통계적 사고

파이썬을 이용한 탐색적 자료 분석

Version 2.0.25

통계적 사고

파이썬을 이용한 탐색적 자료 분석

Version 2.0.25

번역: 이광춘
원저자: Allen B. Downey

xwMOOC
http://www.xwmooc.net

Green Tea Press
9 Washburn Ave
Needham MA 02492

The original form of this book is LaTeX source code. Compiling this code has the effect of generating a device-independent representation of a textbook, which can be converted to other formats and printed.

The LaTeX source for this book is available from `http://thinkstats2.com`.

서문

탐색적 자료 분석(EDA, Exploratory Data Analysis)에 대한 소개서로 작성되었다.

This book is an introduction to the practical tools of exploratory data analysis. The organization of the book follows the process I use when I start working with a dataset:

- Importing and cleaning: Whatever format the data is in, it usually takes some time and effort to read the data, clean and transform it, and check that everything made it through the translation process intact.

- Single variable explorations: I usually start by examining one variable at a time, finding out what the variables mean, looking at distributions of the values, and choosing appropriate summary statistics.

- Pair-wise explorations: To identify possible relationships between variables, I look at tables and scatter plots, and compute correlations and linear fits.

- Multivariate analysis: If there are apparent relationships between variables, I use multiple regression to add control variables and investigate more complex relationships.

- Estimation and hypothesis testing: When reporting statistical results, it is important to answer three questions: How big is the effect? How

much variability should we expect if we run the same measurement again? Is it possible that the apparent effect is due to chance?

- Visualization: During exploration, visualization is an important tool for finding possible relationships and effects. Then if an apparent effect holds up to scrutiny, visualization is an effective way to communicate results.

This book takes a computational approach, which has several advantages over mathematical approaches:

- I present most ideas using Python code, rather than mathematical notation. In general, Python code is more readable; also, because it is executable, readers can download it, run it, and modify it.

- Each chapter includes exercises readers can do to develop and solidify their learning. When you write programs, you express your understanding in code; while you are debugging the program, you are also correcting your understanding.

- Some exercises involve experiments to test statistical behavior. For example, you can explore the Central Limit Theorem (CLT) by generating random samples and computing their sums. The resulting visualizations demonstrate why the CLT works and when it doesn't.

- Some ideas that are hard to grasp mathematically are easy to understand by simulation. For example, we approximate p-values by running random simulations, which reinforces the meaning of the p-value.

- Because the book is based on a general-purpose programming language (Python), readers can import data from almost any source. They are not limited to datasets that have been cleaned and formatted for a particular statistics tool.

The book lends itself to a project-based approach. In my class, students work on a semester-long project that requires them to pose a statistical question, find a dataset that can address it, and apply each of the techniques they learn to their own data.

To demonstrate my approach to statistical analysis, the book presents a case study that runs through all of the chapters. It uses data from two sources:

- The National Survey of Family Growth (NSFG), conducted by the U.S. Centers for Disease Control and Prevention (CDC) to gather "information on family life, marriage and divorce, pregnancy, infertility, use of contraception, and men's and women's health." (See `http://cdc.gov/nchs/nsfg.htm`.)

- The Behavioral Risk Factor Surveillance System (BRFSS), conducted by the National Center for Chronic Disease Prevention and Health Promotion to "track health conditions and risk behaviors in the United States." (See `http://cdc.gov/BRFSS/`.)

Other examples use data from the IRS, the U.S. Census, and the Boston Marathon.

This second edition of *Think Stats* includes the chapters from the first edition, many of them substantially revised, and new chapters on regression, time series analysis, survival analysis, and analytic methods. The previous edition did not use pandas, SciPy, or StatsModels, so all of that material is new.

제 1 절 How I wrote this book

When people write a new textbook, they usually start by reading a stack of old textbooks. As a result, most books contain the same material in pretty much the same order.

I did not do that. In fact, I used almost no printed material while I was writing this book, for several reasons:

- My goal was to explore a new approach to this material, so I didn't want much exposure to existing approaches.
- Since I am making this book available under a free license, I wanted to make sure that no part of it was encumbered by copyright restrictions.
- Many readers of my books don't have access to libraries of printed material, so I tried to make references to resources that are freely available on the Internet.
- Some proponents of old media think that the exclusive use of electronic resources is lazy and unreliable. They might be right about the first part, but I think they are wrong about the second, so I wanted to test my theory.

The resource I used more than any other is Wikipedia. In general, the articles I read on statistical topics were very good (although I made a few small changes along the way). I include references to Wikipedia pages throughout the book and I encourage you to follow those links; in many cases, the Wikipedia page picks up where my description leaves off. The vocabulary and notation in this book are generally consistent with Wikipedia, unless I had a good reason to deviate. Other resources I found useful were Wolfram MathWorld and the Reddit statistics forum, `http://www.reddit.com/r/statistics`.

제 2 절 Using the code

The code and data used in this book are available from `https://github.com/AllenDowney/ThinkStats2`. Git is a version control system that allows you to keep track of the files that make up a project. A collection of files

under Git's control is called a **repository**. GitHub is a hosting service that provides storage for Git repositories and a convenient web interface.

The GitHub homepage for my repository provides several ways to work with the code:

- You can create a copy of my repository on GitHub by pressing the Fork button. If you don't already have a GitHub account, you'll need to create one. After forking, you'll have your own repository on GitHub that you can use to keep track of code you write while working on this book. Then you can clone the repo, which means that you make a copy of the files on your computer.
- Or you could clone my repository. You don't need a GitHub account to do this, but you won't be able to write your changes back to GitHub.
- If you don't want to use Git at all, you can download the files in a Zip file using the button in the lower-right corner of the GitHub page.

All of the code is written to work in both Python 2 and Python 3 with no translation.

I developed this book using Anaconda from Continuum Analytics, which is a free Python distribution that includes all the packages you'll need to run the code (and lots more). I found Anaconda easy to install. By default it does a user-level installation, not system-level, so you don't need administrative privileges. And it supports both Python 2 and Python 3. You can download Anaconda from `http://continuum.io/downloads`.

If you don't want to use Anaconda, you will need the following packages:

- pandas for representing and analyzing data, `http://pandas.pydata.org/`;
- NumPy for basic numerical computation, `http://www.numpy.org/`;

- SciPy for scientific computation including statistics, http://www.scipy.org/;
- StatsModels for regression and other statistical analysis, http://statsmodels.sourceforge.net/; and
- matplotlib for visualization, http://matplotlib.org/.

Although these are commonly used packages, they are not included with all Python installations, and they can be hard to install in some environments. If you have trouble installing them, I strongly recommend using Anaconda or one of the other Python distributions that include these packages.

After you clone the repository or unzip the zip file, you should have a folder called ThinkStats2/code with a file called nsfg.py. If you run nsfg.py, it should read a data file, run some tests, and print a message like, "All tests passed." If you get import errors, it probably means there are packages you need to install.

Most exercises use Python scripts, but some also use the IPython notebook. If you have not used IPython notebook before, I suggest you start with the documentation at http://ipython.org/ipython-doc/stable/notebook/notebook.html.

I wrote this book assuming that the reader is familiar with core Python, including object-oriented features, but not pandas, NumPy, and SciPy. If you are already familiar with these modules, you can skip a few sections.

I assume that the reader knows basic mathematics, including logarithms, for example, and summations. I refer to calculus concepts in a few places, but you don't have to do any calculus.

If you have never studied statistics, I think this book is a good place to start. And if you have taken a traditional statistics class, I hope this book will help repair the damage.

—

Allen B. Downey is a Professor of Computer Science at the Franklin W. Olin College of Engineering in Needham, MA.

Contributor List

If you have a suggestion or correction, please send email to `downey@allendowney.com`. If I make a change based on your feedback, I will add you to the contributor list (unless you ask to be omitted).

If you include at least part of the sentence the error appears in, that makes it easy for me to search. Page and section numbers are fine, too, but not quite as easy to work with. Thanks!

- Lisa Downey and June Downey read an early draft and made many corrections and suggestions.
- Steven Zhang found several errors.
- Andy Pethan and Molly Farison helped debug some of the solutions, and Molly spotted several typos.
- Andrew Heine found an error in my error function.
- Dr. Nikolas Akerblom knows how big a Hyracotherium is.
- Alex Morrow clarified one of the code examples.
- Jonathan Street caught an error in the nick of time.
- Gábor Lipták found a typo in the book and the relay race solution.
- Many thanks to Kevin Smith and Tim Arnold for their work on plasTeX, which I used to convert this book to DocBook.
- George Caplan sent several suggestions for improving clarity.
- Julian Ceipek found an error and a number of typos.
- Stijn Debrouwere, Leo Marihart III, Jonathan Hammler, and Kent Johnson found errors in the first print edition.
- Dan Kearney found a typo.
- Jeff Pickhardt found a broken link and a typo.

- Jörg Beyer found typos in the book and made many corrections in the docstrings of the accompanying code.
- Tommie Gannert sent a patch file with a number of corrections.
- Alexander Gryzlov suggested a clarification in an exercise.
- Martin Veillette reported an error in one of the formulas for Pearson's correlation.
- Christoph Lendenmann submitted several errata.
- Haitao Ma noticed a typo and and sent me a note.
- Michael Kearney sent me many excellent suggestions.
- Alex Birch made a number of helpful suggestions.
- Lindsey Vanderlyn, Griffin Tschurwald, and Ben Small read an early version of this book and found many errors.
- John Roth, Carol Willing, and Carol Novitsky performed technical reviews of the book. They found many errors and made many helpful suggestions.
- Rohit Deshpande found a typesetting error.
- David Palmer sent many helpful suggestions and corrections.
- Erik Kulyk found many typos.
- Nir Soffer sent several excellent pull requests for both the book and the supporting code.
- Joanne Pratt found a number that was off by a factor of 10.

차 례

제 1 장

탐색적 자료 분석

이 책의 주요 논지는 실용적인 방법과 결합된 데이터가 질문에 대답하고, 불확실성하에서 의사결정을 안내하는 것이다.

한가지 사례로, 집사람과 함께 첫번째 아이를 기대할 때 전해들은 질문에 모디브를 얻어 한가지 사례 연구를 제시한다: 첫째 애기는 늦게 낳는 경향이 있을까요?

만약 이 질문을 구글에 검색하면, 상당한 토론글을 볼 수 있다. 몇몇 사람은 사실이라고하고, 다른 사람은 미신이라고 하고, 다른 사람은 첫째 얘들이 일찍 나온다고 애둘러 말하곤 한다.

많은 토론글에서, 사람들은 자신의 주장을 뒷받침하기 위해서 데이터를 제공한다. 다음과 같은 많은 사례를 찾아볼 수 있다.

> "최근에 첫째 아이를 출산한 내 친구 두명은 모두 자연분만 혹은 제왕절개하기 전에 예정일에서 거의 2주나 지났다."
>
> "첫째는 2주 늦게 나왔고 이제 생각하기에 둘째는 2주 빨리 나올것 같다.!!"
>
> "제 생각에는 사실이 아니라고 생각하는데 왜냐하면 언니가 엄마의 첫째인데 다른 많은 사촌과 마찬가지로 빨리 나왔기 때문이다."

이와 같은 보고를 **일화적 증거(anecdotal evidence)**라고 부르는데, 이유는 논문으로 출판되지 않고 대체로 개인적인 데이터에 기반하고 있기 때문이다. 일상적인 대화에서, 일화와 관련해서 잘못된 것은 없다. 그래서 인용한 사람을 콕 집어서 뭐라고 할 의도는 없다.

하지만, 좀더 설득적인 증거와, 좀더 신빙성있는 답을 원할지도 모른다. 이런 기준으로 본다면, 일화적 증거는 대체로 성공적이지 못하다. 왜냐하면:

- 적은 관측치(Small number of observations): 만약 첫째 아기에 대해서 임신 기간이 좀더 길다면, 아마도 차이는 자연적인 변동과 비교하여 적을 것이다. 이 경우에, 차이가 존재한다는 것을 확실히 하기 위해서는 많은 임신 사례를 비교해야할 것이다.

- 선택 편의(Selection bias): 임신기간에 관한 토론에 참가한 사람들은 첫째 아이가 늦게 태어났기 때문에 관심이 있을 수 있다. 이 경우에 데이터를 선택하는 과정이 결과를 왜곡할 수도 있다.

- 확증 편의(Confirmation bias): 주장을 믿는 사람들은 주장을 확증해주는 사례에 좀더 기여할 듯 하다. 주장에 의구심을 갖는 사람들은 반례를 좀더 들것 같다.

- 부정확(Inaccuracy): 일화는 종종 개인 이야기로, 기억이 부정확하고, 잘못 표현되며, 부정확하게 반복된다.

그렇다면, 어떻게 더 잘 할 수 있을까요?

제 1 절 통계적 접근방법

일화적 접근법의 한계를 극복하기 위해서 통계도구를 사용하는데 다음을 포함한다:

- 자료 수집(Data collection): 대규모 국가적 조사에서 나온 자료를 사용한다. 통상 국가적 조사는 명시적으로 U.S 모집단에 대한 통계적으로 타당한 추론을 도출하도록 설계된다.

- 기술통계(Descriptive statistics): 데이터를 간결하게 요약하는 통계량을 생성하고 다른 방식으로 평가하는데 데이터를 시각화한다.

- 탐색적 데이터 분석 (Exploratory data analysis): 관심있는 질문을 다룰 수 있는 패턴, 차이점, 다른 특징을 찾는다. 동시에 일관되지 못함을 점검하고 한계를 식별한다.

- 추정(Estimation): 일반적인 모집단의 특징을 추정하는데 샘플에서 추출된 데이터를 사용한다.

- 가설 검증 (Hypothesis testing): 두 그룹간에 차이처럼 명백한 효과를 확인하는데 있어서 효과가 우연히 발생했는지 평가한다.

함정에 빠지는 것을 피하기 위해서 상기 단계를 조심스럽게 밟아서 좀더 당위성을 가지고 좀더 옳을 것 같은 결론에 도달할 수 있다.

제 2 절 가족 성장 국가 조사 (National Survey of Family Growth)

1973년 이래로 미국 질병 통제예방 센터 (Disease Control and Prevention, CDC)에서 가족 성장 국가 조사 (National Survey of Family Growth, NSFG)를 수행하고 있다. 조사 목적은 "가족 생활, 결혼 및 이혼, 임신, 출산, 피임, 그리고 남녀 건강에 대한 정보를 수집하고," 조사 결과는 "건강 서비스 및 건강 교육 프로그램, 그리고 가족, 출산, 건강에 대한 통계적 조사를 수행"하는데 사용된다. 자세한 사항은 다음 웹사이트를 참조한다. `http://cdc.gov/nchs/nsfg.htm`.

첫째 아이가 늦게 낳는지와 다른 문제를 조사하려고 상기 조사로 수집된 데이터를 사용한다. 데이터를 효과적으로 사용하기 위해서는, 조사 설계(design of the study)를 이해할 필요가 있다.

NSFG는 **횡단적 연구(cross-sectional study)**로 특정 시점에 한 집단에 대한 스냅샷 정보를 수집한다. 가장 흔한 대안 연구는 **종단적 연구(longitudinal study)**로 한 집단을 여러 시점에 걸쳐 반복적으로 관찰하는 것이다.

NSFG는 7번 조사를 수행했다; 각 조사 전개를 **사이클(cycle)**이라고 한다. 2002년 1월에서 2003년 3월까지 수행된 여섯번째 사이클에서 나온 데이터를 사용한다.

조사 목적은 **모집단(population)**에 대한 결론을 도출하는 것으로, NSFG 목표 모집단은 15-44 연령의 미국민이다. 이상적으로 데이터를 전체 모집단의 모든 사람에게서 데이터를 수집하여야 하지만, 현실적으로 불가능하다. 대신에 **표본(sample)**으로 불리는 모집단의 일부에서 데이터를 수집한다. 조사에 참여한 사람을 **응답자(respondents)**라고 부른다.

일반적으로 종단적 연구는 **대표성(representative)**을 가져야 하는데 목표 모집단의 모든 멤버가 동일한 참여 가능성을 가져야한다는 의미다. 이러한 이상은 실무에서 달성하기는 어렵다. 하지만 조사를 수행함에 있어 최대한 근접하도록 노력해야 한다.

NSFG는 대표적이지 않다; 대신에 의도적으로 **오버샘플링(oversampling)**했다. 조사 설계자가 세 집단 (히스패닉, 흑인, 10대)에 대해서 미국인구에서 차지하는 것보다 높은 비율로 조사를 실시한다. 사유는 유효한 통계적 추론을 이끌어 내기 위해서 각 그룹에 대해 충분이 큰 응답자를 확보하기 위해서다.

물론, 오버샘플링 단점은 조사로부터 나온 통계량에 기반하여 일반 모집단에 대한 결론을 도출하기는 쉽지 않다. 나중에 이점에 대해서는 다시 다룰 것이다.

이러한 유형의 데이터를 작업할 때, **코드북(codebook)**과 친근해지는 것이 중요하다. 코드북은 조사 서례, 조사 질문, 응답자 기록을 문서화한다. 코드북과 NSFG 데이터에 대한 사용자 가이드는 웹사이트에서 찾아볼 수 있다. `http://www.cdc.gov/nchs/nsfg/nsfg_cycle6.htm`

제 3 절 데이터 가져오기

이 책에 사용된 코드와 데이터는 https://github.com/AllenDowney/ThinkStats2 사이트에서 사용할 수 있다. 다운로드와 코드로 작업하는 것에 대한 자세한 정보는 2을 참조한다.

코드를 다운로드하면, ThinkStats2/code/nsfg.py이라는 파일이 있다. 실행하면, 데이터 파일을 읽고, 몇가지 테스트를 수행하고, "All tests passed." 라는 메시지를 출력한다.

```
$ python nsfg.py
(13593, 244)
nsfg.py: All tests passed.
```

프로그램이 무엇을 수행하는지 살펴보자. NSFG 6번째 사이클에서 임신 데이터는 파일명이 2002FemPreg.dat.gz이다. 고정폭 칼럼을 가진 일반 텍스트(ASCII) 파일형식으로 gzip으로 압축되어 있다. 파일 각각 라인은 한개 임신에 대한 데이터를 포함하는 **레코드(record)**가 된다.

파일 형식(format)은 2002FemPreg.dct 파일에 문서화되어 기술되어 있고 Stata 딕셔너리 파일이다. Stata는 통계 소프트웨어 시스템 (통계 팩키지)의 일종으로 이러한 맥락에서 "딕셔너리"는 각 행마다 각 변수의 위치를 식별하는데 사용되는 인덱스, 형식, 변수명 목록 정보를 담고 있다.

예를 들어 2002FemPreg.dct 파일에서 몇줄이 다음에 있다.

```
infile dictionary {
  _column(1)  str12  caseid    %12s  "RESPONDENT ID NUMBER"
  _column(13) byte   pregordr   %2f  "PREGNANCY ORDER (NUMBER)"
}
```

딕셔너리는 변수 두개를 기술한다: caseid는 응답자 ID를 대표하는 12 자리 문자로 된 문자열이다; pregorder는 1 바이트 정수형으로 응답자에 대한 임신 정보를 나타낸다.

다운로드한 코드에는 thinkstats2.py 파일이 있는데 파이썬 모듈로 이 책에서 사용되는 많은 클래스와 함수를 포함하고 있다. Stata 딕셔너리와 NSFG 데이터 파일을 읽어 들일 수 있다. 다음에 nsfg.py 프로그램에서 어떻게 사용되는지 사용례가 있다.

```
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
```

ReadStataDct은 딕셔너리 파일이름을 받아서 dct를 반환한다. dct는 딕셔너리에서 받은 정보를 담고 있는 FixedWidthVariables 객체다. dct는 데이터 파일을 읽는 FixedWidthVariables을 제공한다.

제 4 절 데이터프레임(DataFrames)

ReadFixedWidth 결과는 데이터프레임(DataFrame)으로 판다스(pandas)에서 제공하는 가장 근원적인 자료구조다. 판다스는 이 책에서 사용하는 파이썬 자료 및 통계 팩키지 이름이다. 데이터프레임은 각 레코드마다 행(이 경우에 각 임신마다 한 행이 됨)과 각 변수에 대한 열을 포함하고 있다.

데이터외에 데이터프레임은 변수명과 변수 자료형을 포함하고 있으며, 데이터에 접근 및 변경하는 방법을 제공한다.

df를 출력하면, 행과 열, 데이터프레임 모양(13593 행/레코드, 244 열/변수)에 대한 잘려진 일부를 볼 수 있다.

```
>>> import nsfg
>>> df = nsfg.ReadFemPreg()
>>> df
...
[13593 rows x 244 columns]
```

columns 속성은 유니코드 문자열로 칼럼명 시퀀스(sequence)를 반환한다.

```
>>> df.columns
Index([u'caseid', u'pregordr', u'howpreg_n', u'howpreg_p', ... ])
```

결과는 인덱스(Index)로 또다른 판다스 자료구조다. 추후 인덱스(Index)에 관해서 좀더 배울 것이지만, 지금은 리스트처럼 다루기로 한다.

```
>>> df.columns[1]
'pregordr'
```

데이터프레임 칼럼에 접근하기 위해서는 칼럼이름을 키(key)로 사용할 수도 있다.

```
>>> pregordr = df['pregordr']
>>> type(pregordr)
<class 'pandas.core.series.Series'>
```

결과는 시리즈(Series)로 또다른 판다스 자료구조다. 시리즈는 몇가지 추가 기능을 가진 파이썬 리스트다. 시리즈를 출력하면, 인덱스와 상응하는 값이 출력된다.

```
>>> pregordr
0     1
1     2
```

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

상기 예제에서, 인덱스는 0에서 13592 정수형 자료지만, 일반적으로 정렬가능한 임의의 자료형도 가능하다. 요소값도 정수형이지만, 임의의 자료형도 가능하다.

마지막 행은 변수명, 시리즈 길이, 그리고 자료형 정보가 있다; `int64`은 NumPy에서 제공하는 자료형 중의 하나다. 만약 32-비트 컴퓨터에서 상기 예제를 실행한다면, `int32`가 출력된다.

정수 인덱스와 슬라이스(slice)를 사용해서 시리즈 요소값(element)에 접근할 수 있다.

```
>>> pregordr[0]
1
>>> pregordr[2:5]
2    1
3    2
4    3
Name: pregordr, dtype: int64
```

인덱스 연산자 실행 결과는 `int64`이고, 슬라이스 연산자 결과는 또 다른 시리즈다.

점 표기법(dot notation)을 사용해서 데이터프레임 칼럼을 접근할 수도 있다.

```
>>> pregordr = df.pregordr
```

칼럼명이 유효한 파이썬 식별자라면 이 표기법은 잘 동작한다. 그래서 문자로 시작해야 하고, 공백을 포함하지 말아야 하고 등등을 지켜줘야 있다.

제 5 절 변수 (Variables)

NSFG 데이터셋에서 이미 변수 두개, `caseid`와 `pregordr`을 살펴봤다. 전체적으로 244개 변수가 있다는 것도 확인했다. 책에서 탐색적 자료분석으로 다음 변수를 사용한다.

- `caseid`는 응답자의 정수형 ID다.
- `prglngth`는 정수형으로 주로 임신 기간을 나타낸다.

- outcome은 출산 결과에 대한 정수형 코드값이다. 코드값 1 은 정상출산을 나타낸다.
- pregordr은 임신 일련번호다; 예를 들어, 응답자의 첫번째 임신 코드값은 1, 두번째 임신 코드값은 2, 등등.
- birthord는 정상출산에 대한 일련번호다; 응답자의 첫번째 아이 코드값은 1 등등. 정상출산이 아닌 경우에는 필드값이 공백이다.
- birthwgt_lb와 birthwgt_oz은 출산시 아이 체중에 대한 파운드와 온스 정보를 담고 있다.
- agepreg는 임신 후기 산모 나이를 나타낸다.
- finalwgt는 응답자와 연관된 통계적 가중치다. 부동소수점 값으로 응답자가 대표하는 미국 인구중에 비중을 나타낸다.

주의깊이 코드북을 읽게되면, 변수 중의 상당수가 **재코드(recodes)**된 것을 볼 수 있는데 조사에서 수집된 **원자료 (raw data)** 일부분은 아니다. 재코드된 변수는 원자료를 이용하여 계산된 것이다.

예를 들어, 정상출산에 대한 prglngth 변수는 만약 있다면 wksgest (회임기간 주차 정보)와 동일하다; 만약 회임기간 정보가 없다면 mosgest * 4.33 공식(회임기간 월차 정보 곱하기 한달 평균 주차 정보)을 사용해서 추정한다.

재코드는 자료 정합성과 일관성을 점검하는 로직에 기반한다. 스스로 원자료를 처리할 납득이갈만한 이유가 없다면, 일반적으로 재코드된 자료가 있다면 그대로 사용하는 것이 좋은 생각이다.

제 6 절 변환 (Transformation)

이와 같이 데이터를 가져올 때, 오류를 점검하고, 특수값을 처리하고, 데이터를 다른 형식으로 변환하고, 계산을 수행해야 한다. 이와 같은 작업을 통상 **데이터 정제(data cleaning)**라고 부른다.

nsfg.py는 CleanFemPreg 함수가 있어서 사용할 변수를 사전에 정제한다.

```
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.birthwgt_lb + df.birthwgt_oz / 16.0
```

agepreg 변수는 임신 말기에 산모 나이정보를 담고 있다. 데이터 파일에서 agepreg 변수는 백분의 정수로 인코딩되어 있다. 그래서 첫번째은 100으로 agepreg 변수를 나눠서 연도로 부동소수점 값을 생성한다.

birthwgt_lb와 birthwgt_oz 변수는 정상출산 임신에 대한 신생아의 체중으로 파운드와 온스 표현된 정보를 담고 있다. 추가로 몇몇 특수값도 있다.

```
97 NOT ASCERTAINED
98 REFUSED
99 DON'T KNOW
```

숫자 부호로 표현된 특수값은 *위험*한데, 이유는 만약 적절하게 처리되지 않는다면, 99 파운드 신생아처럼 가공된 결과를 생성할 수 있다. replace 메쏘드는 특수값을 np.nan으로 바꾼다. np.nan는 "not a number."를 나타내는 특수 부동소수점값이다. inplace 플래그는 replace에 새로운 시리즈를 생성하는 대신에 기존 시리즈를 변경하게 한다.

IEEE 부동소수점 표준의 일부분으로, 만약 인자중 하나가 nan 이면, 모든 수학 연산은 nan을 반환한다.

```
>>> import numpy as np
>>> np.nan / 100.0
nan
```

그래서, nan으로 연산한 것은 올바른 연산을 하고, 대부분의 판다스 함수는 nan을 적절하게 다룬다. 하지만 결측값(missing value)을 다루는 것은 반복되는 이슈가 된다.

CleanFemPreg 함수 마지막 줄은 파운드와 온스를 하나의 값, 파운드로 조합하는 새로운 칼럼 totalwgt_lb을 생성한다.

중요한 사항: 데이터프레임에 새로운 칼럼을 추가할 때, 다음과 같은 딕셔너리 구문을 사용해야 한다.

```
    # CORRECT
    df['totalwgt_lb'] = df.birthwgt_lb + df.birthwgt_oz / 16.0
```

다음과 같은 점표기법은 안된다.

```
    # WRONG!
    df.totalwgt_lb = df.birthwgt_lb + df.birthwgt_oz / 16.0
```

점표기법 버젼은 데이터프레임 객체에 속성을 추가하지만, 그 속성이 새로운 칼럼으로 다뤄지는 것은 아니다.

제 7 절 타당성 검사(Validation)

데이터를 다른 소프트웨어 환경에서 내보내고 또 다른 소프트웨어 환경으로 가져오기 할 때, 오류가 발생할지도 모른다. 그리고 전혀 새로운 데이터셋에 익숙해질

때, 데이터를 부정확하게 해석하거나 오해가 생겨나기도 한다. 만약 데이터의 타당성을 확보할 시간을 갖게 된다면, 나중에 업무시간을 절약하고 오류를 회필할 수 있다.

데이터 타당성을 확보하는 한 방법은 기초 통계량을 계산하고 공표된 결과값과 비교하는 것이다. 예를 들어, NSFG 코드북에는 각 변수를 요약한 장표가 있다. outcome에 대한 테이블이 있는데 각 임신 결과값을 코드화한 것이다.

```
value label           Total
1 LIVE BIRTH             9148
2 INDUCED ABORTION       1862
3 STILLBIRTH              120
4 MISCARRIAGE            1921
5 ECTOPIC PREGNANCY       190
6 CURRENT PREGNANCY       352
```

시리즈 클래스는 value_counts 라는 메쏘드를 제공하는데 각 값이 출현하는 횟수를 계수한다. 만약 데이터프레임에서 outcome 시리즈를 선택한다면, value_counts를 사용해서 공표된 값과 비교할 수 있다.

```
>>> df.outcome.value_counts().sort_index()
1    9148
2    1862
3     120
4    1921
5     190
6     352
```

value_counts 결과는 시리즈이고 sort_index가 인덱스별로 시리즈를 정렬한다. 그래서 ㄱ 결과값이 순서대로 나타난다.

결과값을 공표된 장표와 비교하면, outcome 값이 올바른 것처럼 보인다. 마찬가지로, birthwgt_lb에 대한 공표된 장표가 다음에 있다.

```
value label                Total
. INAPPLICABLE            4449
0-5 UNDER 6 POUNDS          1125
6 6 POUNDS                2223
7 7 POUNDS                3049
8 8 POUNDS                1889
9-95 9 POUNDS OR MORE          799
```

그리고, 계수된 값이 다음에 있다.

```
>>> df.birthwgt_lb.value_counts().sort_index()
0        8
1       40
```

```
2        53
3        98
4       229
5       697
6      2223
7      3049
8      1889
9       623
10      132
11       26
12       10
13        3
14        3
15        1
51        1
```

6, 7, 8 에 대한 계수는 확인됐고, 만약 0-5와 9-95 범위 숫자를 더한다면, 계수도 맞는 것으로 확인된다. 하지만, 만약 좀더 자세히 들여다본다면, 오류가 있는 값을 하나 발견할 것이다. 51 파운드 신생아!

이 오류를 다루기 위해서, `CleanFemPreg` 프로그램에 한줄 코드를 추가한다.

```
df.birthwgt_lb[df.birthwgt_lb > 20] = np.nan
```

상기 문장은 유효하지 않은 값을 `np.nan`으로 바꾼다. 꺾쇠괄호에 있는 표현식은 부울(`bool`) 자료형 시리즈를 산출한다. 여기서 True는 조건이 참인 것을 나타낸다. 불리언 시리즈가 인덱스로 사용될 때, 조건을 만족하는 요소값(element)만 선택한다.

제 8 절 해석 (Interpretation)

효과적으로 데이터를 가지고 작업하기 위해서는 동시에 두 관점에 대해서 생각해야만 한다: 통계적관점과 문맥적 관점.

사례로 응답자 몇명에 대한 응답 시퀀스를 살펴보자. 데이터 파일이 구성된 방식 때문에 각 응답자에 대한 임신 데이터를 수집하기 위해서는 몇가지 작업을 해야 한다. 다음이 작업을 수행하는 함수다.

```
def MakePregMap(df):
    d = defaultdict(list)
    for index, caseid in df.caseid.iteritems():
        d[caseid].append(index)
    return d
```

df는 임신 데이터가 있는 데이터프레임이다. iteritems 메쏘드는 임신 각각에 대해 인덱스(행 번호)와 caseid를 하나씩 열거한다.

d는 딕셔너리로 각 ID에서 인덱스 리스트로 매핑한다. 만약 defaultdict과 친숙하지 않다면, 파이썬 collections 모듈로 생각하면 된다. d를 사용해서, 응답자를 찾아내고 해당 응답자 임신에 대한 인덱스를 얻을 수 있다.

다음 사례는 응답자 한명을 찾아 응답자하신 여성분의 임신 결과를 리스트로 출력한다.

```
>>> caseid = 10229
>>> indices = preg_map[caseid]
>>> df.outcome[indices].values
[4 4 4 4 4 4 1]
```

indices는 응답자 번호 10229에 대응되는 임신 인덱스 리스트다.

인덱스로 df.outcome에 리스트를 사용하서 표식된 행을 선택하고 시리즈를 산출한다. 전체 시리즈를 출력하는 대신에 NumPy 배열인 values 속성을 선택했다.

코드 결과값 1은 정상출산을 나타낸다. 코드 결과값 4는 유산을 나타낸다; 즉, 통상 알려진 의학적인 원인 없이 자연스럽게 끝난 임신.

통계적으로 해당 응답자가 유별난 것은 아니다. 유산은 흔하며, 다수 혹은 그 이상을 보고한 다른 응답자도 있다.

하지만 문맥을 기억한다면, 데이터가 말하는 것은 6번 임신했고, 매번 유산으로 끝난 한 여성의 이야기다. 7번째로 가장 최근 임신은 정상출산으로 마무리 되었다. 만약 연민을 가지고 데이터를 생각한다면, 데이터가 전하는 이야기로 감동받는 것은 어쩌면 자연스럽다.

NSFG 데이터셋의 각 레코드는 아주 많은 개인적이고 어려운 질문에 대한 정직한 응답을 제공한 사람을 대표한다. 해당 데이터를 사용해서 가족 생활, 출산, 건강에 관한 통계적 질문에 답할 수 있다. 동시에 데이터로 대표되는 사람을 사려깊이 생각하고, 존경과 감사를 가질 의무도 있다.

제 9 절 연습 문제 (Exercises)

Exercise 1.1 In the repository you downloaded, you should find a file named chap01ex.ipynb, which is an IPython notebook. You can launch IPython notebook from the command line like this:

```
$ ipython notebook &
```

If IPython is installed, it should launch a server that runs in the background and open a browser to view the notebook. If you are not familiar with IPython, I suggest you start at `http://ipython.org/ipython-doc/stable/notebook/notebook.html`.

You can add a command-line option that makes figures appear "inline"; that is, in the notebook rather than a pop-up window:

```
$ ipython notebook --pylab=inline &
```

Open `chap01ex.ipynb`. Some cells are already filled in, and you should execute them. Other cells give you instructions for exercises you should try.

A solution to this exercise is in `chap01soln.ipynb`

Exercise 1.2 Create a file named `chap01ex.py` and write code that reads the respondent file, `2002FemResp.dat.gz`. You might want to start with a copy of `nsfg.py` and modify it.

The variable `pregnum` is a recode that indicates how many times each respondent has been pregnant. Print the value counts for this variable and compare them to the published results in the NSFG codebook.

You can also cross-validate the respondent and pregnancy files by comparing `pregnum` for each respondent with the number of records in the pregnancy file.

You can use `nsfg.MakePregMap` to make a dictionary that maps from each `caseid` to a list of indices into the pregnancy DataFrame.

A solution to this exercise is in `chap01soln.py`

Exercise 1.3 The best way to learn about statistics is to work on a project you are interested in. Is there a question like, "Do first babies arrive late," that you want to investigate?

Think about questions you find personally interesting, or items of conventional wisdom, or controversial topics, or questions that have political consequences, and see if you can formulate a question that lends itself to statistical inquiry.

Look for data to help you address the question. Governments are good sources because data from public research is often freely available. Good places to start include `http://www.data.gov/`, and `http://www.science.gov/`, and in the United Kingdom, `http://data.gov.uk/`.

Two of my favorite data sets are the General Social Survey at `http://www3.norc.org/gss+website/`, and the European Social Survey at `http://www.europeansocialsurvey.org/`.

