cumprod(1)#横向累计乘

Out[**42**]:

array([[ 0, 0, 0],

[ 3, 12, 60],

[ 6, 42, 336]], dtype=int32)

Python方差算法确认：

我们讲了两个方差算法（总体方差和样本方差），到底python采用哪一个呢？看看下面代码，你们就明白了：

In [**1**]: import numpy as np

In [**2**]: array1=np.array([1,2,3])

In [**3**]: ((array1-array1.sum()/3)\*\*2).sum()/3

Out[**3**]: 0.66666666666666663

In [**4**]: ((array1-array1.sum()/3)\*\*2).sum()/2

Out[**4**]: 1.0

In [**5**]: np.var(array1)

Out[**5**]: 0.66666666666666663

Python采用了总体方差算法。我个人建议用样本方差算法。

**布尔型数组的方法**

数值数组可以十分容易的转化为bool数组：

In [**1**]: import numpy as np

In [**3**]: arr=np.arange(12).reshape(3,4)

In [**4**]: arr

Out[**4**]:

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11]])

In [**7**]: arr[1,2:]

Out[**7**]: array([6, 7])

In [**9**]: arr[1,2:]=np.array([-4,-6])

In [**10**]: arr

Out[**10**]:

array([[ 0, 1, 2, 3],

[ 4, 5, -4, -6],

[ 8, 9, 10, 11]])

In [**11**]: arr>0

Out[**11**]:

array([[False, True, True, True],

[ True, True, False, False],

[ True, True, True, True]], dtype=bool)

In [**13**]: boolm=(arr>0)

In [**14**]: boolm

Out[**14**]:

array([[False, True, True, True],

[ True, True, False, False],

[ True, True, True, True]], dtype=bool)

布尔数组的sum方法：

In [**15**]: boolm.sum()

Out[**15**]: 9

In [**16**]: np.sum(boolm)

Out[**16**]: 9

布尔数组的all和any方法：

In [**17**]: boolm.any()

Out[**17**]: True

In [**18**]: boolm.all()

Out[**18**]: False

#any（）说明数组里有true，all（）检验数组是不是全是true。可用于检验是否存在特征数据

用bool值索引选取数值：

In [**6**]: arr=np.arange(12).reshape(3,4)

In [**7**]: arr

Out[**7**]:

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11]])

In [**8**]: arr[1,2:]=np.array([-4,-6])

In [**9**]: arr

Out[**9**]:

array([[ 0, 1, 2, 3],

[ 4, 5, -4, -6],

[ 8, 9, 10, 11]])

In [**10**]: arr[(arr>=2) & (arr<=8 )]

Out[**10**]: array([2, 3, 4, 5, 8])

In [**11**]: arr[(arr>=2)|(arr<=8 )]

Out[**11**]: array([ 0, 1, 2, 3, 4, 5, -4, -6, 8, 9, 10, 11])

In [**12**]: arr[1,2:]=[3,4]

In [**13**]: arr

Out[**13**]:

array([[ 0, 1, 2, 3],

[ 4, 5, 3, 4],

[ 8, 9, 10, 11]])

**数组的排序与整理**

与列表一样，数组也可以进行排序。我们这里先复习下列表的排序:

In [**44**]: list3=[3,6,3,-2,-7,-1]

In [**45**]: list3.sort()

In [**46**]: list3

Out[**46**]: [-7, -2, -1, 3, 3, 6]

下面我们来看下这种方法应用到数组的效果，以二维数组为例。

In [**24**]: tesatarr1=np.random.randn(4,4)

In [**28**]: np.floor(tesatarr1)

Out[**28**]:

array([[ 0., 0., 0., -2.],

[-2., 0., -1., 0.],

[-2., 0., 0., 1.],

[-2., 0., -1., 1.]])

In [**30**]: testarr1=np.floor(tesatarr1)

In [**31**]: testarr1.sort(0)

In [**34**]: testarr1

Out[**34**]:

array([[-2., 0., -1., -2.],

[-2., 0., -1., 0.],

[-2., 0., 0., 1.],

[ 0., 0., 0., 1.]])

#按列进行排序，’0’代表纵向轴，也就是列。

In [**35**]: testarr1.sort(1)

In [**36**]: testarr1

Out[**36**]:

array([[-2., -2., -1., 0.],

[-2., -1., 0., 0.],

[-2., 0., 0., 1.],

[ 0., 0., 0., 1.]])

#按行进行排序，’1’代表横向轴，也就是行。

我们发现这种方法有一种缺陷，它往往会改变数组的内容。为了避免这一点，我们调用numpython的sort函数。

#首先我创建一个新的数组

In [**52**]: testarr1[1:,1:]=np.array([[4,-9,0],[-3,-1,6.7],[-3.8,9.6,8]])

In [**53**]: testarr1

Out[**53**]:

array([[-2. , -2. , -1. , 0. ],

[-2. , 4. , -9. , 0. ],

[-2. , -3. , -1. , 6.7],

[ 0. , -3.8, 9.6, 8. ]])

In [**54**]: tesarry5=np.sort(testarr1)

#把改变的数组副本传给一个变量，保存下来

In [**55**]: testarr1

Out[**55**]:

array([[-2. , -2. , -1. , 0. ],

[-2. , 4. , -9. , 0. ],

[-2. , -3. , -1. , 6.7],

[ 0. , -3.8, 9.6, 8. ]])

#原数组没有改变，np.sort函数只是返回了一个改变的数组副本。如没有变量接收，会自动消除。

In [**56**]: tesarry5

Out[**56**]:

array([[-2. , -2. , -1. , 0. ],

[-9. , -2. , 0. , 4. ],

[-3. , -2. , -1. , 6.7],

[-3.8, 0. , 8. , 9.6]])

主要用于数据整理！

数据分析的第一步就是数据的清洗与整理，使杂乱无章的数据整洁有序。特别是带有日期数据，很多时候去重是必须的一个步骤。（时间不可能重复，比如不可能出现两个国庆节，尽管我们十分希望如此）。

为此，Numpy 提供了一些十分有用的针对一维数组的函数，最常用的要数np.unique了。

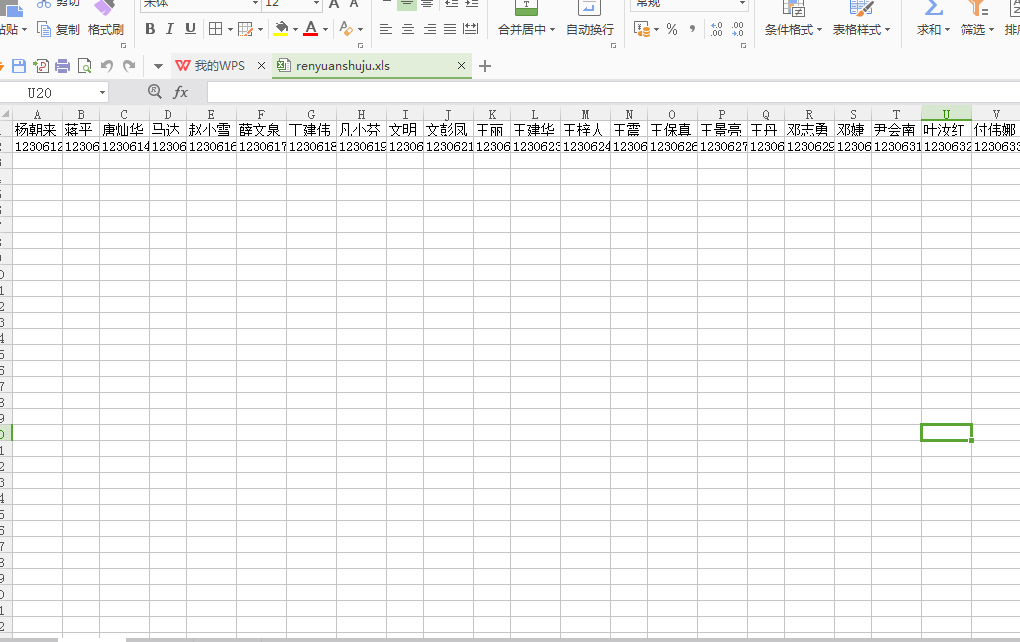
In [**35**]: arrayTest\_12=np.array([1,7,4,-9,3.4,5,3,3.5,7.2,9,8,7.2])

In [**37**]: np.unique(arrayTest\_12)

Out[**37**]: array([-9. , 1. , 3. , 3.4, 3.5, 4. , 5. , 7. , 7.2, 8. , 9. ])

这里还有一个函数值的我们去关注，它就是np.in1d,可用于寻找一维数组的特征值。

举例：寻找幸运工号。单位开展幸运工号抽奖，抽到者奖励1000元。代码如下：



In [**1**]: import numpy as np

In [**2**]: import xlrd

In [**3**]: name=r'C:\Users\dongfeng\Desktop\renyuanshuju.xls'

In [**4**]: test1=xlrd.open\_workbook(name)

In [**5**]: table = test1.sheet\_by\_name(u'Sheet1')

In [**6**]: daten1=[]

In [**7**]: nrows = table.nrows

In [**9**]: for j in range(0,nrows,1):

   ...: daten1.append(list(table.row\_values(j)))

   ...:

In [**12**]: def search\_lucker(worknr1,worknr2,worknr3):

    ...: lucker=[]

    ...: bollmatrix=np.in1d(daten1[1],[worknr1,worknr2,worknr3])

    ...: for i in range(len(bollmatrix)):

    ...: if bollmatrix[i]:

    ...: b=daten1[0][i]

    ...: lucker.append(b)

    ...: return lucker

    ...:

In [**13**]: search\_lucker(1230633,1230646,1230666)

Out[**13**]: ['付伟娜', '刘伟华', '杨国栋']

其它有用的数组

In [**21**]: import numpy as np

In [**22**]: test1=np.arange(16).reshape(4,4)

In [**23**]: test1[0:3,0:3]=np.array([[-3,-6,-4],[-1,2,7],[3,8,2]])

In [**24**]: test1[0,2]=np.array([4])

In [**25**]: test2=np.add(test1,np.ones((4,4)))

In [**26**]: test1,test2

Out[**26**]:

(array([[-3, -6, 4, 3],

[-1, 2, 7, 7],

[ 3, 8, 2, 11],

[12, 13, 14, 15]]), array([[ -2., -5., 5., 4.],

[ 0., 3., 8., 8.],

[ 4., 9., 3., 12.],

[ 13., 14., 15., 16.]]))

In [**27**]: np.intersect1d(test2,test1)

Out[**27**]: array([ 3., 4., 8., 12., 13., 14., 15.])

In [**29**]: np.intersect1d(test2[0,:],test1[0,:])

Out[**29**]: array([ 4.])

In [**31**]: np.union1d(test2[0,:],test1[0,:])

Out[**31**]: array([-6., -5., -3., -2., 3., 4., 5.])

In [**32**]: np.setdiff1d(test2[0,:],test1[0,:])

Out[**32**]: array([-5., -2., 5.])

In [**33**]: np.setxor1d(test2[0,:],test1[0,:])

Out[**33**]: array([-6., -5., -3., -2., 3., 5.])

# 数组文件的输入与输出

在磁盘上的存取：

In [**38**]: arr=np.arange(1,9,2)

In [**39**]: arr

Out[**39**]: array([1, 3, 5, 7])

In [**41**]: np.save('testarraysave',arr)

In [**43**]: np.load('testarraysave.npy')

Out[**43**]: array([1, 3, 5, 7])

#也可一次存储多个数组

In [**44**]: arr1=np.arange(6)

In [**45**]: np.savez('arraysavemore',m=arr,n=arr1)

In [**48**]: indirectarr=np.load('arraysavemore.npz')

In [**50**]: indirectarr['m'],indirectarr['n']

Out[**50**]: (array([1, 3, 5, 7]), array([0, 1, 2, 3, 4, 5]))

通过文本文件存取（重要）

绝对路径直接把数据读取到Python编程页面：

In [**65**]: !type "C:\Users\dongfeng\Desktop\yong344523.csv"

0.3,1.3,2.3,3.3,4.3,5.3,6.3,7.3

1.3,2.3,3.3,4.3,5.3,6.3,7.3,8.3

2.3,3.3,4.3,5.3,6.3,7.3,8.3,9.3

3.3,4.3,5.3,6.3,7.3,8.3,9.3,10.3

4.3,5.3,6.3,7.3,8.3,9.3,10.3,11.3

5.3,6.3,7.3,8.3,9.3,10.3,11.3,12.3

6.3,7.3,8.3,9.3,10.3,11.3,12.3,13.3

7.3,8.3,9.3,10.3,11.3,12.3,13.3,14.3

8.3,9.3,10.3,11.3,12.3,13.3,14.3,15.3

9.3,10.3,11.3,12.3,13.3,14.3,15.3,16.3

10.3,11.3,12.3,13.3,14.3,15.3,16.3,17.3

11.3,12.3,13.3,14.3,15.3,16.3,17.3,18.3

12.3,13.3,14.3,15.3,16.3,17.3,18.3,19.3

13.3,14.3,15.3,16.3,17.3,18.3,19.3,20.3

14.3,15.3,16.3,17.3,18.3,19.3,20.3,21.3

15.3,16.3,17.3,18.3,19.3,20.3,21.3,22.3

16.3,17.3,18.3,19.3,20.3,21.3,22.3,23.3

17.3,18.3,19.3,20.3,21.3,22.3,23.3,24.3

18.3,19.3,20.3,21.3,22.3,23.3,24.3,25.3

19.3,20.3,21.3,22.3,23.3,24.3,25.3,26.3

20.3,21.3,22.3,23.3,24.3,25.3,26.3,27.3

21.3,22.3,23.3,24.3,25.3,26.3,27.3,28.3

22.3,23.3,24.3,25.3,26.3,27.3,28.3,29.3

23.3,24.3,25.3,26.3,27.3,28.3,29.3,30.3

24.3,25.3,26.3,27.3,28.3,29.3,30.3,31.3

25.3,26.3,27.3,28.3,29.3,30.3,31.3,32.3

26.3,27.3,28.3,29.3,30.3,31.3,32.3,33.3

27.3,28.3,29.3,30.3,31.3,32.3,33.3,34.3

相对路径直接把数据读取到Python编程页面：

In [**66**]: !type Desktop\yong344523.csv

0.3,1.3,2.3,3.3,4.3,5.3,6.3,7.3

1.3,2.3,3.3,4.3,5.3,6.3,7.3,8.3

2.3,3.3,4.3,5.3,6.3,7.3,8.3,9.3

3.3,4.3,5.3,6.3,7.3,8.3,9.3,10.3

4.3,5.3,6.3,7.3,8.3,9.3,10.3,11.3

5.3,6.3,7.3,8.3,9.3,10.3,11.3,12.3

6.3,7.3,8.3,9.3,10.3,11.3,12.3,13.3

7.3,8.3,9.3,10.3,11.3,12.3,13.3,14.3

8.3,9.3,10.3,11.3,12.3,13.3,14.3,15.3

9.3,10.3,11.3,12.3,13.3,14.3,15.3,16.3

10.3,11.3,12.3,13.3,14.3,15.3,16.3,17.3

11.3,12.3,13.3,14.3,15.3,16.3,17.3,18.3

12.3,13.3,14.3,15.3,16.3,17.3,18.3,19.3

13.3,14.3,15.3,16.3,17.3,18.3,19.3,20.3

14.3,15.3,16.3,17.3,18.3,19.3,20.3,21.3

15.3,16.3,17.3,18.3,19.3,20.3,21.3,22.3

16.3,17.3,18.3,19.3,20.3,21.3,22.3,23.3

17.3,18.3,19.3,20.3,21.3,22.3,23.3,24.3

18.3,19.3,20.3,21.3,22.3,23.3,24.3,25.3

19.3,20.3,21.3,22.3,23.3,24.3,25.3,26.3

20.3,21.3,22.3,23.3,24.3,25.3,26.3,27.3

21.3,22.3,23.3,24.3,25.3,26.3,27.3,28.3

22.3,23.3,24.3,25.3,26.3,27.3,28.3,29.3

23.3,24.3,25.3,26.3,27.3,28.3,29.3,30.3

24.3,25.3,26.3,27.3,28.3,29.3,30.3,31.3

25.3,26.3,27.3,28.3,29.3,30.3,31.3,32.3

26.3,27.3,28.3,29.3,30.3,31.3,32.3,33.3

27.3,28.3,29.3,30.3,31.3,32.3,33.3,34.3

继续把数据转化为数组：

In [**72**]: testarray\_1=np.loadtxt('Desktop\yong344523.csv',delimiter=',')

In [**73**]: testarray\_1

Out[**73**]:

array([[ 0.3, 1.3, 2.3, 3.3, 4.3, 5.3, 6.3, 7.3],

[ 1.3, 2.3, 3.3, 4.3, 5.3, 6.3, 7.3, 8.3],

[ 2.3, 3.3, 4.3, 5.3, 6.3, 7.3, 8.3, 9.3],

[ 3.3, 4.3, 5.3, 6.3, 7.3, 8.3, 9.3, 10.3],

[ 4.3, 5.3, 6.3, 7.3, 8.3, 9.3, 10.3, 11.3],

[ 5.3, 6.3, 7.3, 8.3, 9.3, 10.3, 11.3, 12.3],

[ 6.3, 7.3, 8.3, 9.3, 10.3, 11.3, 12.3, 13.3],

[ 7.3, 8.3, 9.3, 10.3, 11.3, 12.3, 13.3, 14.3],

[ 8.3, 9.3, 10.3, 11.3, 12.3, 13.3, 14.3, 15.3],

[ 9.3, 10.3, 11.3, 12.3, 13.3, 14.3, 15.3, 16.3],

[ 10.3, 11.3, 12.3, 13.3, 14.3, 15.3, 16.3, 17.3],

[ 11.3, 12.3, 13.3, 14.3, 15.3, 16.3, 17.3, 18.3],

[ 12.3, 13.3, 14.3, 15.3, 16.3, 17.3, 18.3, 19.3],

[ 13.3, 14.3, 15.3, 16.3, 17.3, 18.3, 19.3, 20.3],

[ 14.3, 15.3, 16.3, 17.3, 18.3, 19.3, 20.3, 21.3],

[ 15.3, 16.3, 17.3, 18.3, 19.3, 20.3, 21.3, 22.3],

[ 16.3, 17.3, 18.3, 19.3, 20.3, 21.3, 22.3, 23.3],

[ 17.3, 18.3, 19.3, 20.3, 21.3, 22.3, 23.3, 24.3],

[ 18.3, 19.3, 20.3, 21.3, 22.3, 23.3, 24.3, 25.3],

[ 19.3, 20.3, 21.3, 22.3, 23.3, 24.3, 25.3, 26.3],

[ 20.3, 21.3, 22.3, 23.3, 24.3, 25.3, 26.3, 27.3],

[ 21.3, 22.3, 23.3, 24.3, 25.3, 26.3, 27.3, 28.3],

[ 22.3, 23.3, 24.3, 25.3, 26.3, 27.3, 28.3, 29.3],

[ 23.3, 24.3, 25.3, 26.3, 27.3, 28.3, 29.3, 30.3],

[ 24.3, 25.3, 26.3, 27.3, 28.3, 29.3, 30.3, 31.3],

[ 25.3, 26.3, 27.3, 28.3, 29.3, 30.3, 31.3, 32.3],

[ 26.3, 27.3, 28.3, 29.3, 30.3, 31.3, 32.3, 33.3],

[ 27.3, 28.3, 29.3, 30.3, 31.3, 32.3, 33.3, 34.3]])

我们还可以把数组储存到指定地点

In [**82**]: np.savetxt("desktop\gsdgusdh",testarray\_1,delimiter=',')

#指定地点

In [**83**]: np.savetxt("desktop\gsdgusdh.txt",testarray\_1,delimiter=',')

#指定地点和格式

**项目：销售数据清洗**

In [**1**]: import numpy as np

In [**2**]: def get\_repeated\_label(array\_1dim):

   ...: label\_1dim=[]

   ...: for k in range(len(array\_1dim)-1):

   ...: for i in range(k+1,len(array\_1dim),1):

   ...: if array\_1dim[k]==array\_1dim[i]:

   ...: label\_1dim.append(i)

   ...:

   ...: return np.unique(label\_1dim)

   ...:

In [**33**]: def invalid\_value\_cleaning (array\_variable):

    ...: for i in range(np.shape(array\_variable)[0]):

    ...: for j in range(np.shape(array\_variable)[1]):

    ...: if type(array\_variable[i,j])==float or type(array\_variable[i,j])==int:

    ...: pass

    ...: else:

    ...: array\_variable[i,j]=0

    ...: array\_variable[np.where(np.isnan(array\_variable.astype(np.float64)))]=0

    ...: return array\_variable

...:

In [**4**]: def fillup\_mean\_value(array\_parameter):

   ...: for index\_j in range(np.shape(array\_parameter)[1]):

   ...: array\_columns\_nr=list(np.where(array\_parameter==0)[1])

   ...: if index\_j in array\_columns\_nr:

   ...: appearance\_sum=array\_columns\_nr.count(index\_j)

   ...: if appearance\_sum>0:

   ...: mean\_1=(array\_parameter[:,index\_j]).sum()/(np.shape(array\_parameter)[0]-appearance\_sum)

   ...: bollmatrix\_1=array\_parameter[:,index\_j]==0

   ...: array\_parameter[bollmatrix\_1,index\_j]=mean\_1

   ...:

   ...: return array\_parameter

   ...:

In [**5**]: def delete\_rows(list\_variable,array\_variable):

   ...: i=0

   ...: list\_variable.sort()

   ...: array\_medium=array\_variable

   ...: for j in list\_variable:

   ...: if i==0:

   ...: array\_medium=np.delete(array\_medium,j,0)

   ...: else:

   ...: array\_medium=np.delete(array\_medium,j-i,0)

   ...: i=i+1

   ...: return array\_medium

In [**6**]: import xlrd

In [**7**]: name=r'C:\Users\dongfeng\Desktop\salesdetails.xlsx'

In [**8**]: #把路径赋给变量name

In [**9**]: information\_1=xlrd.open\_workbook(name)

In [**10**]: information\_2 = information\_1.sheet\_by\_name(u'Sheet1')

In [**11**]: dataload\_1=[]

In [**12**]: nrows = information\_2.nrows

In [**13**]: for j in range(1,nrows,1):

    ...: dataload\_1.append(list(information\_2.row\_values(j)))

In [**14**]: Data\_Deal\_with=np.array(dataload\_1,dtype=np.object)

In [**15**]: Data\_Deal\_with

Out[**15**]:

array([[20170323.0, 126.0, 229.0, 159.0, 134300.0],

[20170324.0, 128.0, 175.0, 111.0, 221.0],

[20170325.0, 117.0, 77.0, 40.0, 171.0],

[20170326.0, 93.0, 30.0, 28.0, 106.0],

[20170327.0, 73.0, 46.0, 30.0, 123.0],

[20170328.0, '￥$#', 28.0, 16.0, 74.0],

[20170329.0, 46.0, 13.0, 10.0, 50.0],

[20170330.0, 32.0, 11.0, 19.0, 30.0],

[20170331.0, 25.0, 70000.0, 11.0, 53.0],

[20170401.0, 41.0, 10.0, 16.0, 27.0],

[20170401.0, 41.0, 10.0, 16.0, 27.0],

[20170402.0, 21.0, 166.0, 66.0, 24.0],

[20170403.0, 60.0, 67.0, 275.0, 17.0],

[20170404.0, 74.0, 46.0, 51.0, 18.0],

[20170405.0, 76.0, 25.0, 42.0, 12.0],

[20170406.0, 54.0, 32.0, '', 11.0],

[20170407.0, 186.0, 18.0, 18.0, 97700.0],

[20170408.0, 121.0, 'sales ist Null', 23.0, 440.0],

[20170409.0, 83.0, 17.0, 16.0, 107.0],

[20170410.0, 87.0, 12.0, 15.0, 117.0],

[20170411.0, 35.0, 16.0, 22.0, 62.0],

[20170411.0, 35.0, 16.0, 22.0, 62.0],

[20170412.0, '', 23.0, 15.0, 128.0],

[20170413.0, 25.0, 17.0, 17.0, ''],

[20170414.0, 24.0, 27.0, '', 30.0],

[20170415.0, 13.0, 36.0, 12.0, 65.0],

[20170415.0, 13.0, 36.0, 12.0, 65.0],

[20170416.0, 9.0, 45.0, 147.0, 22.0],

[20170417.0, 14.0, 54.0, 22.0, 22.0],

[20170418.0, 16.0, 21.0, 4.0, 15.0],

[20170419.0, 8.0, 13.0, 4.0, 23.0],

[20170420.0, '%$', 59.0, 32452.0, 11.0],

[20170421.0, 6.0, 89.0, 4.0, 13.0],

[20170422.0, 5.0, 76.0, 31.0, 11.0]], dtype=object)

In [**16**]: Columns\_1=Data\_Deal\_with[:,0]

In [**17**]: Columns\_1

Out[**17**]:

array([20170323.0, 20170324.0, 20170325.0, 20170326.0, 20170327.0,

20170328.0, 20170329.0, 20170330.0, 20170331.0, 20170401.0,

20170401.0, 20170402.0, 20170403.0, 20170404.0, 20170405.0,

20170406.0, 20170407.0, 20170408.0, 20170409.0, 20170410.0,

20170411.0, 20170411.0, 20170412.0, 20170413.0, 20170414.0,

20170415.0, 20170415.0, 20170416.0, 20170417.0, 20170418.0,

20170419.0, 20170420.0, 20170421.0, 20170422.0], dtype=object)

In [**18**]: get\_repeated\_label(Columns\_1)

Out[**18**]: array([10, 21, 26])

In [**19**]: delete\_rows(get\_repeated\_label(Columns\_1),Data\_Deal\_with)

Out[**19**]:

array([[20170323.0, 126.0, 229.0, 159.0, 134300.0],

[20170324.0, 128.0, 175.0, 111.0, 221.0],

[20170325.0, 117.0, 77.0, 40.0, 171.0],

[20170326.0, 93.0, 30.0, 28.0, 106.0],

[20170327.0, 73.0, 46.0, 30.0, 123.0],

[20170328.0, '￥$#', 28.0, 16.0, 74.0],

[20170329.0, 46.0, 13.0, 10.0, 50.0],

[20170330.0, 32.0, 11.0, 19.0, 30.0],

[20170331.0, 25.0, 70000.0, 11.0, 53.0],

[20170401.0, 41.0, 10.0, 16.0, 27.0],

[20170402.0, 21.0, 166.0, 66.0, 24.0],

[20170403.0, 60.0, 67.0, 275.0, 17.0],

[20170404.0, 74.0, 46.0, 51.0, 18.0],

[20170405.0, 76.0, 25.0, 42.0, 12.0],

[20170406.0, 54.0, 32.0, '', 11.0],

[20170407.0, 186.0, 18.0, 18.0, 97700.0],

[20170408.0, 121.0, 'sales ist Null', 23.0, 440.0],

[20170409.0, 83.0, 17.0, 16.0, 107.0],

[20170410.0, 87.0, 12.0, 15.0, 117.0],

[20170411.0, 35.0, 16.0, 22.0, 62.0],

[20170412.0, '', 23.0, 15.0, 128.0],

[20170413.0, 25.0, 17.0, 17.0, ''],

[20170414.0, 24.0, 27.0, '', 30.0],

[20170415.0, 13.0, 36.0, 12.0, 65.0],

[20170416.0, 9.0, 45.0, 147.0, 22.0],

[20170417.0, 14.0, 54.0, 22.0, 22.0],

[20170418.0, 16.0, 21.0, 4.0, 15.0],

[20170419.0, 8.0, 13.0, 4.0, 23.0],

[20170420.0, '%$', 59.0, 32452.0, 11.0],

[20170421.0, 6.0, 89.0, 4.0, 13.0],

[20170422.0, 5.0, 76.0, 31.0, 11.0]], dtype=object)

In [**34**]: invalid\_value\_cleaning(Data\_information\_12)

Out[**34**]:

array([[20170323.0, 126.0, 229.0, 159.0, 134300.0],

[20170324.0, 128.0, 175.0, 111.0, 221.0],

[20170325.0, 117.0, 77.0, 40.0, 171.0],

[20170326.0, 93.0, 30.0, 28.0, 106.0],

[20170327.0, 73.0, 46.0, 30.0, 123.0],

[20170328.0, 0, 28.0, 16.0, 74.0],

[20170329.0, 46.0, 13.0, 10.0, 50.0],

[20170330.0, 32.0, 11.0, 19.0, 30.0],

[20170331.0, 25.0, 70000.0, 11.0, 53.0],

[20170401.0, 41.0, 10.0, 16.0, 27.0],

[20170402.0, 21.0, 166.0, 66.0, 24.0],

[20170403.0, 60.0, 67.0, 275.0, 17.0],

[20170404.0, 74.0, 46.0, 51.0, 18.0],

[20170405.0, 76.0, 25.0, 42.0, 12.0],

[20170406.0, 54.0, 32.0, 0, 11.0],

[20170407.0, 186.0, 18.0, 18.0, 97700.0],

[20170408.0, 121.0, 0, 23.0, 440.0],

[20170409.0, 83.0, 17.0, 16.0, 107.0],

[20170410.0, 87.0, 12.0, 15.0, 117.0],

[20170411.0, 35.0, 16.0, 22.0, 62.0],

[20170412.0, 0, 23.0, 15.0, 128.0],

[20170413.0, 25.0, 17.0, 17.0, 0],

[20170414.0, 24.0, 27.0, 0, 30.0],

[20170415.0, 13.0, 36.0, 12.0, 65.0],

[20170416.0, 9.0, 45.0, 147.0, 22.0],

[20170417.0, 14.0, 54.0, 22.0, 22.0],

[20170418.0, 16.0, 21.0, 4.0, 15.0],

[20170419.0, 8.0, 13.0, 4.0, 23.0],

[20170420.0, 0, 59.0, 32452.0, 11.0],

[20170421.0, 6.0, 89.0, 4.0, 13.0],

[20170422.0, 5.0, 76.0, 31.0, 11.0]], dtype=object)

In [**35**]: Indormation\_after\_handling=invalid\_value\_cleaning(Data\_information\_12)

In [**37**]: fillup\_mean\_value(Indormation\_after\_handling)

Out[**37**]:

array([[20170323.0, 126.0, 229.0, 159.0, 134300.0],

[20170324.0, 128.0, 175.0, 111.0, 221.0],

[20170325.0, 117.0, 77.0, 40.0, 171.0],

[20170326.0, 93.0, 30.0, 28.0, 106.0],

[20170327.0, 73.0, 46.0, 30.0, 123.0],

[20170328.0, 53.266666666666666, 28.0, 16.0, 74.0],

[20170329.0, 46.0, 13.0, 10.0, 50.0],

[20170330.0, 32.0, 11.0, 19.0, 30.0],

[20170331.0, 25.0, 70000.0, 11.0, 53.0],

[20170401.0, 41.0, 10.0, 16.0, 27.0],

[20170402.0, 21.0, 166.0, 66.0, 24.0],

[20170403.0, 60.0, 67.0, 275.0, 17.0],

[20170404.0, 74.0, 46.0, 51.0, 18.0],

[20170405.0, 76.0, 25.0, 42.0, 12.0],

[20170406.0, 54.0, 32.0, 1202.7142857142858, 11.0],

[20170407.0, 186.0, 18.0, 18.0, 97700.0],

[20170408.0, 121.0, 2464.7586206896553, 23.0, 440.0],

[20170409.0, 83.0, 17.0, 16.0, 107.0],

[20170410.0, 87.0, 12.0, 15.0, 117.0],

[20170411.0, 35.0, 16.0, 22.0, 62.0],

[20170412.0, 53.266666666666666, 23.0, 15.0, 128.0],

[20170413.0, 25.0, 17.0, 17.0, 8666.777777777777],

[20170414.0, 24.0, 27.0, 1202.7142857142858, 30.0],

[20170415.0, 13.0, 36.0, 12.0, 65.0],

[20170416.0, 9.0, 45.0, 147.0, 22.0],

[20170417.0, 14.0, 54.0, 22.0, 22.0],

[20170418.0, 16.0, 21.0, 4.0, 15.0],

[20170419.0, 8.0, 13.0, 4.0, 23.0],

[20170420.0, 53.266666666666666, 59.0, 32452.0, 11.0],

[20170421.0, 6.0, 89.0, 4.0, 13.0],

[20170422.0, 5.0, 76.0, 31.0, 11.0]], dtype=object)

In [**39**]: DataFormat=fillup\_mean\_value(Indormation\_after\_handling)

In [**43**]: Format\_DataFrame=pd.DataFrame(DataFormat)

In [**44**]: format\_array\_Frame=lambda x:'%u' % x

In [**45**]: Format\_DataFrame.applymap(format\_array\_Frame)

Out[**45**]:

0 1 2 3 4

0 20170323 126 229 159 134300

1 20170324 128 175 111 221

2 20170325 117 77 40 171

3 20170326 93 30 28 106

4 20170327 73 46 30 123

5 20170328 53 28 16 74

6 20170329 46 13 10 50

7 20170330 32 11 19 30

8 20170331 25 70000 11 53

9 20170401 41 10 16 27

10 20170402 21 166 66 24

11 20170403 60 67 275 17

12 20170404 74 46 51 18

13 20170405 76 25 42 12

14 20170406 54 32 1202 11

15 20170407 186 18 18 97700

16 20170408 121 2464 23 440

17 20170409 83 17 16 107

18 20170410 87 12 15 117

19 20170411 35 16 22 62

20 20170412 53 23 15 128

21 20170413 25 17 17 8666

22 20170414 24 27 1202 30

23 20170415 13 36 12 65

24 20170416 9 45 147 22

25 20170417 14 54 22 22

26 20170418 16 21 4 15

27 20170419 8 13 4 23

28 20170420 53 59 32452 11

29 20170421 6 89 4 13

30 20170422 5 76 31 11

数据清洗项目2

def cleanstr(wrr):#参数是列表，就是你要处理的字符串

list\_1=[]

list\_2=[]

str\_1=''

for j in range(len(wrr)):

try:

#if wrr[j]=='.' and j!=(len(wrr)-1) and wrr[j+1]!=' 'and wrr[j+1]!='\n' and wrr[j-1]!='\n' and type(wrr[j+1])!=str and type(wrr[j-1])!=str:

if wrr[j]=='.' and j!=(len(wrr)-1) and (type(float(wrr[j-1]))!=str or type(float(wrr[j-1]))!=str):

list\_1.append(wrr[j])

elif type(float(wrr[j]))==float:

list\_1.append(int(wrr[j]))

elif wrr[j]==',':

wrr[j]='#'

except ValueError:

wrr[j]='#'

list\_1.append(wrr[j])

for j in range(len(list\_1)):

#print(len(list\_1))

if type(list\_1[j]) is int or list\_1[j]=='.':

str\_1=str\_1+str(list\_1[j])

elif list\_1[j]=='#':

list\_2.append([str\_1])

str\_1=''

continue

str\_1=''

for k in range(-1,-len(list\_1),-1):

if list\_1[k]!='#':

str\_1=str\_1+str(list\_1[k])

else:

break

str\_1new=str\_1[::-1]

list\_2.append([str\_1new])

list\_2\_new=np.unique(list\_2)

try:

return (float('%.2f'%(list\_2\_new[1:].astype(np.float64).min())),(list\_2\_new[1:]).astype(np.float64))

except ValueError:

return 'value invaild'

def strrepair(columns\_str\_ar):#参数是待处理的那一列数据

for j in range(1,columns\_str\_ar.shape[0]):

#print(j)

if type(columns\_str\_ar[j])==str:

type(columns\_str\_ar[j])

#print(Data\_1['IELTS'][j])

str\_1=columns\_str\_ar[j]

#print(Data\_1['IELTS'][j])

ind=cleanstr(list(str\_1))

if type(ind)!=str:

Data\_1.loc[j,'TOEFL\_L':'TOEFL\_W']=ind[0]

columns\_str\_ar[j] =ind[1].max()

return columns\_str\_ar

数组的组合与分割

import numpy as np

#下面我们着重讨论一下数组的组合与分割：

a1=np.arange(16).reshape(4,4)

a2=np.arange(2,34,2).reshape(4,4)

a1

Out[10]:

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11],

[12, 13, 14, 15]])

a2

Out[11]:

array([[ 2, 4, 6, 8],

[10, 12, 14, 16],

[18, 20, 22, 24],

[26, 28, 30, 32]])

1. 水平组合：

np.hstack((a1,a2))

Out[13]:

array([[ 0, 1, 2, 3, 2, 4, 6, 8],

[ 4, 5, 6, 7, 10, 12, 14, 16],

[ 8, 9, 10, 11, 18, 20, 22, 24],

[12, 13, 14, 15, 26, 28, 30, 32]])

np.concatenate((a1,a2),axis=1)

Out[15]:

array([[ 0, 1, 2, 3, 2, 4, 6, 8],

[ 4, 5, 6, 7, 10, 12, 14, 16],

[ 8, 9, 10, 11, 18, 20, 22, 24],

[12, 13, 14, 15, 26, 28, 30, 32]])

#要想使两个数组通过hstack函数拼接成功，我们要求：1.数组的维度必须相同；2.行数必须相同。

1. 垂直组合：

np.vstack((a1,a2))

Out[16]:

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11],

[12, 13, 14, 15],

[ 2, 4, 6, 8],

[10, 12, 14, 16],

[18, 20, 22, 24],

[26, 28, 30, 32]])

np.concatenate((a1,a2),axis=0)

Out[17]:

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11],

[12, 13, 14, 15],

[ 2, 4, 6, 8],

[10, 12, 14, 16],

[18, 20, 22, 24],

[26, 28, 30, 32]])

#通过vstack函数进行垂直组合，各数组列数必须相同，此规律也适合一维数组。

1. 沿着纵轴方向组合

np.dstack((a1,a2))

Out[18]:

array([[[ 0, 2],

[ 1, 4],

[ 2, 6],

[ 3, 8]],

[[ 4, 10],

[ 5, 12],

[ 6, 14],

[ 7, 16]],

[[ 8, 18],

[ 9, 20],

[10, 22],

[11, 24]],

[[12, 26],

[13, 28],

[14, 30],

[15, 32]]])

#通过dtack函数进行数组沿轴纵向对接各数组必须列数与行数均相等。

1. 列组合

np.column\_stack((a1,a2))

Out[19]:

array([[ 0, 1, 2, 3, 2, 4, 6, 8],

[ 4, 5, 6, 7, 10, 12, 14, 16],

[ 8, 9, 10, 11, 18, 20, 22, 24],

[12, 13, 14, 15, 26, 28, 30, 32]])

#通过column\_stack函数对两个一维的数组进行组合，其过程其实就是先把它们各自进行转置，然后按列进行组合。

#通过column\_stack函数对两个二维数组进行列向组合，我们要求两个数组行数必须相同。

#通过column\_stack对一个一维数组和二维数组进行列向组合，我们只需要求一维数组的长度与二维数组的行数相同.

1. 行组合

np.row\_stack((a1,a2))

Out[20]:

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11],

[12, 13, 14, 15],

[ 2, 4, 6, 8],

[10, 12, 14, 16],

[18, 20, 22, 24],

[26, 28, 30, 32]])

#通过row\_stack函数对两个一维数组进行行向组合，只要这两个数组的长度相同即可。

#通过row\_stack函数对两个数组进行行向组合，它们的列数必须是相同的。

#通过row\_stack函数对一维数组和二维数组进行行向组合，一维数组的长度等于二维数组的列数

数组分割

1. 垂直分割：

np.hsplit(a1,4)

Out[22]:

[array([[ 0],

[ 4],

[ 8],

[12]]), array([[ 1],

[ 5],

[ 9],

[13]]), array([[ 2],

[ 6],

[10],

[14]]), array([[ 3],

[ 7],

[11],

[15]])]

#通过hspilt函数对一维数组进行分割，函数的第二位置是分割成的块数。只要数组长度能够被分割的块数整除，我们就可以把数组分割成给定的块数。

#通过hspilt函数对二维数组进行分割，函数的第二位置是分割成的块数。只要数组的列数能够被分割的块数整除，我们就可以把数组分割成给定的块数

2.水平分割：

np.vsplit(a1,4)

Out[24]:

[array([[0, 1, 2, 3]]),

array([[4, 5, 6, 7]]),

array([[ 8, 9, 10, 11]]),

array([[12, 13, 14, 15]])]

#水平分割不能切一维数组

#通过vspilt函数对二维数组进行分割，函数的第二位置是分割成的块数。只要数组的行数能够被分割的块数整除，我们就可以把数组分割成给定的块数

3，纵向深度分割：（只适合三维以上数组）

array([[[[ 0, 1, 2],

[ 3, 4, 5],

[ 6, 7, 8]],

[[ 9, 10, 11],

[12, 13, 14],

[15, 16, 17]],

[[18, 19, 20],

[21, 22, 23],

[24, 25, 26]]],

[[[27, 28, 29],

[30, 31, 32],

[33, 34, 35]],

[[36, 37, 38],

[39, 40, 41],

[42, 43, 44]],

[[45, 46, 47],

[48, 49, 50],

[51, 52, 53]]],

[[[54, 55, 56],

[57, 58, 59],

[60, 61, 62]],

[[63, 64, 65],

[66, 67, 68],

[69, 70, 71]],

[[72, 73, 74],

[75, 76, 77],

[78, 79, 80]]]])

np.dsplit(four\_dimar,3)

Out[31]:

[array([[[[ 0, 1, 2]],

[[ 9, 10, 11]],

[[18, 19, 20]]],

[[[27, 28, 29]],

[[36, 37, 38]],

[[45, 46, 47]]],

[[[54, 55, 56]],

[[63, 64, 65]],

[[72, 73, 74]]]]), array([[[[ 3, 4, 5]],

[[12, 13, 14]],

[[21, 22, 23]]],

[[[30, 31, 32]],

[[39, 40, 41]],

[[48, 49, 50]]],

[[[57, 58, 59]],

[[66, 67, 68]],

[[75, 76, 77]]]]), array([[[[ 6, 7, 8]],

[[15, 16, 17]],

[[24, 25, 26]]],

[[[33, 34, 35]],

[[42, 43, 44]],

[[51, 52, 53]]],

[[[60, 61, 62]],

[[69, 70, 71]],

[[78, 79, 80]]]])]

意义不大！！

# 二，线性代数

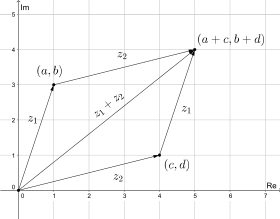
主要讲与人工智能和数据分析相关的线性代数知识。比如什么是矢量，什么是矩阵，矩阵的加减乘除。矩阵对角化，三角化，秩，QR法，最小二法。等等

**矢量：**

高中数学中都学过复数，负数表达式是：

a+bi

复数实际上和二维的矢量是异性同构的。所谓异性同构，是一个很严格的数学定义，这里不是重点，不做介绍。我们可以近似的把它理解为本质是一样的，只是外形不同，就如正直的绅士和善良的乞丐那样。唯一的不同仅仅是外形。回忆下高中我们是如何在一个平面上描述负数的。



既然复数和二维矢量是一个东西，仅仅是外表不同，那么矢量在线性代数中如何描述呢。

坐标描述：

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

几何描述：

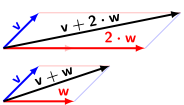
二维的矢量在二维矢量空间中和复数在复平面中是一样的。一个箭头带着常常的尾巴，箭头代表着它的方向，尾巴说明它是有长度的。矢量像一支支离弦之箭，在矢量空间翱翔。矢量空间是一个十分重要的数学概念。它包括很多子空间，你比如欧几里矢量空间、拓扑矢量空间、赋范空间、预希耳伯特空间、酉空间（复数矢量空间）。每个空间都有它的特别之处和独有的操作法则。我们关注的重点是欧几里空间，可能以后会稍微提一下赋范空间，其它的这里不做描述，尽管他们有些十分十分重要，和数据分析也密切相关。比如拓扑矢量空间。但是它实在太难了，如果你不打算当数学家的话，还是不要去招惹它。如果你感兴趣，我们可以私下聊。

我们一直在说二维矢量，那么多维矢量又是什么样呢。

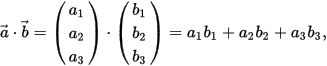
例如坐标描述四维矢量：

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

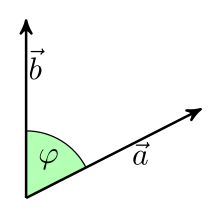
几何描述在有限维度矢量空间均和二维空间一样。



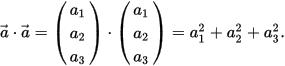
和复数中的模类似，在欧几里空间中存在范数2，它是由标积派生而出。标积的定义如下：



CodeCogsEqn



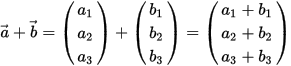
当 This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program. 时：



This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

矢量的加：

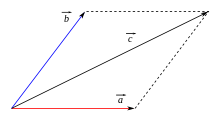
和 



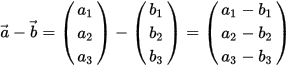
矢量加法适用结合律和交换律。

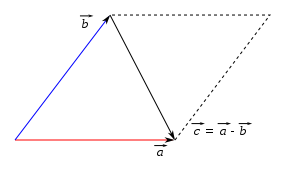
This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.



矢量的减：





矢量的三角不等式：

e2e96cba97072dc61902656e9e2885d20c85a686

此不等式成立的原因，三角形两边之和大于第三边。

注：矢量可以在欧几里空间任意平移，不改变矢量本身。

矢量的除法：

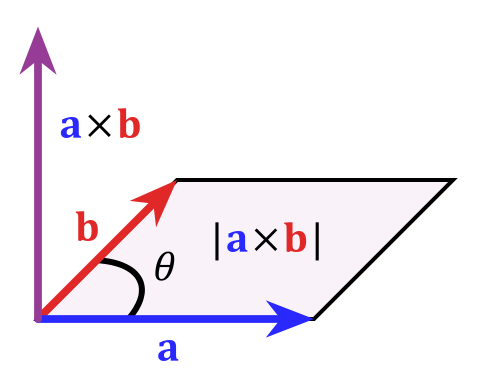
目前数学上还没有这个定义，如果你能定义出矢量除法，并且合理。那菲尔茨奖肯定颁发给你。

叉积：

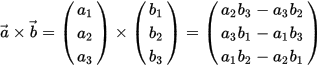
两个矢量的叉积又是矢量，其方向垂直于这两个矢量撑起的平面，其长度为：

4b3c2ec70a128d2025970ce02b0a0064b11f2f55

这个值恰恰也是上面平面的面积。



在卡迪尔坐标下，叉积可表示成：



叉积满足负交换律。

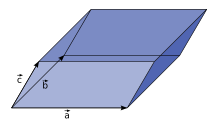
2f70833d578bae2a64da9d1998b56c623f67d2d1

交叉乘积：

人们通过下面的等式定义交叉乘积：

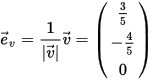
69f4c0d69d34b8fb497469da0f6e12232d350b3a

它的结果是一个标量。这个标量等于这三个矢量撑起平行六面体的体积（作业？）



如果这三个矢量构成右手系统，则0484daa6e1ff1a906bc9075871860737ce37aa6a是正值，反之，假如构成左手系统。则0484daa6e1ff1a906bc9075871860737ce37aa6a是负值。如果这些矢量是线性相关。则d508d1f9e0cc13febaa97c80d7018d39afdacf69。

矢量的归一化：



# （选学）矢量的基底变换

通常我们遇到矢量都是借助于欧几里空间的卡迪尔坐标系统标准基底来描述的。什么是基底呢。

基底定义：基底是一个矢量空间的子集。这个子集不仅能够派生出整个矢量空间，也就是说任意一个这个矢量空间的矢量都可以通过这个基底唯一表达。这个我们称之为派生系统。而且子集中各元素线性不相关。（房子理论）

如何派生出整个矢量空间?

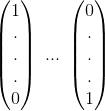
假设V是一个n维矢量空间，This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.， 是一个V的一个基底。那么我们可以这样来表达整个V矢量空间。

IMG_256

线性不相关：

IMG_256

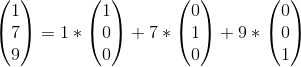
现在我们把基底限制到单位卡迪尔坐标系统标准基底基底IMG_256，也就是：



IMG_256

IMG_256

# 例子：假如矢量V1=（1,7,9）是关于标准基底的矢量，试用基本基底展开这个矢量。



我们把上面的例子普通化到n维空间,得到如下公式，

n



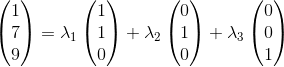
如果我们选择非标准基底，结果会怎样呢？

假如矢量V1=（1,7,9）是关于标准基底的矢量，现有另一个也是关于标准基底矢量空间基底，（（1,1,0），（0,1,0），（0,0,1），请尝试用这个基底来表示V1。

办法：我们尝试把V1分别分解到这三个矢量上；公式如下：

7899f

这里矢量IMG_256是我们的V1，IMG_256是我们这里的基底矢量中的某一个。由此可得：



求上面等式可得：



这个就是矢量V1在新基底下的列矢量表述形式。

我们可以看到同一个矢量在不同的基底下列的表述完全是不一样的。这就要求我们，在做矢量运算时，一定要注意两点：

1. 这个矢量是在哪一个基底下被描述的。
2. 各矢量之间的运算必须在同一基底下才是正确的。

上面所讲的内容就是矢量的基底变换。它主要应用于一些与矢量运算，矩阵运算相关的数据处理，人工智能问题。通过基底变换很多复杂的矩阵变得十分简单。节省大量的运算时间。

注意：标积中的两个矢量必须关于相同基底，不同基底必须转化成相同基底才能运算。

**矩阵**

# （选学）域（体）和环：

（代数学）域，有些书也称作体，是一个带有‘+’和‘\*’操作的特殊集合。并且这个集合里的任意元素都满足下列特性。

1. 加法特性：

A. 结合律

B. 交换律

C. 中性元素‘0’元素存在且满足0+a=a（中性元素就是说任何元素与其做+操作结果还是该元素本身）

D. 逆元素—a存在，即满足（任何元素的逆元素与该元素进行加操作结果都等于中性元素）

1. 乘特性

A. 结合律

B.交换律

C. 中性元素‘1’元素存在且满足1\*a=a（中性元素就是说任何元素与其做‘\*’操作结果还是该元素本身）

D. 逆元素存在：对于任意存在且满足：任何元素的逆元素与该元素进行加操作结果都等于中性元素）

1. 分配率（‘+’和‘\*’操作结合）

左分配律：

右分配律：

实数，有理数，和复数都是典型的域。

环，环是另一个代数概念，他的定义与体区别不多，唯一的不同在‘\*’操作上。环在乘特性处只满足结合律，也就是说他没有乘操作中性元素和逆元素，也不满足交换律。

整数是一个典型的环。

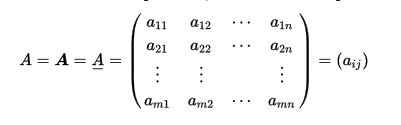
线性函数，

假设V，W 是两个关于域K（注：没有特殊说明本讲义的体是实数）的矢量空间，一个映射叫做线性映射（函数）。如果对于V中任意两个元素x和y以及K中元素a，f 满足下列条件：



矩阵是矢量空间的线性映射的一种表现形式。它是矢量线性映射最佳表现形式。因此直到现在他还是研究多元性线性问题的一种主要手段。

矩阵的书写：



其中（a11 a12 ... a1n）... (am1 am2 ... amn)是行矢量,(a11 a21 ... am1) ... (a1n a2n ... amn)列矢量。

矩阵的加减乘：

+：



例：



例：





减法和加法没有任何区别。因为A-B=A +（-B）

其它矩阵“+”操作：













\*：

1. 与标量相乘



例子



1. 矩阵相乘：

假设A,B,C是三个矩阵，且A\*B=C。则下列等式满足：

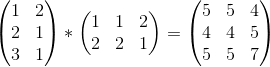


通过这个等式我们可以确定整个矩阵C的元素。

例：

IMG_256

例：



通常

IMG_256

常用的乘操作公式：  以及

是三个矩阵，那么下列等式总是有效。







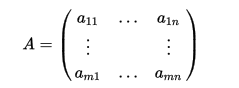
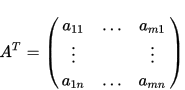
，

矩阵转置

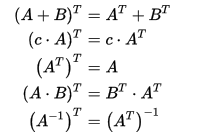
简单表示法

一般表示法：

其它与转置矩阵有关的公式：



逆矩阵

一个实数平方矩阵叫正则矩阵如果存在一个矩阵满足下列等式：



其中是单位矩阵。矩阵B是唯一确定，并被称为A的逆矩阵。A的逆矩阵通常用来表示。

逆矩阵常见的性质：







（通过这个公式，人们可以把难以求逆的矩阵分解成容易求逆的矩阵）







IMG_256

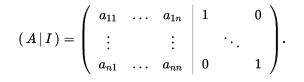


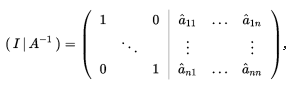




逆矩阵的确定：高斯-约旦算法

通过元素消元法把左边原矩阵先转化为三角矩阵。此时，如果矩阵的三角区出现“0”元素行，求解流程自动结束。



例子：

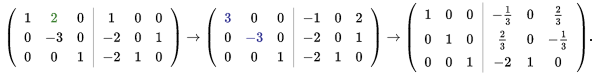


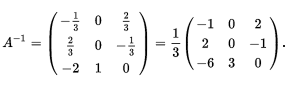


例子：









逆矩阵的确定：行列式法

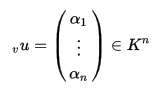
逆矩阵的确定：特征多项式法：

（选学）矩阵和线性函数的关系

假设K是一个域，V 和 W 是两个有穷维矢量空间，分别带有维数n和m。和是V和W的基底。这两个矢量空间能分别与标准坐标空间Kn和Km同构，这是因为对于任意都可以用下面唯一的等式表达：



等式的右边可以立马写成列坐标的形式：

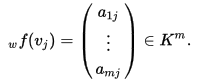


通过上面的等式，我们能容易地证明V和W与Kn和Km同构（一一映射并且是线性的）。

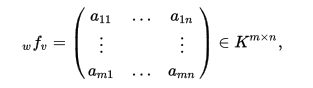
假设是一个线性映射，那么V的一个基底矢量的象可以用下面的等式来表达：



同样等式的右边也可以写成坐标的形式:



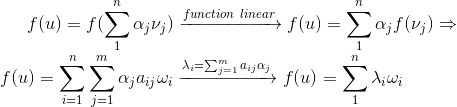
我们把每次求出的排成一排,,组成下面的一个矩阵.



