Python 数据科学项目实战

[美] 伦纳德·阿佩尔辛(Leonard Apeltsin) 著

殷海英 史跃东 译

清華大學出版社

北 京

北京市版权局著作权合同登记号 图字：01-2022-1218

Leonard Apeltsin
Data Science Bookcamp Five Python Projects
EISBN: 9781617296253

图书在版编目(CIP)数据

Python数据科学项目实战 / (美) 伦纳德·阿佩尔辛(Leonard Apeltsin) 著；殷海英，史跃东译. —北京：清华大学出版社，2022.10
书名原文：Data Science Bookcamp, Five Python Projects
ISBN 978-7-302-61814-0

Ⅰ. ①P… Ⅱ. ①伦… ②殷… ③史… Ⅲ. ①软件工具—程序设计 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字(2022)第 167975 号

责任编辑：王 军
装帧设计：孔祥峰
责任校对：成凤进
责任印制：宋 林

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座 邮 编：100084
社 总 机：010-83470000 邮 购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：艺通印刷（天津）有限公司
经 销：全国新华书店
开 本：170mm×240mm 印 张：37 字 数：945 千字
版 次：2022 年 11 月第 1 版 印 次：2022 年 11 月第 1 次印刷
定 价：139.00 元

产品编号：095564-01

译者序

在介绍本书之前，我要和大家分享一个小故事。我所在的学院去年秋季迎来两位新老师。主任让我们这些“过来人”为这两位新加入的老师分享一些教学经验和教训。我记得当时我与他们分享的是“案例的力量”。那可能是我从事教学工作的第三年，每个学习的期末，学生都会在评价系统中为本学期所选的课程评分，其中主要是对教学人员的评分。很感谢我的学生们，我的评分平时基本都是院里最高，但就在那个学期，我所讲授的“Python 编程基础”课程，在评分表中居然出现 5 个 strong disagree(5 分制中的 0 分)，我的玻璃心碎了一地。一开始，我觉得会不会因为学生的恶作剧，或者单纯是因为不喜欢我本人呢？也许吧，一百多位学生，有 5 个 0 分评价也还算正常，同事和我的老师也让我不要在意。但我仔细看了这几张匿名调查表的内容，这些学生是很认真地在每个项目下写了注释，他们对我不满的主要原因是，过于注重理论知识的讲解，缺少案例，他们完全不知道学完这个课程后，要如何将所学的知识应用到实际工作和研究中。这确实是我所讲授课程存在的巨大缺陷，我真的很想当面感谢这些学生，但因为调查表是匿名的，无法与他们联络。在来年的课程中，针对每个知识点，我设计了小型场景实验，并在每章后面，加入了与本章相关的综合实验。在来年的期末调查表中，这门课获得 98%的满意率。我想告诉那两位新同事的是：理论基础固然重要，但在学生的学习过程中，一定要加入精心设计的实验和示例，这样才能让学生更好地了解并运用所学知识。

Python 现在可以说是运用最广泛的编程语言之一，使用 Python 的人不只局限在计算机相关专业的从业者，很多来自金融领域、医疗领域以及其他我们无法想象的领域的人，每天都在使用 Python 处理各种数据、使用机器学习进行预测以及完成各种有趣的工作。长久以来，很多使用 Python 的人都存在一个困扰，他们知道 Python 的具体技术，但无法与实际工作结合起来，虽然有很多“Python 案例”书籍和博客，但内容过于零散，不够系统和完整。如果你也存在这样的困扰，那么这本《Python 数据科学项目实战》可以说就是为解决你的困扰而写的。本书不仅提供丰富的案例，而且通过非常严谨和完整的结构，介绍使用 Python 进行数据科学工作所需的知识。本书通过 5 个案例进行讲解。第一个案例介绍如何在 Python 中处理与概率论相关的内容，这是很多其他书籍所没有的，如果你去研读概率论相关的专业书籍，那一定是非常痛苦的，毕竟你没打算成为概率论方面的专家。本书在第一个案例中介绍的内容，可以满足你在工作中所涉及的绝大多数与概率论相关的需求。第二个案例介绍如何在 Python 中使用统计学相关的知识，对数据进行处理，这里通过非常巧妙的方式，讲解很多枯燥的统计学知识，让你对这些内容有更深刻的了解。第三个案例介绍如何使用 Python 处理与地理信息相关的数据，并介绍如何使用 scikit-learn 库实现高效聚类。第四个案例介绍如何使用当今比较流行的自然语言处理技术处理实际问题。第五个案例介绍网络理论和监督学习相关的内容。通过这 5 个案例的学习，我相信，一定能让你对 Python 及数据科学有更深刻的认识，并将这

些技术应用到你的工作和学习中。

最后，我想衷心感谢清华大学出版社的王军老师，感谢他帮我出版了多本机器学习、人工智能以及高性能计算的书籍，感谢他为我提供一种新的与大家分享知识的方式。

——殷海英

作于加利福尼亚州埃尔赛贡多市

作 者 简 介

Leonard Apeltsin 是 Anomaly 的数据科学主管。他的团队应用高级分析来发现医疗保健欺诈、浪费和滥用的情况。在加盟 Anomaly 之前，Leonard 领导了 Primer AI 的机器学习开发工作；Primer AI 是一家专门从事自然语言处理的初创公司。作为创始成员，Leonard 帮助 Primer AI 团队从 4 名员工发展到近 100 名员工。在进入创业公司之前，Leonard 在学术界工作，他发现了遗传相关疾病的隐藏模式。他的发现发表在《科学》和《自然》杂志的附属期刊上。Leonard 拥有卡内基梅隆大学的生物学和计算机科学学士学位，以及加州大学旧金山分校的生物信息学博士学位。