If it seems like someone has already answered your question, look closely to see whether the answer is justified. There might be flaws in the data or the analysis that make the conclusion unreliable. In that case you could perform a different analysis of the same data, or look for a better source of data.

If you find a published paper that addresses your question, you should be able to get the raw data. Many authors make their data available on the web, but for sensitive data you might have to write to the authors, provide information about how you plan to use the data, or agree to certain terms of use. Be persistent!

제 10 절 용어사전

- **일화적 증거 (anecdotal evidence)**: 제대로 설계된 조사에 의한 것보다는 우연히 수집된 종종 개인적인 증거.
- **모집단 (population)**: 조사에서 관심을 가지는 그룹. "모집단"은 종종 한 무리의 사람을 가르키지만, 용어가 또한 다른 대상에 대해서도 사용된다.
- **종단적 연구 (cross-sectional study)**: 특정 시점에 모집단에 관한 자료를 수집하는 연구.
- **사이클 (cycle)**: 반복되는 종단 연구에서, 연구 반복을 사이클이라고 한다.
- **횡단적 연구 (longitudinal study)**: 시간을 두고 모집단을 추적하는 연구로 동일 그룹에서 반복적으로 데이터를 수집한다.
- **레코드(record)**: 데이셋에서, 한 사람 혹은 다른 피험자에 관한 정보 집합.
- **응답자 (respondent)**: 조사에 응답한 사람.
- **표본 (sample)**: 자료 수집하는데 사용된 모집단의 부분집합.
- **대표성 (representative)**: 만약 모집단의 모든 멤버가 표본에 뽑힐 가능성이 동일한다면 표본은 대표성이 있다.
- **오버샘플링 (oversampling)**: 적은 표본 크리로 생기는 오류를 피하기 위해서 하위 모집단의 대표성을 키우는 기법.
- **원자료 (raw data)**: 점검, 계산, 해석이 거의 없거나 전혀 없는 상태로 기록되고 수집된 값.

- **재코드 (recode)**: 원자료에 적용된 계산 혹은 다른 로직으로 생성된 값.
- **자료 정제 (data cleaning)**: 데이터 타당성 확보, 오류 식별, 자료형간의 변환, 자료 표현 등을 포함하는 프로세스.

제 2 장

분포 (Distribution)

제 1 절 히스토그램

변수를 기술하는 가장 좋은 방법중의 하나는 데이터셋에 나타나는 값과 각 값이 얼마나 나타나는지를 보고하는 것이다. 이러한 기술법을 변수 **분포(distribution)**라고 부른다.

가장 일반적인 분포 표현은 **히스토그램(histogram)**으로 각 값의 **빈도(frequency)**를 보여주는 그래프다. 이러한 맥락에서 "빈도"는 값이 출현하는 횟수를 의미한다.

파이썬에서 빈도를 계산하는 효율적인 방법은 딕셔너리를 사용하는 것이다. 시퀀스 값 t가 주어진 상태에서,

```
hist = {}
for x in t:
    hist[x] = hist.get(x, 0) + 1
```

결과는 값을 빈도로 매칭하는 딕셔너리다. 대안으로 collections 모듈에 정의된 Counter 클래스를 사용할 수 있다.

```
from collections import Counter
counter = Counter(t)
```

결과는 Counter 객체로 딕셔너리의 하위클래스가 된다.

또다른 선택지는 판다스 value_counts 메쏘드를 사용하는 것으로 앞장에서 살펴봤다. 하지만, 이 책에서 히스토그램을 나타내는 Hist 클래스를 생성하고 Hist 클래스에서 동작하는 메쏘드를 제공한다.

그림 2.1: Histogram of the pound part of birth weight.

제 2 절 히스토그램 표현하기

Hist 생성자는 시퀀스, 딕셔너리, 판다스 시리즈, 혹은 다른 Hist를 받을 수 있다. 다음과 같이 Hist 객체 인스턴스를 생성할 수 있다.

```
>>> import thinkstats2
>>> hist = thinkstats2.Hist([1, 2, 2, 3, 5])
>>> hist
Hist({1: 1, 2: 2, 3: 1, 5: 1})
```

Hist 객체는 `Freq` 메쏘드를 제공하는데 값을 받아 빈도를 반환한다.

```
>>> hist.Freq(2)
2
```

꺾쇠 연산자도 동일한 것을 수행한다.

```
>>> hist[2]
2
```

만약 찾는 값이 없다면, 빈도는 0이다.

```
>>> hist.Freq(4)
0
```

`Values` 메쏘드는 Hist에 정렬되지 않는 리스트 값을 반환한다.

```
>>> hist.Values()
[1, 5, 3, 2]
```

정렬된 값으로 루프를 돌리려면, 내장함수 `sorted`를 사용할 수 있다.

```
for val in sorted(hist.Values()):
    print(val, hist.Freq(val))
```

`Items` 메쏘드를 사용해서 값-빈도(value-frequency) 짝을 반복처리할 수 있다.

```
for val, freq in hist.Items():
     print(val, freq)
```

제 3 절 히스토그램 그리기 (Plotting histograms)

이 책에서 저자는 `thinkplot.py` 모듈을 작성해서 Hists를 그리는 함수와 `thinkstats2.py`에 정의된 객체를 제공한다. `pyplot`에 기반하고 있고 `matplotlib` 패키지의 일부다. `matplotlib`을 설치하는 방법은 2을 참조하세요.

`thinkplot`으로 `hist`를 그리기 위해서 다음을 시도해 보세요.

그림 2.2: Histogram of the ounce part of birth weight.

그림 2.3: Histogram of mother's age at end of pregnancy.

```
>>> import thinkplot
>>> thinkplot.Hist(hist)
>>> thinkplot.Show(xlabel='value', ylabel='frequency')
```

`http://greenteapress.com/thinkstats2/thinkplot.html` 웹사이트에서 `thinkplot`에 대한 문서를 참조할 수 있다.

제 4 절 NSFG 변수

이제 NSFG에 있는 데이터로 다시 돌아가자. 이장에 있는 코드는 `first.py`다. 코드를 다운로드하고 작업에 대한 정보는 2 장을 참조하라.

새로운 데이터셋을 가지고 작업을 시작할 때, 한번에 하나씩 사용하려는 변수를 탐색하길 제안한다. 시작하는 좋은 방법은 히스토그램을 그려보는 것이다.

6에서 `agepreg`를 백분년에서 년단위로 변환했고, `birthwgt_lb`와 `birthwgt_oz`을 조합해서 `totalwgt_lb` 한 단위량으로 만들었다. 이번 절에서 히스토그램의 몇가지 기능을 시연하기 위해서 이 변수들을 사용한다.

데이터를 읽고, 정상 출산에 대한 레코드를 선택해서 시작해보자.

```
preg = nsfg.ReadFemPreg()
live = preg[preg.outcome == 1]
```

꺾쇠 표현식은 부울 시리즈(boolean Series)로 데이터프레임에서 행을 선택하고 새로운 데이터프레임을 반환한다. 다음에 정상출산에 대한 `birthwgt_lb` 히스토그램을 생성하고 플롯해서 그려낸다.

```
hist = thinkstats2.Hist(live.birthwgt_lb, label='birthwgt_lb')
thinkplot.Hist(hist)
thinkplot.Show(xlabel='pounds', ylabel='frequency')
```

Hist에 인자가 판다스 시리즈일 때, `nan` 값은 탈락한다. `label`은 문자열로 Hist가 그려질 때 범례에 나타난다.

그림 2.4: Histogram of pregnancy length in weeks.

그림 2.1가 결과를 보여준다. 가장 많이 관찰되는 값을 **최빈값(mode)**이라고 하고 이 경우에 7 파운드다. 분포는 근사적으로 종모양인 **정규(normal)** 분포 모양으로 **가우스(Gaussian)** 분포라고도 한다. 하지만, 순수 정규분포와 달리, 분포가 비대칭이다; 오른쪽보다 왼쪽으로 좀더 확장된 **꼬리(tail)**가 있다.

그림 2.2은 변수 `birthwgt_oz`의 히스토그램으로 출생 체중의 온스 부분이다. 이론적으로 분포가 **균등(uniform)**하길 기대한다; 즉, 모든 값이 동일한 빈도를 가져야 한다. 사실 0 이 다른 값과 비교하여 좀더 흔하고, 1과 15는 좀더 흔하지 않은데, 아마도 이유는 응답자가 정수값에 가까운 출생 체중을 반올림한것으로 추측한다.

그림 2.3은 `agepreg`의 히스토그램으로 임신 말기 산모 나이다. 분포는 대략 종모양이지만, 이 사례의 경우 꼬리가 좀더 왼쪽보다 오른쪽으로 뻗여나갔다. 대부분의 산모는 20대이지만 30대도 적은 수지만 존재한다.

그림 2.4은 `prglngth`의 히스토그램으로 주간(week) 단위 임신기간이다. 가장 흔한 값은 39주다. 외쪽 꼬리가 오른쪽 꼬리보다 길다; 조산 신생아가 일반적이지만, 임신기간이 43주를 넘어가지는 않고, 만약 의사가 판단하기에 필요하다면 임신기간에 관여하는 것이 일반적이다.

제 5 절 특이점 (Outliers)

히스토그램을 보면, 가장 흔한 값과 분포 모양을 식별하기는 쉽다. 하지만, 드문 값이 항상 눈에 잘 뜨이지는 않는다.

계속 진행하기 전에, **특이점 (outliers)**을 점검하는 것이 좋은 생각이다. 특이점은 이상치라고 불리는 극단값으로 측정과 기록에서 오류일 수도 있고, 드분 사건의 정확한 기록일 수도 있다.

Hist는 `Largest`, `Smallest` 메쏘드를 제공하는데, 정수 `n`을 받아 히스토그램에서 최대값과 최소값 `n`개를 반환한다.

```
for weeks, freq in hist.Smallest(10):
    print(weeks, freq)
```

정상 출산에 대한 임신기간 리스트에서 가장 작은 최소값 10개는 `[0, 4, 9, 13, 17, 18, 19, 20, 21, 22]`이다. 10 주보다 적은 값은 확실한 오류다; 가장 그럴듯한 설명은 아마도 결과를 올바르게 코드화하지 못한 것이다. 30주 이상되는 값은 아마도 적합하다. 10주에서 30주 사이의 값은 확실하다고 하기가 어렵다; 몇몇 값은 아마도 오류지만 몇몇은 미숙아를 나타낸다.

범위의 반대편에서 가장 큰 값은 다음과 같다.

```
weeks  count
43     148
44     46
45     10
46     1
47     1
48     7
50     2
```

만약 임신기간이 42주를 넘어가면 대부분의 의사는 유도분만(induced labor)을 추천한다. 그래서 몇몇 긴 임신기간값은 놀라움을 준다. 특히, 임신 50주는 의학적으로 가능하지 않아 보인다.

특이점을 다루는 가장 좋은 방법은 "특정 분야의 전문 지식(domain knowledge)"에 달려있다; 즉, 데이터 출처와 데이터가 의미하는 바에 대한 정보. 그리고 어떠한 분석방법을 수행할지에 달려있다.

이번 예제에서, 동기 부여 질문은 첫번째 아이가 일찍 (혹은 늦게) 태어나는 경향이 있느냐는 것이다. 사람들이 이 질문을 할 때, 대체로 전체 임신기간에 관심이 있다. 그래서, 이번 분석에서는 27주 이상이 되는 임신에 초점을 맞출 것이다.

제 6 절 첫번째 아이 (First babies)

첫째 아이와 나머지 아이에 대한 임신 기간 분포를 이제 비교할 수 있다. `birthord` 변수를 사용해서 정상 출산 데이터프레임을 나누고 히스토그램을 연산해서 그린다.

```
firsts = live[live.birthord == 1]
others = live[live.birthord != 1]

first_hist = thinkstats2.Hist(firsts.prglngth)
other_hist = thinkstats2.Hist(others.prglngth)
```

그리고 나서, 동일 축에 히스토그램을 플롯하여 그린다.

```
width = 0.45
thinkplot.PrePlot(2)
thinkplot.Hist(first_hist, align='right', width=width)
thinkplot.Hist(other_hist, align='left', width=width)
thinkplot.Show(xlabel='weeks', ylabel='frequency')
```

`thinkplot.PrePlot` 함수는 그래프를 그리려고하는 히스토그램 갯수를 인자로 받는다; 인자 정보를 이용해서 적절한 색깔 집합을 선택할 수 있다.

그림 2.5: Histogram of pregnancy lengths.

`thinkplot.Hist`는 `align='center'`을 기본값으로 사용해서 각 막대는 값에 대해서 중앙에 위치한다. 그림에서는 `align='right'`와 `align='left'`를 사용해서 각 값의 양쪽으로 해당 막대를 위치시켰다.

`width=0.45` 옵션값으로, 두 막대의 전체 넓이는 0.9로, 각 막대페어 간에 약간 공백을 두었다.

마지막으로, 축을 조정해서 27주와 46주 사이 데이터만 보이게 만들었다. 그림 2.5이 지금까지의 결과를 보여준다.

히스토그램은 최빈값을 즉시 명확하게 보여준다는 점에서 유용하다. 하지만, 두 분포를 비교하는데는 가장 좋은 선택이 되지는 못하다. 이번 예제에서 "첫째가 아닌 아이"보다 "첫째 아이" 숫자가 더 적다. 그래서, 히스토그램에서 명백한 차이는 표본크기에서 나온다. 다음 장에서 확률질량함수(probability mass functions)를 사용해서 이 문제를 다뤄본다.

제 7 절 분포 요약하기

히스토그램은 표본을 완전히 기술하는 분포다; 즉, 히스토그램이 주어진다면 (순서대로는 할 수 없지만) 표본에 있는 값을 재구성할 수 있다.

만약 분포에 대한 자세한 사항이 중요하다면, 히스토그램으로 표현하는 것이 필요할지도 모른다. 하지만, 종종 기술 통계량 몇개로 분포를 요약할 때도 있다.

보고하는데 사용되는 특성치 몇개는 다음과 같다.

- 중심경향성 (central tendency): 값들이 특정한 점을 주심으로 군집하는 경향이 있는가?
- 모드 (modes): 하나 이상 군집(cluster)이 있는가?
- 퍼짐 (spread): 값들에 얼마나 많은 변동이 있는가?
- 꼬리 (tails): 모드(최빈값)에서 멀어질 때, 확률이 얼마나 빨리 떨어지는가?
- 특이점 (outliers): 모드(최빈값)에서 멀리 떨어진 극단값이 있는가?

상기 질문에 대답하도록 설계된 통계량이 **요약통계 (summary statistics)**다. 지금까지 가장 흔한 요약통계는 **평균(mean)**으로 분포의 중심경향성을 기술하는 의도가 있다.

각 값이 x_i인 n개 표본이 있다면, 평균 $\bar{x}$는 값을 합한 뒤에 표본갯수로 나눈 것이다; 다른 말로,

$$\bar{x} = \frac{1}{n}\sum_i x_i$$

"평균 (mean)"와 "평균(average)"은 때때로 상호호환적으으로 사용된다. 하지만 다음과 같이 이 책에서 구별한다.

- 표본 "평균 (mean)"은 앞선 공식으로 계산되는 요약 통계다.
- "평균 (average)"은 중심경향성을 기술하려고 선택하는 다수 요약통계량 중 하나다.

때때로 평균(mean)은 값들의 집합을 잘 기술한다. 예를 들어, 사과는 거의 크기가 동일하다. (최소한 마트에서 팔리는 사과는 그렇다.) 그래서, 사과 6개를 사고, 전체 무게가 3 파운드라면, 사과 각각은 0.5 파운드라고 주장하는 것은 일리있는 요약이 된다.

하지만, 호박은 좀더 다양하다. 텃밭에서 호박 몇종을 기른다고 가정하자. 어느날 1 파운드 장식용 호박 3개, 3 파운드 호박파이용 호박 2개, 그리고 519 파운드 대서양 거인 호박 한개를 수확했다. 표본 평균은 100 파운드다. 하지만, "텃밭에 호박 평균 무게는 100 파운드"라고 말한다면 오도할 수 있다. 호박 무게에 대해서 전형적인 호박이 없기 때문에 유의미한 평균은 없다.

제 8 절 분산 (Variance)

호박 무게를 요약하는 단 하나의 숫자가 없다면, 숫자 두개로 좀더 잘 할 수 있다; 평균(meand)과 **분산 (variance)**.

분산은 분포의 변동(variability)과 퍼짐(spread)을 기술하려는 요약통계다. 값들의 집합에 대한 분산은 다음과 같다.

$$S^2 = \frac{1}{n}\sum_i (x_i - \bar{x})^2$$

$x_i - \bar{x}$ 항은 "평균으로부터 편차(deviation from the mean)"라고 하고, 분산은 평균제곱편차다. 분산의 제곱근(S)을 **표준편차 (standard deviation)**라고 한다.

이전에 경험이 있다면, 분모에 n 대신에 $n-1$로 분산을 계산하는 공식을 봤을 것이다. 이 통계치는 표본을 사용해서 모집단 분산을 추정하는데 사용된다. 이것에 대해서는 나중에 8장에서 다시 다룰 것이다.

판다스 자료구조는 평균, 분산, 표준편차를 계산하는 메쏘드를 제공한다.

```
mean = live.prglngth.mean()
var = live.prglngth.var()
std = live.prglngth.std()
```

모든 정상 분만에 대해서, 평균 임신기간은 38.6주, 표준편차는 2.7주로 의미하는 바는 일반적으로 2-3주 편차가 있다는 것이다.

임신 기간의 분산은 7.3으로 해석하기가 여렵다. 특히, 단위가 $주^2$, 즉 "주 제곱"이다. 분산은 몇몇 계산에서는 유용하지만, 좋은 요약통계는 아니다.

제 9 절 효과 크기 (Effect size)

효과 크기(effect size)는 효과의 크기를 기술하려는 요약 통계다. 예를 들어, 두 집단간에 차이를 기술하기 위해서 평균값의 차이는 명확한 한가지 선택이 된다.

첫째 아이에 대한 평균 임신기간은 38.601; 첫째를 제외한 아이들의 평균 임신기간은 38.523이다. 차이는 0.078주가 되고, 시간으로는 13시간이다. 전형적인 임신기간에 대한 일부분으로 보면, 차이는 약 0.2%가 된다.

만약 추정치가 정확하다고 가정하면, 이와 같은 차이는 실질적인 중요성은 없다. 사실, 대규모 임신사례를 관찰하지 않고 누구도 이와 같은 차이를 인지할 것 같지는 않다.

효과의 크기를 전달하는 또 다른 방법은 집단간(between groups)의 차이를 집단내(within groups) 변동성과 비교하는 것이다. 코헨(Cohen) d가 이러한 의도를 가진 통계량이다; 다음과 같이 정의된다.

$$d = \frac{\bar{x_1} - \bar{x_2}}{s}$$

$\bar{x_1}$와 $\bar{x_2}$은 집단 평균값이고, s는 "합동 표준편차(pooled standard deviation)"다. 코헨(Cohen) d를 계산하는 파이썬 코드가 다음에 있다.

```
def CohenEffectSize(group1, group2):
    diff = group1.mean() - group2.mean()

    var1 = group1.var()
    var2 = group2.var()
    n1, n2 = len(group1), len(group2)

    pooled_var = (n1 * var1 + n2 * var2) / (n1 + n2)
    d = diff / math.sqrt(pooled_var)
    return d
```

이 예제애서, 평균값의 차이는 0.029 표준편차로 크지 않다. 이러한 관점에서 본다면, 남성과 여성의 키 차이는 약 1.7 표준편차다.(`https://en.wikipedia.org/wiki/Effect_size` 참조)

제 10 절 결과 보고하기

첫째 아이와 첫째가 아닌 아이간에 임신 기간(만약 차이가 있다면)에 차이를 기술하는 몇가지 방법을 살펴봤다.

질문은 누가 질문을 하느냐에 달려있다. 과학자는 아무리 작더라도 존재하는 (실제) 차이에 관심이 있을지 모른다. 의사는 단지 **임상적으로 유의한 (clinically significant)** 효과에만 관심을 둘 수 있다; 즉, 치료결정에 영향을 주는 차이. 임신한 여성은 늦게 혹은 빨리 출산할 확률 같은 본인에게 관련된 결과에만 관심이 있을 수 있다.

어떻게 결과를 보고할지도 또한 목적에 달려있다. 만약 효과의 중요성을 시연하려고 한다면, 차이를 강조하는 요약통계량을 선택할 수도 있다. 만약 환자를 안심시키고자 한다면, 차이를 염두에 둔 통계량을 선택할지도 모른다.

물론, 본인의 결정은 또한 전문가 윤리에 따라야만 한다. 설득하려고 한다면 괜찮다; 스토리를 명확하게 전달하는 시각화 도구와 통계 보고서를 설계*해야만 한다.* 하지만, 또한 보고서를 정직하게 만들고, 불확실성과 한계를 인정하는데도 최선을 다해야 한다.

제 11 절 연습 문제

Exercise 2.1 Based on the results in this chapter, suppose you were asked to summarize what you learned about whether first babies arrive late.

Which summary statistics would you use if you wanted to get a story on the evening news? Which ones would you use if you wanted to reassure an anxious patient?

Finally, imagine that you are Cecil Adams, author of *The Straight Dope* (`http://straightdope.com`), and your job is to answer the question, "Do first babies arrive late?" Write a paragraph that uses the results in this chapter to answer the question clearly, precisely, and honestly.

Exercise 2.2 In the repository you downloaded, you should find a file named `chap02ex.ipynb`; open it. Some cells are already filled in, and you

should execute them. Other cells give you instructions for exercises. Follow the instructions and fill in the answers.

A solution to this exercise is in `chap02soln.ipynb`

For the following exercises, create a file named `chap02ex.py`. You can find a solution in `chap02soln.py`.

Exercise 2.3 The mode of a distribution is the most frequent value; see `http://wikipedia.org/wiki/Mode_(statistics)`. Write a function called `Mode` that takes a Hist and returns the most frequent value.

As a more challenging exercise, write a function called `AllModes` that returns a list of value-frequency pairs in descending order of frequency.

Exercise 2.4 Using the variable `totalwgt_lb`, investigate whether first babies are lighter or heavier than others. Compute Cohen's d to quantify the difference between the groups. How does it compare to the difference in pregnancy length?

제 12 절 용어사전

- 분포 (distribution): 표본에 나타나는 값과 개별 값의 빈도
- 히스토그램 (histogram): 값에서 빈도로 매핑, 혹은 매핑을 보여주는 그래프.
- 빈도 (frequency): 표본에서 값이 나타나는 횟수.
- 최빈값 (mode): 표본에서 가장 빈도가 높은 값 혹은 가장 빈도가 높은 값중의 하나.
- 정규분포 (normal distribution): 이상적인 종모양 분포; 가우스 분포로도 알려져 있다.
- 균등분포 (uniform distribution): 모든 값이 동일한 본도를 갖는 분포.
- 꼬리 (tail): 가장 높고 가장 낮은 극단에 있는 분포 부분.
- 중심경향성 (central tendency): 표본 혹은 모집단 특성치; 직관적으로 평균 혹은 전형적인 값.
- 특이점 (outlier): 중심경향성에서 떨어진 값.
- 퍼짐 (spread): 분포에 있는 값들이 얼마나 퍼져있는지에 대한 측정.

- 요약통계 (summary statistic): 중심경향성 혹은 퍼짐같이 분포의 일부 측면을 정량화하는 통계치.
- 분산 (variance): 퍼짐을 정량화하는데 종종 사용되는 요약통계.
- 표준편차 (standard deviation): 분산의 제곱근으로 또한 퍼짐을 측정하는데 사용된다.
- 효과 크기 (effect size): 집단간의 차이처럼 효과 크기를 정량화하는 요약통계.
- 임상적으로 유의한 (clinically significant): 집단간 차이처럼, 실제로 연관있는 결과.

제 3 장

확률 질량 함수

이번 장에서 사용되는 코드는 `probability.py`에 있다. 코드를 다운로드하고 작업하는 것에 대한 정보는 2을 참조한다.

제 1 절 Pmf

분포를 표현하는 또다른 방식은 **확률 질량 함수(probability mass function)** (PMF)로 각 값을 확률로 매핑한다. **확률(probability)**은 표본 크기 `n`의 일부로서 표현되는 빈도다. 빈도에서 확률을 얻기 위해서, `n`으로 나누는데 이를 **정규화(normalization)**라고 부른다.

Hist가 주어지면, 각 값에 확률값을 매핑하는 딕셔너리를 만들 수 있다.

```
n = hist.Total()
d = {}
for x, freq in hist.Items():
    d[x] = freq / n
```

혹은 `thinkstats2`에서 제공하는 Pmf 클래스를 사용할 수도 있다. Hist와 마찬가지로, Pmf 생성자는 판다스, 시리즈, 딕셔너리, Hist, 다른 Pmf 객체를 받을 수 있다. 다음에 간단한 리스트를 입력값으로 받는 예제가 있다.

```
>>> import thinkstats2
>>> pmf = thinkstats2.Pmf([1, 2, 2, 3, 5])
>>> pmf
Pmf({1: 0.2, 2: 0.4, 3: 0.2, 5: 0.2})
```

Pmf는 정규화 과정을 거쳐 전체 확률값이 1이 된다.

Pmf와 Hist 객체는 많은 점에서 비슷하다; 사실, 공통 부모 클래스에서 많은 메쏘드를 상속받았다. 예를 들어, `Values`와 `Items` 메쏘드는 두 객체 모두에 동일한 방식으로 동작한다. 가장 큰 차이점은 Hist가 값을 정수형 계수기(integer counter)로 매핑한다는 것이고; Pmf는 값을 부동소수점 확률값으로 매핑한다는 것이다.

값과 연관된 확률값을 조회하려면, `Prob`를 사용한다.:

```
>>> pmf.Prob(2)
0.4
```

꺾쇠 연산자도 동등한 기능을 한다.

```
>>> pmf[2]
0.4
```

기존 Pmf를 값과 연관되어 있는 확률값을 증가시킴으로써 변경할 수 있다.

```
>>> pmf.Incr(2, 0.2)
>>> pmf.Prob(2)
0.6
```

혹은 확률에 일정량을 곱할 수도 있다.

```
>>> pmf.Mult(2, 0.5)
>>> pmf.Prob(2)
0.3
```

만약 Pmf를 변경하면, 결과는 정규화되지 않을지도 모른다; 즉, 확률값을 다 합하면 1이 되지 않을지도 모른다. 확률값을 합한 결과를 반환하는데 사용되는 `Total` 메쏘드를 호출해서 확인한다.

```
>>> pmf.Total()
0.9
```

다시 정규화하기 위해서, `Normalize`를 호출한다:

```
>>> pmf.Normalize()
>>> pmf.Total()
1.0
```

Pmf 객체는 `Copy` 메쏘드를 제공하는데, 이를 통해서 원본에 영향을 주지않고, 사본을 만들고 변경 작업을 할 수 있다.

이번 절에 표기법이 일관성을 갖지 않는 것처럼 보일지도 모르지만 체계가 있다; Pmf를 클래스 명칭으로, pmf는 클래스의 인스턴스로, PMF는 확률질량함수에 대한 수학적 개념으로 각각을 표기하는데 사용한다.

그림 3.1: PMF of pregnancy lengths for first babies and others, using bar graphs and step functions.

제 2 절 PMF 플롯으로 그리기

thinkplot은 Pmf 플롯을 그리는 두가지 방식을 제공한다.

- Pmf를 막대그래프로 그리기 위해서 thinkplot.Hist을 사용한다. 만약 Pmf에 값의 개수가 작다면 막대그래프가 가장 유용한다.
- 계단 함수로 Pmf를 그래프 그리기 위해서는, thinkplot.Pmf을 사용할 수 있다. 만약 값이 많고, Pmf가 매끄럽다면(smooth) 탁월한 선택이 된다. 이 함수는 Hist 객체에서도 동작한다.

추가로, pyplot은 hist 함수를 제공하는데 값(value) 시퀀스를 받아서 히스토그램을 계산하고, 그래프로 그린다. Hist 객체를 사용하기 때문에, pyplot.hist은 사용하지 않는다.

그림 3.1은 막대그래프(왼쪽)와 계단함수(오른쪽)를 사용해서 첫째 아이와 첫째가 아닌 아이에 대한 임신기간 PMF를 보여준다.

히스토그램 대신에 PMF 플롯을 그려서, 표본차이로 오도되지 않고 두 분포를 비교할 수 있다. 그림을 해석하면, 첫번째 아이는 다른 아이들보다 정시(39주차)에 출산하지 않고, 늦게(41, 42주차) 출산할 것 같다.

그림 3.1을 생성하는 코드가 다음에 있다:

```
thinkplot.PrePlot(2, cols=2)
thinkplot.Hist(first_pmf, align='right', width=width)
thinkplot.Hist(other_pmf, align='left', width=width)
thinkplot.Config(xlabel='weeks',
                 ylabel='probability',
                 axis=[27, 46, 0, 0.6])

thinkplot.PrePlot(2)
thinkplot.SubPlot(2)
thinkplot.Pmfs([first_pmf, other_pmf])
thinkplot.Show(xlabel='weeks',
               axis=[27, 46, 0, 0.6])
```

PrePlot 메쏘드는 옵션 매개변수 rows와 cols을 받아서 그림 격자(grid)를 만드는데 이경우 한 행에 그림 두개를 넣는 격자가 된다. 첫번째 그림(왼쪽)은 thinkplot.Hist을 사용해서 앞에서 봤던 Pmf를 화면에 출력한다.

그림 3.2: Difference, in percentage points, by week.

PrePlot에 두번째 호출로 색깔 생성자를 초기화한다. 그리고 나서 SubPlot이 두번째 그림(오른쪽)으로 바꿔서, thinkplot.Pmfs을 사용해서 Pmf를 화면에 출력한다. axis을 사용해서 그림 두개 모두 동일한 축(axis)에 놓여지도록 확실히 한다. 그림 두개를 비교하려고 한다면, 축을 통일하는 것이 좋다.

제 3 절 다른 시각화 방법

히스토그램과 PMF은 데이터를 탐색하고 패턴과 관계를 식별하는데 유용하게 사용된다. 데이터에서 무슨 정보가 담겨져있고, 어떻게 돌아가는지 아이디어를 얻게 된다면, 다음 단계는 최대한 깔끔하게 식별한 패턴화할 수 있는 시각화 설계하는 것이다.

NSFG데이터에서 분포에서 가장 큰 차이는 모드(최빈치)에 있다. 그래서, 그래프에서 이 부분만을 확대하여 들여다보고, 차이를 강조하도록 자료를 변환하는 것이 의미가 있다.

```
weeks = range(35, 46)
diffs = []
for week in weeks:
    p1 = first_pmf.Prob(week)
    p2 = other_pmf.Prob(week)
    diff = 100 * (p1 - p2)
    diffs.append(diff)

thinkplot.Bar(weeks, diffs)
```

상기 코드에서 weeks가 임신주 범위다; diffs는 퍼센트에서 두 PMF간 차이가 된다. 그림 3.2는 막대그래프로 결과를 보여준다. 그림이 패턴을 좀더 명확하게 한다: 첫째 아이는 임신 39주차에 덜 태어날 것 같고, 임신 41, 42주차에 더 태어날 것 같다.

지금까지 결론을 잠정적으로 내렸다. 동일한 데이터셋을 사용해서 명백하게 차이를 식별하고 나서 차이를 명확하게 만드는 시각화 방식을 선택했다. 효과 차이가 실질적인지 확실하다고 할 수는 없다; 확률변동(random variation)일 수도 있다. 나중에 이 문제를 다시 다룰 것이다.

제 4 절 학급 크기 패러독스 (class size paradox)

진도를 더 나가기 전에, Pmf 객체를 가지고 할 수 있는 한가지 계산(computation)을 시연하고자 한다; 다음 예제를 "학급 크기 패러독스(class size paradox)"라고 명명한다.

많은 미국 대학에서, 학생대교수 비율은 약 10:1이 된다. 하지만 종종 학생들이 평균 학급크기가 10보다 큰 것을 발견하고 놀라곤 한다. 불일치에 대한 두가지 이유가 있다.

- 학생들이 학기당 일반적으로 4–5 과목을 수강하지만, 교수는 1 혹은 2 교과목만 가르친다.
- 적은 학급 수업을 즐기는 학생 숫자는 적지만, 큰 학급 수업에 학생수는 많다.

첫 이유는 명확하지만, 두번째 이유는 다소 모호하다. 사례를 살펴보자. 대학에서 다음과 같은 학급 크기 분포로 한 학기에 65 교과목을 개설한다고 가정하자.

size	count
5- 9	8
10-14	8
15-19	14
20-24	4
25-29	6
30-34	12
35-39	8
40-44	3
45-49	2

만약 학교 총장에게 평균 학급크기를 물어본다면, 총장은 PMF를 생성하고, 평균을 계산하고 나서 학급 평균 크기가 23.7이라고 보고한다. 다음에 코드가 있다.

```
d = { 7: 8, 12: 8, 17: 14, 22: 4,
      27: 6, 32: 12, 37: 8, 42: 3, 47: 2 }

pmf = thinkstats2.Pmf(d, label='actual')
print('mean', pmf.Mean())
```

하지만, 학생집단을 대상으로 수업에 학생이 있는지 물어보고, 평균을 계산한다면, 평균 학급크기가 더 크다고 생각할 것이다. 얼마나 더 큰지 살펴보자.

먼저, 학생들이 관측한 분포를 계산하자. 여기서 각 학급 크기와 연관된 확률은 학급에 있는 학생으로 "편의(bias)"가 있다.

그림 3.3: Distribution of class sizes, actual and as observed by students.

```
def BiasPmf(pmf, label):
    new_pmf = pmf.Copy(label=label)

    for x, p in pmf.Items():
        new_pmf.Mult(x, x)

    new_pmf.Normalize()
    return new_pmf
```

각 학급 크기 x마다, 확률값에 학급 크기를 관측한 학생수 x를 곱한다. 결과는 편의분포를 나타내는 새로운 Pmf가 된다.

이제 실제와 관측된 분포 모두를 플롯 그래프로 그릴 수 있다.

```
    biased_pmf = BiasPmf(pmf, label='observed')
    thinkplot.PrePlot(2)
    thinkplot.Pmfs([pmf, biased_pmf])
    thinkplot.Show(xlabel='class size', ylabel='PMF')
```

그림 3.3은 결과를 보여준다. 편의된 분포에서 작은 학급은 더 작고, 큰 학급은 더 많다. 편의된 분포 평균은 29.1로 실제 평균값보다 약 25%더 많다.

이 연산을 거꾸로 하는 것도 또한 가능하다. 대학 학급 크기 분포를 알고자 한다고 가정하자. 하지만, 대학 총장으로부터 신뢰성 있는 자료를 얻을 수는 없다. 대안은 무작위 학생 표본을 골라 학급에 학생수가 얼마인지 설문하는 것이다.

결과는 앞선 살펴봤던 이유로 편의가 있을지 모르지만, 이것을 사용해서 실제 분포를 추정한다. 다음에 Pmf 불편의(unbiased) 함수가 있다.

```
def UnbiasPmf(pmf, label):
    new_pmf = pmf.Copy(label=label)

    for x, p in pmf.Items():
        new_pmf.Mult(x, 1.0/x)

    new_pmf.Normalize()
    return new_pmf
```

`BiasPmf` 함수와 비슷하다; 유일한 차이점은 곱하는 대신에 각 확률값을 x로 나눈다는 것이다.