IMG_256

我们也叫他映射矩阵.

下面我们来证实上面映射矩阵的构成是合理的。



n

m

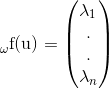
n

m

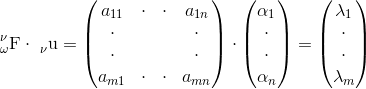
(\*)

IMG_256又可以写成坐标的形式:

gif



观察（\*）等式，我们可以立即把等式写成矩阵和矢量相乘的形式。



最小二乘法：

1. 普通最小二乘法

线性回归是回归分析法中最重要的一种，他的主要作用是通过最小二乘法为一系列试验点或者叫观测值寻找最佳拟合函数，尽量使观测值落到函数图像上或者在函数图像的周围。这个拟合函数通常是很多非线性函数线性组合而成，线性回归因此而得名。

**情况1**：我们首先从最简单的一元一次函数开始，假设



是最好的拟合n对给定测量值的直线。我们目的是根据这些测量值确定这个一元函数的系数和。最好拟合意味着，



有最小值。其中，

窗体顶端

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

被称作残差函数。

上面等式的含义可以通过下面的图来解释：

gif

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

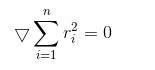
This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

窗体顶端

窗体顶端

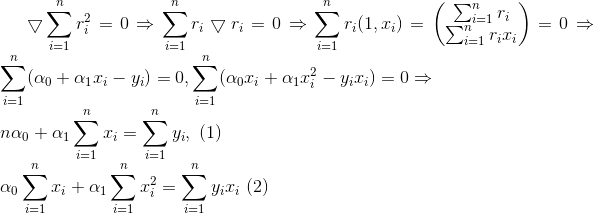
窗体底端

只有在



的情况下，我们才有可能得到最小值（二次函数项决定其只能有最小值）。上式化简可得：

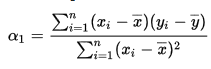
窗体顶端



求（1）和（2）等式可得：



还可以用下面的等式表示：



其中，。

**情况2：**上面的例子我们用的拟合函数是非线性的一次直线函数，其实更多时候我们也用多项式作为拟合函数，更确切地说，用多项式函数的线性组合来做拟合。

我们通常如下定义一个多项式：

窗体顶端

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

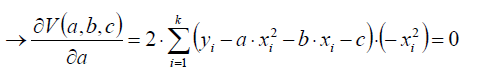
这里我们给大家一个简单的例子，我们把一个二次函数作为我们的拟合函数：



按照上面那个例子的思路，则有：

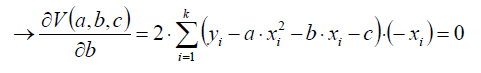


我们继续通过下面的全微分方程确定常系数a，b，c：

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

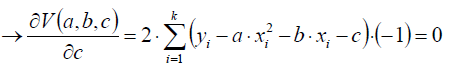
（4）

→



（5）

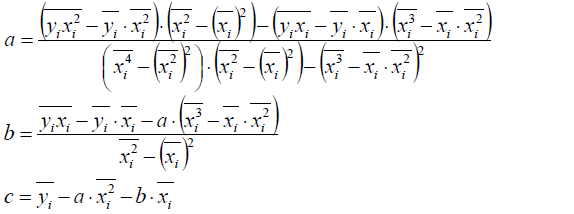
→



（6）

→

通过（4），（5）和（6）得



注：字母上面的横线代表平均值的意思。

**情况1的普通化：**

现在我们把直线方程拟合推广到直线方程线性组合函数拟合，直线方程的线性组合如下：

窗体顶端

窗体顶端

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

窗体底端

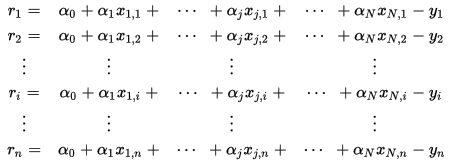
假定我们有n对观测值：



窗体顶端

其中This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.。我们把这些观测值矢量部分依次代入线性组合方程并与标量部分相减得：

窗体底端



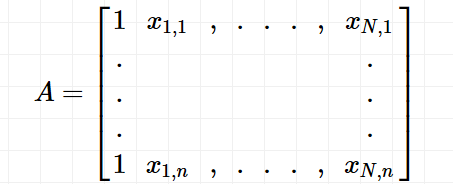
把上面的n个等式矢量化得：



其中r,A,ɑ,y有下列结构：

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

窗体底端



窗体顶端

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.是矢量，他的最小值就是他的2范数有最小值，即：



上面的等式要想有值存在，必须满足下面的全微分方程。



0

窗体顶端



在没有对上式求微分前，我们先介绍以下两个辅助等式：





借助这两个辅助等式（想知道辅助等式成立原因的同学电邮我！）对上式求微分，对应，对应y。



最终我们得到下列等式：



我们可以继续化简上面的方程，最终得到：

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

窗体顶端

**最普通化的线性组合**

我们接下来关注最为普通化的非线性或者线性函数的线性组合：



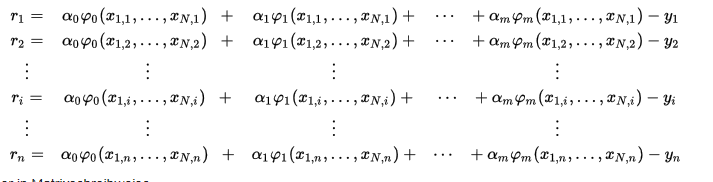
其中是非线性的或者是线性的，且各个之间是线性不相关的。

依靠给出的n对测量数据：



确定这个函数的，从而确定最佳拟合函数。

我们再次回到实验值与理论值之间的差：



对上面的等式矢量化得：



其中矩阵A的数据结构如下：



这个矩阵的元素值是的函数值，这个矩阵的行号是数据对次序编号，这个矩阵的列号是函数的次序编号。由于一共有n对数据和m+1个函数组成线性组合。因此这个矩阵是n\*m+1矩阵。

这个问题解决方法和上面问题的解决方法如出一辙，也是先求2范数的极值：

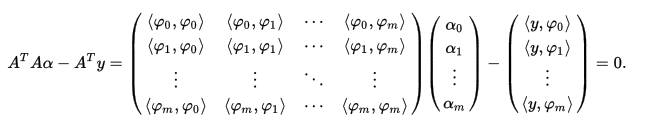
=0



也就是说下面等式成立：



人们常称上面等式为正规方程组，因为：



其中

因为A是n\*m+1矩阵，因此是一个m+1\*m+1的矩阵，如果矩阵A有全秩，也就是说A的秩是m+1，且n≥m+1的话，那么是正定矩阵且矢量唯一确定。唯一确定是因为A的秩是m+1，那么的秩也是m+1，也就是说是一一映射且满射，因此存在且唯一。下面我们仍需证明是正定的：

根据正定定义我们有：

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

因为是一一映射且满射, 因此:

窗体顶端

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

也就是说是正定的（hesse矩阵），更进一步来说我们所求的极值的确是最小值。

经过我们的努力，我们终于把这个复杂问题转化成求解线性方程组，通过求解方程组，最终确定值，从而确定整个线性组合函数。

最小二范数最小值的数值算法：

方程组



的求解方法有很多种，因为是对称的且正定的，这样的一个矩阵我们可选的数值算法有很多种，譬如：Cholesky-分解和CG方法，在对精度没有特别严格要求的话，且矩阵条件数比较低的话（可以粗糙地理解为低条件数的矩阵通常在计算中会有较高的计算精度）。这些方法是可以使用的。然而如果我们不幸运，遇到一个高条件数的矩阵A，那么的条件数将会以平方的形式增加，结果造成极其高的条件数，这种情况下，这些算法就完全不适用了。

这个问题的解决要归功于一个比较古老的算法，-QR分解。接下来我们简单地讲一下QR分解：

**QR分解与线性方程**

一个矩阵拥有一个唯一确定的简化型QR-分解。



其中是一个正交的矩阵，是一个上三角矩阵。

这种简化版分解可以扩展为完全QR分解：



如果通过增加正交的列  把 扩充成矩阵Q，通过增加0数值行把扩充成矩阵R，以致下列等式成立：



如果人们预先确定好矩阵的对称角元素符号，通常全为正，那么QR分解将会对任一且的矩阵A是唯一的。

QR分解的应用：

线性方程组的求解：

假定，并且rang（A）=n，我们可以通过以下算法来确定这个方程的解，也即确定的具体值：

1. 首先确定A的QR-分解。
2. 把A的QR分解代入线性方程组



1. R是三角矩阵，我们能迅速求解下列等式：



尽管我们可以通过上面方法求解，但并不能一定成功求出解，一个关于线性方程组有解、有多少解的判断规则有两种：

第一种：一般性规则，适合任何形式矩阵：





适合在没有进行求解之前对线性方程组是否有解，有解是否只有一个解进行判断。

第二种：分类判别规则

适合在没有进行求解之前对线性方程组是否有解，有解是否只有一个解或是无穷多解进行判断

* 平方矩阵：

如果平方矩阵的行列式不等于0有唯一解；

如果平方矩阵的行列式等于0，我们要看其副行列式：



{1,...,n}



上面等说是说所有的副行列式都等于0时，我们才说Ax=b有无穷多解；但是只要有一个行列式不为0，那么Ax=b无解。

* 超定矩阵

假如矩阵行数大于列数，我们称这种矩阵超定矩阵，通常他是无解的。但是也不能排除无解。假如有解（满足一般性判定的有解规则），通过上面的算法对矩阵进行QR分解过后，我们用最小算法思想来进行求解。具体内容见后面章节。

* 低确定矩阵

假定，那么A就是一个低确定矩阵，假如A有解，那么A的核一定含有非0矢量（因为A不可能是一一映射），并且Rang（A）= m，那么A一定是满射，这就说明非齐次方程的解一定是一个仿射子空间：



其中K是齐次线性方程组Ax=0的解组成的矢量子空间，也就是A的核。是任意一个非齐次方程组的解。此外，带有最小二范数的解（即矢量x的模有最小值）位于矩阵核的正交补，这样的解可以通过AT的QR分解来确定。

* 确定AT的QR分解：
* 确定的解



* 确定

注意：由于低确定矩阵很少用到数据分析上，由于对于它的介绍就到此为止！

下面我们简要的阐述下带有最小二范数的解位于矩阵核的正交补的原因：

通常我们把用下列集合表示核的正交补：



假设v是带有最小范数的解，则：



因为取最小值，因此必须小于0，这就是说常矢量和之间的夹角是钝角。





从上图我们很快发现，对于常矢量，只有在变量矢量和相互垂直时，有最小长度，也就是说有最小2范数。

**如何计算QR分解：**

豪斯霍尔德变换：

在欧几里德空间里豪斯霍尔德变换描述的是一个矢量关于一个通过0矢量的超平面（可看做镜子）的反射。(反射是把一个物体变换成它的[镜像](https://zh.wikipedia.org/wiki/%E9%95%9C%E5%83%8F)的[映射](https://zh.wikipedia.org/wiki/%E5%87%BD%E6%95%B0)。要反射一个平面图形，通常需要“镜子”是一条直线（反射轴）此时我们称这种反射为轴反射，对于三维空间中的反射通常就要使用[平面](https://zh.wikipedia.org/wiki/%E5%B9%B3%E9%9D%A2)作为镜子。此时我们称这种反射为面反射。)

对于超过三维的空间，反射的镜子，也就是反射面是超平面。这个反射面可以通过这个反射面的一个法线矢量v来定义。

假设这个法线矢量v是一个列矢量，并且已经事先给出。那么通过这个矢量v我们可以定义一个反射：



其中是行矢量，是外积。

假设v是单位矢量，则：



把H操作作用到变量x：



几何意义：

<v, x>v

**反射面**

v

x - <v, x>v

x

- <v, x>v

Hx

- <v, x>v

豪斯霍尔德的特性：

对称性：



正交性：



对合性



H的特征值和特征向量

1. v是特征向量，-1是特征值



1. 假设V是v形成的一维子空间V的正交补，



那么中任意u是特征向量，1是特征值，尤其是的基底（w1,...wn-1）。也就是说是特征向量空间。



通过0矢量，且它的集合表达式正是通过0点超平面的法线表达式（或称作正交式）。这就说明它是H的超平面反射面。另外，可逆H把任意 分解映射到V和，这就意味着H的象是V和其正交补的内直合。





豪斯霍尔德变换应用：QR分解

第一步：构造反射矩阵：

给定一个任意矢量，找到一个矢量u，构造一个关于u的反射到单位矢量e的反射矩阵H，使



我们通过角平分线的方法找到法线矢量u

**u**

**u**

**v**

**u**

**u**

**-e**

**e**

我们分别在通过矢量v方向上的直线和矢量e方向上的直线以及矢量v方向上的直线和矢量 -e方向上的直线围成的角里做角平分线，在这些角平分线上寻找我们所需的u；为了保证角平分线方向的正确性，我们要求e方向上矢量长度为，即：



通过u我们可以定义一个反射：



下面我们来验证：



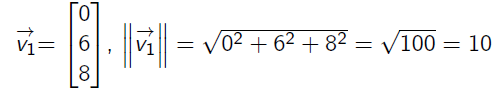
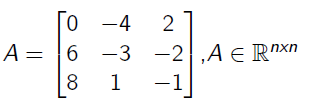
通常在实际计算中，我们选择e为标准矢量。那么根据上图我们可以把两个法线矢量u按如下方式分类求解：

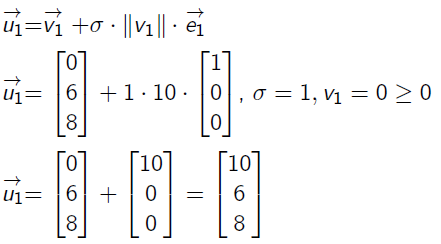
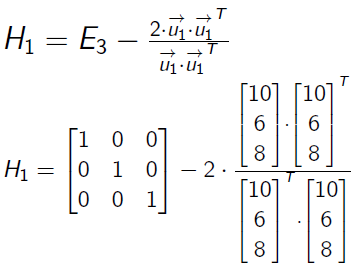


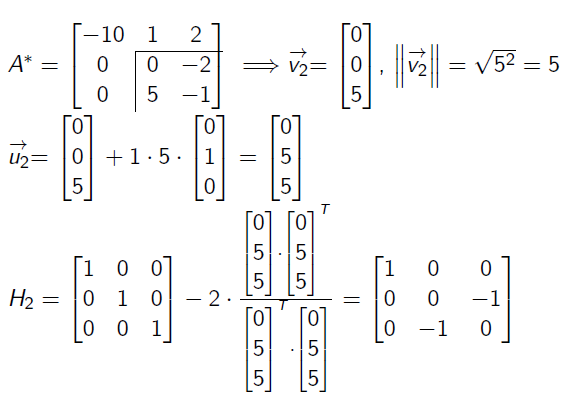
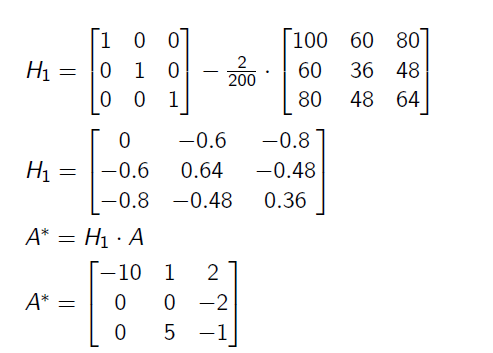
上面分类也可以把加号变成减号，减号变成加号。

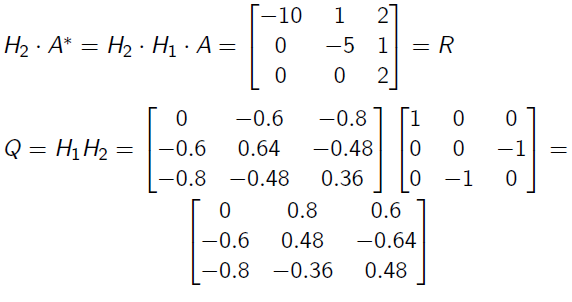
第二步，举例一个3X3 矩阵的QR分解：

给定矩阵：







我们下面用u的另一种表示方法借助Python来进行QR分解：



H\_1=np.identity(3)-(2/200)\*np.array([[100,-60,-80],[-60,36,48],[-80,48,64]])

A\_0=np.array([[0,-4,2],[6,-3,-2],[8,1,-1]])

A\_1=np.dot(H\_1,A\_0)

#计算出H1和A1

A\_1

Out[8]:

array([[ 10. , -1. , -2. ],

[ 0. , -4.8, 0.4],

[ 0. , -1.4, 2.2]])

u\_2=(np.array([0,-4.8,-1.4]))-(np.sqrt(4.8\*\*2+1.4\*\*2)\*np.array([0,1,0]))

#计算u2

c\_1=np.array([0,0,0])

c\_2=np.array([0,-9.8,-1.4])\*-9.8

c\_3=np.array([0,-9.8,-1.4])\*-1.4

newarray=np.column\_stack((c\_1,np.column\_stack((c\_2,c\_3))))

newarray

Out[26]:

array([[ 0. , -0. , -0. ],

[ 0. , 96.04, 13.72],

[ 0. , 13.72, 1.96]])

#计算u2.u2.T外积

H\_2=np.identity(3)-(2/np.dot(np.array([0. , -9.8, -1.4]),np.array([0. , -9.8, -1.4])))\*newarray

#计算H2

R=np.dot(H\_2,A\_1)

R

Out[48]:

array([[ 1.00000000e+01, -1.00000000e+00, -2.00000000e+00],

[ 0.00000000e+00, 5.00000000e+00, -1.00000000e+00],

[ 0.00000000e+00, -2.22044605e-16, 2.00000000e+00]])

Q=np.dot(H\_1,H\_2)

Q

Out[50]:

array([[ 0. , -0.8 , 0.6 ],

[ 0.6 , -0.48, -0.64],

[ 0.8 , 0.36, 0.48]])

#验证正交矩阵的正交性

np.dot(Q[:,0],Q[:,1])

Out[52]: 0.0

np.dot(Q[:,2],Q[:,2])

Out[54]: 1.0

np.dot(Q[:,0],Q[:,2])

Out[55]: 0.0

np.dot(Q[:,1],Q[:,1])

Out[56]: 1.0000000000000004

验证QR分解是否正确

np.dot(Q,R)

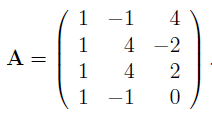
Out[58]:

array([[ 0., -4., 2.],

[ 6., -3., -2.],

[ 8., 1., -1.]])

下面我们再举一个4X3的矩阵：



A=np.array([[1,-1,4],[1,4,-2],[1,4,2],[1,-1,0]])

modA\_0=np.dot(A[:,0],A[:,0])

u\_1=A[:,0]-np.sqrt(modA\_0)\*np.identity(4)[:,0]

u\_1

Out[68]: array([-1., 1., 1., 1.])

H\_1=np.identity(4)-(2/np.dot(u\_1,u\_1))\*np.column\_stack((u\_1\*u\_1[0],np.column\_stack((u\_1\*u\_1[1],np.column\_stack((u\_1\*u\_1[2],u\_1\*u\_1[3]))))))

H\_1

Out[72]:

array([[ 0.5, 0.5, 0.5, 0.5],

[ 0.5, 0.5, -0.5, -0.5],

[ 0.5, -0.5, 0.5, -0.5],

[ 0.5, -0.5, -0.5, 0.5]])

A\_1=np.dot(H\_1,A)

A\_1

Out[75]:

array([[ 2., 3., 2.],

[ 0., 0., 0.],

[ 0., 0., 4.],

[ 0., -5., 2.]])

v\_2=np.array([0,0,0,-5])

u\_2=v\_2-np.sqrt(np.dot(v\_2,v\_2))\*np.identity(4)[:,1]

u\_2

Out[81]: array([ 0., -5., 0., -5.])

H\_2=np.identity(4)-(2/np.dot(u\_2,u\_2))\*np.column\_stack((u\_2\*u\_2[0],np.column\_stack((u\_2\*u\_2[1],np.column\_stack((u\_2\*u\_2[2],u\_2\*u\_2[3]))))))

H\_2

Out[84]:

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

[ 0., 0., 0., -1.],

[ 0., 0., 1., 0.],

[ 0., -1., 0., 0.]])

A\_2=np.dot(H\_2,A\_1)

R=A\_2

R

Out[86]:

array([[ 2., 3., 2.],

[ 0., 5., -2.],

[ 0., 0., 4.],

[ 0., 0., 0.]])

Q=np.dot(H\_1,H\_2)

Q

Out[93]:

array([[ 0.5, -0.5, 0.5, -0.5],

[ 0.5, 0.5, -0.5, -0.5],

[ 0.5, 0.5, 0.5, 0.5],

[ 0.5, -0.5, -0.5, 0.5]])

A\_0=np.dot(Q,R)

A\_0

Out[96]:

array([[ 1., -1., 4.],

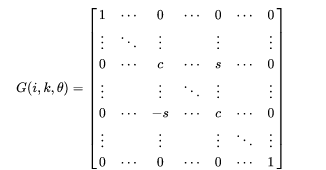
[ 1., 4., -2.],

[ 1., 4., 2.],

[ 1., -1., 0.]])

吉文斯旋转变换：

这个变换是通过下面的一个正交矩阵来完成的：



这里的 c = cos(θ) 出现在第i行和第i列交叉点以及第k行和第k列交叉点，s = sin(θ) 出现在第 i行和第k列交叉点上，-s= -sin(θ)则出现在第 k行和第 i列的交叉点上。1处在所有不等于（i,i）和(k,k)矩阵G(i,k,θ)的对角线上。

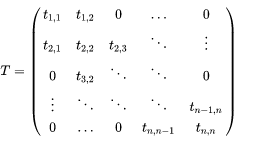
乘积 G(i, k, θ)Tx 表示向量 x 在 (i,k)平面中的逆时针旋转 θ 弧度。因此人们给它命名为吉文斯旋转。

吉文斯旋转变换可用于矩阵的QR分解。尤其适合下列矩阵类型：

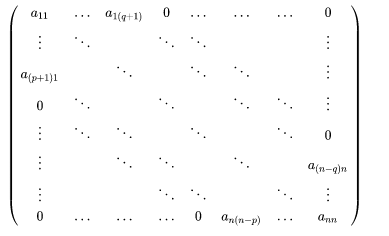
1. 稀疏矩阵

稀疏矩阵（Sparse matrix），是其元素大部分为零的矩阵。反之，如果大部分元素都非零，则这个矩阵是稠密的.

1. 三对角矩阵和带状矩阵

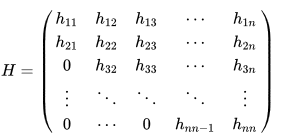


三对角矩阵



带状矩阵

1. 海森伯格矩阵

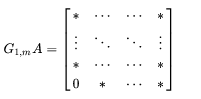
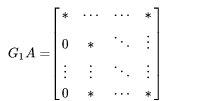


下面我们通过一个例子来说明它的算法

假如给定一个矩阵，如果把转变成0，那么它的吉文斯旋转矩阵是：



我们从第一列开始，确切地说从第一列最下面一个不为0的元素开始，依次利用上面公式把转化成0（假如这里的某一个Ai,1为0，自动跳过不做运算），与此对应需要用的吉文斯矩阵分别为。

其中

然后对第二列最下面一个不为0的元素开始，即把转化为0，与此对应需要用的吉文斯矩阵分别为：



由此我们得到变换第二列的总矩阵：



依次类推，最终通过下列等式



得到一个三角矩阵R。

与此对应的Q矩阵是：



算法的python实现如下：

In [**1**]: def generate\_matrix(array\_var\_1,array\_row\_v,array\_column\_v):

   ...: n\_dim=max(np.shape(array\_var\_1))

   ...: array\_1=np.zeros((n\_dim,n\_dim))

   ...: rou\_1=(array\_var\_1[array\_row\_v,array\_column\_v])/(np.sqrt(array\_var\_1[array\_column\_v,array\_column\_v]\*\*2+array\_var\_1[array\_row\_v,array\_column\_v]\*\*2))

   ...: #print(rou\_1)

   ...: array\_1[array\_row\_v,array\_column\_v]=-rou\_1

   ...: #print(array\_1[array\_row\_v,array\_column\_v])

   ...: array\_1[array\_column\_v,array\_row\_v]=rou\_1

   ...: rou\_2=(array\_var\_1[array\_column\_v,array\_column\_v])/(np.sqrt(array\_var\_1[array\_column\_v,array\_column\_v]\*\*2+array\_var\_1[array\_row\_v,array\_column\_v]\*\*2))

   ...: #print(rou\_2)

   ...: array\_1[array\_column\_v,array\_column\_v]=rou\_2

   ...: array\_1[array\_row\_v,array\_row\_v]=rou\_2

   ...: for i in range(n\_dim):

   ...: if (i!=array\_row\_v) & (i!=array\_column\_v):

   ...: #print(array\_row\_v,array\_column\_v)

   ...: array\_1[i,i]=1

   ...: return array\_1

def QR\_Given(array\_var):

import sys

List\_array=[]

indi\_array=array\_var

if array\_var.shape[0]>=array\_var.shape[1]:

for j in range(np.shape(array\_var)[1]):

for i in range(np.shape(array\_var)[0]-1,j,-1):

#print(i,j)

rou=np.sqrt(indi\_array[j,j]\*\*2+indi\_array[i,j]\*\*2)

if rou==0:

break

elif indi\_array[i,j]==0:

continue

else:

G\_ji=generate\_matrix(indi\_array,i,j)

#print(G\_ji)

List\_array.append(G\_ji)

indi\_array=np.dot(G\_ji,indi\_array)

else:

print('不合规矩阵，请把行数设置成大于或等于列数的矩阵')

sys.exit(1)

R=indi\_array

Q\_1=List\_array[-1].T

for k in range(-2,-len(List\_array)-1,-1):

Q\_1=np.dot(List\_array[k].T,Q\_1)

return(Q\_1,R)

格拉姆-施密特正交化

第一步：Q矩阵的确定

假设 是已经正交化的矢量（这些矢量是矩阵的列），通过我们计算新的正交矢量:

# 。

其中：





矢量系统正交化的条件：

待正交化的矢量系统必须是线性非相关矢量系统。

通过上面的方法我们可以把一个矩阵的列转化为一组正交化矢量系统，这就是说我们可以借助矩阵正交化得到QR分解中的Q矩阵。

第二步：R矩阵的确定

我们先确定R矩阵对角线上的元素：

，

其中



以及矩阵其它位置上的元素：

。

对于i ≠j

Python 实现：

def QR\_zerlegung(arr\_para):

import sys

recycle\_index=0

sum\_col\_indirect=0

arr\_carrier=[]

correct\_list=[]

arr\_row=arr\_para.shape[0]

arr\_col=arr\_para.shape[1]

arr\_rank=np.linalg.matrix\_rank(arr\_para)

if (arr\_row<=arr\_col) or (arr\_rank!=arr\_col):

print('不合规矩阵，请把行数设置成大于或等于列数的矩阵,且列为线性无关矢量系统')

sys.exit(1)

else:

for col\_index in range(arr\_col):

if col\_index==0:

new\_col=arr\_para[:,col\_index]/np.linalg.norm(arr\_para[:,col\_index])

norm\_1=arr\_para[:,col\_index]

#print(new\_col)

else:

# print(new\_col)

arr\_carrier\_array=np.array(arr\_carrier)

sum\_col\_indirect=0

for c\_index in range(recycle\_index):

sum\_col\_indirect=sum\_col\_indirect+(np.dot(arr\_carrier\_array[c\_index],arr\_para[:,col\_index]))\*arr\_carrier\_array[c\_index]

#print(c\_index,sum\_col\_indirect,arr\_carrier\_array[c\_index],arr\_para[:,col\_index])

new\_col=(arr\_para[:,col\_index]-(sum\_col\_indirect))/(np.linalg.norm((arr\_para[:,col\_index]-(sum\_col\_indirect))))

norm\_1=(arr\_para[:,col\_index])-(sum\_col\_indirect)

#print(norm\_1)

arr\_carrier.append(list(new\_col))

correct\_list.append(list(norm\_1))

#print(correct\_list)

recycle\_index+=1

NormQ\_arr=np.array(correct\_list).T

#print(Q\_arr\_withoutnorm)

Q\_arr=np.array(arr\_carrier).T

R\_in\_operation=np.zeros((arr\_col,arr\_col))

for row\_ar in range(arr\_col):

for col\_ar in range (arr\_col):

if col\_ar==row\_ar:

R\_in\_operation[col\_ar,col\_ar]=np.linalg.norm(NormQ\_arr[:,col\_ar])

while col\_ar<arr\_col-1:

col\_ar+=1

R\_in\_operation[row\_ar,col\_ar]=np.dot(Q\_arr[:,row\_ar],arr\_para[:,col\_ar])

return Q\_arr,R\_in\_operation

a1=np.array([[1,-1,4],[1,4,-2],[1,4,2],[1,-1,0]])

QR\_zerlegung(a1)

Out[11]:

(array([[ 0.5, -0.5, 0.5],

[ 0.5, 0.5, -0.5],

[ 0.5, 0.5, 0.5],

[ 0.5, -0.5, -0.5]]), array([[ 2., 3., 2.],

[ 0., 5., -2.],

[ 0., 0., 4.]]))

我们终于结束了我们的QR项目，我们只所以花费这么多篇幅去讲解这个算法是因为它不仅是数值计算中最基础最重要的算法之一，而且与后续机器学习算法息息相关。因此对其做比较有深度的了解还是很有必要的。

当然，还有很多数值算法也是对数据分析十分重要的，但是我们时间有限，只能把最重要的讲给大家。

接下来我们继续关注通过QR来求解最小二乘法：



如果Q是正交矩阵。

同理，我们也可以得到：



接下来，我们对A进行QR分解可得：

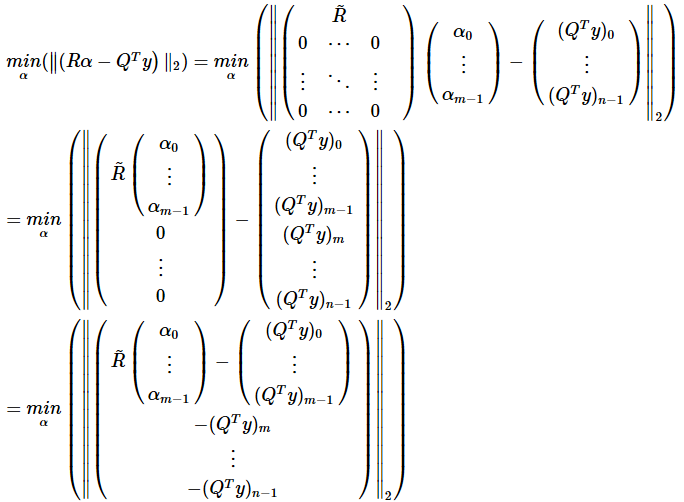
。

其中R大都是一个行大于列的矩阵，我们假设，那么作为上三角矩阵，R有下列外形：

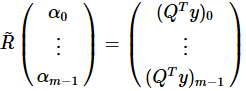


带有。

通过上式的帮助我们很容易的求出使范数2达到最小值时的α。



如果要想使二范数拥有最小值，很显然下列等式必须被满足：



这个等式的求解十分简单，我们能够迅速得得到答案，即的值，最终确定我们的数据拟合函数。

至于这个最小二范数的大小很显然取决于,...,。

1. **统计学中的最小二乘法—普通和广义最小二估计：**

在统计学回归分析中对于多个事先给定变量（数据）的线性回归问题人们称之为多重线性回归。对于一个多重线性模型的估计，常用的方式是最小二估计（英语：ordinary least squares；缩写：OLS）。与OLS相对应的是广义最小二估计GLS （generalised least squares）。在GLS方法中人们把一个带有广义误差结构的多重线性模型应用到问题的解决中。**关于GLS和OLS我们这里不做讲解，因为他相对复杂，并且纯粹以统计学的角度来分析回归问题，这对只有较少统计学知识的学生来说理解起来非常吃力。因此它的具体详细描述将会出现在数据分析三中。**

1. **带有附加条件的最小二乘法：**

在很多情况下，我们会遇到带有附加条件的最小二乘法问题。这些附加条件可以是等式也可以是不等式。对于等式附加条件我们可以直接把它应用到最优化问题上以达到降维的目的。（常用方法有Lagrange算子法）对于带有不等式附加条件的最小化问题，解决起来就相当麻烦。常见的带有不等式附加条件的最小化问题表达如下：



这样表达是一个典型的凸二次规划（或称优化）问题（如果我们假定f=Aα中的A是正定矩阵的话）。

**他的解决目前最为有效的方法是Active-Set-方法。这种算法十分难，并且属于数学优化法内容，因此我们这里不做深入探讨。具体算法内容会出现在数据分析三中。**

1. **非线性最小二乘法**

**3.1 高斯牛顿法**

随着高效率计算机蓬勃发展非线性最小二乘法显得日益重要。在这里函数中的参数是非线性的，例如：







以上函数中的参数都是以非线性的方式出现在函数中。

但是



=

上式的参数确是以线性的方式出现在等式中，他就是我们上面所讲的线性最小二乘法的范畴。

非线性模型的最小二乘法可以通过高斯牛顿法进行求解。如在线性模型那里一样，非线性最小二算法也是一个关于最小二范数的问题。

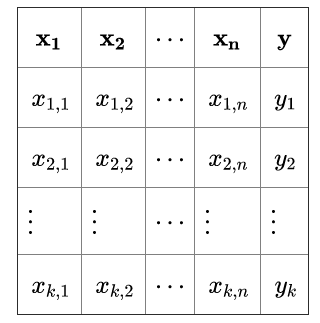


其中，



高斯牛顿法的基础知识：

1. 假如我们做k次测量，则会有k个测量值，每个测量值由n个特征，这就意味着每个测量值都是n维矢量；见下图：



1. 建立能够描述和之间关联的模型函数。这个函数通常有个与函数值非线性相关的参数。我们的目标是通过确定这些参数来确定能最大程度的拟合y值的函数。通常情况下我们要求p≤k。
2. 我们通过确定。

确定的算法（高斯牛顿法）：

1）给出带有待定参数的函数，函数选择或者说线性模型的建立多以数据散点分布形式和个人经验。

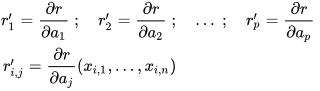
1. 列出残差函数r和残差矢量**r**



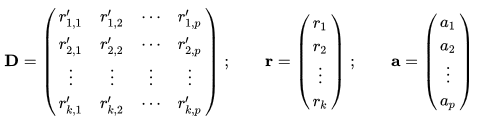
和

带有***r***

1. 计算残差函数对参数的一阶偏微分。



1. 构造迭代矩阵



这里***D***是矢量残差函数***r***对矢量***a***雅克比矩阵（多元矢量函数的求导或者叫多元多维函数求导）。

对迭代矩阵的几点解释：

* 矩阵中每个元素索引的列位对应的是矢量变量***a***中的元素索引，索引从1到p；行位对应的数据的顺序编号。比如，我们一共有k个数据，编号从1到k。
* 要注意矩阵中列的先后顺序与的顺序保持一致，比如第一列对应的是对a1的偏微分，以此类推。
* 单个已知数据链的长度n不影响***D、r、a***的构建。
* 在开始迭代之前，参数的起始值和误差范围C，一个大于0的值，必须预先确定。

1. 迭代

* 矩阵迭代等式构建：



注意这里的索引i和i+1代表的是步骤的含义。后一步骤是前一步骤的改善。但是在计算中我们经常用上式的等价式：



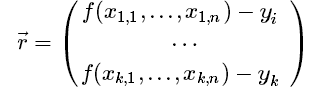
（比较线性回归公式，那就会发现迭代等式的本质）

等式右边可以通过已知条件进行求解。这样整个等式就变成了一个关于矩阵的非齐次线性方程组。这样转变的好处计算精度高速度快。（）

* D的计算：

先求出，然后把第一行数据（参见上面表格）代入到中得：。然后再把第二行数据代入到得：。依次类推求出矩阵的第一列数据。然后我们把数据按照上面的方法带入到，得出矩阵***D***的第二列，依次类推，得出整个矩阵***D***。

* ***r***的计算：



* 如果，迭代终止。（意味着迭代对矢量**a**i的改善已经十分有限，换句话说迭代无法造成矢量**a**i的明显改变）

备注：

1. 行数必须大于列数
2. 是正定且对称矩阵，因此十分适合用Cholesky分解来求解由它创建的线性方程组。Cholesky分解是一个相对简单矩阵分解方法，**会在数据分析中级讲解。**
3. 为了达到拟合的快速收敛，应该对参量的初始值的量级有粗略估计。不恰当估计会造成结果发散。
4. C越趋近于0，此方法越能精确地确定，然而迭代次数会增加。

**3.2 莱文贝格－马夸特方法**

这个方法也是用来解决不带附加条件的非线性模型的最小二问题。他是高斯牛顿方法的变种，也可以说是一种改进算法。尽管它和高斯牛顿方法一样不能保证方法的收敛性（无法收敛就意味着函数无法拟合数据点）

我们再次回到最小二表达式：



确定当范数2的平方达到最小值时，参数矢量的值。

第一步，确定起始值，和迭代结束值C（精度）

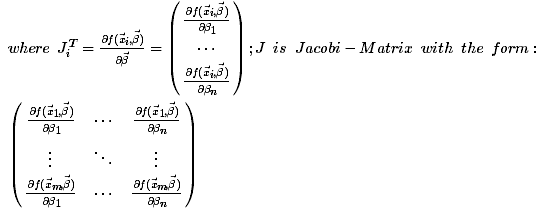
选择均匀标准的起始值，有利于数值的计算。例如：



第二步，确定莱文贝格马夸特算法核心等式

在点的泰勒级数近似：

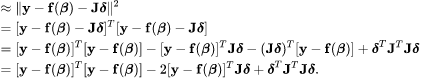




通过上面的泰勒近似我们可以得到：



继续把和写成矢量的形式：



是范数2的平方值，当它取最小值时，满足下列等式：



借助和我们得到：



加入阻尼因子得到莱文贝格算法核心等式：



阻尼因子是非负值数，并且每一次迭代都要被调整，如果***S***变小迅速，（减小λ影响力，）我们选择小的阻尼因子（接近高斯-牛顿方法）。反之，我选择大的阻尼因子（增大影响力。）（最速下降法）。

阻尼因子取决于初始问题的规整与标准。马夸特建议引入一个起始值λ0，和一个因子V〉1。初始设置λ = λ0，从起始点出发，一步以后，计算S（β1），第二步从λ0/ν. 出发计算计算S（β2），两者与S（β0）比较，如果都大于S（β0），那么我们通过与v的连乘增加阻尼因子直到达到满意的结果。即：选择合适的k，以致使λ0νk作为阻尼因子，S有令人满意的变化。

如果在第二步后，使用阻尼因子λ0/ν 已经使S（β2）减小，那么这个阻尼因子可以作为新的阻尼因子继续迭代算法，如果使用λ0/ν没有使S减小，反而使S增大，但是λ0却得到小的S（β0），那么对阻尼因子不在做任何改变，直接使用初始λ0优化下去。

第三步，迭代

* 通过上式我们依据起始值***β***0求出***δ***0，然后由此得出***β***1=***β***0 +***δ***0，然后再把***β***1代入上面等式求出***δ***1，两者相加求出***β***2=***β***1 +***δ***1，依次类推直到。迭代结束
* 或者我们也可以通过S变化来结束迭代：



对于大的λ，莱文贝格算法显现出他的不足之处，的逆矩阵根本无法使用。通过用矩阵的对角元素构成的对角矩阵替代矩阵I我们可以完美解决这个问题。



上面的等式就是马夸特等式。它是莱文贝格的改进型。

相关函数介绍：

scipy.optimize.least\_squares(fun, x0, jac='2-point', bounds=(-inf, inf), method='trf', ftol=1e-08, xtol=1e-08, gtol=1e-08, x\_scale=1.0, loss='linear', f\_scale=1.0, diff\_step=None, tr\_solver=None, tr\_options={}, jac\_sparsity=None, max\_nfev=None, verbose=0, args=(), kwargs={})

此函数可以解决带有变量约束条件和无约束条件的非线性问题。

残差f(x)和损失函数rho(x)必须事先给定，最小二乘法函数least\_squares找到成本函数F(x)一个局部最小值。

minimize F(x) = 0.5 \* sum(rho(f\_i(x)\*\*2), i = 0, ..., m - 1)

subject to lb <= x <= ub

损失函数的目的是减少异常解的影响。

**参数：**

1. **fun：callable**

计算残差矢量的函数（也即fi(x)），函数表达：fun(x, \*args, \*\*kwargs)，也就是说，整个计算过程都是关于它的第一个参数的最小化。函数的变量x必须是一个长度为n（shape(n)）的一维数组（X=X1,,...Xn永远不会是标量，即使是n=1，他也是数组）。返回值是一个一维的长度为shape(n)的array\_like或者一个标量。如果变量x是复数或者函数fun返回复数残差，那么这些复数必须打包（被改变）成实参的实函数：

注意：X1,,...Xn是我们前面算法介绍中的待定参数，不是事先给定的数据点。

例子：

from scipy.optimize import least\_squares

import numpy as np

#定义残差函数，已知数据点y=0.5+0.5j 和是数值1，待定参数是“z”.

def f(z):

return z-(0.5+0.5j)

#定义打包函数

def f\_wrap(x):

fx = f(x[0] + 1j\*x[1])

return np.array([fx.real, fx.imag])

#把输入的复数变成一维shape(2)array\_like, 输出同样被打包成array\_like。

res\_wrapped = least\_squares(f\_wrap, (0.1, 0.1), bounds=([0, 0], [1, 1]))

z = res\_wrapped.x[0] + res\_wrapped.x[1]\*1j

z

Out[10]: (0.49999999999925893+0.49999999999925893j)

#输入函数，参数起始值，约束条件，算出z。

1. ***x0 : array\_like with shape (n,) or float***

独立变量的初始值。如果是浮点数，它经常是单元素一维数组

1. ***jac : {‘2-point’, ‘3-point’, ‘cs’, callable}, optional（微分很难求，或者函数是由数据确定）***

计算雅可比矩阵（m行n列）矩阵中的元素（i,j）是函数f[i]}对函数x[j]的偏微分。大括号内的关节字描述的是数值微分（数值估计有限差分）。三点法相对精确，但与两点法相比需要更多2倍于计算步骤。Cs应用复数步长微分，直到可能的最大精度；当然这里fun是复函数且在整个复平面是解析函数。

注意：Method ’Im’总是使用2点法计算的Jac

如果选择callable法，就是调用函数jac(x, \*args, \*\*kwargs)，它应该返回一个比较好的或者精确地雅克比矩阵。

**bounds** : 2-tuple of array\_like, optional

独立变量的下界和上界约束。默认是没有约束。元组中的元素，如果是数组，必须匹配X0，如果是标量意味着一个约束为所有变量。利用正负np.inf可以消除所有或部分变量的约束。

**Method:** {‘trf’,’dogbox’,’im’},optional

1. 置信域反射算法，适合带约束条件的问题优化，尤其是大型稀疏（雅可比矩阵）问题的优化。（带有约束条件的优选法问题，算法很新，直到现在，还没有收集到比较详细完善的资料，schlüsselpoint：Vergleichen Taylorreihe von S（β）mit m（x））
2. 矩形置信域Dogleg 算法，适合带有约束的小数据问题，不推荐应用到雅可比矩阵不是满秩问题上。(算法理解需要很多优选法的知识，在数据分析二中二次模型优化有介绍）
3. ‘im’ Levenberg-Marquardt算法不能处理稀疏雅可比矩阵。一种解决相对较小的（数据）非强迫问题的最高效的方法。

**ftol :** float, optional

公差，借助于成本函数的变化，它可判断是否终止优化，默认1e-8. 如果成本函数变化满足 dF < ftol \* F, 优化过程结束。并且局部二次模型与数据模型达到充分的一致性。

**xtol** : float, optional

公差，借助于独立变量的变化，它可判断是否终止优化，默认1e-8。具体判别方式依赖于所应用的method。

1. 对于’trf’和dogbox：norm(dx) < xtol \* (xtol + norm(x))
2. 对于’Im’：Delta < xtol \* norm(xs),Delta是置信域半径，xs参看x\_scale。

**x\_scale :** array\_like or ‘jac’, optional

变量的特征尺度，设置x\_scale等同于用缩放变量xs = x / x\_scale重新定义中的问题。其本质通过它实现变量标准化。

**loss** : str or callable, optional

定义损失函数。常用关键字如下：

‘linear’(default):rho(z) = z. 标准最小二算法问题

‘soft\_l1’ : rho(z) = 2 \* ((1 + z)\*\*0.5 - 1). 用于稳健性最小二算法

‘huber’：rho(z) = z if z <= 1 else 2\*z\*\*0.5 - 1. 类似于‘soft\_l1’。

cauchy’ : rho(z) = ln(1 + z). 严重削弱了离群值的影响，但可能会在优化过程中造成困难。

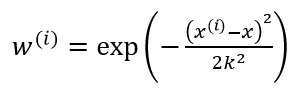
‘arctan’ : rho(z) = arctan(z).限制单个残差的最大损失，具有类似于“柯西”的性质

**返回值：**

例子x.（“The MINPACK-2 Test Problem Collection”.）

1. **加权回归**

加权回归是对每个数据点通过下面exp函数计算出权重：



**X —** 将要预测的点。

 — 与相对应的权重，它代表需要预测的数据仅与到事先给定数据的距离有关，距离越近，关系越大，反之越小。

k — 是波长参数，它控制了权值随距离下降的速率，该函数形式上类似高斯分布(正态分布)，但并没有任何高斯分布的意义。

通过线性回归损失函数J(θ):



我们可以求出：



如同在普通最小二算法那里做的一样，我么最终可以求出：



其中W是一个对角线上含有1的平方对角矩阵。维度为m×m，如果有m个特征数据。

缺点：由于对于每一个要预测的点，都要重新依据整个数据集计算一个线性回归模型出来，使得算法代价极高。

Python实现加权预测：

**训练阶段：**

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

from scipy import linalg

tita\_list=[]

def upload\_csv(name):

array\_data=np.loadtxt(fname=name,dtype=np.float64,delimiter=',')

array\_data\_x=array\_data[:,:array\_data.shape[1]-1],array\_data\_y=array\_data[:,array\_data.shape-1]

return array\_data\_x,array\_data\_y

def singel\_predition(predition\_point,Arrpara\_x,Arrpara\_y,k):

X= np.column\_stack((np.ones( Arrpara\_x.shape[0]),Arrpara\_x))

#print(X)

Y= Arrpara\_y

#print(Y)

m = np.shape(X)[0]

weights = np.identity(m)#生成单位矩阵

for i in range(m):

#print(predition\_point,Arrpara\_x[i])

diff\_all\_points = predition\_point - Arrpara\_x[i]

#print(diff\_all\_points)

weights[i,i] = np.exp((diff\_all\_points\*\*2)/(-2.0 \* k\*\*2)) # 权值计算和生成权值矩阵

#print(weights[i,i])

#print(weights)

XWX = np.dot(X.T,np.dot(weights,X))

#print(XWX) #对x值进行加权计算

if linalg.det(XWX) == 0.0:

#print('singular Matrix')

return

tita = np.dot(linalg.inv(XWX),np.dot(X.T,np.dot(weights,Y)))# 计算回归系数

tita\_list.append(tita)

return np.dot (np.hstack((np.array([1]),np.array([predition\_point]))),tita)#返回y值

#以上是对一个点的加权预测：

def other\_point\_prediction(array\_x,array\_y,k):

tita\_list=[]

m=np.shape(array\_x)[0]

null\_array=np.zeros(m)

for i in range(m):

null\_array[i]=singel\_predition(array\_x[i],array\_x,array\_y,k)

return null\_array

以上是多点加权

数据的调选，开始训练

Data1=pd.read\_csv(r'Desktop\bankloan.csv',delimiter=',')

del Data1['Unnamed: 9']

Data2=Data1.values

D\_X=Data2[:501,4];

D\_Y=Data2[:501,5]

通过图像训练出好的**k**

from matplotlib.animation import FuncAnimation#引入函数动画、

import seaborn as sns#美化图形包

sns.set\_style("whitegrid")

#设置图像主图风格（白色格子）

fig, ax = plt.subplots()

fig.set\_tight\_layout(True)#紧凑的布局

#单个作图

#tita\_list=[]

DY\_pre=other\_point\_prediction(D\_X,D\_Y,10)

DY\_pre\_pd=pd.Series(DY\_pre)

DY\_pre\_pd[DY\_pre\_pd.isnull()]=0

DY\_pre\_pd

DY\_Pre\_ar=DY\_pre\_pd.values

#分别作图

ax.scatter(D\_X,D\_Y,c='green',marker='\*')

line=ax.plot(D\_X,DY\_Pre\_ar)

Plt.show()

for k in range(1,20,1):

fig, ax = plt.subplots()

fig.set\_tight\_layout(True)#紧凑的布局

theory\_y=other\_point\_prediction(D\_X,D\_Y,0.1\*k)

plt.scatter(D\_X,D\_Y,c='k',marker='.')

plt.scatter(D\_X,theory\_y,c='r',marker='\*')

a=0.1\*k

ax.set\_xlabel(a)

#循环作图

DY\_pre=other\_point\_prediction(D\_X,D\_Y,k\*0.2)

DY\_pre\_pd=pd.Series(DY\_pre)

DY\_pre\_pd[DY\_pre\_pd.isnull()]=0

DY\_pre\_pd

DY\_Pre\_ar=DY\_pre\_pd.values

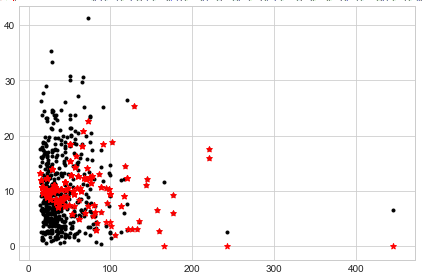
#分别作图

ax.scatter(D\_X,D\_Y,c='green',marker='\*')

Pre\_image=ax.plot(D\_X,DY\_Pre\_ar)

直观观察图形发现k=0.4，拟合较好。

下面是k=0.43时预测点与训练点分布图：



用均分根误差和平均绝对百分误差训练出比较精确的k值：

list\_2=[]

list\_3=[]

list\_4=[]

for k in range(2,100,1):

#tita\_list=[]

DY\_pre=other\_point\_prediction(D\_X,D\_Y,0.1\*k)

DY\_pre\_pd=pd.Series(DY\_pre)

DY\_pre\_pd[DY\_pre\_pd.isnull()]=0

# print(DY\_pre\_pd)

DY\_Pre\_ar\_1=DY\_pre\_pd.values

#print(DY\_pre\_ar\_1)

#(D\_Y-DY\_pre\_ar\_1)

stand\_diff=np.sqrt(np.sum((DY\_Pre\_ar\_1-D\_Y)\*\*2)/D\_Y.shape[0])

Mape=np.sum(np.abs((DY\_Pre\_ar\_1-D\_Y)/D\_Y))/D\_Y.shape[0]

#print(stand\_diff)

list\_2.append(k\*0.1)

list\_3.append(stand\_diff)

list\_4.append(Mape)

Rmse\_model=(np.array(list\_2),np.array(list\_3));Mape\_model=(np.array(list\_2),np.array(list\_4))

#寻找最小均方根误差和其对应的K值，这个值就是最佳可能K（=0.4）值。

Minium\_Rmse=np.min(Rmse\_model[1])

(Rmse\_model[0])[(np.where(Rmse\_model[1]==Minium\_Rmse)[0])[0]]

Out[250]: 0.40000000000000002

Minium\_Rmse

Out[260]: 6.1468185193105551（小于10可接受）

#同样的方法找到最小平均绝对百分误差和其对应的k值；这个值也是最佳可能K（=0.4）值

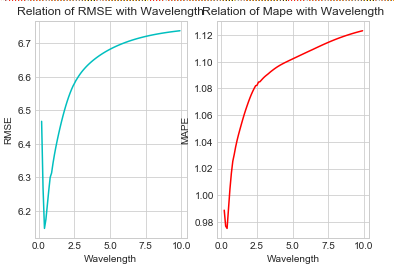
Minium\_Mape=np.min(Mape\_model[1])

(Mape\_model[0])[(np.where(Mape\_model[1]==Minium\_Mape)[0])[0]]

Out[259]: 0.40000000000000002

Minium\_Mape

0.97515871480952243（小于1已经十分理想）



最终我们的训练值确定为k=0.4

**测试阶段：**

D\_XT=Data2[501:601,4]

D\_YT=Data2[501:601,5]

#tita\_list=[]

DY\_pre=other\_point\_prediction(D\_XT,D\_YT,0.4)

DY\_pre\_pd=pd.Series(DY\_pre)

DY\_pre\_pd[DY\_pre\_pd.isnull()]=0

DY\_pre\_pd

DY\_Pre\_ar=DY\_pre\_pd.values

Square\_diff=np.sqrt(np.sum((DY\_Pre\_ar-D\_YT)\*\*2)/D\_YT.shape[0])

Square\_diff

Out[307]: 6.62889589872667#**非常理想的值，测试通过**

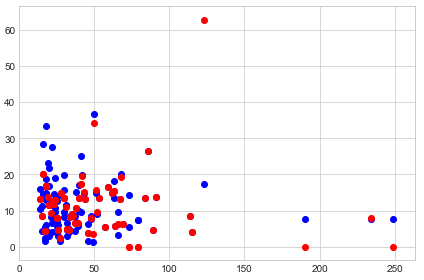
figt, axt= plt.subplots()

figt.set\_tight\_layout(True)

axt.scatter(D\_XT,D\_YT,c='b')

axt.scatter(D\_XT,DY\_Pre\_ar,c='r')

plt.show()



#仅从图片上看，测试效果极好。证明我们的模型建的还算成功。

还可以用RESE或者MAPE理论进行检验。

**预测阶段**

由于数据过于离散，所以我们不能用最近回归系数，而是用整个回归参数的变化趋势

#tita\_list=[]

DY\_pre=other\_point\_prediction(Data2[0:601,4],Data2[0:601,5],0.4)

DY\_pre\_pd=pd.Series(DY\_pre)

DY\_pre\_pd[DY\_pre\_pd.isnull()]=0

DY\_pre\_pd

DY\_Pre\_ar=DY\_pre\_pd.values

fig3, ax3 = plt.subplots()

fig3.set\_tight\_layout(True)

blist=[];klist=[]

for j in range(len(tita\_list)):

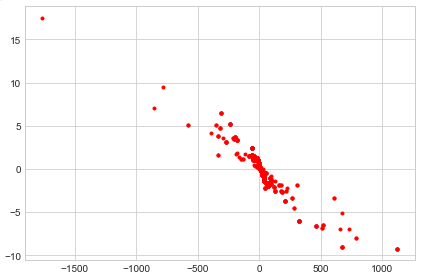
blist.append((tita\_list[j])[0])

klist.append((tita\_list[j])[1])

b\_array=np.array(blist);k\_array=np.array(klist)

ax3.scatter(b\_array,k\_array,c='r',marker='.')