序　言

又有一位有前途的候选人在数据科学面试中失败了，我开始想知道为什么。那是 2018 年，我在初创公司努力扩大数据科学团队。我面试了几十个看似合格的候选人，结果他们都没有被录取。最近被拒的申请者是名校经济学博士。不久前，应聘者在完成为期 10 周的训练营后转入了数据科学领域。我请应聘者讨论一个与我们公司非常相关的分析问题。他们立即提出了一种不适用于这种情况的流行算法。当我试图讨论算法的不兼容性时，应聘者不知所措。他们不知道算法实际上是如何工作的，也不知道在什么情况下使用它。这些细节在训练营中没有教给他们。

在被拒绝的应聘者离开后，我开始反思自己的数据科学教育。早在 2006 年，数据科学还不是一个令人垂涎的职业选择，DS 训练营还不存在。那时，我是一个贫穷的研究生，在生活成本昂贵的旧金山努力支付房租。当时需要我研究分析数百万种与疾病遗传相关的联系。我意识到我的技能可以转移到其他分析领域，因此我的数据科学咨询公司诞生了。

在我的研究生导师不知情的情况下，我开始随机向湾区公司寻求分析工作。自由职业收入帮助我支付了账单，因此我不能对我处理的数据驱动的任务过于挑剔。我报名参加各种数据科学任务，从简单的统计分析到复杂的预测建模。有时我会发现自己被一个看似棘手的数据问题压得喘不过气来，但最后还是坚持了下来。在奋斗历程中，我认识到各种分析技术的细微差别，以及如何采用最合理的方式将它们结合起来，从而提供最佳解决方案。更重要的是，我了解了常见技术如何失败，以及如何避免这些失败，从而提供有影响力的结果。随着我的技能不断增长，我的数据科学事业开始蓬勃发展。最终，我成为该领域的佼佼者。

在为期 10 周的训练营中，通过死记硬背，我能取得同样的成功吗？可能不会。许多训练营优先考虑独立算法的研究，而不是更有凝聚力的解决问题技能。此外，对算法优势的炒作往往会强调其弱点。因此，学生有时无法处理现实世界中的数据科学问题。这也是我撰写本书的原因。

我决定将自己的数据科学学习过程分享给大家，让读者了解一组逐步具有挑战性的分析问题。此外，我选择为你提供有效处理这些问题所需的工具和技术。我的目标是从整体上帮助你培养分析问题和解决问题的能力。这样当应征那些初级数据科学职位时，你将更有可能得到工作。

前　言

开放式解决问题的能力对于数据科学职业至关重要。遗憾的是，这些能力不能仅通过阅读获得。要成为问题解决者，你必须坚持不懈地解决难题。考虑到这一点，我围绕案例研究构思了本书：以现实世界情况为模型的开放式问题。案例研究范围从在线广告分析到使用新闻数据跟踪疾病暴发。完成这些案例研究后，你将可以开始你的数据科学事业。

本书的目标读者

本书的目标读者是具有基本的分析基础且有兴趣转行到数据科学职业的人。我的设想是，他也许是一位想探索更多的分析机会的经济学大四学生，或者是一位已经毕业的化学专业学生正在寻找以数据为中心的职业道路。又或者，读者可能是一位成功的前端 Web 开发人员，其数学背景非常有限，但也想尝试数据科学。本书的潜在读者都没有上过数据科学课程，这让他们在进行各种数据分析时感到力不从心。本书的目的是消除这些技能缺陷。

本书的读者需要了解 Python 编程的最基本知识。自学 Python 入门知识的水平应该能足以探索本书中的练习。至于数学知识，读者只需要理解基本的高中三角函数即可。

本书组织结构

本书包含 5 个难度由浅入深的案例研究。每个案例研究都以你需要解决的问题的详细陈述开始。问题陈述之后是用 2~5 章介绍解决问题所需的数据科学技能。这些技能部分涵盖了 Python 基础库以及数学和算法技术。每个案例研究的最后一章都描述了问题的解决方案。

案例研究 1 与基本概率论有关。

- 第 1 章讨论如何使用简单的 Python 计算概率。
- 第 2 章介绍概率分布的概念。该章还介绍 Matplotlib 可视化库，通过它可以对分布进行可视化。
- 第 3 章讨论如何使用随机模拟来估计概率。该章引入 NumPy 数值计算库，从而促进有效的模拟执行。
- 第 4 章包含案例研究的解决方案。

案例研究 2 从概率扩展到统计。

- 第 5 章介绍中心性和离散性的简单统计测量。该章还介绍 SciPy 科学计算库，其中包含一个有用的统计模块。

- 第 6 章深入探讨可用于进行统计预测的中心极限定理。
- 第 7 章讨论各种统计推断技术，这些技术可用于将有趣的数据模式与随机噪声区分开。此外，该章说明了错误使用推理的危险以及如何更好地避免这些危险发生。
- 第 8 章介绍 Pandas 库，可用于在统计分析之前对表格数据进行预处理。
- 第 9 章包含案例研究的解决方案。

案例研究 3 侧重于介绍地理数据的无监督聚类。

- 第 10 章介绍如何使用中心性度量将数据聚类到组中。该章还引入 scikit-learn 库以促进高效聚类。
- 第 11 章侧重于介绍地理数据提取和可视化。在该章中，使用 GeoNamesCache 库从文本中进行提取并使用 Cartopy 地图绘制库实现可视化。
- 第 12 章包含案例研究的解决方案。

案例研究 4 侧重于介绍使用大规模数值计算的自然语言处理。

- 第 13 章说明如何使用矩阵乘法有效地计算文本之间的相似度。NumPy 的内置矩阵优化被广泛用于此目的。
- 第 14 章展示如何利用降维来进行更有效的矩阵分析。该章结合 scikit-learn 的降维方法讨论数学理论。
- 第 15 章将自然语言处理技术应用于超大文本数据集。该章讨论如何更好地探索和聚类这类文本数据。
- 第 16 章展示如何使用 Beautiful Soup HTML 解析库从在线数据中提取文本。
- 第 17 章包含案例研究的解决方案。

