Think Stats： 用Python数据分析

前言

本书介绍了跟数据分析相关的工具。参照平时我分析数据时的流程，组织本书内容：

* 数据导入与清理：无论数据格式怎样，通常都需要花时间和精力来读入数据，清洗和改变格式，check that everything made it through the translation process intact.
* 单变量探研：通常， 我会先集中研究单个变量，确定这些变量背后的意义，观察这些值的分布，最终确定合适的汇总性统计。
* 逐对探研：为了探测出变量之间可能的关联，我会观察相关表格和单点图，计算相关性和线性拟合。
* 多元分析： 如果发现明显的关联关系，我会使用多回归来添加控制变量，并审查更复杂的关联关系。
* 评估和假设性测试：在做统计结果报告时，回答这些问题是很重要的：效果有多大？如果再重新测量的话，会有多大的可复再性？那些明显的效果改变的可能性有多大？
* 数据可视化：在数据探研过程中，数据可视化可以很好地帮助我们找到可能的关联和效果。最终，结果为了经受住考验，数据可视化是一个有效的方式来跟人交流结果。

本书采用了程序模式（而非数学模式），相对于数学模式，程序模式有如下好处：

* 用Python代码，而不是数据公式来表达想法。Python代码更有可读性，同时，由于Python可直接执行，读者可下载执行并修改它。
* 每一章都包含了练习题，读者可下载下来体验并借以加固对学习的理解。一般来说，你写代码时，你就是在通过代码来表达理解；而调试代码时， 你又是在纠正理解。
* 一些练习题包含了试验来测试统计学特性。例如，你可以通过生成随机样本并计算他们和的方式来探研CLT。最终的可视化结果演示了为什么CLT是可行的，什么时候又失效。
* 用数学方式不容易理解一些想法时，使用模拟会更容易理解。例如，我们可以运行随机模拟来近似地估算出P值，这样可以加固对P值的理解。
* 由于本书使用了Python这个通用的编程语言，读者可以从几乎任何数据源导入数据。这些数据不再局限于那些已经清理且为特定统计工具而格式化好的数据。

本书以工程方式展开。在我的课上，学生们会参与到一个学期长度的工程中来，这个工程需要学生来提出统计方面的问题，寻找相关数据来解决这个问题，解决过程中使用学到的技术。

为了演示我解决统计分析问题的思路，贯穿本书所有章节的有一个案例分析。这个案例使用了下面两个来源的数据：

* 美国家庭增长调查（NSFG），这个调查是由美国疾病控制和预防中心实施的，旨在收集这些数据：美国家庭生活、婚姻与离婚、怀孕、不孕情况、避孕情况和男女性健康。
* BRFSS（）， 这个调查由执行， 旨在。

其它的例子使用的数据有IRS、美国统计局和Boston Marathon。

本书的第二版包含了第一版中的所有章节，其中不少章节是重写的，新写了些关于回归、时间序列分析、生存分析和分析方法。第一版本时没有使用Pandas,SciPy或StatsModels，这样所有的这些材料都是新的。

* 1. 怎么写本书

人们在写一新教科书时， 通常会先读一大堆旧的书。 这样的结果，大多数书都有差不多的内容，又是几乎一样的顺序。

我并没有这样做。实际上， 在写本书时，我几乎没有用到打印版的材料，有这样的几个原因：

* 我的目标是用新的方式来处理这些材料，所以也没多大的兴趣在已经存在的方式上。
* 由于我想通过一种免费的方式发布此书，我也想确保此书中任何部分都不受版权的限制。
* 本书的不少读者并不能方便地从图书馆借到书，所以我尽力使用网上可得到的材料作为引用。
* 一些旧媒体的支持者认为过多地使用电子材料是懒惰和不可靠的。在懒惰方面，他们是对的，不过在第二个问题上，我认为是错误的，所以我想测试下我的理论。

本书写作过程中， 我使用最多的是Wikipedia.一般来说，Wiki上读到的统计材料是很好的（尽管写作的过程中我还是做了些修改）。本书里也包含进了不少Wiki页面引用，我也鼓励大家来看这些Wiki材料.大多数情况下，Wiki页面的内容填补了我描述的空白。词汇和标记也保持着跟Wiki的一致，除非我有足够的理由来修改。其它一些有用的材料是Wolfram MatchWorld和Reddit统计论坛里的内容。

* 1. 使用代码

本书使用的代码和数据都可以在<https://github.com/AllenDowney/ThinkStats2>里找到，Git是一个版本控制软件，使用它可以跟踪项目中的文件。Git控制的文件集合称为仓库。GitHub是提供了Git仓库和方便的操作界面。