**#KB图**



def ols\_func(array\_x,array\_y):

X= np.column\_stack((np.ones(array\_x.shape[0]),array\_x))

alpha=np.dot(linalg.inv(np.dot(X.T,X)),np.dot(X.T,array\_y))

return alpha

**#一般线性回归**

ols\_func(b\_array,k\_array)

Out[331]: array([ 0.09331623, -0.01177014])

fig4, ax4 = plt.subplots()

fig4.set\_tight\_layout(True)

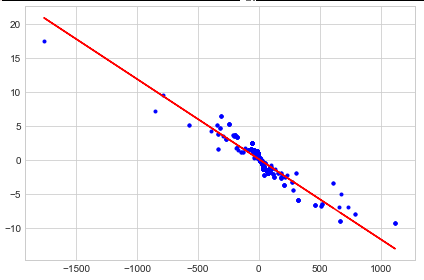
KB\_Vector=ols\_func(b\_array,k\_array)

ax4.plot(b\_array,KB\_Vector[1]\*b\_array+KB\_Vector[0],'r-')

ax4.scatter(b\_array,k\_array,c='b',marker='.')

plt.show()

**#KB的回归直线**



mintest\_1=1000

trager=0

for j in range(b\_array.shape[0]):

min\_test=abs(KB\_Vector[1]\*b\_array[j]+KB\_Vector[0]-k\_array[j])

if mintest\_1>min\_test:

mintest\_1=min\_test

trager=j

mintest\_1,trager

Out[37]: (5.2966018529998493e-06, 151)

通过b\_array[151]和k\_array[151]我们找到我们所需的K，B值

直到现在我们才找到真正的三个参数

K=--0.10904557466830056，B=19.107027979865393

， k=0.4

下面我们进行预测：

预测最多能预测一步或者几步。我们这里只预测一步：

X=83事先给定，预测结果为： 10.056245282396446,查询该企业后续资料14比较，相对误差在39%左右

之所以结果误差如此之大，主要是因为源数据过于分散，没有太明显的规律的缘故。

下面我们用另一种方法进行在计算一次：

就是给定要预测点X=83利用函数singel\_predition进行预测:

Data1=pd.read\_csv(r'Desktop\bankloan.csv',delimiter=',')

del Data1['Unnamed: 9']

Data2=Data1.values

D\_X=Data2[:601,4];

D\_Y=Data2[:601,5]

结果预测值为：

4.5794274182908339（B=-291.22916582，K=3.56395895）相对误差62%.

因此我们采取第一种方法。

接下来我们

下面我们看一个数据，这个数据相对来说比较规整，规律性强些。这些数据都是随机产生的，是比较好用的训练数据。下面我们用我们的模型去拟合它。

arr1=np.arange(100)

arr2=-arr1\*np.random.randn(100)

DY\_pre=other\_point\_prediction(arr1,arr2,0.4)

fig\_3, ax\_3 = plt.subplots()

fig\_3.set\_tight\_layout(True)

ax\_3.scatter(arr1,arr2,c='b')

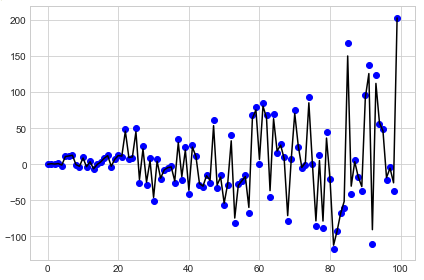
ax\_3.plot(arr1,DY\_pre,('k-'))

Out[397]: [<matplotlib.lines.Line2D at 0x756ce22e80>]

plt.show()

C:\Users\dongfeng\Anaconda3\lib\site-packages\matplotlib\figure.py:1999: UserWarning: This figure includes Axes that are not compatible with tight\_layout, so results might be incorrect.

warnings.warn("This figure includes Axes that are not compatible "



**几乎是完美拟合。**

1. **线性回归模型的正则化**
2. 岭回归

回归算法的本质是解决一个线性方程，



标准估计方法是普通的最小二线性回归。然而，如果没有x满足方程或超过一个x，即解决方案不是唯一的，（常见行对于列）那么就说这个问题是病态的。在这种情况下（病态A），普通最小二乘估计会导致超定（过拟合），或更常见的欠定（下拟合）方程组（A.TA\*α=A.T\*b）。

此外，岭回归还可以处理矩阵列的多重共线性问题，即相关性比较大的时候，会使得标准最小二乘求解不稳定, 因为此时矩阵A.T\*A行列式会近似为0。

通常最小二乘寻求最小化平方残差的总和，其可以紧凑地写成 ：



为了优先选择具有理想特性的特定解决方案，可以在这种最小化中加入正则化项：



其中Г是Tikhonov matrix矩阵，在许多情况下，这个矩阵被选为单位矩阵的倍数，Г=αI，此外，优先考虑规范较小的解决方案; 这就是所谓的L2正规化。

这种正则化改善了问题的条件，从而使问题得到了直接的数值解决。一个岭回归系数（即参数的确定）能如下计算：





**Python 的实现(非人工智能模拟)**

import numpy as np

import matplotlib.pyplot as plt

from scipy import linalg

def ridge\_regression(X\_array,Y\_array,lambda\_scalar):

X= np.column\_stack((np.ones( X\_array.shape[0]),X\_array))

alpha\_vector=(np.dot(X.T,X)+lambda\_scalar\*np.identity(X.shape[1]))

if linalg.det(alpha\_vector)==0.0:

print('singular Matrix')

return

alpha=np.dot(linalg.inv(alpha\_vector),np.dot(X.T,Y\_array))

return alpha

ridge\_data=ridge\_regression(arr\_1,arr\_2,0.1)

test\_data=ridge\_data[0]+ridge\_data[1]\*arr\_1

Arr\_1=np.array([ 4.5, 1.5, 2. , 2. , 1. , 2. , 2. , 3.5, 2.5, 2. , 4. ,

4.5, 4.5, 1.5, 4.5, 3.5, 2. , 2. , 3.5, 3.5, 1.5, 2.5,

3.5, 4. , 3.5, 4.5, 4.5, 1.5, 2.5, 3.5, 2.5, 2.5, 3.5,

3.5, 3. , 1. , 3. , 4. , 4. , 1.5, 4.5, 4.5, 2. , 2. ,

2. , 2. , 3. , 2. , 2.5, 1. , 4.5, 2.5, 3.5, 4.5, 4.5,

4.5, 3.5, 1. , 3. , 2. , 1.5, 4.5, 4.5, 4. , 3. , 3. ,

4. , 3. , 3.5, 3. , 4. , 1. , 4.5, 4. , 1.5, 3.5, 2. ,

4.5, 1.5, 1.5, 4. , 1. , 1. , 3.5, 1.5, 2.5, 2.5, 4. ,

3.5, 1.5, 2. , 1.5, 4.5, 4. , 4.5, 2. , 2. , 1.5, 2. ,

4. ])

Arr\_2=np.array([-0.07384915, -2.52776066, -1.0510301 , -1.01431582, 0.80815987,

0.93939175, 0.84833624, 0.16433826, -0.57019626, 0.20809428,

-0.36047602, -1.24902177, 0.29816839, 0.09042217, 1.47995212,

0.61001338, 1.37561497, -0.23334861, -0.81698986, -1.95380674,

0.64738488, -1.1436313 , -0.77313257, -0.13152878, 0.54353456,

0.1159813 , -0.53186671, 1.15907011, 2.12879954, -0.04383236,

0.52294959, -1.46061376, -0.43381713, 0.51359618, 1.45826544,

0.14253001, -0.79502784, -0.93006914, -2.07191989, -1.19867084,

0.84082914, -0.31282836, -1.29572674, 1.58181603, -0.10395556,

-0.93582985, 0.67583541, 0.01458932, -2.38878361, 0.60461051,

0.14677116, 0.08754279, 0.46965903, 2.51927566, 0.75624953,

0.81733071, 1.45611223, -0.10192756, 0.36294282, -1.54922232,

-0.3811177 , -0.83288047, -0.14161153, -0.87859021, -0.75357088,

-1.19036074, 0.11330122, 0.81394884, -0.3311788 , 1.000382 ,

0.72591856, -0.84329686, 0.32361859, -1.23861977, 1.70029021,

1.1717579 , -0.86601853, 0.48123254, -0.59832553, -1.12667686,

-0.53272368, 0.19337457, 1.0944413 , -0.19531637, 0.12624554,

1.01825491, 2.32429972, 0.28445279, 0.6138655 , -0.63299542,

1.40344257, -0.35323474, -0.15341559, 0.65588421, -0.28390616,

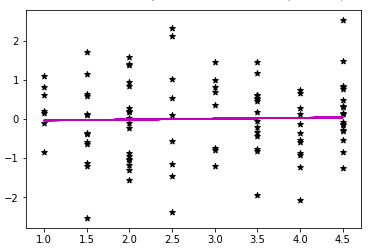
0.17266444, 0.26684605, 0.59602339, -1.16780386, -0.60059027])

fig,acs=plt.subplots()

acs.plot(arr\_1,test\_data,'m--')

acs.scatter(arr\_1,arr\_2,c='k',marker='\*')

plt.show()



回归效果极差，下面我们使用多项式组合进行岭回归。

1.创建多项式数据矩阵

def data\_matrix\_generated(arr\_seed\_1dim,polynom\_grad):

matrix=arr\_seed\_1dim

for i in range(2,polynom\_grad+1):

matrix=np.column\_stack((matrix,np.power(arr\_seed\_1dim,i)))

return matrix

1. 建立岭回归多项式混合模型

X\_as=data\_matrix\_generated(Arr\_2,10)

ridge\_regression(X\_as,Arr\_2,0.1)

Out[36]:

array([ 4.40092880e-04, 9.74537956e-01, -8.72719855e-04,

4.84562441e-02, 3.22213222e-03, -2.70997852e-02,

-2.32747107e-03, 5.65412766e-03, 5.34445807e-04,

-3.92378517e-04, -3.85499364e-05])

a\_vector=ridge\_regression(X\_as,Arr\_2,0.1)

arr=np.linspace(0,4.5,1050)

m\_12=data\_matrix\_generated(arr,10)

X= np.column\_stack((np.ones( m\_12.shape[0]),m\_12))

Y\_point=np.dot(X,a\_vector)

fig\_2,asx\_2=plt.subplots(dpi=140)

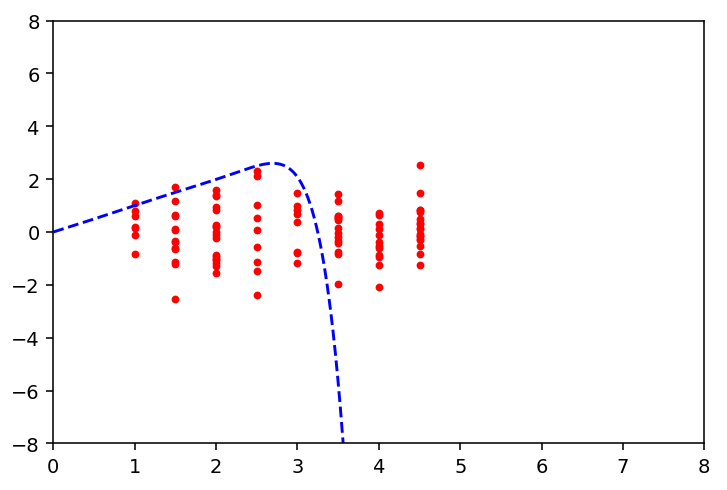
asx\_2.plot(arr,Y\_point,'b--')

asx\_2.scatter(Arr\_1,Arr\_2,c='r',marker='.')

plt.ylim(-8,8)

plt.xlim(0,8)

plt.show()



仅仅是稍有改善，与我们的要求差的还远。调节多项式度数和修正lambda值，我们可以找到一个158次方多项式非常拟合我们的图像。

for i in range(1,200,10):

X\_as=data\_matrix\_generated(Arr\_1,i)

a\_vector=ridge\_regression(X\_as,Arr\_2,0.05)

arr=np.linspace(0,4.5,1050)

m\_12=data\_matrix\_generated(arr,i)

X= np.column\_stack((np.ones( m\_12.shape[0]),m\_12))

Y\_point=np.dot(X,a\_vector)

fig\_2,asx\_2=plt.subplots(dpi=140)

asx\_2.plot(arr,Y\_point,'b--')

asx\_2.scatter(Arr\_1,Arr\_2,c='r',marker='.')

plt.xlabel(i)

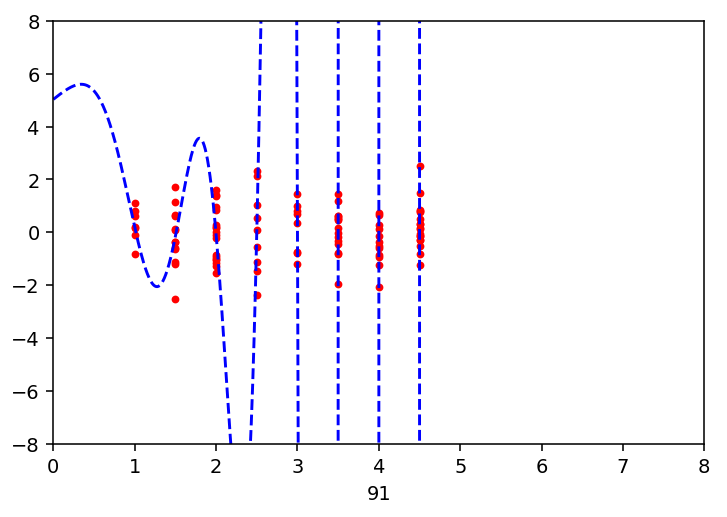
print(i,a\_vector,len(a\_vector))

plt.ylim(-8,8)

plt.xlim(0,8)

plt.show()

循环画图，找到多项式的最佳幂数和Lambda值。运行结果是幂数:91，lambda的最佳值是0.05



回归系数：

5.02863431e+00 2.43584609e+00 -4.49461214e-01 -3.16668534e+00

-4.85987520e+00 -4.35359979e+00 -3.17324057e-01 4.58616680e+00

8.32923865e+00 -8.96635687e+00 1.53361605e+00 4.72474501e-01

-1.05066870e-01 1.95053886e-02 -5.73166535e-03 -2.30857852e-03

1.60303716e-04 1.04268493e-04 1.23238941e-05 4.32136758e-07

1.26179879e-06 -2.73253722e-07 -4.77777409e-08 -7.17587134e-09

-6.47541015e-09 5.39911095e-10 1.86026200e-10 1.95448805e-11

2.30773770e-11 -1.65299839e-12 4.00307445e-14 -1.77026398e-13

1.60236920e-14 -2.50498014e-15 3.87032432e-16 -2.06404836e-16

-2.18742735e-17 1.81485135e-17 1.10921706e-18 1.18601011e-19

-7.64499604e-20 1.67189432e-20 -1.13932625e-20 -3.08376617e-22

9.20703376e-25 -1.17100627e-22 8.91024245e-24 6.02462626e-25

8.57953978e-25 9.04776204e-26 4.28753741e-27 -5.40558509e-27

-2.49400412e-27 -7.12017579e-29 -1.37313932e-28 -1.12509414e-30

-1.86919586e-30 -2.62999076e-30 1.05491295e-31 4.35669743e-33

3.07850677e-33 -1.50116612e-33 2.71335426e-34 5.26748640e-36

-8.78290659e-36 2.92787176e-36 -3.35925170e-37 7.07841178e-37

6.47069018e-39 5.61841660e-39 -1.50703154e-39 5.49178874e-40

1.02147876e-40 -3.72526161e-42 6.86869133e-42 -1.87826851e-42

1.73833157e-43 8.61670370e-44 -3.46381145e-46 5.23157251e-45

2.81804829e-46 3.17648722e-49 4.48044227e-47 -1.23281259e-47

-1.92264854e-48 -1.10516897e-49 1.70532154e-49 -3.05862403e-50

6.69990963e-52 8.20521003e-52 -1.22409404e-52 2.25992518e-54

X\_as=data\_matrix\_generated(Arr\_1,91)

a\_vector=ridge\_regression(X\_as,Arr\_2,0.05)

arr=np.linspace(0,4.5,1050)

m\_12=data\_matrix\_generated(arr,91)

X= np.column\_stack((np.ones( m\_12.shape[0]),m\_12))

Y\_point=np.dot(X,a\_vector)

#Y\_point=np.dot(X,a\_vector)

fig\_2,asx\_2=plt.subplots(dpi=140)

asx\_2.plot(arr,Y\_point,'b--')

asx\_2.scatter(Arr\_1,Arr\_2,c='r',marker='.')

plt.xlabel(91)

#print(i,a\_vector,len(a\_vector))

plt.ylim(-8,8)

plt.xlim(0,8)

plt.show()

X\_as=data\_matrix\_generated(Arr\_1,91)

X= np.column\_stack((np.ones( X\_as.shape[0]),X\_as))

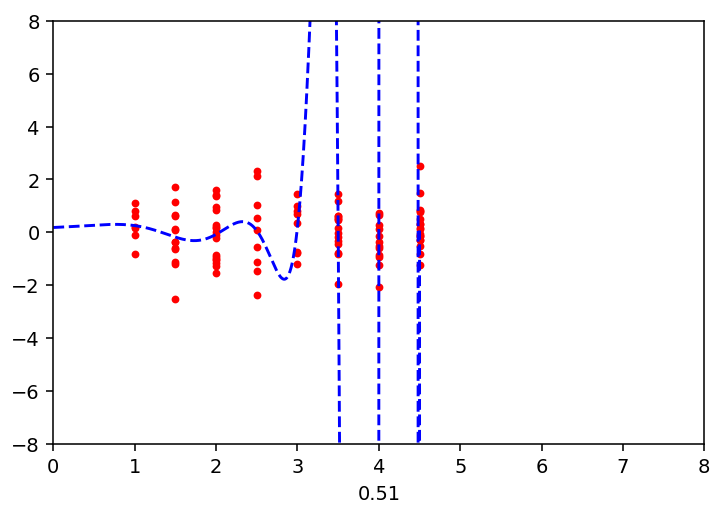
np.sum(np.abs((np.dot(X,a\_vector)-Arr\_2)/Arr\_2))/X.shape[0]

Out[55]: 76.353962017487703

Mape检验结果：76.353962017487703，值很差，说明我们的模型是十分不成功的。调节参数，重新找好的参数；

原因分析：尽管表面上线拟合的很好，实际上并不一定反映全部事实。例如，我们从上面的图像中会发现，拟合函数最右边的几纵行非常倾斜，自变量的稍微变化，就会导致预测值的巨大变化，最终导致预测巨大的误差。

经过继续的参数调节，我们找到 grad=158..



def test\_lambda\_value(Arr\_1,Arr\_2):

trager=1000

for lambda\_1 in range(2,200,10):

X\_as=data\_matrix\_generated(Arr\_1,158)

a\_vector=ridge\_regression(X\_as,Arr\_2,lambda\_1\*0.01)

#arr=np.linspace(0,4.5,1050)

#m\_12=data\_matrix\_generated(arr,150+i)

X=np.column\_stack((np.ones( X\_as.shape[0]),X\_as))

Y\_prodition= np.dot(X,a\_vector)

Mape=np.sum(np.abs((Arr\_2-Y\_prodition)/Arr\_2))/Arr\_1.shape[0]

#print(Mape)

if trager>Mape:

trager=Mape

zusatz=lambda\_1

#print(trager)

return trager,zusatz

test\_lambda\_value(Arr\_1,Arr\_2)

Out[48]: (1.2911408567946514, 62)

在lambda值在0.02到2之间取值时，它的Mape大致在1.1上下，这样结果已经十分好了。证明我们的模型是十分理想的。

下面通过MAPE来寻找模型：

import numpy as np

import matplotlib.pyplot as plt

from scipy import linalg

def ridge\_regression(X\_array,Y\_array,lambda\_scalar):

X= np.column\_stack((np.ones( X\_array.shape[0]),X\_array))

alpha\_vector=(np.dot(X.T,X)+lambda\_scalar\*np.identity(X.shape[1]))

if linalg.det(alpha\_vector)==0.0:

print('singular Matrix')

return

alpha=np.dot(linalg.inv(alpha\_vector),np.dot(X.T,Y\_array))

return alpha

def data\_matrix\_generated(arr\_seed\_1dim,polynom\_grad):

matrix=arr\_seed\_1dim

for i in range(2,polynom\_grad+1):

matrix=np.column\_stack((matrix,np.power(arr\_seed\_1dim,i)))

return matrix

def Mape(Arr\_1,Arr\_2):

trager=100000

for lambda\_1 in range(1,10):

for grad\_1 in range(2,200):

X\_as=data\_matrix\_generated(Arr\_1,grad\_1)

a\_vector=ridge\_regression(X\_as,Arr\_2,lambda\_1\*0.01)

#arr=np.linspace(0,4.5,1050)

#m\_12=data\_matrix\_generated(arr,150+i)

X=np.column\_stack((np.ones( X\_as.shape[0]),X\_as))

Y\_prodition = np.dot(X,a\_vector)

Mape=np.sum(np.abs((Arr\_2-Y\_prodition)/Arr\_2))/Arr\_1.shape[0]

#print(Mape)

if trager>Mape:

trager=Mape

zusatz=lambda\_1

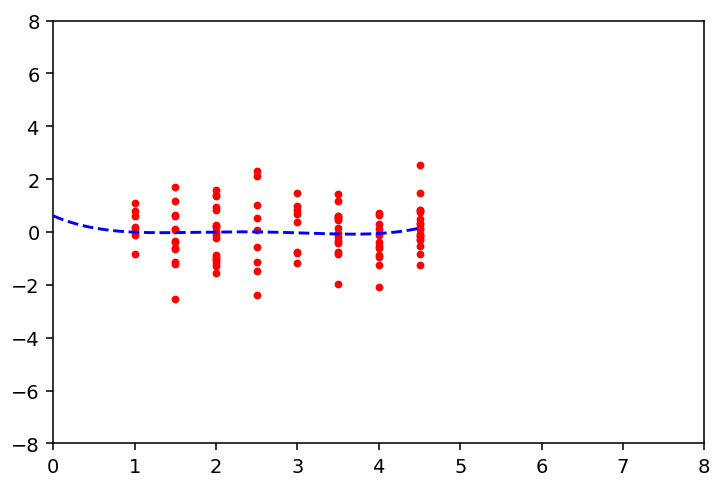
zusatz\_2=grad\_1

#print(trager)

return trager,zusatz,zusatz

**(1.0205712852072406, 9, 4)是其返回结果**

说明通过这种方法我们可以优化出一条最高幂为4的多项式。对其画图得：



一眼可以看出，通过这种方法找到拟合曲线根本无法完美拟合。因此我这里强调，不要用检验模型的方法去寻找优化曲线。正常的顺序是

1. 找到能尽量拟合已知点的曲线。
2. 然后用这种方法去检验是否合理。

岭回归寻找最佳lambda值：

绘制岭轨迹图（横轴是log（λ），纵坐标为回归系数，横坐标为log(lambda).

查看数据的多重共线性：

首先，我们需测定log（lambda）与回归系数之间的关系：

def test\_lambda\_value(Arr\_1,Arr\_2):

list\_a\_vector=[]

#trager=1000

for lambda\_1 in range(0,200,10):

X\_as=data\_matrix\_generated(Arr\_1,158)

a\_vector=ridge\_regression(X\_as,Arr\_2,lambda\_1\*0.01)

#arr=np.linspace(0,4.5,1050)

#m\_12=data\_matrix\_generated(arr,150+i)

X=np.column\_stack((np.ones( X\_as.shape[0]),X\_as))

Y\_prodition= np.dot(X,a\_vector)

#Mape=np.sum(np.abs((Arr\_2-Y\_prodition)/Arr\_2))/Arr\_1.shape[0]

#print(Mape)

#if trager>Mape:

#trager=Mape

#zusatz=lambda\_1

list\_a\_vector.append((a\_vector,lambda\_1\*0.01))

return list\_a\_vector

test\_lambda\_value(Arr\_1,Arr\_2)

data\_1=test\_lambda\_value(Arr\_1,Arr\_2)

Data\_list=[]

Datalist\_ind=[]

for i in range(len(data\_1)):

Datalist\_ind.append(data\_1[i][1])

Data\_list.append(Datalist\_ind)

#X矩阵的形成

Datalist\_ind=[]

for j in range(data\_1[1][0].shape[0]):

for i in range(len(data\_1)):

Datalist\_ind.append(data\_1[i][0][j])

Data\_list.append(Datalist\_ind)

Datalist\_ind=[]

Data\_list\_ar=np.array(Data\_list)

Y矩阵的形成

#横轴是以10为底的log（lambda）

pfi\_1=plt.figure(figsize=(4,3),dpi=220)

acs\_1=pfi\_1.add\_subplot(1,1,1)

#plt.xlim(0,1)

#plt.ylim(-15,15)

acs\_1.plot(np.log(Data\_list\_ar[0][10:]).T,(Data\_list\_ar[1:,10:]).T)

plt.show()

#普通岭轨迹图

pfi\_3=plt.figure(figsize=(4,3),dpi=220)

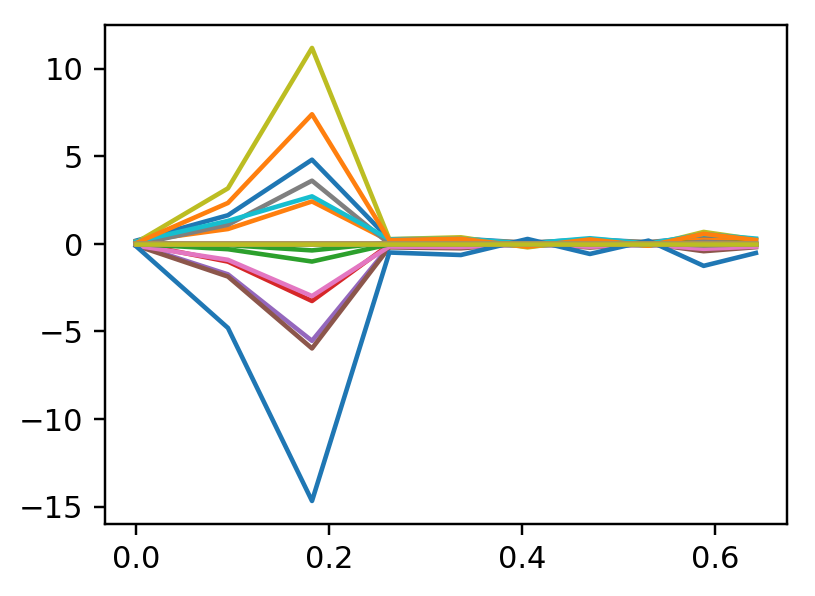
acs\_3=pfi\_3.add\_subplot(1,1,1)

#plt.xlim(0,1)

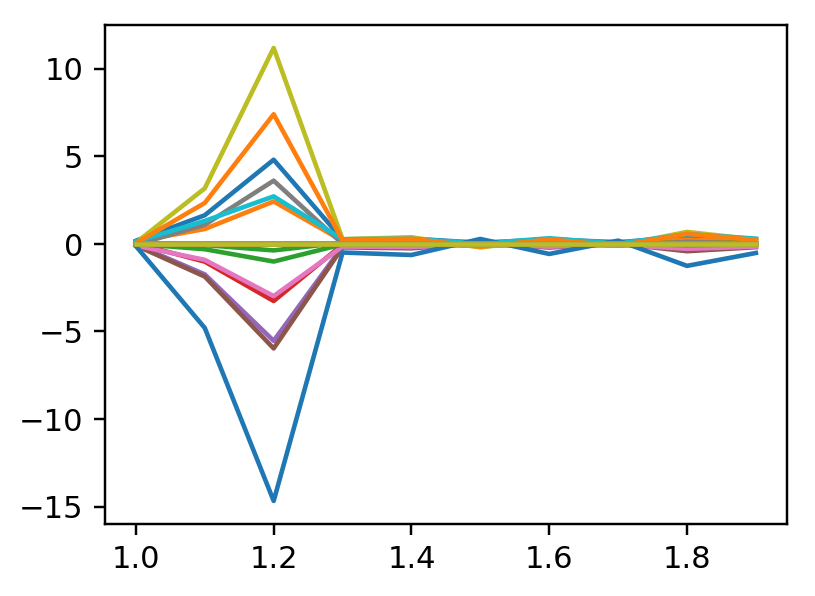
#plt.ylim(-15,15)

acs\_3.plot(Data\_list\_ar[0][10:].T,(Data\_list\_ar[1:,10:]).T)

plt.show()



**对数岭轨迹图**



**普通岭轨迹图**

由于原始数据病态（无解，多解，共线性等）性质，（它是有众多细微差别的数据组成的矩阵（导致弱相关行列式，也即：多重共线性））所以岭轨迹波动大，并且与普通数据矩阵的岭回归相比，外形会稍有所不同。但不管多波动，最终还是基本趋于稳定，也即回归系数不再随**λ**的变化而变化。这个特点正反映了岭回归的优点；当然，如果轨迹线无法趋于稳定，那我们就要考虑是否用线性模型来处理这个问题了。

在病态矩阵里每一个目标变量微小的变动都会产生巨大的方差（回归系数产生方差）。对于这种情况就需要设置一个比较合适的正则化参数来减少离差（噪声）。这正是我引入岭回归的重要原因之一。岭回归能够使产生噪音（振荡）的回归系数趋于稳定（看图）。

什么是岭轨迹图，他有什么作用呢？

岭轨迹图是Lambda与回归系数的关系图，在最左边，lambda=0， 回归系数与线性回归一致，最右边系数全部缩减为0，期间.尽管还稍许波动，但最终还是归结于0。

前面我们已经看到，我们无法准确地确定λ，甚至连大致的范围也无法确定。它的确定可以通过岭轨迹进行近似确定。也可以通过lamnda损失函数法来近似确定lambda值。也即找到一个lambda值，使得残差平方和趋于稳定（即增加幅度细微），那么这个lambda值就是我们需要的最优lambda值。具体实现方法可以通过做lambda与损失函数关系图来实现。这里我们选择用上面岭轨迹图来选择合适的lambda值。这是岭轨迹的第一个应用。

λ一般选择原则：

一，观察法

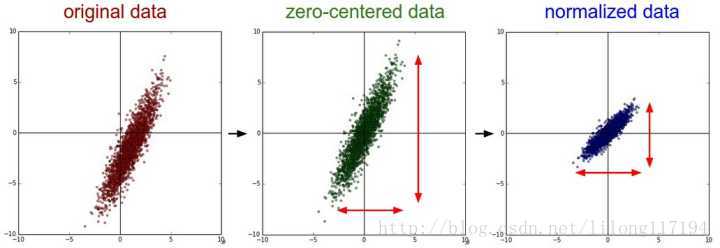
1. 各回归系数的岭估计基本稳定；（喇叭口附近均可）比如我们的例子，我们可以选择λ=1.3
2. 用最小二乘估计时符号不合理的回归系数，其岭估计的符号变得合理； （可操作性差）
3. 回归系数没有不合乎实际意义的值；
4. 理论计算（方差扩大因子法）

假设数据矩阵已经去中心化且标准化，

数据z-score标准化和去中心化：

数据标准化：是指数值减去均值，再除以标准差； 数据中心化：是指变量减去它的均值。

目的：通过中心化和标准化处理，得到均值为0，标准差为1的服从标准正态分布的数据。（前提是原始数据是服从正态或近似正态分布数据）计算过程由下式表示：



原因：在一些实际问题中，我们得到的样本数据都是多个维度的，即一个样本是用多个特征来表征的。很显然，这些特征的量纲和数值得量级都是不一样的，而通过标准化处理，可以使得不同的特征具有相同的尺度（Scale，可给学生举例去量纲，1000公斤/200公斤=5，5个scale）。这样，在学习参数的时候，不同特征对参数的影响程度就一样了。简言之，当原始数据不同维度上的特征的尺度（单位）不一致时，需要标准化步骤对数据进行预处理。所以，对于多特征数据，标准化处理十分必要的。

适用范围：

z-score标准化方法适用于属性A的最大值和最小值未知的情况，或有超出取值范围的离群数据的情况。该种归一化方式要求原始数据的分布可以近似为高斯分布，否则归一化的效果会变得很糟糕。

主要应用领域：

在分类、聚类算法中，需要使用距离来度量相似性的时候、或者使用PCA技术进行降维的时候，Z-score 标准化被常用。

此时可以回忆我们以前讲的归一化，让学生自己作比较：

归一化不易数据扩充，因为归一化依靠数据额最大值与最小值。

归一化后的数据不一定复合标正态分布。

归一化应用领域：

在不涉及距离度量、协方差计算、数据不符合正太分布的时候，可以使用。

不需要归一化、标准化的模型：

ICA不需要归一化（问学生因为独立成分如果归一化了就不独立了？）。

基于平方损失的最小二乘法OLS不需要归一化。

小项目：数据处理，去中心化和标准化：

Dat\_quantity=1000

mu = np.array([[1, 3]])

arr\_pos = np.array([[1, 0.5], [1.5, 3]]) #创建正定矩阵

R = cholesky(arr\_pos) #用cholesky进行分解同时判断是否为正定，如果他是正定，必定可以被分解

s = np.dot(np.random.normal(size=(Dat\_quantity,2)), R) + mu

ser\_1,tyt\_1=plt.subplots()

tyt\_1.scatter(s[:,0],s[:,1])

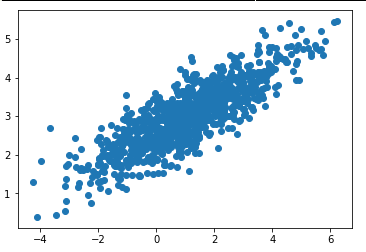
Tr=(s-s.mean(0))/(s.std(0))

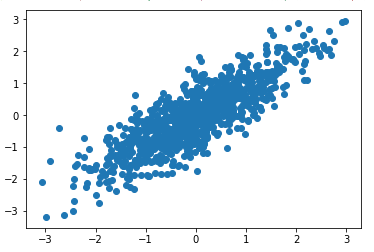
ser\_2,tyt\_2=plt.subplots()

tyt\_2.scatter(Tr[:,0],Tr[:,1])

plt.show()

注意：数据的标准化处理都是对单一属性数据处理的，即，按列处理。





OLS（ordinary Least Square）的回归系数方差为σ2，那么岭回归的回归系数的方差为：



通过岭回归与OLS回归系数的方差的比值我们得到方差扩大因子：



k增大，扩大因子减小，所以我们逐步增大k值，直到所有回归系数都小于或等于10.

这个方法的缺点：

1. 对于多回归系数问题，计算量太大，太过于麻烦。因为

代表的是复相关系数，确切的说是i列矢量对于所有其他列矢量的复相关系数，一个复相关系数的确定就需要一个线性回归方程。就像我们的例子，需要159个线性回归才能解决。这将是一个庞大计算量。

B,很多时候并带不来更好的拟合效果。

复相关系数：

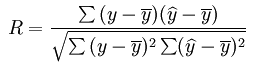
复相关系数是测量一个变量与其他多个变量之间线性相关程度的指标。它不能直接测算，只能采取一定的方法进行间接测算。

为了测定一个变量y与其他多个变量X1,X2,...,Xk之间的相关系数，可以考虑构造一个关于X1,X2,...,Xk的线性组合，通过计算该线性组合与y之间的简单相关系数作为变量y与X1,X2,...,Xk之间的复相关系数。具体计算过程如下：

第一步，用y对X1,X2,...,Xk作回归，得：



第二步，计算y和 的简单相关系数，此简单相关系数即为y与X1,X2,...,Xk之间的复相关系数。复相关系数的计算公式为：



复相关系数与简单相关系数的区别是简单相关系数的取值范围是[-1,1]，而复相关系数的取值范围是[0,1]。这是因为，在两个变量的情况下，回归系数有正负之分，所以在研究相关时，也有正相关和负相关之分；但在多个变量时，回归系数有两个或两个以上，其符号有正有负，不能按正负来区别，所以复相关系数也就只取正值。



回归系数选择是岭轨迹的第二个应用：（经验）

1.开始（lambda=0）十分大，但是随着lambda的增加，迅速向0靠近的回归系数不重要。可以考虑剔除。

2.开始（lambda=0）十分小，但是随着lambda的增加，回归系数绝对值迅速增速很快，说明该系数重要。

3.如果两个系数已知不稳定，但从形状上看，总体上来说和是稳定的，这两个系数可以合并成一个新的系数。也可以选择不合并，不影响预测效果。

4. 在岭回归中设计矩阵X已经中心化和标准化了，这样可以直接比较标准化岭回归系数的大小。可以剔除掉标准化岭回归系数比较稳定且值很小的自变量。

5.随着λ的增加，回归系数不稳定，震动趋于零的自变量也可以剔除。

岭轨迹的第三个应用：多重共线性判断

岭的出现说明数据矩阵存在多重共线性！如果模型仅用于预测，则只要拟合程度好，可不处理多重共线性问题，存在多重共线性的模型用于预测时，往往不影响预测结果。

岭回归缺点：

取λ的方法比较多，但是结果差异较大。

岭轨迹图取lambda值，全靠分析师经验。

回归系数筛选，随意性强，主观性强。

岭回归可以判断多重共线性存在，但是很难确认那一列和那一列存在多重共线性。下面多重共线性的判断方法。

我们在在讲线性代数时已经线性不相关的概念和相信相关的概念，这里不再重复。线性相关还有一个名字，就叫做复共线性，也即：





上面的等式告诉我们这样的事实，如果你发现两列存在倍数关系或多列线性组合与一列或一列倍数存在相等关系，那么我们就说多重线性关系存在。

当然上面是比较粗糙的判断，简单有效的判断方式是VIF法（方差扩大因子法）

方差扩大因子(VIF )定义为：



其中 为 矩阵X的Xj 对其余p-1个自变量的复相关系数的平方。



通过VIF我们可以判断矩阵的共线性状况：

VIF in (0,5] 没有复共线性

VIF in (5,10] 有较弱复共线性

VIF in (10,100] 有中等或较强的复共线性

VIF >100, 有严重的复共线性。

另外还有特征值法：

X.TX的最小的特征值 ，无复共线性

，弱复共线性。

，中等或较强的复共线性

，严重复共线性

条件数法：

条件数定义：CN=最大特征值/最小特征值

CN in （0,30）没有复共线性

（30,100）弱复共线性

（100,1000）中等较强复共线性

1. ∞）强复共线性

复相关系数和VIF的python实现：

import pandas as pd

import numpy as np

from scipy import linalg

#复相关系数

def ols\_func(array\_x,array\_y):

X= np.column\_stack((np.ones(array\_x.shape[0]),array\_x))

alpha=np.dot(linalg.inv(np.dot(X.T,X)),np.dot(X.T,array\_y))

return alpha

def multi\_corr\_coeff(arry\_tst):

mul\_coff=[]

for j in range(arry\_tst.shape[1]):

arr\_del=np.delete(arry\_tst,j,1)

re\_coff=ols\_func(arr\_del,arry\_tst[:,j])

X=np.column\_stack((np.ones(arr\_del.shape[0]),arr\_del))

y\_th=np.dot(X,re\_coff)

mul\_coff\_single=(np.dot((arry\_tst[:,j]-arry\_tst[:,j].mean()),(y\_th-arry\_tst[:,j].mean())))/(np.linalg.norm(arry\_tst[:,j]-arry\_tst[:,j].mean(),2)\*np.linalg.norm(y\_th-arry\_tst[:,j].mean(),2))

mul\_coff.append(mul\_coff\_single)

return mul\_coff

#VIF计算

import pandas as pd

import numpy as np

from scipy import linalg

def ols\_func(array\_x,array\_y):

X= np.column\_stack((np.ones(array\_x.shape[0]),array\_x))

alpha=np.dot(linalg.inv(np.dot(X.T,X)),np.dot(X.T,array\_y))

return alpha

def multi\_corr\_coeff(arry\_tst):

mul\_coff=[]

for j in range(arry\_tst.shape[1]):

arr\_del=np.delete(arry\_tst,j,1)

re\_coff=ols\_func(arr\_del,arry\_tst[:,j])

X=np.column\_stack((np.ones(arr\_del.shape[0]),arr\_del))

y\_th=np.dot(X,re\_coff)

mul\_coff\_single=(np.dot((arry\_tst[:,j]-arry\_tst[:,j].mean()),(y\_th-arry\_tst[:,j].mean())))/(np.linalg.norm(arry\_tst[:,j]-arry\_tst[:,j].mean(),2)\*np.linalg.norm(y\_th-arry\_tst[:,j].mean(),2))

mul\_coff.append(mul\_coff\_single)

return 1/(1-np.power(mul\_coff,2))

R语言：

a <- c(1, 1, 2, 3, 4)

b <- c(2, 2, 3, 2, 1)

c <- c(4, 6, 7, 8, 9)

d <- c(4, 3, 4, 5, 4)

df <- data.frame(a, b, c, d)

vif\_df <- vif(df)

print(vif\_df)

Variables VIF

a 22.95

b 3.00

c 12.95

d 3.00

from pandas.core import datetools

import statsmodels.api as sm

import pandas as pd

import numpy as np

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

from statsmodels.tools.tools import add\_constant

df = pd.DataFrame(

{'a': [1, 1, 2, 3, 4],

'b': [2, 2, 3, 2, 1],

'c': [4, 6, 7, 8, 9],

'd': [4, 3, 4, 5, 4]}

)

df

Out[18]:

a b c d

0 1 2 4 4

1 1 2 6 3

2 2 3 7 4

3 3 2 8 5

4 4 1 9 4

pd.Series([variance\_inflation\_factor(df.values, i)for i in range(df.shape[1])],index=df.columns)

Out[12]:

a 47.136986

b 28.931507

c 80.315068

d 40.438356

dtype: float64

#这个答案并不正确，我们仍需加上1个‘1’列，使其成为回归数据矩阵才可以。

X\_1=add\_constant(df)

X\_1

Out[20]:

const a b c d

0 1.0 1 2 4 4

1 1.0 1 2 6 3

2 1.0 2 3 7 4

3 1.0 3 2 8 5

4 1.0 4 1 9 4

pd.Series([variance\_inflation\_factor(X\_1.values, i) for i in range(X\_1.shape[1])],index=X\_1.columns)

Out[22]:

const 136.875

a 22.950

b 3.000

c 12.950

d 3.000

#替换方式：

X = df.assign(const=1)

X

Out[26]:

a b c d const

0 1 2 4 4 1

1 1 2 6 3 1

2 2 3 7 4 1

3 3 2 8 5 1

4 4 1 9 4 1

pd.Series([variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])],index=X.columns)

Out[27]:

a 22.950

b 3.000

c 12.950

d 3.000

const 136.875

dtype: float64

鲍鱼年龄数据：

#岭回归

import numpy as np

import matplotlib.pyplot as plt

from scipy import linalg

def ridge\_regression(X\_array,Y\_array,lambda\_scalar):

X= np.column\_stack((np.ones( X\_array.shape[0]),X\_array))

alpha\_vector=(np.dot(X.T,X)+lambda\_scalar\*np.identity(X.shape[1]))

if linalg.det(alpha\_vector)==0.0:

print('singular Matrix')

return

alpha=np.dot(linalg.inv(alpha\_vector),np.dot(X.T,Y\_array))

return alpha

import pandas as pd

data\_fish=pd.read\_csv(r'Desktop\data\_new.csv',header=None,delimiter=',').values

from math import exp

coff\_vector=ridge\_regression(data\_fish[:200,:8],data\_fish[:200,-1],exp(20))

#岭轨迹线

def ridge\_trace(X, Y, lambda\_1=30):

list\_tuple=[]

for j in range(lambda\_1):

coff\_vector=ridge\_regression(X,Y,exp(j-5))

list\_tuple.append((coff\_vector,exp(j-5)))

return list\_tuple

ridge\_trace(data\_fish[:200,:8],data\_fish[:200,-1],lambda\_1=30)

#所需矩阵形成

data\_1=ridge\_trace(data\_fish[:200,:8],data\_fish[:200,-1],lambda\_1=30)

Data\_list=[]

Datalist\_ind=[]

for i in range(len(data\_1)):

Datalist\_ind.append(data\_1[i][1])

Data\_list.append(Datalist\_ind)

#X矩阵的形成

Datalist\_ind=[]

for j in range(data\_1[1][0].shape[0]):

for i in range(len(data\_1)):

Datalist\_ind.append(data\_1[i][0][j])

Data\_list.append(Datalist\_ind)

Datalist\_ind=[]

Data\_list\_ar=np.array(Data\_list)

#Y矩阵形成

pfi\_1=plt.figure(figsize=(4,3),dpi=220)

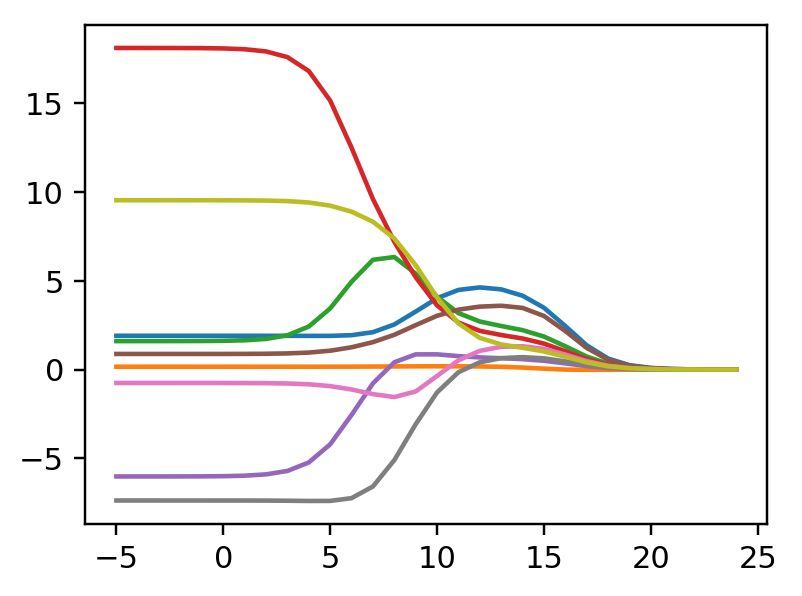
acs\_1=pfi\_1.add\_subplot(1,1,1)

#plt.xlim(0,1)

#plt.ylim(-15,15)

acs\_1.plot(np.log(Data\_list\_ar[0][:]).T,(Data\_list\_ar[1:,:]).T)

plt.show()



data\_fish\_1=np.column\_stack((np.ones(data\_fish[200:,:8].shape[0]),data\_fish[200:,:8]))