案例研究 5 侧重于对网络理论和监督机器学习的讨论。

- 第 18 章结合 NetworkX 图分析库介绍基本网络理论。
- 第 19 章展示如何利用网络流在网络数据中寻找聚类。该章将概率模拟和矩阵乘法用于实现有效的聚类。
- 第 20 章介绍一种基于网络理论的简单监督机器学习算法。该章还使用 scikit-learn 说明常见的机器学习评估技术。
- 第 21 章讨论其他机器学习技术，这些技术依赖内存高效的线性分类器。
- 第 22 章深入探讨之前介绍的监督学习方法的缺陷。随后使用非线性决策树分类器来规避这些缺陷。
- 第 23 章包含案例研究的解决方案。

本书的每一章都建立在前几章中介绍的算法和库的基础上。因此，我们鼓励你从头到尾阅读本书，以减少困惑。但如果你已经熟悉书中的某些内容，可直接跳过它们。最后，强烈建议你在阅读解决方案之前自己解决每个案例研究的问题。独立解决每一个问题将使本书的价值最大化。

另外，读者可扫描封底二维码，来下载源代码。

关于本书封面

本书封面中的插图标题是 Habitante du Tyrol，意思是“蒂罗尔州居民”。插图取自 Jacques Grasset de Saint-Sauveur(1757—1810)创作的各国服饰集，标题为 *Costumes de Différents Pays*，于 1797 年在法国出版。每一幅插图都是手工精细绘制和着色的。Jacques Grasset de Saint-Sauveur 的藏品种类繁多，生动展现了 200 年前世界各地的城镇和地区在文化上的巨大差异。人们彼此隔绝，说着不同的方言和语言。无论是在街上还是在乡村，只要看一眼他们的衣着，就很容易知道他们住在哪里、从事什么行业以及在社会中处于什么地位。

从那时起，我们的穿着方式发生了改变。地域的多样性在当时是如此丰富，但现在已经逐渐消失。现在很难区分不同地域的居民，更不用说区分不同的城镇、地区或国家了。也许我们用文化多样性换来了更多样化的个人生活——当然也换来了更多样化和快节奏的科技生活。

在一个计算机书籍同质化严重的时代，Manning 出版社将图书封面以两个世纪前丰富多样的地区生活为基础，通过 Jacques Grasset de Saint-Sauveur 的图片使其重焕活力，颂扬了计算机行业的创造性和主动性。

致　谢

撰写本书非常辛苦。我绝对不可能独自完成。幸运的是，我的家人和朋友在这段艰难的旅程中给予了大力支持。首先，我要感谢我的母亲 Irina Apeltsin。在那些困难的日子里，当我面前的任务似乎无法克服时，她就是我的动力。此外，我还要感谢我的祖母 Vera Fisher，在我为本书翻阅材料时，她的务实建议使我走上了正轨。

此外，我要感谢我儿时的朋友 Vadim Stolnik。Vadim 是一位出色的平面设计师，他帮助我完成了书中的无数插图。另外，我要感谢我的朋友兼同事 Emmanuel Yera，在我最初的写作过程中，他一直支持我。此外，我不得不提一下我亲爱的舞伴 Alexandria Law，她在我感到苦闷时，让我精神振奋，还帮助我挑选了本书的封面。

接下来，我要感谢 Manning 的编辑 Elesha Hyde。在过去的 3 年中，你孜孜不倦地工作，以确保我向读者提供真正有价值的东西。我将永远感谢你的耐心、乐观和对质量的不懈承诺。你推动我成为一个更好的作者，我的读者最终会从这些努力中受益。此外，我要感谢技术开发编辑 Arthur Zubarov 和技术校对员 Rafaella Ventaglio。你们的投入帮助我制作了一本更好、更有条理的书籍。我还要感谢项目编辑 Deirdre Hiam、文案编辑 Tiffany Taylor、校对员 Katie Tennant，以及 Manning 的其他所有参与过本书工作的人。

致所有审稿人：Adam Scheller、Adriaan Beiertz、Alan Bogusiewicz、Amaresh Rajasekharan、Ayon Roy、Bill Mitchell、Bob Quintus、David Jacobs、Diego Casella、Duncan McRae、Elias Rangel、Frank L Quintana、Grzegorz Bernas、Jason Hales、JeanFrançois Morin、Jeff Smith、Jim Amrhein、Joe Justesen、John Kasiewicz、Maxim Kupfer、Michael Johnson、Michał Ambroziewicz、Raffaella Ventaglio、Ravi Sajnani、Robert Diana、Simone Sguazza、Sriram Macharla 和 Stuart Woodward。谢谢你们，你们的建议使本书变得更出色。

目　录

案例研究 1

在纸牌游戏中寻找制胜策略

问题描述

想赢一点钱吗？让我们用纸牌游戏下一个小赌注。在你面前是一副洗好的纸牌。所有 52 张牌都面朝下。一半的牌是红色的，另一半的牌是黑色的。我将一张一张地翻转纸牌。如果我翻转的最后一张牌是红色的，你将赢得 1 美元；否则，你将损失 1 美元。

这里有一个规则：你可以随时要求我停止游戏。一旦你说“停止”，我会翻转下一张牌并结束游戏。下一张牌将作为最后一张牌。如果它是红色的，你将赢得 1 美元，如图 CS1-1 所示。

我们可以根据你的要求多次玩这个游戏。所有纸牌每次都会重新洗牌。每一轮结束后，都会进行赌资的结算。你赢得这场游戏的最佳方法是什么？

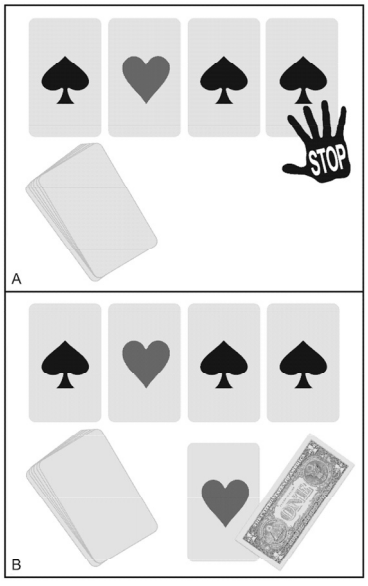


图 CS1-1　翻牌游戏。首先洗牌，然后我来翻纸牌。图 A 为我刚刚翻开了第 4 张牌，你让我停下来；图 B 为我翻转第 5 张也就是最后一张牌。最后一张牌是红色的，你赢了 1 美元