我仓库的GitHub主页里提供了多种方式来使用代码：

* 通过Fork方式来创建一个Copy。
* 直接Clone我的的仓库，不过，这样你不能再修改代码后上传。
* 不使用Git情况下，可直接从GitHub页面里下载Zip包。

所有的代码都是用Python2和Python3写的，不需要再做转换。

写本书的过程中，我使用了Anaconda这个工具（包含了pandas、NumPy、SciPy、StatsModels和matplotlib）。Anaconda可方便安装，默认情况下，安装时，是用户级的，而非系统级的，这样你不再需要管理员权限。它都支持Python2和Python3。

本书中使用的Python包，虽然是通用的，但并没包含在标准的Python安装文件里，在一些环境中也不好安装。如果你安装这些包有问题时， 我强烈推荐你使用Anaconda或其它一些已经包含这些工具的安装包。

在clone了Git仓库后（或解压了Zip包后），你将看到ThinkStats2这个文件夹下有一个叫nsfg.py的文件。运行此文件，它会读一个数据文件，执行些测试，打印些消息，如“All tests passed”。如果执行中有什么错，很有可能意味着还有些Python包需要安装。

大多数练习是使用Python脚本的，不过有些也使用IPython notebook。如果以前没有使用过IPython的话，可参考<http://ipython.org/ipython-doc/stable/notebook/notebook.html>文档。

写本书时一个假定：读者对Python和OOP已经熟悉，对pandas、NumPy和SciPy没有要求。如果你对这三个已经很熟悉了，可以跳过其中的一些章节。

数学方面，我也假定读者知道些基本的数学常识，如对数和求和。期间我会提到一些微积分的概念，不过你不需要微积分有了解。

如果你没有学习过统计学，我认为此书是一个不错的开始。

1. 探研数据分析

本书的命题：武装了可用方法的数据可以解答问题并帮助在不确定情况下做决定。

作为一个例子，列举一个案例分析。妻子和我在等待第一个孩子时，听说一般第一个孩子会延迟一段时间出生，从统计上来说，对否？这个案例就是这个问题引起的。

如果搜索下的话，会发现不少这方面的讨论。一些人认为这个说法是对的，一些人认为不靠谱，也有一些有完全相反的说法：第一个孩子一般会早生。

一些讨论中，人们会提供些数据来证明他们的观点。我发现下面这些比较典型：

“我两个朋友最近刚生了第一个孩子，他们都是有两周的延迟。”

“我第一个孩子比预产期晚了两周，现在我认为第二个孩子要早出生两周了！！”

“我不认为这个说法是对的，因为我姐姐是妈妈的第一个孩子，她还早出生了，不少表兄妹都这样。”

这样的报道称为轶事证明，由于他们都是基于些未经公开发布和个人的数据。一般来说， 这样的轶事也无可厚非，所以我也不想指责有这样说法的人。

不过， 我们还是想更有说服力和可靠的答案。这些轶事证明大多是失败的，因为：

* 观察的数据量太小:对第一个孩子来说，如果怀孕周期长的话，the difference is probably small compared to natural variation。
* 选择性偏见： 参与讨论的人们可能会因为他们的第一个孩子出生而感兴趣。这样情况下，选择的数据会把结果带偏。
* 确认的偏见：相信这个说法的人会贡献证据来证明，反对的人会提供反对的证据。
* 不准确：轶事通常是个人讲述的， 经常会记错、表述错误并不准确地重复等。

这样，我们怎么才能做的更好呢？

* 1. 统计学方式

为了纠正轶事的不限制，我们使用统计工具，包括：

* 数据收集： 我们将使用全美国范围收集的数据，这些数据当时就是为了有效地统计美国人口的。
* 描述性统计：我们将生成统计结果， 它将简明地给出数据摘要，使用（evaluate）不同的方法来可视化数据。
* 探研式数据分析:我们将找寻模式，区别和其它的特性（针对我们感兴趣的问题）。同时，我们将校验数据的一致性并找到不足。
* 估算：我们将使用样式数据来评估出些整体特征。
* 假设测试：在发现有明显效果的地方（如两组有不同时）,我们会评估这个效果是不是偶然造成的。

经过上面步骤后，我们可以得出一个可信且极有可能是正确的结论。

* 1. NSFG

从1973年开始，美国的CDC开始执行NSFG调查，旨在收集“这些方面的信息：家庭生活、结婚和离婚、怀孕及不孕情况、避孕使用、男女性健康”。收到的数据为了制定健康服务部署和健康教育计划。