Es\_list=[]

coff\_train\_list=[]

for i in range(30):

lambda\_1=exp(i-5)

coff\_train=ridge\_regression(data\_fish[:200,:8],data\_fish[:200,-1],lambda\_1)

#训练数据

ES=np.std(np.dot(data\_fish\_1,coff\_train)-data\_fish[200:,-1])

#测试数据

#误差方差

coff\_train\_list.append(coff\_train)

Es\_list.append(ES)

np.min(Es\_list)

Out[40]: 2.5945450536803398

#找出最小值

#找出对应的回归系数

Es\_list\_ar=np.array(Es\_list)

np.where(Es\_list\_ar==2.5945450536803398)

Out[44]: (array([2], dtype=int64),)

index\_es=np.where(Es\_list\_ar==2.5945450536803398)

#查找误差最小值以及对应的回归系数

coff\_train\_list[index\_es[0][0]]

Out[51]:

array([ 2.10845579, 0.17206362, 6.18664744, 9.61021664, -0.77620417,

1.54861182, -1.38009651, -6.58327635, 8.33260787])

Lasson 回归：

Lasso算法从最先开始就是为最小二问题的解决而产生的，与岭回归不同；岭回归限定了所有回归系数的平方和不大于t,也就是把所有系数限制在一个多维球体内，包括球面，这样就避免了在使用普通最小二乘法回归的时候，当两个变量相关时，可能产生的两个正负大系数（喇叭口），随着正则项的lambda的增大，高幅振荡的系数迅速转向平稳（参看岭轨迹线）

Lasso算法采用了同样的思路，只不过是另一种限制方式，范数约束。下面我们先看看范数的定义：



当p=1时，上面的范数是一范数

当p=2时，上面的范数是二范数

当p=3时，上面的范数是三范数

以此类推！

在二维情况下这个约束区域是一个正方形围成的区域(让学生说出原因)，这一点与岭回归有很大不同。为了更好理解lasso回归，我们用几何分析法来阐述它的基本原理，我们先回到岭回归的几何含以上：

我们知道岭回归的损失函数达式为：



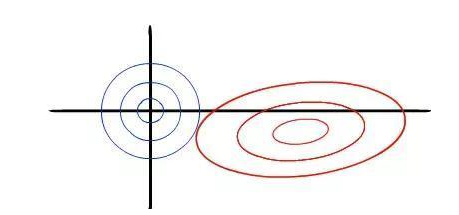


这是他的拉格朗日算子形式形式，在求损失函数最小值的时候，他其实等价于下列等式：

（#）

(参看数据分析二的朗格朗日乘子法)

它的几何意义是在一个圆域内寻找损失函数的最小值。如下图：

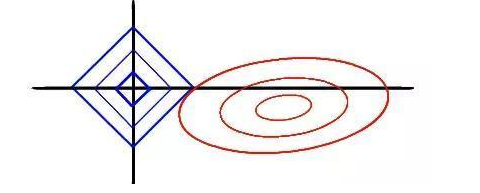


损失函数是一个标量函数，我们假设它是从二维到一维的一个曲面函数（α1,α2是两个自变量参数，S（α1,α2）是一关于这两个变量的标量函数，他们的坐标（α1,α2，S（α1,α2））在三维空间中是一个曲面，上图可看做它在二维空间的投影）红色区域可看做它在二维平面的投影，一个个相套的圆圈是等值线，更形象描述是等高线。

在圆域内寻找唯一最小值意味着等高线（不必要是最小值等高线，因为相切的话，即使红色区域中有最小值，由于不在圆域内，因此不作数。）与某一圆域相切。（相交的话会产生很多最小值）

当lambda的值增大，曲面所覆盖范围扩展（等高线要增大），投影也会扩展（只看等式‘#’，不改变函数，函数值增大，只有通过扩大定义域来实现），产生新的等高线，为了保证相切，t会相对缩小，回归系数会向0缩进。反之，lambda值减小等高线回缩，t增大，回归系数会全部逐步增大。

相对于岭回归，lasso回归的是在一个方形域内寻找唯一最小值，和岭回归原理一样，lasso回归也是lambda增加，t减小。只不过在减小的过程中，会有系数不断的减小为0，其原因是，方形的顶点非常容易与等值线相切，且相切时，部分系数归0，这一点是岭回归很难做到的。如下图所示，当顶点与等价线相切时，y方向回归系数会自动归零。



Lasso回归算法：

我们把lasso回归损失函数分为两部分，然后依次求导：

第一部分：



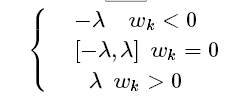
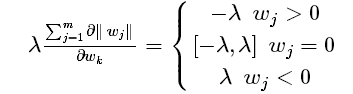




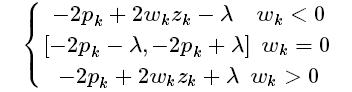


其中：

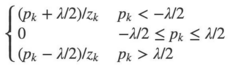
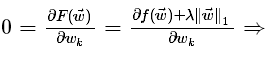
第二部分求导：



两部分合并得：



令导数等于0得：



注：Lagrange方程的求导等于0，等价于求出差方和在附加的指定区域内的极值。等式的右边(求导第二部分）是典型的坐标下降法。也就是通过坐标下降法（改变lambda来实现）来确定回归系数。

编程思想：

1. 通过上面的公式,每次选取一维进行优化，优化出一维，然后再循环到下一维，直到全部优化成功。
2. 然后优化出的回归系数矢量代入到损失函数计算，与初始损失函数值比较，如果差别没有达到阈值，继续循环计算。
3. 优化出新的回归系数，再次代入计算出损失函数值，与上一次的损失函数值作比较，如果绝对差别小于阈值，循环结束，返回矢量回归系数值。

Python实现：

import itertools

from math import exp

import numpy as np

import matplotlib.pyplot as plt

def lasso\_regression(X\_array, y, lambd, threshold=0.1):

#通过坐标下降(coordinate descent)法获取LASSO回归系数

# 计算残差平方和

X=np.column\_stack((np.ones( X\_array.shape[0]),X\_array))

square\_diff\_sum = lambda X, y, w: np.dot((np.dot(X,w)-y).T,(np.dot(X,w)-y))

#数据的初始化，参数是0数组，差方的初始化通过w的0数组来实现的

m, n = X.shape

w = np.zeros(n)#创造承载器

r = square\_diff\_sum(X, y, w)

# 迭代器

itertoll = itertools.count(1)

for it in itertoll:

for k in range(n):

#print(k)

z\_k = np.dot(X[:,k].T,X[:,k])

#print(z\_k)

p\_k = 0

for i in range(m):

p\_k += X[i,k]\*(y[i] - sum([X[i,j]\*w[j] for j in range(n) if j != k]))

#print(p\_k,lambd/2)

if p\_k < -lambd/2:

w\_k = (p\_k + lambd/2)/z\_k

elif p\_k > lambd/2:

w\_k = (p\_k - lambd/2)/z\_k

else:

w\_k = 0

w[k] = w\_k

square\_diff\_sum\_indirect = square\_diff\_sum(X, y, w)

error\_abs = abs(square\_diff\_sum\_indirect- r)#绝对差别

r = square\_diff\_sum\_indirect

if error\_abs< threshold:

break

return w

lasso\_regression(X\_data[:200,:8],X\_data[:200,-1],0.2,threshold=0.1)

Out[15]:

array([ 2.74821383, 0.15701497, 10.5184226 , 2.29322912,

-3.19005458, 1.61443273, -0.80406541, -6.53799699, 9.02441384])

def lasso\_trace(X, Y, lambda\_1=30):

list\_tuple=[]

for j in range(lambda\_1):

coff\_vector=lasso\_regression(X,Y,exp(j-15),threshold=0.1)

list\_tuple.append((coff\_vector,exp(j-15)))

return list\_tuple

lasso\_trace(X\_data[:200,:8],X\_data[:200,-1],lambda\_1=30)

data\_1=lasso\_trace(X\_data[:200,:8],X\_data[:200,-1],lambda\_1=30)

Data\_list=[]

Datalist\_ind=[]

for i in range(len(data\_1)):

Datalist\_ind.append(data\_1[i][1])

Data\_list.append(Datalist\_ind)

#X矩阵的形成

Datalist\_ind=[]

for j in range(data\_1[1][0].shape[0]):

for i in range(len(data\_1)):

Datalist\_ind.append(data\_1[i][0][j])

Data\_list.append(Datalist\_ind)

Datalist\_ind=[]

Data\_list\_ar=np.array(Data\_list)

#Y矩阵形成

pfi\_1=plt.figure(figsize=(4,3),dpi=220)

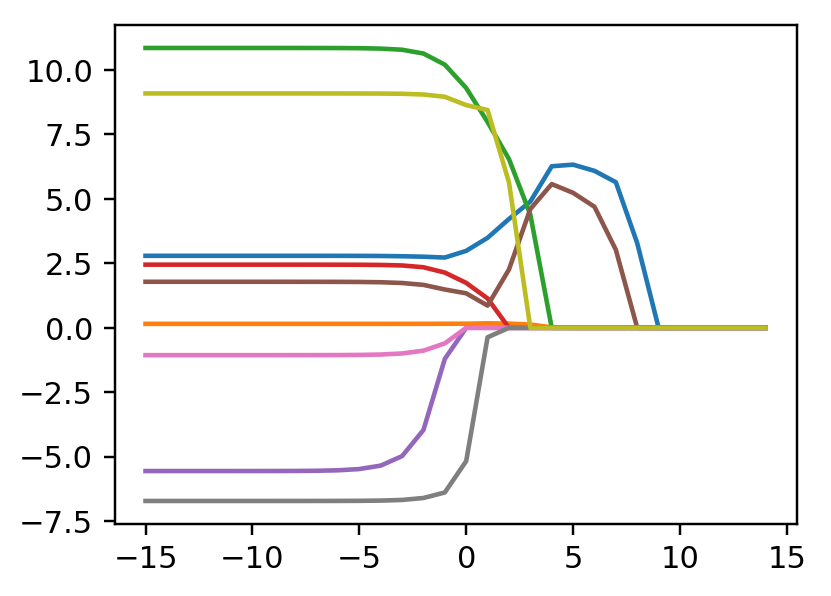
acs\_1=pfi\_1.add\_subplot(1,1,1)

#plt.xlim(0,1)

#plt.ylim(-15,15)

acs\_1.plot(np.log(Data\_list\_ar[0][:]).T,(Data\_list\_ar[1:,:]).T)

plt.show()



通过与岭轨迹图进行对比发现，随着λ的增大，很多系数逐渐趋近于0，但是岭回归没有这种情况，它的系数都是在0附近徘徊。而lasso回归是真正的不断有系数变为0。

1. **逻辑回归**

逻辑回归属于概率性非线性回归，分为二分类和多分类的回归模型。这里我们只对二分类模型做较为深入的探讨：

二分类意味着模型目标值只有两个，1和0；概率性意味着目标值为1或者0与一定概率相对应的。具体来讲，在n个相互独立自变量作用下，因变量取1的概率是p，取0的概率是（1-p），我们研究的重心是概率p与独立自变量的关系。

假设是n个独立的自变量，那么目标值为1和为0的概率比的自然对数为：



从上式我们可以轻易得知：



Log回归模型的建立要依赖于线性回归，具体来讲，是独立随机变量X1,...,Xn 的值的线性组合等于随机变量Y的优势比：





# 三，pandas

Pandas的数据结构，缺失数据处理，层次化索引等等

**Pandas的数据结构**

**Series:**

Series数据的建立 ->

In [**1**]: import numpy as np

In [**2**]: import pandas as pd

In [**3**]: pddata\_1=pd.Series([1,2,3])

In [**4**]: pddata\_1

Out[**4**]:

0 1

1 2

2 3

dtype: int64

数据取值和获取搜索索引

In [**7**]: pddata\_1.values

Out[**7**]: array([1, 2, 3], dtype=int64)

In [**8**]: pddata\_1.index

Out[**8**]: RangeIndex(start=0, stop=3, step=1)

例子：

In [**9**]: pddata\_2=pd.Series(['this','is','pandas'])

In [**10**]: pddata\_2.values

Out[**10**]: array(['this', 'is', 'pandas'], dtype=object)

In [**11**]: pddata\_2.index

Out[**11**]: RangeIndex(start=0, stop=3, step=1)

带有独特索引的Series数据：

In [**15**]: pddata\_3=pd.Series([344,33,-456],index=['andereas','hachenberger','dieter'])

In [**16**]: pddata\_3

Out[**16**]:

andereas 344

hachenberger 33

dieter -456

dtype: int64

In [**17**]: pddata\_3.index,pddata\_3.values

Out[**17**]:

(Index(['andereas', 'hachenberger', 'dieter'], dtype='object'),

array([ 344, 33, -456], dtype=int64))

例子：

In [**21**]: testa=[245,788,90]

In [**22**]: pddata\_4=pd.Series(testa,index=['wir','sind','Menschen'])

In [**23**]: pddata\_4

Out[**23**]:

wir 245

sind 788

Menschen 90

dtype: int64

In [**25**]: pddata\_4['wir']

Out[**25**]: 245

In [**27**]: pddata\_4[['wir','sind','Menschen']]

Out[**27**]:

wir 245

sind 788

Menschen 90

dtype: int64

例子：

In [**31**]: seriesdata\_1=pd.Series(['Profession','student','bechelor','professor','informatics developer'],index=['Name','Linker','Hamacher','Schuhmacher','Heintze'])

In [**32**]: seriesdata\_1

Out[**32**]:

Name Profession

Linker student

Hamacher bechelor

Schuhmacher professor

Heintze informatics developer

dtype: object

Series数据中的过滤、计算。

In [**33**]: seriesdata\_2=pd.Series([34,-4,-45,-37,32,9,1,3],index=['a','d','e','y','f','t','u','o'])

In [**34**]: seriesdata\_2

Out[**34**]:

a 34

d -4

e -45

y -37

f 32

t 9

u 1

o 3

dtype: int64

In [**35**]: seriesdata\_2[seriesdata\_2>1]

Out[**35**]:

a 34

f 32

t 9

o 3

dtype: int64

In [**36**]: seriesdata\_2[seriesdata\_2>0]

Out[**36**]:

a 34

f 32

t 9

u 1

o 3

dtype: int64

In [**37**]: seriesdata\_2\*3

Out[**37**]:

a 102

d -12

e -135

y -111

f 96

t 27

u 3

o 9

dtype: int64

In [**38**]: np.sin(seriesdata\_2)

Out[**38**]:

a 0.529083

d 0.756802

e -0.850904

y 0.643538

f 0.551427

t 0.412118

u 0.841471

o 0.141120

dtype: float64

Serie 和字典十分相似，因此原本字典的函数也可以用：

我们先复习下字典：

例子：

In [**39**]: dic={'Winne':178,'Johanis':189,'Banach':186}

In [**40**]: dic['Li']=176

In [**41**]: dic

Out[**41**]: {'Banach': 186, 'Johanis': 189, 'Li': 176, 'Winne': 178}

例子：

In [**19**]: list11=['zhang','wang','li']

In [**20**]: list22=range(3)

In [**21**]: indirect=zip(list22,list11)

In [**22**]: diction={index:value for index,value in indirect}

In [**23**]: diction

Out[**23**]: {0: 'zhang', 1: 'wang', 2: 'li'}

例3：

In [**26**]: list11=['a','b','c']

In [**27**]: list22=[1,2,3]

In [**28**]: dfgeg=zip(list11,list22)

In [**29**]: sss=dict((index,value) for index,value in dfgeg)

In [**30**]: sss

Out[**30**]: {'a': 1, 'b': 2, 'c': 3}

应用字典函数“in”和“Update”到Series

例子

In [**34**]: datas\_pys=pd.Series(range(4),index=['i','want','to','do'])

In [**35**]: datas\_pys

Out[**35**]:

i 0

want 1

to 2

do 3

dtype: int32

In [**36**]: 'want' in datas\_pys

Out[**36**]: True

例子：

In [**56**]: datas\_pys.update(pd.Series([2,3,4],index=['want','to','do']))

In [**57**]: datas\_pys

Out[**57**]:

i 0

want 2

to 3

do 4

dtype: int32

In [**58**]: s1 = pd.Series([1, 2, 3])

In [**59**]: s2 = pd.Series([4, 5, 6])

In [**60**]: s3 = pd.Series([4, 5, 6], index=[3,4,5])

In [**61**]: s1.append(s2)

Out[**61**]:

0 1

1 2

2 3

0 4

1 5

2 6

dtype: int64

In [**62**]: s1.append(s3)

Out[**62**]:

0 1

1 2

2 3

3 4

4 5

5 6

dtype: int64

In [**63**]: s1.append(s2, ignore\_index=True)

Out[**63**]:

0 1

1 2

2 3

3 4

4 5

5 6

dtype: int64

重新编号

字典可以直接转化为Serie

In [**68**]: dicttna={1:'foo',3:'drt',8:'tyue'}

In [**69**]: serie\_12=pd.Series(dicttna)

In [**70**]: serie\_12

Out[**70**]:

1 foo

3 drt

8 tyue

dtype: object

isnull和notnull函数可用于检测数据缺失。

In [**79**]: dit\_113={'lin':139,'zhang':134,'wang':173,'tan':None}

In [**80**]: serie\_123=pd.Series(dit\_113)

In [**81**]: serie\_123

Out[**81**]:

lin 139.0

tan NaN

wang 173.0

zhang 134.0

dtype: float64

In [**82**]: pd.isnull(serie\_123)

Out[**82**]:

lin False

tan True

wang False

zhang False

dtype: bool

In [**83**]: pd.notnull(serie\_123)

Out[**83**]:

lin True

tan False

wang True

zhang True

dtype: bool

serie\_123.notnull()

Out[85]:

lin True

tan False

wang True

zhang True

dtype: bool

Series 的索引可以修改：

In [**87**]: serie\_123.index

Out[**87**]: Index(['lin', 'tan', 'wang', 'zhang'], dtype='object')

In [**88**]: serie\_123.index=['lin', 'tan', 'shan', 'zhang']

In [**90**]: serie\_123

Out[**90**]:

lin 139.0

tan NaN

shan 173.0

zhang 134.0

dtype: float64

**DataFrame**

简单数据框的构成：

由字典直接形成

In [**93**]: sales=[12365,34563,45673,23461,89034]

In [**94**]: seller=['zhanghui','dongyibo','yangqian','liujuntao','zhangshanshan']

In [**95**]: sales\_quantity=[213,305,452,302,190]

In [**96**]: table\_sales={'Seller':seller,'Sales':sales,'SalesQuantity':sales\_quantity}

In [**97**]: framesample\_1=pd.DataFrame(table\_sales)

In [**98**]: framesample\_1

Out[**98**]:

Sales SalesQuantity Seller

0 12365 213 zhanghui

1 34563 305 dongyibo

2 45673 452 yangqian

3 23461 302 liujuntao

4 89034 190 zhangshanshan

与Series一样，索引会自动加上。

我们还可以指定列序列的左右顺序。

In [**99**]: framesample\_1=pd.DataFrame(table\_sales,columns=['SalesQuantity','Seller','Sales'])

In [**100**]: framesample\_1

Out[**100**]:

SalesQuantity Seller Sales

0 213 zhanghui 12365

1 305 dongyibo 34563

2 452 yangqian 45673

3 302 liujuntao 23461

4 190 zhangshanshan 89034

如果数据框中不包含所要找的值，自动返回NaN

In [**101**]: framesample\_1=pd.DataFrame(table\_sales,columns=['SalesQuantity','Seller','Sales','Profit'])

In [**102**]: framesample\_1

Out[**102**]:

SalesQuantity Seller Sales Profit

0 213 zhanghui 12365 NaN

1 305 dongyibo 34563 NaN

2 452 yangqian 45673 NaN

3 302 liujuntao 23461 NaN

4 190 zhangshanshan 89034 NaN

索引DataFrame 的数据。类似字典

列数据索引

In [**104**]: framesample\_1['Seller']

Out[**104**]:

0 zhanghui

1 dongyibo

2 yangqian

3 liujuntao

4 zhangshanshan

Name: Seller, dtype: object

行数据索引：

In [**106**]: framesample\_1.ix[2]

Out[**106**]:

SalesQuantity 452

Seller yangqian

Sales 45673

Prodit NaN

Name: 2, dtype: object

Dateframe的索引也可以改变。

In [**108**]: framesample\_1.index=[5,6,7,8,9]

In [**109**]: framesample\_1

Out[**109**]:

SalesQuantity Seller Sales Prodit

5 213 zhanghui 12365 NaN

6 305 dongyibo 34563 NaN

7 452 yangqian 45673 NaN

8 302 liujuntao 23461 NaN

9 190 zhangshanshan 89034 NaN

NAN值可以通过赋值语句替换。

In [**13**]: framesample\_1['Profit']=[278,967,654,234,432]

In [**14**]: framesample\_1

Out[**14**]:

SalesQuantity Seller Sales Profit

5 213 zhanghui 12365 278

6 305 dongyibo 34563 967

7 452 yangqian 45673 654

8 302 liujuntao 23461 234

9 190 zhangshanshan 89034 432

当用Series赋值时，可以精确地赋值到行列交叉位，没有指定的行列交叉位不会被赋值，将会以NAN的形式显示。

In [**17**]: val\_Series=pd.Series([0,1,1],index=[6,7,9])

In [**19**]: framesample\_1['Profit']=val\_Series

In [**20**]: framesample\_1

Out[**20**]:

SalesQuantity Seller Sales Profit

5 213 zhanghui 12365 NaN

6 305 dongyibo 34563 0.0

7 452 yangqian 45673 1.0

8 302 liujuntao 23461 NaN

9 190 zhangshanshan 89034 1.0

为不存在的列赋值会产生新的列。

In [**21**]: framesample\_1['loss']=pd.Series([165,0,0,34,0],index=range(5,10,1))

In [**22**]: framesample\_1

Out[**22**]:

SalesQuantity Seller Sales Profit loss

5 213 zhanghui 12365 NaN 165

6 305 dongyibo 34563 0.0 0

7 452 yangqian 45673 1.0 0

8 302 liujuntao 23461 NaN 34

9 190 zhangshanshan 89034 1.0 0

可用del删除列

In [**25**]: del framesample\_1['Profit']

In [**26**]: framesample\_1

Out[**26**]:

SalesQuantity Seller Sales loss

5 213 zhanghui 12365 165

6 305 dongyibo 34563 0

7 452 yangqian 45673 0

8 302 liujuntao 23461 34

9 190 zhangshanshan 89034 0

嵌套字典：

In [**28**]: Qtditc={'TeacherLiu':{'height':172,'weight':67,'age':34},'TeacherHuang':{'height':182,'weight':77,'age':36},'TeacherTao':{'height':192,'weight':98,'age':56} }

In [**29**]: Qtditc

Out[**29**]:

{'TeacherHuang': {'age': 36, 'height': 182, 'weight': 77},

'TeacherLiu': {'age': 34, 'height': 172, 'weight': 67},

'TeacherTao': {'age': 56, 'height': 192, 'weight': 98}}

如果把嵌套字典传给数据框，构造数据框，外键为列，内键为行索引。

In [**30**]: GetNewFrame=pd.DataFrame(Qtditc)

In [**31**]: GetNewFrame

Out[**31**]:

TeacherHuang TeacherLiu TeacherTao

age 36 34 56

height 182 172 192

weight 77 67 98

数据框的转置

In [**32**]: GetNewFrame.T

Out[**32**]:

age height weight

TeacherHuang 36 182 77

TeacherLiu 34 172 67

TeacherTao 56 192 98

Series组成的字典也可以直接转化为数据框：

In [**33**]: Popdic={'LinFeng':pd.Series([23,45,165],index=['age','weight','height']),'Zhangduoli':pd.Series([43,75,175],index=['age','weight','height']),'JinChang':pd.Series([51,46,185],index=['age','weight','height'])}

In [**35**]: Getnumerpop=pd.DataFrame(Popdic)

In [**36**]: Getnumerpop

Out[**36**]:

JinChang LinFeng Zhangduoli

age 51 23 43

weight 46 45 75

height 185 165 175

数据框名字的添加：

In [**45**]: Getnumerpop.columns.name='Name'

In [**46**]: Getnumerpop.index.name='personal information'

In [**47**]: Getnumerpop

Out[**47**]:

Name JinChang LinFeng Zhangduoli

personal information

age 51 23 43

weight 46 45 75

height 185 165 175

数据框的值：

In [**53**]: Getnumerpop.values

Out[**53**]:

array([[ 51, 23, 43],

[ 46, 45, 75],

[185, 165, 175]], dtype=int64)

数据框的index：

例1：

Getnumerpop.index=['p\_age','p\_weight','p\_height']

Getnumerpop.index

Out[57]: Index(['p\_age', 'p\_weight', 'p\_height'], dtype='object')

Getnumerpop

Out[58]:

Name JinChang LinFeng Zhangduoli

p\_age 51 23 43

p\_weight 46 45 75

p\_height 185 165 175

In [64]:index1=Getnumerpop.index

In [65]:index2=GetNewFrame.index

In [66]:index1.append(index2)

Out[66]: Index(['p\_age', 'p\_weight', 'p\_height', 'age', 'height', 'weight'], dtype='object')

例2

index2.intersection(index1)

Out[82]: Index([], dtype='object')

例3：

index2.union(index1)

Out[86]: Index(['age', 'height', 'p\_age', 'p\_height', 'p\_weight', 'weight'], dtype='object')

例4

index1.delete(1)

Out[94]: Index(['p\_age', 'p\_height'], dtype='object')

例5

index1.insert(1,'pheight')

Out[97]: Index(['p\_age', 'pheight', 'p\_weight', 'p\_height'], dtype='object')

index1.is\_monotonic

Out[104]: False

index1.is\_unique

Out[105]: True

index1.unique()

Out[107]: Index(['p\_age', 'p\_weight', 'p\_height'], dtype='object')

# Series索引重建

# In [2]: import numpy as np

# In [3]: import pandas as pd

In [**4**]: Seriestest\_1=pd.Series([10,-34,-89,36,50],index=['a','b','c','d','e'])

In [**5**]: Seriestest\_1

Out[**5**]:

a 10

b -34

c -89

d 36

e 50

dtype: int64

In [**7**]: Seriestest\_2=Seriestest\_1.reindex(['c','b','d','a','e','f'])

In [**8**]: Seriestest\_2

Out[**8**]:

c -89.0

b -34.0

d 36.0

a 10.0

e 50.0

f NaN

dtype: float64

NAN值可以被替换

In [**10**]: Seriestest\_2=Seriestest\_1.reindex(['c','b','d','a','e','f'],fill\_value=0)

In [**11**]: Seriestest\_2

Out[**11**]:

c -89

b -34

d 36

a 10

e 50

f 0

dtype: int64

使用ffill 和 bfill函数可以自动向前和向后补充缺失索引。**（缺失值补充**）

In [**6**]: testarray1=pd.Series([23,22,34],index=[3,7,9])

In [**10**]: test\_12=testarray1.reindex(range(11),method='ffill')

In [**11**]: test\_12

Out[**11**]:

0 NaN

1 NaN

2 NaN

3 23.0

4 23.0

5 23.0

6 23.0

7 22.0

8 22.0

9 34.0

10 34.0

dtype: float64

In [**12**]: test\_12=testarray1.reindex(range(11),method='bfill')

In [**13**]: test\_12

Out[**13**]:

0 23.0

1 23.0

2 23.0

3 23.0

4 22.0

5 22.0

6 22.0

7 22.0

8 34.0

9 34.0

10 NaN

dtype: float64

数据框的索引修改：（**reindex可以任意删除，添加，交换行列**）

对于数据框，reindex可以修改行索引和列，或两个都更改。

In [**14**]: #这里我用一个新的方法构造数据框

In [**18**]: frame\_1=pd.DataFrame(np.arange(9).reshape(3,3),index=['row1','row2','row3'],columns=['one','two','three'])

In [**19**]: frame\_1

Out[**19**]:

one two three

row1 0 1 2

row2 3 4 5

row3 6 7 8

In [**20**]: frame2=frame\_1.reindex(['row0','row1','row2','row3'])

In [**21**]: frame2

Out[**21**]:

one two three

row0 NaN NaN NaN

row1 0.0 1.0 2.0

row2 3.0 4.0 5.0

row3 6.0 7.0 8.0

In [**22**]: #reindex 修改列

In [**28**]: frame3=frame2.reindex(columns=['four','three'])

In [**29**]: frame3

Out[**29**]:

four three

row0 NaN NaN

row1 NaN 2.0

row2 NaN 5.0

row3 NaN 8.0

In [**30**]: #同时修改列和行索引

In [**31**]: frame4=frame\_1.reindex(['row1','row2','row3','row4'],columns=['five','three','six'])

In [**32**]: frame\_1

Out[**32**]:

one two three

row1 0 1 2

row2 3 4 5

row3 6 7 8

In [**33**]: frame4

Out[**33**]:

five three six

row1 NaN 2.0 NaN

row2 NaN 5.0 NaN

row3 NaN 8.0 NaN

row4 NaN NaN NaN

利用ix函数修改数据框行索引与列，快捷整洁，节约时间。但是Ix已经几乎被启用，现在用loc函数。

In [**35**]: frame5=frame\_1.reindex(columns=['one','four','two','three'])

In [**36**]: frame5

Out[**36**]:

one four two three

row1 0 NaN 1 2

row2 3 NaN 4 5

row3 6 NaN 7 8

In [**40**]: frame5['four']=pd.Series([34,56,78],index=['row1','row2','row3'])

In [**41**]: frame5

Out[**41**]:

one four two three

row1 0 34 1 2

row2 3 56 4 5

row3 6 78 7 8

In [**42**]: frame5.ix[['row2','row1','row3'],['one','two','four','three']]

C:\Users\dongfeng\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1: DeprecationWarning:

.ix is deprecated. Please use

.loc for label based indexing or

.iloc for positional indexing

See the documentation here:

http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated

"""Entry point for launching an IPython kernel.

Out[**42**]:

one two four three

row2 3 4 56 5

row1 0 1 34 2

row3 6 7 78 8

#现在用loc的比较多，ix已经不再推荐使用。

In [**44**]: frame5.loc[['row2','row1','row3'],['one','two','four','three']]

Out[**44**]:

one two four three

row2 3 4 56 5

row1 0 1 34 2

row3 6 7 78 8

指定轴上项目的丢弃：

Drop 函数应用到“Series”

In [**46**]: Seriestest\_1=pd.Series([233,356,997],index=['as','a','sample'])

In [**47**]: Seriestest\_1

Out[**47**]:

as 233

a 356

sample 997

dtype: int64

Seriestest\_1.drop('a')

Out[95]:

as 233

sample 997

dtype: int64

In [**48**]: #drop函数或者说方法返回的是一个在指定轴上删除指定值的新对象

Drop函数应用到“DataFrame”

Drop函数删除指定值时必须指定轴，如果不指定轴，默认零轴。也就是说删除“1”轴上的指定项时必须指定轴号。删除“0”轴上的指定项时不需要指定轴号。

In [**53**]: Dataframe\_1\_test=pd.DataFrame(np.floor(np.random.randn(4,4)),index=['xu','liu','zhang','feng'],columns=['stufe','klasse','degree','group'])

In [**54**]: Dataframe\_1\_test

Out[**54**]:

stufe klasse degree group

xu -1.0 0.0 -1.0 1.0

liu 0.0 -2.0 -1.0 -1.0

zhang 0.0 -3.0 -2.0 1.0

feng -1.0 0.0 -1.0 0.0

In [**57**]: Dataframe\_1\_test.drop(['stufe','klasse'],axis=1)

Out[**57**]:

degree group

xu -1.0 1.0

liu -1.0 -1.0

zhang -2.0 1.0

feng -1.0 0.0

In [**59**]: Dataframe\_1\_test.drop(['xu','feng'])

Out[**59**]:

stufe klasse degree group

liu 0.0 -2.0 -1.0 -1.0

zhang 0.0 -3.0 -2.0 1.0

In [**60**]: Dataframe\_1\_test.drop(['liu'])

Out[**60**]:

stufe klasse degree group

xu -1.0 0.0 -1.0 1.0

zhang 0.0 -3.0 -2.0 1.0

feng -1.0 0.0 -1.0 0.0

索引与选取

Series的索引和字典没有太大不同,唯一的区别是Series不仅可以通过设置的索引进行检索，也可以用默认的索引进行检索。另外Series也可以做切片，下面我们先看几个例子：

In [**6**]: Testser\_1=pd.Series([23,43,789,674,90,65],index=['apple','pear','persimmon','watermelon','strawberry','orange'])

In [**7**]: Testser\_1

Out[**7**]:

apple 23

pear 43

persimmon 789

watermelon 674

strawberry 90

orange 65

dtype: int64

#按默认索引检索

In [**9**]: Testser\_1[2:4]

Out[**9**]:

persimmon 789

watermelon 674

dtype: int64

#按给定索引检索

In [**25**]: Testser\_1['apple':'watermelon']

Out[**25**]:

apple 23

pear 43

persimmon 789

watermelon 674

dtype: int64

#注意，按给定标签（也叫索引）检索切片末端不会减一末端是被包含的，这就是说末端是封闭区间。（是’]’，不是‘）’）

In [**26**]: Testser\_1[['apple','watermelon']]

Out[**26**]:

apple 23

watermelon 674

dtype: int64

#注意上面两个例子的区别

#也可单个索引，按默认和按给定索引（也叫标签）均可检索

In [**27**]: Testser\_1['apple']

Out[**27**]: 23

In [**28**]: Testser\_1[0]

Out[**28**]: 23

#也可以按照bool值进行索引，此时检索不再按照给定的默认索引，而是与Series中的数据进行比较。

In [**30**]: Testser\_1[Testser\_1>100]

Out[**30**]:

persimmon 789

watermelon 674

dtype: int64

给Series赋值：

In [**31**]: Testser\_1['apple':'watermelon']=[112,22,32,42]

In [**32**]: Testser\_1

Out[**32**]:

apple 112

pear 22

persimmon 32

watermelon 42

strawberry 90

orange 65

dtype: int64

In [**34**]: Testser\_1['apple':'watermelon']=56

In [**35**]: Testser\_1

Out[**35**]:

apple 56

pear 56

persimmon 56

watermelon 56

strawberry 90

orange 65

dtype: int64

例子：

In [**37**]: Testser\_1[[2,3]]=34

In [**38**]: Testser\_1

Out[**38**]:

apple 56

pear 56

persimmon 34

watermelon 34

strawberry 90

orange 65

dtype: int64

In [**39**]: Testser\_1['apple']=34

In [**40**]: Testser\_1

Out[**40**]:

apple 34

pear 56

persimmon 34

watermelon 34

strawberry 90

orange 65

dtype: int64

DataFrame的索引

In [**5**]: Data=pd.DataFrame(np.arange(16).reshape(4,4),index=('one','two','three','four'),columns=('wir','sie','ihr','ich'))

In [**6**]: Data

Out[**6**]:

wir sie ihr ich

one 0 1 2 3

two 4 5 6 7

three 8 9 10 11

four 12 13 14 15

In [**7**]: Data['wir']

Out[**7**]:

one 0

two 4

three 8

four 12

Name: wir, dtype: int32

#单个索引时只能索引列，因此用“bool”值来检索时只能先检索出列数据，然后再与设定值比较，得出bool值Series，见下例：

In [**22**]: Data[Data['ihr']>6]

Out[**22**]:

wir sie ihr ich

three 8 9 10 11

four 12 13 14 15

Bool Series是可以当做索引的，见下

**Data[pd.Series([True,False,True,False],index=['one','two','three','four'])]**

**Out[44]:**

**wir sie ihr ich**

**one 0 1 2 3**

**three 8 9 10 11**

但这种索引容易出错，往往结果与我们的目的不一致，比如：

ata=pd.DataFrame([[1,2,3,4],[0,4,8,2],[0.3,0.6,2.3,4],[3,8,9,0]],index=('one','two','three','four'),columns=('wir','sie','ihr','ich'))

ata

Out[54]:

wir sie ihr ich

one 1.0 2.0 3.0 4

two 0.0 4.0 8.0 2

three 0.3 0.6 2.3 4

four 3.0 8.0 9.0 0

ata[ata['ihr']>3]

Out[55]:

wir sie ihr ich

two 0.0 4.0 8.0 2

four 3.0 8.0 9.0 0

我们想选出大于‘3’，而结果并非如此。

In [**8**]: Data[['wir','ihr']]

Out[**8**]:

wir ihr

one 0 2

two 4 6

three 8 10

four 12 14

In [**13**]: #DataFrame 切片只能依靠行索引

In [**16**]: Data['one':'three']

Out[**16**]:

wir sie ihr ich

one 0 1 2 3

two 4 5 6 7

three 8 9 10 11

In [**17**]: Data[1:3]

Out[**17**]:

wir sie ihr ich

two 4 5 6 7

three 8 9 10 11

bool值索引：

In [**23**]: Data>8

Out[**23**]:

wir sie ihr ich

one False False False False

two False False False False

three False True True True

four True True True True

In [**25**]: Data[Data>8]=0

In [**26**]: Data

Out[**26**]:

wir sie ihr ich

one 0 1 2 3

two 4 5 6 7

three 8 0 0 0

four 0 0 0 0

上面我们已经讲过单个索引时我们只能索引列，为了解决这个问题numpy创造了ix方法。但是ix函数或许太老了，目前已经有新的函数loc来替代它。通过这个函数你几乎可以为所欲为。loc函数只能用给定行列索引进行检索

In [**31**]: Data.loc['two']

Out[**31**]:

wir 4

sie 5

ihr 6

ich 7

Name: two, dtype: int32

In [**33**]: Data.loc['four','wir':'ihr']

Out[**33**]:

wir 12

sie 13

ihr 14

Name: four, dtype: int32

In [**34**]: Data.loc['four',['wir','ihr']]

Out[**34**]:

wir 12

ihr 14

Name: four, dtype: int32

Data.loc[['one','four'],['wir','ich']]

Out[136]:

wir ich

one 0 3

four 12 15

In [**35**]: Data.loc['one':'three','ich']

Out[**35**]:

one 3

two 7

three 11

Name: ich, dtype: int32

In [**38**]: Data.loc['one':,'wir']

Out[**38**]:

one 0

two 4

three 8

four 12

Name: wir, dtype: int32

bool值索引

In [**45**]: Data.loc[Data.wir>5,:'ich']

Out[**45**]:

wir sie ihr ich

three 8 9 10 11

four 12 13 14 15

Reindex 方法：

In [**50**]: Data

Out[**50**]:

wir sie ihr ich

one 0 1 2 3

two 4 5 6 7

three 8 9 10 11

four 12 13 14 15

In [**51**]: Data.reindex(index=['one','three','four','two'])

Out[**51**]:

wir sie ihr ich

one 0 1 2 3

three 8 9 10 11

four 12 13 14 15

two 4 5 6 7

In [**52**]: Data.reindex(columns=['wir','ihr','sie','ich'])

Out[**52**]:

wir ihr sie ich

one 0 2 1 3

two 4 6 5 7

three 8 10 9 11

four 12 14 13 15

Xs方法：根据标签选取单行和单列，返回‘Series’

In [**66**]: Data.xs('one')

Out[**66**]:

wir 0

sie 1

ihr 2

ich 3

Name: one, dtype: int32

In [**69**]: Data.xs('sie',axis=1)

Out[**69**]:

one 1

two 5

three 9

four 13

Name: sie, dtype: int32

Get\_value 方法得到数据框内单个值：

Data.get\_value('three','ihr')

Out[72]: 10

简单的算数运算和数据对齐

加法

Series数据相加必须按照索引一一相加，如果索引不能配对，将返回空值。

In [**1**]: import numpy as np

In [**2**]: import pandas as pd

In [**3**]: Se1=pd.Series([2.3,78,2.5,3.8],index=['as','a','pupil','and'])

In [**4**]: Se2=pd.Series([2,7.7,2.1,3.3,9,12],index=['as','a','studen','teacher','and','or'])

In [**6**]: tesseq\_1=Se1+Se2

In [**7**]: tesseq\_1

Out[**7**]:

a 85.7

and 12.8

as 4.3

or NaN

pupil NaN

studen NaN

teacher NaN

dtype: float64

与Series相同，DataFrame数据相加必须按照行索引和列一一相加，如果索引和列不能配对（也就是两个数据的行列标签不相等时），将返回空值。（先行索引匹配，然后在进行列匹配）

In [**22**]: add1=pd.DataFrame(np.arange(9).reshape(3,3),index=['ein','zwei','drei'],columns=list('bde'))

In [**23**]: add2=pd.DataFrame(np.arange(12).reshape(4,3),index=['zwei','sechs','drei','seven'],columns=list('bef'))

In [**26**]: add1

Out[**26**]:

b d e

ein 0 1 2

zwei 3 4 5

drei 6 7 8

In [**27**]: add2

Out[**27**]:

b e f

zwei 0 1 2

sechs 3 4 5

drei 6 7 8

seven 9 10 11

In [**24**]: Test\_add=add1+add2

In [**25**]: Test\_add

Out[**25**]:

b d e f

drei 12.0 NaN 15.0 NaN

ein NaN NaN NaN NaN

sechs NaN NaN NaN NaN

seven NaN NaN NaN NaN

zwei 3.0 NaN 6.0 NaN

这里我们发现如果我们没有把两个数据框的行标签和列完全一一对应或者说一一相等的话，就会出现很多空值。空值的重新赋值比较麻烦，所以尽量不要出现这种情况。

接下来我们尝试给这个数据框赋值。也就是说消除空值.

首先我们根据数据缺失的实际状况构造出一个新的数据框：

In [**9**]: Newframe=pd.DataFrame(np.array([[2,45,11,30],[-34,45,89,63],[44,90,36,27]]),index=['ein','sechs','seven'],columns=['b','d','e','f'])

In [**10**]: Test\_add.add(Newframe,fill\_value=0)

Out[**10**]:

b d e f

drei 12.0 NaN 15.0 NaN

ein 2.0 45.0 11.0 30.0

sechs -34.0 45.0 89.0 63.0

seven 44.0 90.0 36.0 27.0

zwei 3.0 NaN 6.0 NaN

In [**12**]: Diframe=Test\_add.add(Newframe,fill\_value=0)

In [**13**]: Diframe

Out[**13**]:

b d e f

drei 12.0 NaN 15.0 NaN

ein 2.0 45.0 11.0 30.0

sechs -34.0 45.0 89.0 63.0

seven 44.0 90.0 36.0 27.0

zwei 3.0 NaN 6.0 NaN

In [**26**]: Diframe.loc['drei','d']=36

In [**28**]: Diframe.loc['zwei','d']=0

In [**29**]: Diframe.loc['drei','f']=0

In [**30**]: Diframe.loc['zwei','f']=0

In [**31**]: Diframe

Out[**31**]:

b d e f

drei 12.0 36.0 15.0 0.0

ein 2.0 45.0 11.0 30.0

sechs -34.0 45.0 89.0 63.0

seven 44.0 90.0 36.0 27.0

zwei 3.0 0.0 6.0 0.0

减法乘法和除法

In [**32**]: S1=pd.Series([3,6,9])

In [**34**]: S2=pd.Series([4,3.2,3])

S3=S1-S2

S3

Out[38]:

0 -1.0

1 2.8

2 6.0

dtype: float64

S4=S1\*S2

S4

Out[40]:

0 12.0

1 19.2

2 27.0

dtype: float64

S5=S1/S2

S5

Out[42]:

0 0.750

1 1.875

2 3.000

dtype: float64

In [**43**]: Frame1=pd.DataFrame([[2,2,1],[1,1,3],[0,2,3]],index=['Nr1','Nr2','Nr3'],columns=['mon','tue','wen'])

In [**44**]: Frame2=pd.DataFrame([[1,3,1],[0,1,2],[2,0,1]],index=['Nr1','Nr2','Nr3'],columns=['mon','tue','wen'])

In [**45**]: Frame3=Frame1-Frame2

In [**46**]: Frame3

Out[**46**]:

mon tue wen

Nr1 1 -1 0

Nr2 1 0 1

Nr3 -2 2 2

In [**47**]: Frame4=Frame1\*Frame2

In [**48**]: Frame4

Out[**48**]:

mon tue wen

Nr1 2 6 1

Nr2 0 1 6

Nr3 0 0 3

In [**49**]: Frame5=Frame1/Frame2

In [**50**]: Frame5

Out[**50**]:

mon tue wen

Nr1 2.000000 0.666667 1.0

Nr2 inf 1.000000 1.5

Nr3 0.000000 inf 3.0

DataFrame 和 Series之间运算

数据框与Serie是之间的加减运算分为沿行和沿列运算的。

匹配列索引，沿行运算

In [**1**]: import numpy as np

In [**2**]: import pandas as pd

In [**3**]: Frame1=pd.DataFrame([[2,2,1],[1,1,3],[0,2,3]],index=['Nr1','Nr2','Nr3'],columns=['mon','tue','wen'])

In [**4**]: Frame1

Out[**4**]:

mon tue wen

Nr1 2 2 1

Nr2 1 1 3

Nr3 0 2 3

In [**7**]: S1=pd.Series([2,2,1],index=['mon','tue','wen'])

In [**8**]: Frame1-S1

Out[**8**]:

mon tue wen

Nr1 0 0 0

Nr2 -1 -1 2

Nr3 -2 0 2

这里我们要注意到数据框沿着行一行行的被Series减去。这种计算方法在Python里被称作广播。

如果某个索引不出现在在数据框的列索引或Series的索引里就会出现NaN。这对数据分析也是不利的，因此要求我们在设置索引时一定要一一对应。

In [**9**]: S1=pd.Series([2,2,1],index=['mon','wen','fri'])

In [**10**]: Frame1-S1

Out[**10**]:

fri mon tue wen

Nr1 NaN 0.0 NaN -1.0

Nr2 NaN -1.0 NaN 1.0

Nr3 NaN -2.0 NaN 1.0

匹配行索引，沿列运算(广播）

In [**11**]: S2=pd.Series([2,1,0],index=['Nr1','Nr2','Nr3'])

In [**15**]: Frame1.sub(S2,axis=0)

Out[**15**]:

mon tue wen

Nr1 0 0 -1

Nr2 0 0 2

Nr3 0 2 3

(乘法和除法可以按照.Multiply 和.devide方法就行求解！！！）

函数如何应用到数据框

适合数组的方法与函数，也可应用到pandas的数据结构上。

In [**17**]: frame12=pd.DataFrame(np.arange(12).reshape(4,3),index=['r1','r2','r3','r4'],columns=['c1','c2','c3'])

In [**18**]: frame12

Out[**18**]:

c1 c2 c3

r1 0 1 2

r2 3 4 5

r3 6 7 8

r4 9 10 11

In [**20**]: frame13=frame12\*-1

In [**21**]: frame13

Out[**21**]:

c1 c2 c3

r1 0 -1 -2

r2 -3 -4 -5

r3 -6 -7 -8

r4 -9 -10 -11

In [**22**]: np.abs(frame13)

Out[**22**]:

c1 c2 c3

r1 0 1 2

r2 3 4 5

r3 6 7 8

r4 9 10 11

In [**23**]: f=lambda x:x.max()-x.min()

In [**24**]: frame12.apply(f)

Out[**24**]:

c1 9

c2 9

c3 9

dtype: int64

In [**25**]: frame12.apply(f,axis=1)

Out[**25**]:

r1 2

r2 2

r3 2

r4 2

dtype: int64

.Min（.max）方法和min（max）函数在计算上没有区别，只是方法有更多的选择性。

In [**26**]: frame12.min()

Out[**26**]:

c1 0

c2 1

c3 2

dtype: int32

#不做声明，默认轴为‘0’意思为沿行标签操作

#更多选择性

In [**37**]: frame12.min(0)

Out[**37**]:

c1 0

c2 1

c3 2

dtype: int32

In [**38**]: frame12.min(1)

Out[**38**]:

r1 0

r2 3

r3 6

r4 9

dtype: int32

In [**28**]: np.min(frame12)

Out[**28**]:

c1 0

c2 1

c3 2

dtype: int32

Lamda 函数应用：

In [**23**]: f=lambda x:x.max()-x.min()

In [**24**]: frame12.apply(f)

Out[**24**]:

c1 9

c2 9

c3 9

dtype: int64

In [**25**]: frame12.apply(f,axis=1)

Out[**25**]:

r1 2

r2 2

r3 2

r4 2

dtype: int64

#应用apply方法可以得到更为整齐的结果，试比较例1和例2

In [**6**]: frame11=pd.DataFrame(np.arange(12).reshape(3,4),index=['r1','r2','r3'],columns=['c1','c2','c3','c4'])

In [**13**]: def table\_extrem (x):

    ...: return pd.Series([x.min(),x.max()],index=['min','max'])

    ...:

例子1：

In [**14**]: table\_extrem(frame11)

Out[**14**]:

min c1 0

c2 1

c3 2

c4 3

dtype: int32

max c1 8

c2 9

c3 10

c4 11

dtype: int32

dtype: object

例子2.

In [**15**]: frame11.apply(table\_extrem)

Out[**15**]:

c1 c2 c3 c4

min 0 1 2 3

max 8 9 10 11

# DataFrame 的格式化（十分有用）

In [**16**]: frame11=pd.DataFrame(np.random.randn(3,4),index=['r1','r2','r3'],columns=['c1','c2','c3','c4'])

In [**17**]: frame11

Out[**17**]:

c1 c2 c3 c4

r1 -2.061714 0.584658 -0.540976 0.090904

r2 -0.517271 -0.077818 0.163807 0.418174

r3 0.321513 0.769480 2.131075 -0.535560

In [**18**]: formatierung=lambda x:'%.2f' % x

In [**19**]: frame11.applymap(formatierung)

Out[**19**]:

c1 c2 c3 c4

r1 -2.06 0.58 -0.54 0.09

r2 -0.52 -0.08 0.16 0.42

r3 0.32 0.77 2.13 -0.54

排序和排名

Series 排序：

In [**118**]: import numpy as np

In [**119**]: import pandas as pd

In [**120**]: project\_1=pd.Series(np.arange(1,5),index=['a','d','e','f'])

In [**121**]: project\_1

Out[**121**]:

a 1

d 2

e 3

f 4

dtype: int32

In [**122**]: project\_1.sort\_index()

Out[**122**]:

a 1

d 2

e 3

f 4

dtype: int32

DataFrame 排序：

In [**124**]: DataFrame\_12=pd.DataFrame(np.arange(8).reshape(2,4),index=['r1','r2'],columns=['ted','lie','qiu','send'])

In [**125**]: DataFrame\_12.sort\_index()

Out[**125**]:

ted lie qiu send

r1 0 1 2 3

r2 4 5 6 7

#给行标签排序

In [**126**]: DataFrame\_12.sort\_index(axis=1)

Out[**126**]:

lie qiu send ted

r1 1 2 3 0

r2 5 6 7 4

#给列标签排序

上面的程序行标签和列标签（也叫列索引）都是按升序进行排列。我们也可以按降序排列数据框。

In [**7**]: DataFrame\_12.sort\_index(axis=1,ascending=False)

Out[**7**]:

ted send qiu lie

r1 0 3 2 1

r2 4 7 6 5

我们上面都是按标签（索引）进行排序，其实我们还可以用sort\_values方法对Series按值进行排序：

In [**15**]: Test\_series\_1=pd.Series([3,6,2,9,1,0],index=['r1','r2','r3','r4','r5','r6'])

In [**16**]: Test\_series\_1

Out[**16**]:

r1 3

r2 6

r3 2

r4 9

r5 1

r6 0

dtype: int64

In [**17**]: Test\_series\_1.sort\_values()

Out[**17**]:

r6 0

r5 1

r3 2

r1 3

r2 6

r4 9

dtype: int64

我们也可以对DataFrame进行按值排序

In [**18**]: Dataffdic=pd.DataFrame({'liu':[-1,0,2,-3],'wang':[3,2,-5,7],'jiang':[1,0,1,5]})

In [**19**]: Dataffdic

Out[**19**]:

jiang liu wang

0 1 -1 3

1 0 0 2

2 1 2 -5

3 5 -3 7

In [**20**]: Dataffdic.sort\_index(by='liu')#对'liu'列进行排序

C:\Users\dongfeng\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1: FutureWarning: by argument to sort\_index is deprecated, pls use .sort\_values(by=...)