概述

为解决这个问题，我们需要知道如何执行以下操作。

- 通过样本空间分析，计算可观察事件的概率。
- 在区间值范围内绘制事件的概率。
- 使用 Python 模拟随机过程，如抛硬币和洗牌。
- 使用置信区间分析评估我们从模拟中得出的决策的置信度。

第1章

使用 Python 计算概率

本章主要内容

- 概率论基础
- 计算单个观察的概率
- 计算一系列观察的概率

生活中很少有事情是确定的；大多数事情都是偶然的。每当我们购买彩票或投资股市时，都希望有一些特定的结果，但这种结果永远无法保证。随机性已渗透到我们的日常生活中。幸运的是，这种随机性仍然可以被降低并被控制。我们知道，一些不可预测的事件比其他事件的发生概率更低，并且某些决策比其他风险更大的决策具有更少的不确定性。例如，开车上班比骑摩托车更安全；将你的部分储蓄投资在退休账户上比将其全部押在 21 点游戏上更安全。通常我们可以很好地对这些不确定的事情做出适当的决策，因为即使是最不可预测的系统，仍然会表现出一些可预测的行为。概率论中已对这些行为作了大量研究。概率论是数学的一个内在复杂的分支。但你可以在不了解数学基础知识的情况下，理解该理论的各个方面。事实上，不需要使用数学方程就可以利用 Python 解决困难的概率问题。这种无方程的概率方法要求我们对数学家所说的样本空间有一个基本了解。

1.1 样本空间分析：一种用于测量结果不确定性的无方程方法

某些行为具有可衡量的结果。样本空间是一个行为的所有可能结果的集合。让我们以简单的抛硬币为例，硬币将落在正面或反面。因此，抛硬币会产生两种可衡量的结果：正面或反面。通过将这些结果存储在 Python 集合中，可以创建一个掷硬币的样本空间(如代码清单 1-1 所示)。

代码清单 1-1　创建抛硬币的样本空间

```
sample_space = {'Heads', 'Tails'}
```

将元素存储在大括号中会创建一个 Python 集合。Python 集合是唯一的无序元素的集合

假设随机选择 sample_space 的一个元素。这个元素多少次会是正面(即 Heads)？样本空间包含两个可能的元素，每个元素在集合中占据相等比例的空间。因此，我们知道出现正面的可能性是50%。这个可能性被正式定义为结果的概率。sample_space 中的所有结果具有相同的概率，等于1/len(sample_space)，如代码清单 1-2 所示。

代码清单 1-2 计算出现正面的概率

```
probability_heads = 1 / len(sample_space)
print(f'Probability of choosing heads is {probability_heads}')

Probability of choosing heads is 0.5
```

选择正面的概率等于 0.5。这与抛硬币的动作直接相关。假设硬币是标准无偏差的，这意味着硬币出现正面和反面的概率相等。因此，抛硬币在概念上等同于从 sample_space 中随机选择一个元素。硬币正面朝上的概率为 0.5，反面朝上的概率也等于 0.5。

我们已经为两个可测量的结果分配了概率，但还可以提出其他问题。硬币落在正面或反面的概率是多少？或者更奇特的是，硬币在空中一直旋转，既不正面也不反面着陆的概率是多少？为找到准确的答案，需要定义事件的概念。事件是 sample_space 中满足某些事件条件的那些元素的子集(如图 1-1 所示)。事件条件是一个简单的布尔函数，其输入是单个 sample_space 元素。只有当元素满足条件约束时，该函数才返回 True。

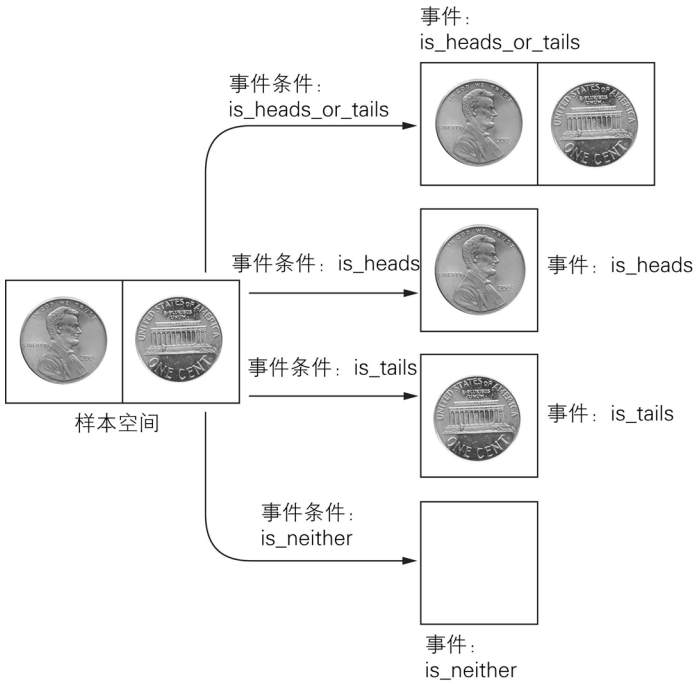


图 1-1 应用于样本空间的 4 个事件条件。样本空间包含两个结果：正面和反面。箭头代表事件条件。每个事件条件都是一个返回“是或否”的函数。每个函数过滤掉那些不满足要求的结果。剩余的结果形成一个事件。每个事件对应样本空间中满足特定条件的子集。4 种事件可能为：正面、反面、正面或反面，以及既不是正面也不是反面

让我们定义两个事件条件：一个是硬币落在正面或反面，另一个是硬币既不是正面也不是反面(如代码清单 1-3 所示)。

代码清单 1-3　定义事件条件

```
def is_heads_or_tails(outcome): return outcome in {'Heads', 'Tails'}
def is_neither(outcome): return not is_heads_or_tails(outcome)
```