我们将使用这些数据来调查是否第一个孩子出生会晚些和其它问题。为了有效地理解这些数据，我们需要理解此调查的构思。

NSFG是一个跨Sectional的调研，也即在时间的某个点上本调研取一个组织的快照。跟跨Sectional相对的是纵向的调研，这种调研方式会在一定时间内重复地观察一个组织。

NSFG已经进行了7次了，它每一次执行称为一个Cycle。我们将使用Cycle6的数据，这个数据是从2002年1月到2003年3月。

调研的目标是针对人口情况得出统计结论，NSFG调研的目标人口是美国15到44年龄段的人。理想情况下，调查应该收集人口中每一个成员的数据，不过这个很不现实。相反，我们从一个称为样本的人口子集里收集数据。参与调查的人口称为调查对象。

通常情况下，跨Sectional的调研应该是representative，意味着：每一个人口对象都有平等的机会参与。这个目标也很达成，不过， 还是要尽可能地接近。

NSFG并不是representative，相反它估计设计成oversampled（采样过密）。调研的设计者募集了三组（西班牙裔、非洲裔和十几岁的青年人），比例要其比整个美国人口比例高， 以便能推导出足够的统计结论。

当然，采样过于密集的一个问题是，用这个统计结果可能不容易得到更为普遍的结论。后续我们会再讨论这个不足。

在处理这些数据时， 很有必要熟悉下codebook，这个文档描述了NSFG调研的设计，调研的问题和受访的反馈。相当材料请见http://www.cdc.gov/nchs/nsfg/nsfg\_cycle6.htm

* 1. 导入数据

代码和使用的数据可以从<https://github.com/AllenDowney/ThinkStats2>得到。

下载后， 可以找到这个文件ThinkStats2/code/nsfg.py。运行一下，它会读一个数据文件，执行一些测试，并打印下这样的Message：“All tests passed”。

我们看下，都做了什么。NSFG Cysle6的怀孕情况数据在一个叫2002FemPreg.dat.gz的文件中。它是一个gzip压缩的文本文件，有固定宽度的列。文件中每行是一个记录，它含有怀孕情况的数据。

文件的格式记录在2002FemPreg.dct中， 这是一个Stata字典文件。Stata是一个统计软件系统，本语义环境中的字典文件是一个列表，每一项都有变量名字、变量类型和索引（这个索引会告诉我们从哪一行中找到每一个变量）

例如，下面是从2002FemPreg.dct里摘出的几行：

Infile dictionary {

\_column(1) str12 caseid %12s “RESPONDENT ID NUMBER”

\_column(13) byte pregordr %2f “PREGNANCY ORDER (NUMBER)”

}

这个字典里描述了两个变量：caseid是一个12字符的字符串，表示受调查者ID;pregorder是一个单字节的整形，表示受调者怀孕的序数。

下载下来的代码里有一个thinkstats2.py文件， 它是一个Python模块，里面包含了本书中使用的类和函数，如读Stata字典和NSFG数据文件的函数。下面是这些函数在nsfg.py中使用：

def ReadFemPreg(dct\_file=’2002FemPreg.dct’, dat\_file=’2002FemPreg.dat.gz’):

dct = thinkstats2.ReadStataDct(dct\_file)

df = dct.ReadFixedWidth(dat\_file, compression=’gzip’)

CleanFemPreg(df)

Return df

ReadStataDct接收字典文件的名字返回一个dct，这个dct是一个FixedWidthVariables对象，此对象中含有字典文件中的信息。dct提供一个ReadFixedWidth，用它读取数据文件。

* 1. DataFrames

ReadFixedWidth的结果是DataFrame，它是pandas这个数据处理包中的基础数据结构。为每一条记录，DataFrame包含了一行，在当前的怀孕数据中，每一变量有一列。

除了数据外，DataFrame也含有变量名字和变量类型，也还有存取和修改数据的方法。

如果打印df，将看到一个截断的行和列，DataFrame的形状，这是一个有13593行数据（记录），及244列（变量）

>>> import nsfg

>>> df = nsfg.ReadFemPreg()

>>> df

…

[13593 rows x 244 columns]

DataFrame太大不方便显示，所以结果是截断的。最后一行显示了行数和列数。

Columns属性以Unicode字符串的形式返回列名：

>>> df.columns

Index([u'caseid', u'pregordr', u'howpreg\_n', u'howpreg\_p', ... ])

返回的是一个Index，这也是pandas包的数据结构。随后，我们将仔细研究这个Index，不过现在我们将它视为list：

>>> df.columns[1]

‘pregordr’

可以用列名作为key来访问DataFrame列：

>>> pregordr = df[‘pregordr’]

>>>type(pregordr)

<class 'pandas.core.series.Series'>

结果是一个Series，这又是另一个pandas数据结构。Series像Python的list,不过多了一些额外的特征。打印一个Series，可以看到它的索引及对应的值：

0 1  
1 2  
2 1  
3 2  
...  
13590 3  
13591 4  
13592 5  
Name: pregordr, Length: 13593, dtype: int64

本例中， 索引是从0开始到13592的序列，不过通常来说这些索引可以是任意的可排放类型。元素也可以是整型，不过也可以是任意类型。

最后一行显示了变量名字，Series的长度和数据类型;int64是NumPy提供的数据类型。如果运行在32位的机器上，将看到int32.