"""Entry point for launching an IPython kernel.

Out[**20**]:

jiang liu wang

3 5 -3 7

0 1 -1 3

1 0 0 2

2 1 2 -5

In [**21**]: #目前sort\_index方法对数据框某列的值进行排序已经落后，现在用新的方法sort\_values 对列值进行排序。

In [**22**]: Dataffdic.sort\_values(by='liu')

Out[**22**]:

jiang liu wang

3 5 -3 7

0 1 -1 3

1 0 0 2

2 1 2 -5

#由于按‘liu’列进行排序，所以我们看到‘liu’列的数值排序是正确的。由于排序是整行移动，所以在对‘liu’列进行排序时，‘jiang’列和‘wang’列数值顺序一定会随其变得杂乱无章。

下面我们讲讲排名，排名和排序从结果上讲是完全不同的，排序是把一个没有顺序的标签或数据按照符号的自然排序规则（比如如果升序排列字符串“as”在顺序排位上应该在“at”之前）或者数值的太小进行排列。排名是通过打破数据的平级关系，从而产生一种有级别差的顺序。通俗的解释。一组没有排名的数据他们在级别上等级的，然后我们通过某种运算规则，使每个数据变成一个名次数值。

另外，排序可以对数据值本身和标签进行操作。而排名通常对数据本身操作。

In [**1**]: import numpy as np

In [**2**]: import pandas as pd

In [**3**]: Series\_1=pd.Series(np.array([3,8,34,33]),index=['r1','r2','r3','r4'])

In [**4**]: Frame\_1=pd.DataFrame(np.array([[3,2,5,0],[7,4,12,13],[1,0,1,-3],[3,6,2,0]]),index=['r1','r2','r3','r4'],columns=['a','b','c','d'])

Series排名的四种方法

1. ‘average’法

In [**6**]: Series\_1.rank(method='average')

Out[**6**]:

r1 1.0

r2 2.0

r3 4.0

r4 3.0

dtype: float64

1. ‘min’法

In [**8**]: Series\_1.rank(method='min')

Out[**8**]:

r1 1.0

r2 2.0

r3 4.0

r4 3.0

dtype: float64

1. ‘max’法

In [**7**]: Series\_1.rank(method='max')

Out[**7**]:

r1 1.0

r2 2.0

r3 4.0

r4 3.0

dtype: float64

1. ‘first’法

In [**9**]: Series\_1.rank(method='first')

Out[**9**]:

r1 1.0

r2 2.0

r3 4.0

r4 3.0

dtype: float64

上面所有的方法都是产生升序排名的结果，他们也可以产生降序的效果，如果我们填加“ascending”方法。

Series\_1.rank(ascending=False, method='first')

Out[19]:

r1 4.0

r2 3.0

r3 1.0

r4 2.0

dtype: float64

DataFrame的排名

在Series排名上，我们没有发现这四种方法的区别，他们一定有区别，只是我们这个Series的排名结果就是这样，没有显现出区别。我们下面通过DataFrame来区别这四种方法。

In [**14**]: Frame\_1.rank(method='first')

Out[**14**]:

a b c d

r1 2.0 2.0 3.0 2.0

r2 4.0 3.0 4.0 4.0

r3 1.0 1.0 1.0 1.0

r4 3.0 4.0 2.0 3.0

In [**15**]: Frame\_1.rank(method='average')

Out[**15**]:

a b c d

r1 2.5 2.0 3.0 2.5

r2 4.0 3.0 4.0 4.0

r3 1.0 1.0 1.0 1.0

r4 2.5 4.0 2.0 2.5

In [**16**]: Frame\_1.rank(method='min')

Out[**16**]:

a b c d

r1 2.0 2.0 3.0 2.0

r2 4.0 3.0 4.0 4.0

r3 1.0 1.0 1.0 1.0

r4 2.0 4.0 2.0 2.0

In [**17**]: Frame\_1.rank(method='max')

Out[**17**]:

a b c d

r1 3.0 2.0 3.0 3.0

r2 4.0 3.0 4.0 4.0

r3 1.0 1.0 1.0 1.0

r4 3.0 4.0 2.0 3.0

我们发现这四种排名的方式是不一样的，尽管是不一样的，但是排名这个目的都十分正确的实现了。仅仅是穿了不同外壳。

上面都是逐行排名，实际上我们还可以逐列排名。

In [**20**]: Frame\_1.rank(axis=1,method='max')

Out[**20**]:

a b c d

r1 3.0 2.0 4.0 1.0

r2 2.0 1.0 3.0 4.0

r3 4.0 2.0 4.0 1.0

r4 3.0 4.0 2.0 1.0

In [**21**]: Frame\_1.rank(axis=1,method='first')

Out[**21**]:

a b c d

r1 3.0 2.0 4.0 1.0

r2 2.0 1.0 3.0 4.0

r3 3.0 2.0 4.0 1.0

r4 3.0 4.0 2.0 1.0

上面就是逐列排名。个人认为‘first’法排名最为规整，且符合我们的习惯思维。

名词解释：

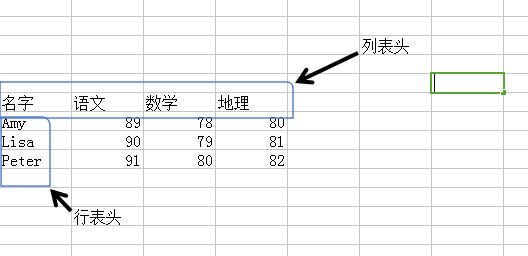
逐列排名（或排序）实际上对一行数据进行排名。

逐行排名（或排序）实际上对一列数据进行排名。

计算机默认逐行排名。

带有重复值的轴索引

对于数据框来说，轴索引就是指的行标签（或者叫行索引）或列标签（或者叫列索引），也就是我们Excel里的行表头和列表头。如下图



很多时候我们要求轴索引是唯一的，但这个要求并不是强制性的，很多时候它可

以重复，重复的轴索引对我们数据分析人员来说，不见得是坏事。

例子:

In [**1**]: import numpy as np

In [**2**]: import pandas as pd

In [**3**]: framne\_repeat=pd.DataFrame(np.arange(16).reshape(4,4),index=['r1','r2','r3','r1'],columns=['a','b','b','c'])

In [**4**]: framne\_repeat

Out[**4**]:

a b b c

r1 0 1 2 3

r2 4 5 6 7

r3 8 9 10 11

r1 12 13 14 15

In [**5**]: framne\_repeat['b']

Out[**5**]:

b b

r1 1 2

r2 5 6

r3 9 10

r1 13 14

In [**8**]: framne\_repeat.loc['r1']

Out[**8**]:

a b b c

r1 0 1 2 3

r1 12 13 14 15

In [**12**]: framne\_repeat.loc[('r1')]

Out[**12**]:

a b b c

r1 0 1 2 3

r1 12 13 14 15

上面两个例子可以看出framne\_repeat.loc['r1']和framne\_repeat.loc[('r1')]运行结果是一样的。也就是说加不加括号都无所谓。

带有重复轴索引的Series

In [**7**]: testserie\_1=pd.Series(np.array([1,2,3,9,10,3]),index=['a','b','a','d','e','a'])

In [**8**]: testserie\_1

Out[**8**]:

a 1

b 2

a 3

d 9

e 10

a 3

dtype: int32

In [**9**]: testserie\_1['a']

Out[**9**]:

a 1

a 3

a 3

dtype: int32

可以通过is\_unique 函数来判断是否存在重复轴索引。

In [**1**]: import pandas as pd

In [**2**]: import numpy as np

In [**3**]: testserie\_1=pd.Series(np.array([1,2,3,9,10,3]),index=['a','b','a','d','e','a'])

In [**4**]: testserie\_1.is\_unique

Out[**4**]: False

In [**5**]: #False值说明有重复轴索引

In [**6**]: #如果对没有重复值的索引进行索引时，返回一个标量。

In [**7**]: testserie\_1['b']

Out[**7**]: 2

Serie 和 Dataframe 的计算

dataFrame\_1=pd.DataFrame([range(4),[np.nan,2,3.6,0.9],[1.2,4,6,7],[3.4,7.9,0.4,8]],index=['r1','r2','r3','r4'],columns=['c1','c2','c3','c4'])

dataFrame\_1

Out[9]:

c1 c2 c3 c4

r1 0.0 1.0 2.0 3.0

r2 NaN 2.0 3.6 0.9

r3 1.2 4.0 6.0 7.0

r4 3.4 7.9 0.4 8.0

dataFrame\_1.sum()

Out[12]:

c1 4.6

c2 14.9

c3 12.0

c4 18.9

dtype: float64

dataFrame\_1.sum(1)

Out[13]:

r1 6.0

r2 6.5

r3 18.2

r4 19.7

dtype: float64

只要不是整行都是NaN值，python自动排除NAN值然后进行计算。通过skipna可以禁止该功能。

dataFrame\_1.sum(axis=1,skipna=False)

Out[22]:

r1 6.0

r2 NaN

r3 18.2

r4 19.7

dtype: float64

dataFrame\_1.idxmax()

Out[23]:

c1 r4

c2 r4

c3 r3

c4 r4

dtype: object

#求出每列最大值

dataFrame\_1.idxmax(1)

Out[7]:

r1 c4

r2 c3

r3 c4

r4 c4

dtype: object

#求出每行最大值

dataFrame\_1.idxmin(1)

Out[9]:

r1 c1

r2 c4

r3 c1

r4 c3

dtype: object

求出每行最小值，空值不参与比较。

dataFrame\_1.idxmin(axis=1,skipna=False)

Out[10]:

r1 c1

r2 NaN

r3 c1

r4 c3

dtype: object

#修改skipna的默认值”True”,可以修改NaN值自动忽略功能。

#累计加

按列累计加，遇到NaN值自动忽略，不参与运算。

dataFrame\_1.cumsum()

Out[8]:

c1 c2 c3 c4

r1 0.0 1.0 2.0 3.0

r2 NaN 3.0 5.6 3.9

r3 1.2 7.0 11.6 10.9

r4 4.6 14.9 12.0 18.9

dataFrame\_1.cumsum(1)

Out[10]:

c1 c2 c3 c4

r1 0.0 1.0 3.0 6.0

r2 NaN 2.0 5.6 6.5

r3 1.2 5.2 11.2 18.2

r4 3.4 11.3 11.7 19.7

dataFrame\_1.describe()

Out[12]:

c1 c2 c3 c4

count 3.000000 4.000 4.000000 4.00000

mean 1.533333 3.725 3.000000 4.72500

std 1.724336 3.050 2.388863 3.34203

min 0.000000 1.000 0.400000 0.90000

25% 0.600000 1.750 1.600000 2.47500

50% 1.200000 3.000 2.800000 5.00000

75% 2.300000 4.975 4.200000 7.25000

max 3.400000 7.900 6.000000 8.00000

#上面的分位数我们后面会有介绍

去重，值计数以及值资格判断

series\_3=pd.Series(['c','d','we','a','c','d','e','f','we'])

series\_3.unique()

Out[25]: array(['c', 'd', 'we', 'a', 'e', 'f'], dtype=object)

#通过uique方法的调用，我们立即可以去除Series中的重复的项目

serie\_unique=series\_3.unique()

serie\_unique.sort()

serie\_unique

Out[40]: array(['a', 'c', 'd', 'e', 'f', 'we'], dtype=object)

#通常去重后要进行排序

通过value\_counts()方法对Series中的值进行计数

series\_3.value\_counts()

Out[43]:

we 2

c 2

d 2

a 1

f 1

e 1

dtype: int64

我们可以通过这种方法求出指定元素出现的次数。

serie\_nr=series\_3.value\_counts()

serie\_nr['we']

Out[46]: 2

上面我们看到都是value\_counts()作为方法在应用，其实它也可以作为函数（或者叫Pandas的方法）单独应用。

pd.value\_counts(series\_3.values)

Out[47]:

we 2

c 2

d 2

a 1

f 1

e 1

dtype: int64

也可以不把计算内容做降序处理：

pd.value\_counts(series\_3.values,sort=False)

Out[49]:

e 1

d 2

f 1

a 1

c 2

we 2

dtype: int64

value\_counts()作为单独函数使用时，是一种超级牛的函数，他可以使用到任何序列和数组。

看下面例子：

list\_1=[1,3,4,5,2,6,8,4,3,8,9]

pd.value\_counts(list\_1)

Out[51]:

8 2

4 2

3 2

9 1

6 1

5 1

2 1

1 1

dtype: int64

tuple\_1=1,2,4,3,2,5,6,3,9

pd.value\_counts(tuple\_1)

Out[54]:

3 2

2 2

9 1

6 1

5 1

4 1

1 1

dtype: int64

array\_1=[1,2,3,4,52,3,2]

pd.value\_counts(array\_1)

Out[56]:

3 2

2 2

4 1

1 1

52 1

dtype: int64

series\_3

Out[60]:

0 c

1 d

2 we

3 a

4 c

5 d

6 e

7 f

8 we

dtype: object

series\_3.isin(['a','c'])

Out[61]:

0 True

1 False

2 False

3 True

4 True

5 False

6 False

7 False

8 False

dtype: bool

series\_3.isin(['a'])

Out[63]:

0 False

1 False

2 False

3 True

4 False

5 False

6 False

7 False

8 False

dtype: bool

#这是一个布尔Series，我们可以把他作为角码选出我们指定值在Series里的所有信息。

boolmatrix\_2=series\_3.isin(['a','c'])

series\_3[boolmatrix\_2]

Out[68]:

0 c

3 a

4 c

dtype: object

我们也可以把value\_counts()用到数据框，会得到一个你预料不到的结果。尽管这个结果很奇怪，但他却是巧妙的导出频数分布图。

dataFrame\_1

Out[69]:

c1 c2 c3 c4

r1 0.0 1.0 2.0 3.0

r2 NaN 2.0 3.6 0.9

r3 1.2 4.0 6.0 7.0

r4 3.4 7.9 0.4 8.0

dataFrame\_1.fillna(0)

Out[71]:

c1 c2 c3 c4

r1 0.0 1.0 2.0 3.0

r2 0.0 2.0 3.6 0.9

r3 1.2 4.0 6.0 7.0

r4 3.4 7.9 0.4 8.0

dataFrame\_2=dataFrame\_1.fillna(0)

result=dataFrame\_2.apply(pd.value\_counts).fillna(0)

result

Out[74]:

c1 c2 c3 c4

0.0 2.0 0.0 0.0 0.0

0.4 0.0 0.0 1.0 0.0

0.9 0.0 0.0 0.0 1.0

1.0 0.0 1.0 0.0 0.0

1.2 1.0 0.0 0.0 0.0

2.0 0.0 1.0 1.0 0.0

3.0 0.0 0.0 0.0 1.0

3.4 1.0 0.0 0.0 0.0

3.6 0.0 0.0 1.0 0.0

4.0 0.0 1.0 0.0 0.0

6.0 0.0 0.0 1.0 0.0

7.0 0.0 0.0 0.0 1.0

7.9 0.0 1.0 0.0 0.0

8.0 0.0 0.0 0.0 1.0

NAN数据处理，

在数据处理工作中，我们经常遇到缺失数据，它们的处理往往很费时间。Pandas设计之初，就已经考虑到这种情况。可以说，快速轻松地处理缺失数据是pandas最大优点之一。

#NaN值和None值都可以被当做空值处理：

series\_1=pd.Series([1,np.nan,3,None,2.3,6])

series\_1

Out[5]:

0 1.0

1 NaN

2 3.0

3 NaN

4 2.3

5 6.0

dtype: float64

缺失值得快速补充法

frame\_123=pd.DataFrame([[1,np.nan,np.nan,2,3],[1,2,3,None,8],[6,2.3,8,None,3],[2,3,2,3,2],[3.2,8.9,3,4.5,3],[2,3,0.45,None,8]])

frame\_123

Out[8]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3

1 1.0 2.0 3.00 NaN 8

2 6.0 2.3 8.00 NaN 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 NaN 8

pd.isnull(frame\_123)

Out[9]:

0 1 2 3 4

0 False True True False False

1 False False False True False

2 False False False True False

3 False False False False False

4 False False False False False

5 False False False True False

boolmatrix\_1=pd.isnull(frame\_123)

frame\_123[boolmatrix\_1]=0

frame\_123

Out[13]:

0 1 2 3 4

0 1.0 0.0 0.00 2.0 3

1 1.0 2.0 3.00 0.0 8

2 6.0 2.3 8.00 0.0 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 0.0 8

Dropna过滤NaN数据

series\_23=pd.Series([1,4,3,np.nan,4.5,None,3.4,4.5,6,np.nan,4,5,3])

series\_23

Out[20]:

0 1.0

1 4.0

2 3.0

3 NaN

4 4.5

5 NaN

6 3.4

7 4.5

8 6.0

9 NaN

10 4.0

11 5.0

12 3.0

dtype: float64

series\_23.dropna()

Out[21]:

0 1.0

1 4.0

2 3.0

4 4.5

6 3.4

7 4.5

8 6.0

10 4.0

11 5.0

12 3.0

dtype: float64

也可以通过Notnull

series\_23[series\_23.notnull()]

Out[22]:

0 1.0

1 4.0

2 3.0

4 4.5

6 3.4

7 4.5

8 6.0

10 4.0

11 5.0

12 3.0

dtype: float64

Dropna 用于DataFrame和用于Series稍有不同。用于DataFrame，含有NaN值的行会被删去。

frame\_123=pd.DataFrame([[1,np.nan,np.nan,2,3],[1,2,3,None,8],[6,2.3,8,None,3],[2,3,2,3,2],[3.2,8.9,3,4.5,3],[2,3,0.45,None,8]])

frame\_123

Out[28]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3

1 1.0 2.0 3.00 NaN 8

2 6.0 2.3 8.00 NaN 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 NaN 8

frame\_123.dropna()

Out[27]:

0 1 2 3 4

3 2.0 3.0 2.0 3.0 2

4 3.2 8.9 3.0 4.5 3

使用how=’all’只会消除全为NaN的行：

matrix\_2=frame\_123.dropna(how='all')

matrix\_2

Out[32]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3

1 1.0 2.0 3.00 NaN 8

2 6.0 2.3 8.00 NaN 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 NaN 8

列行的添加和丢弃

frame\_123

Out[47]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3

1 1.0 2.0 3.00 NaN 8

2 6.0 2.3 8.00 NaN 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 NaN 8

frame\_123[5]=[1,3,56,32,np.nan,0.7]

frame\_123

Out[49]:

0 1 2 3 4 5

0 1.0 NaN NaN 2.0 3 1.0

1 1.0 2.0 3.00 NaN 8 3.0

2 6.0 2.3 8.00 NaN 3 56.0

3 2.0 3.0 2.00 3.0 2 32.0

4 3.2 8.9 3.00 4.5 3 NaN

5 2.0 3.0 0.45 NaN 8 0.7

#列的添加

frame\_123

Out[47]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3

1 1.0 2.0 3.00 NaN 8

2 6.0 2.3 8.00 NaN 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 NaN 8

frame\_123.loc[6,:]=[np.nan,2,2.4,5,6]

frame\_123

Out[53]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3.0

1 1.0 2.0 3.00 NaN 8.0

2 6.0 2.3 8.00 NaN 3.0

3 2.0 3.0 2.00 3.0 2.0

4 3.2 8.9 3.00 4.5 3.0

5 2.0 3.0 0.45 NaN 8.0

6 NaN 2.0 2.40 5.0 6.0

#行的添加

frame\_123

Out[55]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3

1 1.0 2.0 3.00 NaN 8

2 6.0 2.3 8.00 NaN 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 NaN 8

frame\_123[5]=np.nan

frame\_123

Out[57]:

0 1 2 3 4 5

0 1.0 NaN NaN 2.0 3 NaN

1 1.0 2.0 3.00 NaN 8 NaN

2 6.0 2.3 8.00 NaN 3 NaN

3 2.0 3.0 2.00 3.0 2 NaN

4 3.2 8.9 3.00 4.5 3 NaN

5 2.0 3.0 0.45 NaN 8 NaN

frame\_123[5]=np.nan

frame\_123

Out[57]:

0 1 2 3 4 5

0 1.0 NaN NaN 2.0 3 NaN

1 1.0 2.0 3.00 NaN 8 NaN

2 6.0 2.3 8.00 NaN 3 NaN

3 2.0 3.0 2.00 3.0 2 NaN

4 3.2 8.9 3.00 4.5 3 NaN

5 2.0 3.0 0.45 NaN 8 NaN

frame\_123.dropna(axis=1,how='all')

Out[60]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3

1 1.0 2.0 3.00 NaN 8

2 6.0 2.3 8.00 NaN 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 NaN 8

frame\_1=pd.DataFrame(np.random.randn(7,4))

frame\_1

Out[65]:

0 1 2 3

0 -0.668146 0.034772 0.482339 1.444138

1 -1.167959 -0.703595 0.641404 -1.100771

2 -1.657068 -2.038607 0.141572 2.525831

3 -1.869547 -0.291923 0.275511 -0.459739

4 -0.287525 -0.966589 2.145633 0.703735

5 0.499207 -0.385792 -1.192131 -1.679805

6 -0.529885 2.053872 0.970785 -0.733382

frame\_1.loc[:4,1]=np.nan;frame\_1.loc[:2,2]=np.nan

frame\_1

Out[67]:

0 1 2 3

0 -0.668146 NaN NaN 1.444138

1 -1.167959 NaN NaN -1.100771

2 -1.657068 NaN NaN 2.525831

3 -1.869547 NaN 0.275511 -0.459739

4 -0.287525 NaN 2.145633 0.703735

5 0.499207 -0.385792 -1.192131 -1.679805

6 -0.529885 2.053872 0.970785 -0.733382

[:4,1]注意这里切片是切到“4”，不是到“3”

frame\_1.dropna()

Out[72]:

0 1 2 3

5 0.499207 -0.385792 -1.192131 -1.679805

6 -0.529885 2.053872 0.970785 -0.733382

#把不想看的数据行部分赋值NaN，然后在通过dropna语句把含有NaN的行删除掉。

缺失数据的补充：

通过fillna指令我们可以更快地补充缺失数据。这与通过isnull把数据框转化为bool数据框，然后把bool数据框作为索引来赋值‘0’要快的多。（前面已经讲过）

但是这个指令只能赋一个值。通过字典，我们可以给每一列赋不同的值。

frame\_1.fillna({1:5,2:4})

Out[7]:

0 1 2 3

0 1.651360 5.000000 4.000000 -0.600997

1 -0.020649 5.000000 4.000000 -0.529843

2 -0.850476 5.000000 4.000000 -0.893813

3 0.812226 5.000000 0.614715 0.112509

4 0.074320 5.000000 1.145727 0.168718

5 -0.516371 -1.022050 0.774645 0.705847

6 0.617606 0.881843 -0.619870 -1.527961

\_=frame\_1.fillna(0,inplace=True)

frame\_1

Out[13]:

0 1 2 3

0 1.651360 0.000000 0.000000 -0.600997

1 -0.020649 0.000000 0.000000 -0.529843

2 -0.850476 0.000000 0.000000 -0.893813

3 0.812226 0.000000 0.614715 0.112509

4 0.074320 0.000000 1.145727 0.168718

5 -0.516371 -1.022050 0.774645 0.705847

6 0.617606 0.881843 -0.619870 -1.527961

#可以直接对数据框修改，也就是说可以直接改变源数据，不产生副本。

frame\_1=pd.DataFrame(np.random.randn(7,4));frame\_1.loc[:4,1]=np.nan;frame\_1.loc[:2,2]=np.nan

frame\_1

Out[107]:

0 1 2 3

0 -0.748195 NaN NaN -0.539246

1 -2.360077 NaN NaN -0.061078

2 -1.743350 NaN NaN -0.233060

3 -2.100371 NaN 0.187509 0.893321

4 -0.477253 NaN 1.876733 0.208389

5 0.770411 -2.262797 0.321646 -2.490215

6 1.265209 -0.453140 0.045817 0.951077

frame\_1.fillna({1:pd.Series([2,3,7,9,0],index=[0,1,2,3,4]),2:pd.Series([1,9,8],index=[0,1,2])})

Out[108]:

0 1 2 3

0 -0.748195 2.000000 1.000000 -0.539246

1 -2.360077 3.000000 9.000000 -0.061078

2 -1.743350 7.000000 8.000000 -0.233060

3 -2.100371 9.000000 0.187509 0.893321

4 -0.477253 0.000000 1.876733 0.208389

5 0.770411 -2.262797 0.321646 -2.490215

6 1.265209 -0.453140 0.045817 0.951077

层次化索引：

series\_1=pd.Series(np.random.randn(12),index=[['r1','r1','r1','t1','t1','s1','s1','s1','s1','p1','q1','q1'],[9,8,7,6,5,4,3,2,1,0,12,11]])

series\_1

Out[8]:

r1 9 1.536272

8 0.378965

7 -0.292986

t1 6 -0.254990

5 0.765191

s1 4 2.204928

3 0.662662

2 -0.595029

1 -1.905753

p1 0 -1.029524

q1 12 -1.038748

11 -1.192589

dtype: float64

series\_1.index

Out[9]:

MultiIndex(levels=[['p1', 'q1', 'r1', 's1', 't1'], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12]],

labels=[[2, 2, 2, 4, 4, 3, 3, 3, 3, 0, 1, 1], [9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 11, 10]])

可以直接索引外层标签，也就是第一层标签。

series\_1['s1']

Out[6]:

4 -0.012306

3 0.800679

2 -0.862986

1 0.048458

dtype: float64

**也可以做切片。但是外层标签必须是**

1. **经过排序的，必须！**
2. **每个外层标签的第一个字母不能不一致，要大写，全大写，要小写全小写。**

Series\_123=pd.Series(np.random.randint(10),index=[['apple','apple','apple','apricot','apricot','apricot','banana','banana','blackberry','cherry'],[3,2,5,6,0,4,9,11,21,17]])

Series\_123['apple']

Out[4]:

3 9

2 9

5 9

dtype: int64

Series\_123['apple':'banana']

Out[64]:

apple 3 5

2 5

5 5

apricot 6 5

0 5

4 5

banana 9 5

11 5

dtype: int64

Series\_123['apple':'apricot']

Out[65]:

apple 3 5

2 5

5 5

apricot 6 5

0 5

4 5

dtype: int64

Series\_123['apple':'blackberry']

Out[67]:

apple 3 5

2 5

5 5

apricot 6 5

0 5

4 5

banana 9 5

11 5

blackberry 21 5

dtype: int64

Series\_67=pd.Series(np.random.randint(10),index=[['apple','apple','apple','apricot','apricot','apricot','banana','banana','blackberry','cherry'],[3,2,5,6,5,4,9,5,21,17]])

Series\_67

Out[7]:

apple 3 4

2 4

5 4

apricot 6 4

5 4

4 4

banana 9 4

5 4

blackberry 21 4

cherry 17 4

dtype: int64

Series\_67[:,5]

Out[8]:

apple 4

apricot 4

banana 4

dtype: int64

#对于Series，中括号的第一个位置是代表外层标签，第二位置代表内存标签

例子：

Series\_67['apple',5]

Out[10]: 4

带有层次化索引的Series可以立马转化为数据框，反之，数据框也可以转化为带有层次化索引的Series

Series\_67.unstack()

Out[20]:

2 3 4 5 6 9 17 21

apple 4.0 4.0 NaN 4.0 NaN NaN NaN NaN

apricot NaN NaN 4.0 4.0 4.0 NaN NaN NaN

banana NaN NaN NaN 4.0 NaN 4.0 NaN NaN

blackberry NaN NaN NaN NaN NaN NaN NaN 4.0

cherry NaN NaN NaN NaN NaN NaN 4.0 NaN

#Series的内层索引变化为列标签，外层索引转化为行标签,并且重复索引删除，索引保持唯一性。

Series\_67.unstack(0)

Out[22]:

apple apricot banana blackberry cherry

2 4.0 NaN NaN NaN NaN

3 4.0 NaN NaN NaN NaN

4 NaN 4.0 NaN NaN NaN

5 4.0 4.0 4.0 NaN NaN

6 NaN 4.0 NaN NaN NaN

9 NaN NaN 4.0 NaN NaN

17 NaN NaN NaN NaN 4.0

21 NaN NaN NaN 4.0 NaN

#axis=0时，Series的内层索引变化为行标签，外层索引转化为列标签

Series\_67.unstack(1)

Out[24]:

2 3 4 5 6 9 17 21

apple 4.0 4.0 NaN 4.0 NaN NaN NaN NaN

apricot NaN NaN 4.0 4.0 4.0 NaN NaN NaN

banana NaN NaN NaN 4.0 NaN 4.0 NaN NaN

blackberry NaN NaN NaN NaN NaN NaN NaN 4.0

cherry NaN NaN NaN NaN NaN NaN 4.0 NaN

Series\_67.unstack().stack()

Out[29]:

apple 2 4.0

3 4.0

5 4.0

apricot 4 4.0

5 4.0

6 4.0

banana 5 4.0

9 4.0

blackberry 21 4.0

cherry 17 4.0

dtype: float64

Gamedata=pd.DataFrame(np.array([[31,27,24,60],[26,28,13,29],[27,17,12,27],[29,9,5,18],[31,12,4,70],[45,11,12,12]]),index=[['Morning','Morning','Morning','Afternoon','Afternoon','Afternoon'],[1,2,3,1,2,3]],columns=[['Junior','Junior','Youth','Youth'],['Zhang','Wang','Li','Zhao']])

Gamedata

Out[35]:

Junior Youth

Zhang Wang Li Zhao

Morning 1 31 27 24 60

2 26 28 13 29

3 27 17 12 27

Afternoon 1 29 9 5 18

2 31 12 4 70

3 45 11 12 12

Gamedata=pd.DataFrame(np.array([[31,27,24,60],[26,28,13,29],[27,17,12,27],[29,9,5,18],[31,12,4,70],[45,11,12,12]]),index=[['A\_Morning','A\_Morning','A\_Morning','B\_Afternoon','B\_Afternoon','B\_Afternoon'],[1,2,3,1,2,3]],columns=[['Junior','Junior','Youth','Youth'],['Zhang','Wang','Li','Zhao']])

Gamedata.index.names=['time','sequence']

#因为行有双层索引，因此需要两个名字

#列也是如此，也是两个名字

Gamedata.columns.names=['age','name']

Gamedata

Out[50]:

age Junior Youth

name Zhang Wang Li Zhao

time sequence

A\_Morning 1 31 27 24 60

2 26 28 13 29

3 27 17 12 27

B\_Afternoon 1 29 9 5 18

2 31 12 4 70

3 45 11 12 12

Gamedata['Junior']

Out[51]:

name Zhang Wang

time sequence

A\_Morning 1 31 27

2 26 28

3 27 17

B\_Afternoon 1 29 9

2 31 12

3 45 11

Gamedata.loc['Morning']

Out[31]:

Age Junior Youth

Name Zhang Wang Li Zhao

series\_nr

1 31 27 24 60

2 26 28 13 29

3 27 17 12 27

Gamedata.loc['Morning',2]

Out[32]:

Age Name

Junior Zhang 26

Wang 28

Youth Li 13

Zhao 29

Name: (Morning, 2), dtype: int32

多重标签的顺序互换：

Gamedata.swaplevel('sequence','time')

Out[52]:

age Junior Youth

name Zhang Wang Li Zhao

sequence time

1 A\_Morning 31 27 24 60

2 A\_Morning 26 28 13 29

3 A\_Morning 27 17 12 27

1 B\_Afternoon 29 9 5 18

2 B\_Afternoon 31 12 4 70

3 B\_Afternoon 45 11 12 12

Gamedata.swaplevel('age','name',axis=1)

Out[60]:

name Zhang Wang Li Zhao

age Junior Junior Youth Youth

time sequence

A\_Morning 1 31 27 24 60

2 26 28 13 29

3 27 17 12 27

B\_Afternoon 1 29 9 5 18

2 31 12 4 70

3 45 11 12 12

Gamedata.sortlevel(1)

C:\Users\dongfeng\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1: FutureWarning: sortlevel is deprecated, use sort\_index(level= ...)

"""Entry point for launching an IPython kernel.

Out[55]:

age Junior Youth

name Zhang Wang Li Zhao

time sequence

A\_Morning 1 31 27 24 60

B\_Afternoon 1 29 9 5 18

A\_Morning 2 26 28 13 29

B\_Afternoon 2 31 12 4 70

A\_Morning 3 27 17 12 27

B\_Afternoon 3 45 11 12 12

#sortlevel则根据单个级别的值进行排序，上例根据sequence列数据。

Gamedata.swaplevel('age','name',axis=1).sortlevel(0)

C:\Users\dongfeng\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1: FutureWarning: sortlevel is deprecated, use sort\_index(level= ...)

"""Entry point for launching an IPython kernel.

Out[61]:

name Zhang Wang Li Zhao

age Junior Junior Youth Youth

time sequence

A\_Morning 1 31 27 24 60

2 26 28 13 29

3 27 17 12 27

B\_Afternoon 1 29 9 5 18

2 31 12 4 70

3 45 11 12 12

#交换两层列标签，然后对第二行列标签排序

Gamedata

Out[62]:

age Junior Youth

name Zhang Wang Li Zhao

time sequence

A\_Morning 1 31 27 24 60

2 26 28 13 29

3 27 17 12 27

B\_Afternoon 1 29 9 5 18

2 31 12 4 70

3 45 11 12 12

Gamedata.swaplevel('age','name',axis=1).sortlevel(1)

C:\Users\dongfeng\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1: FutureWarning: sortlevel is deprecated, use sort\_index(level= ...)

"""Entry point for launching an IPython kernel.

Out[63]:

name Zhang Wang Li Zhao

age Junior Junior Youth Youth

time sequence

A\_Morning 1 31 27 24 60

B\_Afternoon 1 29 9 5 18

A\_Morning 2 26 28 13 29

B\_Afternoon 2 31 12 4 70

A\_Morning 3 27 17 12 27

B\_Afternoon 3 45 11 12 12

#交换两层行标签，然后对第二列行标签排序

根据级别计算：

Gamedata

Out[62]:

age Junior Youth

name Zhang Wang Li Zhao

time sequence

A\_Morning 1 31 27 24 60

2 26 28 13 29

3 27 17 12 27

B\_Afternoon 1 29 9 5 18

2 31 12 4 70

3 45 11 12 12

Gamedata.sum(level='sequence')

Out[64]:

age Junior Youth

name Zhang Wang Li Zhao

sequence

1 60 36 29 78

2 57 40 17 99

3 72 28 24 39

#每个级别包含两个元素，（因为此级别标签分别重复两次。）

Gamedata.sum(level='name',axis=1)

Out[68]:

name Li Wang Zhang Zhao

time sequence

A\_Morning 1 24 27 31 60

2 13 28 26 29

3 12 17 27 27

B\_Afternoon 1 5 9 29 18

2 4 12 31 70

3 12 11 45 12

每个级别只包含一个元素，因此无法相加，只能排序。

Gamedata.sum(level='age',axis=1)

Out[69]:

age Junior Youth

time sequence

A\_Morning 1 58 84

2 54 42

3 44 39

B\_Afternoon 1 38 23

2 43 74

3 56 24

#每个级别包含两个元素，分别两两相加。

Gamedata.sum(level='time')

Out[71]:

age Junior Youth

name Zhang Wang Li Zhao

time

A\_Morning 84 72 49 116

B\_Afternoon 105 32 21 100

每个级别包含三个元素，分别相加即可

把数据框的列转化为行索引：

frame\_1123=pd.DataFrame({'a':range(4),'b':range(4,0,-1),'c':['one','one','two','two'],'d':[0,1,2,3]})

frame\_1123

Out[73]:

a b c d

0 0 4 one 0

1 1 3 one 1

2 2 2 two 2

3 3 1 two 3

frame\_1224=frame\_1123.set\_index(['c','d'])

frame\_1224

Out[76]:

a b

c d

one 0 0 4

1 1 3

two 2 2 2

3 3 1

#我们发现列从数据框中消失，变成了双重行索引。

然而而这些列也可以不消失。

frame\_1123.set\_index(['c','d'],drop=False)

Out[79]:

a b c d

c d

one 0 0 4 one 0

1 1 3 one 1

two 2 2 2 two 2

3 3 1 two 3

我们可以把双重索引再转化到数据框的数据区域。

frame\_1123.reset\_index()

Out[82]:

index a b c d

0 0 0 4 one 0

1 1 1 3 one 1

2 2 2 2 two 2

3 3 3 1 two 3

面板数据：

Panel实质上是一个三维的数据框，3维说明其有三个轴，每个轴的含义如下：

* **items:** 0轴， 每个项目对应其中的一个DataFrame
* \*\*major\_axis（主轴）: \*\* 1轴，它是每个DataFrame的index
* \*\*minor\_axis（副轴）: \*\* 2轴，它是每个DataFrame的column

创建panel：

1. 数组创建,三维数组

Dataframe\_3 = pd.Panel(np.random.randn(2,5,4),items=['Item1','Item2'],major\_axis=pd.date\_range('2000-01-01','2000-01-05'),minor\_axis=['A','B','C','D'])

Dataframe\_4=pd.Panel(np.array([[[1,4,67,45],[34,56,2,0],[78,90,3,4],[4,23,67,5],[34,89,67,1]],[[67,90,64,7],[789,345,6,2],[33,89,467,8],[43,93,2,8],[33,74,89,6]]]),items=['Item1','Item2'],major\_axis=pd.date\_range('2000-01-01','2000-01-05'),minor\_axis=['A','B','C','D'])

Dataframe\_3

Out[107]:

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 5 (major\_axis) x 4 (minor\_axis)

Items axis: Item1 to Item2

Major\_axis axis: 2000-01-01 00:00:00 to 2000-01-05 00:00:00

Minor\_axis axis: A to D

Dataframe\_4

Out[108]:

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 5 (major\_axis) x 4 (minor\_axis)

Items axis: Item1 to Item2

Major\_axis axis: 2000-01-01 00:00:00 to 2000-01-05 00:00:00

Minor\_axis axis: A to D

1. 通过字典创建

data = {'Item1' : pd.DataFrame(np.random.randn(4, 3),index=pd.date\_range('2017-09-05','2017-09-08'),columns=['X','D','F']),'Item2' : pd.DataFrame(np.random.randn(4, 2),index=pd.date\_range('2017-09-05','2017-09-08'),columns=['X','D'])}

panel\_1=pd.Panel(data)

panel\_1

Out[115]:

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 4 (major\_axis) x 3 (minor\_axis)

Items axis: Item1 to Item2

Major\_axis axis: 2017-09-05 00:00:00 to 2017-09-08 00:00:00

Minor\_axis axis: D to X

1. 通过数据框创建

midx = pd.MultiIndex(levels=[['one', 'two'], ['x','y']], labels=[[1,1,0,0],[1,0,1,0]])

midx

Out[117]:

MultiIndex(levels=[['one', 'two'], ['x', 'y']],

labels=[[1, 1, 0, 0], [1, 0, 1, 0]])

第一层行标是['one', 'two']，labels[1, 1, 0, 0]；第二层行标是 ['x','y']，labels[1, 0, 1, 0]，注意’one’对应‘0’，’two’对应‘1’

df = pd.DataFrame({'A':[1,2,3,4],'B':[5,6,7,8]},index=midx)

df

Out[119]:

A B

two y 1 5

x 2 6

one y 3 7

x 4 8

df.to\_panel()

面板操作

1. 选取

Dataframe\_3['Item1']

Out[124]: Dataframe\_3

A B C D

2000-01-01 0.836830 -1.856472 0.345340 0.576771

2000-01-02 -1.529758 -1.646630 0.635996 -0.337408

2000-01-03 0.451765 0.156648 -1.225328 -0.177641

2000-01-04 1.123645 0.364549 0.684536 1.558884

2000-01-05 -0.082263 1.472391 -0.379373 2.410845

Dataframe\_3.major\_axis

Out[128]:

DatetimeIndex(['2000-01-01', '2000-01-02', '2000-01-03', '2000-01-04',

'2000-01-05'],

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

Dataframe\_3['Item1']

Out[131]:

A B C D

2000-01-01 0.836830 -1.856472 0.345340 0.576771

2000-01-02 -1.529758 -1.646630 0.635996 -0.337408

2000-01-03 0.451765 0.156648 -1.225328 -0.177641

2000-01-04 1.123645 0.364549 0.684536 1.558884

2000-01-05 -0.082263 1.472391 -0.379373 2.410845

Dataframe\_3.major\_xs(Dataframe\_3.major\_axis[2])

Dataframe\_3.minor\_axis

Out[134]: Index(['A', 'B', 'C', 'D'], dtype='object')

Dataframe\_3.minor\_xs(Dataframe\_3.minor\_axis[3])

Out[137]:

Item1 Item2

2000-01-01 0.576771 1.387176

2000-01-02 -0.337408 -0.159394

2000-01-03 -0.177641 -1.622084

2000-01-04 1.558884 0.393800

2000-01-05 2.410845 -0.249931

转置：

Dataframe\_3=pd.Panel(np.random.randn(2,5,4),items=['Item1','Item2'],major\_axis=pd.date\_range('2000-01-01','2000-01-05'),minor\_axis=['A','B','C','D'])

Dataframe\_3

Out[5]:

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 5 (major\_axis) x 4 (minor\_axis)

Items axis: Item1 to Item2

Major\_axis axis: 2000-01-01 00:00:00 to 2000-01-05 00:00:00

Minor\_axis axis: A to D

Dataframe\_3['Item1','2000-01-03','B']

Out[9]: 2.3125077493899666

#转置前的元素定位查询

Dataframe\_3.transpose(0,2,1)

Out[10]:

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 4 (major\_axis) x 5 (minor\_axis)

Items axis: Item1 to Item2

Major\_axis axis: A to D

Minor\_axis axis: 2000-01-01 00:00:00 to 2000-01-05 00:00:00

Data\_transpose=Dataframe\_3.transpose(0,2,1)

Data\_transpose['Item1','B','2000-01-03']

Out[12]: 2.3125077493899666

#转置后的同一元素查询，两者查询结果相同，说明转置成功。

面板转化为分层索引数据框

Dataframe\_3.to\_frame()

Out[16]:

Item1 Item2

major minor

2000-01-01 A 1.345174 -1.719012

B -1.075240 1.258681

C 0.366470 -0.004046

D -0.149155 -0.405295

2000-01-02 A 0.005256 0.204166

B -0.032269 -0.667655

C 1.825649 1.050139

D -1.505179 0.534874

2000-01-03 A 0.876495 -0.414982

B 2.312508 -0.731893

C -0.129701 -1.470191

D -0.637856 -0.083188

2000-01-04 A -0.287276 0.633456

B -0.623472 -0.229308

C 0.530747 -0.896306

D 0.229674 0.288064

2000-01-05 A 0.163915 -1.836235

B 0.618230 1.353955

C 0.260995 0.808007

D 1.673127 -0.275785

切片：

Dataframe\_3['Item2',:,'B']

Out[18]:

2000-01-01 1.258681

2000-01-02 -0.667655

2000-01-03 -0.731893

2000-01-04 -0.229308

2000-01-05 1.353955

Freq: D, Name: B, dtype: float64

Dataframe\_3['Item2','2000-01-02':'2000-01-04','B']

Out[19]:

2000-01-02 -0.667655

2000-01-03 -0.731893

2000-01-04 -0.229308

Freq: D, Name: B, dtype: float64

Dataframe\_3['Item2','2000-01-02']

Out[20]:

A 0.204166

B -0.667655

C 1.050139

D 0.534874

Name: 2000-01-02 00:00:00, dtype: float64

Dataframe\_3['Item2','2000-01-02':'2000-01-04']

Out[21]:

A B C D

2000-01-02 0.204166 -0.667655 1.050139 0.534874

2000-01-03 -0.414982 -0.731893 -1.470191 -0.083188

2000-01-04 0.633456 -0.229308 -0.896306 0.288064

Dataframe\_3['Item1':'Item2','2000-01-02':'2000-01-04','C']

Out[22]:

Item1 Item2

2000-01-02 1.825649 1.050139

2000-01-03 -0.129701 -1.470191

2000-01-04 0.530747 -0.896306

Dataframe\_3[:,'2000-01-02':'2000-01-04','C':'D']

Out[28]:

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 3 (major\_axis) x 2 (minor\_axis)

Items axis: Item1 to Item2

Major\_axis axis: 2000-01-02 00:00:00 to 2000-01-04 00:00:00

Minor\_axis axis: C to D

#切出新面板。

随机数

随机数的生成屈服于一定分布，除了高斯分布，现在的python numpy.random模块还增加了一些其他分布。下面我做一下总结，方便大家记忆。

标准正态分布随机数σ=1,µ=0：

np.random.randn(4,4)

Out[13]:

array([[-0.10262253, -0.34324026, 1.76336179, 0.73546259],

[-1.29204273, -0.54047084, -0.49733134, -0.89416892],

[ 1.58563988, -0.14842537, -0.1117877 , 1.57606747],

[-0.46865011, 1.24499143, -0.06426564, 0.87932994]])

正态分布随机数：

np.random.normal(size=(4,4))

Out[15]:

array([[ 0.69885558, 1.69580874, 0.240318 , -0.0466421 ],

[-0.44963037, -0.1993139 , 0.746295 , -0.03377611],

[-0.1084134 , 1.65107746, 0.22775636, -0.42271574],

[-1.09199745, -0.15236048, 0.10510021, -1.32721006]])

np.random.normal(loc=[5.0,3.0,3.2,3,3.4,6.5,3.2,8.9,0],scale=[2.3,0.6,4.6,3.8,2,8,9,6,9],size=(9,9))

Out[44]:

array([[ 5.55280088, 4.5657248 , 12.57482226, -2.01739909,

2.89826871, 11.11609874, 3.84952928, 13.65527037,

-0.12038878],

[ 5.3845092 , 3.36821979, -2.22280282, 2.14065085,

3.35813688, 14.57232254, 13.2843872 , 15.37903668,

19.54550294],

[ 4.85078713, 4.03884281, 10.71342147, 7.15918121,

6.153268 , 8.98984918, 3.84036791, 6.47475206,

-11.35343032],

[ 6.27549427, 3.47531205, 8.30637283, 5.78373146,

3.05024564, 8.71573735, 5.03794128, 13.04055462,

-7.50368084],

[ 4.06839516, 3.55830323, 8.91660567, 1.48480807,

4.31611217, 28.54705664, -0.49793645, 6.2371168 ,

-1.19082079],

[ 6.31198136, 3.07881922, 1.92953421, 3.09186354,

7.16997125, 14.93156824, 3.56096463, 4.71242126,

-0.75186999],

[ 6.95627061, 2.94712057, 6.2722816 , 2.01850533,

1.57346614, 5.66339522, 8.65377144, 10.383041 ,

-6.69733312],

[ 4.90679121, 2.63961038, -1.56924057, -1.44700098,

4.36599423, 12.2371418 , -4.5516104 , 2.43685714,

-1.35353516],

[ 5.16451268, 2.63733899, 3.99069112, 6.56078357,

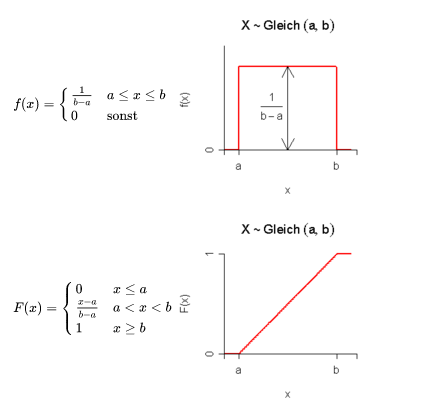
2.0441953 , -0.32367404, -10.89682055, 6.08930397,

1.96603181]])

均匀分布随机数

连续均匀分布：

连续型随机变量X的分布函数为：F(x)=(x-a)/(b-a)，a≤x≤b;0，x<a;1,x>b称随机变量X服从[a,b]上的均匀分布，记为X~U[a,b]。



若[x1,x2]是[a,b]的任一子区间，则

P{x1≤x≤x2}=(x2-x1)/(b-a)

均匀分布概率密度：f〔x〕= 1/(b-a)，a≦x≦b。f(x)=0，其他。

离散均匀分布：

1. 一般性离散均匀分布:

假设是一系列随机变量值且带有对于i<j。是由这些随机变量值组成的一个载体。那么离散型均匀分布的概率函数如下：



与此相对应的概率分布函数为：



1. 整数型离散均匀分布

假设a,b是两个整数，且，b-a=n-1，如果我们选择



作为载体，那么离散型均匀分布的概率函数如下：



与此相对应的概率分布函数为：



[t]是向下取整函数。

1. 自然数型离散均匀分布

假设两个自然数a=1，b=n，载体是：



那么离散型均匀分布的概率函数如下：



与此相对应的概率分布函数为：



[t]是向下取整函数。

np.random.rand(4,4)

Out[16]:

array([[ 0.4582242 , 0.49030239, 0.6255966 , 0.71307996],

[ 0.74651474, 0.08978701, 0.03992286, 0.85919246],

[ 0.18199355, 0.22821585, 0.94448593, 0.30911679],

[ 0.88791155, 0.97482975, 0.49228201, 0.6012444 ]])

均匀分布整数性随机数

np.random.randint(2,10,36)

Out[22]:

array([4, 2, 7, 9, 2, 8, 6, 4, 3, 5, 4, 8, 5, 3, 8, 2, 8, 6, 7, 6, 2, 6, 9,

9, 3, 8, 5, 6, 5, 7, 2, 2, 2, 6, 6, 9])

np.random.randint(5,12,(4,4))

Out[23]:

array([[ 6, 8, 8, 11],

[ 8, 10, 5, 7],

[ 9, 9, 9, 11],

[ 6, 10, 9, 11]])

产生二项分布随机数

np.random.binomial(1000,0.763,(4,4))

Out[25]:

array([[777, 758, 773, 755],

[760, 764, 771, 752],

[782, 758, 741, 758],

[763, 776, 757, 780]])

产生Beta分布随机数

np.random.beta(5,23,(4,4))

Out[26]:

array([[ 0.31271072, 0.23019834, 0.32032812, 0.20875184],

[ 0.14522475, 0.08526916, 0.22433647, 0.14442391],

[ 0.49284859, 0.17982631, 0.18972398, 0.34466781],

[ 0.06896375, 0.08824936, 0.09545425, 0.17308059]])

#输出呈beta分布在a=5,b=23时的16个数

np.random.chisquare(5,(4,4))

Out[28]:

array([[ 13.66373665, 1.25217053, 1.94050077, 5.66638455],

[ 0.70500103, 1.90989872, 2.52310923, 8.72662214],

[ 4.3429405 , 4.29082345, 2.21783624, 2.59295517],

[ 5.90457062, 3.5626025 , 1.14237117, 1.69882688]])

#输出呈卡方分布

伽马分布随机数

np.random.gamma(4,1.5,(4,4))

Out[29]:

array([[ 9.15959086, 8.71333152, 6.99486415, 2.84570767],

[ 12.56953127, 6.6805434 , 3.95665885, 13.90761613],

[ 6.87298242, 7.21044347, 13.48616811, 8.15694026],

[ 3.09350054, 3.65100471, 9.05640254, 12.19534025]])

(0,1)之间均匀分布随机数

np.random.uniform(0,1,(4,4))

Out[32]:

array([[ 0.23605074, 0.41409252, 0.0862554 , 0.3981574 ],

[ 0.6931572 , 0.14249284, 0.04572392, 0.86994581],

[ 0.1177653 , 0.25513802, 0.01181843, 0.59893541],

[ 0.96099555, 0.28696332, 0.63344723, 0.55314167]])

np.random.uniform(4,5,(4,4))

Out[15]:

array([[ 4.00587234, 4.23536906, 4.81200889, 4.63869131],

[ 4.40269477, 4.36290501, 4.09109586, 4.53140592],

[ 4.57763724, 4.5731731 , 4.53120769, 4.27633871],

[ 4.23523484, 4.47128557, 4.94877405, 4.05479443]])

np.random.seed(3)#确定随机数生成种子

List=[0,7,4,2,3,4,3,2,5,0]

np.random.permutation(List)

Out[36]: array([2, 3, 0, 2, 4, 3, 4, 5, 0, 7])

np.random.shuffle(List)

List

Out[39]: [3, 7, 0, 4, 4, 3, 5, 2, 0, 2]

对列表的随机排列！

例子：等概率随机数实现：

def uniform\_sampling(array\_test\_1dim,sample\_quantity,sampling\_number):

arrayreturn=[]

for i in range(sampling\_number):

#print (i)

index\_array=np.random.randint(0,len(array\_test\_1dim),sample\_quantity)

b=[]

for j in index\_array:

b.append(array\_test\_1dim[j])

#print (b)

arrayreturn.append(b)

return np.array(arrayreturn)

汇总和计算描述统计

概率入门与高斯分布：

我们可以这样定义一个随机变量，随机变量可以看做一个事件或者一种现象可以出现各种各种的状况，或者说可以处在各种各样的状态，并且无论他出现何种状态，这种状态的出现都是随机的。

随机变量有两种：

1离散型:

离散型（discrete）随机变量即在一定区间内变量取值为有限个或可数个。

1. 连续性：

连续型（continuous）随机变量即在一定区间内变量取值有无限个，或数值无法一一列举出来。更标准的定义是拥有概率密度或者连续分布函数。

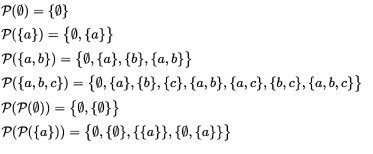
1. 幂集

一个集合X的幂集P（X）是一个新的集合，它由集合X的所有子集组成。更确切的说P（x）是由X派生的。其数学表达方式如下：



注意：空集和X本身也属于P（X），幂集的元素个数等于2n，n是集合X的元素数量。

例子：



1. σ—代数



假设Ω是一个非空集合并且P（Ω）是由Ω派生一个幂集。 是幂集合的一个子集，我们称这个集合为σ—代数，如果他满足以下条件：

1. 
2. 如果 那么 



（3）如果，那么

3.测度



是一个由Ω派生的（或者说与Ω有关的）σ-代数，函数





叫做 上的测度，如果它满足以下条件：



并且：



其中是来自于 的相互相离不相交的子集。



4.测度空间







叫测度空间，如果Ω是非空集合， 是一个σ-代数，µ是一个在 上测度。

1. 密度函数

是一个测度空间，µ和ν是测度空间的两个测度，并且





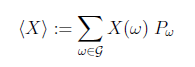
其中，fν是一个ν的关于µ的正的可积密度函数，A是一个任意属于 的集合。

应用的概率论中，他有以下形式。



P(A)是A上概率，fp是概率P的密度函数。也叫概率密度函数。λ（x）是雷贝克测度，fp是概率P的密度函数。也叫概率密度函数。也就是说上面是是一个雷贝克积分，实际工作中，我们大部分计算要依靠黎曼积分。因此上式可把变成dx。

1. 期望值，在前面章节我们提到过期望值。对于离散的随机变量，它的期望值是：



随机变量是事件的函数，也可以说是事件的实现。一个随机变量在离散情况下能够实现可数个事件，从而产生可数个随机结果。比如我们扔色子（这是个事件），可能有六个结果，也就是与此对应的随机变量可以产生六个结果。它们分别是1，2，3，4，5，6。代入上面等式得：



上面这个例子随机变量每次的实现概率为1/6。 因为色子的每一面出现的概率是均等的，且他们的和又要等于1。

上面谈的概率是理论值，实际操作中往往难以实现。根据大数定理，只要我们足够多的重复，每个面出现的概率都会无限趋进于理论值。

算数平均值是一个十分随机的结果，比如扔色子，分两轮扔，每轮扔十次，比较两轮平均值，会发现差别。这一特点在期望值那里是没有的，只要随机变量不变，事件不变，那么期望值就恒定不变。人们在实践中发现，算数平均值随着实验次数的增加而趋紧于理论值。但却不一定会达到理论值。这正是期望值这个名字的由来。

估计量的期望值有些时候能够等于一个参数的真值，这时候我们说我们的估计为无偏估计，否则为有偏估计。

**例如我们计算一个观测数据集的方差，往往会用一些经验公式，这个经验公式就是估计，得到的值是估计量。然后我们对这个经验公式求期望值，得到的结果就是这个估计量的期望值。如果这个期望值等同于这个方差的真值，那么我么就说是无偏差估计，否则就是有偏估计！真值是一个变量本身所具有的真实值，它是一个理想的概念，一般是无法得到的。所以在计算误差时，一般用约定真值或相对真值来代替。对于统计学我个人理解是理论值，由公式计算出的值，如果公式能计算出值的话。**

在实际应用中，我们经常用算数平均值来代替期望值。前提是数据样本数量要足够多。

对于连续的随机变量，假设他有实数到实数的概率密度函数，我们通常用下列等式来描述：



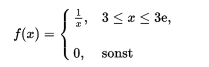
他的期望值。其中λ(x)是lebesgue测度，也就是说这个期望值是关于Lebesgue测度的Lebesgue积分。Lebesgue积分是一种更广泛的积分。通常情况下，我们应用更具体和实用的黎曼积分进行期望值计算，也就是通过下列等式计算期望值。



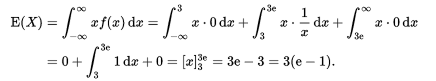
作为等价的等式，我们也可以直接用概率分布函数（注意与概率密度函数不同）来计算期望值：



典型例子：



是一个随机变量X的实数概率密度函数，那么这个随机变量的期望值是：



概率函数与概率密度函数在实数范围内利用黎曼积分可以如下表达：

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

我们可以把F（x）看作在（-∞，x）上f（t）的**一个原**函数。

典型例子：

指数分布的概率密度函数：



他的概率分布函数是：



This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

期望值的一般化定义：



其中X是一个从概率空间映射到的关于P测度可积分的随机变量，其中是布尔σ-代数，。

经典例子：

给定一个概率空间，其中 ∑ 是Ω的幂集合且是一个σ-代数，P是一个测度，且带有下列值：

i=1，2，3

X是一个Ω上的随机变量且带有下列值：





那么X的期望值可以如下计算：



两个随机变量，一个概率密度函数的期望值：

X和Y是两个随机变量，它们有共同的概率分布密度函数f（x，y），则一个函数g（x，y）的期望值可以按照下列等式计算：



This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.