此外，为完整起见，我们定义另外两个基本事件的事件条件(如代码清单 1-4 所示)。

代码清单 1-4　定义额外的事件条件

```
def is_heads(outcome): return outcome == 'Heads'
def is_tails(outcome): return outcome == 'Tails'
```

我们可以将事件条件传递给通用的 get_matching_event 函数，代码清单 1-5 中对该函数进行了定义。它的输入是一个事件条件和一个通用样本空间。该函数遍历通用样本空间并返回 event_condition(outcome)为 True 的结果集。

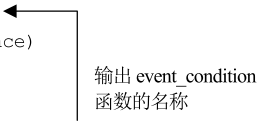
代码清单 1-5　定义事件检测函数

```
def get_matching_event(event_condition, sample_space):
    return set([outcome for outcome in sample_space
                if event_condition(outcome)])
```

我们在 4 个事件条件上执行 get_matching_event 函数，然后输出 4 个提取的事件(如代码清单 1-6 所示)。

代码清单 1-6　使用事件条件检测事件

```
event_conditions = [is_heads_or_tails, is_heads, is_tails, is_neither]

for event_condition in event_conditions:
    print(f"Event Condition: {event_condition.__name__}")
    event = get_matching_event(event_condition, sample_space)
    print(f'Event: {event}\n')

Event Condition: is_heads_or_tails
Event: {'Tails', 'Heads'}

Event Condition: is_heads
Event: {'Heads'}

Event Condition: is_tails
Event: {'Tails'}

Event Condition: is_neither
Event: set()
```



我们已经成功地从 sample_space 中提取 4 个事件。每个事件发生的概率是多少？前面证明了均匀材质硬币的单个元素结果的概率是 1/len(sample_space)。这个属性可以泛化为包括多元素事件。

事件的概率等于 len(event)/len(sample_space)，但只有在所有结果都以相同的可能性发生的情况下才适用。换句话说，一枚均匀材质硬币出现多元素事件的概率等于事件大小除以样本空间大小。现在我们使用事件大小计算 4 个事件的概率，如代码清单 1-7 所示。

代码清单 1-7 计算事件概率

```
def compute_probability(event_condition, generic_sample_space):
    event = get_matching_event(event_condition, generic_sample_space)
    return len(event) / len(generic_sample_space)
```

compute_probability 函数提取与输入事件条件相关联的事件，以计算其概率

概率等于事件大小除以样本空间大小

```
for event_condition in event_conditions:
    prob = compute_probability(event_condition, sample_space)
    name = event_condition.__name__
    print(f"Probability of event arising from '{name}' is {prob}")

Probability of event arising from 'is_heads_or_tails' is 1.0
Probability of event arising from 'is_heads' is 0.5
Probability of event arising from 'is_tails' is 0.5
Probability of event arising from 'is_neither' is 0.0
```

执行上述代码，输出不同的事件概率，最小的是 0.0，最大的是 1.0。这些值代表概率的上限和下限。概率不可能低于 0.0 或高于 1.0。

分析有偏硬币

我们已计算了一枚无偏硬币的概率。如果硬币是有偏差的，会发生什么？例如，假设一枚硬币正面朝上的概率是反面的 4 倍。如何计算未以相同方式加权的结果可能性？可以构造一个由 Python 字典表示的加权样本空间(如代码清单 1-8 所示)。每个结果都被视为一个键，其值映射到相关的权重。在本示例中，Heads 的权重是 Tails 的 4 倍，因此将 Tails 映射到 1，Heads 映射到 4。

代码清单 1-8 表示加权样本空间

```
weighted_sample_space = {'Heads': 4, 'Tails': 1}
```

新样本空间存储在字典中。这允许我们将样本空间的大小重新定义为所有字典权重的总和。在 weighted_sample_space 中，该总和将等于 5(如代码清单 1-9 所示)。

代码清单 1-9 检查加权样本空间大小

```
sample_space_size = sum(weighted_sample_space.values())
assert sample_space_size == 5
```

我们可以用类似的方式重新定义事件大小。每个事件都是一组结果，这些结果映射到权重。对权重进行求和将得到事件大小。因此，满足 is_heads_or_tails 事件条件的事件的大小也是 5(如代码清单 1-10 所示)。

代码清单 1-10　检查加权事件大小

```
event = get_matching_event(is_heads_or_tails, weighted_sample_space)
event_size = sum(weighted_sample_space[outcome] for outcome in event)
assert event_size == 5
```

提示：这个函数迭代输入的样本空间中的每个结果。因此，将字典作为输入将得到预期结果。这是因为 Python 迭代字典键，而不是许多其他流行的编程语言中的键值对

通过对样本空间大小和事件大小的广义定义，可以创建一个 compute_event_probability 函数(如代码清单 1-11 所示)。该函数将 generic_sample_space 变量作为输入，该变量可以是加权字典或未加权的集合。

代码清单 1-11　定义广义事件概率函数

```
def compute_event_probability(event_condition, generic_sample_space):
    event = get_matching_event(event_condition, generic_sample_space)
    if type(generic_sample_space) == type(set()):
        return len(event) / len(generic_sample_space)

    event_size = sum(generic_sample_space[outcome]
                     for outcome in event)
    return event_size / sum(generic_sample_space.values())
```

检查 generic_event_space 是否是一个集合

我们现在可以输出有偏硬币的所有事件概率，而无须重新定义 4 个事件条件函数(如代码清单 1-12 所示)。

代码清单 1-12　计算加权事件概率

```
for event_condition in event_conditions:
    prob = compute_event_probability(event_condition, weighted_sample_space)
    name = event_condition.__name__
    print(f"Probability of event arising from '{name}' is {prob}")

Probability of event arising from 'is_heads' is 0.8
Probability of event arising from 'is_tails' is 0.2
Probability of event arising from 'is_heads_or_tails' is 1.0
Probability of event arising from 'is_neither' is 0.0
```