제 5 절 데이터프레임 인덱싱 (DataFrame indexing)

4 절에서 판다스 데이터프레임을 읽고, 데이터프레임을 사용해서 데이터 열(칼럼)을 선택하고 변경했다. 이제 행선택을 살펴보자. NumPy 난수 행렬을 생성하고, 이것을 사용해서 데이터프레임으로 초기화한다.

```
>>> import numpy as np
>>> import pandas
>>> array = np.random.randn(4, 2)
>>> df = pandas.DataFrame(array)
>>> df
          0         1
0 -0.143510  0.616050
1 -1.489647  0.300774
2 -0.074350  0.039621
3 -1.369968  0.545897
```

초기 설정값(by default)으로 행과 열 모두 0에서 시작하는 숫자로 번호가 매겨진다. 하지만, 열이름을 지정할 수 있다.

```
>>> columns = ['A', 'B']
>>> df = pandas.DataFrame(array, columns=columns)
>>> df
          A         B
0 -0.143510  0.616050
1 -1.489647  0.300774
2 -0.074350  0.039621
3 -1.369968  0.545897
```

행이름도 지정할 수 있다. 행이름 집합을 **index**라고 한다; 행이름 자체는 **labels**이라고 한다.

```
>>> index = ['a', 'b', 'c', 'd']
>>> df = pandas.DataFrame(array, columns=columns, index=index)
>>> df
          A         B
a -0.143510  0.616050
b -1.489647  0.300774
c -0.074350  0.039621
d -1.369968  0.545897
```

앞장에서 살펴보았듯이, 단순 인덱싱으로 열을 선택하면 시리즈를 반환한다.

```
>>> df['A']
a   -0.143510
b   -1.489647
```

```
c   -0.074350
d   -1.369968
Name: A, dtype: float64
```

레이블(label)로 행을 선택하려면 `loc` 속성을 사용하면 되고 시리즈를 반환한다.

```
>>> df.loc['a']
A   -0.14351
B    0.61605
Name: a, dtype: float64
```

레이블(label) 보다 행의 정수 위치정보를 알고 있다면, `iloc`속성을 사용하고, 실행 결과로 시리즈를 반환한다.

```
>>> df.iloc[0]
A   -0.14351
B    0.61605
Name: a, dtype: float64
```

또한 `loc`는 레이블 리스트를 인자로 받고, 결과로 데이터프레임을 반환한다.

```
>>> indices = ['a', 'c']
>>> df.loc[indices]
          A         B
a -0.14351  0.616050
c -0.07435  0.039621
```

마지막으로 레이블(label)로 행 범위를 선택하는데 슬라이스를 사용할 수 있다.

```
>>> df['a':'c']
           A         B
a -0.143510  0.616050
b -1.489647  0.300774
c -0.074350  0.039621
```

혹은 정수 위치정보를 사용할 수도 있다.

```
>>> df[0:2]
           A         B
a -0.143510  0.616050
b -1.489647  0.300774
```

어느 경우든지 관계없이 결과는 데이터프레임이 된다. 하지만 첫번째 결과는 슬라이스 끝값을 포함하지만, 두번째는 끝값을 포함하지 않는 것을 주목하라.

저자 충고: 만약 행에 단순히 정수값이 아닌 레이블(label)이 있다면, 레이블을 일관되게 사용하고 정수 위치정보 사용을 피하라.

제 6 절 연습 문제

Solutions to these exercises are in `chap03soln.ipynb` and `chap03soln.py`

Exercise 3.1 Something like the class size paradox appears if you survey children and ask how many children are in their family. Families with many children are more likely to appear in your sample, and families with no children have no chance to be in the sample.

Use the NSFG respondent variable `NUMKDHH` to construct the actual distribution for the number of children under 18 in the household.

Now compute the biased distribution we would see if we surveyed the children and asked them how many children under 18 (including themselves) are in their household.

Plot the actual and biased distributions, and compute their means. As a starting place, you can use `chap03ex.ipynb`.

Exercise 3.2 In Section 7 we computed the mean of a sample by adding up the elements and dividing by n. If you are given a PMF, you can still compute the mean, but the process is slightly different:

$$\bar{x} = \sum_i p_i \, x_i$$

where the x_i are the unique values in the PMF and $p_i = PMF(x_i)$. Similarly, you can compute variance like this:

$$S^2 = \sum_i p_i \, (x_i - \bar{x})^2$$

Write functions called `PmfMean` and `PmfVar` that take a Pmf object and compute the mean and variance. To test these methods, check that they are consistent with the methods `Mean` and `Var` provided by Pmf.

Exercise 3.3 I started with the question, "Are first babies more likely to be late?" To address it, I computed the difference in means between groups of babies, but I ignored the possibility that there might be a difference between first babies and others *for the same woman*.

To address this version of the question, select respondents who have at least two babies and compute pairwise differences. Does this formulation of the question yield a different result?

Hint: use `nsfg.MakePregMap`.

Exercise 3.4 In most foot races, everyone starts at the same time. If you are a fast runner, you usually pass a lot of people at the beginning of the race, but after a few miles everyone around you is going at the same speed.

When I ran a long-distance (209 miles) relay race for the first time, I noticed an odd phenomenon: when I overtook another runner, I was usually much faster, and when another runner overtook me, he was usually much faster.

At first I thought that the distribution of speeds might be bimodal; that is, there were many slow runners and many fast runners, but few at my speed.

Then I realized that I was the victim of a bias similar to the effect of class size. The race was unusual in two ways: it used a staggered start, so teams started at different times; also, many teams included runners at different levels of ability.

As a result, runners were spread out along the course with little relationship between speed and location. When I joined the race, the runners near me were (pretty much) a random sample of the runners in the race.

So where does the bias come from? During my time on the course, the chance of overtaking a runner, or being overtaken, is proportional to the difference in our speeds. I am more likely to catch a slow runner, and more likely to be caught by a fast runner. But runners at the same speed are unlikely to see each other.

Write a function called `ObservedPmf` that takes a Pmf representing the actual distribution of runners' speeds, and the speed of a running observer, and returns a new Pmf representing the distribution of runners' speeds as seen by the observer.

To test your function, you can use `relay.py`, which reads the results from the James Joyce Ramble 10K in Dedham MA and converts the pace of each runner to mph.

Compute the distribution of speeds you would observe if you ran a relay race at 7.5 mph with this group of runners. A solution to this exercise is in `relay_soln.py`.

제 7 절 용어사전

- 확률질량함수 (Probability mass function, PMF): 값을 확률로 매핑하는 함수로 분포를 표현.

- 확률 (probability): 표본크기의 일부로 표현되는 빈도수.
- 정규화 (normalization): 확률값을 얻기 위해서 표본 크기로 빈도수를 나누는 과정.
- 인덱스 (index): 판다스 데이터프레임에서, 인덱스는 행 레이블(label)을 포함하는 특별한 열(칼럼)이다.

제 4 장

누적분포함수

이번 장에서 사용되는 코드는 `cumulative.py`에 있다. 코드를 다운로드하고 작업하는 것에 대한 정보는 2을 참조한다.

제 1 절 PMF의 한계점

PMF는 값(value)의 수가 작다면 잘 동작한다. 하지만, 값의 갯수가 증가함에 따라, 각 값과 연관된 확률값이 더 작아지고 확률잡음(random noise)의 효과는 증가한다.

예를 들어, 출산 체중 분포에 관심이 있다고 하자. NSFG 데이터에서 `totalwgt_lb` 변수가 파운드로 출생 체중을 기록한다. 그림 4.1이 첫번째 아이와 첫째가 아닌 아이에 대한 체중값을 PMF로 보여준다.

전반적으로 분포는 정규분포의 종모양을 닮았다. 평균 근처에 값이 많고 체중이 더 높거나 더 낮아지면 값이 작아진다.

하지만 그림을 이해하기는 어렵다. 뾰족한 것과 골자기가 많고, 두 집단 분포 사이에 명백한 차이도 보인다. 여럿중에서 어느 면이 유의미한지 분간하기는 쉽지 않다. 또한 전반적인 패턴을 보기도 어렵다; 예를 들어, 여러분이 보기에 어느 분포가 평균값이 더 높은가?

데이터를 구간(bin)에 담는 것으로 이러한 문제는 완화될 수 있다; 즉, 값 범위를 서로 겹쳐지지 않는 구간으로 나누고 각 구간(bin)에 값 갯수를 계수한다. 구간에

그림 4.1: PMF of birth weights. This figure shows a limitation of PMFs: they are hard to compare visually.

담는 것(binning)은 유용하지만, 적정한 구간 크기를 잡는 것은 까다롭다. 잡음을 평활(smooth out)하기 위해서 충분히 큰 통을 잡는 것이 또한 유용한 정보도 평활할 수도 있다.

이러한 문제를 회피하는 대안이 누적분포함수(cumulative distribution function, CDF)로 이번 장 학습주제다. 하지만, CDF를 설명하기 전에 백분위수(percentile)를 먼저 설명해야 한다.

제 2 절 백분위수 (Percentiles)

만약 전국 단위 표준시험을 치르게 되면, 원점수와 **백분위 순위(percentile rank)** 형태로 시험결과를 받아보게 된다. 이러한 맥락에서 백분위 순위는 시험 당사자보다 적은 점수를 얻는 사람들이 된다. 그래서 만약 "백분위수 90번째"라면, 시험을 치른 90% 사람보다 혹은 동등하다는 의미가 된다.

다음에 `scores` 시퀀스 값에서 상대적으로 `your_score` 값에 대한 백분위 순위를 계산하는 방법이 있다.

```
def PercentileRank(scores, your_score):
    count = 0
    for score in scores:
        if score <= your_score:
            count += 1

    percentile_rank = 100.0 * count / len(scores)
    return percentile_rank
```

예제로 만약 시퀀스 점수가 55, 66, 77, 88, 99이고, 시험 점수로 88점을 받았다면, 백분위 순위는 `100 * 4 / 5`, 80이 된다.

값이 주어진다면, 백분위 순위를 찾기는 쉽다; 반대 방향으로는 다소 더 어렵다. 만약 백분위 순위가 주어진 상태에서 해당하는 값을 찾고자 한다면, 한 선택지는 값을 정렬하고 원하는 값을 찾는 것이다.

```
def Percentile(scores, percentile_rank):
    scores.sort()
    for score in scores:
        if PercentileRank(scores, score) >= percentile_rank:
            return score
```

계산 결과는 **백분위수(percentile)**가 된다. 예를 들어, 50번째 백분위 수는 백분위 순위가 50 인 값이 된다. 시험점수 분포에서 50번째 백분위 수는 77이다.

`Percentile` 구현코드가 그다지 효율적이지 않다. 더 나은 접근법은 백분위 순위를 사용해서 해당하는 백분위수 인덱스를 계산하는 것이다.

```
def Percentile2(scores, percentile_rank):
    scores.sort()
    index = percentile_rank * (len(scores)-1) // 100
    return scores[index]
```

"백분위수 (percentile)"와 "백분위 순위 (percentile rank)" 차이가 혼동스러울 수 있고 항상 용어를 정확하게 구별하여 사용하지는 않는다. 요약하면, `PercentileRank` 함수는 값을 인자로 받아 값 집합에서 백분위 순위를 계산한다; `Percentile` 함수는 백분위 순위를 인자로 받아 해당하는 값을 계산한다.

제 3 절 CDF

이제 백분위수와 백분위 순위를 이해하고 있기 때문에, **누적분포함수(cumulative distribution function, CDF)**를 다룰 준비가 되었다. CDF는 값을 백분위 순위로 매핑하는 함수다.

CDF는 x의 함수로 x는 분포에 나타나는 임의값이다. 특정한 x 값에 대해서 CDF(x)를 평가하기 위해서, x와 동일하거나 작은 분포의 분수값을 계산한다.

시퀀스 `sample`와 값 `x`를 인자로 받는 함수로 어떤 느낌인지 다음에 코드가 있다.

```
def EvalCdf(sample, x):
    count = 0.0
    for value in sample:
        if value <= x:
            count += 1

    prob = count / len(sample)
    return prob
```

함수가 거의 `PercentileRank`과 동일하지만, 백분위 순위가 0–100인 반면에 결과값이 0–1 범위를 갖는 확률이라는 점이 차이가 있다.

예제로 표본값으로 `[1, 2, 2, 3, 5]`을 수집했다고 가정하자. 다음에 CDF로부터 값이 몇개 있다.

$$CDF(0) = 0$$

$$CDF(1) = 0.2$$

$$CDF(2) = 0.6$$

$$CDF(3) = 0.8$$

$$CDF(4) = 0.8$$

$$CDF(5) = 1$$

그림 4.2: Example of a CDF.

그림 4.3: CDF of pregnancy length.

표본에 있는 값뿐만 아니라 x의 임의값에 대해서 CDF를 평가할 수 있다. 만약 x가 표본에 가장 작은 값보다 작다면, CDF(x)는 0. 만약 x가 가장 큰 값보다 크다면, CDF(x)는 1.

그림 4.2이 CDF를 그래픽으로 표현한 것이다. 표본 CDF는 계단 함수다.

제 4 절 CDF 표현하기 (Representing CDFs)

`thinkstats2`은 CDF를 표현하는 Cdf라는 클래스를 제공한다. Cdf 가 제공하는 기본 메쏘드는 다음과 같다.

- `Prob(x)`: `x` 값이 주어졌을 때, $p = \text{CDF}(x)$ 확률값을 계산한다. 꺾쇠 연산자는 `Prob`와 동일하다.
- `Value(p)`: 확률 `p`가 주어졌을 때, 상응하는 값 `x`를 계산한다; 즉, `p`의 **CDF 역함수(inverse CDF)**다.

Cdf 생성자는 인자로 리스트, 판다스 시리즈, Hist, Pmf, 혹은 또다른 Cdf를 받을 수 있다. NSFG 데이터에서 임신 기간 분포에 대한 Cdf를 생성하는 코드가 다음에 있다.

```
live, firsts, others = first.MakeFrames()
cdf = thinkstats2.Cdf(live.prglngth, label='prglngth')
```

`thinkplot`은 `Cdf`라는 함수를 제공해서 Cdf를 선그래프를 그릴 수 있다.

```
thinkplot.Cdf(cdf)
thinkplot.Show(xlabel='weeks', ylabel='CDF')
```

그림 4.3에 결과가 있다. CDF를 읽는 한가지 방법은 백분위수를 찾는 것이다. 예를 들어, 임신 기간 10%는 36주차보다 더 짧고, 90%는 41주차보다 더 짧은 것처럼 보인다. 또는 CDF를 통해서 분포 모양을 시각적으로 표현할 수도 있다. 흔한 값은 CDF에서 급격하거나 수직적 부분으로 나타난다; 이번 예제에서 39주차 모드(최빈값)가 명확히 보인다. 30주차 밑으로 값이 몇개 없어서 이 범위에 있는 CDF는 평평하다.

CDF에 익숙해지는데 시간이 좀 필요하다. 하지만, 한번 익숙해지면, PMF보다 더 많은 정보를 좀더 명확하게 보여줄 것으로 생각된다.

그림 4.4: CDF of birth weights for first babies and others.

제 5 절 CDF 비교하기

CDF는 특히 분포를 비교하는데 유용하다. 예를 들어, 첫째 아이와 첫째 아이가 아닌 아이에 대한 출산 체중 CDF를 플롯으로 그리는 코드가 다음에 있다.

```
first_cdf = thinkstats2.Cdf(firsts.totalwgt_lb, label='first')
other_cdf = thinkstats2.Cdf(others.totalwgt_lb, label='other')

thinkplot.PrePlot(2)
thinkplot.Cdfs([first_cdf, other_cdf])
thinkplot.Show(xlabel='weight (pounds)', ylabel='CDF')
```

그림 4.4에 결과가 있다. 그림 4.1와 비교하여, 좀더 명확하게 분포 모양과 분포 간의 차이를 그림에서 보여준다. 첫째 아이 체중이 평균 이상에서 조금더 커다란 불일치성을 보이고, 분포 전반에 걸쳐 다소 가볍다는 것을 볼 수 있다.

제 6 절 백분위수 기반 통계량 (Percentile-based statistics)

CDF를 계산하게 되면, 백분위수와 백분위 순위를 계산하기는 쉽다. Cdf 클래스가 두가지 메쏘드를 제공한다.

- `PercentileRank(x)`: x가 주어지면, $100 \cdot \text{CDF}(x)$ 백분위 순위를 계산한다.
- `Percentile(p)`: 백분위 순위 rank가 주어지면, 해당하는 값 x를 계산한다. `Value(p/100)`과 동등하다.

백분위수 (`Percentile`)는 백분위수 기반 요양 통계량을 계산하는데 사용될 수 있다. 예를 들어 50번째 백분위수는 **중위수 (median)**로 알려진 분포를 반으로 나누는 값이다. 평균과 마찬가지로 중위수는 분포의 중심경향성을 측정하는 측도다.

사실, 각기 다른 특성을 가진 "중위수(median)"에 대한 정의가 몇개 있다. 하지만, `Percentile(50)`가 단순하고 계산하기 효율적이다.

또 다른 백분위수 기반 통계량이 **사분위수 범위 (interquartile range, IQR**로 분포의 퍼짐을 측정하는 측도다. IQR는 75번째와 25번째 백분위수 간의 차이다.

그림 4.5: CDF of percentile ranks for a random sample of birth weights.

좀더 일반적으로, 백분위수는 종종 분포 모양을 요약하는데 쓰여진다. 예를 들어, 수입 분포는 종종 "분위수 (quintiles)"로 보고된다; 즉, 20번째, 40번째, 60번째, 80번째 백분위수로 쪼개진다. 다른 분포는 10개 "십분위(deciles)"으로 나눠진다. 이와 같이 CDF에서 동일간격으로 표현되는 통계량을 **분위수(quantiles)**라고 한다. 좀더 자세한 정보는 다음 웹사이트를 참고 바란다. `https://en.wikipedia.org/wiki/Quantile`.

제 7 절 난수 (Random numbers)

정상 출산 모집단에서 임의 표본을 추출하고, 출생 체중 백분위 순위를 찾아낸다고 가정하자. 이제 백분위 순위 CDF를 계산한다고 가정하자. 분포가 어떨 것으로 생각하는가?

다음에 어떻게 계산하는지 코드가 있다. 첫째로 출생 체중 Cdf를 생성한다.

```
weights = live.totalwgt_lb
cdf = thinkstats2.Cdf(weights, label='totalwgt_lb')
```

그리고 나서, 표본을 생성하고, 표본에 있는 각 값에 대한 백분위 순위를 계산한다.

```
sample = np.random.choice(weights, 100, replace=True)
ranks = [cdf.PercentileRank(x) for x in sample]
```

`sample`은 100개 출생 체중 임의 표본이며 복원 추출하였다; 즉, 동일한 값이 한번이상 추출될 수 있다. `ranks`는 백분위 순위 리스트다.

마지막으로 백분위 순위 Cdf를 만들고 플롯으로 그린다.

```
rank_cdf = thinkstats2.Cdf(ranks)
thinkplot.Cdf(rank_cdf)
thinkplot.Show(xlabel='percentile rank', ylabel='CDF')
```

그림 4.5이 결과를 보여준다. CDF가 근사적으로 직선이다. 분포가 균등하다라는 의미다.

결과가 명확하지 않을 수도 있지만, CDF가 정의된 방식의 결과다. 그림이 보여주는 정보는 표본의 10%가 10번째 백분위수 보다 밑에 있고, 표본의 20%가 20번째 백분위수 보다 밑에 있고 등등, 정확히 예측했던 것이다.

그래서, CDF 모양에 관계없이, 백분위 순위 분포는 균등하다. 이 속성이 유용한데, 이유는 주어진 CDF에서 난수를 생성하는데 있어서 간단하면서도 효율적인 알고리즘 설계하는 기초가 되기 때문이다; 다음에 난수를 생성하는 방법이 있다.

- 0–100 범우에서 균등하게 백분위 순위를 고른다.
- `Cdf.Percentile`를 사용해서 선택한 백분위 순위에 상응하는 값을 분포에서 찾는다.

Cdf는 상기 알고리즘을 구현한 것으로 `Random`이 함수명이다.

```
# class Cdf:
    def Random(self):
        return self.Percentile(random.uniform(0, 100))
```

Cdf는 `Sample` 메쏘드를 제공하는데 정수 `n`을 인자로 받아 Cdf에서 임의로 추출한 `n`개 리스트를 반환한다.

제 8 절 백분위 순위 비교하기

백분위 순위는 다른 집단에 대해서 측정값을 비교하는데 유용하다. 예를 들어, 달리기 경주에서 참가자는 대체로 나이와 성별로 무리를 만든다. 다른 연령 집단에 있는 사람을 비교하는데 경주시간을 백분위 순위로 변환할 수 있다.

몇년전에 매사추세츠(Massachusetts)주에서 제임스 조이스(James Joyce) 10킬로 마라톤을 뛰었다; 42분 44초로 주파해서 1633명중에서 97번째로 완주했다. 참가자 1633명 중 1537명 참가자보다 빨리 도착해서 저자의 최종 백분위 순위는 94%다.

좀더 일반적으로, 위치와 필드 크기 정보가 주어진다면, 백분위 순위를 계산할 수 있다.

```
def PositionToPercentile(position, field_size):
    beat = field_size - position + 1
    percentile = 100.0 * beat / field_size
    return percentile
```

“40에서 49세 남성”, M4049로 표기된 저자가 속한 연령집단, 256명 중에서 26번째로 완주했다. 그래서, 저자가 속한 연령집단에서 백분위 순위는 90%가 된다.

10년정도 더 마라톤을 뛴다면 (그리고 계속해서 뛰고 싶다.), M5059 그룹에 있을 것이다. 저자가 속한 집단에서 동일한 백분위수를 유지한다고 가정한다면, 완주하는데 얼마나 더 시간이 필요할까?

M4049 집단에 있는 저자의 백분위 순위를 M5059 집단에 위치로 전환하면 상기 질문에 대답할 수 있다. 다음에 프로그램 코드가 있다.

```
def PercentileToPosition(percentile, field_size):
    beat = percentile * field_size / 100.0
    position = field_size - beat + 1
    return position
```

M5059 집단에 171 명이 있어서 동일한 백분위 순위를 유지하려면 17번째와 18번째 사이에서 완주해야 한다. M5059에서 17번째 마라토너가 46:05로 완주해서, 40대 동일한 백분위 순위를 유지하는데 46:05 시간이 완주 목표시간이 된다.

제 9 절 Exercises

For the following exercises, you can start with `chap04ex.ipynb`. My solution is in `chap04soln.ipynb`.

Exercise 4.1 How much did you weigh at birth? If you don't know, call your mother or someone else who knows. Using the NSFG data (all live births), compute the distribution of birth weights and use it to find your percentile rank. If you were a first baby, find your percentile rank in the distribution for first babies. Otherwise use the distribution for others. If you are in the 90th percentile or higher, call your mother back and apologize.

Exercise 4.2 The numbers generated by `random.random` are supposed to be uniform between 0 and 1; that is, every value in the range should have the same probability.

Generate 1000 numbers from `random.random` and plot their PMF and CDF. Is the distribution uniform?

제 10 절 용어사전

- 백분위 순위 (percentile rank): 분포에서 주어진 값과 동일하거나 적은 값의 퍼센티지.
- 백분위수 (percentile): 주어진 백분위 순위와 연관된 값.
- 누적분포함수 (cumulative distribution function, CDF): 값에서 누적확률값으로 매핑하는 함수. CDF(x)는 x와 동일하거나 작은 표본비율이다.
- 역 CDF (inverse CDF): 누적 확률(p)에서 해당값으로 매핑하는 함수.
- 중위수 (median): 종종 중심경향성 측도로 사용되는 50번째 백분위수.

- 사분위 범위 (interquartile range): 퍼짐의 측도로 사용되는 75번째와 25번째 백분위수 간 차이.
- 분위수 (quantile): 동일 간격 백분위 순위에 상응하는 스퀀스 값; 예를 들어, 분포 사분위수는 25번째, 50번째, 75번째 백분위수가 된다.
- 복원 (replacement): 표본추출 과정의 속성. "복원추출 (With replacement)"는 동일한 값이 한번이상 추출될 수 있다는 의미다; "비복원추출 (Without replacement)"는 값이 한번 추출될면, 모집단에서 제거된다는 의미가 된다.

제 5 장

분포 모형화 (Modeling distributions)

지금까지 사용한 분포는 **경험적 분포 (empirical distributions)**라고 부른다. 이 유는 필연적으로 유한 표본인 경험적 관측치에 기반하고 있기 때문이다.

수학 함수인 CDF로 특징 지어지는 **해석 분포 (analytic distribution)**가 대안이 된다. 해석 분포가 경험적 분포를 모형화하는데 사용될 수 있다. 이러한 맥락에서 **모형(model)**은 불필요한 부분을 덜어낸 단순화가 된다. 이번 장에서 자주 사용되는 분포를 제시하고 이를 사용하여 다양한 출처를 가진 데이터를 모형화한다.

이번 장에서 사용되는 코드는 `analytic.py`에 있다. 코드를 다운로드하고 작업하는 것에 대한 정보는 2을 참조한다.

제 1 절 지수분포 (exponential distribution)

지수 분포 (exponential distribution)로 시작하는데 이유는 상대적으로 단순하기 때문이다. 지수분포 CDF는 다음과 같다.

$$\text{CDF}(x) = 1 - e^{-\lambda x}$$

모수 λ가 분포 형상(shape)을 결정한다. 그림 5.1에서 $\lambda = 0.5, 1, 2$ 값을 가진 CDF가 대략 모양이 어떤지 볼 수 있다.

그림 5.1: CDFs of exponential distributions with various parameters.

그림 5.2: CDF of interarrival times (left) and CCDF on a log-y scale (right).

현실 세계에서 일련의 사건을 보고, 사건 간에 시간(**도착간격 시간, interarrival times**)을 측정할 때 지수분포가 등장한다. 만약 사건이 언제든지 균등하게 발생할 것 같다면 도착간격 시간 분포는 지수분포같은 경향이 있다.

일례로, 출생간 발생시간을 살펴보자. 1997년 12월 18일 호주 브리즈번 [1] 에서 44명 신생아가 출생했다. 모든 44명 신생아 출생 시간이 지역신문에 출간되었다; 전체 데이터셋은 `ThinkStats2` 저장소 `babyboom.dat` 파일에 담겨있다.

```
df = ReadBabyBoom()
diffs = df.minutes.diff()
cdf = thinkstats2.Cdf(diffs, label='actual')

thinkplot.Cdf(cdf)
thinkplot.Show(xlabel='minutes', ylabel='CDF')
```

`ReadBabyBoom` 함수가 데이터 파일을 읽어들이고 `time`, `sex`, `weight_g`, `minutes` 칼럼으로 구성된 데이터프레임을 반환한다. 여기서 `minutes`가 자정 이후 출생시간을 분으로 변환한 시간정보를 담고 있다.

`diffs`는 연속되는 출생시간 사이 차이가 되고 `cdf`는 출생간격 시간 분포가 된다. 그림 5.2 (왼편)이 CDF를 나타낸다. 전형적인 지수분포 형상을 지닌 처럼 보이지만, 어떻게 분간할 수 있을까?

한 방법은 **보완 CDF (complementary CDF)**를 플롯으로 그리는 것이다. 보완 CDF는 log-y 척도로 $1 - \text{CDF}(x)$이다. 지수분포 데이터에 대해서는 결과가 직선이다. 왜 그런지 살펴보자.

독자가 생각하기에 지수분포를 따르는 데이터셋을 보완 CDF(CCDF) 플롯으로 그리면, 다음과 같은 함수가 나올 것으로 기대한다.

$$y \approx e^{-\lambda x}$$

양변에 로그를 취하면 다음과 같다.

$$\log y \approx -\lambda x$$

그래서, log-y 척도로 CCDF는 기울기 $-\lambda$인 직선이 된다. 다음에 플롯을 생성하는 방법이 있다.

```
thinkplot.Cdf(cdf, complement=True)
```

[1] 예제에 나오는 자료와 정보는 저널 논문에 기반한다. Dunn, "A Simple Dataset for Demonstrating Common Distributions," Journal of Statistics Education v.7, n.3 (1999)

그림 5.3: CDF of normal distributions with a range of parameters.

```
thinkplot.Show(xlabel='minutes',
               ylabel='CCDF',
               yscale='log')
```

`complement=True` 인자가 있어서, `thinkplot.Cdf`이 플롯을 그리기 전에 보완 CDF를 계산한다. 그리고 `yscale='log'`를 통해서 `thinkplot.Show`가 로그 척도로 y축을 고정한다.

그림 5.2 (오른편)에 결과가 있다. 정확하게 직선이 아니다. 이 데이터에 대해서 완벽한 모델로 지수 분포가 아니라는 것이 표시된다. 기본 가정—출생이 아무 때고 균등하게 발생—이 정확하게 사실이 아닐 것이다. 그럼에도 불구하고 지수 분포로 이 데이터셋을 모형화하는 것이 합리적일 것이다. 이와 같은 단순화로 단 하나의 모수로 분포를 요약할 수 있다.

모수 λ가 율(rate)로 해석될 수 있다; 즉, 평균적으로 단위 시간에 발생하는 사건 수. 예제에서 44명의 신생아가 24시간내에 태어난다. 그래서 율값이 분당 $\lambda = 0.0306$이 된다. 지수분포 평균은 $1/\lambda$ 으로 신생아 간에 출생 평균 시간은 32.7 분이 된다.

제 2 절 정규 분포 (normal distribution)

가우스 분포(Gaussian distribution)라고도 불리는 **정규 분포 (normal distribution)**가 흔히 사용되는데 이유는 많은 현상을 기술하고 최소한 근사적으로도 기술할 수 있기 때문이다. 4 절에서 다루게 되는데 이와 같은 보편성에는 이유가 있다.

$$\mathrm{CDF}(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-t^2/2} dt$$

정규 분포는 모수 두개로 특성화된다: 평균 μ, 표준편차 σ. 모수 $\mu = 0$과 $\sigma = 1$을 갖는 정규분포를 **표준 정규 분포 (standard normal distribution)**라고 한다. 정규분포 CDF는 닫힌 형식 해법(closed form solution)을 갖지 않는 적분으로 정의된다. 하지만, 효율적으로 계산하는 알고리즘이 있다. 알고리즘 중 하나가 SciPy을 통해 제공된다: `scipy.stats.norm`이 정규분포를 표현하는 객체다. 표준 정규분포 CDF를 계산하는 `cdf` 메쏘드를 제공한다.

```
>>> import scipy.stats
>>> scipy.stats.norm.cdf(0)
0.5
```

그림 5.4: CDF of birth weights with a normal model.

결과값은 맞다: 표준 정규분포 중위수는 0 (평균과 같다)이고, 값의 절반이 중위수 아래 위치한다. 그래서 CDF(0)은 0.5 이다.

`norm.cdf`은 옵션 모수를 받는다: `loc`가 평균을 특정하고, `scale`는 표준편차를 특정한다.

`thinkstats2`는 상기 함수를 좀더 사용하기 쉽게 한다. `EvalNormalCdf` 메쏘드는 `mu`과 `sigma`을 인자로 받아 `x`에 CDF를 계산한다.

```
def EvalNormalCdf(x, mu=0, sigma=1):
    return scipy.stats.norm.cdf(x, loc=mu, scale=sigma)
```

그림 5.3에 정규분포에 다양한 모수를 넣어 그린 CDF가 있다. 곡선의 S자(sigmoid) 형상이 정규분포를 식별할 수 있게 하는 특성이다.

앞장에서 NSFG 출생 체중 분포를 살펴봤다. 모든 정상 출산 체중에 대한 경험적 CDF와 동일한 평균과 분산으로 정규분포 CDF를 중첩하여 플롯으로 그린 것이 그림 5.4이다.

정규분포가 이 데이터셋에 대해서 좋은 모형이 된다. 그래서 만약 모수 $\mu = 7.28$ 과 $\sigma = 1.24$으로 분포를 요약한다면, 결과 오차(모형과 데이터 간 차이)가 작다.

백분위수 10번째 아래에서 데이터와 모형 사이에 불일치가 있다. 정규분포에서 예측되는 것보다 더 많이 체중이 적은 아이가 있다. 만약 조산아에 특별히 관심이 있다면, 분포에서 이 부분을 잘 적합하는 것이 중요하다. 그래서 정규분포를 사용하는 것이 적절하지 않을 수도 있다.

제 3 절 정규확률그림 (Normal probability plot)

지수분포와 몇가지 분포에서 대해서 해석분포(analytic distribution)가 특정 데이터셋에 대해서 적합한 모형인가를 테스트하는데 사용할 수 있는 간단한 변환이 있다.

정규분포에 대해서 그러한 변환은 없다. 하지만, **정규확률그림 (normal probability plot)**으로 불리는 대안이 있다. 정규확률그림을 생성하는 방식이 두개 있다.: 어려운 방식과 쉬운 방식. 어려운 방식에 관심이 있다면 `https://en.wikipedia.org/wiki/Normal_probability_plot`에서 자세한 정보를 얻을 수 있다. 다음에 쉬운 방식이 있다.

1. 표본에 있는 값을 정렬한다.

그림 5.5: Normal probability plot for random samples from normal distributions.

2. 표준정규분포($\mu = 0, \sigma = 1$)에서 표본과 동일한 크기를 갖는 난수을 생성하고 정렬한다.

3. 표본에서 나온 정렬된 값과 난수를 플롯으로 그린다.

만약 표본 분포가 근사적으로 정규분포라면, 결과는 절편 `mu`, 기울기 `sigma`를 갖는 직선이다. `thinkstats2`에 `NormalProbability`이 있다. 표본을 인자로 받아서 넘파이(NumPy) 배열 두개를 반환한다.

```
xs, ys = thinkstats2.NormalProbability(sample)
```

`ys`는 `sample`에서 정렬된 값이 담겨있다; `xs`에는 표준정규분포에서 생성된 난수가 담겨있다.

`NormalProbability`을 테스트하기 위해서 다양한 모수를 가진 정규분포에서 모조 샘플을 생성했다. 그림 5.5에 결과가 있다. 선들이 근사적으로 직선으로, 평균에 있는 값보다 벗어난 값을 꼬리에 갖는다.

이제 실제 데이터에 적합을 시도해 보자. 앞절로부터 출생 체중 데이터에 대해 정규확률그림을 생성하는 코드가 다음에 있다. 모형을 표현하는 회색선과 실제 데이터를 표현하는 파란선을 플롯으로 그린다.

```
def MakeNormalPlot(weights):
    mean = weights.mean()
    std = weights.std()

    xs = [-4, 4]
    fxs, fys = thinkstats2.FitLine(xs, inter=mean, slope=std)
    thinkplot.Plot(fxs, fys, color='gray', label='model')

    xs, ys = thinkstats2.NormalProbability(weights)
    thinkplot.Plot(xs, ys, label='birth weights')
```

`weights`는 출생 체중 판다스 시리즈다; `mean`과 `std`은 각각 평균과 표준편차다.

`FitLine`이 시퀀스 `xs`, 절편, 기울기를 인자로 받는다; 반환하는 `xs`와 `ys`는 `xs` 값에서 계산되어 인자로 받은 모수를 가진 직선이다.

`NormalProbability`은 `xs`와 `ys`를 반환하는데 표준정규분포에서 나온 값과 `weights`에서 나온 값을 담고 있다. 만약 체중 분포가 정규분포를 따른다면, 데이터도 모델과 매칭되어야 한다.

그림 5.6: Normal probability plot of birth weights.

그림 5.6이 전체 정상 출생과 더불어 만삭(임신기간이 36주 이상)에 대한 결과를 보여준다. 두 곡선 모두 평균 근처에서 모형과 매칭되고 꼬리에서 차이가 난다. 가장 무거운 아이가 모형이 예측한 것보다 더 무겁고, 가장 가벼운 아이는 더 가볍다.