其中

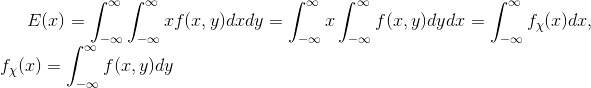


This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

特别是，当g（x，y）=x是只关于一个随机变量X的函数时：



人们通常这样变化这个等式：



期望值的线性特征：

(1)



X1 和X2是两个随机变量。可相关也可以不相关。不相关很容易理解。相关时为什么也满足线性呢？（课堂作业题）

(2)

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

其中d是常数。

(3)



其中This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.是n个随机变量。

1. 三角公式：





其中X，Y是两个随机变量。

1. 如果 几乎必然，（如果This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.，我们说几乎必然，如果This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.，我们说几乎不可能），且E（X）和E（Y）存在，那么我们把这种特性称作单调性。
2. 随机相关和随机不相关

有两个随机变量，X，Y，我们称它们随机不相关，如果X变量的取值不受Y变量影响的话。反之，则称为随机相关。

1. 随机不相关



其中Xi 和Xj 是随机不相关的两个变量。也就是：i ≠j

如果有更多不相关的随机变量相乘，则下列等式有效：



1. 随机相关（不做深度介绍）



我们讲了随机变量的期望值，下面简单介绍下随机变量函数的期望值：



如果收敛，那么一定存在。

其中是随机变量X的函数，是随机变量X的概率分布密度函数。

对于离散的随机变量我们同样可以定义其函数期望值：



关于期望值还有很多话要说，但是时间有限，我们的讲义就到此为止。如果同学们觉得还需要加深，我们可以继续加深。

方差

我们已经讨论过离散型分布的方差，在那里我们把随机变量的每个变量值xi实现概率看做1/n,把（X-<X>）2的实现概率近似的看做1/N和1/(N-1)。从而得出总体方差和样本方差，即：

和 

这些公式仅仅是经验公式，或者说仅仅是一种估计，所以很多时候我们称这些公式为估计量。方差的精确描述还是要依据理论公式。

方差的理论公式分为两种类型：

随机变量离散分布的方差：

一个随机变量X，带有一个有限的或者无限可数的的值集合

，

那么，我们称这个随机变量为离散分布随机变量。





这里，是随机变量X获取xi（或者说X有实现xi时的概率）变量值时的概率。µ是X的期望值，离散的情况下也可称作平均值，它的计算如下：



连续性分布随机变量的方差：

一个随机变量X被称为连续性分布随机变量，如果有一个这样的概率密度函数



以致使概率分布函数



对于任意实数t都能用下列等式来表达：

。

从上面的定义我们可以看出，对于连续分布的随机变量人们只有通过概率密度函数才能对一段区间上的随机变量概率进行计算。

一个连续分布随机变量，或称作连续随机变量的方差可按下面的等式进行计算：





方差的线性变化：

给定两个常数，那么它们满足下列陈诉：

1. 常数的方差为0：
2. 平移不变性：随机变量的平移一个常数不会影响数据分散度。
3. 随机变量乘以一个常数会造成方差以常数平方倍的程度增大或者减小。



把B.和C.综合起来得到：



特别是a=-1时：



这就意味着：随机变量前符号的变化不影响方差。

方差的和与乘积：

随机变量的任意和的方差按下式计算：



从这个公式中我们得出以下公式：





如果这些随机变量是相互独立的随机变量（非关联条件已经足够，因为非相关一定可以到处非关联，反之不可），我们可以继续得到：





如果这些非关联的随机变量有共同方差，那么下列等式有效：

、

这个公式说明样本量增大，样本平均值方差下降。

如果两个随机变量是相互独立的，那么它们乘的方差满足下列等式：



方差的实际意义：

1. 距离的解释

如果我们定义随机变量X是一个点，随机变量期望值是另外一个点，那么可以看做是两点距离平方的期望值（或者是平均值）。

注意：这里X是一个随机变量，他可以取无穷可数或者有穷的随机变量值，我们这里所说的距离实际上是每个随机变量值与期望值之间的距离。

1. 作为确定性的尺度

离散分布：

方差可以描述概率函数的宽度，就像随机性或确定性一样，是一个可以观察到的统计现象。对于大的方差，偏向随机状况的描述；对于小的方差，更偏向确定性的描述。在方差等于0的特殊情况下出现完全确定的状况。这就是说，如果方差等于0，那就意味着随机变量X以100%概率获取随机变量X的期望值E（X），反之，亦可。（）。这样的随机变量实际上是一个常量，只有常量才具有完全确定的概念。另外意味着对于所有的随机变量值。这样的一个随机变量分布统计学称作反常分布。

概率函数只会出现在离散分布随机变量，其定义如下：

对于离散的随机变量X，它的概率函数f（x）是：



连续分布：

与离散的情况不同，连续的随机变量的方差等于0的话就意味着，对于任意随机变量值，概率密度函数均为0。在连续情况下，方差描述的是概率密度函数的宽度，这个宽度代表的是不确定性的一个尺度，这个不确定与随机变量紧密相关。确切地说，宽度越小，随机变量X越能准确预测他即将获取的值。

**关于方差，还有一些重要内容以后再补充**

动差生成函数

动差又被称为矩，随机变量X的动差生成函数或叫做矩母函数可以通过下面的等式进行定义：

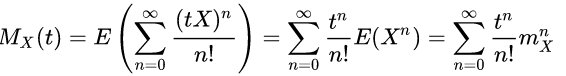


其中t可以是实数，也可以是复数，当然这一切前提是期望值存在，期望值不存在，也就无从谈起矩母函数。

假设t=0，那么这个矩母函数为常量1. 很多情况下，我们往往把动差生成函数的变量围绕着t=0发展成泰勒级数，即：



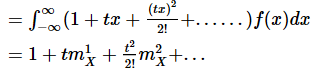




其中，是变量X的n阶矩。矩母函数依赖于随机变量X的分布。如果一个分布的随机变量X的矩母函数在t=0的邻域存在，那我们就说这个分布有矩母函数，这种说法虽然有点不十分精确，但却十分常见。如果一个分布的矩母函数只在t=0的情况下存在，那么人们就可以说，这个分布没有矩母函数。

我们继续假定这个随机变量X是连续随机变量，那么的期望值，也就是矩母函数是：

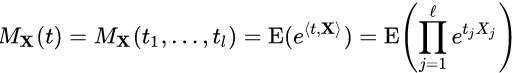




其中是随机变量X的k阶矩。

多维矩母函数：

现在我们从实数随机变量转换到矢量随机变量上来：



累积量：

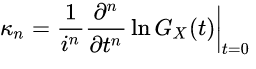
给定随机变量X的矩母函数，那么函数：



叫做累计生成函数。分布X（随机变量X的分布）的n阶累积量可以通过下列等式来定义：



常用的是另一种表达式：



其中被称作特征函数。

累计生成函数的一些特性：

平移恒定性：

假设X是一个随机变量，那么对于任意实常数c有：

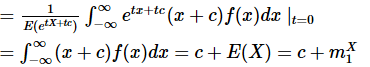


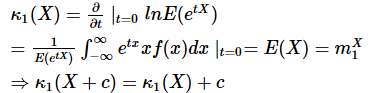


（n ≥ 2）

我们这里只关注X是连续随机量的情况，也就是说：







后面我们可以用数学归纳法进行证明：



（n ≥ 2）

齐次性：



这个等式的证明也可以通过数学归纳法进行证明！

可加性：

假设X1和X2是两个随机独立变量且Y = X1 + X2，则下列等式成立：



数学归纳法可以证明，我们这里不再进行证明。

对于随机独立变量X1和X2，它们的和的特征函数满足下列关系：



利用定义直接可以证明（课堂作业）。

对上式两边直接求对数：

{displaystyle ln G_{Y}(t)=ln G_{X_{1}}(t)+ln G_{X_{2}}(t)=sum _{n=1}^{infty }{frac {(mathrm {i} t)^{n}}{n!}}left[kappa _{n}(X_{1})+kappa _{n}(X_{2})right]=sum _{n=1}^{infty }{frac {(mathrm {i} t)^{n}}{n!}}kappa _{n}(Y)}

上面等式可以通过泰勒级数进行证明，留作课堂作业！

假设是N个独立的相互不相关的随机量，它们的合满足下列等式：



直接通过就可得出上式。

累积量与矩函数：

什么是矩函数？要想了解矩函数，我们必须再次把目光落到矩母函数，在矩母函数这一章，我们已经已经定义了随机量的矩：



其实它也可以直接通过G(t)函数来表示一个随机变量（通常也可叫做随机量）的n-截距。





通过上面截距表达式，我们可以得到：













以此继续可以得到：



观察上面等式等式最终我们得到以下递推公式：



除了矩函数还有中心矩函数，一个随机量的中心矩函数通过如下等式来定义：



其中。

下面是一些常用中心矩函数：





**方差**



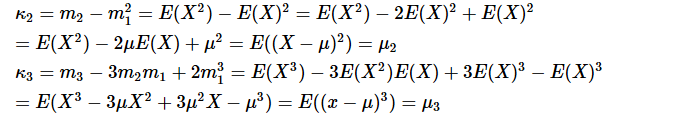
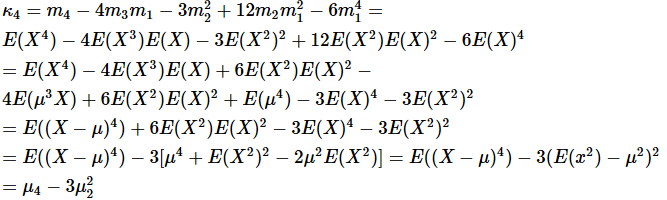
**标准化后是拱度**

**标准化后是偏度**



累积量和中心矩函数的关系：





LnG(t)的泰勒级数：

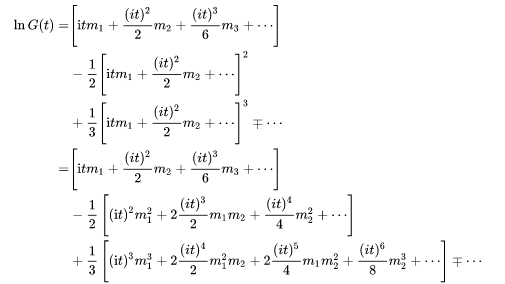
下面我们围绕着G(t) = 1发展泰勒级数：



同样我们也可以把G（t）围绕着t=0发展为泰勒级数：



把G(t)代入到lnG(t)得到：



按照t的升序排列上式得到：



矩作为累积量的函数：

参考“累积量与矩函数”这一章节，我们很容易得到：



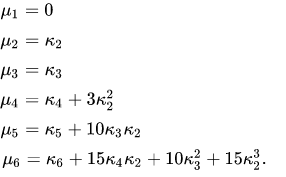








类似的方法我们还可以得到累积量作为中心矩的函数：

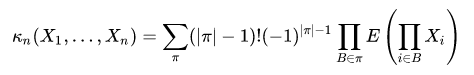


多元累积量：

更多随机变量或者随机变量矢量的多元累积量可以通过下式来定义：



它的累积量可以组合形来定义：



其中π是集和{1,...,n}全部分割组成的集和中的一个元素，也就是说π是一个任意分割；B是π中元素，也就是分割的一个任意分割块。是一个分割π中的分割块的数量。

关于集合的分割：

在集合理论里一个集合M的分割又是一个集合P，在这个集合P里，所有元素都是M的非空子集，以致集合M的任意元素都属于集合P中的某一元素。

换句话说：一个集合的分割是把该集合分解成相互不相交的非空子集。

空集的唯一分割还是空集。

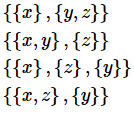
每个一元素集合仅有一个分割

每个非空集合M仅有一个单元素分割人们称这样的分割为普通分割。

例：集合是集合的一个分割。

例：集合{x,y,z}有5个分割





多元累积量的一个完美例子：



我们也可以通过累积量来表示矩函数（左边是矩函数，X1\*...\*Xn相乘关系）：



多元矩函数几个例子：







如果这些随机变量是独立非相关随机变量，那么他们的累积量等于0（作业！），假设部分随机变量独立非相关（至少两个随机量）那么与此对应的混合累积量等于0.

多元累积量的复合线性：



**中心极限定理：**

给定一个随机量Y，



其中X1,...,Xn是独立等分布随机变量。

根据累积量的齐次性和可加性得：



因为的阶是N，常系数的阶是。意味着阶为的式子。略等于意味着Y随机量的n-阶累积量近似于或者主要是或者取决于一个阶为式子。

通过上面等式分别求出1-阶到4-阶的累积量：





我们发现从3-阶累积量开始，等式右边式子的阶开始为负数，这就意味着，在N→∞，3-阶的累积量将会消失！



对于任意n≥3

这就是说对于这个特殊的随机量Y（Y是乘以一个系数的独立随机量组合），当组成它的独立随机变量的数量趋近于无穷时，它的累积量除了1阶和2阶，其它都自动为0. 高斯分布是具有这种特性的唯一分布。**也就是说具有Y结构的随机变量在组成它的独立非相关随机变量的数量趋近无穷时趋近于高斯分布（密度函数）。**这句话就是所谓的**中心极限定理**。

注：

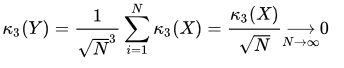
1. 上面的叙述尽管并不是一个严格的证明，然而几乎把这个定理证明所需思路讲得十分清楚，唯一没有说明的是为什么高斯分布‘是具有这种特性的唯一分布’。关于这个定理的证明大家可以在任何一本教科书上找到，有兴趣的同学可以查阅相关资料。
2. 实际应用中如果有很多独立非相关随机量对测试造成影响，我们可以通过一个高斯分布随机量来描述这些独立随机变量的总影响。

例子：

假设随机变量是同一个随机变量，也就是说。它的方差为，平均值为0并带有任意高阶的矩（保证任意高阶累计量的存在）







例子：

给定一个如下定义的随机变量Z：





Z的期望值等于0，即使Xi的期望值不等于0.

根据平移不变性继续可以得到：

同一分布下的中心极限定理的变种：

假设是一个随机变量序列，这些随机变量在同一个概率测度为P的概率空间内呈现出相同的概率分布，并且是独立不相关的（缩写i.i.d.=Independent and identically distributed random variables独立等分布随机变量）。继续假定，不仅期望值而且标准方差存在并且是有限的。

我们现在只关注这个序列的前n项和：



它的期望值和方差分别是：





如果我们通过上面等式的帮助对Sn进行标准化处理：



那么的分布函数在n → ∞ 时逐点收敛于标正态分布N（0,1）的分布函数。

这就是同一分布下的中心极限定理。更为确切的数学表达为：

（￥￥）

对于任意。

**注：借助随机变量定义分布函数：**

假设X是实数随机变量，那么人们称函数：



是X的分布函数。其中说明随机变量X取小于或等于x值的概率。

我们还可以把（￥￥）等式进行如下变形：


这个公式才是我们经常用到的公式！

**大数定理**

对于由独立等分布随机变量组成随机变量



根据累积量的齐次性和可加性得：



（\*\*）

右边阶的估算来自于分子分母的比值；因为分子是N个单一随机变量累积量的和，它有阶。因此整个式子的阶为。

和在中心极限定理一样，这里的意味着阶为的的式子。略等于意味着Y随机量的n-阶累积量近似于或者主要是或者取决于一个阶为的式子。

通过（\*\*）我们得到下面不同阶的累积量：



对于n ≥ 2累积量且随机变量的数量N趋近于无穷时：



这就是说当N趋近于无穷时，只有1阶累积量不为0。

下面我们讨论随着N的增大，随机变量Y的分布变化。随着N的增大，或者说当N=N0足够大时，除了一阶和二阶，其他阶累积量都已经趋近于0时，我们说此时Y的分布已经是高斯分布。其期望值也就是说其平均值为：





下面一个章节我们可以看到，高斯分布概率密度函数是一个围绕着对称轴：



对称分布的函数。其方差是：（同分布，方差唯一）



随着N的继续增大（直到无穷），方差会逐渐减下，也就是说高斯密度函数的宽度会逐渐减小，直到在处形成一个peak（deltaform-function）。

**明天继续**

Gauß分布

一个连续的随机变量X，如果它的概率密度可以通过下面的等式来定义。



那么我们就可以说这个随机变量是按高斯分布的，或者说这个随机变量的概率分布是高斯分布。

随机变量X实际上是一个函数，而x是随机变量X的一个函数值。这个值是把一个事件ω实现后的结果，也即下面等式成立：



通俗点讲，随机变量X要想拥有x函数值的概率密度是f（x），也就是说X处于x状态时的概率密度是f（x）.

下面我们通过一个物理的小例子来解释下究竟什么是概率密度。

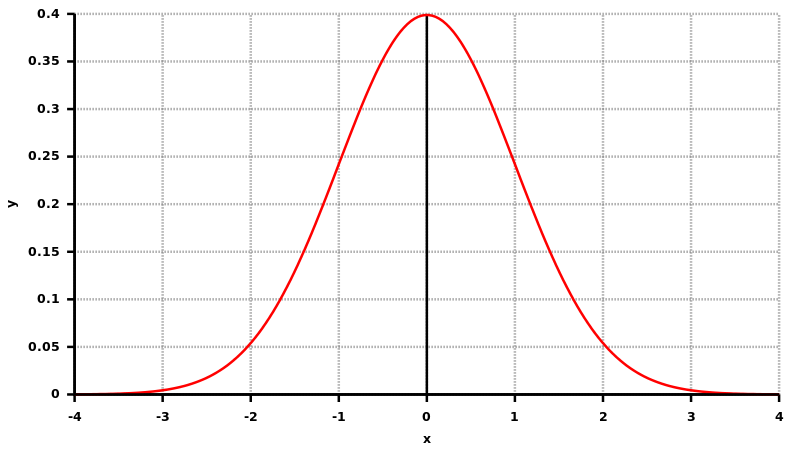
这里我们把概率密度理解为一个电子（带电粒子）在单位时间内出现在单位体积内的概率，也就是上图所示一秒内带电粒子在那个小立方体中出现的概率。

高斯分布也叫分布，或者正态分布，这说明它与µ和σ两个参数相关。µ和σ分别是期望值和方差。不同的期望值和方差可以得到不同的高斯曲线。

例如：当µ = 0和σ2=1时，我们有：



我们称这种高斯分布为标准正态分布。下面是它的分布函数



正态分布概率分布函数可以通过下面的等式进行定义：



令，那么我们可以得到。

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

把上面两个结果代入到上面的积分等式中得：

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

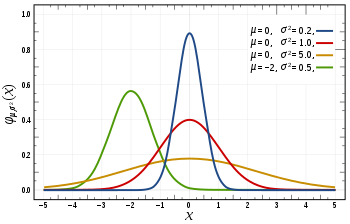
上面的等式可以直接写成：

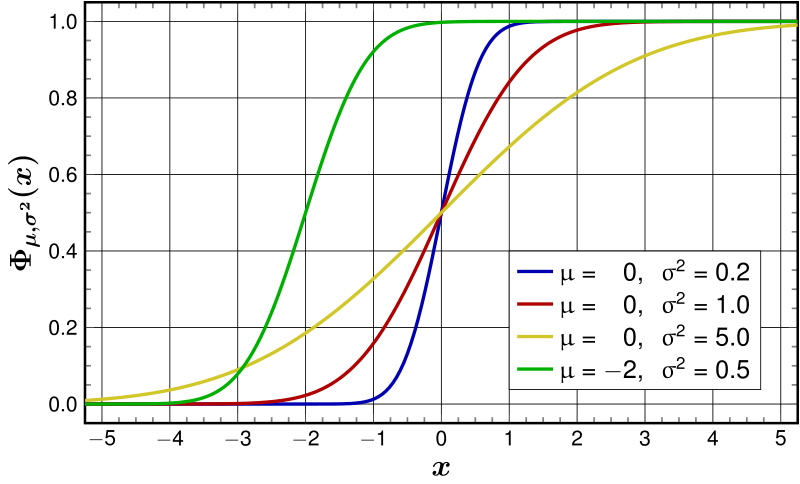
This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

这个等式正好是标准正态分布的概率分布函数。

高斯分布的概率密度函数和概率分布函数的对称性：

概率密度函数的外形是一种高斯钟形曲线，它的高度和宽度与σ有关（σ越大曲线越宽越矮）。直线是它的对称轴，也就是说随机变量的概率密度是围绕着期望值对称分布的。高斯分布的概率函数是关于点的点对称的分布函数。对于概率密度函数满足，概率分布函数满足对于任意属于实数的x。





密度函数的拐点和最大值

通过密度函数的一阶导数和二阶导数人们能够确定密度函数的最大值和拐点。

密度函数的一阶导数为：



当它的一阶导数等于0时，也就是当x=µ，我们可以得到他的最大值。

密度函数的一阶导数为：



当它的二阶导数等于0时，也就是当x=µ±σ，我们可以得到他的拐点值

密度函数的正则化：



这就说明随机变量在整个实数轴出现的概率是100%。

高斯分布的期望值：

标准正态分布期望值的计算：



经典例子：

假设Y是一个正态分布的随机变量，并与标正态分布的随机变量变量X存在下列关系：

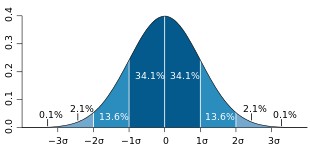


则通过上面的式子我们可以求出：



高斯分布的分布区间（标正态分布为例）十分重要！！！！

我们观察下图，可以得出以下结论。



68.3%的随机变量值或者称作随机变量的实现将会出现在区间µ±σ。

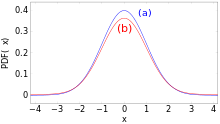
95.4%的随机变量值或者称作随机变量的实现将会出现在区间µ±2σ。

99.7%的随机变量值或者称作随机变量的实现将会出现在区间µ±3σ。

因为在实践中很多随机变量值都是近似以正态分布的，因此正态分布的很多值经常作为经验值而被利用。例如，σ经常作为一个区间的一半宽度，这个区间包括了2/3的随机变量值（样本值）

但是这种做法并不可取，因为很多时候由标准样板方差形成的µ±σ区间所包含的样本数据量要远超2/3。例如，一个样本的样本值近似符合正态分布，他可以通过一个复合型正态分布来表示：





从图象上看（a）正态分布曲线和（b)复合正态分布曲线几乎是差不多，按道理说复合正态分布的σ值应该也和正态分布曲线的大小相差不大，但实际情况并非如此。复合正态分布的曲线拥有更大的σ值，以至于92.5%的值分布在µ±σ内。

分布于2倍（µ±2σ）或者甚至3倍（µ±3σ）标准方差区间外的样本值我们通常认为是偏差值。它反映出数据采集出现比较严重的错误。但是有时候他也反映出数据分布存在偏度。

正态分布的区间概率计算：

假设X是一个随机变量，且符合正态分布，则在分布区间的概率是：



其中是标正态分布的分布函数。

反之，对于给定的概率，通过下列等式：



人们可以确定与此对应的分布区间。



这个性质十分有用，下面我们证明其正确性：

由于µ的数值只是确定密度函数对称轴所处位置，因此不影响积分结果，我们不妨使µ =0，也就是我们把函数整体移到以µ = 0为对称轴的位置。这样只需关注在

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

的积分。

Step1:计算[-∞，-zσ]上的积分；

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

Step2:计算[zσ，∞]上的积分；

窗体底端

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

窗体底端

Step3:接下来我们观察标正态分布函数在[-∞，-z]上的积分；

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

我们发现这三步计算的结果是相同的，我们不妨假定：

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

那么我们可以轻易得出和I的关系：

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

下面我们计算非标正态密度函数在This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.上的积分：

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

I

I

zσ

-zσ

依据上图我们可以立即得出：

This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.

窗体底端

高斯分布的变异系数：



其中σ为标准方差，µ 为期望值。

高斯分布的拱度

高斯分布的拱度与µ 和σ不相关且等于3（标准正态分布的拱度），为了能够更好的估算其它分布的拱度，我们经常拿正态分布的拱度与它们进行比较。

高斯分布的偏度

高斯分布的偏度与µ 和σ不相关且等于0

高斯分布还有很多性质我这里，由于时间有限，我们这里暂且不做介绍。

峰度和偏度

峰度：

拱度是用来描述概率分布函数，或者统计密度函数或者频率函数（几乎等同于概率分布函数）图形尖度或者说陡度的刻度量。

拱度是是四截矩的，拱度越小，函数分布就越均匀。拱度越大，分布就越陡峭，分布中就含有更多的随机变量极端值。但其分布的范围却相对较狭窄。（与高斯分布作比较）

峰度是用来表示被观察函数拱度与正态随机变量的概率密度函数拱度之间的差。

一个随机变量概率分布函数（或者叫频率分布函数）的拱度可以利用下面的经验公式进行计算。



其中This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.是观测值，s是标准方差，This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.是随机变量值的算术平均值，

是观测值的标准化。它有下面一些特性：



拱度是非负函数，如果它的值小于3，则其频率分布较为平缓，反之，则较为陡峭。（与正态分布相比）

一个随机变量X的拱度（理论计算）



其中µ是随机变量的期望值。

拱度与峰度关系：

为了更好的评估一个分布函数的拱度，我们往往把他与正态分布函数的拱度作比较，由此我们引入了峰度概念：



其中是峰度，是被比较分布的拱度，3是高斯分布的拱度。

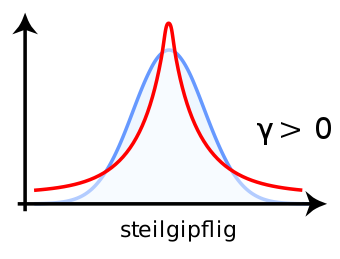
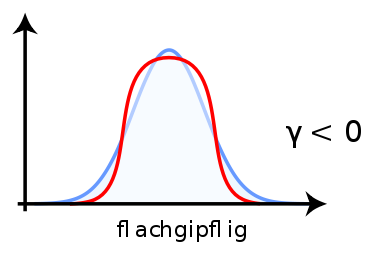
注意：千万注意，拱度别和峰度搞混！！！！！！

峰度的含义：

峰度等于0，说明被比较分布的拱度与高斯分布的拱度相同。

峰度大于0，说明被比较分布的拱度大于高斯分布的拱度。也就是被比较分布的顶端拱形更尖锐。

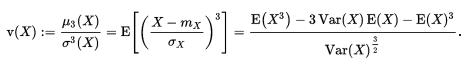
峰度小于0，说明被比较分布的拱度小于高斯分布的拱度。也就是被比较分布的顶端拱形更平滑。



偏度：

与峰度类似，偏度也是一个统计学常数，它描述了概率分布函数的非对称性和强度。具体来讲，它说明了与对称分布相比一个分布函数是否或着以多大程度向右或者向左偏离。

一个随机变量的偏度是三截矩的，他的理论计算式如下：



其中IMG_256.

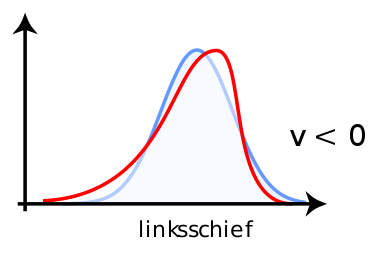
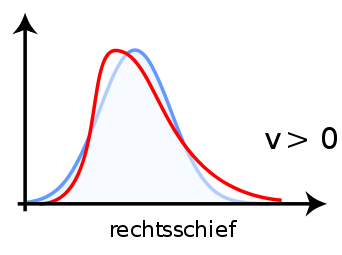
上面的等式让学生证明。

偏度的含义：

偏度的值可以为正，可以为负或者甚至是无法定义。偏度为负（负偏态）就意味着在概率密度函数左侧的尾部比右侧的长，绝大多数的值（包括中位数在内）位于平均值的右侧。（经验法则）

偏度为正（正偏态）就意味着在概率密度函数右侧的尾部比左侧的长，绝大多数的值（但不一定包括中位数）位于平均值的左侧。（经验法则）

偏度为零就表示数值相对均匀地分布在平均值的两侧，但不一定意味着其为对称分布。（经验法则）



偏度的性质：

1. 偏度在线性变化下是不变的



其中a>0。

B,对于n个相互独立的，标准化的随机变量，他们的合随机变量满足下列等式：



注：随机变量标准化：

一个拥有µ = 0和σ=1的随机变量X叫做标准化随机变量。

偏度的经验计算公式：

实际应用中我们用下列公式计算偏度：



其中This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.是观测值，s是标准方差，This is the rendered form of the equation. You can not edit this directly. Right click will give you the option to save the image, and in most browsers you can drag the image onto your desktop or another program.是随机变量值的算术平均值，

如在峰度那做的那样；我们下面把随机变量X标准化。



我们现在验证其的确已被标准化：





最终我们得到下面的公式：

IMG_256

然而这种估计公式（经验公式）并不是无偏估计，它偏离偏度的理论期望值。因此，实际工作，我们经常用下面的公式计算其偏度：



协方差：

假设X和Y是两个实数的可积分的随机变量,它们的积同样也是可积的，这也就是说假设期望值E（X），E（Y）和E（XY）都是存在的，那么我们称



为X和Y的协方差。

对于离散的和连续的随机变量我们分别有下列表达式：



协方差的含义：

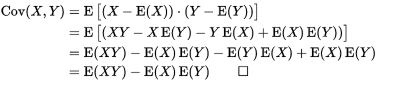
1. 如果协方差是正值，那么X和Y之间存在一个单调关联。这就是说X的值随着Y值的增加而增加，随着Y值的减下而减下。
2. 如果协方差是负值，那么X和Y之间存在一个反向的单调关联。这就是说X的值随着Y值的增加而减小，随着Y值的减下而增大。
3. 如果协方差等于0，那么X和Y之间不存在单调关联（非单调关联倒是可能）

协方差虽然说明了两个随机变量间的关联方向，但是并没有谈到这种关联到何种程度。为了使这种关联量化，也就是使其具有可比性，我们必须把协方差标准化。常用的借助于标准方差的标准化导致了相关系数的产生。

协方差的偏移定理：



因为：



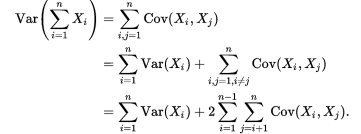
方差和协方差关系：





随机变量合的方差：

利用协方差的双线性性质和对称性得：



双线性和对称性后面有证明。

有限求和的特殊情况：



由双线性或者直接由定义可得：



随机变量差的协方差：



→

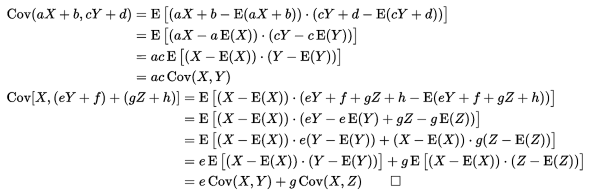
协方差的双线性，半正定性，对称性：

1. 双线性





证明：



1. 对称性





1. 半正定性



因为：



1. 柯西不等式



因为协方差并不能消除随机变量单位对结果的影响，因此获得一个无量纲且能反映协方差意义和程度的系数是不可避免。对此，人们定义了关联系数，其表达式如下：



非相关或者非关联：

非相关定义：

是一个概率空间，和（母集和delta代数）两个测度空间，两个随机变量



和



叫不相关，如果任意，满足下列等式：



简化上面等式，可得：



意义：P测度作用到含有两个随机变量的表达式时，他的计算结果等于分别作用他们各自测度空间结果的乘积：

上面等式简化成函数就是：



非关联：

如果并且最终导致那么我们称X和Y不关联。

注意：不相关一定不关联，反之不行。证明十分简单。



反之不行，举例证明：

两个在[-1,1]上均匀分布且相关的随机变量X2和X，然后他们确是不关联的。



因为：





**后面还有好多精彩反例，有机会加上去！！！**

关联系数

上一章我们在讲协方差时定义了关联系数，接下来我们着重讨论一下关联系数关联系数的理论表达式如下：

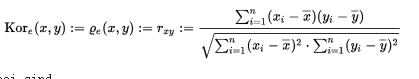


我们可以用期望值改写上面的定义等式：



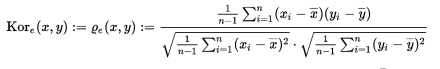
如果，我们说这两个随机变量是非关联的。

它的经验公式如下：

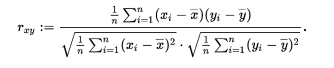


其中是n对测量值对，和是平均值。

这个公式是有偏估计，下面我给出皮尔逊-关联系数：

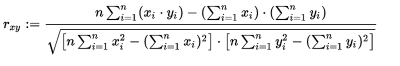


这个公式还可以有下面的变种：



以上两个公式的计算结果完全相同，因为1/n 和1/(n-1)在等式中不参与运算。

我们还可以把上式继续化简：



特性与含义：

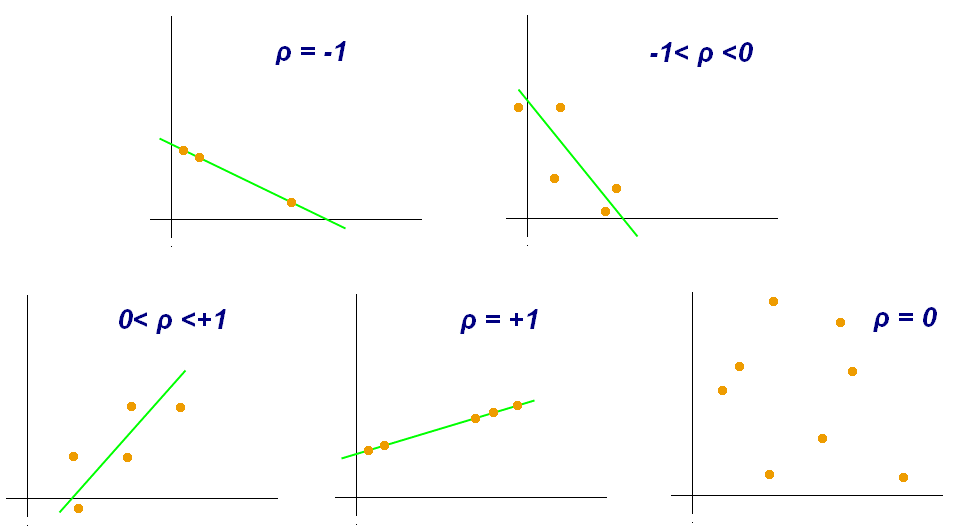
对称性：

归一性：

取值范围：（注意,这个性质的得出来自柯西不等式）

和几乎等价。

两个随机变量不相关，那么他们的关联系数为0.



**看图，让学生自己总结出规律**

(1)、当相关系数为0时，X和Y两变量无关系。

(2)、当X的值增大（减小），Y值增大（减小），两个变量为正相关，相关系数在0.00与1.00之间。

(3)、当X的值增大（减小），Y值减小（增大），两个变量为负相关，相关系数在-1.00与0.00之间。

相关系数的绝对值越大，相关性越强，相关系数越接近于1或-1，相关度越强，相关系数越接近于0，相关度越弱。

通常情况下通过以下取值范围判断变量的相关强度：  
相关系数     0.8-1.0     极强相关  
                 0.6-0.8     强相关  
                 0.4-0.6     中等程度相关  
                 0.2-0.4     弱相关  
                 0.0-0.2     极弱相关或无相关

皮尔逊关联系数适用范围：

当两个变量的标准差都不为零时，相关系数才有定义，皮尔逊相关系数适用于：

(1)、两个变量之间是线性关系，都是连续数据。

(2)、两个变量的总体是正态分布，或接近正态的单峰分布。

(3)、两个变量的观测值是成对的，每对观测值之间相互独立。

(4)、相关系数必须是显著地，就是说它不能在‘0’点附近。

**其它精彩内容后续会有补充**

分位数:

分位数是将总体的全部数据按大小顺序排列后，处于各等分位置的变量值。通常我们把数据四等分，即把总体数据分成四个相等部分，每个部分占总数据数目的25%。处在各分位点的数值就是四分位数，四分位数有三个，第一个叫下四分位数，第二个四分位数就是中位数，第三个四分位数称为上四分位数，分别用Q1、Q2、Q3表示。第三四分位数与第一四分位数的差距又称四分位距。

三个四分位数的确定：

方法1：

Q1的位置= (n+1) × 0.25

Q2的位置= (n+1) × 0.5

Q3的位置= (n+1) × 0.75

n表示数据的数据个数。

上面的是大家常用的n+1法。还有一种是n-1法

方法2：

Q1的位置=1+（n-1）x 0.25

Q2的位置=1+（n-1）x 0.5

Q3的位置=1+（n-1）x 0.75

Python 用的是n-1法

N+1法的例1：

数据：6, 47, 49, 15, 42, 41, 7, 39, 43, 40, 36

由小到大排列的结果: 6, 7, 15, 36, 39, 40, 41, 42, 43, 47, 49

一共11项

Q1 的位置=（11+1） × 0.25=3， Q2 的位置=（11+1）× 0.5=6， Q3的位置=（11+1） × 0.75=9

Q1 = 15，Q2 = 40，Q3 = 43

N+1法例子二

数据：7, 15, 36, 39, 40, 41

N=6，

Q1位置=（6+1）\*0.25=1.75, Q2位置=7\*0.5=3.5，Q3位置=7\*0.75=5.25

Q1位于第一个数和第二个数之间，因为小数部分大于0.5，所以可以按下列等式计算：

较大数\*小数部分+较小数\*（1-小数部分）=较小数+（较大数-较小数）\*小数部分=15\*0.75+7\*0.25=13.

Q2位于第三个树和第四个数之间，因为小数部分等于0.5，所以可以按下列等式计算：

（较大数+较小数）/2=37.5

Q3位于第五个数和第六个数之间，因为小数部分小于0.5，所以可以按下列等式计算：

较大数\*(小数部分)+较小数\*（1-小数部分）=较小数+（较大数-较小数）\*小数部分=0.25\*41+0.75\*40 = 40.25.

N-1法的例子：

数据总量: 6, 47, 49, 15, 42, 41, 7, 39, 43, 40, 36

由小到大排列的结果: 6, 7, 15, 36, 39, 40, 41, 42, 43, 47, 49；n=11

Q1位置1+（n-1）\*0.25=3.5，Q2位置：1+（n-1）\*0.5=6，Q3位置：1+（n-1）\*0.75=8.5

Q1位于第3位置到第4位置之间，Q1=较小数+（较大数-较小数）\*小数部分=15+21\*0.5=25.5

Q2等于6，直接得出答案，Q2=40

Q3位于第8位置到第九位置之间,Q3=42+1\*0.5=42.5

应用：

研究数据的变化趋势。



众数

众数是在一组数据中,出现次数最多的数据，是一组数据中的原数据，而不是相应的次数。

众数能反映出一组数据部分特征，很多时候，他还能暗示某种事物的将要发生。

from scipy import stats

import numpy as np

import pandas as pd

arraytest\_1=np.array([1,3,5,3,4,7,6,7,6,8,9,87,8,86,54,2,1,2,2,2,2,21,3,4,24])

stats.mode(arraytest\_1)

Out[7]: ModeResult(mode=array([2]), count=array([5]))

置信度：

在统计学中，一个概率样本的置信区间是对这个样本的某个总体参数（平均值）的区间估计。置信区间展现的是这个参数的真实值有一定概率落在该区间。

我们通常用到的是平均值置信度。

（备注：置信度和置信水平是两种说法一个东西，它与显著性水平相加等于1）

例子1：大选中某人的支持率为55%，而置信度0.95上的置信区间是（50%,60%），那么他的真实支持率有百分之九十五的机率落在百分之五十和百分之六十之间，因此他的真实支持率不足一半的可能性小于百分之2.5

例子2，假设全班考试的平均分数为65分，则

置信 区间间隔 宽窄度表达的意思

0－100分 100 宽等于什么也没告诉你

30－80分 50 较窄你能估出大概的平均分了（55分）

60－70分 10 窄到你几乎能判定全班的平均分了（65分）

-->0-100分时，区间宽度100，他的平均值最可能是50，因为置信区间来自50±50.

-->30-80分时，区间宽度50，它的平均值最可能是55，因为置信区间来自55±25.

-->60-70分时，区间宽度10，它的平均值最可能是65，因为置信区间来自65±5.

置信区间的变化：

在置信水平相同的情况下，样本量越多，置信区间越窄。

置信区间变窄的速度不像样本量增加的速度那么快，也就是说并不是样本量增加一倍，置信区间也变窄一倍（实践证明，样本量要增加4倍，置信区间才能变窄一倍），所以当样本量达到一个量时（通常是1,200），就不再增加样本了。

通过置信区间的计算公式来验证置信区间与样本量的关系：

置信区间=样本的推断值±（可靠程度系数×）

在样本量相同的情况下，置信水平越高，置信区间越宽。

在置信区间不变的情况下，样本量越多，置信水平越高。

平均值置信区间确定：

第一步：求一个样本的均值

第二步：计算出抽样误差。

人们经过实践，通常认为调查：

100个样本的抽样误差为±10%

500个样本的抽样误差为±5%

1200个样本时的抽样误差为±3%

第三步：用第一步求出的“样本均值”加、减第二步计算的“抽样误差”，得出置信区间的两个端点。

**常见统计参数的python调用**

偏度计算

arraytest\_4=pd.Series(np.array([3,6,23,4,3,6,7,8,21,12,17,11]))

stats.skew(arraytest\_4)

Out[29]: 0.7776194339665535

峰度计算

list\_4

Out[20]: [3, 6, 23, 4, 3, 6, 7, 8, 21, 12, 17, 11]

np.unique(list\_4)

Out[21]: array([ 3, 4, 6, 7, 8, 11, 12, 17, 21, 23])

aer=pd.Series(list\_4)

aer.kurt()

Out[24]: -0.44856357708607852

关联系数和协方差

Frame\_1=pd.DataFrame([[1,3.6,2.4],[6,29,21],[34,21,2]],index=['2015','2016','2017'],columns=['A','B','C'])

Frame\_1.corr()

Out[36]:

A B C

A 1.000000 0.344333 -0.390334

B 0.344333 1.000000 0.729967

C -0.390334 0.729967 1.000000

Frame\_1.cov()

Out[37]:

A B C

A 316.333333 79.533333 -75.366667

B 79.533333 168.653333 102.913333

C -75.366667 102.913333 117.853333

Frame\_1['A']

Out[38]:

2015 1

2016 6

2017 34

Name: A, dtype: int64

Frame\_1['A'].corr(Frame\_1['B'])

Out[39]: 0.34433347158312189

Frame\_1['A'].cov(Frame\_1['B'])

Out[40]: 79.533333333333331

# 四，数据加载，存储，清理，转换，合并与重塑

数据的加载与存储

Python在文本文件的加载与存储方面极其方便，这是它成为深受大家喜爱语言的原因之一。

Pandas 提供了一些直接将表格文件读取为DataFrame对象的函数

下面我们一一讲解：

A, read\_csv

import pandas as pd

!type example\_1.txt

系统找不到指定的文件。我们做一下简单修改

!type Desktop\example\_1.txt

Liu,Zhang,Wang,Li,Class

23,34,78,32,'primary'

77,32,89,66,'intermediate'

99,34,78,66,'senior'

66,34,6,33,'intermediate'

也可以直接读取‘CSV’文件

!type Desktop\example\_1.csv

Liu,Zhang,Wang,Li,Class

23,34,78,32,'primary'

77,32,89,66,'intermediate'

99,34,78,66,'senior'

66,34,6,33,'intermediate'

直接读取的数据相当于源数据，即没有转化为数据框的数据。

下面我们直接把CSV文件读成数据框：

frame\_1=pd.read\_csv('Desktop\example\_1.csv')

frame\_1

Out[10]:

Liu Zhang Wang Li Class

0 23 34 78 32 'primary'

1 77 32 89 66 'intermediate'

2 99 34 78 66 'senior'

3 66 34 6 33 'intermediate'

#我们发现我们得到一个完美的数据框

下面我们尝试用‘read\_table’

frame\_1=pd.read\_table('Desktop\example\_1.csv')

frame\_1

Out[15]:

Liu,Zhang,Wang,Li,Class

0 23,34,78,32,'primary'

1 77,32,89,66,'intermediate'

2 99,34,78,66,'senior'

3 66,34,6,33,'intermediate'

我们发现被读成的DataFrame排列很混乱，下面我们加上分割符号‘，’再运行一次。

frame\_1=pd.read\_table('Desktop\example\_1.csv',sep=',')

frame\_1

Out[13]:

Liu Zhang Wang Li Class

0 23 34 78 32 'primary'

1 77 32 89 66 'intermediate'

2 99 34 78 66 'senior'

3 66 34 6 33 'intermediate'

总结：用read\_table一定要加分割符号。

上面的例子中的csv数据我们都加了列名，如果没有列名会怎么样呢?

frame\_2=pd.read\_table('Desktop\example\_2.csv',sep=',')

frame\_2

Out[19]:

23 34 78 32 'primary'

0 77 32 89 66 'intermediate'

1 99 34 78 66 'senior'

2 66 34 6 33 'intermediate'

我们发现计算机自动把第一行数据当做列名，这样的话我们的源数据就遭到破坏。

为了避免这种错误，我们加入‘header’属性。让计算机自动加列名或者自己显性指定：

frame\_3=pd.read\_csv('Desktop\example\_2.csv',header=None)

frame\_3

Out[21]:

0 1 2 3 4

0 23 34 78 32 'primary'

1 77 32 89 66 'intermediate'

2 99 34 78 66 'senior'

3 66 34 6 33 'intermediate'

frame\_3=pd.read\_csv('Desktop\example\_2.csv',names=['a','b','c','d','class'])

frame\_3

Out[23]:

a b c d class

0 23 34 78 32 'primary'

1 77 32 89 66 'intermediate'

2 99 34 78 66 'senior'

3 66 34 6 33 'intermediate'

源文件中去掉双引号，这里双引号也会消失！

我们可以把最右边的列直接转化成行索引列。

frame\_3=pd.read\_csv('Desktop\example\_2.csv',names=['a','b','c','d','class'],index\_col='class')

frame\_3

Out[28]:

a b c d

class

'primary' 23 34 78 32

'intermediate' 77 32 89 66

'senior' 99 34 78 66

'intermediate' 66 34 6 33

除了可以读取为普通的数据框，还可以读取为带有重索引的数据框：

pd.read\_csv('Desktop\example\_5.csv',index\_col=['group','games'])

Out[45]:

a b c a.1 b.1 c.1 a.2 b.2 c.2

group games

one Att 4 3 4 2 4 3 4 2 1

Cgt 4 3 4 2 4 3 4 2 1

Aer 4 3 4 2 4 3 4 2 1

two Att 4 3 4 2 4 3 4 2 1

Cgt 4 3 4 2 4 3 4 2 1

Aer 4 3 4 2 4 3 4 2 1

对于那些用数量不等空格或者字符串隔开的数据，我们可以通过正则操作符一步处理到位，把数据处理整齐：

!type Desktop\exampel\_6.csv

A B C'

aaa -3.45 2.36 8.90'

bbb 0.334 0.457 -4.5'

ccc 0.76 -7.34 -8.99'

ddd 0.37 -7.8 -4.45'

pd.read\_csv('Desktop\exampel\_6.csv',sep='\s+')

Out[63]:

A B C'

aaa -3.450 2.360 8.90'

bbb 0.334 0.457 -4.5'

ccc 0.760 -7.340 -8.99'

ddd 0.370 -7.800 -4.45'

如果这里的源数据没有引号的话，这里引号也会自动消失。

利用skiprows语句可以跳过任意无意义行信息。

!type Desktop\example\_9.csv

title:Good Boy

Little Robert asked his mother for two cents. "What did you do with the money I gave you yesterday?"

a,b,c,d,name

"I gave it to a poor old woman," he answered.