仅用几行代码，就构建了一个用于解决许多概率问题的工具。我们可将此工具应用于比抛硬币更复杂的问题。

1.2　计算非平凡概率

我们将使用 compute_event_probability 解决几个示例问题。

1.2.1　问题 1：分析一个有 4 个孩子的家庭

假设一个家庭有 4 个孩子。正好有两个孩子是男孩的概率是多少？我们将假设每个孩子是男孩

或女孩的可能性均等。因此，可以构建一个未加权的样本空间(如代码清单 1-13 所示)，其中每个结果代表一种 4 个孩子性别的可能序列，如图 1-2 所示。

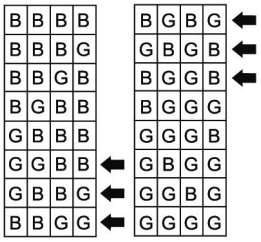


图 1-2 4 个兄弟姐妹的样本空间。样本空间中的每一行包含 16 种可能结果中的 1 种。每个结果都代表 4 个孩子性别的独特组合。每个孩子的性别用字母表示：B 代表男孩，G 代表女孩。两个男孩的结果用箭头标记。有 6 个这样的箭头；因此，两个男孩的概率等于 6/16

代码清单 1-13 计算孩子性别的样本空间

```
possible_children = ['Boy', 'Girl']
sample_space = set()
for child1 in possible_children:
    for child2 in possible_children:
        for child3 in possible_children:
            for child4 in possible_children:
                outcome = (child1, child2, child3, child4)
                sample_space.add(outcome)
```

4 个孩子的每个可能序列都由一个四元素元组表示

我们通过 4 个嵌套的 for 循环来探索 4 个孩子的出生顺序，但这不是常规的编码方式，效率较低。我们实际可以使用 Python 的内置 itertools.product 函数更轻松地生成样本空间，该函数返回输入列表中所有元素的所有组合(如代码清单 1-14 所示)。这里将 possible_children 列表的 4 个实例输入 itertools.product 中。然后 product 函数遍历列表的所有 4 个实例，计算列表元素的所有组合。最终输出样本空间。

代码清单 1-14 使用 product 函数计算样本空间

运算符解包存储在列表中的多个参数，然后将这些参数传递给指定的函数。因此，调用 product((4* [possible_children]))等效于调用 product(possible_children,possible_children,possible_children,possible_ children)

```
from itertools import product
all_combinations = product(*(4 * [possible_children]))
assert set(all_combinations) == sample_space
```

注意，运行这行代码后，all_combinations 将为空。这是因为 product 返回一个 Python 迭代器，它只能迭代一次。对我们来说，这不是问题。我们将更有效地计算样本空间，在以后的代码中将不会使用 all_combinations

我们可以通过执行 set(product(possible_children,repeat=4))使代码更高效(如代码清单 1-15 所示)。通常，运行 product(possible_children,repeat=n)会返回 n 个孩子的所有可能组合的可迭代对象。

代码清单 1-15 将 repeat 传递到 product

```
sample_space_efficient = set(product(possible_children, repeat=4))
assert sample_space == sample_space_efficient
```

现在计算家庭中有两个男孩的 sample_space 比例(如代码清单 1-16 所示)。我们定义一个 has_two_boys 事件条件，然后将该条件传递给 compute_event_probability。

代码清单 1-16　计算两个男孩的概率

```
def has_two_boys(outcome): return len([child for child in outcome
                                       if child == 'Boy']) == 2

prob = compute_event_probability(has_two_boys, sample_space)
print(f"Probability of 2 boys is {prob}")

Probability of 2 boys is 0.375
```

在一个有 4 个孩子的家庭中，恰好有两个男孩出生的概率是 0.375。这意味着，我们预计 37.5%的有 4 个孩子的家庭中的男孩和女孩数量相同。当然，实际观察到的有两个男孩的家庭的百分比会因随机因素而有所不同。

1.2.2　问题 2：分析掷骰子游戏

假设有一个标准的六面骰子，其各面编号为 1~6。骰子被掷 6 次。这 6 次投掷的结果加起来是 21 的概率是多少？

首先定义随机投掷可以得到的可能值，这个值的范围是 1~6 的整数(如代码清单 1-17 所示)。

代码清单 1-17　定义六面骰子的所有可能投掷值

```
possible_rolls = list(range(1, 7))
print(possible_rolls)

[1, 2, 3, 4, 5, 6]
```

接下来，使用 product 函数为 6 次连续的投掷创建样本空间(如代码清单 1-18 所示)。

代码清单 1-18　创建 6 次连续投掷的样本空间

```
sample_space = set(product(possible_rolls, repeat=6))
```

最后，定义一个 has_sum_of_21 事件条件，随后将其传递给 compute_event_probability(如代码清单 1-19 所示)。

代码清单 1-19　计算掷骰子总和的概率

```
def has_sum_of_21(outcome): return sum(outcome) == 21

prob = compute_event_probability(has_sum_of_21, sample_space)
print(f"6 rolls sum to 21 with a probability of {prob}")

6 rolls sum to 21 with a probability of 0.09284979423868313
```

从理论上讲，投掷一个骰子 6 次相当于同时投掷 6 个骰子

通过结果可以看到有差不多 9%的可能性使 6 次投掷的总和为 21。注意，我们的分析可以通过

lambda 表达式更简洁地编码(如代码清单 1-20 所示)。lambda 表达式是不需要名称的单行匿名函数。在本书中，我们会使用 lambda 表达式将短函数传递给其他函数。

代码清单 1-20 使用 lambda 表达式计算概率

```
prob = compute_event_probability(lambda x: sum(x) == 21, sample_space)
assert prob == compute_event_probability(has_sum_of_21, sample_space)
```

lambda 表达式允许我们在一行代码中定义短函数。lambda x:在功能上等同于 func(x):。因此，lambda x: sum(x)==21 在功能上等同于 has_sum_of_21