단지 만삭 아이만 선택해서, 가장 가벼운 몇몇 아이를 제거하면 분포 아래쪽 꼬리에 있는 불일치를 줄일 수 있다.

정규분포 모형이 평균에서부터 몇 표준편차 내에서 분포를 잘 기술하지만, 꼬리 부근에서는 아니라고 플롯 그래프를 통해서 알 수 있다. 실제 목적에 얼마나 부합되는지는 목적에 달려있다.

제 4 절 로그 정규분포 (lognormal distribution)

만약 값을 로그 취한 집합이 정규분포라면, 이 값은 **로그 정규분포 (lognormal distribution)**다. 로그 정규분포 CDF는 x를 $\log x$로 치환한 정규분포 CDF와 동일하다.

$$CDF_{lognormal}(x) = CDF_{normal}(\log x)$$

로그 정규분포 모수는 일반적으로 μ와 σ로 표기한다. 하지만, 기억할 것은 모수가 평균과 표준편차는 *아니다*; 로그 정규분포 평균은 $\exp(\mu + \sigma^2/2)$이고, 표준편차는 조금 복잡한다.(`http://wikipedia.org/wiki/Log-normal_distribution`를 참조한다)

만약 표본이 근사적으로 로그 정규분포이고 log-x 척도로 CDF 플롯을 그린다면, 정규분포 특성의 형상을 갖는다. 표본이 로그 정규분포 모형과 얼마나 잘 적합하는지 테스트하기 위해서, 표본값에 로그를 취해서 정규확률그림을 생성할 수 있다.

예제로 성인 체중 분포를 살펴보는데 근사적으로 로그 정규분포다.[2]

만성 질환 예방 및 건강 증진을 위한 국립 센터 (National Center for Chronic Disease Prevention and Health Promotion)에서는 행동 위험 요인 감시 시스템

[2] `http://mathworld.wolfram.com/LogNormalDistribution.html` 사이트에서 주석(인용없이)으로 이 가능성에 대해서 제보를 받았다. 나중에 로그 변환과 원인을 제안하는 논문을 발견했다: Penman and Johnson, "The Changing Shape of the Body Mass Index Distribution Curve in the Population," Preventing Chronic Disease, 2006 July; 3(3): A74. Online at `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1636707`.

그림 5.7: Normal probability plots for adult weight on a linear scale (left) and log scale (right).

(Behavioral Risk Factor Surveillance System, BRFSS)[3]의 일부문으로 매년 조사를 실시한다. 2008년 414,509 응답자를 대상으로 인터뷰했고 인구통계, 건강, 건강 위험에 관해서 설문했다. 수집한 데이터 중에 398,484 응답자 킬로그램으로 표시된 체중정보가 있다.

이 책을 위한 저장소에 BRFSS에 관한 데이터를 담고 있는 고정폭 아스키(ASCII) 파일, `CDBRFS08.ASC.gz`와 더불어 파일을 읽고 데이터를 분석하는 `brfss.py`파일도 함께 있다.

그림 **??** (왼편)이 정규분포 모형에 선형 척도로 성인 체중 분포를 나타낸다. 그림 **??** (오른편)이 로그 정규분포 모형으로 로그 척도로 동일한 분포를 나타낸다. 로그 정규 모형이 더 나은 적합이지만 데이터를 이와 같이 표현하는 것이 차이점을 특별히 인상적으로 만들지는 못한다.

그림 5.7이 성인 체중 w에 대한 정규확률그림과 로그 변환한 체중 $\log_{10} w$에 대한 정규확률그림을 보여준다. 이제 데이터가 정규분포 모형에서 상당히 벗아난 것이 명확하다. 다른 한편으로 로그 정규모형은 데이터에 대한 좋은 매칭을 보여준다.

제 5 절 파레토 분포 (Pareto distribution)

파레토 분포 (Pareto distribution)는 경제학자 빌프레도 파레토(Vilfredo Pareto) 이름에서 나왔는데, 이것을 사용해서 부(`http://wikipedia.org/wiki/Pareto_distribution` 참조)의 분포를 기술했다. 그후, 도시와 마을 크기, 모래 입자와 운석, 산불과 지진을 포함한 자연과학과 사회과학 현상을 기술하는데 사용되었다.

파레토 분석 CDF는 다음과 같다.

$$CDF(x) = 1 - \left(\frac{x}{x_m}\right)^{-\alpha}$$

모수 x_m와 α이 분포 위치와 형상을 결정한다. x_m이 가능한 최소값이다. 그림 5.8에 $x_m = 0.5$와 다양한 α 값으로 표현한 파레토 분포 CDF가 있다.

[3]질병통제 예방센터(Centers for Disease Control and Prevention, CDC). 행동 위험 요인 감시 시스템 조사 자료(Behavioral Risk Factor Surveillance System Survey Data). Atlanta, Georgia: 미국 보건 복지부 (U.S. Department of Health and Human Services), 질병통제 예방센터 (Centers for Disease Control and Prevention), 2008.

그림 5.8: CDFs of Pareto distributions with different parameters.

그림 5.9: CCDFs of city and town populations, on a log-log scale.

경험적 분포가 파레토 분포에 적합성을 나타내는 간단한 시각 테스트가 있다: log-log 척도에서 CCDF는 직선으로 보인다. 왜 그런지 살펴보자.

만약 선형척도에서 파레토 분포에서 나온 표본 CCDF를 플롯으로 그린다면, 다음과 같은 함수가 기대된다.

$$y \approx \left(\frac{x}{x_m}\right)^{-\alpha}$$

양변에 로그를 취하면 다음과 같다.

$$\log y \approx -\alpha(\log x - \log x_m)$$

그래서 만약 $\log x$에 $\log y$를 플롯으로 그리면, 기울기 $-\alpha$과 절편 $\alpha \log x_m$을 가진 직선처럼 보여야만 한다.

예제로, 도시와 마을 크기를 살펴보자. 미국 인구 조사국 (U.S. Census Bureau)에서 미국에 있는 모든 도시와 마을 인구정보를 게시한다.

웹사이트 `http://www.census.gov/popest/data/cities/totals/2012/SUB-EST2012-3.html`에서 데이터를 다운로드했다; 책 저장소에 `PEP_2012_PEPANNRES_with_ann.csv` 파일 이름으로 되어있다. 저장소에 `populations.py` 파이썬 프로그램이 있어 파일을 읽고 인구 분포를 플롯으로 그린다.

그림 5.9에 log-log 척도로 인구 CCDF가 있다. 10^{-2} 아래 가장 큰 1% 도시와 마을이 직선을 따라 아래를 향한다. 몇몇 연구자와 마찬가지로 분포 꼬리가 파레토 모형에 적합하다고 결론을 내릴 수 있다.

다른 한편으로 로그 정규분포도 또한 데이터를 잘 모형화한다. 그림 5.10에 인구 CDF의 로그 정규분포 모형(왼쪽), 정규확률그림(오른쪽)이 있다. 두 플롯 그림 모두 데이터와 모형 사이 좋은 적합을 보여준다.

모형 어느 것도 완벽하지 않다. 파레토 모형은 단지 가장 큰 1% 도시에만 적용되지만 분포의 그 특정 부분에 더 적합이 잘 된다. 로그 정규 모형은 99% 다른 부분에 더 적합이 잘 된다. 어느 모형이 적절한가는 분포 어느 부분에 관련이 있는지에 달려 있다.

그림 5.10: CDF of city and town populations on a log-x scale (left), and normal probability plot of log-transformed populations (right).

제 6 절 난수 생성하기 (Generating random numbers)

해석 CDF(analytic CDF)를 사용해서 주어진 분포함수 $p = \mathrm{CDF}(x)$로 난수를 생성한다. 역 CDF를 계산하는 효율적인 방식이 있다면, 0과 1 사이 균등분포에서 p를 선택하고 나서 $x = ICDF(p)$를 선택함으로써 적절한 분포 난수를 생성할 수 있다.

예를 들어 지수분포 CDF는 다음과 같다.

$$p = 1 - e^{-\lambda x}$$

x에 대해 풀게되면 다음이 된다.

$$x = -\log(1 - p)/\lambda$$

그래서 파이썬 코드로 다음과 같이 작성할 수 있다.

```
def expovariate(lam):
    p = random.random()
    x = -math.log(1-p) / lam
    return x
```

함수 `expovariate`는 `lam`을 인자로 받아 모수 `lam`인 지수분포에서 생성된 난수를 반환한다.

파이썬 코드 구현에 두가지 주의점이 있다: `lambda`가 파이썬 예약어(keyword)여서 모수를 `lam`으로 했다. 또한 $\log 0$가 정의되어 있지 않기 때문에, 약간 더 주의해야 한다. `random.random` 구현코드는 0을 반환할 수 있지만 1은 할 수 없다. 그래서, $1 - p$은 1이 될 수는 있지만 0은 될 수 없어서 `log(1-p)`은 항상 정의된다.

제 7 절 왜 모형인가?

이 장의 시작에서 많은 현실 세계 현상이 해석 분포(analytic distribution)으로 모형화 될 수 있다고 말했다. "그래서," "뭐?" 라고 물을 수도 있다.

모든 모형처럼, 해석분포는 추상화다. 의마하는 바은 관련없다고 고려되는 생략한다는 것이다. 예를 들어, 관찰된 분포에는 표본 특유의 측정오류나 유별난 점이 포함될 수 있다; 해석 모형은 이러한 별스러운 점을 매끈하게 평활한다.

해석 모형은 또한 일종의 데이터 압축이다. 모형이 테이터셋에 잘 적합될 때, 적은 모수 집합으로 대량의 데이터를 요약할 수 있다.

때때로, 자연현상에서 나온 데이터가 해석 분포에 잘 적합되는 것을 보면 참 놀랍니다. 하지만, 이같은 관찰이 물리적 시스템에 대한 통찰(insight)을 제공한다. 때때로 관측된 분포가 왜 특별한 형태를 갖는지 설명할 수 있다. 예를 들어, 파레토 분포는 긍정적 피드백을 생성하는 과정(generative process)의 결과다 (소위 선호적 연결 과정 (preferential attachment processes): `http://wikipedia.org/wiki/Preferential_attachment` 사이트를 참조.)

또한, 14장에서 살펴보듯이 해석 분포가 수학적 분석으로 이끈다. .

하지만, 모든 모형이 완전하지 않다는 것을 기억하는 것이 중요하다. 현실 세계에서 나온 데이터는 결코 완벽하게 해석 분포에 적합되지 않는다. 종종 데이터가 모형에서 생성된 것처럼 사람들이 말을 한다; 예를 들어, 사람 신장분포가 정규분포에서 나왔다거나 소득 분포가 로그 정규분포에서 나왔다고 말한다. 문자 그대로 받아들이면, 이러한 주장은 진실이 아니다; 항상 현실 세계와 수학적 모형 사이에는 차이가 있다.

현실 세계의 중요한 측면을 포착하고 불필요한 세부사항을 배제하면 모형은 유용하다. 하지만, "중요하거나(relevant)" "불필요한(unneeded)" 것은 모형을 사용해서 무엇을 계획중인가에 달려있다.

제 8 절 연습문제

For the following exercises, you can start with `chap05ex.ipynb`. My solution is in `chap05soln.ipynb`.

Exercise 5.1 In the BRFSS (see Section 4), the distribution of heights is roughly normal with parameters $\mu = 178$ cm and $\sigma = 7.7$ cm for men, and $\mu = 163$ cm and $\sigma = 7.3$ cm for women.

In order to join Blue Man Group, you have to be male between 5'10" and 6'1" (see `http://bluemancasting.com`). What percentage of the U.S. male population is in this range? Hint: use `scipy.stats.norm.cdf`.

Exercise 5.2 To get a feel for the Pareto distribution, let's see how different the world would be if the distribution of human height were Pareto. With the parameters $x_m = 1$ m and $\alpha = 1.7$, we get a distribution with a reasonable minimum, 1 m, and median, 1.5 m.

Plot this distribution. What is the mean human height in Pareto world? What fraction of the population is shorter than the mean? If there are 7 billion people in Pareto world, how many do we expect to be taller than 1 km? How tall do we expect the tallest person to be?

Exercise 5.3 The Weibull distribution is a generalization of the exponential distribution that comes up in failure analysis (see `http://wikipedia.org/wiki/Weibull_distribution`). Its CDF is

$$CDF(x) = 1 - e^{-(x/\lambda)^k}$$

Can you find a transformation that makes a Weibull distribution look like a straight line? What do the slope and intercept of the line indicate?

Use `random.weibullvariate` to generate a sample from a Weibull distribution and use it to test your transformation.

Exercise 5.4 For small values of n, we don't expect an empirical distribution to fit an analytic distribution exactly. One way to evaluate the quality of fit is to generate a sample from an analytic distribution and see how well it matches the data.

For example, in Section 1 we plotted the distribution of time between births and saw that it is approximately exponential. But the distribution is based on only 44 data points. To see whether the data might have come from an exponential distribution, generate 44 values from an exponential distribution with the same mean as the data, about 33 minutes between births.

Plot the distribution of the random values and compare it to the actual distribution. You can use `random.expovariate` to generate the values.

Exercise 5.5 In the repository for this book, you'll find a set of data files called `mystery0.dat`, `mystery1.dat`, and so on. Each contains a sequence of random numbers generated from an analytic distribution.

You will also find `test_models.py`, a script that reads data from a file and plots the CDF under a variety of transforms. You can run it like this:

```
$ python test_models.py mystery0.dat
```

Based on these plots, you should be able to infer what kind of distribution generated each file. If you are stumped, you can look in `mystery.py`, which contains the code that generated the files.

Exercise 5.6 The distributions of wealth and income are sometimes modeled using lognormal and Pareto distributions. To see which is better, let's look at some data.

The Current Population Survey (CPS) is a joint effort of the Bureau of Labor Statistics and the Census Bureau to study income and related variables. Data collected in 2013 is available from `http://www.census.gov/hhes/www/cpstables/032013/hhinc/toc.htm`. I downloaded `hinc06.xls`, which is an Excel spreadsheet with information about household income, and converted it to `hinc06.csv`, a CSV file you will find in the repository for this book. You will also find `hinc.py`, which reads this file.

Extract the distribution of incomes from this dataset. Are any of the analytic distributions in this chapter a good model of the data? A solution to this exercise is in `hinc_soln.py`.

제 9 절 용어 사전

- 경험적 분포 (empirical distribution): 표본 값의 분포
- 해석 분포 (analytic distribution): CDF가 해석함수인 분포.
- 모형 (model): 유용한 단순화. 해석 분포는 종종 많이 복잡한 경험적 분포에 대한 좋은 모형이다.
- 도착간격시간 (interarrival time): 두 사건 사이 경과 시간.
- 보완 CDF (complementary CDF): 값 x에서 x를 넘는 값 비율로 매칭하는 함수, $1 - \mathrm{CDF}(x)$.
- 표준정규분포 (standard normal distribution): 평균 0과 표준편차 1을 갖는 정규분포.
- 정규확률그림 (normal probability plot): 표본값과 표준정규분포에서 나온 난수를 대비하여 플롯하여 그린 그림.

제 6 장

확률밀도함수

이번 장에서 사용되는 코드는 `density.py`에 있다. 코드를 다운로드하고 작업하는 것에 대한 정보는 2을 참조한다.

제 1 절 PDF

CDF 미분을 **확률밀도함수 (probability density function)**, PDF라고 한다. 예를 들어, 지수분포 PDF는 다음과 같다.

$$\text{PDF}_{expo}(x) = \lambda e^{-\lambda x}$$

정규분포 PDF는 다음과 같다.

$$\text{PDF}_{normal}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$

x 특정한 값에 대한 PDF를 계산하는 것이 대체로 유용하지는 않다. 결과가 확률이 아니기 때문이다; 확률 *밀도 (density)*다.

물리학에서 밀도는 단위 체적당 질량이다; 질량을 계산하려면, 체적을 곱하거나 혹은 만약 밀도가 상수가 아니라면 체적에 대해 적분해야 한다.

마찬가지로, **확률 밀도 (probability density)**는 단위 x당 확률을 측정한다. 확률 질량을 계산하려면, x에 대해서 적분해야 한다.

`thinkstats2`는 Pdf라는 클래스로 확률밀도함수를 나타낸다. 모든 Pdf 객체는 다음 메쏘드를 제공한다:

- Density, x를 인자로 받아 x에서 분포 밀도를 반환한다.
- Render는 이산 집합값 대해 밀도를 평가하고 시퀀스 짝을 반환한다: 정렬된 xs과 상응하는 확률 밀도ds.
- MakePmf, 이산 집합값에 대해 Density를 평가하고 Pdf에 그사하는 정규화된 Pmf를 반환한다.
- GetLinspace, Render와 MakePmf에서 사용되는 기본설정 집합 점(point)을 반환한다.

Pdf는 추상화된 부모 클래스로 의미하는 것은 인스턴스화하면 안된다; 즉, Pdf 객체를 생성할 수 없다. 대신에 Pdf를 상속받아 Density과 GetLinspace 정의하는 자식 클래스를 정의해야 한다. Pdf는 Render과 MakePmf을 제공한다.

예를 들어, thinkstats2는 정규밀도함수를 평가하는 NormalPdf라는 이름의 클래스를 제공한다.

```
class NormalPdf(Pdf):

    def __init__(self, mu=0, sigma=1, label=''):
        self.mu = mu
        self.sigma = sigma
        self.label = label

    def Density(self, xs):
        return scipy.stats.norm.pdf(xs, self.mu, self.sigma)

    def GetLinspace(self):
        low, high = self.mu-3*self.sigma, self.mu+3*self.sigma
        return np.linspace(low, high, 101)
```

NormalPdf 객체는 모수 mu과 sigma을 담고 있다. Density는 scipy.stats.norm을 사용하는데 정규분포를 표현한고, 다른 메쏘드와 더불어 cdf와 pdf를 제공한다.(2절 참조).

다음 예제는 BRFSS에 있는 성인여성신장(cm 단위) 평균과 분산으로 NormalPdf를 생성한다(4절 참조). 그리고 나서, 평균에서 1 표준편차 지점에 분포 밀도를 계산한다.

```
>>> mean, var = 163, 52.8
>>> std = math.sqrt(var)
>>> pdf = thinkstats2.NormalPdf(mean, std)
>>> pdf.Density(mean + std)
0.0333001
```

그림 6.1: A normal PDF that models adult female height in the U.S., and the kernel density estimate of a sample with $n = 500$.

결과는 cm 당 확률질량 단위로 약 0.03이다. 한번더, 확률밀도는 그 자체로 의미는 없다. 하지만, Pdf를 플롯으로 그린다면, 분포 형상을 볼 수 있다.

```
>>> thinkplot.Pdf(pdf, label='normal')
>>> thinkplot.Show()
```

`thinkplot.Pdf`은 평활 함수(smooth function)로 Pdf 플롯을 그린다. 계단함수로 Pmf을 그리는 `thinkplot.Pmf`와 대조된다. 그림 6.1에 결과가 있다. 다음 절에서 살펴볼 표본에서 추정한 PDF로 함께 플롯되어 그려져 있다.

Pdf를 근사하는데 `MakePmf`를 사용할 수도 있다.

```
>>> pmf = pdf.MakePmf()
```

기본설정으로, `mu - 3*sigma`에서 `mu + 3*sigma` 사이에 동일 간격을 지닌 101 점이 Pmf에 있다. 선택사양으로 `MakePmf`와 `Render`는 키워드 인자로 `low`, `high`, `n`을 갖는다.

제 2 절 핵밀도추정 (Kernel density estimation)

핵밀도추정 (Kernel density estimation,KDE)은 표본을 받아 데이터에 적합하는 적절한 평활 PDF를 찾는 알고리즘이다. `http://en.wikipedia.org/wiki/Kernel_density_estimation` 웹사이트에서 좀더 자세한 정보를 얻을 수 있다.

`scipy`에 KDE 구현된 것이 있고, `thinkstats2`는 이를 사용해서 `EstimatedPdf`라는 클래스를 제공한다.

```
class EstimatedPdf(Pdf):

    def __init__(self, sample):
        self.kde = scipy.stats.gaussian_kde(sample)

    def Density(self, xs):
        return self.kde.evaluate(xs)
```

`__init__`이 표본을 인자로 받아 핵밀도추정값을 계산한다. 결과는 `gaussian_kde` 객체고 `evaluate` 메쏘드를 제공한다.

`Density`가 값 혹은 시퀀스를 인자로 받아 `gaussian_kde.evaluate`을 호출하고 결과 밀도를 반환한다. 단어 "가우스 (Gaussian)"가 나오는데 이유는 KDE를 평활하는데 가우스 분포에 기반한 필터를 사용하기 때문이다.

그림 6.2: A framework that relates representations of distribution functions.

다음에 정규분포에서 표본을 생성하고 표본에 적합하기 위해서 EstimatedPdf를 만드는 예제가 있다.

```
>>> sample = [random.gauss(mean, std) for i in range(500)]
>>> sample_pdf = thinkstats2.EstimatedPdf(sample)
>>> thinkplot.Pdf(sample_pdf, label='sample KDE')
```

`sample`은 무작위 신장 500개 리스트다. `sample_pdf`는 Pdf 객체로 추정된 KDE 표본정보를 담고 있다. 동일 간격 값의 시퀀스에서 밀도를 평가함으로써 `pmf`는 Pmf 객체로 Pdf를 근사한다.

그림 6.1에 정규밀도함수와 무작위 신장 500개 표본에 기반한 KDE가 있다. 추정값이 원분포에 좋은 매칭이다.

KDE로 밀도함수를 추정하는 것은 몇가지 목적으로 유용한다.

- *시각화 (Visualization):* 프로젝트 탐색단계에서, CDF가 대체로 분포를 가장 잘 시각화한다. CDF를 살펴본 후에, 추정 PDF가 분포에 대한 적절한 모형인지 결정할 수 있다. 만약 그렇다면, CDF에 익숙하지 않은 관계에게 분포를 제시하는데 더 좋은 선택지가 될 수 있다.

- *보간 (Interpolation):* 추정 PDF는 표본에서 모집단 모형으로 가는 한 방법이다. 만약 모집단 분포가 매끄럽다고 믿을 이유가 있다면, KDE를 사용해서 표본에 없는 값에 대해 밀도를 보간한다.

- *모의실험 (Simulation):* 모의실험은 종종 표본 분포에 기반한다. 만약 표본크기가 작다면, KDE를 사용해서 표본분포를 평활하는 것이 적절하다. 관측점을 중복하기 보다 KDE가 모의실험을 통해서 좀더 가능한 결과값을 탐색하도록 한다.

제 3 절 분포 프레임워크 (distribution framework)

현재까지 PMF, CDF, PDF를 살펴봤다; 잠시 복습 시간을 가져본다. 그림 6.2에 함수가 어떻게 서로 연관되는지 나타나 있다.

PMF로 시작했는데, PMF는 이산 집합 값에 대한 확률을 나타낸다. PMF에서 CDF를 얻기 위해서는, 확률 질량을 더해서 누적 확률을 얻는다. CDF에서 PMF로 돌아가기 위해서는, 누적 확률 차이를 계산한다. 다음 몇 절에 걸쳐 이와 같은 연산을 어떻게 구현했는지 살펴볼 것이다.

PDF는 연속형 CDF 미분이다; 혹은, 동등하게 CDF는 PDF의 적분이다. PDF는 값을 확률 밀도로 매핑한다는 것을 기억하라; 확률값을 얻기 위해서, 적분해야 한다.

이산형에서 연속 분포를 얻기 위해서, 다양한 평활(smoothing) 작업을 수행할 수 있다. 평활의 한 형태는 데이터가 (지수 혹은 정규 분포처럼) 해석 연속 분포(analytic continuous distribution)에서 왔다고 가정하는 것이다. 또 다른 선택 옵션은 핵밀도추정(kernel density estimation)이다.

평활의 반대가 **이산화 (discretizing)**, 혹은 양자화(quantizing)다. 만약 이산 점에서 PDF를 평가한다면, PDF에 근사하는 PMF를 생성할 수 있다. 수치적분(numerical integration)을 사용해서 좀더 잘 근사할 수도 있다.

연속CDF와 이산CDF를 구별하기 위해서, 이산CDF는 "누적 질량 함수 (cumulative mass function)"가 되는 것이 좋을지도 모른다. 하지만, 저자가 알고 있는 바로는, 누구도 그 용어를 사용하지 않는다.

제 4 절 Hist 구현

`thinkstats2`에서 제공되는 기본 기능 사용법을 알아야 한다: Hist, Pmf, Cdf, and Pdf. 다음 절에는 구현된 방식에 대한 상세한 정보가 나와있다. 학습 교재가 좀더 효율적으로 이들 클래스를 사용하는지 도움을 줄 수 있지만, 엄격히 말해서 반듯이 필요하지는 않다.

Hist와 Pmf는 `_DictWrapper`라는 부모 클래스를 상속받는다. 클래스 앞 밑줄은 클래스가 "내부(internal)"라는 것을 나타낸다; 즉, 다른 모듈에 코드로 사용되면 않된다. 명칭이 무엇인지 나타낸다: 딕셔너리 랩퍼(dictionary wrapper). 주요 속성은 `d`로, 값을 빈도로 매핑하는 딕셔너리다.

값은 임의 해쉬형(hashable type)이 될 수 있다. 빈도는 정수형이어야 하지만, 임의 숫자형도 될 수 있다.

`_DictWrapper`는 Hist와 Pmf에 대한 적절한 메쏘드를 담고 있는데, `__init__`, `Values`, `Items`, `Render`가 포함된다. `Set`, `Incr`, `Mult`, `Remove` 변경 메쏘드도 제공한다. 모든 메쏘드는 딕셔너리 연산으로 구현되었다. 예를 들어,

```
# class _DictWrapper

    def Incr(self, x, term=1):
        self.d[x] = self.d.get(x, 0) + term

    def Mult(self, x, factor):
        self.d[x] = self.d.get(x, 0) * factor
```

```
    def Remove(self, x):
        del self.d[x]
```

Hist는 또한 Freq을 제공하는데 주어진 값에 대한 빈도를 찾는다.

Hist 연산자와 메쏘드는 딕셔너리에 기반하고 있어서, 이들 메쏘드는 상수 시간 연산이다; 즉, Hist가 점점 커짐에 따라 실행시간이 증가하지 않는다.

제 5 절 Pmf 구현

Pmf가 정수 빈도 대신에 값을 부동소수점 확률에 매핑하는 것을 제외하고, Pmf와 Hist는 거의 동일하다. 확률을 다 더한 합계가 1 이라면, Pmf는 정규화되었다.

Pmf는 Normalize 함수를 제공하는데, 확률을 합을 계산하고 갯수로 나눈다.

```
# class Pmf

    def Normalize(self, fraction=1.0):
        total = self.Total()
        if total == 0.0:
            raise ValueError('Total probability is zero.')

        factor = float(fraction) / total
        for x in self.d:
            self.d[x] *= factor

        return total
```

fraction이 정규화한 후에 확률 합계를 알아낸다; 기본 설정값은 1 이다. 만약 전체 확률이 0 이라면, Pmf는 정규화될 수 없어서, Normalize는 ValueError 오류를 일으킨다.

Hist와 Pmf는 동일한 생성자를 갖고 있다. 인자로 dict, Hist, Pmf 혹은 Cdf, 판다스 시리즈, (값, 빈도) 리스트 쌍, 혹은 값 시퀀스를 받을 수 있다.

만약 Pmf를 인스턴스화 한다면, 결과는 정규화된다. 만약 Hist를 인스턴스화 한다면, 결과는 정규화되지 않는다. 정규화되지 않은 Pmf를 생성하기 위해서, 빈 Pmf를 생성해서 변경할 수 있다. Pmf 변경자는 Pmf를 다시 정규화하지 않는다.

제 6 절 Cdf 구현

CDF가 값을 누적 확률로 매핑해서 Cdf를 `_DictWrapper`로 구현할 수 있다. 하지만 CDF에 있는 값은 정렬되어 있고 `_DictWrapper`에 있는 값은 정렬되어 있지 않다. 또한, 역 CDF를 계산하는 것은 유용하다; 즉, 누적 확률에서 값으로 매핑. 그래서, 저자가 선택한 구현 방법은 두 정렬된 리스트다. 이런 방식으로 이진검색을 사용해서 로그 시간에 앞으로 혹은 역으로 조회할 수 있다.

Cdf 생성자는 인자로 값 시퀀스 혹은 판다스 시리즈, 값에서 확률로 매핑하는 딕셔너리, (값, 확률) 시퀀스 쌍, Hist, Pmf, 혹은 CDF를 받는다. 혹은 만약 인자가 두개 주어진다면, 두 인자를 각각 정렬된 값 시퀀스와 상응하는 누적확률 시퀀스로 처리한다.

시퀀스, 판다스 시퀀스, 혹은 딕셔너리가 주어지면, 생성자가 Hist를 만든다. 그리고 나서 Hist를 사용해서 속성을 초기화한다.

```
        self.xs, freqs = zip(*sorted(dw.Items()))
        self.ps = np.cumsum(freqs, dtype=np.float)
        self.ps /= self.ps[-1]
```

`xs`는 정렬된 리스트 값이다; `freqs`는 상응하는 빈도 리스트다. `np.cumsum`이 누적 빈도 합계를 계산한다. 전체 빈도로 나누면 누적확률이 된다. `n`개 값에 대해서, Cdf를 생성하는 시간은 $n \log n$에 비례한다.

다음에 `Prob` 구현한 코드가 있다. 값을 받아 누적 확률을 반환한다.

```
# class Cdf
    def Prob(self, x):
        if x < self.xs[0]:
            return 0.0
        index = bisect.bisect(self.xs, x)
        p = self.ps[index - 1]
        return p
```

`bisect` 모듈은 이진 검색 구현을 제공한다. 그리고, `Value`를 구현했는데 누적 확률을 받아 상응하는 값을 반환한다.

```
# class Cdf
    def Value(self, p):
        if p < 0 or p > 1:
            raise ValueError('p must be in range [0, 1]')

        index = bisect.bisect_left(self.ps, p)
        return self.xs[index]
```

Cdf가 주어지면, 연속 누적 확률 (consecutive cumulative probabilities) 사이 차이를 계산해서 Pmf를 계산할 수 있다. Cdf 생성자를 호출하고 Pmf를 전달한다면, `Cdf.Items`를 호출해서 차이를 계산한다.

```
# class Cdf
    def Items(self):
        a = self.ps
        b = np.roll(a, 1)
        b[0] = 0
        return zip(self.xs, a-b)
```

`np.roll`는 `a` 요소를 오른쪽으로 이동하고, 마지막을 처음으로 "돌린다(roll)" `b` 첫 요소를 0으로 바꾸고 나서 `a-b` 차이를 계산한다. 결과는 넘파이 확률 배열이다.

Cdf는 `Shift`와 `Scale`를 제공한다. Cdf에 값을 변경하지만, 확률은 불변형(immutable)으로 처리되어야 한다.

제 7 절 적률 (Moments)

어느 때고 표본을 얻고 하나의 숫자로 줄일 수 있다. 그 숫자가 통계량(statistic)이다. 지금까지 살펴본 통계량은 평균, 분산, 중위수, 그리고 사분위수다.

원적률 (raw moment)은 일종의 통계량이다. 만약 x_i 표본 값이 있다면, k번째 원적률은 다음과 같다.

$$m'_k = \frac{1}{n}\sum_i x_i^k$$

혹은, 파이썬 표기법으로 표현하면, 다음과 같다.

```
def RawMoment(xs, k):
    return sum(x**k for x in xs) / len(xs)
```

$k = 1$일 때, 결과는 표본 평균 $\bar{x}$가 된다. 다른 원적률은 그 자체로 의미가 있지 않다. 하지만, 다른 계산에 사용된다.

중심적률(central moments)이 더 유용하다. k번째 중심적률은 다음과 같다.

$$m_k = \frac{1}{n}\sum_i (x_i - \bar{x})^k$$

혹은 파이썬으로 표현하면 다음과 같다.

```
def CentralMoment(xs, k):
    mean = RawMoment(xs, 1)
    return sum((x - mean)**k for x in xs) / len(xs)
```

$k = 2$일 때, 결과는 두번째 중심 적률로 분산으로 인지하고 있을 것이다. 분산의 정의가 왜 이러한 통계량이 적률로 불리는지 힌트를 준다. 각 위치 x_i에 자를 따라 추를 달고 평균 주위에서 자를 돌리면, 회전추의 관성 적률은 값의 분산이다. 만약 관성 적률에 익숙하지 않다면, `http://en.wikipedia.org/wiki/Moment_of_inertia` 웹사이트를 참조한다.

적률에 기반한 통계를 보고할 때, 단위(unit)에 관한 생각이 중요하다. 예를 들어, 값 x_i가 cm 이라면, 첫 원적률은 또한 cm이다. 하지만, 두번째 적률은 cm^2이고, 세번째 적률은 cm^3... 등등이 된다.

이러한 단위 때문에, 적률은 그 자체로 해석하기 어렵다. 두번째 적률에 대해서 분산에 제곱근을 취한 표준편차를 쓰는 것이 이러한 이유다. 그러면 x_i와 단위가 같아진다.

제 8 절 왜도 (Skewness)

왜도 (Skewness)는 분포 형태를 기술하는 속성이다. 만약 분포가 중심 경향성 주변에서 대칭이라면, 기울어지지 않았다. 만약 값들이 오른쪽으로 좀더 뻗어져 있다면, "오른쪽으로 기울어져 (right skewed)" 있고, 만약 값들이 왼쪽으로 치우쳐 있다면, "왼쪽으로 기울어져 (left skewed"있다.

"기울어짐 (skewed)"을 사용하는 것다는 것이 "편의(biased)"를 함축하지는 않는다. 단지 왜도는 분포 형태만을 기술한다; 표본 추출과정에 편의가 있는지에 관해 어떤 것도 나타내지 않는다.

흔히 몇몇 통계량이 분포 왜도를 계량화하는데 사용된다. 주어진 값 시퀀스가 x_i가 주어졌을 때, **표본 왜도 (sample skewness)** g_1은 다음과 같이 계산된다.

```
def StandardizedMoment(xs, k):
    var = CentralMoment(xs, 2)
    std = math.sqrt(var)
    return CentralMoment(xs, k) / std**k


def Skewness(xs):
    return StandardizedMoment(xs, 3)
```

g_1은 제3 표준 적률로 정규화되었다는 것으로 단위가 없다.

음수 왜도는 분포가 왼쪽으로 기울어짐; 양수 왜도는 분포가 오른쪽으로 기울어짐을 나타낸다. g_1의 규모는 왜도 강도를 나타내지만, 그 자체로 해석하기는 쉽지 않다.

실무에서, 표본 왜도를 계산하는 것이 대게 좋은 생각이 되지는 못하다. 만약 어떤 특이점(outlier)이 있다면, g_1에 대해서 균형이 맞지 않는 효과를 미친다.

그림 6.3: Estimated PDF of birthweight data from the NSFG.

분포 비대칭을 평가하는 또 다른 방법은 평균과 중위수 사이 관계를 살펴보는 것이다. 극단값(extreme value)이 중위수보다 평균에 미치는 효과가 더 크다. 그래서 왼쪽으로 기울어진 분포에서 평균은 중위수보다도 더 작다. 오른쪽으로 기울어진 분포에서 평균은 더 크다.