"You're a good boy," said the mother proudly. "Here are two cents more. But why are you so interested in the old woman?"

group1,12,2,2,3,robert

group2,10,19,18,17,linz

"She is the one who sells the candy."

group3,29,28,27,27,hans

group4,34,35,35,36,manfried

整理后数据

pd.read\_csv('Desktop\example\_9.csv',skiprows=[0,1,3,4,7])

Out[69]:

a b c d name

group1 12 2 2 3 robert

group2 10 19 18 17 linz

group3 29 28 27 27 hans

group4 34 35 35 36 manfried

缺失数据要么没有，要么用某个标记值表示。请看下面两个例子：

!type Desktop\example\_10.csv

Liu,Zhang,Wang,Li,Class

23,34,78,32,NA

77,32,,66,'intermediate'

99,34,78,66,'senior'

66,34,6,33,'intermediate'

pd.read\_csv('Desktop\example\_10.csv')

Out[83]:

Liu Zhang Wang Li Class

0 23 34 78.0 32 NaN

1 77 32 NaN 66 'intermediate'

2 99 34 78.0 66 'senior'

3 66 34 6.0 33 'intermediate'

#上面例子产生NAN值，（直接在数字处空一位，就会产生NaN，比如数据1,2,4对应列标签，A,B,C,D）

!type Desktop\example\_11.csv

23,34,78,32,' '

77,32,89,66,'intermediate'

99,34, ,66,'senior'

66,34,6,33,'intermediate'

pd.read\_csv('Desktop\example\_11.csv')

Out[86]:

23 34 78 32 ' '

0 77 32 89 66 'intermediate'

1 99 34 66 'senior'

2 66 34 6 33 'intermediate'

#上面的例子什么都不显示（只有空字符串和空格）

通过na\_values python可以用字符串标识NAN值。但要注意源数据中字符串上不能有引号。

!type Desktop\example\_12.csv

A,B,C,D

1,2,3,4

5,me,7,8

8,10,11,me

12,13,14,16

pd.read\_table('Desktop\example\_12.csv',na\_values=['me'],sep=',')

Out[14]:

A B C D

0 1 2.0 3 4.0

1 5 NaN 7 8.0

2 8 10.0 11 NaN

3 12 13.0 14 16.0

可以用字符串字典的形式标识多空值数据框

!type Desktop\example\_13.csv

A,B,C,D

1,2,3,4

too,to,56,33

45,dee,78,69

12,21,34,43

pd.read\_table('Desktop\example\_13.csv',na\_values={'A':'too','B':['to','dee']},sep=',')

Out[18]:

A B C D

0 1.0 2.0 3 4

1 NaN NaN 56 33

2 45.0 NaN 78 69

3 12.0 21.0 34 43

爬虫数据例子：

df=pd.read\_csv('Desktop\pachong.csv',encoding='gbk',sep=',',header=None, skiprows=[14,47,49,50,51,59,68,83,104,125,127,128,136,150,173,184])

用这个指令可以处理中文数据

文本文件的块读取

如果只想读取数据一部分，或者数据过大，我们想逐步读取，我们可以通过附加属性函数来实现。

通过nrows属性函数可以选择所需要的行数

!type Desktop\example\_14.csv

one,two,three,four

0,1,2,3

4,5,6,7

8,9,10,11

12,13,14,15

16,17,18,19

20,21,22,23

12,13,14,15

16,17,18,19

20,21,22,23

12,13,14,15

16,17,18,19

20,21,22,23

12,13,14,15

16,17,18,19

20,21,22,23

0,1,2,3

4,5,6,7

8,9,10,11

12,13,14,15

16,17,18,19

20,21,22,23

12,13,14,15

16,17,18,19

pd.read\_table('Desktop\example\_14.csv',sep=',',nrows=8)

Out[21]:

one two three four

0 0 1 2 3

1 4 5 6 7

2 8 9 10 11

3 12 13 14 15

4 16 17 18 19

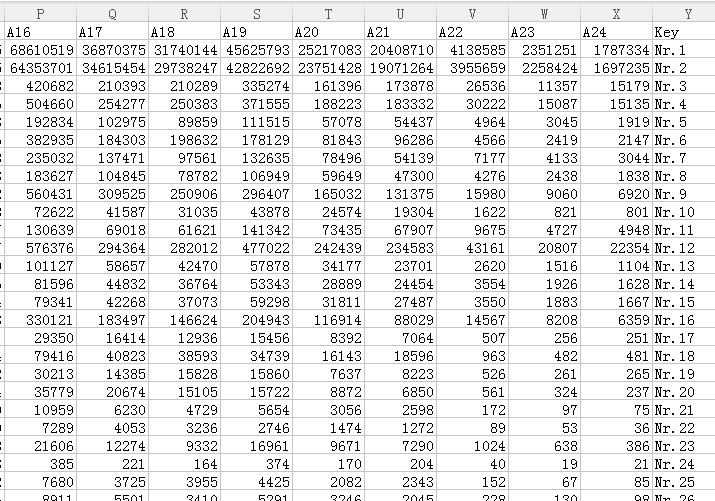
5 20 21 22 23

6 12 13 14 15

7 16 17 18 19

实验项目：数据的分块读取：

原始数据的部分截取：



block\_2=pd.read\_csv('Desktop\lockreading.csv',chunksize=59)

key\_quantity=pd.Series([])

#block\_2是对整个数据分块后的返回值，可看做一个三维数据框，key\_quantity是一个存储行编号数量的series

for block\_var in block\_2:

key\_quantity=key\_quantity.add(block\_var['Key'].value\_counts(),fill\_value=0)

**{千万注意：这里fill\_value很重要，因为在循环过程中，由于相加的两个Seires的行标签不可能每一次都完全匹配，行标签不完全匹配的两个Series相加如果会产生空值，这种情况下会影响下次循环series的相加，因为空值+实数会继续产生空值，从而计数错误。为了消除这种现象的产生，最好的办法是每一次相加后产生的空值立马填充为‘0’值，这样就不会影响下一步两个series的相加。**

**看下面例子：**

**import pandas**

**import pandas as pd**

**serie\_1=pd.Series([1,2,3],index=list('abc'))**

**serie\_2=pd.Series([4,5,8],index=list('abd'))**

**serie\_3=serie\_1+serie\_2**

**serie\_3**

**Out[6]:**

**a 5.0**

**b 7.0**

**c NaN**

**d NaN**

**dtype: float64**

**#假如我们添加’fill\_value=0‘**

**serie\_1.add(serie\_2,fill\_value=0)**

**Out[8]:**

**a 5.0**

**b 7.0**

**c 3.0**

**d 8.0**

**dtype: float64 }**

**接下来我们继续我们的项目：**

key\_quantity=key\_quantity.sort\_values(ascending=False)

key\_quantity

Out[16]:

Nr.36 10.0

Nr.30 10.0

Nr.32 10.0

Nr.33 10.0

Nr.34 10.0

Nr.35 10.0

Nr.37 10.0

Nr.38 10.0

Nr.39 10.0

Nr.29 10.0

Nr.40 10.0

Nr.41 10.0

Nr.28 10.0

Nr.27 10.0

Nr.26 10.0

Nr.25 10.0

Nr.24 10.0

Nr.23 10.0

Nr.31 10.0

Nr.42 9.0

Nr.21 7.0

Nr.20 7.0

Nr.2 7.0

Nr.19 7.0

Nr.18 7.0

Nr.17 7.0

Nr.22 7.0

Nr.15 7.0

Nr.14 7.0

Nr.3 7.0

Nr.13 7.0

Nr.12 7.0

Nr.11 7.0

Nr.10 7.0

Nr.16 7.0

Nr.9 7.0

Nr.8 7.0

Nr.52 7.0

Nr.7 7.0

Nr.6 7.0

Nr.59 7.0

Nr.58 7.0

Nr.57 7.0

Nr.56 7.0

Nr.55 7.0

Nr.54 7.0

Nr.53 7.0

Nr.51 7.0

Nr.4 7.0

Nr.50 7.0

Nr.5 7.0

Nr.49 7.0

Nr.48 7.0

Nr.47 7.0

Nr.46 7.0

Nr.45 7.0

Nr.44 7.0

Nr.43 7.0

Nr.1 7.0

dtype: float64

#上面这个Series左边是行编号，（注意不是行索引，行编号是key列中的值，它是每一行的编号），右边是每个编号在整个数据里出现的次数。

通过一个小函数我们能够瞬间遍查每个数据块寻找任何一个给定的行编号。

def search\_rowindizies(str\_test):

list\_1=[]

for block\_var in block\_2:

boll=block\_var['Key']==str\_test

st=block\_var.loc[boll,:'A24'].values

lst=st.astype(np.float32)

#alst=list(lst)

list\_1.append(lst)

return list\_1

block\_2=pd.read\_csv('Desktop\lockreading.csv',chunksize=59)

search\_rowindizies('Nr.42')

Out[19]:

[array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32),

array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32),

array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32),

array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32),

array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32),

array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32),

array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32),

array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00],

[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32)]

**#每次执行函数都要调用一次block\_2,因为数据用完一次后自动与源数据断开，Block\_2自动清零消失**

到此为止，我们的项目结束！！

把数据下载到csv 文件

DataFrame\_1=pd.DataFrame(np.arange(16).reshape(4,4),index=list('asdt'),columns=list('sgdt'))

DataFrame\_1

Out[5]:

s g d t

a 0 1 2 3

s 4 5 6 7

d 8 9 10 11

t 12 13 14 15

DataFrame\_1.to\_csv('Desktop\DataFrame\_1.csv')

#数据储存到桌面，文件名是DataFrame\_1。默认情况下，To\_csv方法只会把数据存为用逗号隔开的csv数据。我们可以通过!type函数直接调用这个已存到桌面上的文件，就会发现它的确如此，如下：

!type Desktop\DataFrame\_1.csv

,s,g,d,t

a,0,1,2,3

s,4,5,6,7

d,8,9,10,11

t,12,13,14,15

#上面是没有转化为数据框的源数据格式，也就是csv文件的直接调取。

当然，我们也可以直接将数据直接存其它分割方式，例如下面的例子

DataFrame\_1.to\_csv('Desktop\DataFrame\_2.csv',sep='#')

!type Desktop\DataFrame\_2.csv

#s#g#d#t

a#0#1#2#3

s#4#5#6#7

d#8#9#10#11

t#12#13#14#15

DataFrame\_2=DataFrame\_1.copy()

DataFrame\_2.loc['a','s']=np.nan

DataFrame\_2

Out[32]:

s g d t

a NaN 1 2 3

s 4.0 5 6 7

d 8.0 9 10 11

t 12.0 13 14 15

DataFrame\_2.to\_csv('Desktop\DataFrame\_3.csv')

!type Desktop\DataFrame\_3.csv

,s,g,d,t

a,,1,2,3

s,4.0,5,6,7

d,8.0,9,10,11

t,12.0,13,14,15

#再重新通过！type指令调回已存储的文件后，发现数据框原有的NaN值变成空位置，然而，空位置在csv文件中并不好辨认，因此我们需要把含有nan值的数据框存储为nan值被指定的容易辨认的符号替代的csv文件。

DataFrame\_2

Out[50]:

s g d t

a NaN 1 2 3

s 4.0 5 6 7

d 8.0 9 10 11

t 12.0 13 14 15

DataFrame\_2.to\_csv('Desktop\DataFrame\_4.csv',na\_rep='Cat')

!type Desktop\DataFrame\_4.csv

,s,g,d,t

a,Cat,1,2,3

s,4.0,5,6,7

d,8.0,9,10,11

t,12.0,13,14,15

#用!type调用储存在桌面的文件DataFrame\_4.csv，我们会发现原来在数据框的NaN值在存储后被‘Cat’替代

这里我们要注意空位置与空字符串不一样

试比较：

pd.read\_csv('Desktop\example\_16.csv')

Out[42]:

a b c d

0 ' ' 123 456 89

1 22 65 ' ' 12

2 345 89 0 6

3 23 ' ' 33 66

!type Desktop\example\_16.csv

a,b,c,d

' ',123,456,89

22,65,' ',12

345,89,0,6

23,' ',33,66

上面是空字符串

pd.read\_csv('Desktop\example\_17.csv')

Out[44]:

a b c d

0 NaN 123.0 456.0 89

1 22.0 65.0 NaN 12

2 345.0 89.0 0.0 6

3 23.0 NaN 33.0 66

!type Desktop\example\_17.csv

a,b,c,d

,123,456,89

22,65,,12

345,89,0,6

23,,33,66

上面是空位置，空位置再读取成数据框时会产生NaN值。

如果没有特殊指定，数据框行和列的标签会随数据一起存储为csv文件，当然我们也可以不这样。如下：

DataFrame\_1

Out[51]:

s g d t

a 0 1 2 3

s 4 5 6 7

d 8 9 10 11

t 12 13 14 15

DataFrame\_1.to\_csv('Desktop\Dataframe\_12.csv',index=False,header=False)

!type Desktop\Dataframe\_12.csv

0,1,2,3

4,5,6,7

8,9,10,11

12,13,14,15

我们发现行标签和列标签都不见了。

我们还可以只存储数据框的部分列，并指定顺序：

DataFrame\_1.to\_csv('Desktop\Dataframe\_13.csv',index=False,columns=['s','t','d'])

调用存储文件，测试！！

!type Desktop\Dataframe\_13.csv

s,t,d

0,3,2

4,7,6

8,11,10

12,15,14

我们发现只有部分列被存储，且列的顺序按照指定。

Series 也可以通过to\_csv来存储信息到桌面。

dates\_1=pd.date\_range('01/06/2017',periods=14)

dates\_1

Out[60]:

DatetimeIndex(['2017-01-06', '2017-01-07', '2017-01-08', '2017-01-09',

'2017-01-10', '2017-01-11', '2017-01-12', '2017-01-13',

'2017-01-14', '2017-01-15', '2017-01-16', '2017-01-17',

'2017-01-18', '2017-01-19'],

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

series\_12=pd.Series(np.arange(14),index=dates\_1)

series\_12.to\_csv('Desktop\series\_123.csv')

!type Desktop\series\_123.csv

2017-01-06,0

2017-01-07,1

2017-01-08,2

2017-01-09,3

2017-01-10,4

2017-01-11,5

2017-01-12,6

2017-01-13,7

2017-01-14,8

2017-01-15,9

2017-01-16,10

2017-01-17,11

2017-01-18,12

2017-01-19,13

csv文件也可以从桌面直接被读取成Series：

pd.Series.from\_csv('Desktop\series\_123.csv')

Out[71]:

2017-01-06 0

2017-01-07 1

2017-01-08 2

2017-01-09 3

2017-01-10 4

2017-01-11 5

2017-01-12 6

2017-01-13 7

2017-01-14 8

2017-01-15 9

2017-01-16 10

2017-01-17 11

2017-01-18 12

2017-01-19 13

dtype: int64

注：

1. Read\_csv可以读取文件，url，文件型对象，但被加载文件必须有分割符，默认的分割符为逗号。
2. Read\_table可以读取文件，url，文件型对象，但被加载文件必须有分割符，默认的分割符为制表符‘\t’。这一点与read\_csv不同，烦请再加载有逗号的数据时用seq属性注明，也即seq=‘，’！
3. Read\_fwf读取（或称加载）没有分隔符数据，但是各数据之间间距要恒定。
4. Read\_clipboard 通常用来读取网络数据,在使用前，必须把网页内容先复制到粘贴板上，例如：

pd.read\_clipboard(sep='\\s+')

Out[73]:

北 京 220956 209468 11488 697.02 264.30 253555 44087 701.42

0 天 津 93162 90080 3082 330.68 126.98 106063 15983 340.10

1 河 北 209740 200012 9728 963.39 327.27 262396 62384 982.31

2 山 西 131802 114466 17336 628.55 192.27 214625 100159 669.39

3 内蒙古 71196 65627 5569 306.82 110.15 101829 36202 308.10

4 　 　 　 　 　 　 　 　 　NaN

5 辽 宁 211502 199611 11891 838.09 305.58 268741 69130 846.25

上面函数常用关键字列表

|  |  |
| --- | --- |
| filepath\_or\_buffer: | 文件系统位置，url，文件型对象的字符串 |
| delimiter或sep | 源文件各数据间分隔符或正则表达式 |
| header | 上载数据成数据框时，数据框的列名，默认为上载数据的第一行。即‘0’行。如果不需要列标签，那么使header=None |
| index\_col | 就是给行层次化索引命名，行层次索引一般有两列，内列和外列。 |
| skiprows | 需要忽略的行数，跳过无用行 |
| na\_values | 实质就是把源数据中的指定字符串转化为空值。 |
| converters | 由列名或者说列号或者说列标签和函数组成字典，例如{‘A’:f}说明f函数应用到’A’列中的每一个数据。 |
| nrows | 需要读取的数据行数 |
| skip\_footer | 需要忽略的行数，注意从源数据最后一行向上数 |
| encoding | 用于指明unicode文本文件的文本编码格式 |
| Squeeze | 如果数据仅有一列，自动返回series |
| thousands | 千分位分隔符，如’.’或’,’ |

分隔符格式的手工处理

通常用read\_table或者read\_csv直接加载csv文件或者TXT文件都是没有问题的，但有时csv文件十分混乱，以致无法加载，也是十分常见的，这就要求我们在加载前要对原始csv文件进行处理，使其能够被顺利加载。

我们先看一个单字符分隔符文件

import pandas as pd

import numpy as np

!type Desktop\example\_20.csv

"f","g","h","d"

"1","1","4","5"

"7","9","0","4"

"2","8","11",0

import csv

调用内置csv模块，将已打开文件直接传给csv.reader

f=open('Desktop\example\_20.csv')

getting\_1=csv.reader(f)

getting\_1

Out[25]: <\_csv.reader at 0x3077b74320>

#我们无法看到getting\_1的庐山真面目，怎么办？通过for循环，我们看到getting\_1中的每个元素。这是一个十分好的方法，请大家务必注意：

for subgetting in getting\_1:

print(subgetting)

['f', 'g', 'h', 'd']

['1', '1', '4', '5']

['7', '9', '0', '4']

['2', '8', '11', '0']

我下面再举一个具体的例子：

range(1,9,1)

Out[27]: range(1, 9)

for subrange in range(1,9,1):

print(subrange)

1

2

3

4

5

6

7

8

当然这里我们还有一个终极解决方案，也就是通过list。

Gate\_1=list(csv.reader(open('Desktop\example\_20.csv')))

Gate\_1

Out[32]:

[['f', 'g', 'h', 'd'],

['1', '1', '4', '5'],

['7', '9', '0', '4'],

['2', '8', '11', '0']]

#直到这里我们终于看到了csv.reader()的庐山真面目。

header,values=Gate\_1[0],Gate\_1[1:]

dic\_new\_1={u:v for u,v in zip(header,zip(\*values))}

dic\_new\_1

Out[36]:

{'d': ('5', '4', '0'),

'f': ('1', '7', '2'),

'g': ('1', '9', '8'),

'h': ('4', '0', '11')}

最终我们把相对杂乱无章的数据转化为规整的字典。

解释：zip（\*2dim-Array）

这个操作相当于把嵌套列表每一行一一对应先组成元组，然后组成列表：

c=[[1,2,3],[4,7,0]]

zip(\*c)

Out[40]: <zip at 0x3077b9ce08>

list(zip(\*c))

Out[41]: [(1, 4), (2, 7), (3, 0)]

我们可以自定义csv文件的读取或写入格式：

Import csv

class mydialect(csv.Dialect):

lineterminator = '\n' #用于写操作时的行结束符，读操作时将忽略此项，也就是读操作时无需在文件数据行尾处加‘\n’

delimiter = '#' #可以读取用‘#’分割的字段（或者数据）

quotechar='"' #用于带有特殊字符的字段（通常指的是带有单引号的字符串）的引用符号，例如源数据这样的字段’12#3’通过csv.reader函数读取到python程序页面后会变成”’12”,”3’”。

quoting=csv.QUOTE\_ALL #读取所有字段（和数据）

skipinitialspace=True #或略分隔符后面的空格

doublequote=True #如果读取的字段含有引用符号，则整个字段加双引号。

下面是一个比较简单的例子：

Import csv

class mydialect(csv.Dialect):

lineterminator = '\n'

delimiter = '#'

quotechar='"'

quoting=csv.QUOTE\_ALL

skipinitialspace=True

doublequote=True

myfile\_1=open('Desktop\example\_223.csv')

content\_11=csv.reader(myfile\_1,dialect=mydialect)

for contents in content\_11:

print(contents)

['At', 'bee', 'cotton', 'death', "common'hero'tree"]

['1', '2', '3', '4']

['5', '6', '7', '8', '9']

['9', '10', '11', '12', "'12", "3'"]

!type Desktop\example\_223.csv

At#bee#cotton# death#common'hero'tree

1#2# 3#4

5#6#7#8#9

9#10#11#12#'12#3'

我们上面一直在讨论通过定义csv.Dialect的一个子类来读取独有格式的csv，下面我们谈谈写入

class mydialect(csv.Dialect):

lineterminator = '\n'

delimiter = '#'

quotechar='"'

quoting=csv.QUOTE\_ALL

skipinitialspace=True

doublequote=True

with open('Desktop\example\_25.csv','w') as newsetting\_1: #首先open()函数打开文件'Desktop\example\_25.csv'，如果不存在这个文件，则创建它，返回文件对象newsetting\_1。

writer\_object=csv.writer(newsetting\_1,dialect=mydialect) #创建一个写对象'writer\_object'，可理解成给打开文件添加独有编码风格“mydialect”。然后返回一个带有独有编码风格的新文件对象

writer\_object.writerow(["'one'",0, 8]) #可通过列表把内容一步步地写入到打开的文件

writer\_object.writerow(['two',62,43])

mylist=[['three',46,88],['four', 3,12]]

writer\_object.writerows(mylist)

!type Desktop\example\_25.csv

one'#"0"#"8"

two#"62"#"43"

three#"46"#"88"

four#"3"#"12"

JSON数据

JSON是一种数据格式，它是通过HTTP请求在web浏览器与其他应用程序之间数据传送格式之一。

例如：

Json\_documents="""

{"Name":"Wes","places\_lived":["United States","Spain","Germany","Japan"],"pet":null,"siblings":[{"name":"Scott","age":25,"pet":"Zuko","weight":56,"height":172},{"name":"Katie","age":33,"pet":"Cisco","weight":89,"height":189}]}

"""

这是一个Json数据格式字符串数据对象，可以通过python标准库中的json.loads函数即可将JSON字符串数据转化为python字符串数据。

import json

getting\_1=json.loads(Json\_documents)

getting\_1

Out[36]:

{'Name': 'Wes',

'pet': None,

'places\_lived': ['United States', 'Spain', 'Germany', 'Japan'],

'siblings': [{'age': 25,

'height': 172,

'name': 'Scott',

'pet': 'Zuko',

'weight': 56},

{'age': 33, 'height': 189, 'name': 'Katie', 'pet': 'Cisco', 'weight': 89}]}

试比较这两种数据结构，我们会发现：

1. JSON格式的字符串数据对象的字符串没有排序，确切的说字典或者说JSON对象的键和其所对应的子字典内容都没有任何排序。而python格式的字典缺有严格工整的排序，无论键还是键所对应的字典都有严格排序。
2. 所有字符串的双引号都变成单引号
3. JSON格式的字典最前面和最后面都有‘”””’（也可用“’”）标识，转化成Python格式后所有标识全部消失。
4. JSON的空值用null表达，Python用None表达。

有一点我们必须要注意，JSON格式的对象（JSON有对象（即字典）、数组、字符串、数字，bool值以及null等数据结构）的键必须是字符串。如果不是字符串将会在转换时出现错误提示:

Json\_documents="""

{23:"Wes","places\_lived":["United States","Spain","Germany","Japan"],"pet":null,"siblings":[{"name":"Scott","age":25,"pet":"Zuko","weight":56,"height":172},{"name":"Katie","age":33,"pet":"Cisco","weight":89,"height":189}]}

"""

getting\_1=json.loads(Json\_documents)

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

JSONDecodeError Traceback (most recent call last)

<ipython-input-38-66d19b29dad0> in <module>()

----> 1 getting\_1=json.loads(Json\_documents)

~\Anaconda3\lib\json\\_\_init\_\_.py in loads(s, encoding, cls, object\_hook, parse\_float, parse\_int, parse\_constant, object\_pairs\_hook, \*\*kw)

352 parse\_int is None and parse\_float is None and

.

.

.

357 raise JSONDecodeError("Expecting value", s, err.value) from None

JSONDecodeError: **Expecting property name enclosed in double quotes**: line 2 column 2 (char 2)

我们当然也可以把python格式的数据对象转化为JSON格式，使用python的函数json.dumps可以实现这一愿望：

例子：

python\_format\_1={'Name': 'Wes',

'pet': None,

'places\_lived': ['United States', 'Spain', 'Germany', 'Japan'],

'siblings': [{'age': 25,

'height': 172,

'name': 'Scott',

'pet': 'Zuko',

'weight': 56},

{'age': 33, 'height': 189, 'name': 'Katie', 'pet': 'Cisco', 'weight': 89}]}

json\_format\_1=pd.json.dumps(python\_format\_1)

json\_format\_1

Out[6]:'{"Name":"Wes","pet":null,"places\_lived":["United States","Spain","Germany","Japan"],"siblings":[{"age":25,"height":172,"name":"Scott","pet":"Zuko","weight":56},{"age":33,"height":189,"name":"Katie","pet":"Cisco","weight":89}]}'

再把json格式数据转化为python数据格式后，我们需要把数据作进一步处理，以便我们更好的处理数据。最常用的做法是把数据继续转化为数据框：

newDataFrame=pd.DataFrame(python\_format\_1['siblings'])

newDataFrame

Out[8]:

age height name pet weight

0 25 172 Scott Zuko 56

1 33 189 Katie Cisco 89

python\_format\_1['siblings']表示的是一个由字典组成的列表，通过它可以直接生成一个数据框，这是一个很好的数据框创建方法。我们下面一起做一些用这种模式创建数据框的练习：

例子1：

dict\_1={'c1':[1,2,3,4,5],'c2':[21,65,32,90,0],'c3':[90,67,5,6,3],'c4':[1,2,1,2,0],'c5':[9,0,8,7,0]}

dict\_2={'c1':[1,3,3,4,5],'c2':[21,5,32,90,3],'c3':[90,7,5,6,3],'c4':[1,2,155,2,0],'c5':[9,450,8,7,0]}

newframe\_0=pd.DataFrame(dict\_1)

newframe\_0

Out[19]:

c1 c2 c3 c4 c5

0 1 21 90 1 9

1 2 65 67 2 0

2 3 32 5 1 8

3 4 90 6 2 7

4 5 0 3 0 0

例子2：

newframe\_0=pd.DataFrame([dict\_1])

newframe\_0

Out[17]:

c1 c2 c3 c4 \

0 [1, 2, 3, 4, 5] [21, 65, 32, 90, 0] [90, 67, 5, 6, 3] [1, 2, 1, 2, 0]

c5

0 [9, 0, 8, 7, 0]

例子3：

newframe\_1=pd.DataFrame([dict\_1,dict\_2]):

newframe\_1

Out[15]:

c1 c2 c3 c4 \

0 [1, 2, 3, 4, 5] [21, 65, 32, 90, 0] [90, 67, 5, 6, 3] [1, 2, 1, 2, 0]

1 [1, 3, 3, 4, 5] [21, 5, 32, 90, 3] [90, 7, 5, 6, 3] [1, 2, 155, 2, 0]

c5

0 [9, 0, 8, 7, 0]

1 [9, 450, 8, 7, 0]

例子4

dic\_3={'c6':[1,3,3,4,5],'c7':[21,5,32,90,3],'c8':[90,7,5,6,3],'c9':[1,2,155,2,0],'c10':[9,450,8,7,0]}

newFrame\_4=pd.DataFrame([dict\_1,dic\_3])

newFrame\_4

Out[22]:

c1 c10 c2 c3 \

0 [1, 2, 3, 4, 5] NaN [21, 65, 32, 90, 0] [90, 67, 5, 6, 3]

1 NaN [9, 450, 8, 7, 0] NaN NaN

c4 c5 c6 c7 \

0 [1, 2, 1, 2, 0] [9, 0, 8, 7, 0] NaN NaN

1 NaN NaN [1, 3, 3, 4, 5] [21, 5, 32, 90, 3]

c8 c9

0 NaN NaN

1 [90, 7, 5, 6, 3] [1, 2, 155, 2, 0]

#数据框的列索引由字典的键确定，字典列表中的有多少字典键就有多少列，譬如字典列表的第一个字典（0位置字典）有5个键，第二个字典（1位置字典）有5个键，那么生成的数据框就有10个列。字典列表的长度代表了行的数量，就如例子中字典列表的长度是2，那么数据框就有两行，0行和1行。

通过columns我们可以指定需要显示的数据列：

newframe\_5=pd.DataFrame([dict\_1,dict\_2],columns=['c2','c3'])

newframe\_5

Out[24]:

c2 c3

0 [21, 65, 32, 90, 0] [90, 67, 5, 6, 3]

1 [21, 5, 32, 90, 3] [90, 7, 5, 6, 3]

数据的二进制格式存储

定义：将对象转换为可通过网络传输或可以存储到本地磁盘的数据格式的过程称为序列化；反之，则称为反序列化。Python内置的pickle序列化用于实现Python数据类型与Python特定二进制格式之间的转换。

import pandas as pd

frame\_1=pd.read\_csv('Desktop\exampel\_6.csv')

frame\_1=pd.read\_csv('Desktop\exampel\_6.csv',sep='\s+')

frame\_1

Out[6]:

A B C'

aaa -3.450 2.360 8.90'

bbb 0.334 0.457 -4.5'

ccc 0.760 -7.340 -8.99'

ddd 0.370 -7.800 -4.45'

frame\_1.to\_pickle('Desktop\frame\_pickle')

#上面我们把frame\_1存储成pickle格式

我们也可以把数据读回到python。

import pickle

load\_file=open('Desktop\Frame\_pickle','rb')

pickle.load(load\_file)

Out[17]:

A B C'

aaa -3.450 2.360 8.90'

bbb 0.334 0.457 -4.5'

ccc 0.760 -7.340 -8.99'

ddd 0.370 -7.800 -4.45'

**HDF5的存取**

Hierarchical Data Format(HDF)是一种针对大量数据进行组织和存储的文件格式。它包含了数据模型，库，和文件格式标准。以其便捷有效，移植性强，灵活可扩展的特点受到了广泛的关注和应用。

对大数据的组织和存储得益于HDF文件的系统式节点结构，这样的结构不仅支持元数据，而且使多数据集存储成为现实。

Python中的HDF5库有两个接口Pytables和h5py。

我们先看一下h5py，h5py是python的一种工具包，它提供了一种直接而高级的HDF5 API访问接口。

import h5py

import numpy as np

#HDF5的写入

Intentity\_12=np.ones((10,12,300,305))

data\_h5py\_1=h5py.File('Desktop\example\_29-1.h5','w')

#创建一个h5文件

data\_h5py\_1['data']=Intentity\_12

#把数据写入到data\_h5py\_1主键‘data’中

data\_h5py\_1['labels']=range(120)

#把数据写入到data\_h5py\_1主键‘labels’中

data\_h5py\_1.close()

#HDF5的读取

data\_h5py\_2=h5py.File('Desktop\example\_29-1.h5','r')

#h5文件的读取

key\_content=data\_h5py\_2.keys()

#所有键的查看

abc\_123=data\_h5py\_2['data'][:]

#查看主键‘data’的内容

data\_h5py\_2['labels'][:]

#查看主键‘labels’的内容

我们再看另一种接口Pytables，Pytables也是python的一种工具包，它抽像了许多HDF5的细节以提供多种灵活数据容器，及实现表索引和查询功能。

import pandas as pd

store=pd.HDFStore('Desktop\example\_30.h5','w')

#通过HDFStore创建一个h5文件

getframe\_1=pd.DataFrame(np.arange(16).reshape(4,4),index=['r1','r2','r3','r4'],columns=['c1','c2','c3','c4'])

store['layer\_1']=getframe\_1

store['layer\_1\_1']=getframe\_1['c2']

store

Out[50]:

<class 'pandas.io.pytables.HDFStore'>

File path: Desktop\example\_30.h5

/layer\_1 frame (shape->[4,4])

/layer\_1\_1 series (shape->[4])

store['layer\_1']['c4']

Out[51]:

r1 3

r2 7

r3 11

r4 15

Name: c4, dtype: int32

注意：

1. HDF5最好一次写入，不要多次写入,以免破坏文件
2. 创建H5文件时，各种文件类型。

r：只能读

**r+：可读可写，不会创建不存在的文件。如果直接写文件，则从顶部开始写，覆盖之前此位置的内容，如果先读后写，则会在文件最后追加内容。**

w：只能写，覆盖整个文件，文件不存在则创建   
a：只能写，从文件底部添加内容，文件不存在则创建

**数据的合并**

常用的数据合并函数有pandas.merge和pandas.contact以及combine\_first。它们分别有着各自的应用条件。我们根据实际情况选取合适的函数来进行数据合并。

1. pandas.merge

通过merge函数可以实现数据框的合并，但这种合并要经过一个或多个键的衔接，这就要求我们必须在数据框中专门添加一个‘Key’列用于衔接。

例1：

Frame\_1=pd.DataFrame({'schluessel':['Nr.1','Nr.2','Nr.3','Nr.4','Nr.5'],'values\_1':[1,2,2,8,9],'values\_2':[4,7,9,0,1]})

Frame\_2=pd.DataFrame({'schluessel':['Nr.1','Nr.1','Nr.3','Nr.3','Nr.5'],'values\_1':[1,2,2,8,9],'values\_2':[4,7,9,0,1]})

Frame\_1

Out[28]:

schluessel values\_1 values\_2

0 Nr.1 1 4

1 Nr.2 2 7

2 Nr.3 2 9

3 Nr.4 8 0

4 Nr.5 9 1

Frame\_2

Out[29]:

schluessel values\_1 values\_2

0 Nr.1 1 4

1 Nr.1 2 7

2 Nr.3 2 9

3 Nr.3 8 0

4 Nr.5 9 1

pd.merge(Frame\_2,Frame\_1,on='schluessel')

Out[27]:

schluessel values\_1\_x values\_2\_x values\_1\_y values\_2\_y

0 Nr.1 1 4 1 4

1 Nr.1 2 7 1 4

2 Nr.3 2 9 2 9

3 Nr.3 8 0 2 9

4 Nr.5 9 1 9 1

**多对一（重复键列对不重复键列）内连接注意事项**

每次合并前都要指定作为键的列，比如上面我们指定“schluessel”

作为键的列如有重复键要保留，不要去重，看另一个待合并数据框有没有这个键，如果有，把相同的键合并到一起。

如果某些键在各自数据框中都没有重复现象，但是两个数据框相比这些键相同，那么要把两个键合并到一起。

如果某些键在各自数据框中都没有重复现象，且两个数据框相比这些键也不相同，那么这些键将不出现在合并后的键列里

注意：默认是内连接

如果键名不同，我们可以通过关键字“left\_on”和“right\_on”分别安置：

frame\_2=pd.DataFrame({'rkey':['a','c','d'],'data\_2':[3,2,1]})

frame\_1=pd.DataFrame({'lkey':['a','a','c','a','b','c','d','b'],'data\_1':[2,6,9,3,2,1,0,5]})

pd.merge(frame\_1,frame\_2,left\_on='lkey',right\_on='rkey')

Out[5]:

data\_1 lkey data\_2 rkey

0 2 a 3 a

1 6 a 3 a

2 3 a 3 a

3 9 c 2 c

4 1 c 2 c

5 0 d 1 d

在默认的条件下，合并后的键是两个数据框的键的交集，例如上面的例子均是如此，这种键的合并方式被称作内连接，除了内连接还有左连接、右连接以及外连接。下满我们分别比较这几种方式：

frame\_2=pd.DataFrame({'sl':['a','c','d','f'],'data\_2':[3,2,1,7]})

frame\_1=pd.DataFrame({'sl':['a','a','c','a','b','c','d','b'],'data\_1':[2,6,9,3,2,1,0,5]})

pd.merge(frame\_1,frame\_2,on='sl',how='inner')

Out[16]:

data\_1 sl data\_2

0 2 a 3

1 6 a 3

2 3 a 3

3 9 c 2

4 1 c 2

5 0 d 1

#内连接是键的交集

pd.merge(frame\_1,frame\_2,on='sl',how='left')

Out[17]:

data\_1 sl data\_2

0 2 a 3.0

1 6 a 3.0

2 9 c 2.0

3 3 a 3.0

4 2 b NaN

5 1 c 2.0

6 0 d 1.0

7 5 b NaN

pd.merge(frame\_2,frame\_1,on='sl',how='left')

Out[6]:

data\_2 sl data\_1

0 3 a 2.0

1 3 a 6.0

2 3 a 3.0

3 2 c 9.0

4 2 c 1.0

5 1 d 0.0

6 7 f NaN

#左连接以第一个数据框的键列为合并参考键列，观察参考键列中的键是否在两个待合并数据框中有重复，有则在合并时也要重复。

pd.merge(frame\_1,frame\_2,on='sl',how='right')

Out[18]:

data\_1 sl data\_2

0 2.0 a 3

1 6.0 a 3

2 3.0 a 3

3 9.0 c 2

4 1.0 c 2

5 0.0 d 1

6 NaN f 7

#右连接以第二个数据框的键列为合并参考键列，观察参考键列中的键是否在两个待合并数据框中有重复，有则在合并时也要重复。

pd.merge(frame\_1,frame\_2,on='sl')

Out[19]:

data\_1 sl data\_2

0 2 a 3

1 6 a 3

2 3 a 3

3 9 c 2

4 1 c 2

5 0 d 1

我们可以看到，“how=’inner’”和how在缺失的状况下运行结果是一样的。

下面我们讨论多对多键的操作：

内连接

frame\_2=pd.DataFrame({'sl':['a','c','a','d','f'],'data\_2':[3,2,1,7,6]})

frame\_1=pd.DataFrame({'sl':['a','a','c','a','b','c','d','b'],'data\_1':[2,6,9,3,2,1,0,5]})

pd.merge(frame\_2,frame\_1,on='sl')

Out[9]:

data\_2 sl data\_1

0 3 a 2

1 3 a 6

2 3 a 3

3 1 a 2

4 1 a 6

5 1 a 3

6 2 c 9

7 2 c 1

8 7 d 0

**多对多键合并注意事项（内连接）:**

#指定合并键列

#作为键的列如有重复键要保留，不要去重，看另一个待合并数据框有没有这个键，如果有，把相同的键合并到一起并重复。如果在一个数据框重复的键在另一个数据框中也重复出现，那么合并后它重复的次数符合笛卡尔积。

如果某些键在各自数据框中都没有重复现象，但是两个数据框相比这些键相同，那么要把两个键合并到一起。

如果某些键在各自数据框中都没有重复现象，且两个数据框相比这些键也不相同，那么这些键将不出现在合并后的键列里

frame\_2

Out[17]:

data\_2 sl

0 3 a

1 2 c

2 1 a

3 7 d

4 6 f

frame\_1

Out[18]:

data\_1 sl

0 2 a

1 6 a

2 9 c

3 3 a

4 2 b

5 1 c

6 0 d

7 5 b

pd.merge(frame\_2,frame\_1,on='sl',how='left')

Out[16]:

data\_2 sl data\_1

0 3 a 2.0

1 3 a 6.0

2 3 a 3.0

3 2 c 9.0

4 2 c 1.0

5 1 a 2.0

6 1 a 6.0

7 1 a 3.0

8 7 d 0.0

9 6 f NaN

#左连接以第一个数据框的键列为合并参考键列，观察参考键列中的键是否在两个待合并数据框中有重复，有则在合并时也要重复。

如果在一个数据框重复的键在另一个数据框中也重复出现，那么合并后它重复的次数符合笛卡尔积。

pd.merge(frame\_2,frame\_1,on='sl',how='right')

Out[19]:

data\_2 sl data\_1

0 3.0 a 2

1 1.0 a 2

2 3.0 a 6

3 1.0 a 6

4 3.0 a 3

5 1.0 a 3

6 2.0 c 9

7 2.0 c 1

8 7.0 d 0

9 NaN b 2

10 NaN b 5

#右连接以第二个数据框的键列为合并参考键列，观察参考键列中的键是否在两个待合并数据框中有重复，有则在合并时也要重复。

如果在一个数据框重复的键在另一个数据框中也重复出现，那么合并后它重复的次数符合笛卡尔积。

外连接组合了左连接和右连接的效果。

pd.merge(frame\_2,frame\_1,on='sl',how='outer')

Out[20]:

data\_2 sl data\_1

0 3.0 a 2.0

1 3.0 a 6.0

2 3.0 a 3.0

3 1.0 a 2.0

4 1.0 a 6.0

5 1.0 a 3.0

6 2.0 c 9.0

7 2.0 c 1.0

8 7.0 d 0.0

9 6.0 f NaN

10 NaN b 2.0

11 NaN b 5.0

通过上面的一系列例子，我们发现，两个数据框的键列如果不完全一致，或者键不唯一，将是一件很麻烦的事，因此，我建议同学们除非万不得已还是应该给两个待合并的数据框设置带有唯一键的完全相同的键列。

frame\_2=pd.DataFrame({'sl':['a','b','c','d','f'],'data\_2':[3,2,1,7,6]})

frame\_1=pd.DataFrame({'sl':['a','b','c','d','f'],'data\_1':[2,6,9,3,2]})

pd.merge(frame\_2,frame\_1,on='sl')

Out[22]:

data\_2 sl data\_1

0 3 a 2

1 2 b 6

2 1 c 9

3 7 d 3

4 6 f 2

indirect\_Data=pd.merge(frame\_2,frame\_1,on='sl')

indirect\_Data.reindex(columns=['data\_1','data\_2','sl'])

Out[27]:

data\_1 data\_2 sl

0 2 3 a

1 6 2 b

2 9 1 c

3 3 7 d

4 2 6 f

带有双关键字列的数据框的合并

frame\_doppel\_1=pd.DataFrame({'key1':list('wwerrt'),'key2':list('onoodn'),'data1':[1,9,76,9,5,6]})

frame\_doppel\_2=pd.DataFrame({'key1':list('wwt'),'key2':list('onn'),'data1':[1,9,76]})

frame\_doppel\_2

Out[31]:

data1 key1 key2

0 1 w o

1 9 w n

2 76 t n

frame\_doppel\_1

Out[32]:

data1 key1 key2

0 1 w o

1 9 w n

2 76 e o

3 9 r o

4 5 r d

5 6 t n

pd.merge(frame\_doppel\_1,frame\_doppel\_2,on=['key1','key2'],how='outer')

Out[33]:

data1\_x key1 key2 data1\_y

0 1 w o 1.0

1 9 w n 9.0

2 76 e o NaN

3 9 r o NaN

4 5 r d NaN

5 6 t n 76.0

pd.merge(frame\_doppel\_1,frame\_doppel\_2,on=['key1','key2'],how='left')

Out[34]:

data1\_x key1 key2 data1\_y

0 1 w o 1.0

1 9 w n 9.0

2 76 e o NaN

3 9 r o NaN

4 5 r d NaN

5 6 t n 76.0

pd.merge(frame\_doppel\_1,frame\_doppel\_2,on=['key1','key2'],how='right')

Out[35]:

data1\_x key1 key2 data1\_y

0 1 w o 1

1 9 w n 9

2 6 t n 76

pd.merge(frame\_doppel\_1,frame\_doppel\_2,on=['key1','key2'])

Out[36]:

data1\_x key1 key2 data1\_y

0 1 w o 1

1 9 w n 9

2 6 t n 76

#对于有双从键列的数据框，我们先在各自的数据框内把两个键列组成一个元组序列键列，然后按单键列处理即可。（当然，真正的实际原理并非如此）。

frame\_123=pd.DataFrame({'data':[1,2,8,9,0,98,6,56],'key1':list('aacabbed')})

frame\_456=pd.DataFrame({'data\_1':[2,67,1]},index=['a','b','e'])

frame\_456

Out[39]:

data\_1

a 2

b 67

e 1

pd.merge(frame\_123,frame\_456,left\_on='key1',right\_index=True)

Out[40]:

data key1 data\_1

0 1 a 2

1 2 a 2

3 9 a 2

4 0 b 67

5 98 b 67

6 6 e 1

pd.merge(frame\_123,frame\_456,left\_on='key1',right\_index=True,how='outer')

Out[42]:

data key1 data\_1

0 1 a 2.0

1 2 a 2.0

3 9 a 2.0

2 8 c NaN

4 0 b 67.0

5 98 b 67.0

6 6 e 1.0

7 56 d NaN

firstar\_1=pd.DataFrame({'sl\_1':['doctor','doctor','doctor','patient','patient'],'sl\_2':[4,2,4,20,32],'data\_1':np.arange(5)})

secondar\_1=pd.DataFrame(np.arange(12).reshape(4,3),index=[['doctor','doctor','patient','patient'],[4,4,20,20]],columns=['day','hour','second'])

secondar\_1

Out[7]:

day hour second

doctor 4 0 1 2

4 3 4 5

patient 20 6 7 8

20 9 10 11

firstar\_1

Out[8]:

data\_1 sl\_1 sl\_2

0 0 doctor 4

1 1 doctor 2

2 2 doctor 4

3 3 patient 20

4 4 patient 32

combination\_1=pd.merge(firstar\_1,secondar\_1,left\_on=['sl\_1','sl\_2'],right\_index=True)

combination\_1

Out[11]:

data\_1 sl\_1 sl\_2 day hour second

0 0 doctor 4 0 1 2

0 0 doctor 4 3 4 5

2 2 doctor 4 0 1 2

2 2 doctor 4 3 4 5

3 3 patient 20 6 7 8

3 3 patient 20 9 10 11

对于带有重索引的数据框合并时，要先把重索引组成一个个元组对，然后与另一个数据框两个键列组成的元组队进行比较。上面的例子选出两个数据框元组对的交集。多对多相同时符合笛卡尔积

#下面是左连接效果

pd.merge(firstar\_1,secondar\_1,left\_on=['sl\_1','sl\_2'],right\_index=True,how='left')

Out[5]:

data\_1 sl\_1 sl\_2 day hour second

0 0 doctor 4 0.0 1.0 2.0

0 0 doctor 4 3.0 4.0 5.0

1 1 doctor 2 NaN NaN NaN

2 2 doctor 4 0.0 1.0 2.0

2 2 doctor 4 3.0 4.0 5.0

3 3 patient 20 6.0 7.0 8.0

3 3 patient 20 9.0 10.0 11.0

4 4 patient 32 NaN NaN NaN

#下面是左连接效果

pd.merge(firstar\_1,secondar\_1,left\_on=['sl\_1','sl\_2'],right\_index=True,how='right')

Out[6]:

data\_1 sl\_1 sl\_2 day hour second

0 0 doctor 4 0 1 2

2 2 doctor 4 0 1 2

0 0 doctor 4 3 4 5

2 2 doctor 4 3 4 5

3 3 patient 20 6 7 8

3 3 patient 20 9 10 11

#下面是外连接效果

pd.merge(firstar\_1,secondar\_1,left\_on=['sl\_1','sl\_2'],right\_index=True,how='outer')

Out[7]:

data\_1 sl\_1 sl\_2 day hour second

0 0 doctor 4 0.0 1.0 2.0

0 0 doctor 4 3.0 4.0 5.0

2 2 doctor 4 0.0 1.0 2.0

2 2 doctor 4 3.0 4.0 5.0

1 1 doctor 2 NaN NaN NaN

3 3 patient 20 6.0 7.0 8.0

3 3 patient 20 9.0 10.0 11.0

4 4 patient 32 NaN NaN NaN

同时使用双方索引直接合并也是没有问题的，我们看下例：

DataFrame\_1=pd.DataFrame([[2,89,0],[7,9,34],[3.4,5.6,8.9]],index=['a','g','f'],columns=['c1','c2','c3'])

DataFrame\_2=pd.DataFrame([[8,45,6,8],[4,5,3,23],[45,90,6.9,0.7],[4.5,6.8,9.2,7.6]],index=['g','g','a','f'],columns=['c3','c4','c5','c6'])

pd.merge(DataFrame\_1,DataFrame\_2,left\_index=True,right\_index=True,how='outer')

Out[12]:

c1 c2 c3\_x c3\_y c4 c5 c6

a 2.0 89.0 0.0 45.0 90.0 6.9 0.7

f 3.4 5.6 8.9 4.5 6.8 9.2 7.6

g 7.0 9.0 34.0 8.0 45.0 6.0 8.0

g 7.0 9.0 34.0 4.0 5.0 3.0 23.0

下面我们用join函数来实现按索引的合并：

利用join按行索引合并时，两个待合并数据框的列索引不能有重复。

DataFrame\_2\_cor=pd.DataFrame([[8,45,6,8],[4,5,3,23],[45,90,6.9,0.7],[4.5,6.8,9.2,7.6]],index=['g','g','c','b'],columns=['c7','c4','c5','c6'])

DataFrame\_2\_cor

Out[22]:

c7 c4 c5 c6

g 8.0 45.0 6.0 8.0

g 4.0 5.0 3.0 23.0

c 45.0 90.0 6.9 0.7

b 4.5 6.8 9.2 7.6

DataFrame\_1

Out[23]:

c1 c2 c3

a 2.0 89.0 0.0

g 7.0 9.0 34.0

f 3.4 5.6 8.9

DataFrame\_2\_cor.join(DataFrame\_1)

Out[24]:

c7 c4 c5 c6 c1 c2 c3

b 4.5 6.8 9.2 7.6 NaN NaN NaN

c 45.0 90.0 6.9 0.7 NaN NaN NaN

g 8.0 45.0 6.0 8.0 7.0 9.0 34.0

g 4.0 5.0 3.0 23.0 7.0 9.0 34.0

DataFrame\_2\_cor.join(DataFrame\_1,how='inner')

Out[25]:

c7 c4 c5 c6 c1 c2 c3

g 8.0 45.0 6.0 8.0 7.0 9.0 34.0

g 4.0 5.0 3.0 23.0 7.0 9.0 34.0

DataFrame\_2\_cor.join(DataFrame\_1,how='right')