可以像Python中的list那样访问Series中的元素：

>>> pregordr[0]

1

>>>pregordr[2:5]

1. 1
2. 2
3. 3

Name: pregordr, dtype:int64

Index操作的结果是一个int64，slice的结果还是一个Series对象。

也可以通过点的方式来访问DataFrame列:

>>> pregordr=df.pregordr

这样的访问方式只对列名是一个有效的Python标识时才可行，也就是说列名须以字母开头，不能含有空格，等。

1.5 变量

我们已经看到NSFG中的两个变量，caseid和pregordr，也看到NSFG中已经有244个变量。本书中，我们将用到下面的变量：

* caseid是受调者的整型ID。
* prglngth是怀孕的周数。
* outcome码代表了怀孕的结果，代码1代表活着出生。
* pregordr是怀孕的序号，如，受调者头胎时，用1表示，二胎时用2表示。
* birthord是活着出生的序号，第一个孩子时， 用1表示。如果怀孕的最终结果是孩子没能活着出生，这一域是空。
* birthwgt\_lb和birthwgt\_oz代表着孩子出生时体重中的磅数和盎司数。
* agepreg是母亲在怀孕最后时的年龄。
* finalwgt是受调者的统计权重。这是一个浮点数值，代表着美国对应受调者所代表人口的比例。

如果仔细读codebooke的话， 你会发现这里有什么变量的重编码的，也就是意味着它们不是最初收集到的原始部分，而是再计算处理过的。

如，活着出生的prglgnth值等于wksgest(gestation的周数)（如wksgest存在的话），否则将是一个用mosgest\*4.33的再计算值（怀孕的月数乘以每个月里的平均周数）。

重编码的情况也经常基于这样的逻辑：检查数据的一致性和准确性。通常情况下，如果数据存在的话， 就使用它，除非有一个很迫切的理由自己再处理数据。

1.6 格式转换

导入数据后，经常需要检查是否有错误，处理特殊数值，对不同格式的数据转换，进行计算。这些过程我们称为为数据清理。

nsfg.py包含了一个CleanFemPreg函数，它将清理下计划使用的数据。

def CleanFemPreg(df):

df.agepreg /= 100.0

na\_vals = [97,98,99]

df.birthwgt\_lb.replace(na\_vals, np.nan, inplace=True)

df.birthwgt\_oz.replace(na\_vals, np.nan, inplace=True)

df[‘totalwgt\_lb’] = df.birghwgt\_bl + df.birthwgt\_oz / 16.0

agepreg是孩子出生时母亲的年龄。数据文件中，agepreg是用centiyears的数值表达的。这里的第一行是用agepreg除以100，这样得到一个浮点数值的年数。

birthwgt\_lb和birghwgt\_oz包含了婴儿出生时的体重数据。另外，他使用了些特殊的编码：

97 NOT ASCERTAINED

98 REFUSED

99 DON’t KNOW

这些特殊编码的值是很危险的，如果没能很妥当地处理，会生产意想不到的结果，如一个99磅重的婴儿。replace函数w会把这些值用np.nan替换掉，np.nan是一个特殊的浮点数值，表示“不是一个数字”。inplace将告诉replace方法修改已存在的Series而不是新建一个。

IEEE规定，所有涉及到nan的操作将返回nan：

>>> import numpy as pn

>>> np.nan / 100.0

nan

这样，用nan来计算一般来有正确的结果，pandas的绝大多数函数也将恰当地处理nan变量。不过处理缺失的数据将是一个recurring issue。

CleanFemPreg的最后一行创建了一个新的列totalwgt\_lb，它代表着实际的体重值。

一个重要的注意点：当你往DataFrame里添加一列时，你必须使用字典语法，如：

#CORRECT

Df[‘totalwgt\_lb’] = df.birthwgt\_lb +df.birthwgt\_oz /16.0

不能使用点法，如：

#WRONG！

Df.totalwgt\_lb = df.birthwgt\_lb + df.birthwgt\_oz /16.0

点法会往DataFrame对象里添加一个属性，这个属性将不会按新列来处理。

1.7 检验