피어슨 중위수 왜도 계수 (Pearson's median skewness coefficient)는 표본 평균과 중위수 차이에 기반한 왜도 측도다.

$$g_p = 3(\bar{x} - m)/S$$

$\bar{x}$는 표본 평균, m은 중위수, S는 표준편차다. 혹은 파이썬에서 코드로 작성한 함수는 다음과 같다.

```
def Median(xs):
    cdf = thinkstats2.Cdf(xs)
    return cdf.Value(0.5)

def PearsonMedianSkewness(xs):
    median = Median(xs)
    mean = RawMoment(xs, 1)
    var = CentralMoment(xs, 2)
    std = math.sqrt(var)
    gp = 3 * (mean - median) / std
    return gp
```

이 통계량은 **강건(robust)**해서 의미하는 바는 특이점 때문에 쉽게 휘둘리지 않는다는 것이다.

예제로 NSFG 임신 데이터에 있는 출생 체중 왜도를 살펴보자. 다음에 PDF를 추정하고 플롯으로 그리는 코드가 있다.

```
    live, firsts, others = first.MakeFrames()
    data = live.totalwgt_lb.dropna()
    pdf = thinkstats2.EstimatedPdf(data)
    thinkplot.Pdf(pdf, label='birth weight')
```

그림 6.3에 결과가 있다. 왼쪽 꼬리가 오른쪽 꼬리보다 더 길어 보인다. 그래서 분포가 왼쪽으로 기울어진 것으로 추측해 볼 수 있다. 평균은 7.27 lbs 으로 중위수 7.38 lbs 보다 다소 작아서 왼쪽으로 기울어진 것과 일관성이 있다. 그리고 두 왜도 계수가 모두 음수다: 표본 왜도는 -0.59; 피어슨 중위수 왜도는 -0.23.

이제 출생 체중 분포에 대해 BRFSS에 있는 성인 체중 분포와 비교해 보자. 다음에 파이썬 코드가 있다.

그림 6.4: Estimated PDF of adult weight data from the BRFSS.

```
df = brfss.ReadBrfss(nrows=None)
data = df.wtkg2.dropna()
pdf = thinkstats2.EstimatedPdf(data)
thinkplot.Pdf(pdf, label='adult weight')
```

그림 6.4에 결과가 있다. 분포가 오른쪽으로 기울어진 것으로 보인다. 말할 것도 없이, 평균이 79.0으로 중위수 77.3 보다 더 크다. 표본 왜도가 1.1이고 피어슨 중위수 왜도는 0.26이다.

왜도 계수 부호는 분포가 왼쪽 혹은 오른쪽으로 기울어진 것을 나타내지만, 그것을 제외하고 해석하기는 어렵다. 표본 왜도는 덜 강건하다; 즉, 특이점에 더 좌우된다. 결과로서 덜 믿음이 가는데, 기울어진 분포에 적용될 때, 정확하게 가장 관련될 때 그렇다.

피어슨 중위수 왜도는 계산된 평균과 분산에 기반한다. 그래서 특이점에 휘둘리기 쉽지만 제3 적률에 의존하지 않기 때문에 좀더 강건하다.

제 9 절 연습문제

A solution to this exercise is in `chap06soln.py`.

Exercise 6.1 The distribution of income is famously skewed to the right. In this exercise, we'll measure how strong that skew is.

The Current Population Survey (CPS) is a joint effort of the Bureau of Labor Statistics and the Census Bureau to study income and related variables. Data collected in 2013 is available from `http://www.census.gov/hhes/www/cpstables/032013/hhinc/toc.htm`. I downloaded `hinc06.xls`, which is an Excel spreadsheet with information about household income, and converted it to `hinc06.csv`, a CSV file you will find in the repository for this book. You will also find `hinc2.py`, which reads this file and transforms the data.

The dataset is in the form of a series of income ranges and the number of respondents who fell in each range. The lowest range includes respondents who reported annual household income "Under $5000." The highest range includes respondents who made "$250,000 or more."

To estimate mean and other statistics from these data, we have to make some assumptions about the lower and upper bounds, and how the values are distributed in each range. `hinc2.py` provides `InterpolateSample`,

which shows one way to model this data. It takes a DataFrame with a column, `income`, that contains the upper bound of each range, and `freq`, which contains the number of respondents in each frame.

It also takes `log_upper`, which is an assumed upper bound on the highest range, expressed in `log10` dollars. The default value, `log_upper=6.0` represents the assumption that the largest income among the respondents is 10^6, or one million dollars.

`InterpolateSample` generates a pseudo-sample; that is, a sample of household incomes that yields the same number of respondents in each range as the actual data. It assumes that incomes in each range are equally spaced on a log10 scale.

Compute the median, mean, skewness and Pearson's skewness of the resulting sample. What fraction of households reports a taxable income below the mean? How do the results depend on the assumed upper bound?

제 10 절 용어 사전

- 확률밀도함수 (Probability density function, PDF): 연속 CDF 미분으로 값을 확률 밀도에 매핑하는 함수.
- 확률밀도 (Probability density): 확률을 만들기 위해서 범위 값에 대해 적분할 수 있는 양. 예를 들어, 만약 값 단위가 cm이라면, 확률밀도는 cm 당 확률 단위가 된다.
- 핵밀도추정 (Kernel density estimation, KDE): 표본에 기반해서 PDF를 추정하는 알고리즘.
- 이산화 (discretize): 연속함수 혹은 이산 함수를 가진 분포를 근사함. 평활(smoothing)의 반대.
- 원적률 (raw moment): 거듭 제곱되는 데이터 합계에 기반한 통계량
- 중심적률 (central moment): 평균에서 편차 거듭제곱에 기반한 통계량.
- 표준적률 (standardized moment): 단위가 없는 적률 비율.
- 왜도 (skewness): 분포가 얼마나 비대칭인지 나타내는 측도.
- 표본 왜도 (sample skewness): 분포 왜도를 정량화하는데 사용되는 적률 기반 통계량.

- 피어슨 중위수 왜도 계수 (Pearson's median skewness coefficient): 중위수, 평균, 표준편차에 기반한 분포 왜도를 정량화하는데 사용되는 통계량.

- 강건성 (robust): 특이점에 상대적으로 면역되어 휘둘리지 않는다면 통계량은 강건하다.

제 7 장

변수간 관계

지금까지 한번에 한 변수만 살펴봤다. 이번장에서 변수간 관계를 살펴본다. 변수 하나를 알고 있는 것이 다른 변수에 대한 정보를 준다면, 두 변수는 관계가 있다. 예를 들어, 신장과 체중은 관계가 있다; 키가 더 큰 사람이 체중이 더 나가는 경향이 있다. 물론, 완벽한 관계는 아니다: 키 작고 뚱뚱한 사람과 키 크고 마른 사람이 있다. 하지만, 다른 사람의 체중을 추측하려고 한다면, 신장 정보를 모르는 것보다 키 정보를 알고 있는 것이 좀더 정확하게 추정하는데 도움이 된다.

이번 장에서 사용되는 코드는 `scatter.py`에 있다. 코드를 다운로드하고 작업하는 것에 대한 정보는 2을 참조한다.

제 1 절 산점도 (Scatter plots)

관계(relationship)을 확인하는 가장 간단한 방법이 **산점도 (scatter plot)**다. 하지만, 좋은 산점도를 만드는 것이 항상 쉬운 것은 아니다. 예제로, BRFSS 응답자에 대한 신장과 체중 플롯을 그릴 것이다. (4 절 참조)

데이터 파일을 읽어서 신장과 체중 정보를 추출하는 코드가 다음에 있다.

```
    df = brfss.ReadBrfss(nrows=None)
    sample = thinkstats2.SampleRows(df, 5000)
    heights, weights = sample.htm3, sample.wtkg2
```

`SampleRows` 함수는 데이터에서 일부를 임으로 골라낸다.

```
def SampleRows(df, nrows, replace=False):
    indices = np.random.choice(df.index, nrows, replace=replace)
    sample = df.loc[indices]
    return sample
```

그림 7.1: Scatter plots of weight versus height for the respondents in the BRFSS, unjittered (left), jittered (right).

`df`는 데이터프레임이고, `nrows`는 선택할 행의 갯수가 되고, `replace`는 복원 추출을 해야 하는지 나타내는 부울(boolean) 표식이다; 다른 말로, 동일한 행이 한 번 이상 선택되어 추출되는지 나타낸다.

`thinkplot`에는 `Scatter` 메쏘드가 있는데, 산점도를 그린다.

```
    thinkplot.Scatter(heights, weights)
    thinkplot.Show(xlabel='Height (cm)',
                   ylabel='Weight (kg)',
                   axis=[140, 210, 20, 200])
```

그림 7.1 (왼쪽)에 관계 형태를 보여주는 결과가 있다. 예상했던 것처럼, 더 키가 큰 사람이 더 체중이 나가는 경향이 있다.

하지만, 상기 그림이 데이터를 가장 잘 표현하는 것은 아니다. 이유는 데이터가 열에 떼지어 몰려있다. 문제는 신장이 가장 가까운 인치(inch) 단위로 반올림되고, 센티미터로 변환하고 나서, 다시 반올림했다. 변환 과정에서 정보가 유실되었다.

유실된 정보를 다시 되돌릴 수는 없지만, 데이터를 **지터링(jittering)**[1]해서 산점도에 효과를 최소화할 수 있다. 반올림 효과를 되돌리도록 확률 잡음(random noise)를 추가한다는 의미다. 측정값이 가장 근사한 인치(inch) 정보로 반올림되어 있어서, 0.5 인치 즉, 1.3 센티미터까지 차이가 생길 수도 있다. 마찬가지로, 체중은 0.5 kg까지 차이가 생길 수 있다.

```
    heights = thinkstats2.Jitter(heights, 1.3)
    weights = thinkstats2.Jitter(weights, 0.5)
```

`Jitter` 함수를 구현한 것이 다음에 있다.

```
def Jitter(values, jitter=0.5):
    n = len(values)
    return np.random.uniform(-jitter, +jitter, n) + values
```

값은 임의 시퀀스가 될 수 있다; 결과는 넘파이(NumPy) 배열이다.

그림 7.1 (오른편)에 결과가 있다. 지터링(jittering)을 통해서 반올림으로 인한 시각적 효과를 줄이고, 관계 형태를 좀더 명확히 한다. 하지만, 일반적으로 시각화 목적으로만 데이터를 지터링해야 하고, 분석을 위해서 지터링된 데이터 사용은 피해야 한다.

[1] jittering, 지터로 번역을 했는데 통계 사전에는 등록이 되어있지 않고 일반 사전에는 "조금씩 움직이다"라고 나와있다.

그림 7.2: Scatter plot with jittering and transparency (left), hexbin plot (right).

지터링 조차도 데이터를 표현하는 가장 최선의 방법이 되지는 못한다. 중복되는 점이 많아서 그림에서 조밀한 부분에 있는 데이터를 숨기고, 균형이 맞지 않게 특이점을 강조한다. 이와 같은 효과를 **포화 (saturation)**라고 부른다.

이런 문제를 `alpha` 모수로 해결할 수 있는데, 수행하는 역할은 점들을 부분적으로 투명하게 한다.

```
thinkplot.Scatter(heights, weights, alpha=0.2)
```

그림 7.2 (왼편)에 결과가 있다. 겹쳐지는 데이터 점들이 더 어두워 보여서 색이 짙음이 밀도와 비례하여 균형을 맞춘다. 이 버젼으로 그린 플롯을 살펴보면, 앞에서 명확하지 않은 두가지 자세한 사항을 볼 수 있다; 90 kg 즉, 200 파운드 근처에 수평선과 몇군데 신장에서 수직 군집이 보인다. 데이터가 파운드 단위로 자기 응답에 기반하기 때문에, 가장 그럴듯한 설명은 응답자가 반올림한 값으로 보고를 한 것이다.

투명성을 사용해서 중간정도 크기 데이터셋를 처리했다. 하지만, 그림은 단지 BRFSS 자료 414 509 중에서 첫 5000 레코드만 보여준다.

더 커다란 데이터셋을 처리하기 위한, 또 다른 선택지가 육각함 플롯 (hexbin plot)이 된다. 그래프를 육각형 통(hexagonal bin)으로 나누고 각 통을 얼마나 많은 데이터가 들어있는지에 따라 색을 칠한다. `thinkplot`에 `HexBin`메쏘드가 있다.

```
thinkplot.HexBin(heights, weights)
```

그림 7.2 (오른편)에 결과가 있다. 육각함(hexbin)의 장점은 관계 형태도 보여준다는 것이고, 큰 데이터셋에 대해서도 시간과 파일 크기에 둘 관점에서 효율적이다. 단점은 특이점이 보이지 않는다는 것이다.

이 사례를 통해서 강조하고자 하는 것은 오해를 불러 일으키는 산출물 없이 관계를 명확하게 보여주는 산점도를 플롯으로 그리는 것이 쉬운 것은 아니라는 점이다.

제 2 절 관계를 특징짓기

산점도는 변수 사이에 관계에 대한 일반적인 인상을 제공한다. 하지만, 관계 본질에 대한 더 많은 통찰을 주는 다른 시각화방법이 있다. 한 선택지는 변수를 구간화(binning)하고 다른 변수 백분위수를 플롯으로 그리는 것이다.

넘파이(NumPy)와 판다스가 데이터를 구간화하는 함수를 제공한다.

그림 7.3: Percentiles of weight for a range of height bins.

```
df = df.dropna(subset=['htm3', 'wtkg2'])
bins = np.arange(135, 210, 5)
indices = np.digitize(df.htm3, bins)
groups = df.groupby(indices)
```

dropna 메쏘드는 호명된 칼럼(열)에서 nan가 있는 행을 제거한다. arange 메쏘드는 135부터 210까지 (210은 포함하지 않고) 5만큼 증가하는 구간을 가진 넘파이 배열을 생성한다.

digitize 메쏘드는 df.htm3에 있는 각 값을 담고 있는 구간 인텍스를 계산한다. 결과는 정수 인텍스 넘파이(NumPy) 배열이다. 가장 하위 구간에 속하는 값은 인덱스 0에 매핑된다. 가장 상위 구간에 속하는 값은 len(bins)에 매핑된다.

groupby는 GroupBy 객체를 반환하는 데이터프레임 메쏘드다; for 루프에 사용되고, groups은 그룹 이름과 그룹을 나타내는 데이터프레임을 반복돌린다.

```
for i, group in groups:
    print(i, len(group))
```

이제 각 그룹에 대해서, 평균 신장과 체중 CDF를 계산할 수 있다.

```
heights = [group.htm3.mean() for i, group in groups]
cdfs = [thinkstats2.Cdf(group.wtkg2) for i, group in groups]
```

마지막으로, 신장과 체중에 대한 백분위수를 플롯으로 그릴 수 있다.

```
for percent in [75, 50, 25]:
    weights = [cdf.Percentile(percent) for cdf in cdfs]
    label = '%dth' % percent
    thinkplot.Plot(heights, weights, label=label)
```

그림 7.3에 결과가 나와 있다. 140센티미터에서 200센티미터 사이 두 변수 간의 관계는 대략 선형이다. 이 범위에 99%이상 데이터가 몰려있다. 그래서, 극단값에 대해서 너무 걱정할 필요는 없다.

제 3 절 상관 (Correlation)

상관 (correlation)은 두 변수 간 관계강도를 정량화하는데 사용되는 통계량이다.

상관을 측정하는데 걸림돌은 비교하려는 변수가 동일한 단위로 표현되지 않는다. 그리고 설사 단위가 동일하다고 하더라도, 다른 분포에서 두 변수가 나왔다.

상기 문제에 대한 두가지 일반적인 해결책이 있다.

1. 각 값을 평균에서 떨어진 표준편차 숫자인 **표준점수 (standard scores)**로 변환한다. 이러한 변환은 "피어슨 곱적률 상관계수 (Pearson product-moment correlation coefficient)"가 된다.

2. 각 값을 정렬된 리스트 값 인덱스인 **순위 (rank)**로 변환한다. 이러한 변환은 "스피어만 순위 상관계수 (Spearman rank correlation coefficient)"가 된다.

만약 X가 일련 n개 값, x_i라면, 평균을 빼고 표준편차로 나누어서 표준점수로 전환한다. $z_i = (x_i - \mu)/\sigma$.

분모가 편차다; 평균에서 떨어진 차이. σ로 나누면 편차를 **표준화(standardizes)**한다. 그래서 Z 값은 차원이 없다(단위가 없음). 그리고 분포는 평균 0, 분산 1이 된다.

만약 X가 정규 분포라면, Z도 그렇다. 하지만, 만약 X가 기울어지거나 이상점이 있다면, Z도 그렇다; 이와 같은 경우, 백분위 순위를 사용하는 것이 좀더 강건하다. 만약 새로운 변수, R을 계산한다면, r_i는 x_i의 순위가 되고, X가 무슨 분포든지 관계없이, R 분포는 1에서 n까지 균등분포다.

제 4 절 공분산 (Covariance)

공분산 (Covariance)은 함께 변화하는 두 변수의 경향성 측도다. 만약 두 변수 계열 X and Y가 있다면, 평균에 대한 편자는 다음과 같다.

$$dx_i = x_i - \bar{x}$$

$$dy_i = y_i - \bar{y}$$

$\bar{x}$가 X의 표본 평균이고, $\bar{y}$이 Y의 표본 평균이다. 만약 X와 Y가 함께 변한다면, 편차는 동일 기호를 갖는 경향이 있다.

두 변수를 함께 곱한다면, 곱이 양수가 될 때는 편차가 동일한 부호를 갖을 때고, 음수가 될 때는 반대 부호를 갖을 때다. 그래서, 곱을 합하게 되면 함께 변화하는 경향성 측도가 된다.

공분산은 이들 곱의 평균이다.

$$Cov(X, Y) = \frac{1}{n} \sum dx_i \, dy_i$$

n은 두 계열 변수의 길이다. (두 계열 변수는 길이가 동일해야 한다.)

만약 선형대수학을 공부해따면, `Cov`가 길이로 나눈 편차의 내적(dot product)이다. 그래서 만약 두 벡터가 동일하다면 공분산은 극대화되고, 직교(orthogonal)

하면 0, 반대 방향으로 위치하면 부호가 음수가 된다. thinkstats2는 np.dot을 사용해서 Cov를 효과적으로 구현한다.

```
def Cov(xs, ys, meanx=None, meany=None):
    xs = np.asarray(xs)
    ys = np.asarray(ys)

    if meanx is None:
        meanx = np.mean(xs)
    if meany is None:
        meany = np.mean(ys)

    cov = np.dot(xs-meanx, ys-meany) / len(xs)
    return cov
```

초기 설정값으로 Cov가 표본 평균에서 편차를 계산하거나 이미 계산된 평균값을 인자로 넘길 수 있다. 만약, xs와 ys이 파이썬 시퀀스라면, np.asarray가 넘파이(NumPy) 배열로 전환한다. 만약 이미 넘파이(NumPy) 배열이라면, np.asarray는 아무 것도 하지 않는다.

공분산 구현이 설명 목적으로 간단하게 구현되었다. 넘파이(NumPy)와 판다스에 공분산 구현되어 기능을 제공한다. 하지만, 둘다 여기서 다루어지지 않는 작은 표본 크기에 대한 보정기능을 제공한다. 그리고, np.cov는 공분산 행렬을 반환하는데, 공분산 행렬은 지금 당장 필요한 것 이상이다.

제 5 절 피어슨 상관 (Pearson's correlation)

공분산이 몇몇 연산에서는 유용하지만, 요약 통계량으로는 결코 보고되어지지 않는데 이유는 해석하기기 어렵기 때문이다. 다른 문제점 중에서, 단위가 X와 Y 단위의 곱이다. 예를 들어, 이것이 무엇을 의미하든지, BRFSS 데이터셋에서 체중과 신장 공분산은 113 킬로그림-센티미터가 된다.

이 문제에 한 해결책은 표준편차로 편차를 나누고, 표준 점수 곱을 계산하는 것이다.

$$p_i = \frac{(x_i - \bar{x})}{S_X} \frac{(y_i - \bar{y})}{S_Y}$$

S_X와 S_Y은 X와 Y의 표준편차다. 이들 곱의 평균은 다음과 같다.

$$\rho = \frac{1}{n} \sum p_i$$

혹은 S_X 와 S_Y을 빼내서 ρ를 다시 작성할 수 있다.

$$\rho = \frac{Cov(X,Y)}{S_X S_Y}$$

이 값을 초창기 영향력 있는 통계학자 칼 피어슨(Karl Pearson)을 따라 **피어슨 상관 (Pearson's correlation)**이라고 부른다. 계산하기 쉽고 해석하기도 쉽다. 표준점수는 차원이 없어서 ρ다.

`thinkstats2`에서 구현한 코드가 다음에 있다.

```
def Corr(xs, ys):
    xs = np.asarray(xs)
    ys = np.asarray(ys)

    meanx, varx = MeanVar(xs)
    meany, vary = MeanVar(ys)

    corr = Cov(xs, ys, meanx, meany) / math.sqrt(varx * vary)
    return corr
```

별도로 `np.mean`와 `np.var`을 호출하기 보다 다소 더 효율적으로 `MeanVar`은 평균과 분산을 계산한다.

피어슨 상관은 항상 -1과 1 사이다. (1,-1을 포함한다.) 만약, ρ가 양수라면, 상관이 양(positive)으로 한 변수값이 높으면, 다른 변수값도 높은 경향이 있다는 의미다. ρ가 부(negative)이면, 한 변수가값이 높으면, 다른 변수 값은 낮은 경향이 있다는 의미다.

ρ 크기는 상관 강도를 나타낸다. 만약 ρ가 1 혹은 -1 이면, 변수는 완벽하게 상관되어 있다는 의미로, 만약 변수 하나를 알고 있다면 다른 하나에 대해서 완벽한 예측이 가능하다는 의미다.

현실 세계에서 대부분의 상관은 완벽하지 않지만, 여전히 유용하다. 신장과 체중 상관은 0.51로 비슷한 사람과 관련된 변수와 비교하여 볼 때 강한 상관이다.

제 6 절 비선형 관계 (Nonlinear relationships)

만약 피어슨 상관이 거의 0이라면, 변수 사이에 관계가 없다고 단정하고 싶을 것이다. 하지만, 그러한 결론은 타당하지 않다. 피어슨 상관은 단지 *선형 (linear)* 관계만을 측정한다. 만약 비선형 관계가 있다면, ρ는 비선형 강도를 과소평가하게 된다.

그림 7.4: Examples of datasets with a range of correlations.

그림 7.4은 http://wikipedia.org/wiki/Correlation_and_dependence 위키피디아 사이트에서 가져왔다. 그림에서 의도적으로 생성된 데이터셋에 대한 산점도와 상관 계수가 보여진다.

상단 줄에는 다양한 상관계수를 가진 선형 관계를 보여준다; 상단 행에서 다양한 ρ 값이 어떤 느낌인지 감을 얻을 수 있다. 두번째 행이 다양한 기울기를 가진 완전 상관의 예가 있다. 상관이 기울기와 관련이 없다는 것을 보여준다 (곧 기울기를 추정하는 것에 대해 언급할 것이다.) 세번째 행은 명확하게 관련이 있는 변수를 보여준다. 하지만, 관계가 비선형이라, 상관계수는 0이다.

상기 이야기의 교훈은 상관계수를 맹목적으로 계산하기 전에 데이터를 산점도를 그려서 항상 살펴봐야 한다는 것이다.

제 7 절 스피어만 순위 상관 (Spearman's rank correlation)

피어슨 상관은 변수 사이의 관계가 선형이고, 변수가 대략 정규분포를 따른 다면 잘 동작한다. 하지만, 이상값이 존재하는 경우에는 강건하지 못하다. 스피어만 순위 상관이 대안으로 이상값과 기울어진 분포에 대한 효과를 완화한다. 스피어만 상관을 계산하기 위해서 각 값의 **순위 (rank)**를 계산하는데 정렬된 표본에 인덱스에 해당한다. 예를 들어, 표본 `[1, 2, 5, 7]`에서 값 5에 대한 순위는 3이다. 왜냐하면 정렬된 리스트에서 3번째 등장하기 때문이다. 그리고 나서 피어슨 순위 상관을 계산한다.

thinkstats2는 스피어만 순위 상관을 계산하는 함수가 있다.

```
def SpearmanCorr(xs, ys):
    xranks = pandas.Series(xs).rank()
    yranks = pandas.Series(ys).rank()
    return Corr(xranks, yranks)
```

인자를 판다스 시리즈 객체로 전환해서, 각 값에 대한 순위를 계산하고 시리즈를 반환하는 순위 (`rank`) 메쏘드를 사용할 수 있다. 그리고 나서 `Corr`을 사용해서 순위 상관을 계산한다.

직접적으로 `Series.corr`을 사용해서 스피어만 메쏘드를 명세할 수도 있다.

```
def SpearmanCorr(xs, ys):
    xs = pandas.Series(xs)
    ys = pandas.Series(ys)
    return xs.corr(ys, method='spearman')
```

BRFSS 데이터에 대한 스피어만 순위 상관계수는 0.54로 피어슨 상관계수 0.51 보다 약간 더 높다. 차이에 대한 가능한 이유가 몇가지 있다:

- 만약 관계가 비선형이면, 피어슨 상관은 관계의 강도를 과소추정하는 경향이 있다.
- 만약 두 변수 분포중 하나가 기울거나(skewed) 이상점을 포함하게 되면, 피어슨 상관이 (어느 방향이든지) 영향을 받을 수 있다. 스피어만 상관계수가 좀더 강건하다.

BRFSS 예제에서, 체중 분포가 대략 로그 정규분포라는 것을 알고 있다; 로그 변환을 통해서 정규 분포로 근사해서, 기울어짐이 없다. 기울어짐 효과를 제거하는 또다른 방법은 로그 취한 체중과 신장 사이 피어슨 상관을 계산하는 것이다.

```
thinkstats2.Corr(df.htm3, np.log(df.wtkg2)))
```

결과는 0.53으로 순위 상관계수 0.54에 가깝다. 그래서 이것이 의미하는 바는 체중 분포의 기울어짐이 피어슨과 스미어만 상관 사이 대부분 차이를 설명한다.

제 8 절 상관과 인과 (Correlation and causation)

만약 변수 A와 B가 상관되었다면, 세가지 설명이 가능하다: A가 B의 발생 원인, B가 A의 발생 원인, 혹은 다른 요소가 A와 B의 발생 원인. 이와 같은 설명을 "인과관계 (causal relationships)"라고 부른다.

상관 그 자체로는 가능한 설명이 어느 것인지 구별하지는 못한다. 그래서, 어느 것이 사실인지 여러분에게 알려주지는 못한다. 이 원칙은 종종 다음 상용구로 표현된다. "상관은 원인을 함축하지 않는다. (Correlation does not imply causation)". 애석하게도 자체 위키피디아 페이지도 있다. `http://wikipedia.org/wiki/Correlation_does_not_imply_causation`.

인과 증거를 제공하려면 무엇을 할 수 있을까요?

1. 시간을 사용한다. 만약 A가 B전에 온다면, A가 B를 발생시키는 원인 이지만, 다르게는 아니다. (적어도 인과에 관한 상식적인 이해에 따르면) 사건 순서는 인과 방향을 추론하는데 도움이 된다. 하지만, 다른 무언가 A와 B를 발생시키는 원인이 된다는 가능성을 배제하지 못한다.
2. 확률(randomness)을 사용한다. 만약 큰 표본을 임의로 두 집단으로 나누고 거의 모든 변수의 평균을 계산한다면, 차이는 작을 것으로 기대한다. 만약 한 변수를 제외하고 모든 변수에서 집단이 동일하다면, 허위 관계 (spurious relationship)을 제거할 수 있다.

설사 관련 변수가 무엇인지 모른다고 하더라도 이 방식은 동작한다. 하지만, 관련 변수를 알고 있다면 더 잘 동작하는데 이유는 집단이 동일하다는 것을 확인할 수 있기 때문이다.

이와 같은 아이디어가 **무작위 대조군 시험 (randomized controlled trial)**의 동기가 된다. 시험 대상이 임의로 두 (혹은 그 이상) 집단에 배속된다: **처리군 (treatment group)**은 신약 같은 일종의 개입을 받고, **대조군 (control group)**은 개입을 받지 않거나 효과가 알려진 또다른 처리를 받는다.

무작위 대조군 시험은 인과관계를 시연하는 가장 신뢰성 있는 방법이고, 과학-기반 의학의 초석이다. (`http://wikipedia.org/wiki/Randomized_controlled_trial` 참조).

불행하게도 대조군 시험은 실험실 과학, 의학, 그리고 몇몇 학문분야에서만 가능하다. 사회 과학에서, 대조군 실험이 드문데 이유는 불가능하거나 비윤리적이기 때문이다.

또 다른 대안은 **자연 실험 (natural experiment)**을 살펴보는 것인데, 다른 처리 (treatment)가 유사한 집단에 적용된다. 자연 실험의 한가지 위험성은 명확하지 않는 방식으로 집단이 다를지도 모른다는 것이다. 이 주제에 대해서 웹사이트를 참조한다. `http://wikipedia.org/wiki/Natural_experiment`.

몇몇 경우에, **회귀 분석 (regression analysis)**을 사용해서 인과 관계를 추론할 수 있다. 11장에서 다룰 것이다.

제 9 절 연습 문제

A solution to this exercise is in `chap07soln.py`.

Exercise 7.1 Using data from the NSFG, make a scatter plot of birth weight versus mother's age. Plot percentiles of birth weight versus mother's age. Compute Pearson's and Spearman's correlations. How would you characterize the relationship between these variables?

제 10 절 용어 사전

- 산점도 (scatter plot): 두 변수 사이 관계를 데이터 각 행마다 점을 찍어 보임으로써 시각화.
- 지터 (jitter): 시각화 목적으로 데이터에 추가되는 확률 잡음.
- 포화 (saturation): 다수의 점이 겹쳐지게 되어서 발생하는 정보 손실.

- 상관 (correlation): 두 변수 사이 관계 강도를 측정하는 통계량.
- 표준화(standardize): 변수 집합 값을 변환해서 평균 0, 분산 1로 만듦.
- 표준 점수 (standard score): 표준화된 값으로, 평균에서 떨어진 표준편차로 표현.
- 공분산 (covariance): 함께 변화하는 두 변수 경향성 측도.
- 순위 (rank): 인덱스로, 정렬된 목록에 요소가 나타나는 순서.
- 무작위 대조군 시험: 실험계획 (experimental design)으로 대상이 무작위 집단으로 나눠지고, 다른 집단에 다른 처리(treatment)가 주어진다.
- 처리군 집단 (treatment group): 대조군 시험에서 일종의 개입(intervention)을 받는 집단.
- 대조군 집단 (control group): 대조군 시험에서 처리를 받지 않거나 이미 효과가 알려진 처리를 받는 집단.
- 자연 실험 (natural experiment): 대상이 집단으로 자연 구분되는 현상을 이용하는 실험계획. 여기서 집단은 최소한 근사적으로 무작위가 된다.

제 8 장

추정 (Estimation)

이번 장에서 사용되는 코드는 `estimation.py`에 있다. 코드를 다운로드하고 작업하는 것에 대한 정보는 2을 참조한다.

제 1 절 추정 게임

게임 한판합시다. 저자는 생각하는 분포가 있고, 여러분은 그 분포가 무엇인지 맞춰야 한다. 힌트를 두개 준다: 정규분포이고, 정규분포에서 나온 표본이 다음과 같다.

```
[-0.441, 1.774, -0.101, -1.138, 2.975, -2.138]
```

상기 분포의 평균 모수, μ가 무얼까요?

한가지 선택은 μ 추정값(estimate)으로 표본 평균 $\bar{x}$을 사용한다. 상기 예제에서 $\bar{x}$가 0.155 으로, μ = 0.155 추정하는 것은 합리적이다. 이러한 과정을 **추정(estimation)**이라고 부른다. 사용한 통계량 (표본 평균)은 **추정량(estimator)**이라고 부른다.

μ를 추정하는데 표본 평균을 사용하는 것이 너무 명확해서 합리적인 다른 대안을 상상하기는 어렵다. 하지만, 이상값을 도입해서 게임을 바꾸는 것을 가정하자.

저자가 생각하는 분포가 있다. 정규분포이고, 잘못된 자리에 종종 소수점을 넣는 신뢰가 가지 않는 조사원이 수집한 표본이 다음에 있다.

```
[-0.441, 1.774, -0.101, -1.138, 2.975, -213.8]
```

이제 μ 추정값은 무엇이 될까요? 만약 표본 평균을 사용한다면, 추정치는 -35.12가 된다. 이것이 최선의 선택일까요? 대안이 무엇이 될까요?

한가지 선택지는 이상점을 식별하고 버리고 나서, 나머지를 가지고 표본 평균을 계산한다. 다른 선택지는 중위수를 추정량으로 사용한다.

어느 추정량이 가장 좋은가는 상황에 따라 달라지고(예를 들어, 이상값이 있느냐), 목적이 무엇이냐에 달려있다. 오류를 최소화하려고 하는지 혹은 정답을 얻는 가능성을 극대화하려고 하는지.

만약 이상값이 없다면, 표본 평균은 **누적평균제곱오차 (mean squared error), MSE**를 최소화한다. 즉, 만약 게임을 여러번하고, 매번 오차 $\bar{x} - \mu$를 계산하면, 표본 평균은 다음을 최소화 한다.

$$MSE = \frac{1}{m} \sum (\bar{x} - \mu)^2$$

m은 추정 게임을 수행한 횟수다. $\bar{x}$를 계산하는데 사용된 표본 크기인 n와 혼동하면 안된다.

다음에 함수가 있는데 추정 게임을 모사하여 MSE 제곱근인, 누적평균제곱오차의 제곱근(root mean squared error, RMSE)을 계산한다.

```
def Estimate1(n=7, m=1000):
    mu = 0
    sigma = 1

    means = []
    medians = []
    for _ in range(m):
        xs = [random.gauss(mu, sigma) for i in range(n)]
        xbar = np.mean(xs)
        median = np.median(xs)
        means.append(xbar)
        medians.append(median)

    print('rmse xbar', RMSE(means, mu))
    print('rmse median', RMSE(medians, mu))
```

다시 한번, `n`은 표본 크기다. `m`은 게임을 실행한 횟수다. `means`은 $\bar{x}$에 기반한 추정값 리스트다. `medians`은 중위수 리스트다.

다음에 RMSE를 계산하는 함수가 있다.

```
def RMSE(estimates, actual):
    e2 = [(estimate-actual)**2 for estimate in estimates]
    mse = np.mean(e2)
    return math.sqrt(mse)
```

`estimates`는 추정값 리스트다; `actual`은 추정되는 실제 값이다. 실무에서는 물론 `actual`을 모른다; 만약 알고 있다면, 추정할 필요가 없을 것이다. 실험 목적은 두 추정량 성능을 비교하는 것이다.

코드를 실행하면, 표본 평균 RMSE가 0.41로 의미하는 바는 만약 $n = 7$ 표본에 기반하여 $\bar{x}$를 사용해서 분포 평균을 추정한다면, 평균 0.41만큼 떨어졌다고 예측된다. 중위수를 사용하여 평균을 추정하면 RMSE 0.53을 산출하는데 적어도 이 사례에서 $\bar{x}$가 더 낮은 RMSE를 뽑아낸다고 확인해준다.

MSE 최소화는 좋은 특성이다. 하지만, 항상 최선의 전략은 되지 못한다. 예를 들어, 조선소에서 바람 속도 분포를 추정한다고 가정하다. 만약 추정값이 너무 높으면, 시설물을 과도하게 짓게되어서 비용이 상승한다. 하지만, 너무 낮게 추정한다면, 구조물이 붕괴할 수도 있다. 오류 함수로 비용이 좌우 대칭이 아니기 때문에, MSE 최소화가 항상 좋은 전략은 아니다.