Out[26]:

c7 c4 c5 c6 c1 c2 c3

a NaN NaN NaN NaN 2.0 89.0 0.0

f NaN NaN NaN NaN 3.4 5.6 8.9

g 8.0 45.0 6.0 8.0 7.0 9.0 34.0

g 4.0 5.0 3.0 23.0 7.0 9.0 34.0

DataFrame\_2\_cor.join(DataFrame\_1,how='outer')

Out[28]:

c7 c4 c5 c6 c1 c2 c3

a NaN NaN NaN NaN 2.0 89.0 0.0

b 4.5 6.8 9.2 7.6 NaN NaN NaN

c 45.0 90.0 6.9 0.7 NaN NaN NaN

f NaN NaN NaN NaN 3.4 5.6 8.9

g 8.0 45.0 6.0 8.0 7.0 9.0 34.0

g 4.0 5.0 3.0 23.0 7.0 9.0 34.0

#其运行结果完全按照merge.只不过默认是how=’left’。

轴向连接

在Numpy阶段我们讲过数组的合并，比如我们用concatenate连接两个数组：

arr=np.arange(12).reshape(3,4)

arr

Out[3]:

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11]])

np.concatenate((arr,arr),axis=0)

Out[4]:

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11],

[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11]])

np.concatenate((arr,arr),axis=1)

Out[5]:

array([[ 0, 1, 2, 3, 0, 1, 2, 3],

[ 4, 5, 6, 7, 4, 5, 6, 7],

[ 8, 9, 10, 11, 8, 9, 10, 11]])

接下来我们看一下两个或两个以上的Series是如何合并的。

ps\_1=pd.Series([0,1],index=['a','b'])

ps\_2=pd.Series([6,2],index=['c','d'])

ps\_3=pd.Series([7,0,8,6,0],index=['e','f','g','h','k'])

pd.concat((ps\_1,ps\_2,ps\_3))

Out[10]:

a 0

b 1

c 6

d 2

e 7

f 0

g 8

h 6

k 0

dtype: int64

接下来我们使 ’axis=1’，看看结果如何（注意：通常情况下，对于Series的轴是不允许其为“1”的）：

pd.concat((ps\_1,ps\_2,ps\_3),axis=1)

Out[11]:

0 1 2

a 0.0 NaN NaN

b 1.0 NaN NaN

c NaN 6.0 NaN

d NaN 2.0 NaN

e NaN NaN 7.0

f NaN NaN 0.0

g NaN NaN 8.0

h NaN NaN 6.0

k NaN NaN 0.0

pd.concat((ps\_2,ps\_4))

Out[16]:

c 6

d 2

a 0

b 8

c 6

d 2

dtype: int64

pd.concat((ps\_2,ps\_4),axis=1)

Out[17]:

0 1

a NaN 0

b NaN 8

c 6.0 6

d 2.0 2

pd.concat((ps\_2,ps\_4),axis=1,join='inner')

Out[18]:

0 1

c 6 6

d 2 2

我们还可以给合并后的Series指定索引名：

pd.concat((ps\_2,ps\_4),axis=1,join\_axes=[['a','b','u','v']])

Out[19]:

0 1

a NaN 0.0

b NaN 8.0

u NaN NaN

v NaN NaN

通过key参数可以实现把Series合并成层次化Series：

pd.concat((ps\_1,ps\_2\*3,ps\_3),keys=['ein','drei','fuenf'])

Out[6]:

ein a 0

b 1

drei c 18

d 6

fuenf e 7

f 0

g 8

h 6

k 0

dtype: int64

这里如果我们添加axis=1，结果会如何呢？

ein drei fuenf

a 0.0 NaN NaN

b 1.0 NaN NaN

c NaN 18.0 NaN

d NaN 6.0 NaN

e NaN NaN 7.0

f NaN NaN 0.0

g NaN NaN 8.0

h NaN NaN 6.0

k NaN NaN 0.0

我们发现此时原本Series的外层索引变成列索引。

对于有重复行索引的Series合并后结果会如何呢？

**很不幸，无法运行！**

到此，我们需要做一个用concat合并Series的总结：

总结：

待合并的每个series中不能出现重复的行标签

对于axis=0，Series合并后的效果是行标签的直接组合。

对于axis=1，要视合并模式而定，默认合并模式join=“outer”，按照单个的Seires，一列一列地排下去，join=“inner”模式意味着在outer模式的基础上只保留个Series共有部分，其他全删除。

下面我们把contact应用到数据框

import pandas as pd

import numpy as np

frame\_1=pd.DataFrame([[1000,2000],[279,1123],[721,877]],index=['salary','expenditure','surplus'],columns=['mattias','jennifer'])

frame\_2=pd.DataFrame(np.array([[1000,2000,3000],[279,1123,2000],[721,877,1000]])\*12,index=['salary per annum','expenditure per annum','surplus per annum'],columns=['mattias','jennifer','jocker'])

pd.concat((frame\_1,frame\_2),axis=1,keys=['works','boss'])

Out[2]:

works boss

mattias jennifer mattias jennifer jocker

expenditure 279.0 1123.0 NaN NaN NaN

expenditure per annum NaN NaN 3348.0 13476.0 24000.0

salary 1000.0 2000.0 NaN NaN NaN

salary per annum NaN NaN 12000.0 24000.0 36000.0

surplus 721.0 877.0 NaN NaN NaN

surplus per annum NaN NaN 8652.0 10524.0 12000.0

pd.concat((frame\_1,frame\_2),axis=0,keys=['works','boss'])

Out[3]:

jennifer jocker mattias

works salary 2000 NaN 1000

expenditure 1123 NaN 279

surplus 877 NaN 721

boss salary per annum 24000 36000.0 12000

expenditure per annum 13476 24000.0 3348

surplus per annum 10524 12000.0 8652

视axis=1和 =0 而定，Keys可以给数据框的列或行加上重索引

Keys所实现的功能也可以通过字典来实现：

pd.concat({'works':frame\_1,'boss':frame\_2},axis=1)

Out[6]:

boss works

mattias jennifer jocker mattias jennifer

expenditure NaN NaN NaN 279.0 1123.0

expenditure per annum 3348.0 13476.0 24000.0 NaN NaN

salary NaN NaN NaN 1000.0 2000.0

salary per annum 12000.0 24000.0 36000.0 NaN NaN

surplus NaN NaN NaN 721.0 877.0

surplus per annum 8652.0 10524.0 12000.0 NaN NaN

通过names可以给重索引进行命名：

pd.concat({'works':frame\_1,'boss':frame\_2},axis=1,names=['Identity','Names'])

Out[7]:

Identity boss works

Names mattias jennifer jocker mattias jennifer

expenditure NaN NaN NaN 279.0 1123.0

expenditure per annum 3348.0 13476.0 24000.0 NaN NaN

salary NaN NaN NaN 1000.0 2000.0

salary per annum 12000.0 24000.0 36000.0 NaN NaN

surplus NaN NaN NaN 721.0 877.0

surplus per annum 8652.0 10524.0 12000.0 NaN NaN

通过Ignore\_index=Ture可以把自行设置的行或列索引转变成python自动配置的行或列索引。

pd.concat((frame\_1,frame\_2),axis=0,ignore\_index=True)

Out[11]:

jennifer jocker mattias

0 2000 NaN 1000

1 1123 NaN 279

2 877 NaN 721

3 24000 36000.0 12000

4 13476 24000.0 3348

5 10524 12000.0 8652

pd.concat((frame\_1,frame\_2),axis=1,ignore\_index=True)

Out[12]:

0 1 2 3 4

expenditure 279.0 1123.0 NaN NaN NaN

expenditure per annum NaN NaN 3348.0 13476.0 24000.0

salary 1000.0 2000.0 NaN NaN NaN

salary per annum NaN NaN 12000.0 24000.0 36000.0

surplus 721.0 877.0 NaN NaN NaN

surplus per annum NaN NaN 8652.0 10524.0 12000.0

**如果axis=1，那么join是按0轴操作；同理，axis=1，那么join是按1轴操作。**

pd.concat((frame\_1,frame\_2),axis=1,join='outer')

Out[16]:

mattias jennifer mattias jennifer jocker

expenditure 279.0 1123.0 NaN NaN NaN

expenditure per annum NaN NaN 3348.0 13476.0 24000.0

salary 1000.0 2000.0 NaN NaN NaN

salary per annum NaN NaN 12000.0 24000.0 36000.0

surplus 721.0 877.0 NaN NaN NaN

surplus per annum NaN NaN 8652.0 10524.0 12000.0

pd.concat((frame\_1,frame\_2),axis=0,join='inner')

Out[17]:

mattias jennifer

salary 1000 2000

expenditure 279 1123

surplus 721 877

salary per annum 12000 24000

expenditure per annum 3348 13476

surplus per annum 8652 10524

Concat的参数详解：

|  |  |
| --- | --- |
| objs | 参与连接的pandas对象的列表或者元组或者字典。是唯一必需的参数。 |
| axis | 指明按那个轴进行连接，默认为0 |
| join | 指明连接方式，“inner”或”outer”，默认“outer”。指明其他轴向上的索引是按交集（“inner”）还是并集(“outer”)进行合并。 |
|  |  |

# 五，绘图和可视化

Matplotlib入门，pandas下的绘图函数，画布，画布分割等等

Matplotlib入门

调用Matplotlib API 程序包

1，创建画布

import matplotlib as plt

import matplotlib.pyplot as plt

figure\_1=plt.figure()

(或者详细定义画布

figure\_1=plt.figure(1,(4,6),dpi=400,facecolor='green',edgecolor='yellow',frameon=True)）

figure\_1

Out[43]: <matplotlib.figure.Figure at 0x8ea7987320>

1. 分割画布

aapic\_1=figure\_1.add\_subplot(2,2,1)

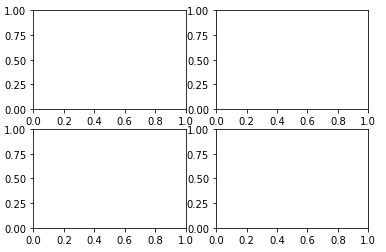
aapic\_2=figure\_1.add\_subplot(2,2,2)

aapic\_3=figure\_1.add\_subplot(2,2,3)

aapic\_4=figure\_1.add\_subplot(2,2,4)

3，show 出你的画布

plt.show()



import matplotlib.pyplot as plt

import numpy as np

figure\_1=plt.figure()

aapic\_1=figure\_1.add\_subplot(2,2,1)

aapic\_2=figure\_1.add\_subplot(2,2,2)

aapic\_3=figure\_1.add\_subplot(2,2,3)

plt.plot(np.random.randn(50).cumsum(),'k--')

Out[7]: [<matplotlib.lines.Line2D at 0xdd16defd0>]

#直接用“plot”绘图，k--代表黑色虚线图

aapic\_2.hist(np.random.randn(100),bins=20,color='k',alpha=0.3)

Out[9]:

(array([ 1., 1., 1., 5., 3., 7., 8., 14., 11., 6., 11.,

6., 13., 5., 2., 3., 0., 1., 0., 2.]),

array([-2.64526298, -2.37172471, -2.09818644, -1.82464817, -1.5511099 ,

-1.27757163, -1.00403336, -0.73049509, -0.45695681, -0.18341854,

0.09011973, 0.363658 , 0.63719627, 0.91073454, 1.18427281,

1.45781108, 1.73134935, 2.00488762, 2.27842589, 2.55196416,

2.82550243]),

<a list of 20 Patch objects>)

aapic\_1.scatter(np.arange(30),np.arange(30)+3\*np.random.randn(30))

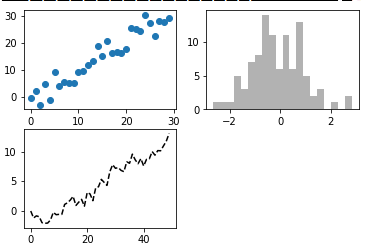
Out[10]: <matplotlib.collections.PathCollection at 0xdd172c390>

#alpha代表透明度，bins代表柱状体个数

plt.show

Out[12]: <function matplotlib.pyplot.show>

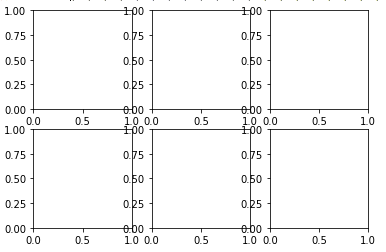
plt.show()



这里还有一个简单且更为方便的画图方法；它可以创建一个新的figure（画布）并返回一个含有以创建的subplot对象的numpy数组。

pic,axes=plt.subplots(2,3)

plt.show(0)



调整Subplot周围的间距：

默认情况下，子图外围是有一定边距的，并且各子图之间上下左右都有一定间距。

间距与子图大小有关，子图像宽，则横向间距小，图像高，则纵向间距小。

除了默认，我们可以选择自己确定间距：

plt.subplots\_adjust(left=None,bottom=None,right=None,top=None,wspace=None,hspace=None)

#wspace和hspace用于控制宽度和高度的百分比，通过调整这两个参数，我们的子图间的上下左右间距会发生变化

例子：

import numpy as np

fig, axes=plt.subplots(2, 2, sharex=True, sharey=True)

for i in range(2):

for j in range(2):

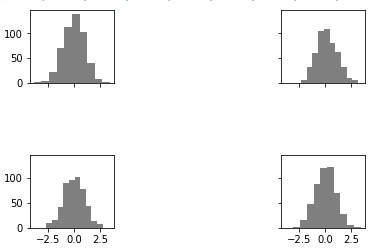
axes[i,j].hist(np.random.randn(500),color='k',alpha=0.5)

plt.subplots\_adjust(wspace=2,hspace=1)

plt.show(0)

<matplotlib.figure.Figure at 0x17becd9908>

<matplotlib.figure.Figure at 0x17befbfda0>



#shareX 和sharey是共X轴和Y轴。调整wspace 盒hspace可以得到不同的子图间距。

for i in range(2):

for j in range(2):

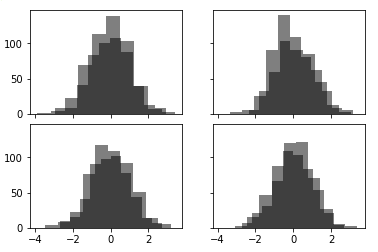
axes[i,j].hist(np.random.randn(500),color='k',alpha=0.5)

plt.subplots\_adjust(wspace=0.2,hspace=0.1)

plt.show(0)

<matplotlib.figure.Figure at 0x17becd9908>

<matplotlib.figure.Figure at 0x17befbfda0>



问：上面图像的X轴和Y轴分别指的是什么？？？？

颜色和线型以及标记

指定图形函数线的颜色，我们可以通过一个指令实现，例：

Pic\_11.plot(X,Y,’g--’) 等价于 ax.plot(x,y,linestyle=’--’,color=’g’)

更多颜色可使用指定其RGB值的形式。对于线性图，我们可以给数据点加上标记(marker)，使人更容易发现那些是数据点。

线型和marker表格：

'-' solid line style

'--' dashed line style

'-.' dash-dot line style

':' dotted line style

'.' point marker

',' pixel marker

'o' circle marker

'v' triangle\_down marker

'^' triangle\_up marker

'<' triangle\_left marker

'>' triangle\_right marker

'1' tri\_down marker

'2' tri\_up marker

'3' tri\_left marker

'4' tri\_right marker

's' square marker

'p' pentagon marker

'\*' star marker

'h' hexagon1 marker

'H' hexagon2 marker

'+' plus marker

'x' x marker

'D' diamond marker

'd' thin\_diamond marker

'|' vline marker

'\_' hline marker

颜色表格：

‘b’ blue

‘g’ green

‘r’ red

‘c’ cyan

‘m’ magenta

‘y’ yellow

‘k’ black

‘w’ white

如果不涉及子图的话，无需先设置画布。可直接画出图形：

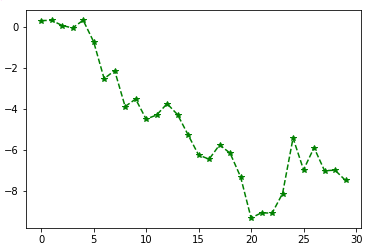
import numpy as np

import matplotlib.pyplot as plt

plt.plot(np.random.randn(30).cumsum(),'g\*--')

Out[3]: [<matplotlib.lines.Line2D at 0xb407f4dd30>]

plt.show()



plt.plot(np.random.randn(30).cumsum(),'g\*--') 等价于

Plt.plot(np.random.randn(30).cumsum(),color=’g’,linestyle=’dashed’,marker=’\*’)

在线性图中，那些非数据点都是根据两个数据点连线插值的，我们可以修改这种插值方式，这里用到drawstyle选项修改：

data=np.array([1,2,3.2,2.3,4.6,7.5,2,3,6.5,7.8,9],dtype=float)

np.unique(data)

Out[9]: array([ 1. , 2. , 2.3, 3. , 3.2, 4.6, 6.5, 7.5, 7.8, 9. ])

data

Out[10]: array([ 1. , 2. , 3.2, 2.3, 4.6, 7.5, 2. , 3. , 6.5, 7.8, 9. ])

data\_uique=np.unique(data)

plt.plot(data\_uique,'o-.',label='line\_point')

Out[28]: [<matplotlib.lines.Line2D at 0xb40947c908>]

plt.plot(data\_uique,'r-',drawstyle='steps-post',label='line')

Out[29]: [<matplotlib.lines.Line2D at 0xb40947cf98>]

plt.legend(loc='best')

Out[30]: <matplotlib.legend.Legend at 0xb4083ca358>

plt.show()



legend函数介绍：

在画一些曲线图时，常常会出现多条曲线同时画在一张图上面，这时候就需要对不同的曲线进行不同的标注，以使读者能够清晰地知道每条曲线代表的含义。当你画很少的几条曲线时，这时画图命令中自动产生的legend能够基本满足你的需要，此时，你不需要做什么；但当你将很多个曲线画在一张图上时，自动产生的legend矩形框往往会覆盖住已经画出来的曲线，很不美观，这时你就需要写专门的代码对legend的位置进行精确的控制，而不能再依靠系统帮你自动控制了。

比如：

plt.legend(loc='upper center', bbox\_to\_anchor=(0.6,0.95),ncol=3,fancybox=True,shadow=True)

Ncol=表示我们的图例（legend）里的线的标识可以排成三列

Loc=标识图例的位置

bbox\_to\_anchor=图例的精确位置，上面bbox\_to\_anchor被赋予的二元组中，第一个数值用于控制legend的左右移动，值越大越向右边移动，第二个数值用于控制legend的上下移动，值越大，越向上移动。

刻度与标签

图像的刻度与标签都是通过一些方法来实现的，这里有几个方法大家在绘图中经常用到：

Xlim（X值范围）、xticks（X轴刻度值）和xticklabels（X轴刻度标签）

其使用方式有以下两种:

1. 调用不带参数值，返回当前参数值，即是现在正用的参数值。
2. 调用时带参数值，使用该参数值。

例：

table\_1=plt.figure()

ax=table\_1.add\_subplot(1,1,1)

ax.plot(np.random.randn(1000).cumsum())

Out[36]: [<matplotlib.lines.Line2D at 0xb409514550>]

ax.plot(np.random.randn(100000).cumsum())

Out[37]: [<matplotlib.lines.Line2D at 0xb40948f780>]

ticks\_1=ax.set\_xticks([0,25000,50000,75000,100000])

scale\_name=ax.set\_xticklabels(['step1','step2','step3','step4','step5'],rotation=45,fontsize=12)

ax.set\_title('Python-03 Practice')

Out[40]: Text(0.5,1,'Python-03 Practice')

ax.set\_xlabel('Steps')

Out[41]: Text(0.5,0,'Steps')

#下面我们添加图例

ax.plot(np.random.randn(100000).cumsum(),color='m',linestyle='-',label='solid')

Out[44]: [<matplotlib.lines.Line2D at 0xb409852748>]

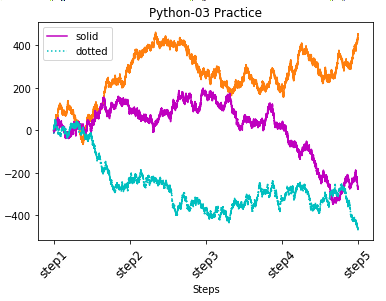
ax.plot(np.random.randn(100000).cumsum(),color='c',linestyle=':',label='dotted')

Out[45]: [<matplotlib.lines.Line2D at 0xb4094f52b0>]

ax.legend(loc='best')#plt.legend也是可以的

Out[46]: <matplotlib.legend.Legend at 0xb409852ac8>

plt.show()



加注解

用text、arrow和annotate等函数进行添加注解，text可以将文本加到图标的指定坐标。

fig,subpic=plt.subplots(1,1) #摆画布

subpic.plot([2,77,90,2.3,4,5,6,45,34,67,35,66,34,23,76],[34,32,35,43,34,23,45,56,44,57,56,33,55,66,54],'b\*-') #画表格

Out[7]: [<matplotlib.lines.Line2D at 0x2eae706be0>]

ticks\_1=subpic.set\_xticks([0,25,50,75,100])

scale\_name=subpic.set\_xticklabels(['jenuary','february','march','april','may'],rotation=40,fontsize=12)

subpic.set\_title('Python-03 Practice\_1',fontsize=16)

Out[11]: Text(0.5,1,'Python-03 Practice\_1')

subpic.set\_xlabel('Weight')

Out[13]: Text(0.5,0,'Weight')

subpic.text(90,35,'key point one',fontsize=10)

Out[15]: Text(90,35,'key point one')

keymenge=[(67,57,'key point two'),(23,66,'key point two')]

for x,y,label in keymenge:

subpic.text(x,y,label,fontsize=12)

subpic.set\_xlim([0,100])

Out[21]: (0, 100)

subpic.set\_ylim([0,100])

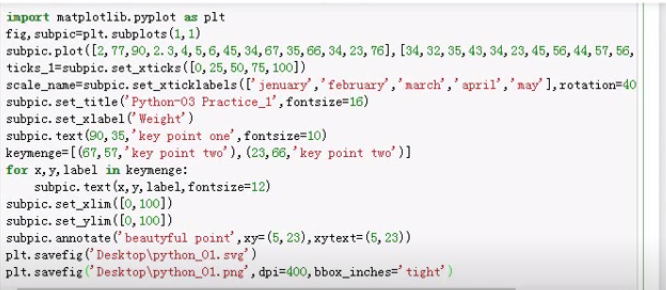
Out[22]: (0, 100)

subpic.annotate('beautyful point',xy=(5,23),xytext=(5,23)) #xy和xytext要一致

Out[18]: Text(5,23,'beautyful point')

plt.savefig('Desktop\python\_01.svg')

plt.savefig('Desktop\python\_01.png',dpi=400,bbox\_inches='tight') #设置分辨率，设为紧凑

plt.show()





Matplotlib.pyplot的画图方式与R语言十分类似，繁琐是他们共同的特点。与Matplotlib.pyplot不同，pandas在作图上不仅方法简单，而且可以完成各种各样的作图工作。

图像的保存

在上一章，我们已经使用了savefig这个函数来存储图像。事实上，Savefig还可以作为对象figure的方法存储画布上的图像，例如：

import matplotlib.pyplot as plt

pic=plt.figure()

pic\_1=pic.add\_subplot(1,1,1)

import numpy as np

pic\_1.plot(np.random.randn(50).cumsum(),'k--')

Out[6]: [<matplotlib.lines.Line2D at 0xa22b23f0b8>]

pic.savefig('Desktop\dong\_123.pdf',dpi=300,bbox\_inches='tight')

Savefig的主要参数如下：

fname: 表示绝对或者相对文件路径的字符串，文件具体格式由后缀来决定，譬如.pdf,.png格式等。

dpi: 图像分辨率，默认100，（每英寸点数）

Facecolor,edgecolor: 背景色，默认为“w”白色

Format: 显示设置文件格式，png，jpeg，pdf等等，但不要与fname里的文件格式发生冲突。

Bbox\_inches，常用值是tight，可剪除图表周围的空白部分。

Pandas作图

线性图

Series 和 DataFrame 都有一个用于生成各类图表的plot方法，默认状态。他只生成线性图。

Series生成线性图，索引（index）直接被绘制成X轴。当然我们也可以关闭使用index绘制X轴。Use\_index=False

X轴的刻度和界限可以用xsticks和xlim选项来进行调节，ysticks和ylim可以调节Y轴

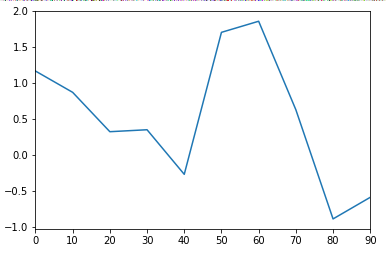
例子

ser=pd.Series(np.random.randn(10).cumsum(),index=np.arange(0,100,10))

ser.plot()

Out[18]: <matplotlib.axes.\_subplots.AxesSubplot at 0xf3e54e0240>

plt.show()



DataFrame的plot方法会在一个子图中为各列数据绘制一条线。自动创建的图例的标签与列索引相同。

例：

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

Frame\_01=pd.DataFrame([[2,4,8,16,32,64],[3,6,12,24,48,96],[5,10,15,20,25,30],[1,3,5,7,9,11],[16,8,4,2,1,0.5]],index=range(0,100,20),columns=['A','B','C','D','E','F'])

pic\_2,subplot\_object=plt.subplots(1,1)

Frame\_01.plot(kind='line',ax=subplot\_object,subplots=False,layout=False,logx=True,xlim=[0,100])

C:\Users\dongfeng\Anaconda3\lib\site-packages\matplotlib\axes\\_base.py:2923: UserWarning: Attempted to set non-positive xlimits for log-scale axis; invalid limits will be ignored.

'Attempted to set non-positive xlimits for log-scale axis; '

Out[3]: <matplotlib.axes.\_subplots.AxesSubplot at 0x5441e71cf8>

plt.show()

Frame\_01

Out[5]:

A B C D E F

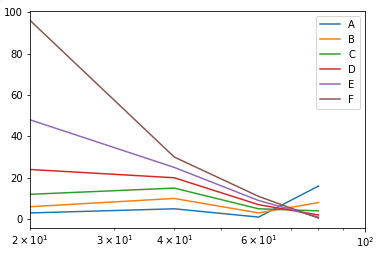
0 2 4 8 16 32 64.0

20 3 6 12 24 48 96.0

40 5 10 15 20 25 30.0

60 1 3 5 7 9 11.0

80 16 8 4 2 1 0.5



Series.plot 方法的常用参数汇总：

|  |  |
| --- | --- |
| label | 用于设置图例的标签 |
| ax | 确定要被绘制的matplotlib subplot对象。如果没有设置，则使用当前matplotlib subplot |
| style | 设置传给matplotlib的风格字符串（'g\*--'） |
| alpha | 图表的填充不透明度（数值为0到1之间的数） |
| kind | 各种图形样式line, bar, barh, kde, density, scatter |
| logy | 在Y轴上使用对数标尺 |
| use\_index | 将对象（Series and DataFrame）的索引用作**刻度标签** |
| rot | 旋转度数（0到360） |
| xticks | 用作X轴刻度的值 |
| yticks | 用作y轴刻度的值 |
| xlim | x的值域 |
| ylim | y的值域 |
| grid | 设置是否显示轴网格线 |

专用于DataFrame的plot的参数

|  |  |
| --- | --- |
| Subplots | 将依据数据框中的每个列绘制的图分别放置到单个的subplot（子画框）里 |
| sharex | Subplots=true时，设定是否共享X的刻度和值域 |
| sharey | Subplots=true时，设定是否共享y的刻度和值域 |
| figsize | 元组，用来表示图像大小（宽，高） |
| title | 设置图像标题 |
| Legend | 设定是否添加一个subplot图例 |
| sort\_columns | 设定是否以字母表中字母先后排列顺序绘制各列。 |

柱状图

柱状图分为水平柱状图和垂直柱状图。当kind=’bar’生成垂直柱状图；kind=’barh’生成水平柱状图。

1. Series生成柱状图

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

figure\_1,get\_information\_of\_pict=plt.subplots(2,1)

datas=pd.Series(np.random.rand(20),index=list('qwertyuiopasdfghjklz'))

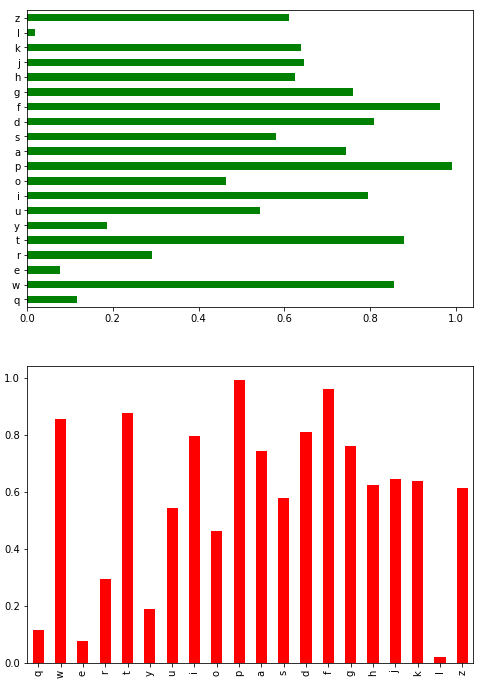
datas.plot(kind='barh',ax=get\_information\_of\_pict[0],figsize=(8,12),color='g')

Out[4]: <matplotlib.axes.\_subplots.AxesSubplot at 0x5be13ea6d8>

datas.plot(kind='bar',ax=get\_information\_of\_pict[1],figsize=(8,12),color='r')

Out[5]: <matplotlib.axes.\_subplots.AxesSubplot at 0x5be18c41d0>

plt.show()



1. 数据框生成柱状图

import numpy as np

import pandas as pd

Framedata\_1=pd.DataFrame(np.arange(16).reshape(4,4)\*\*(1/2)+np.arange(16).reshape(4,4)\*3+6,index=['spring','sommer','autumn','winter'],columns=['Benz','BMW','Porsche','VW'])

Out[11]:

|  | **Benz** | **BMW** | **Porsche** | **VW** |
| --- | --- | --- | --- | --- |
| **spring** | 6.000000 | 10.000000 | 13.414214 | 16.732051 |
| **sommer** | 20.000000 | 23.236068 | 26.449490 | 29.645751 |
| **autumn** | 32.828427 | 36.000000 | 39.162278 | 42.316625 |
| **winter** | 45.464102 | 48.605551 | 51.741657 | 54.872983 |

import numpy as np

import pandas as pd

plt.rcParams['font.sans-serif']=['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

Framedata\_1=pd.DataFrame(np.arange(16).reshape(4,4)\*\*(1/2)+np.arange(16).reshape(4,4)\*3+6,index=['spring','sommer','autumn','winter'],columns=['Benz','BMW','Porsche','VW'])

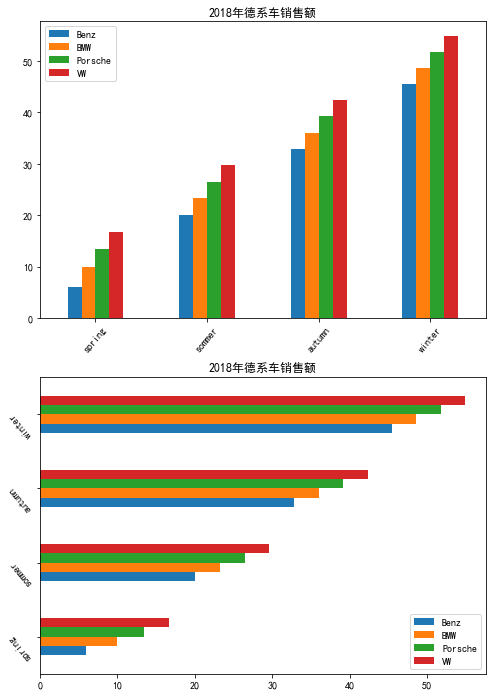
import matplotlib.pyplot as plt

picrange,pic\_inf=plt.subplots(2,1)

Framedata\_1.plot(kind='bar',ax=pic\_inf[0],title='2018年德系车销售额',rot=50,figsize=(8,12))

Framedata\_1.plot(kind='barh',ax=pic\_inf[1],title='2018年德系车销售额',rot=130,figsize=(8,12))

plt.show()



​import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif']=['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

Framedata\_1=pd.DataFrame(np.arange(16).reshape(4,4)\*\*(1/2)+np.arange(16).reshape(4,4)\*3+6,index=['spring','sommer','autumn','winter'],columns=['Benz','BMW','Porsche','VW'])

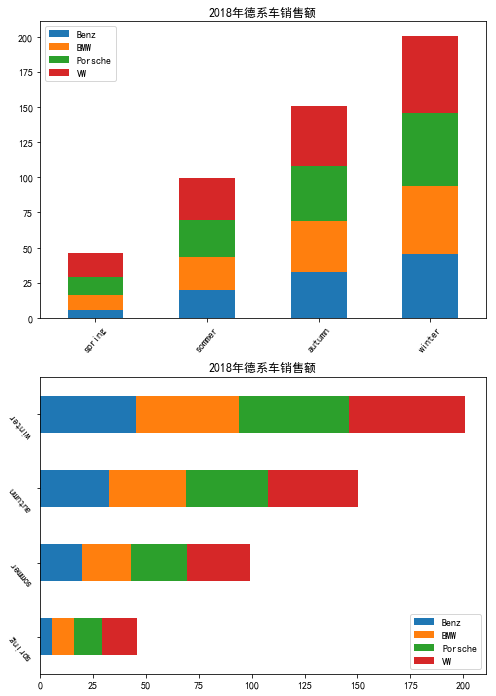
import matplotlib.pyplot as plt

picrange,pic\_inf=plt.subplots(2,1)

Framedata\_1.plot(kind='barh',ax=pic\_inf[1],stacked=True,title='2018年德系车销售额',rot=130,figsize=(8,12))

Framedata\_1.plot(kind='bar',ax=pic\_inf[0],stacked=True,title='2018年德系车销售额',rot=50,figsize=(8,12))

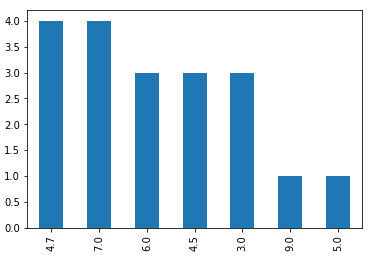
plt.show()



serie\_1=pd.Series([3,4.5,5,3,6,9,4.7,4.7,4.7,4.7,4.5,4.5,6,6,3,7,7,7,7])

serie\_1.value\_counts().plot(kind='bar')

plt.show()



#应用serie\_1.value.counts()在series中寻找数据重复的次数并作为纵坐标，Series中的数据作为横坐标。

直方图与密度图：

首先我们要区分直方图与柱状图。

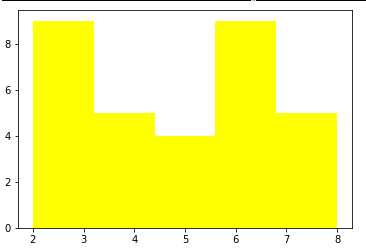
柱状图：

* 柱状图的某一个轴（X或Y轴）可以没有严格的刻度，并且柱的宽度随图形大小，柱的数量等因素的变化而变化，并没有严格的公式来保证，因此没有实际意义，仅仅用来区分类别。通常柱状图是用条形的长度表示各类别对应的实际数据（譬如频数）的大小。
* 柱状图是分开排列
* 主要用于展示分类数据

直方图：

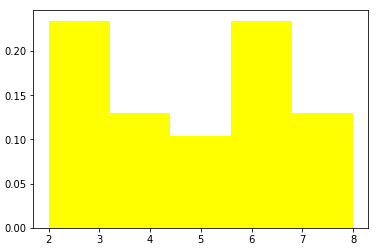
* 直方图通常是用面积表示各组数据大小（例如频数），矩形的高度表示每一组的频数或频率或其他匹配数据，宽度则表示各组的组距，因此其高度与宽度均有意义。
* 由于分组数据具有连续性，直方图的各矩形通常是连续排列。
* 直方图主要用于展示数据型数据。

plt.hist(np.array([1,2,3,4,5,1.2,1.34,1.78,2.1,2.4,2.8,2.9,3.1,3.5,3.7,4.2,4.9,5.6,5.3,5.8,5.9,5,3,6,7,8.5,6.4,7.3,7.8,7.2,5.6,6.6,6.45,6.99,3.45,2.36,5.67,8.13]),bins=5,normed=False,range=(2,8),color='yellow')



plt.hist(np.array([1,2,3,4,5,1.2,1.34,1.78,2.1,2.4,2.8,2.9,3.1,3.5,3.7,4.2,4.9,5.6,5.3,5.8,5.9,5,3,6,7,8.5,6.4,7.3,7.8,7.2,5.6,6.6,6.45,6.99,3.45,2.36,5.67,8.13]),bins=5,normed=True,range=(2,8),color='yellow')

plt.hist(np.array([1,2,3,4,5,1.2,1.34,1.78,2.1,2.4,2.8,2.9,3.1,3.5,3.7,4.2,4.9,5.6,5.3,5.8,5.9,5,3,6,7,8.5,6.4,7.3,7.8,7.2,5.6,6.6,6.45,6.99,3.45,2.36,5.67,8.13]),bins=5,normed=True,range=(2,8),color='yellow')



关于Normed（数据标准化）算法的解释：

sf,axes=plt.subplots()

data= np.array([1,1,2,3,3,3,3,3,4,5.1])

counts= axes.hist(data, normed= True)

counts

Out[9]:

(array([ 0.48780488, 0. , 0.24390244, 0. , 1.2195122 ,

0. , 0. , 0.24390244, 0. , 0.24390244]),

array([ 1. , 1.41, 1.82, 2.23, 2.64, 3.05, 3.46, 3.87, 4.28,

4.69, 5.1 ]),

<a list of 10 Patch objects>)

np.diff(counts[1])

Out[10]: array([ 0.41, 0.41, 0.41, 0.41, 0.41, 0.41, 0.41, 0.41, 0.41, 0.41])

#组距的计算

(counts[0]\*np.diff(counts[1])).sum()

Out[13]: 1.0

#可以看到标准化后的结果和组距相乘然后求和等于1，这才是标准化的实际意义，而不是看标准化的结果是否都必须小于1。

#下面我们确定Python标准化的算法：

sf,axes=plt.subplots()

data= np.array([1,1,2,3,3,3,3,3,4,5.1])

counts\_1= axes.hist(data, normed= False)

counts\_1

Out[14]:

(array([ 2., 0., 1., 0., 5., 0., 0., 1., 0., 1.]),

array([ 1. , 1.41, 1.82, 2.23, 2.64, 3.05, 3.46, 3.87, 4.28,

4.69, 5.1 ]),

<a list of 10 Patch objects>)

probality\_1=counts\_1[0]/np.sum(counts\_1[0]\*np.diff(counts\_1[1]))

probality\_1

Out[16]:

array([ 0.48780488, 0. , 0.24390244, 0. , 1.2195122 ,

0. , 0. , 0.24390244, 0. , 0.24390244])

最终我们得出算法公式为：

**频数矩阵/Sum(频数矩阵\*组距矩阵)**

验证算法：

plt.hist(np.array([1,2,3,4,5,1.2,1.34,1.78,2.1,2.4,2.8,2.9,3.1,3.5,3.7,4.2,4.9,5.6,5.3,5.8,5.9,5,3,6,7,8.5,6.4,7.3,7.8,7.2,5.6,6.6,6.45,6.99,3.45,2.36,5.67,8.13]),bins=5,normed=True,range=(2,8),color='yellow')

Out[17]:

(array([ 0.234375 , 0.13020833, 0.10416667, 0.234375 , 0.13020833]),

array([ 2. , 3.2, 4.4, 5.6, 6.8, 8. ]),

<a list of 5 Patch objects>)

pro1=plt.hist(np.array([1,2,3,4,5,1.2,1.34,1.78,2.1,2.4,2.8,2.9,3.1,3.5,3.7,4.2,4.9,5.6,5.3,5.8,5.9,5,3,6,7,8.5,6.4,7.3,7.8,7.2,5.6,6.6,6.45,6.99,3.45,2.36,5.67,8.13]),bins=5,range=(2,8),color='yellow')

pro1[0]

Out[19]: array([ 9., 5., 4., 9., 5.])

pro\_12=pro1[0]/np.sum(pro1[0]\*np.diff(pro1[1]))

pro\_12

Out[21]: array([ 0.234375 , 0.13020833, 0.10416667, 0.234375 , 0.13020833])

密度图通过kind的KDE关键字来实现的，这里的密度指的是概率密度。它是通过计算观测数据可能产生的概率密度分布而产生的

ser\_1=np.random.normal(0,1,200)

ser\_2=np.random.normal(16,1,200)

pis\_1,sdr\_1=plt.subplots()

A=np.concatenate((ser\_1,ser\_2))

ser\_1=np.random.normal(0,1,200)

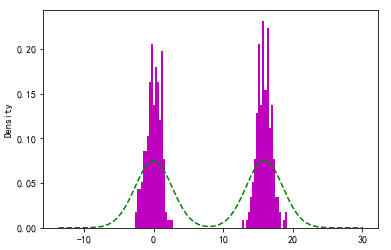
ser\_2=np.random.normal(12,1,200)

value\_1=pd.Series(A)

value\_1.hist(bins=75,color='m',normed=True)

value\_1.plot(kind='kde',style='g--')

plt.show()



**散布图**

散布图必须通过两个数据序列才能绘制而成。

也可以通过一个数据框绘制成散布矩阵,我们先绘制一个复杂的散布矩阵

n [1]: import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

X2=np.random.normal(3,1,1001)

X1=np.arange(1,1002,1)

X3=np.random.randn(1001)

X4=np.random.rand(1001)

X4=np.random.rand(1001)

X5=np.random.beta(2,1,1001)

test\_data=np.column\_stack((np.column\_stack((np.column\_stack((np.column\_stack((X1,X2)),X3)),X4)),X5))

test\_frame=pd.DataFrame(test\_data)

pd.scatter\_matrix(test\_frame,diagonal='kde',color='g',figsize=(10,10))

C:\Users\dongfeng\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1: FutureWarning: pandas.scatter\_matrix is deprecated. Use pandas.plotting.scatter\_matrix instead

"""Entry point for launching an IPython kernel.

Out[7]:

array([[<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B2D89E8>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x0000003422A85240>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x0000003422A9E2B0>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342AA85860>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342AAD87F0>],

[<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342AAD8828>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B33FFD0>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B3860F0>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B39BDA0>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B419518>],

[<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B452898>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B489828>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B4C3828>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B4D4F98>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B522F98>],

[<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B5684A8>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B5A23C8>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B53A7F0>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B603A58>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B63AF28>],

[<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B67D4A8>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B6A32B0>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342C6B2518>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342C6EA908>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342C722828>]], dtype=object)

plt.show()

# 

散布矩阵数据索引图如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X1X1密度图 | X2X1 | X3X1 | X4X1 | X5X1 |
| X1X2 | X2X2密度图 | X3X2 | X4X2 | X5X2 |
| X1X3 | X2X3 | X3X3密度图 | X4X3 | X5X3 |
| X1X4 | X2X4 | X3X4 | X4X4密度图 | X5X4 |
| X1X5 | X2X5 | X3X5 | X4X5 | X5X5密度图 |

接下来我们通过简单的散布图来验证我们的散布矩阵图：

X1X1密度图：

# 

X3X1 和 X1X3：

fig\_123,axec=plt.subplots(1,2,dpi=140)

for j in range(1):

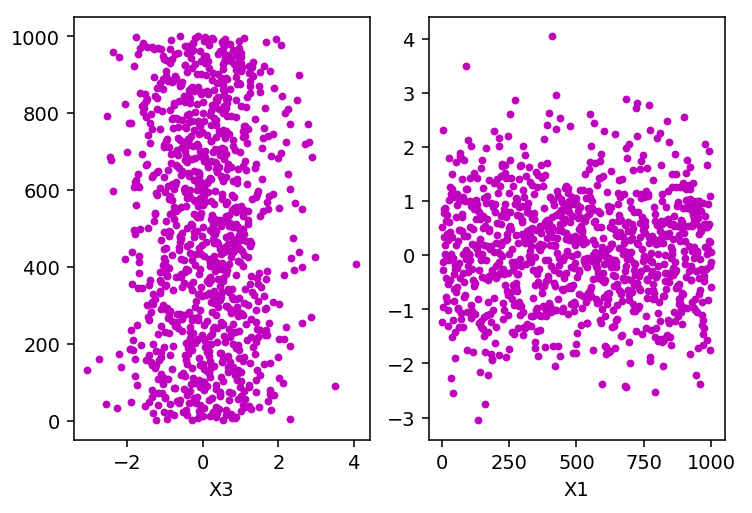
axec[j].scatter(test\_frame[2],test\_frame[0],color='m',marker='.')

axec[j].set\_xlabel('X3')

axec[j+1].scatter(test\_frame[0],test\_frame[2],color='m',marker='.')

axec[j+1].set\_xlabel('X1')

plt.show()



# 数据地图（basemap）

In [**1**]: from mpl\_toolkits.basemap import Basemap

   ...: import matplotlib.pyplot as plt

   ...: import pandas as pd

   ...: import numpy as np

   ...: from matplotlib import cm

# 绘制基础地图，选择绘制的区域，因为是绘制中国地图，故选取如下经纬度，lat\_0和lon\_0是地图中心的维度和经度。

china\_map=Basemap(projection='stere',lat\_0=34,lon\_0=115,llcrnrlat=28 ,urcrnrlat=42,llcrnrlon=105,urcrnrlon=129,rsphere=(2000,2000),resolution='l',area\_thresh=350)

参数解释：

Projection- 地图投影方式，常用的有'ortho'、'merc'、'stere'和’cyl’,’cass’、’lcc’等。

llcrnrlat- 所需地图域左下角的纬度（度）。

urcrnrlat- 所需地图域的右上角的纬度（度）。

llcrnrlon- 所需地图域左下角的经度（度）。

urcrnrlon- 所需地图域的右上角的经度（度）。

china\_map.drawmapboundary() # 绘制边界

china\_map.fillcontinents(color='y',lake\_color='b',zorder=0) # 填充大陆，发现填充之后无法显示散点图，应该是被覆盖了,因此取消

china\_map.drawstates(color='m') # 绘制省

china\_map.drawcoastlines(color='r') # 绘制海岸线，必须绘制，即使是不靠海也需绘制

china\_map.drawcountries(color='r')

#linewidth 设置线宽

#linestyle 设置线形。默认为 solid，可以是 dash，也可以是 matplotlib 其它选项。

#color 设置颜色。默认为 black(k)。

#antialiased 抗锯齿选项。默认为 True.

china\_map.drawrivers(linewidth=0.5, linestyle='solid', color='#1E90FF')

#zorder 设置图层位置。默认情况下由 Basemap 设置.

china\_map.drawlsmask(ocean\_color='#1E90FF')

china\_map.drawcountries(color='y') # 绘制国家，不太适合此例子，但需保留

#china\_map.bluemarble()

parallels = np.arange(28.,42.,2.)

china\_map.drawparallels(parallels,labels=[1,0,0,0],fontsize=10) # 绘制纬线

meridians = np.arange(105.,129.,3.)

china\_map.drawmeridians(meridians,labels=[0,0,0,1],fontsize=10) # 绘制经线

data\_1=pd.read\_csv(r'Desktop\dizhenshuju.csv',header=None,delimiter=',').values

lat=data\_1[:,2];lon=data\_1[:,1];Seismic\_grade=data\_1[:,3]

Seismic\_grade\_float=np.array(Seismic\_grade,dtype=np.float64)

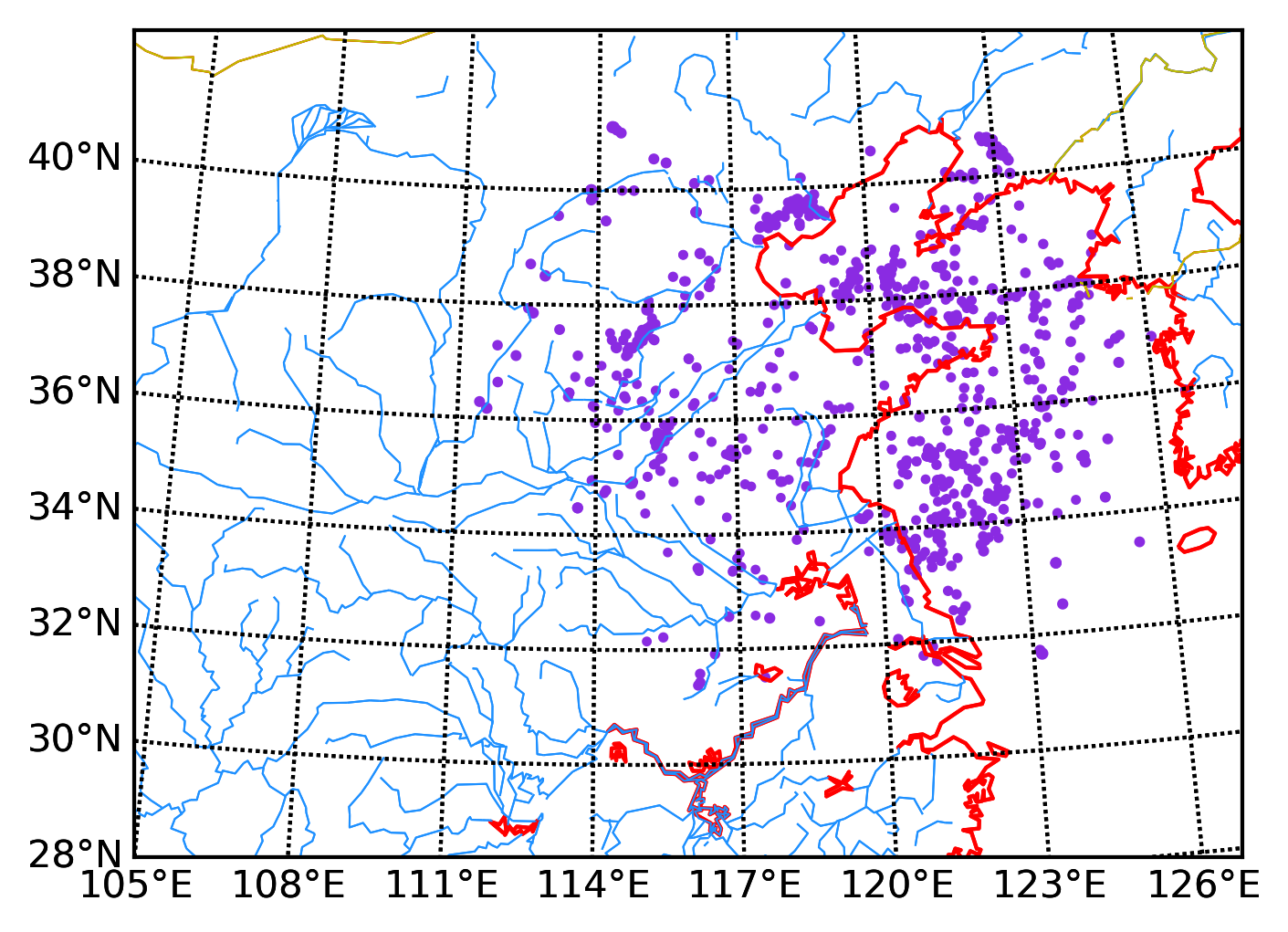
class\_1=(Seismic\_grade\_float/np.max(Seismic\_grade\_float))\*5

x,y = china\_map(lon,lat)#地图上的精度维度匹配参数x，y

plt.scatter(x,y,s=class\_1,cmap=cm.hsv,c='#8A2BE2') # 使用matplotlib的散点图绘制函数

plt.savefig('Desktop\dong\_1214.pdf',dpi=300,bbox\_inches='tight')

plt.show()



# 六，数据的分组与聚合分析法

数据分析方法介绍，最根本的分析法是分组分析与聚合

# 数据分析建模和预测

线性回归，多项式回国，指数回归。互联网独立平台数据建模与分析（比较典型的例子）

# 其它数据分析软件介绍(若时间允许）

Access，Excel透视表与它的分析软件，Power Pivot， visual basic application