1.2.3 问题 3：使用加权样本空间计算掷骰概率

我们刚刚计算了 6 次掷骰子总和为 21 的可能性。现在，使用加权样本空间重新计算这个概率，需要将未加权样本空间集合转换为一个加权样本空间字典。这将要求我们确定所有可能的骰子值总和。然后必须计算每个总和在所有可能的骰子组合中出现的次数。这些组合已存储在我们计算的 sample_space 集合中。通过将骰子总和映射到它们的出现次数，将得到一个 weighted_sample_space 结果(如代码清单 1-21 所示)。

代码清单 1-21 将骰子值总和映射到出现次数

该模块返回所有键都使用默认值的字典。例如，defaultdict(int)返回一个字典，其中每个键的默认值都设置为 0

weighted_sample 字典将每个由 6 次投掷所得到的总和映射到出现次数

```
from collections import defaultdict
weighted_sample_space = defaultdict(int)
for outcome in sample_space:
    total = sum(outcome)
    weighted_sample_space[total] += 1
```

每个结果都包含 6 次投掷骰子的唯一组合

计算 6 个独立骰子投掷的总和

更新骰子求和值的出现计数

在重新计算概率之前，我们先简要地探讨 weighted_sample_space 的属性。样本空间中的所有权重并不都相等——有些权重比其他的要小得多。例如，只有一种方法可以使掷出的骰子和为 6：必须精确地掷出 6 个 1 才能达到这种效果。因此，期望 weighted_sample_space[6]等于 1。我们期望 weighted_sample_space[36]也等于 1，因为必须投掷出 6 个 6，才能达到这种效果(如代码清单 1-22 所示)。

代码清单 1-22 查看非常罕见的投掷结果组合

```
assert weighted_sample_space[6] == 1
assert weighted_sample_space[36] == 1
```

相比之下，weighted_sample_space[21]的值明显更高(如代码清单 1-23 所示)。

代码清单 1-23　查看更常见的投掷结果组合

```
num_combinations = weighted_sample_space[21]
print(f"There are {num_combinations } ways for 6 die rolls to sum to 21")

There are 4332 ways for 6 die rolls to sum to 21
```

如输出所示，有 4 332 种方式使 6 次骰子掷出的总和为 21。例如，可以掷出 4 个 4、1 个 3 以及 1 个 2；或者可以掷出 3 个 4，然后 1 个 5、1 个 3 和 1 个 1(如代码清单 1-24 所示)。还有其他数以千计的投掷组合可以得到 21 点，这就是为什么掷出 21 点比 6 点更常见。

代码清单 1-24　探索掷出 21 点的不同方法

```
assert sum([4, 4, 4, 4, 3, 2]) == 21
assert sum([4, 4, 4, 5, 3, 1]) == 21
```

注意，之前观察到的掷出 21 点的 4 332 种方法等于掷出 21 点的未加权事件的长度。此外，weighted_sample 中值的总和等于 sample_space 的长度(如代码清单 1-25 所示)。因此，未加权和加权事件概率计算之间存在直接联系。

代码清单 1-25　比较加权事件和常规事件

```
event = get_matching_event(lambda x: sum(x) == 21, sample_space)
assert weighted_sample_space[21] == len(event)
assert sum(weighted_sample_space.values()) == len(sample_space)
```

现在使用 weighted_sample_space 字典重新计算概率(如代码清单 1-26 所示)。最终掷出 21 点的概率应该保持不变。

代码清单 1-26　计算掷骰子的加权事件概率

```
prob = compute_event_probability(lambda x: x == 21,
                                 weighted_sample_space)
assert prob == compute_event_probability(has_sum_of_21, sample_space)
print(f"6 rolls sum to 21 with a probability of {prob}")

6 rolls sum to 21 with a probability of 0.09284979423868313
```

使用加权样本空间相对于未加权样本空间有什么好处？答案是更少的内存使用。正如接下来将看到的，未加权的 sample_space 集合的元素数量是加权样本空间字典的 150 倍(如代码清单 1-27 所示)。

代码清单 1-27　比较加权事件与未加权事件的样本空间大小

```
print('Number of Elements in Unweighted Sample Space:')
print(len(sample_space))
print('Number of Elements in Weighted Sample Space:')
print(len(weighted_sample_space))
Number of Elements in Unweighted Sample Space:
46656
Number of Elements in Weighted Sample Space:
31
```

1.3 计算区间范围内的概率

到目前为止，我们只分析了满足某个单一值的事件条件。现在将分析某一区间范围内的事件条件。区间是两个边界点中间(包括两个边界点)的所有数字的集合。这里定义一个 is_in_interval 函数来检查某个数字是否在指定的区间内(如代码清单 1-28 所示)。我们将通过最小值参数和最大值参数控制区间边界。

代码清单 1-28 定义区间函数

```
def is_in_interval(number, minimum, maximum):
    return minimum <= number <= maximum
```

定义包含最小值和最大值在内的闭合区间。但你也可以根据需要定义开放区间。在开放区间中，最多只有一个边界值是包含在内的

给定 is_in_interval 函数，我们可以计算事件的关联值落在某个数值范围内的概率。例如，计算连续 6 次掷骰子其总和为 10~21(含)的可能性(如代码清单 1-29 所示)。

代码清单 1-29 计算区间内的概率

```
prob = compute_event_probability(lambda x: is_in_interval(x, 10, 21),
                                 weighted_sample_space)
print(f"Probability of interval is {prob}")

Probability of interval is 0.5446244855967078
```

lambda 函数接收输入 x，如果 x 落在 10~21 之间，则返回 True。这个单行 lambda 函数被用作事件条件

在超过 54%的情况下，6 次掷骰子的总和将落入该区间范围内。因此，如果掷出的总和为 13 或 20，我们不会感到惊讶。

使用区间分析评估极端情况

区间分析对于解决概率和统计中的一类非常重要的问题很关键。其中一个问题是对极端情况的评估：问题归结为观察到的数据是否过于极端以至于不可信。

当发生不平常的随机事件时，数据将表现为很极端。例如，假设我们观察 10 次抛硬币的实验(硬币是标准无偏差的)，并且该硬币 10 次中有 8 次正面朝上。对于一枚标准无偏差的硬币来说，这是一个合理的结果吗？还是硬币暗中偏向于正面向上？为找出答案，必须回答以下问题：10 次抛硬币导致极端数量的正面朝上的概率是多少？我们将极端数量定义为 8 或更多。因此，可以将问题描述如下：在 10 次抛硬币的实验中，产生 8~10 次正面的概率是多少？