다른 예제로, 6면 주사위 세개를 던져서 합계를 예측하는 것을 가정하자. 정확하게 합계를 맞춘다면, 상을 받게 된다; 맞추지 못하면 아무 것도 없다. 이경우, MSE를 최소화하는 값은 10.5가 된다. 하지만, 좋지 못한 추정값이 된다. 왜냐하면 주사위 세개를 던져 합은 결코 10.5가 되지 못한다. 이 게임에서, 맞출 가장 높은 확률을 가진 추정량이 필요하다. 그것은 **최대우도추정량 (maximum likelihood estimator, MLE**이다. 만약 10 혹은 11일 선택하면, 우승 가능성이 8분의 1이 되고 할 수 있는 최선이 된다.

제 2 절 분산 추정

저자가 마음에 두고 있는 분포가 있다. 정규분포고, 다음에 (친숙한) 표본이 있다.

```
[-0.441, 1.774, -0.101, -1.138, 2.975, -2.138]
```

상기 분포의 분산 σ^2가 무엇일까요? 다시 한번, 명백한 선택은 표본 분산, S^2를 추정량으로 사용하는 것이다.

$$S^2 = \frac{1}{n}\sum(x_i - \bar{x})^2$$

표본크기가 큰 경우, S^2가 적절한 추정량이 되지만, 작은 표본에 대해서 너무 작아지는 경향이 있다. 이와 같은 불행한 성질로 인해서, **편의 (biased)** 추정량이라고 부른다. 만약 추정을 많이 반복한 뒤에 기대 총 (혹은 평균) 오류가 0이라면, 추정량은 **불편의(unbiased)**다.

다행스럽게도, σ^2에 대한 불편 추정량인 또 다른 간단한 통계량이 있다.

$$S_{n-1}^2 = \frac{1}{n-1}\sum(x_i - \bar{x})^2$$

S^2가 왜 편의가 있고, S_{n-1}^2가 왜 불편 추정량인지에 대한 설명은 웹사이트 `http://wikipedia.org/wiki/Bias_of_an_estimator` 참조바랍니다.

이 추정량에 대한 가장 큰 문제점은 명칭과 기호가 일관되지 못하게 사용된다는 데 있다. "표본 분산"이 S^2 혹은 S_{n-1}^2, 그리고 기호 S^2이 한쪽 혹은 양쪽에 사용될 수 있다.

추정 게임을 모의시험하고 S^2 와 S_{n-1}^2의 성능을 시험하는 함수가 다음에 있다.

```
def Estimate2(n=7, m=1000):
    mu = 0
    sigma = 1

    estimates1 = []
    estimates2 = []
    for _ in range(m):
        xs = [random.gauss(mu, sigma) for i in range(n)]
        biased = np.var(xs)
        unbiased = np.var(xs, ddof=1)
        estimates1.append(biased)
        estimates2.append(unbiased)

    print('mean error biased', MeanError(estimates1, sigma**2))
    print('mean error unbiased', MeanError(estimates2, sigma**2))
```

한번 더, `n`가 표본 크기이고, `m`은 게임을 수행한 횟수다. `np.var`는 기본 설정으로 S^2을 계산하고, 만약 인자로 `ddof=1`을 넣으면 S_{n-1}^2을 계산한다. `ddof`는 "델타 자유도 (delta degrees of freedom)"의 축약어다. 웹사이트에서 자세한 사항 참조한다. `http://en.wikipedia.org/wiki/Degrees_of_freedom_(statistics)`

`MeanError`는 추정값과 실제값 사이 평균 차이를 계산한다.

```
def MeanError(estimates, actual):
    errors = [estimate-actual for estimate in estimates]
    return np.mean(errors)
```

코드를 실행하면, S^2에 대한 평균 오차는 -0.13이다. 기대한 것처럼, 편의 추정량이 너무 적은 경향이 있다. S_{n-1}^2에 대해서, 평균 오차는 0.014로 10배 더 적다. `m`이 커감에 따라, 평균 오차 S_{n-1}^2가 0에 수렴해간다.

MSE나 편의(bias) 같은 성질은 많은 추정 게임에 기반한 장기 기대가 된다. 이장에서와 같은 모의시험을 수행해서, 추정량을 비교하고, 바람직한 성질을 갖는지 점검할 수 있다.

하지만, 추정량을 실제 데이터에 적용했을 때, 단지 하나 추정값만 얻는다. 추정값이 편의가 없다고 말하는 것이 의미가 없을 수 있다; 불편(unbiased)하다는 것은 추정량(estimator)의 성질이지 추정값(estimate)의 성질은 아니다.

적절한 성질을 갖는 추정량을 선택하고 추정값을 생성하고 나면, 다음 단계는 추정값의 불확실을 특성화해야 한다. 다음 절의 주제다.

제 3 절 표집 분포 (Sampling distributions)

야생동물 보호구역에서 고릴라를 연구하고 있는 과학자를 가정해보자. 보호구역에 서식하고 있는 성인 암컷 고릴라 평균 체중을 알고자 한다. 체중을 측정하려고 한다면, 고릴라를 진정시켜야 하는데, 위험하고, 비용이 많이 들며, 아마도 고릴라 자체도 해로울 수 있다. 하지만, 체중 정보 수집이 중요하다면, 9마리 고릴라를 표본으로 체중을 측정하는 것은 용인될지 모른다. 보호구역 모집단은 알고 있다고 가정하자. 그래서 성인 암컷 대표 표본을 고를 수 있다. 미지의 모집단 평균 μ를 추정하는데 표본 평균 $\bar{x}$를 사용할 수 있다.

고릴라 암컷 9마리 체중을 재서, 표본평균 $\bar{x} = 90$ kg과 표본 표준편차 $S = 7.5$ kg을 찾았다. 표본평균은 μ에 대한 불편 추정량이고, 장기적으로 MSE를 최소화한다. 그래서 결과를 요약하는 단 하나 추정값을 보고한다면, 90 kg을 보고할 것이다.

하지만, 이 추정값에 대해서 얼마나 신뢰를 가져야할까? 훨씬 더 큰 모집단에서 단지 $n = 9$ 마리 고릴라만 체중을 측정했다면, 우연히도 운이 좋지 못하다면 가장 무거운 (혹은 가장 가벼운) 고릴라만 고를 수 있다. 확률 선택(random selection)에 의해서 발생되는 추정값의 변동을 **표집 오차 (sampling error)**라고 한다.

표집 오차를 정량화하기 위해서, 가설(hypothetical) 값 μ와 σ를 갖는 표본추출 과정을 모의시험하고, $\bar{x}$가 얼마나 변화하는지 살펴볼 수 있다.

모집단 μ와 σ 실제값을 모르기 때문에, $\bar{x}$와 S을 추정값으로 사용한다. 그래서 정답을 찾고 있는 질문은 "만약 μ와 σ의 실제값이 90 kg와 7.5 kg이고, 동일한 실험을 많이 수행한다면, 추정된 평균 $\bar{x}$가 얼마나 변화할까요?"

다음 함수가 질문에 답을 한다.

```
def SimulateSample(mu=90, sigma=7.5, n=9, m=1000):
    means = []
    for j in range(m):
```

그림 8.1: Sampling distribution of $\bar{x}$, with confidence interval.

```
        xs = np.random.normal(mu, sigma, n)
        xbar = np.mean(xs)
        means.append(xbar)

    cdf = thinkstats2.Cdf(means)
    ci = cdf.Percentile(5), cdf.Percentile(95)
    stderr = RMSE(means, mu)
```

`mu`와 `sigma`는 모수에 대한 *가설 (hypothetical)* 값이다. `n`이 표본 크기로 체중을 측정한 고릴라 숫자다. `m`은 모의 시험을 수행한 횟수다.

매번 반복할 때마다, 주어진 모수를 가진 정규분포에서 `n`개 값을 골라 표본 평균 `xbar`를 계산한다. 1000번 모의시험하고 나서, 추정값 분포 `cdf`를 계산한다. 결과가 그림 8.1에 나타나 있다. 분포를 추정값의 **표집 분포 (sampling distribution)**라고 부른다. 만약 반복해서 실험을 계속한다면, 추정값이 얼마나 변화하는지 보여준다.

표집 분포 평균이 가정값(hypothetical value) μ와 매우 가깝다. 따라서, 평균적으로 실험이 정답을 산출한다는 의미다. 1000회 반복한 후에는 가장 적은 값이 82 kg, 가장 큰 값이 98 kg이다. 이 범위가 함의하는 바는 추정값이 8 kg 만큼 차이가 있을 수 있다는 것이다.

표집 분포를 요약하는 두 가지 흔한 방법이 있다.

- **표준 오차 (Standard error SE)**는 평균적으로 추정값이 얼마나 차이가 있을지 측정하는 측도다. 각 모의 실험마다, 오차 $\bar{x} - \mu$를 계산하고 나서 누적평균제곱오차에 제곱근(root mean squared error, RMSE)을 씌워 계산한다. 상기 예제에서는 대략 2.5 kg이 된다.
- **신뢰 구간 (confidence interval, CI)**은 표집 분포를 주어진 비율로 포함할 범위가 된다. 예를 들어, 신뢰 구간 90%는 5번째부터 95번째 백분위수 범위가 된다. 상기 예제에서, 90% CI는 $(86, 94)$ kg가 된다.

표준 오차와 신뢰구간은 또한 혼동의 원천이기도 하다.

- 종종 표준 오차(standard error)와 표준편차(standard deviation)를 혼동한다. 표준 편차는 측정량에 내재한 변동성을 기술한다; 상기 예제에서, 고릴라 체중의 표준 편차는 7.5 kg이다. 표준 오차는 추정값에 내재한 변동성을 기술한다. 상기 예제에서, 표본 9 측정에 기반한 평균의 표준 오차는 2.5 kg 이다.

  차이점을 기억하는 방법은, 표본 크기가 증가함에 따라, 표준 오차는 점점 더 작아진다; 표준 편차는 그렇지 않다.

- 90% 확률로 실제 모수 μ가 90% 신뢰구간 내에 속한다고 생각하곤 한다. 슬프게도, 이것은 사실이 아니다. 이와 같은 주장을 하려고 한다면, 베이즈 방법(Bayesian method)을 사용해야 한다.(원저작자 책, *Think Bayes*을 참조).

  표집 분포는 다른 질문에 대한 답이다; 만약 실험을 반복한다면, 추정값이 얼마나 변화할 것인지 정보를 제시함으로써 추정값이 얼마나 신뢰할만한지 정보를 제공한다.

신뢰 구간과 표준 오차는 표집 오차만 정량화한다는 것을 기억하는 것이 중요하다; 즉, 모집단 일부만 측정함으로써 생기는 오차. 표집 분포가 다른 오차를 설명하지는 않는다. 다른 오차는 표집 편의(sampling bias)와 측정 오차 (measurement error)가 포함되고 다음 절의 주제다.

제 4 절 표집 편의 (Sampling bias)

자연보호구역에 고릴라 체중 대신에, 도시에 거주하고 있는 여성 평균 체중을 알고자 한다고 가정하자. 여성 대표 표본을 골라 체중을 측정하는 허가가 날 것 같지는 않다.

간단한 대안은 "전화 표집 (telephone sampling)"이 될 것이다; 즉, 전화번호부에서 임의 번호를 골라, 전화하고, 성인 여성과 통화하여, 체중이 얼마인지 조사하는 것이다.

전화 표집은 분명한 한계가 있다. 예를 들어, 표본이 전화번호가 전화번호부에 등재된 사람으로 국한된다. 그래서 전화가 없는 사람(평균보다 더 가난할지 모르는 사람)과 전화번호부에 등재되지 않은 사람(훨씬 더 부자일 수 있는 사람)은 제외된다. 또한, 낮 시간에 집으로 전화를 걸게 되면, 직장을 가진 사람이 표본에 있을 가능성은 줄어든다. 그리고 만약 전화에 응답한 사람만 표본으로 추출한다면, 전화를 공용으로 사용하는 사람을 표본으로 덜 추출할 것 같다.

수입, 실업, 가구 규모 같은 요인이 체중과 관련이 있다면(관련이 있을 듯하다.), 조사 결과가 어떻게든지 영향을 받을 수 있다. 이러한 문제를 **표집 편의 (sampling bias)**라고 부른다. 왜냐하면 표집과정(sampling process)의 한 특성이기 때문이다.

이러한 표집 과정은 또한 자기 선택(self-selection)에 취약한데 일종의 표집 편의다. 일부 사람은 질문에 대답을 거부한다. 그리고 만약 거부하는 경향이 체중과 관련된다면, 결과에도 영향을 줄 수 있다.

마지막으로, 체중을 측정하기보다, 체중이 얼마인지 설문한다면, 결과가 정확하지 않을 수도 있다. 만약 실제 본인 체중에 불편하다면, 심지어 도움되는 응답자

조차도 체중을 올리거나 내릴지도 모른다. 그리고 모든 응답자가 도움이 되는 것은 아니다. 이와 같은 비정확성이 **측정 오차 (measurement error)** 사례가 된다.

추정량을 보고할 때, 표집 오차를 정량화하기 위해서 표준 오차 혹은 신뢰구간, 혹은 모두 보고하는 것이 유용한다. 하지만, 표집 오차는 단지 오차의 단지 하나일 뿐이고, 종종 가장 큰 것이 아니라는 것을 기억하는 것이 중요하다.

제 5 절 지수분포 (Exponential distributions)

한번더 추정 게임을 시도해보자. *저자가 마음속에 분포를 하나 염두에 두고 있다.* 지수분포로, 다음에 표본이 있다.

```
[5.384, 4.493, 19.198, 2.790, 6.122, 12.844]
```

상기 분포 모수 λ는 무엇이라고 생각하나요?

일반적으로 지수분포 평균은 $1/\lambda$다. 그래서 역산해서 다음을 선택할 수 있다.

$$L = 1/\bar{x}$$

L은 λ의 추정량이다. 그리고 보통 다른 추정량과 다르다; 또한 최우추정량(maximum likelihood estimator)이다. (`http://wikipedia.org/wiki/Exponential_distribution#Maximum_likelihood` 참조). 그래서, 만약 정확하게 λ 추측 확률을 극대화하려고 한다면, L이 나갈 방향이 된다.

하지만, $\bar{x}$가 이상점이 존재하는 경우 강건하지 않다는 것을 알고 있다. 그래서, L도 동일한 문제를 작고 있다고 예상한다.

표본 중위수에 기반한 대안을 선택할 수 있다. 지수 분포 중위수는 $\ln(2)/\lambda$이다. 다시 역산해서 계산하면, 추정량을 다음과 같이 정의한다.

$$L_m = \ln(2)/m$$

여기서, m은 표본 중위수다.

이들 추정량 성능을 시험하기 위해서, 표집 과정을 모의 시험할 수 있다.

```
def Estimate3(n=7, m=1000):
    lam = 2

    means = []
    medians = []
    for _ in range(m):
```

```
        xs = np.random.exponential(1.0/lam, n)
        L = 1 / np.mean(xs)
        Lm = math.log(2) / thinkstats2.Median(xs)
        means.append(L)
        medians.append(Lm)

    print('rmse L', RMSE(means, lam))
    print('rmse Lm', RMSE(medians, lam))
    print('mean error L', MeanError(means, lam))
    print('mean error Lm', MeanError(medians, lam))
```

$\lambda = 2$로 실험을 하면, L RMSE는 1.1 이 된다. 중위수 기반 추정량 L_m, RMSE는 1.8 이 된다. 실험으로부터 L 이 MSE를 최소화했는지 분간할 수 없다. 하지만 적어도 L_m 보다 더 좋은 듯 하다.

슬프게도 두 추정량 모두 편의가 있다. L에 대해 평균 오차는 0.33; L_m에 대해 평균 오차는 0.45 다. 그리고 어느 것도 `m`이 증가함에 따라 0 으로 수렴하지 않는다.

$\bar{x}$는 분포 평균의 불편 추정량 $1/\lambda$임이 밝혀졌다. 하지만, L은 λ의 불편 추정량은 아니다.

제 6 절 연습문제

For the following exercises, you might want to start with a copy of `estimation.py`. Solutions are in `chap08soln.py`

Exercise 8.1 In this chapter we used $\bar{x}$ and median to estimate μ, and found that $\bar{x}$ yields lower MSE. Also, we used S^2 and S^2_{n-1} to estimate σ, and found that S^2 is biased and S^2_{n-1} unbiased.

Run similar experiments to see if $\bar{x}$ and median are biased estimates of μ. Also check whether S^2 or S^2_{n-1} yields a lower MSE.

Exercise 8.2 Suppose you draw a sample with size $n = 10$ from an exponential distribution with $\lambda = 2$. Simulate this experiment 1000 times and plot the sampling distribution of the estimate L. Compute the standard error of the estimate and the 90% confidence interval.

Repeat the experiment with a few different values of n and make a plot of standard error versus n.

Exercise 8.3 In games like hockey and soccer, the time between goals is roughly exponential. So you could estimate a team's goal-scoring rate by

observing the number of goals they score in a game. This estimation process is a little different from sampling the time between goals, so let's see how it works.

Write a function that takes a goal-scoring rate, `lam`, in goals per game, and simulates a game by generating the time between goals until the total time exceeds 1 game, then returns the number of goals scored.

Write another function that simulates many games, stores the estimates of `lam`, then computes their mean error and RMSE.

Is this way of making an estimate biased? Plot the sampling distribution of the estimates and the 90% confidence interval. What is the standard error? What happens to sampling error for increasing values of `lam`?

제 7 절 용어 사전

- 추정 (estimation): 표본에서 분포 모수를 추정하는 과정.
- 추정량 (estimator): 모수를 추정하는데 사용되는 통계량.
- 누적평균제곱오차 (mean squared error, MSE): 추정 오차 측도.
- 제곱근 평균제곱오차 (root mean squared error, RMSE): MSE 제곱근으로 일반적인 오차 규모를 좀더 유의미하게 표현.
- 최우추정량 (maximum likelihood estimator, MLE): 가장 올바를 것 같은 점추정값을 계산하는 추정량.
- (추정량의) 편의 (bias of an estimator): 반복되는 실험을 평균낼 때, 실제 모수 값보다 높은 혹은 낮은 추정량의 경향성.
- 표집 오차 (sampling error): 우연에 의한 변동과 한정된 표본 크기 때문에 추정값에 생기는 오차.
- 표집 편의 (sampling bias): 모집단을 대표하지 못하는 표집 과정 때문에 추정값에 생기는 오차.
- 측정 오차 (measurement error): 데이터를 수집하고 기록하는 부정확성 때문에 추정값에 생기는 오차.
- 표집 분포 (sampling distribution): 만약 실험을 많이 반복한다면 생기는 통계량 분포.

- 표본 오차 (standard error): 추정값의 RMSE로 표집 오차 (하지만 다른 오차는 제외) 때문에 생기는 변동성을 정량화한다.
- 신뢰구간 (confidence interval): 만약 실험을 많이 반복하면 추정량 예상 범위를 표현하는 구간.

제 9 장

가설 검정 (Hypothesis testing)

이번 장에서 사용되는 코드는 `hypothesis.py`에 있다. 코드를 다운로드하고 작업하는 것에 대한 정보는 2을 참조한다.

제 1 절 전통적 가설 검정 (Classical hypothesis testing)

NSFG에서 데이터를 탐색하면서, 첫째 아이와 첫째가 아닌 아이들 간 차이를 포함해서 몇가지 "외관 효과 (apparent effects)"를 봤다. 지금까지 액면 그대로 이러한 효과를 받아들였다; 이번 장에서 이를 검정한다.

다루고자 하는 근본적인 질문은 표본에서 본 효과가 더 큰 모집단에서 나타날 것인가다. 예를 들어, NSFG 표본에서 첫째 아이와 첫째 아이가 아닌 아이들에 대한 평균 임신 기간에 차이를 봤다. 알고자 하는 것은 이 효과가 미국 여성에 대한 진정한 차이를 반영하는지 혹은, 우연히 표본에서 생겨난 것이냐다.

피셔 귀무 가설 검정(Fisher null hypothesis testing), 네이만 피어슨 의사결정 이론(Neyman-Pearson decision theory), 베이즈 추론(Bayesian inference)[1]을 포함해서 질문을 구성하는 방법이 몇가지 있다. 여기 제시하는 것은 대부분의 사람이 실무에서 사용하는 세가지 방법의 일부분으로 **전통적 가설 검정 (classical hypothesis testing)**이라고 저자가 작명한 것이다.

전통적 가설 검정의 목적은 다음 질문에 답하는 것이다. "표본과 명백한 효과가 주어졌다면, 우연히 그런 효과를 목도하는 확률이 얼마인가?" 다음에 질문에 답을 하는 방법이 있다.

[1] 베이즈 추론에 대한 좀더 많은 정보는 후속해서 출간되는 *Think Bayes*를 참조한다.

- 첫번째 단계는 **검정 통계량 (test statistic)**을 선택해서 외관 효과 크기를 정량화한다. NSFG 예제에서, 명백한 효과는 첫번째 아이와 첫째가 아닌 아이들 사이에 임신 기간에 차이가 된다. 그래서, 검정 통계량에 대한 자연스러운 선택이 두 집단 사이에 평균 차이가 된다.
- 두번째 단계는 **귀무 가설 (null hypothesis)**을 정의하는 것으로, 명백한 효과가 사실이 아니라는 가정에 기반한 시스템 모형이다. NSFG 예제에서, 귀무 가설은 첫째 아이와 첫째가 아닌 아이들 사이에 차이가 없다가 된다; 즉, 두 집단 임신 기간은 동일한 분포다.
- 세번째 단계는 **p-값(p-value)**을 계산하는 것으로, 만약 귀무 가설이 사실이라면 명백한 효과를 볼 확률값이 된다. NSFG 예제에서, 평균에 실제 차이를 계산하고 나서, 귀무 가설 아래에서 차이를 크게 혹은 더 크게 볼 확률을 계산한다.
- 마지막 단계는 결과를 해석하는 것이다. 만약 p-값이 낮다면, 효과에 **통계적 유의성 (statistically significant)**이 있다고 한다. 우연히 발생할 것 같지 않다는 의미가 된다. 이 경우 더 큰 모집단에서 효과가 나타날 것 같다고 추론한다.

이 과정의 로직(logic)은 모순(contradiction)에 의한 증명과 유사하다. 수학 명제 (mathematical statement) A를 증명하기 위해, 일시적으로 A 가 거짓이라고 가정한다. 만약 가정이 모순을 이끌게 되면, A 는 사실 참이어야 한다고 결론낸다.

유사하게, “효과가 사실 (This effect is real)” 같은 가설을 검정하기 위해서, 일시적으로 효과가 없다고 가정한다. 이것이 귀무가설이다. 이 가정에 기반해서, 외관 효과 확률을 계산한다. 이것이 p-값이다. 만약 p-값이 작다면, 귀무 가설이 사실이 아닐 것 같다고 결론낸다.

제 2 절 HypothesisTest

`thinkstats2`에는 전통적 가설 검정 구조를 표현하는 `HypothesisTest` 클래스가 있다.

다음에 클래스 정의가 있다.

```
class HypothesisTest(object):

    def __init__(self, data):
        self.data = data
        self.MakeModel()
        self.actual = self.TestStatistic(data)
```

```
    def PValue(self, iters=1000):
        self.test_stats = [self.TestStatistic(self.RunModel())
                           for _ in range(iters)]

        count = sum(1 for x in self.test_stats if x >= self.actual)
        return count / iters

    def TestStatistic(self, data):
        raise UnimplementedMethodException()

    def MakeModel(self):
        pass

    def RunModel(self):
        raise UnimplementedMethodException()
```

HypothesisTest는 추상 부모 클래스로 메쏘드 몇개에 대한 완전한 정의와 다른 메쏘드를 위한 자리잡기(place-keeper) 기능 제공한다. HypothesisTest에 기반한 자식 클래스는 __init__와 PValue 을 상속받고, TestStatistic, RunModel, 그리고 선택옵션으로 MakeModel 메쏘드를 제공한다.

__init__은 적절한 어떤 형식의 데이터도 받아들인다. MakeModel 호출해서 귀무 가설을 표현을 구축하고 나서, 데이터를 TestStatistic에 전달하고 표본 효과 크기를 계산한다.

PValue는 귀무 가설 아래에서 외관 효과 확률을 계산한다. 매개 변수로 iters을 받는데 실행할 모의 시험 횟수다. 첫번째 행은 모의 시험 데이터를 생성하고, 검정 통계량을 계산하고, test_stats에 저장한다. 결과는 관측된 검정 통계량 self.actual와 동일하거나 큰 test_stats 에 있는 비율이다.

간단한 예제[2]로, 250번 동전을 던져서 앞면 140회, 뒷면 110회 나왔다고 가정하자. 이 결과에 기초하여, 동전에 편의가 있는지 의심할지도 모른다; 즉, 좀더 앞면이 나올 것 같다. 이 가설을 검정하기 위해서, 확률을 계산해서 만약 동전이 정말 공정하다면, 그런 차이가 있는지 살펴본다.

```
class CoinTest(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):
        heads, tails = data
        test_stat = abs(heads - tails)
        return test_stat

    def RunModel(self):
```

[2] 인용 출처: MacKay, *Information Theory, Inference, and Learning Algorithms*, 2003.

```
        heads, tails = self.data
        n = heads + tails
        sample = [random.choice('HT') for _ in range(n)]
        hist = thinkstats2.Hist(sample)
        data = hist['H'], hist['T']
        return data
```

모수 data는 정수 짝이다: 앞면 숫자, 뒷면 숫자. 검정 통계량은 둘 사이 절대값 차이다. 그래서 self.actual은 30 이다.

RunModel은 동전이 정말 공정하다고 가정하고 동전던지기를 모의 시험한다. 250번 동전던지기 표본을 생성하고, Hist를 사용해서 앞면과 뒷면 횟수를 계수하고 나서, 정수 짝을 반환한다.

이제 해야할 일은 CoinTest 인스턴스화 하고, PValue를 호출한다.

```
    ct = CoinTest((140, 110))
    pvalue = ct.PValue()
```

결과는 약 0.07 이 된다. 만약 동전이 공정하다면, 30 만큼 차이는 이번에 약 7% 확률로 기대할 수 있다.

이러한 결과를 어떻게 해석해야 할까요? 통상 관례로, 5%가 통계적 유의성의 임계치다. 만약 p-값이 5%보다 적다면, 효과가 유의적이라고 생각하고; 그렇지 않다면, 반대로 생각한다.

하지만, 5% 선택은 작위적이고, (나중에 살펴보겠지만) p-값은 검정 통계량과 귀무 가설 모형에 의존성이 있다. 그래서, p-값이 정밀한 측정으로 생각하지는 말아야 한다.

p-값을 해석함에 있어 저자가 추전하는 것은 p-값 크기에 따라 달리하는 것이다: 만약 p-값이 1%보다 작다면 효과가 우연에 의한 것일 것 같지는 않다; 만약 p-값이 10%보다 크다면, 효과는 우연으로 설명될 수 있을 것 같다. p-값이 1%과 10% 사이라면, 경계값으로 생각해야한다. 그래서, 상기 사례를 통해서는 동전에 편의가 있는지 없는지 데이터가 강력한 증거를 주지는 못한다고 결론낸다.

제 3 절 평균 차이 검정 (Testing a difference in means)

검정할 가장 일반적인 효과중의 하나가 두 그룹 사이 평균 차이다. NSFG 데이터에서 첫번째 아이에 대한 평균 임신기간이 약간 더 길었고, 평균 출산 체중은 약간 더 적었다. 이제 이러한 효과가 통계적으로 유의적인지 살펴보자.

이 예제에서, 귀무가설은 두 집단에 대한 분포가 동일하다는 것이다. 귀무 가설을 모형화하는 한 방법은 **순열(permutation)**에 의한 것이다; 즉, 첫번째 아이와

첫째가 아닌 아이들에 대한 값을 뽑아서 뒤섞어서, 두 집단을 하나의 큰 집단으로 처리한다.

```
class DiffMeansPermute(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):
        group1, group2 = data
        test_stat = abs(group1.mean() - group2.mean())
        return test_stat

    def MakeModel(self):
        group1, group2 = self.data
        self.n, self.m = len(group1), len(group2)
        self.pool = np.hstack((group1, group2))

    def RunModel(self):
        np.random.shuffle(self.pool)
        data = self.pool[:self.n], self.pool[self.n:]
        return data
```

data는 한쌍 시퀀스로 각 시퀀스는 각 집단이 된다. 검정 통계량은 평균에 있어 절대값 차이다.

MakeModel은 집단 크기 n과 m을 기록하고 집단을 넘파이(NumPy) 배열 self.pool에 결합한다.

RunModel은 합동값(pooled value)을 뒤섞고 두 집단 n과 m으로 쪼개서 귀무가설을 모의 시험한다. 항상 그렇지만, RunModel에서 반환되는 값은 관측 데이터와 동일한 형식이다.

임신 기간에 차이를 검정하기 위해서, 다음을 실행한다.

```
live, firsts, others = first.MakeFrames()
data = firsts.prglngth.values, others.prglngth.values
ht = DiffMeansPermute(data)
pvalue = ht.PValue()
```

MakeFrames는 NSFG 데이터를 읽고, 모든 정상출산, 첫째 아이, 첫째가 아닌 아이들을 대표하는 데이터프레임을 반환한다. 임신 기간을 넘파이(NumPy) 배열로 추출하고, 데이터로 DiffMeansPermute에 전달하고, p-값을 계산한다. 결과는 약 0.17로 의미하는 바는 약 17% 정도 관측된 효과에 차이를 예상할 수 있다. 그래서, 효과가 통계적으로 유의하지 않다.

HypothesisTest에는 PlotCdf가 있는데 관측 효과 크기를 나타내는 회색선과 검정 통계량 분포를 플롯으로 그린다.

그림 9.1: CDF of difference in mean pregnancy length under the null hypothesis.

```
ht.PlotCdf()
thinkplot.Show(xlabel='test statistic',
               ylabel='CDF')
```

그림 9.1에 결과가 있다. CDF는 0.83에서 관측 차이와 교차하는데, p-값의 보수 0.17이다.

shows the result. The CDF intersects the observed difference at 0.83, which is the complement of the p-value, 0.17.

만약 출생 체중으로 동일한 분석을 실행한다면, 계산된 p-값은 0 이다; 1000 번 시험 후에, 모의시험은 결코 관측 차이가 0.12 lbs 보다 큰 효과를 산출하지 못한다. 그래서, $p < 0.001$ 가 나오고, 출생 체중에 차이는 통계적으로 유의하다고 결론낸다.

제 4 절 다른 검정 통계량 (ther test statistics)

최선의 검정 통계량을 고르는 것은 무슨 질문을 하느냐에 달려 있다. 예를 들어, 만약 관련된 질문이 첫번째 아이에 대해서 임신 기간이 다른지에 관한 것이라면, 앞절에서 수행했던 것과 같이, 평균에 대해 차이 절대값을 검정하는 것이 의미가 있다.

만약 첫번째 아이가 늦게 출생하는 경향이 있다고 생각할 이유가 있다면, 차이값의 절대값을 취하지 않는다; 대신에 다음 검정 통계량을 사용한다.

```
class DiffMeansOneSided(DiffMeansPermute):

    def TestStatistic(self, data):
        group1, group2 = data
        test_stat = group1.mean() - group2.mean()
        return test_stat
```

`DiffMeansOneSided`은 `DiffMeansPermute`에서 `MakeModel`와 `RunModel`을 상속받는다; 유일한 차이는 `TestStatistic`가 차이에 절대값을 취하지 않는다는 것이다. 이런 유형의 검증을 **단측 (one-sided)** 검증이라고 한다. 왜냐하면, 차이 분포의 단측면만 고려하기 때문이다. 앞선 검정은 양쪽을 사용하기 때문에 **양측 (two-sided)** 검증이라고 한다.

이 버젼 검정에 대해서, p-값은 0.09 다. 일반적으로 단측 검정에 대한 p-값은 양측 검정에 대한 p-값의 약 절반이다. 물론 분포 모양에 달려있다.

단측 가설, 첫째 아이가 늦게 태어난다는 것이 양측 가설보다 좀더 구체적이다. 그래서 p-값이 더 작다. 하지만, 더 강한 가정에 조차도, 차이는 통계적으로 유의적이지 않다.

동일한 프레임워크(framework)를 사용해서 표준 편차에 차이도 검정할 수 있다. 3 절에서 첫번째 아이가 늦게 혹은 빨리 출산할 것같은 증거를 일부 보았다. 그래서, 표준 편차가 좀더 클 것이라는 가설을 세울 수 있다. 다음에 이 가설을 시험하는 방법이 있다.

```
class DiffStdPermute(DiffMeansPermute):

    def TestStatistic(self, data):
        group1, group2 = data
        test_stat = group1.std() - group2.std()
        return test_stat
```

이것은 단측 검정인데 이유는 가설이 첫번째 아이 집단에 대한 표준편차가 단지 다르다는 것이 아니라 더 높다는 것이다. p-값이 0.09로, 통계적으로 유의적이지 않다.

제 5 절 상관 검정 (Testing a correlation)

상기 프레임워크를 사용해서 상관도 검정할 수 있다. 예를 들어, NSFG 데이터 셋에서, 산모 나이와 출생 체중 사이 상관관계는 약 0.07 이다. 나이가 많은 산모 아이가 더 체중이 나가는 것처럼 보인다. 하지만 이 효과가 우연에 의한 것일까요?

검정 통계량으로, 피어슨 상관을 사용했지만, 스피어만 상관도 동일하게 동작한다. 만약 양의 상관 관계로 예측하는 합리적 논거가 있다면, 단측 검정을 할 수도 있다. 하지만, 그런 이유가 없기 때문에, 상관 절대값을 사용해서 양측검정을 수행한다.

귀무 가설은 산모 연령과 출생 체중 사이에 상관 관계가 없다는 것이다. 관측값을 뒤섞어서, 산모 연령과 출생 체중 분포가 같지만 변수 사이에 관련이 없는 세상을 모의 시험할 수 있다.

```
class CorrelationPermute(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):
        xs, ys = data
        test_stat = abs(thinkstats2.Corr(xs, ys))
        return test_stat
```

```
    def RunModel(self):
        xs, ys = self.data
        xs = np.random.permutation(xs)
        return xs, ys
```

data는 한쌍 시퀀스다. TestStatistic가 피어슨 상관 절대값을 계산한다. RunModel이 xs를 뒤섞고 모의시험한 데이터를 반환한다.

다음에 데이터를 읽어 들이고, 시험을 수행하는 코드가 있다.

```
    live, firsts, others = first.MakeFrames()
    live = live.dropna(subset=['agepreg', 'totalwgt_lb'])
    data = live.agepreg.values, live.totalwgt_lb.values
    ht = CorrelationPermute(data)
    pvalue = ht.PValue()
```

subset 인자로 dropna를 사용해서 필요한 변수 둘 중에서 결측된 행을 뺀다.

실제 상관계수는 0.07이다. 계산된 p-값은 0 이다; 1000 번 반복한 뒤에, 가장 큰 모의 시험 상관계수 p-값은 0.04다. 그래서, 관측된 상관계수가 작을지라도, 통계적 유의성이 있다.

"통계적 유의성 (statistically significant)"이 항상 효과가 중요하거나, 실무에서 유의적이라는 의미는 아니라는 것을 상기 예제가 상기 시킨다. 단지 유연으로 발생할 것 같지 않는다는 의미다.

제 6 절 비율 검정 (Testing proportions)

카지노를 운영한다고 가정합시다. 그런데 고객 한명이 비뚤어진 주사위를 사용하고 있다고 용의선상에 올려놓는다; 즉, 다른 쪽보다 한쪽 면이 더 많이 나오도록 변형한 주사위. 주인장이 주장하는 사기꾼을 체포하고 주사위를 압수했다. 하지만, 이제 주사위가 비뚤어졌다는 것을 주인장이 증명해야 한다. 주사위를 60번 던져서 다음과 같은 결과를 얻었다.

Value	1	2	3	4	5	6
Frequency	8	9	19	5	8	11

평균적으로 주사위 각 값은 10번씩 나올 것으로 예상된다. 데이터셋에서, 값 3이 기대한 것보다 더 자주 나오고, 값 4는 덜 나오는 것 처럼 보인다.하지만, 이 차이가 통계적으로 유의적일까요?

가설을 검정하기 위해서, 각 값에 대한 기대 빈도, 기대 빈도와 관측 빈도 차이, 전체 절대값 차이를 계산할 수 있다. 상기 예제에서, 주사위 각 면은 60 회 중에서 10 회 나올 것으로 예상된다; 이 기대값에서 편차가 -2, -1, 9, -5, -2, 1 이 된다; 그래서 전체 절대값 차이는 20 이 된다. 우연히 이러한 차이를 얼마나 자주 목도할까?

다음에 상기 질문에 대답하는 HypothesisTest 버젼이 있다.

```
class DiceTest(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):
        observed = data
        n = sum(observed)
        expected = np.ones(6) * n / 6
        test_stat = sum(abs(observed - expected))
        return test_stat

    def RunModel(self):
        n = sum(self.data)
        values = [1, 2, 3, 4, 5, 6]
        rolls = np.random.choice(values, n, replace=True)
        hist = thinkstats2.Hist(rolls)
        freqs = hist.Freqs(values)
        return freqs
```

데이터는 빈도 리스트로 표현된다: 관측값은 `[8, 9, 19, 5, 8, 11]`이 되고 ; 기대 빈도는 모두 10 이다. 검정 통계량은 절대값 차이의 총합이다.

귀무 가설은 주사위가 공정하다는 것으로 `values` 에서 난수 표본을 뽑아 모의 시험한다. `RunModel` 은 `Hist`를 사용해서 계산하고, 빈도 리스트를 반환한다.

이 데이터에 대한 p-값은 0.13으로 의미하는 바는 만약 주사위가 공정하다면, 전체 관측 편차를 약 13% 가능성으로 볼 것으로 기대된다. 그래서, 명백한 효과는 통계적으로 유의하지 않다.

제 7 절 카이제곱 검정 (Chi-squared tests)

이전 절에서 검정 통계량으로 총편차를 사용했다. 하지만, 비율을 검정하는데, 카이제곱 통계량을 사용하는 것이 좀더 일반적이다.

$$\chi^2 = \sum_i \frac{(O_i - E_i)^2}{E_i}$$

여기서, O_i는 관측빈도, E_i는 기대빈도다. 다음에 파이썬 코드가 있다.

```
class DiceChiTest(DiceTest):

    def TestStatistic(self, data):
        observed = data
        n = sum(observed)
```

```
        expected = np.ones(6) * n / 6
        test_stat = sum((observed - expected)**2 / expected)
        return test_stat
```

(절대값을 취하기 보다)편차를 제곱하면 큰 편차에 더 많은 가중치를 준다. expected로 나누게 되면 편차를 표준화한다. 이 경우에 기대 빈도가 모두 같아서 효과가 없다.

카이제곱 통계량을 사용한 p-값은 0.04로 총편차를 사용해서 얻은 0.13보다도 훨씬 작다. 만약 5% 분계점을 심각하게 생각한다면, 효과가 통계적으로 유의하다고 고려할 수 있다. 하지만, 검정 두가지를 모두 고려해서, 결과는 경계선에 있다고 말할 수 있다. 주사위가 삐뚤어졌다는 가능성을 배제하지는 않지만, 고발된 사기꾼에게 유죄 판결을 하지는 않을 것이다.

상기 예제는 중요한 점을 시연해 준다: p-값이 검정 통계량과 귀무 가설 모형에 달려있다. 그리고 때때로, 이러한 선택에 따라 효과가 통계적으로 유의성을 갖거나 갖지 않거나 결정된다.

제 8 절 다시 첫째 아이

이장 앞에서 첫번째 아이와 첫째가 아닌 아이들에 대한 임신 기간을 살펴봤고, 평균과 표준편차에 명백한 차이는 통계적으로 유의적이지 않다고 결론냈다. 하지만, 3 절에서 평균 기간 분포에서 명백한 차이를 몇가지 봤다. 특히, 35주에서 43주차 범위에서 그렇다. 이러한 차이가 통계적 유의성이 있는지 살펴보기 위해서, 카이제곱 통계량에 기반한 검정을 사용할 수 있다.

다음 코드는 앞선 예제로부터 요소를 결합한다.

```
class PregLengthTest(thinkstats2.HypothesisTest):

    def MakeModel(self):
        firsts, others = self.data
        self.n = len(firsts)
        self.pool = np.hstack((firsts, others))

        pmf = thinkstats2.Pmf(self.pool)
        self.values = range(35, 44)
        self.expected_probs = np.array(pmf.Probs(self.values))

    def RunModel(self):
        np.random.shuffle(self.pool)
        data = self.pool[:self.n], self.pool[self.n:]
        return data
```

데이터는 두개 임신 기간 리스트로 표현된다. 귀무가설은 표본 모두 동일한 분포에서 추출되었다는 것이다. MakeModel은 hstack을 사용해서 두 표본을 합동(pooling)해서 분포를 모형화한다. 그리고 나서, RunModel은 합동 표본을 뒤섞고 이것을 두 부분으로 쪼갬으로써 모의 실험 데이터를 생성한다.

MakeModel은 또한 values를 정의하는데, 사용할 주차 범위가 되고, expected_probs는 합동 분포(pooled distribution)에서 각 값의 확률이다.

다음에 검정 통계량을 계산하는 코드가 있다.

```
# class PregLengthTest:

    def TestStatistic(self, data):
        firsts, others = data
        stat = self.ChiSquared(firsts) + self.ChiSquared(others)
        return stat

    def ChiSquared(self, lengths):
        hist = thinkstats2.Hist(lengths)
        observed = np.array(hist.Freqs(self.values))
        expected = self.expected_probs * len(lengths)
        stat = sum((observed - expected)**2 / expected)
        return stat
```

TestStatistic은 첫째 아이와 첫째 아이가 아닌 아이들에 대한 카이제곱 통계량을 계산하고 더한다.

ChiSquared는 임신 기간 시퀀을 받아, 히스토그램을 계산하고, observed를 계산하는데, 이는 self.values에 상응하는 빈도 리스트다. 기대 빈도 리스트를 계산하기 위해서, 표본 크기에 미리 계산된 확률 expected_probs을 곱한다. 그러면 카이제곱 통계량 stat을 반환한다.

NSFG 데이터에 대해서, 전체 카이제곱 통계량은 102로 그 자체로 많은 것을 의미하지는 않는다. 하지만, 1000회 반복 후에, 귀무가설 아래에서 가장 큰 검정 통계량은 32다. 관측 카이제곱 통계량이 귀무가설 아래에서 가능할 것 같지 않다고 결론내린다. 그래서, 외관 효과는 통계적 유의성이 있다.

이번 예제는 카이제곱 검정 한계를 보여준다: 두 집단 사이에 차이가 있다는 것을 나타내지만, 차이가 무엇인지에 관한 구체적인 어떤 것도 말하지는 못한다.

제 9 절 오류 (Errors)

전통적인 가설 검정에서 만약 p-값이 특정 분계점, 흔히 5% 보다 아래라면, 통계적 유이성이 있다고 본다. 이러한 절차는 두가지 질문을 불러온다.

- 만약 효과가 실제로 우연으로 인한다면, 효과를 잘못해서 유의적으로 고려할 확률은 얼마나 될까? 이 확률이 **거짓 양성률 (false positive rate)**이다.

- 만약 효과가 진실이라면, 가설 검정이 실패할 가능성은 얼마나 될까? 이 확률이 **거짓 음성률 (false negative rate)**이다.

거짓 양성률은 상대적으로 계산하기 쉽다: 만약 분계점이 5% 이면, 거짓 양성률은 5% 다. 다음에 이유가 있다.

- 만약 실제 효과가 없다면, 귀무 가설은 사실이다. 그래서 귀무 가설을 모의 시험함으로써, 검정 통계량 분포를 계산할 수 있다. 이 분포를 CDF_T 라고 부른다.

- 실험을 매번 실행할 때마다, 검정 통계량 t 를 얻는데, CDF_T에서 뽑아낸 것이다. 그리고 나서, p-값을 계산하는데, CDF_T에서 나온 난수 값이 t를 초과하는 확률이다. 그래서 $1 - CDF_T(t)$이 된다.

- 만약 $CDF_T(t)$가 95%보다 크다면, p-값이 5% 보다 작다; 즉, 만약 t가 95번째 백분위수를 초과하면 그렇다. 그리고, CDF_T에서 고른 값이 95번째 백분위수를 얼마나 자주 초과할까? 거의 5% 다.

그래서, 5% 분계점을 가진 가설 검정을 수행한다면, 거짓 양성이 20번 중 1번 예상된다.

제 10 절 검정력 (Power)

거짓 음성률은 계산하기 더 어렵다. 왜냐하면, 실제 효과 크기에 의존하고 정상적으로는 실제 효과 크기를 모르기 때문이다. 한가지 선택옵션은 가상 효과크기 (hypothetical effect size) 조건으로 거짓 음성률을 계산하는 것이다.

예를 들어, 만약 두 집단 사이에 관측 차이가 정확하다면, 관측 표본을 모집단 모형으로 사용해서 모의 시험 데이터로 가설 검정을 실행할 수 있다.

```
def FalseNegRate(data, num_runs=100):
    group1, group2 = data
    count = 0

    for i in range(num_runs):
        sample1 = thinkstats2.Resample(group1)
        sample2 = thinkstats2.Resample(group2)
```

```
        ht = DiffMeansPermute((sample1, sample2))
        pvalue = ht.PValue(iters=101)
        if pvalue > 0.05:
            count += 1

    return count / num_runs
```

FalseNegRate는 각 집단마다 하나씩 두 시퀀스 형태로 데이터를 받는다.

루프를 매번 돌 때마다, 각 집단에서 난수 표본을 뽑아서 가설 검정을 실행함으로써 실험을 모의 시험한다. 그리고 나서 결과를 확인하고 거짓 음성 갯수를 계수한다.

Resample은 시퀀스를 받아 복원 추출로 동일 길이를 가진 표본을 뽑는다.

```
def Resample(xs):
    return np.random.choice(xs, len(xs), replace=True)
```

다음에 임신 기간을 검정하는 코드가 있다.

```
    live, firsts, others = first.MakeFrames()
    data = firsts.prglngth.values, others.prglngth.values
    neg_rate = FalseNegRate(data)
```

결과가 약 70%로 의미하는 바는 만약 평균 임신기간에 실제 차이가 0.078 주라면, 이 표본 크기를 가진 실험은 대략 70% 음성 검정을 산출할 것으로 기대된다.

이 결과는 종종 다르게 제시된다: 만약 실제 차이가 0.078 주라면, 대략 30% 만 양성 검정을 기대해야 한다. 이 "정확한 양성률 (correct positive rate)"이 검정 **검정력(power)**이라고 부르고, 종종 "민감도 (sensitivity)"라고 한다. 주어진 크기 효과를 탐지하는 검정 능력을 반영한다.

이 예제에서, 검정은 단지 30% 가능성만으로 양성 결과를 산출한다(다시, 차이가 0.078 주라는 것을 가정하면). 경험 법칙으로, 80% 검정력이 납득할 수 있는 수준이다. 그래서 이 검정은 "검정력 부족 (underpowered)"하다고 말할 수 있다.

일반적으로 음성 가설 검정이 두 집단 사이에 차이가 없다는 것을 의미하는 것은 아니다; 대신에 만약 차이가 있다면, 너무 작아서 표본크기로 탐지할 수 없다는 것을 제시한다.

제 11 절 반복 (Replication)

이번 장에서 시연한 가설검정과정은 엄밀히 말하면 좋은 관례(good practice)는 아니다.

첫째, 다중 검정(multiple tests)을 수행했다. 만약 가설 검정 하나를 실행한다면, 거짓양성(false positive) 가능성은 약 20번 중에 한번으로 수용가능하다. 하지만, 20번 검정을 수행한다면, 대부분 적어도 한번은 거짓양성을 예상해야한다.

둘째, 탐색과 검정에 동일한 데이터셋을 사용했다. 만약 대용량 데이터셋을 탐색하고, 놀라운 효과를 발견하고 나서, 효과가 유의성이 있는지 검정한다면, 거짓양성(false positive)을 생성할 가능성이 있다.

다중 검정을 보상하기 위해서, p-값 분계점을 조정할 수 있다. (`https://en.wikipedia.org/wiki/Holm-Bonferroni_method` 참조). 혹은, 한 데이터셋은 탐색, 다른 데이터셋은 검정으로 데이터를 분할해서 두 문제를 다룰 수 있다.

몇몇 분야에서는 이러한 관례가 요구되고 있거나 적어도 장려한다. 하지만, 이러한 문제를 암묵적으로 출판된 결과를 반복(replication)하면서 다루는 것도 흔하다. 전형적으로, 새로운 결과를 보고하는 첫번째 논문은 탐색적으로 고려된다. 새로운 데이터로 결과를 반복하는 이어지는 논문은 확증적(confirmatory)으로 고려된다.

공교롭게도, 이 장에서 결과를 반복할 기회가 있다. 이책 첫번째 판은 2002년에 나온 NSFG 사이클 6에 기반했다. 2011년 10월 CDC가 2006–2010에 실시한 인터뷰에 기반한 추가 데이터를 출시했다. `nsfg2.py` 에는 이 데이터를 읽고 정제한 코드가 포함되어 있다. 새로운 데이터셋으로 분석한 결과는 다음과 같다.

- 평균 임신 기간에 차이는 0.16 주다. $p < 0.001$ 으로 통계적 유의성이 있다. (원본 데이터셋 0.078주와 비교된다.)
- 출생 체중에 차이는 0.17 파운드로 $p < 0.001$ 이다. (원본 데이터셋 0.12 lbs와 비교된다.)
- 출생 체중과 산모 연령 사이 상관계수는 0.08 로 $p < 0.001$ 이다. (0.07과 비교된다.).
- 카이제곱검정은 $p < 0.001$ 으로 통계적 유의성이 있다. $p < 0.001$ (원본 데이터도 그렇다.).

요약하면, 원본 데이터에서 통계적 유의성이 있는 모든 효과가 새로운 데이터셋에서도 반복되었다. 그리고 임신기간에 차이는 원데이터에서 유의성이 없었으나 새로운 데이터셋에서는 더 커졌고 유의성이 있다.

제 12 절 연습 문제

A solution to these exercises is in `chap09soln.py`.

Exercise 9.1 As sample size increases, the power of a hypothesis test increases, which means it is more likely to be positive if the effect is real. Conversely, as sample size decreases, the test is less likely to be positive even if the effect is real.

To investigate this behavior, run the tests in this chapter with different subsets of the NSFG data. You can use `thinkstats2.SampleRows` to select a random subset of the rows in a DataFrame.

What happens to the p-values of these tests as sample size decreases? What is the smallest sample size that yields a positive test?

Exercise 9.2 In Section 3, we simulated the null hypothesis by permutation; that is, we treated the observed values as if they represented the entire population, and randomly assigned the members of the population to the two groups.

An alternative is to use the sample to estimate the distribution for the population, then draw a random sample from that distribution. This process is called **resampling**. There are several ways to implement resampling, but one of the simplest is to draw a sample with replacement from the observed values, as in Section 10.

Write a class named `DiffMeansResample` that inherits from `DiffMeansPermute` and overrides `RunModel` to implement resampling, rather than permutation.

Use this model to test the differences in pregnancy length and birth weight. How much does the model affect the results?

제 13 절 용어 사전

- 가설 검정 (hypothesis testing): 외관 효과(apparent effect)가 통계적 유의성이 있는지 결정하는 과정.
- 검정 통계량 (test statistic): 효과 크기를 정량화하는데 사용되는 통계량.
- 귀무 가설 (null hypothesis): 외관 효과가 우연에 의한 것이라는 가정에 기반한 시스템 모델.
- p-값 (p-value): 효과가 우연에 의해서 발생한 확률.
- 통계적 유의성 (statistically significant): 우연에 의해서 발생할 것 같지 않다면, 효과는 통계적 유의성이 있다.

- 순열 검정 (permutation test): 관측 데이터셋에서 순열을 생성해서 p-값을 계산하는 방법.
- 재표본 검정 (resampling test): 관측 데이터셋에서 복원으로 표본을 생성해서 p-값을 계산하는 방법.
- 양측 검정 (two-sided test): "효과 크기가 양으로 혹은 음으로 관측 효과보다 얼마나 큰가?" 를 묻는 검정.
- 단측 검정 (one-sided test): "효과 크기가 동일 부호로 관측 효과보다 얼마나 큰가?" 를 묻는 검정.
- 카이-제곱 검정 (chi-squared test): 검정 통계량으로 카이-제곱 통계량을 사용하는 검정.
- 거짓 양성 (false positive): 사실이 아닐 때, 효과가 사실이라는 결론.
- 거짓 음성 (false negative): 우연이 아닐 때 효과가 우연 때문이라는 결론.
- 검정력 (power): 대립가설이 사실일 때, 이를 사실로서 결정할 확률이다. 검정력이 90%라고 하면, 대립가설이 사실임에도 불구하고 귀무가설을 채택할 확률은 10%이다.[3]

[3] http://ko.wikipedia.org/wiki/êšĂìăŢëăě 참조

제 10 장

선형최소제곱 (Linear least squares)

이번 장에서 사용되는 코드는 `linear.py`에 있다. 코드를 다운로드하고 작업하는 것에 대한 정보는 2을 참조한다.

제 1 절 최소제곱 적합 (Least squares fit)

상관계수는 관계 부호와 강도를 측정하지만 기울기는 측정하지 않는다. 기울기를 측정하는 방법이 몇가지 있다; 가장 흔한 방법이 **선형최소제곱 적합 (linear least squares fit)**이다. "선형 적합(linear fit)"은 변수 사이 관계를 모형화하는 선(line)이다. "최소제곱 (least squares)" 적합은 선과 데이터 사이 평균제곱오차 (mean squared error, MSE)를 최소화하는 것이다.

하나 시퀀스 `ys`가 있는데, 또 다른 시퀀스 `xs` 함수로 표현하고자 한다고 가정하자. 만약 `xs`, `ys`와 절편 `inter`, 기울기 `slope` 사이에 선형 관계가 있다면, 각 `y[i]`가 `inter + slope * x[i]`이 될 것으로 예상된다.

하지만, 상관이 완벽하지 않다면, 예측은 단지 근사(approximation)가 된다. 선에서 수직 편차(vertical deviation), 즉 **잔차(residual)**는 다음과 같다.

```
res = ys - (inter + slope * xs)
```

잔차는 측정 오차 같은 확률 요소(random factor), 혹은 알지 못하는 비임의 요소(non-random factor) 때문일지 모른다. 예를 들어, 만약 체중을 신장 함수로 예측한다면, 미지 요소는 식습관, 운동, 신체 유형을 포함할 수 있다.

만약 모수 `inter`와 `slope`이 잘못되면, 잔차는 더 커진다. 그래서 모수는 잔차를 최소화한다는 것이 직관적으로 의미가 있다.

잔차 절대값, 잔차 제곱, 혹은 잔차 세제곱 최소화를 시도해볼만 하다; 하지만, 가장 흔한 선택은 제곱 잔차 합을 최소화하는 것이다. `sum(res**2))`.

왜 그럴까요? 세가지 좋은 이유와 한가지 덜 중요한 이유가 있다.

- 제곱하게 되면 양수 잔차와 음수 잔차를 동일하게 처리하는 기능이 있는데, 보통 원하는 것이다.
- 제곱은 큰 잔차에 더 많은 가중치를 주지만, 가장 큰 잔차가 항상 주도적인 경우에는 그렇게 많은 가중치를 주지는 않는다.
- 만약 잔차가 상관관계가 없고 평균과 상수 (하지만 알려지지 않은 미지) 분산을 가진 정규분포라면, 최소제곱 적합은 또한 inter와 slope의 최대우도추정량이다. https://en.wikipedia.org/wiki/Linear_regression.
- 제곱 잔차를 최소화하는 inter와 slope 값은 효과적으로 계산될 수 있다.

계산 효율성(computational efficiency)이 당면한 문제에 가장 적합한 방법을 선택하는 것보다 더 중요할 때 마지막 이유가 의미있다. 이제는 더 이상 그럴지는 않다. 그래서, 제곱 잔차가 최소화하는 올바른 것인지만 고려한다.

예를 들어, xs을 사용해서 ys 값을 예측하려고 한다면, 과다 추정하는 것이 과소추정하는 것보다 더 좋을 수도 (더 나쁠 수도) 있다. 이런 경우, 각 잔차에 대한 비용함수를 계산하고 전체 비용, sum(cost(res))을 최소화한다. 하지만, 최소제곱 적합을 계산하는 것이 빠르고, 쉽고, 종종 충분히 만족스럽다.

제 2 절 구현 (Implementation)

thinkstats2에 선형최소제곱을 시연하는 간단한 함수가 있다.

```
def LeastSquares(xs, ys):
    meanx, varx = MeanVar(xs)
    meany = Mean(ys)

    slope = Cov(xs, ys, meanx, meany) / varx
    inter = meany - slope * meanx

    return inter, slope
```

LeastSquares는 시퀀스 xs와 ys을 인자로 받고 추정한 모수 inter와 slope을 반환한다. 동작방법에 관한 자세한 사항은 웹사이트 참조. http://wikipedia.org/wiki/Numerical_methods_for_linear_least_squares.

thinkstats2는 FitLine를 제공하는데, inter 와 slope을 인자로 받아서 시퀀스 xs에 대해서 적합선을 반환한다.

그림 10.1: Scatter plot of birth weight and mother's age with a linear fit.

```
def FitLine(xs, inter, slope):
    fit_xs = np.sort(xs)
    fit_ys = inter + slope * fit_xs
    return fit_xs, fit_ys
```

이 함수를 사용해서 산모 연령 함수로 출생 체중에 대한 최소제곱을 계산할 수 있다.

```
    live, firsts, others = first.MakeFrames()
    live = live.dropna(subset=['agepreg', 'totalwgt_lb'])
    ages = live.agepreg
    weights = live.totalwgt_lb

    inter, slope = thinkstats2.LeastSquares(ages, weights)
    fit_xs, fit_ys = thinkstats2.FitLine(ages, inter, slope)
```

추정한 절편과 기울기는 년마다 6.8 lbs, 0.017 lbs 이다. 이러 형태로 값을 해석하기는 어렵다: 절편은 산모 연령이 0 에서 신생아 기대 체중인데, 문맥상 의미가 없고, 기울기는 너무 작아서 쉽게 이해가 되지 않는다.

$x = 0$에 절편을 제시하는 대신에, 절편을 평균에 제시하는 것이 종종 도움이 된다. 이 경우에, 평균 나이는 약 25세이고, 25세 산모에 대한 평균 아이 체중은 7.3 파운드다. 기울기는 년마다 0.27 온스(ounces) 즉, 10년마다 0.17 파운드가 된다.

그림 10.1에 적합선과 함께 출생 체중과 연령을 산점도로 보여준다. 이와 같이, 관계가 선형인지, 적합선이 관계를 나타내는 좋은 모형인지를 평가하는데, 그림을 그려보는 것은 좋은 생각이 된다.

제 3 절 잔차 (Residuals)

또다른 유용한 검정은 잔차를 플롯하여 그리는 것이다. `thinkstats2`에는 잔차를 계산하는 함수가 있다.

```
def Residuals(xs, ys, inter, slope):
    xs = np.asarray(xs)
    ys = np.asarray(ys)
    res = ys - (inter + slope * xs)
    return res
```

`Residuals`는 시퀀스 `xs`와 `ys`, 추정한 `inter`와 `slope`를 인자로 받는다. 실제값과 적합선 사이 차이를 반환한다.

그림 10.2: Residuals of the linear fit.

잔차를 시각화하기 위해서, 2 절에서 살펴본 것과 같이, 응답자를 연령으로 묶고, 각 집단에 백분위수를 계산한다. 그림 10.2에 각 연령 집단에 대한 25번째, 50번째, 75번째 백분위수가 있다. 중위수는 예상한 것처럼 거의 0이고, 사분위수 범위는 약 2 파운드다. 그래서, 만약 산모 연령을 알고 있다면, 1 파운드 내에서 대략 50% 가능성으로 아이 체중을 추측할 수 있다.

이상적으로 이들 직선이 평평해서 잔차가 랜덤(random)임을 나타내고, 평행해서 잔차 분산이 모든 연령 집단에 대해서 동일하다는 것을 나타내야 한다. 사실, 직선은 평행에 가깝워서 좋다; 하지만, 약간의 곡률(curvature)이 있어서 관계가 비선형임을 나타낸다. 그럼에도 불구하고, 선형 적합은 간단한 모형으로 특정 목적에 아마도 잘 부합한다.

제 4 절 추정 (Estimation)

모수 `slope`와 `inter`는 표본에 기반한 추정값이다; 다른 추정값처럼, 표집 편의, 측정 오차, 표집 오차에 휘둘리기 쉽다. 8 장에서 논의한 것처럼, 표집 편의는 비대표 표집(non-representative sampling)에 의해서, 측정 오차는 데이터 수집과 기록 오류에 의해서, 표집 오차는 전체 모집단보다 표본을 측정하는 결과로 발생한다.

표집 오차를 평가하기 위해서, "만약 이 실험을 다시 실행한다면, 추정값에 얼마나 변동성이 예상될까?" 이러한 질문에 모의시험 실험을 진행하고, 추정값에 대한 표집 분포를 계산함으로써 대답할 수 있다.

데이터를 재표본추출(resampling)하으로써 실험을 모의시험할 수 있다; 즉, 관측된 임신을 마치 전체 모집단인 것처럼 처리해서 관측된 표본에서 복원 추출 방식으로 표본을 추출한다.

```
def SamplingDistributions(live, iters=101):
    t = []
    for _ in range(iters):
        sample = thinkstats2.ResampleRows(live)
        ages = sample.agepreg
        weights = sample.totalwgt_lb
        estimates = thinkstats2.LeastSquares(ages, weights)
        t.append(estimates)

    inters, slopes = zip(*t)
    return inters, slopes
```

SamplingDistributions 함수는 인자로 정상 출산마다 한 줄(row)로 된 데이터프레임과 모의 시험 실험 횟수, iters를 인자로 받는다. ResampleRows을 사용해서 관측 임신을 재표본추출한다. 이미 SampleRows를 살펴봤는데, 데이터프레임에서 무작위(random) 행을 선택한다. thinkstats2는 또한 ResampleRows 기능도 제공하는데, 원본과 동일한 크기 표본을 반환한다.

```
def ResampleRows(df):
    return SampleRows(df, len(df), replace=True)
```

재표본추출 후에, 모의 시험 표본을 사용해서 모수를 추정한다. 결과는 시퀀스 두개다: 추정 절편과, 추정 기울기.

표준 오차와 신뢰구간을 출력해서 표집 분포를 요약한다.

```
def Summarize(estimates, actual=None):
    mean = thinkstats2.Mean(estimates)
    stderr = thinkstats2.Std(estimates, mu=actual)
    cdf = thinkstats2.Cdf(estimates)
    ci = cdf.ConfidenceInterval(90)
    print('mean, SE, CI', mean, stderr, ci)
```

Summarize는 추정값과 실제값 시퀀을 인자로 받는다. 추정값 평균, 표준 오차, 그리고, 90% 신뢰구간을 출력한다.

절편에 대해서, 평균 추정값은 6.83, 표준오차(SE)는 0.07, 90% 신뢰구간(CI) (6.71, 6.94)이다. 좀더 간략한 형식으로, 추정 절편 0.0174, SE 0.0028, CI (0.0126, 0.0220)가 된다. 이 신뢰구간 하한과 상한 사이에 거의 두배 차이난다. 그래서, 개략적인 추정값으로 봐야한다.

추정값의 표집 오차를 시각화하기 위해서, 적합선을 모두 플롯으로 그리고, 연하게 채워서 각 연령에 대해서 90% 신뢰구간을 플롯으로 그린다. 다음에 코드가 있다.

```
def PlotConfidenceIntervals(xs, inters, slopes,
                            percent=90, **options):
    fys_seq = []
    for inter, slope in zip(inters, slopes):
        fxs, fys = thinkstats2.FitLine(xs, inter, slope)
        fys_seq.append(fys)

    p = (100 - percent) / 2
    percents = p, 100 - p
    low, high = thinkstats2.PercentileRows(fys_seq, percents)
    thinkplot.FillBetween(fxs, low, high, **options)
```

xs는 산모 연령 시퀀스다. inters와 slopes는 SamplingDistributions에서 생성된 추정 모수다. percent 인자는 플롯을 얼마의 신뢰구간으로 그릴 것인지 나타낸다.

그림 10.3: 50% and 90% confidence intervals showing variability in the fitted line due to sampling error of `inter` and `slope`.

`PlotConfidenceIntervals`는 `inter`와 `slope` 짝에 대해 적합선을 생성하고, 결과를 시퀀스 `fys_seq`에 저장한다. 그리고 나서, `PercentileRows`을 사용해서 `x` 각 값에 대해서 `y` 상한과 하한 백분위수를 선택한다. 90% 신뢰구간에 대해서, 5번째와 95번째 백분위수를 선택한다. `FillBetween`은 두 직선 사이 공간을 채우는 다각형을 그린다.

그림 10.3에는 산모 연령에 대한 함수로 출생 체중에 적합된 곡선에 대한 50% 와 90% 신뢰구간이 있다. 구역의 수직폭(vertical width)이 표집 오차에 대한 효과를 표현한다; 효과가 평균 주변 값에 대해 더 작고, 극단값에 대해 더 크다.

제 5 절 적합도 (Goodness of fit)

선형 모형 품질, 즉 **적합도 (goodness of fit)**를 측정하는 방법이 몇가지 있다. 가장 간단한 방법은 잔차의 표준편차다.

만약 예측을 위해 선형 모형을 사용한다면, `Std(res)`는 예측의 제곱근 평균제곱오차(root mean squared error, RMSE)다. 예를 들어, 만약 출생 체중을 추측하는데 산모 연령을 사용한다면, 추측 RMSE는 1.40 lbs가 된다.

만약 산모 나이를 모른 상태에서 출생 체중을 추측한다면, 추측 RMSE는 `Std(ys)`로, 1.41 lbs다. 그래서, 이 예제에서 산모 연령을 알고 있는 것은 그다지 예측력을 향상시키지 못한다.

적합도를 측정하는 또 다른 방식은 R^2로 표기하고, "R-제곱(R-squared)"라고 부르는 **결정계수 (coefficient of determination)**다.

```
def CoefDetermination(ys, res):
    return 1 - Var(res) / Var(ys)
```

`Var(res)`는 모형을 사용한 추측 MSE 이고, `Var(ys)`는 모형없는 MSE 다. 그래서, 비율이 만약 모형을 사용한다면 남게되는 MSE 부분이다. R^2 는 모형이 제거하는 MSE 부분이 된다.

출산 체중과 산모 나이에 대해, R^2 값은 0.0047 으로, 산모 나이가 출생 체중에 있는 분산 약 0.5%만 예측한다는 의미가 된다.

결정계수와 피어슨 상관계수 사이에 간단한 관계가 있다: $R^2 = \rho^2$. 예를 들어, 만약 ρ 가 0.8 혹은 -0.8 이라면, $R^2 = 0.64$가 된다.

ρ와 R^2가 종종 관계 강도를 정량화하는데 사용될지라도, 예측력(predictive power)에 대해서 해석하기는 쉽지 않다. 저자 견해로는, `Std(res)`가 예측 품질을 가장 잘 표현한다고 본다. 특히, 만약 `Std(ys)`와 연관되어 표현되면 그렇다.

예를 들어, SAT(미국 대학 입학시험에 사용되는 표준국가시험) 타당성에 대해서 얘기할 때, 종종 사람들은 SAT 점수와 다른 지능(IQ) 측정값 사이에 상관을 얘기한다.

한 연구에 의하면, SAT 점수와 IQ 점수 사이에 피어슨 상관은 $\rho = 0.72$ 으로, 강한 상관처럼 보인다. 하지만, $R^2 = \rho^2 = 0.52$ 이여서 SAT 점수는 IQ 분산의 단지 52%만 설명한다.

IQ 점수는 `Std(ys) = 15` 로 정규화된다. 그래서,

```
>>> var_ys = 15**2
>>> rho = 0.72
>>> r2 = rho**2
>>> var_res = (1 - r2) * var_ys
>>> std_res = math.sqrt(var_res)
10.4096
```

IQ 예측하는데 SAT 점수를 사용하는 것은 RMSE를 15 점에서 10.4 점으로 줄인다. 상관계수 0.72 는 RMSE 를 줄이는데 단지 31% 기여한다.

만약 상관계수가 감명적으로 보인다면, R^2이 MSE 축소에 더 나은 지표가 되고, RMSE 축소가 예측력에 대한 더 나은 지표다.

제 6 절 선형 모형 검정 (Testing a linear model)

출생 체중에 대한 산모 연령 효과가 작고, 거의 예측력이 없다. 그래서, 외관 관계(apparent relationship)가 유연에 의해서 가능한 것인가? 선형적합 결과를 검정하는 방법이 몇개 있다.

한 선택옵션은 MSE에 외관 축소(apparent reduction)가 우연에 의한 것인지 검정하는 것이다. 이 경우에 검정 통계량은 R^2 이고, 귀무 가설은 변수 간 관계가 없다가 된다. 5 절에서 산모 연령과 출생 체중 간 상관을 검정했을 처럼, 귀무가설을 순열(permutation)을 통해서 모의 시험할 수 있다. 사실, $R^2 = \rho^2$, 이기 때문에, R^2 단측 검정은 ρ 양측 검정과 상응한다. 이미 이 검정을 수행했고, $p < 0.001$ 라는 것을 발견했다. 그래서, 산모 연령과 출생 체중 간 외관 관계는 통계적 유의성이 있다고 결론낸다.

또 다른 접근법은 외관 기울기(apparent slope)가 우연에 의한 것인지 검정하는 것이다. 귀무가설은 기울기가 실제로 0 이다는 것이다; 이 경우에, 출생 체중을 평균 근처에 확률변동(random variation)으로 모형화할 수 있다. 다음에 이 모형에 대한 HypothesisTest 가 있다.

```
class SlopeTest(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):
        ages, weights = data
        _, slope = thinkstats2.LeastSquares(ages, weights)
        return slope

    def MakeModel(self):
        _, weights = self.data
        self.ybar = weights.mean()
        self.res = weights - self.ybar

    def RunModel(self):
        ages, _ = self.data
        weights = self.ybar + np.random.permutation(self.res)
        return ages, weights
```

데이터는 연령과 체중 시퀀스로 표현된다. 검정 통계량은 LeastSquares로 추정된 기울기다. 귀무 가설 모형은 모든 아기들의 평균 체중과 평균에 대한 편차로 표현된다. 모의 시험 데이터를 생성하기 위해서, 편차를 순열로 배치하고, 평균에 더한다.