我们将通过计算区间概率找到答案。然而，首先需要 10 次抛硬币实验中每种可能结果序列的样本空间。让我们生成一个加权样本空间。如前所述，这比使用非加权样本空间效率更高。

代码清单 1-30 创建了一个 weighted_sample_space 字典。它的键为 10 次抛硬币实验中出现正面朝上的次数总和，范围为 0~10。这个字典的值为出现指定正面朝上次数的可能组合数。因此，我们期望 weighted_sample_space[10]等于 1，因为只有一种可能的方法可以将硬币抛 10 次并得到 10

次正面。同时，预计 weighted_sample_space[9]等于 10，因为这种情况下，9 次正面朝上，1 次反面朝上。而发生反面朝上的那一次可以是第 1 次、第 2 次、第 3 次……或者第 10 次，有 10 种可能。

代码清单 1-30　计算 10 次抛硬币的样本空间

```
def generate_coin_sample_space(num_flips=10):
    weighted_sample_space = defaultdict(int)
    for coin_flips in product(['Heads', 'Tails'], repeat=num_flips):
        heads_count = len([outcome for outcome in coin_flips
                           if outcome == 'Heads'])
        weighted_sample_space[heads_count] += 1
    return weighted_sample_space

weighted_sample_space = generate_coin_sample_space()
assert weighted_sample_space[10] == 1
assert weighted_sample_space[9] == 10
```

为了可以重复使用，我们定义了一个通用函数，该函数返回 num_flips 次投掷硬币的加权样本空间。num_flips 参数预设为 10，表示投掷 10 次

在 num_flips 次抛硬币实验的唯一序列中正面的次数

加权样本空间已准备好。现在可以计算观察到 8~10 次正面向上的概率(如代码清单 1-31 所示)。

代码清单 1-31　计算投掷硬币正面向上的极端概率

```
prob = compute_event_probability(lambda x: is_in_interval(x, 8, 10),
                                 weighted_sample_space)
print(f"Probability of observing more than 7 heads is {prob}")

Probability of observing more than 7 heads is 0.0546875
```

在 10 次抛硬币实验中，得到 7 次以上正面的概率约为 5%。这表明之前看到的 10 次有 8 次正面向上的情况并不常见。这是否意味着硬币是有偏差的？不一定，因为还没考虑过极端的反面向上的情况。即使观察到 8 次反面向上，而不是 8 次正面向上，我们仍然会对硬币产生怀疑。之前的计算区间没有考虑这种极端情况。如果将 8 次正面向上视为极端情况，在不考虑其他极端可能的情况下，8 次或更多次的反面向上将被视为正常的，这样的想法并不正确。为评估硬币的公平性，必须纳入观察到 8 次或更多次反面向上的可能性。这样可以让结果更科学。

我们可以把问题表述为：在 10 次抛硬币实验中，得到 0~2 个正面或 8~10 次正面的概率是多少？或者，更简单地说，在 10 次抛硬币实验中，出现超过 7 次正面或 7 次反面的概率是多少？这可以通过代码清单 1-32 进行计算。

代码清单 1-32　计算极端区间概率

```
prob = compute_event_probability(lambda x: not is_in_interval(x, 3, 7),
                                 weighted_sample_space)
print(f"Probability of observing more than 7 heads or 7 tails is {prob}")

Probability of observing more than 7 heads or 7 tails is 0.109375
```

在 10 次抛硬币实验中，大约有 10%的概率会产生至少 8 次相同的结果。这种可能性很低，但仍在合理范围内。如果没有额外证据，就很难判断这枚硬币是否真的有偏差。因此，我们需要搜集证据。假设再抛 10 次硬币，8 次正面朝上。结果就是 20 次抛硬币中有 16 次正面朝上。我们对硬币公平性的信心下降了，但下降了多少？可以通过测量概率的变化找出答案。如代码清单 1-33 所示，求 20 次抛硬币没有得到 5~15 次正面的概率(即得到超过 15 次正面或者超过 15 次反面的概率)。

代码清单 1-33 分析在 20 次抛硬币实验中正面向上的极端次数

```
weighted_sample_space_20_flips = generate_coin_sample_space(num_flips=20)
prob = compute_event_probability(lambda x: not is_in_interval(x, 5, 15),
                                 weighted_sample_space_20_flips)
print(f"Probability of observing more than 15 heads or 15 tails is {prob}")

Probability of observing more than 15 heads or 15 tails is 0.011817932128906250
```

更新后的概率从大约 0.1 下降到大约 0.01。因此，额外的证据导致我们对硬币公平性的信心仅为原来的 1/10。尽管概率下降，正面与反面的比率仍然保持为 4 比 1。最初和更新后的实验都产生了 80%的正面和 20%的反面。这就引出了一个有趣的问题：为什么随着投掷次数的增加，观察到极端结果的概率会降低？可以通过详细的数学分析找到答案。然而，一个更直观的解决方案是将正面向上的计数在两个样本空间字典中的分布可视化。可视化实际上是字典中每个键(正面向上的次数)与值(这种情况的可能组合数)的关系图。我们可以使用 Python 最流行的可视化库 Matplotlib 绘制这个图。在第 2 章中，将讨论 Matplotlib 的使用及其在概率论中的应用。

1.4 本章小结

- 样本空间是一个行为可以产生的所有可能结果的集合。
- 事件是样本空间的子集，仅包含满足某些事件条件的那些结果。事件条件是一个布尔函数，它将一种结果作为输入并返回 True 或 False。
- 事件的概率等于事件结果占整个样本空间中所有可能结果的比例。
- 概率可以在数值区间内计算。区间是夹在两个边界值之间的所有数的集合。
- 区间概率对于判断观察结果是否极端有重要意义。