다음에 가설 검정을 수행하는 코드가 있다.

```
live, firsts, others = first.MakeFrames()
live = live.dropna(subset=['agepreg', 'totalwgt_lb'])
ht = SlopeTest((live.agepreg, live.totalwgt_lb))
pvalue = ht.PValue()
```

p-값이 0.001 보다 작다, 그래서 설사 추정된 기울기가 작지만, 우연에 의한 것 같지는 않다.

귀무가설을 모의 시험함으로써 p-값을 추정하는 것이 엄격하게는 맞다. 하지만, 더 간단한 대안이 있다. 이미 4 절에서 기울기 표집 분포를 계산한 것을 기억하라. 이를 위해서, 관측된 기울기가 맞다고 가정하고 재표본추출(resampling)으로 실험을 모의 시험했다.

그림 10.4에는 4절과 귀무가설 아래에서 생성된 기울기 분포에서 나온 기울기 표집 분포가 있다. 표집 분포가 추정된 기울기 약 0.017 lbs/년을 중심으로 있고, 귀무가설 아래 기울기가 약 0 을 중심으로 있다; 하지만, 그것을 제외하고 분포는 동일한다. 분포는 또한 4절에서 살펴보게될 이유로 대칭이다.

그래서, p-값을 두 방식으로 추정할 수 있다:

- 귀무가설 아래서 기울기가 관측 기울기를 초과할 확률을 계산한다.

그림 10.4: The sampling distribution of the estimated slope and the distribution of slopes generated under the null hypothesis. The vertical lines are at 0 and the observed slope, 0.017 lbs/year.

- 표집분포에서 기울기가 0 이하인 확률을 계산한다. (만약 추정 기울기가 음수라면, 표집 분포에서 기울기가 0을 초과하는 확률을 계산할 것이다.)

두번째 선택옵션은 더 쉬운데 이유는 어떻든 정상적으로 모수의 표집 분포를 계산하고자 하기 때문이다. 그리고 표본 크기가 작지 않고, *그리고* 잔차 분포가 기울지 않았다면 좋은 근사(approximation)가 된다. 그때조차도, p-값이 정교할 필요가 없기 때문에, 대체로 만족스럽다.

다음에 표집분포를 사용해서 기울기 p-값을 추정하는 코드가 있다.

```
inters, slopes = SamplingDistributions(live, iters=1001)
slope_cdf = thinkstats2.Cdf(slopes)
pvalue = slope_cdf[0]
```

다시한번, $p < 0.001$이 나온다.

제 7 절 가중 재표본추출 (Weighted resampling)

지금까지 NSFG 데이터를 마치 대표 표본인 것처럼 다루었다. 하지만, 2 절에서 언급한 것 같이, 대표 표본은 아니다. 의도적으로 조사는 몇몇 집단을 오버샘플링(oversampling) 하는데 이유는 통계적으로 유의적인 결과를 도출할 확률을 높이기 위해서다; 즉, 이들 집단에 대해서 검정력을 향상하기 위해서다.

조사설계가 많은 목적에 대해서 유용하지만, 표집 과정을 고려하지 않고, 일반 모집단에 대한 값을 추정하는데 표본을 사용할 수 있다는 것을 의미하지는 않는다.

각 응답자에 대해서, NSFG 데이터는 `finalwgt` 변수가 있는데, 응답자가 대표하는 일반 모집단에 속한 사람 숫자다. 이 값은 **표집 가중치 (sampling weight**, 즉 "가중치 (weight)"라고 불린다.

예제로, 만약 3억 인구를 가진 나라에서 100,000 명을 조사한다면, 각 응답자는 3,000 명을 대표한다. 만약 한 집단을 2배 오버샘플링한다면, 오버샘플링된 집단에 각 사람은 더 적은 가중치(약 1500)가 된다.

오버샘플링을 보정하기 위해서, 재표본추출(resampling)을 사용할 수 있다; 즉, 표집 가중치에 비례하는 확률을 사용해서 조사자료에서 표본을 추출한다. 그리고 나서, 추정하려고 하는 임의 정량정보에 대해서 표집 분포, 표본 오차, 신뢰구간을 생성할 수 있다. 예제로, 평균 출생 체중을 표본 가중치를 두고, 안두고 추정할 것이다.

4절에서, ResampleRows를 살펴봤다. 동일한 확률을 각 행에 두고 데이터프레임에서 행을 추출한다. 이제 동일한 것을 표집 가중치에 비례한 확률을 사용해서 수행할 필요가 있다. ResampleRowsWeighted는 데이터프레임을 인자로 받고, finalwgt에 있는 가중치에 따라 행을 재표본 추출하고, 재표본추출된 행을 포함하는 데이터프레임을 반환한다.

```
def ResampleRowsWeighted(df, column='finalwgt'):
    weights = df[column]
    cdf = Cdf(dict(weights))
    indices = cdf.Sample(len(weights))
    sample = df.loc[indices]
    return sample
```

weights는 시리즈다; 시리즈를 딕셔너리로 전환하는 것은 인덱스를 가중치로 매핑한다. cdf에서 값은 인덱스고, 확률은 가중치에 비례한다.

indices는 행인덱스 시퀀스다; sample은 선택된 행을 담고 있는 데이터프레임이다. 복원 표본 추출을 했기 때문에, 동일한 행이 한번이상 나올 수 있다.

이제, 가중치를 두고, 두지 않고 재표본추출 효과를 비교할 수 있다. 가중치를 두지 않고, 다음과 같이 표집분포를 생성한다.

```
    estimates = [ResampleRows(live).totalwgt_lb.mean()
                 for _ in range(iters)]
```

가중치를 두면, 다음과 같다.

```
    estimates = [ResampleRowsWeighted(live).totalwgt_lb.mean()
                 for _ in range(iters)]
```

다음 표에 요약결과가 나와있다.

	mean birth weight (lbs)	standard error	90% CI
Unweighted	7.27	0.014	(7.24, 7.29)
Weighted	7.35	0.014	(7.32, 7.37)

예제애서, 가중치를 둔 효과는 적지만 무시할만하지는 않다. 가중치를 두고, 두지 않고 추정된 평균에 차이는 약 0.08 파운드, 1.3 온스다. 차이가 추정값 표준 오차(0.014 파운드)보다 실질적으로 더 크다. 함축하는 바는 차이는 우연적 요소에 의한 것은 아니라는 것이다.

제 8 절 연습문제

A solution to this exercise is in chap10soln.ipynb

Exercise 10.1 Using the data from the BRFSS, compute the linear least squares fit for log(weight) versus height. How would you best present the estimated parameters for a model like this where one of the variables is log-transformed? If you were trying to guess someone's weight, how much would it help to know their height?

Like the NSFG, the BRFSS oversamples some groups and provides a sampling weight for each respondent. In the BRFSS data, the variable name for these weights is `totalwt`. Use resampling, with and without weights, to estimate the mean height of respondents in the BRFSS, the standard error of the mean, and a 90% confidence interval. How much does correct weighting affect the estimates?

제 9 절 용어 사전

- 선형적합 (linear fit): 변수 사이 관계를 모형화하려는 직선.
- 최소제곱적합 (least squares fit): 잔차제곱합을 최소화하는 데이터셋 모형.
- 잔차 (residual): 실제값과 모형값 편차.
- 적합도 (goodness of fit): 모형이 데이터에 얼마나 잘 적합하는지에 대한 척도.
- 결정계수 (coefficient of determination): 적합도를 계량화하려는 통계량.
- 표집 가중치 (sampling weight): 표본이 모집단 어느 부분을 대표하는지 나타내는데 표본 관측과 연관된 값.

제 11 장

회귀 (Regression)

앞장에서 선형최소적합은 **회귀 (regression)**의 한 사례다. 회귀는 어떤 종류의 데이터를 어떤 종류의 모형에도 적합하는 더 일반적인 문제다. "회귀 (regression)" 용어 사용은 역사적 우연이다; 단어의 본래 의미와는 간접적으로 연관된다.

회귀분석의 목적은 **종속변수 (dependent variables)**라고 불리는 변수집합과 **설명변수 (explanatory variables)**라고 불리는 또 다른 집합변수 사이 관계를 기술하는 것이다.

앞장에서 종속변수로 출생 체중을 예측하는데 설명변수로 산모 연령을 사용했다. 단지 종속변수가 하나 설명변수도 하나라면, **단순회귀(simple regression)**가 된다. 이번 장에서는 하나 이상 설명변수를 갖는 **다중회귀(multiple regression)**로 옮겨간다. 만약 종속변수가 하나이상이라면, 다변량회귀(multivariate regression)이 된다.

만약 종속변수와 설명변수 간 관계가 선형이면, **선형회귀 (linear regression)**가 된다. 예를 들어, 종속변수가 y이고, 설명변수가 x_1과 x_2라면, 다음과 같이 선형회귀모형을 작성할 수 있다.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$$

여기서 β_0는 절편, β_1은 x_1과 연관된 모수, β_2는 x_2와 연관된 모수, 그리고 ε는 확률변동 혹은 다른 미지 요인으로 인한 잔차다.

y에 대한 시퀀스 값과 x_1과 x_2에 대한 시퀀스 값이 주어지면, ε^2 합을 최소화하는 β_0, β_1, β_2 모수를 찾을 수 있다. 이 과정을 **보통최소제곱(ordinary least squares)**이라고 부른다. 연산은 `thinkstats2.LeastSquare`와 유사하지만, 하나 이상의 설명변수를 다루도록 일반화되었다. 자세한 정보는 웹사이트를 참조한다. `https://en.wikipedia.org/wiki/Ordinary_least_squares`

이번 장에서 사용되는 코드는 regression.py에 있다. 코드를 다운로드하고 작업하는 것에 대한 정보는 2을 참조한다.

제 1 절 StatsModels

앞장에서 가독성이 좋은 단순선형회귀모형을 구현한 thinkstats2.LeastSquares를 제시했다. 다중회귀에 대해서 StatsModels로 전환한다. 파이썬 팩키지로 몇가지 형태 회귀분석와 다른 분석 기능을 제공한다. 만약 아나콘다(Anaconda)를 사용하고 있다면, 이미 StatsModels이 설치되어 있다; 그렇지 않다면, 설치해야할지 모른다.

예제로, StatModels을 가지고 앞장 모형을 실행한다.

```
import statsmodels.formula.api as smf

live, firsts, others = first.MakeFrames()
formula = 'totalwgt_lb ~ agepreg'
model = smf.ols(formula, data=live)
results = model.fit()
```

statsmodels은 인터페이스(APIs) 두개를 제공한다; "formula" API는 문자열로 종속변화 설명변수를 식별한다. patsy라는 구문(syntax)을 사용한다; 상기 예제에서, ~ 연산자가 왼편에 종속변수와 오른편에 설명변수를 구별한다.

smf.ols는 인자로 문자열 공식과 데이터프레임 live를 받고, 모형을 표현하는 OLS객체를 반환한다. ols 는 "ordinary least squares" 약자다.

fit메쏘드는 모형을 데이터에 적합하고 결과를 담고 있는 객체 RegressionResults를 반환한다.

결과는 또한 속성(attribute)으로도 접근가능하다. params은 시리즈로 변수명을 모수에 매핑한다. 그래서, 다음과 같은 절편과 기울기를 얻을 수 있다.

```
inter = results.params['Intercept']
slope = results.params['agepreg']
```

추정한 모수는 6.83 와 0.0175으로 LeastSquares로 추정한 것과 동일한다.

pvalues는 시리즈로 변수명과 연관된 p-값을 매핑한다. 그래서 추정한 기울기가 통계적으로 유의한지 점검할 수 있다.

```
slope_pvalue = results.pvalues['agepreg']
```

agepreg와 연관된 p-값은 5.7e-11 으로 예상한 것처럼 0.001 보다 적다.

results.rsquared는 R^2 정보를 담고 있는데 0.0047이다. results는 또한 전체 모형과 연관된 p-값인 f_pvalue도 제공하는데 R^2가 통계적으로 유의한가를 점검하는 검정과 유사하다.

results는 잔차 시퀀스인 resid와 agepreg에 상응하는 적합한 값 시퀀스 fittedvalues도 제공한다.

results 객체는 summary() 메쏘드도 제공하는데, 가독성이 좋은 형식으로 결과를 표현한다.

```
print(results.summary())
```

하지만 (아직) 관련되지도 않은 많은 정보를 출력한다. 그래서 SummarizeResults로 불리는 더 간단한 함수를 사용한다. 다음에 모형 결과가 있다.

```
Intercept       6.83    (0)
agepreg         0.0175  (5.72e-11)
R^2 0.004738
Std(ys) 1.408
Std(res) 1.405
```

Std(ys)는 만약 어떤 설명변수 도움없이 출생 체중을 추측해야 한다면 갖게될 종속변수 표준편차 RMSE가 된다. Std(res)는 만약 추측에 산모 연령 정보를 안다면 갖게될 잔차 표준편차 RMSE가 된다. 이미 살펴봤듯이, 산모 연령을 아는 것이 그다지 예측에 향상을 가져오지는 않는다.

제 2 절 다중회귀 (Multiple regression)

5절에서 첫째 아이 체중이 첫째가 아닌 아이들 체중보다 가벼운 경향이 있고, 그 효과가 통계적 유의적성이 있다는 것을 봤다. 하지만, 이상한 결과로 이유는 첫번째 아이 체중이 더 가볍다고 할 수 있는 분명한 메커니즘(mechanism)은 없다. 그래서 이러한 관계가 **거짓(spurious)**인지 궁금하다.

사실 이 효과에 대한 가능한 설명이 있기는 하다. 출생 체중이 산모 연령에 의존한 것을 봤다. 그리고 첫째 아이 산모는 첫째가 아닌 아이 산모보다 더 어리다는 것을 예상할 수 있다. 계산 몇번으로 상기 설명이 그럴듯한지 점검할 수 있다. 그리고 나서, 다중회귀를 사용해서 좀더 자세히 조사할 것이다. 먼저 체중 차이가 얼마나 큰지 살펴본다.

```
diff_weight = firsts.totalwgt_lb.mean() - others.totalwgt_lb.mean()
```

첫째 아이는 0.125 lbs, 즉 2 온스 가볍다. 그리고 연령 차이는 다음과 같다.

```
diff_age = firsts.agepreg.mean() - others.agepreg.mean()
```

첫째 아이 산모 나이가 3.59 년 더 젊다. 다시 선형 모형을 돌리게 되면, 연령 함수로 출생체중에 변화값를 얻는다.

```
results = smf.ols('totalwgt_lb ~ agepreg', data=live).fit()
slope = results.params['agepreg']
```

기울기는 년당 0.175 파운드다. 만약 기울기를 연령 차이와 곱하게 되면, 산모 연령 때문에 첫째 아이와 첫째가 아닌 아이들에 대한 출생 체중 평균 차이를 얻게된다.

```
slope * diff_age
```

결과는 0.063 으로 관측 차이의 약 절반이다. 그래서 잠정적으로 출생 체중에 관측 차이는 부분적으로 산모 연령 차이로 설명될 수 있다고 결론낸다.

다중 회귀를 사용해서, 관계를 좀더 체계적으로 탐색할 수 있다.

```
    live['isfirst'] = live.birthord == 1
    formula = 'totalwgt_lb ~ isfirst'
    results = smf.ols(formula, data=live).fit()
```

첫번째 행은 isfirst라는 새로운 칼럼(열)을 생성한다. 첫번째 아이는 참(True), 첫번째 아이가 아니면 거싯(False)다. 그리고 나서, 설명변수로 isfirst를 사용해서 모형에 적합한다.

다음에 결과가 있다.

```
Intercept          7.33    (0)
isfirst[T.True]  -0.125   (2.55e-05)
R^2 0.00196
```

isfirst는 부울(boolean)이기 때문에, ols는 이 변수를 **범주형 변수 (categorical variable)**로 처리한다. 의미하는 바는 값이 참(true)과 거짓(false) 같은 범주에 속하기 때문에 숫자로 처리되면 안된다는 것이다. 추정된 모수는 isfirst가 참일때 출생 체중에 대한 효과다. 그래서 결과 -0.125 lbs 는 첫번째 아이와 첫째가 아닌 아이들 간에 출생체중 차이다.

기울기와 절편이 통계적 유의성이 있다. 의미하는 바는 우연으로 발생한 것 같지는 않다. 하지만, 모형 R^2 값이 작다. 의미하는 바는 isfirst 가 출생체중 변동 상당부분을 설명하지는 못한다는 것이다.

결과는 agepreg와 비슷하다.

```
Intercept        6.83     (0)
agepreg          0.0175   (5.72e-11)
R^2 0.004738
```

다시 한번, 모수는 통계적 유의성이 있으나, R^2 값은 낮다.

상기 모형은 지금까지 살펴본 결과를 다시 확인해 준다. 하지만, 이제 변수 두개를 넣어서 모형을 적합할 수 있다. 구문 공식 totalwgt_lb ~ isfirst + agepreg 을 대입하면, 다음을 얻는다:

```
Intercept         6.91     (0)
isfirst[T.True] -0.0698   (0.0253)
agepreg           0.0154   (3.93e-08)
R^2 0.005289
```

조합 모형에서, isfirst에 대한 모수는 약 절반 작아졌다. 의미하는 것은 isfirst의 외관효과가 사실 agepreg으로 설명이 된다는 것이다. 그리고 isfirst에 대한 p-값은 약 2.5%로, 통계적 유의성 경계선상에 있다.

모형에 R^2값이 약간 더 크다. 따라서 변수 두개가 혼자보다 출산 체중에 변동성을 더 많이 (하지만 그다지 많지는 않다) 설명하는 것을 나타낸다.

제 3 절 비선형 관계 (Nonlinear relationships)

agepreg 변수 기여분이 비선형일지 모른다는 것을 기억하면, 이 관계에 더 많은 것을 잡아내기 위해서 변수 추가를 생각해볼 수 있다. 한가지 선택옵션은 연령 제곱 정보을 담고 있는 agepreg2 칼럼(열)을 생성하는 것이다.

```
live['agepreg2'] = live.agepreg**2
formula = 'totalwgt_lb ~ isfirst + agepreg + agepreg2'
```

이제 agepreg와 agepreg2에 대한 모수를 추정함으로써, 효과적으로 포물선에 적합한다.

```
Intercept         5.69     (1.38e-86)
isfirst[T.True] -0.0504    (0.109)
agepreg           0.112    (3.23e-07)
agepreg2        -0.00185   (8.8e-06)
R^2 0.007462
```

agepreg2 모수가 음수라서, 포물선 곡선이 아래쪽 방향으로, 그림 10.2에 나와있는 선 모양과 일관성을 갖는다.

agepreg 이차 모형이 출생 체중에 더 많은 변동성을 설명한다; isfirst에 대한 모수가 이 모형에서 더 작아졌고, 더이상 통계적 유의성은 없다.

agepreg2 처럼 계산된 변수를 사용하는 것이 데이터에 다항식과 다른 함수를 적합하는 흔한 방식이다. 이 과정은 여전히 선형회귀로 간주되는데 이유는 변수가 다른 변수의 비선형 함수가 되는지에 관계없이, 종속 변수가 설명변수의 선형 함수이가 되기 때문이다.

다음 표에 요약된 회귀결과가 있다.

	isfirst	agepreg	agepreg2	R^2
Model 1	-0.125 *	–	–	0.002
Model 2	–	0.0175 *	–	0.0047
Model 3	-0.0698 (0.025)	0.0154 *	–	0.0053
Model 4	-0.0504 (0.11)	0.112 *	-0.00185 *	0.0075

상기 표에 칼럼(열)은 설명변수 이름과 결정계수 R^2다. 각 항목은 추정한 모수와 p-값이 괄호에 들어있고, 0.001보다 작은 p-값을 표기하기 위한 별표가 있다.

출생 체중에 외관 차이는 적어도 부분적으로 산모 연령의 차이로 설명된다고 결론낸다. 모형에 산모 연령을 포함할 때, `isfirst` 효과는 더 작아지고, 잔존효과는 유연일지도 모른다.

상기 예제에서, 산모 연령이 **제어변수 (control variable)**로 역할을 한다; 모형에 `agepreg` 변수를 포함하게 되면 첫 출산 산모와 그렇지 않은 산모 간 연령 차이를 "제어한다(controls for)." 그래서, (혹시 있다면) `isfirst` 효과를 격리할 수 있게 한다.

제 4 절 Data mining

지금까지 설명을 위해서 회귀모형을 사용했다; 예를 들어, 앞절에서 출생체중 외관 효과는 사실 산모 연령 차이 때문이라는 것을 밝혀냈다. 하지만, 이 모형의 R^2 값이 매우 낮아서, 의미하는 것은 거의 예측력이 없다는 것이다. 이번 절에서 더 잘해보려고 한다.

동료중 한명이 곧 출산한다고 하고, 태어날 아이 출생 체중을 맞추는 공식 복권 판매소(betting pool)가 있다고 가정하자. (만약 여럿이 어울려 돈을 걸는데 친숙하지 않다면, 다음을 참조 한다 `https://en.wikipedia.org/wiki/Betting_pool`).

이제 정말 내기에서 *정말* 이기고 싶다고 가정하자. 우승 가능성을 높이기 위해서 무엇을 할 수 있을까? NSFG 데이터셋은 각 임신에 관해서 244개 변수, 각 응답자에 대해서 추가로 308개 변수를 포함하고 있다. 아마도, 변수 중 일부는 예측력이 있다. 어느 변수가 가장 유용한지 알아내기 위해서, 모든 변수를 시도해보면 안될까?

임신 테이블(pregnancy table)에 있는 변수를 검정하는 것은 쉽다. 하지만, 응답자 테이블(respondent table)에 있는 변수를 사용하기 위해서는, 각 임신과 응답자를 매칭해야 한다. 이론적으로 임신 테이블 행을 반복 돌리고, `caseid`를 사용해서 해당하는 응답자를 찾아내고, 해당 테이블에 있는 값을 임신 테이블로 복사한다. 하지만, 시간이 많이 걸리고 느릴것 같다.

더 좋은 선택옵션은 SQL과 다른 관계형 데이터베이스 언어에 정의된 **결합 (join)** 연산으로 이 과정을 인식하는 것이다. (`https://en.wikipedia.org/wiki/Join_(SQL)`ìřÿìąř). 결합(Join)이 데이터프레임 메쏘드로 구현되어 있어서, 다음과 같이 연산을 수행한다.

```
live = live[live.prglngth>30]
resp = chap01soln.ReadFemResp()
resp.index = resp.caseid
join = live.join(resp, on='caseid', rsuffix='_r')
```

복권판매소가 마감일 몇주 전에 문을 연다고 가정하고, 첫번째 행은 임신 기간이 30 주차 이상된 레코드만 선택한다.

다음 행은 응답자 파일을 읽는다. 결과는 정수 인덱스를 갖는 데이터프레임이다; 응답자를 효율적으로 조회하기 위해서, `resp.index` 를 `resp.caseid`으로 교체했다.

"왼편(left)" 테이블인 `live`가 `join` 메쏘드를 호출하고, "오른편(right)" 테이블 `resp`에 전달된다. 키워드 인자 `on`이 두 테이블에서 행을 매칭하는데 사용되는 변수를 지칭한다.

이 예제에서, 칼럼(열) 이름 몇개가 양쪽 테이블에 나타난다. 그래서, `rsuffix`를 통해서, 오른편 테이블에 중복되는 칼럼 명칭에 추가한다. 예를 들어, 양쪽 테이블 모두 응답자 인종을 기호화한 `race`라는 칼럼이 있다. 결합(Join) 결과에는 `race`와 `race_r` 이라는 두 칼럼이 있게 된다.

판다스 구현은 빠르다. NSFG 테이블을 결합(Join)하는 것은 보통의 데스크탑 컴퓨터에서 1초도 걸리지 않는다. 이제 변수 검정을 시작할 수 있다.

```
t = []
for name in join.columns:
    try:
        if join[name].var() < 1e-7:
            continue

        formula = 'totalwgt_lb ~ agepreg + ' + name
        model = smf.ols(formula, data=join)
        if model.nobs < len(join)/2:
            continue

        results = model.fit()
    except (ValueError, TypeError):
        continue

    t.append((results.rsquared, name))
```

모형을 구축한 각 변수에 대해서, R^2를 계산하고, 리스트에 결과값을 덧붙인다. 모든 모형은 `agepreg` 변수를 포함하는데, 이미 일정부분 예측력이 있다는 것을 확인했기 때문이다.

각 설명변수에 변동성이 있는지 점검한다; 만약 그렇지 않다면, 회귀 결과는 신뢰성이 없다. 또한 각 모형에 대해서 관측점 숫자도 점검한다. 너무 많은 `nan` 을 포함한 변수는 예측에 좋은 후보는 아니다.

대부분의 변수에 대해서 어떠한 정제(cleaning)도 수행하지 않았다. 변수 중 일부는 선형회귀에 잘 동작하지 않는 방식으로 부호화되어 있다. 결과로 만약 적

절하게 변수를 정제하지 않는다면, 유용할지도 모른 변수 몇개를 간과하게 된다.

제 5 절 예측 (Prediction)

다음 단계는 결과를 정렬하고 가장 높은 R^2 값을 산출하는 변수를 선택하는 것이다.

```
t.sort(reverse=True)
for mse, name in t[:30]:
    print(name, mse)
```

리스트에 첫 변수는 totalwgt_lb고, birthwgt_lb가 다음이다. 분명하게, 출생 체중을 사용해서, 출생 체중을 예측할 수는 없다.

비슷하게 prglngth 변수가 유용한 예측력이 있지만, 복권판매소에 대해 임신기간(그리고 연관된 변수)은 아직 알려지지 않았다고 가정한다.

첫번째 유용한 예측변수는 babysex으로 아기 성별이 남자인지 여자인지 나타낸다. NSFG 데이터셋에서 남자가 약 0.3 lbs 더 무겁다. 그래서, 아이 성별을 알고 있다고 가정하면, 예측에 설명변수로 사용할 수 있다.

다음은 인종 (race)이다. 응답자가 백인, 흑인, 혹은 다른 인종인지를 나타낸다. 설명 변수로서, 인종은 문제소지가 있다. NSFG 같은 데이터셋에서 인종은 많은 다른 변수와 상관이 있는데, 소득 그리고 다른 사회경제적 요인이 포함된다. 회귀모형에서, 인종은 **대리변수 (proxy variable)**로 동작한다. 그래서 인종과 외관상관이 적어도 부분적으로 다른 요인에 의해서 종종 발생된다.

리스트에 다음 변수는 nbrnaliv다. 임신이 다둥인지를 나타낸다. 쌍둥이와 세쌍둥는 다른 아이보다 더 작은 경향이 있다. 그래서 만약 가상 동료가 쌍둥이를 기대하고 있다는 것을 알고 있다면, 도움이 될 수 있다.

리스트에 있는 다음 변수는 paydu다. 응답자가 집을 가지고 있는지를 나타낸다. 소득과 관련된 변수중의 하나로 예측력이 있는 것으로 밝혀졌다. NSFG같은 데이터셋에서, 소득과 재산은 거의 모든 것과 상관관계가 있다. 상기 예제에서, 소득은 식단, 건강, 건강보험 그리고 출생 체중에 영향을 줄 것 같은 다른 요인과 관련있다.

리스트에 있는 다른 변수중 일부는 아이가 모유 수유한 주차(week) 정보 bfeedwks로 나중까지도 알수 없는 변수다. 이러한 변수를 예측에 사용할 수 없지만, bfeedwks 변수가 출생 체중과 상관될 수 있는 이유를 추측할 필요가 있다.

때때로, 이론에서 시작해서 이론을 검정하는데 데이터를 사용한다. 다르게는 데이터로 시작해서 가능한 이론을 살펴보는 방향으로 추진한다. 이번절에서 시연한

두번째 접근법을 **데이터 마이닝 (data mining)**이라고 부른다. 데이터 마이닝의 장점은 기대하지 않은 패턴을 발견할 수 있다는 것이다. 위험성은 발견한 많은 패턴이 확률 변동(random)이거나 거짓(spurious)이라는 것이다.

잠재성 있는 설명변수를 식별했기 때문에 모형 몇개를 검정하고 이중 하나를 정한다.

```
formula = ('totalwgt_lb ~ agepreg + C(race) + babysex==1 + '
           'nbrnaliv>1 + paydu==1 + totincr')
results = smf.ols(formula, data=join).fit()
```

상기 공식(formula)에는 지금까지 보지 못한 구문이 있다: `C(race)`은 Patsy 공식 파서 (formula parser)에 인종(race)변수를 숫자형식으로 부호화 되어 있지만, 범주형 변수로 처리하게 한다.

`babysex` 변수에 대한 부호화는 남자는 1, 여자는 2로 되어 있다; `babysex==1` 이것은 숫자를 부울(boolean) 전환해서 남자는 참(true), 여자는 거짓(false)가 된다.

비슷하게, `nbrnaliv>1`은 다둥이 출산은 참(true)가 되고, `paydu==1`는 집을 소유한 응답자가 참(true)가 된다.

`totincr` 변수는 숫자 1–14 로 부호화되어 있어, 연간 소득으로 숫자가 하나 증가할때 마다 $5000을 나타낸다. 그래서 $5000 단위로 표현된 숫자 값을 처리할 수 있다.

다음에 모형 적합 결과가 있다.

```
Intercept                 6.63    (0)
C(race)[T.2]              0.357   (5.43e-29)
C(race)[T.3]              0.266   (2.33e-07)
babysex == 1[T.True]      0.295   (5.39e-29)
nbrnaliv > 1[T.True]     -1.38    (5.1e-37)
paydu == 1[T.True]        0.12    (0.000114)
agepreg                   0.00741 (0.0035)
totincr                   0.0122  (0.00188)
```

특히 소득 요인을 제어했는데, 인종에 대한 추정 모수는 저자가 예상한 것보다 더 크다. 흑인은 1, 백인은 2, 나머지 인종은 3으로 부호화했다. 흑인 산모 아이가 0.27–0.36 lbs 만큼 다른 인종 아이보다 더 가볍다.

이미 살펴본 것처럼, 남자 아이가 약 0.3 lbs 더 무겁다; 쌍둥이와 다둥이는 약 1.4 lbs 정도 더 가볍다.

소득을 제어했을 때, 자기 집을 소유한 사람은 약 0.12 lbs 정도 더 아이가 무겁다. 산모 연령 모수는 2 절에서 살펴본 것보다 더 작다. `paydu`와 `totincr`을 포함하여 다른 변수 일부와 연령이 상관됨을 시사한다.

이들 변수 모두가 통계적으로 유의하고 몇몇은 매우 낮은 p-값을 보인다. 하지만, R^2 값은 겨우 0.06 으로 여전히 매우 작다. 모형을 사용하지 않은 RMSE는 1.27 lbs; 모형을 사용하면 1.23으로 떨어진다. 그래서 내기에서 우승할 가능성은 상당하게 증가하지는 않는다. 미안합니다!

제 6 절 로지스틱 회귀 (Logistic regression)

이전 예제에서, 설명 변수 몇개는 숫자형이고, (부울을 포함하여) 몇개는 범주형이다. 하지만, 종속변수가 항상 숫자형이다.

선형회귀는 다른 유형 종속변수를 처리하도록 일반화될 수 있다. 만약 종속변수가 부울(boolean)이라면, 일반화모형은 **로지스틱 회귀(logistic regression)**라고 불린다. 만약 종속변수가 정수 갯수라면, **포아송 회귀 (Poisson regression)**라고 불린다..

로지스틱 회귀 사례로, 복권판매소 시나리오 변형을 생각해보자. 여러분 친구중 한명이 임신했고, 아기가 남자인지 여자인지 예측한다고 가정하자. NSFG 데이터를 사용해서 관례로 남자 아이가 태어나는 확률로 정의되는 "성비(sex ratio)"에 영향을 주는 요인을 찾아낸다.

예를 들어 여자를 0, 남자를 1로 종속변수를 숫자형으로 부호화한다면, 보통최소제곱(ordinary least square) 방법을 적용할 수 있지만, 문제가 있다. 선형 모형은 다음과 같은 형태다.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$$

여기서 y 는 종속변수, x_1과 x_2는 설명변수다. 그리고 나서 잔차를 최소화하는 모수를 찾는다.

상기 접근법 문제는 해석하기 어려운 예측을 산출한다는 것이다. 추정 모수와 x_1와 x_2에 대한 값이 주어지면, 모형은 $y = 0.5$ 으로 예측할 수 있다. 하지만 y 의 유의미한 값은 0과 1이다.

확률로 결과를 해석하고자 하는 유혹이 있다; 예를 들어, x_1와 x_2의 특정 값을 가진 응답자가 남자 아이를 가질 확률이 50%라고 말하고 싶다. 하지만 모형이 $y = 1.1$ 혹 $y = -0.1$으로 예측하는 것도 가능하다. 하지만 타당한 확률값은 아니다.

로지스틱 회귀는 확률보다는 **오즈(odds)** 용어로 예측을 표현함으로써 이러한 문제를 피해간다. 만약 오즈에 친숙하지 않다면, 사건에 대한 "선호 오즈 (odds in favor)"는 일어나지 않을 확률에 대한 일어날 확률 비율이다.

그래서 만약 우리팀 승리 가능성이 75% 라면, 선호 오즈가 3대 1 인데, 이유는 승리 가능성이 패배 가능성보다 3배가 되기 때문이다.

오즈와 확률은 동일한 정보를 다르게 표현한다. 확률이 주어지면, 다음과 같이 오즈를 계산한다.

```
o = p / (1-p)
```

선호 오즈(odds in favor)가 주어지면, 다음과 같이 확률로 전환한다.

```
p = o / (o+1)
```

로지스틱 회귀는 다음 모형에 기반한다.

$$\log o = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$$

여기서, o 는 특정 결과에 대한 선호 오즈다; 예제에서 o 는 남자 아이를 갖는 오즈가 된다.

모수 β_0, β_1, β_2 를 추정한다고 가정하자. (잠시 후에 추정방법을 설명한다) 그리고, x_1와 x_2에 값이 주어졌다. $\log o$ 예측 값을 계산하고 나서 확률로 전환한다.

```
o = np.exp(log_o)
p = o / (o+1)
```

그래서 복권판매소 시나리오에서 남자아이를 갖는 예측 확률을 계산할 수 있다. 하지만, 모수를 어떻게 추정할가요?

제 7 절 모수 추정 (Estimating parameters)

선형회귀와 달리, 로지스틱 회귀는 닫힌 형식 해답(closed form solution)이 없다. 그래서, 초기 해답(solution)을 추측하고 반복적으로 해답에 접근해 간다.

통상적인 목표는 최대우도추정량(maximum-likelihood estimate, MLE)을 찾는 것으로, 데이터 우도(likelihood)를 최대화하는 모수 집합이다. 예를 들어, 다음 데이터가 있다고 가정하자.

```
>>> y = np.array([0, 1, 0, 1])
>>> x1 = np.array([0, 0, 0, 1])
>>> x2 = np.array([0, 1, 1, 1])
```

최초 추측값 $\beta_0 = -1.5$, $\beta_1 = 2.8$, $\beta_2 = 1.1$ 에서 출발한다.

```
>>> beta = [-1.5, 2.8, 1.1]
```

그리고 나서, 각 행에 대해서 `log_o`을 계산한다:

```
>>> log_o = beta[0] + beta[1] * x1 + beta[2] * x2
[-1.5 -0.4 -0.4  2.4]
```

그리고 로그 오즈를 확률로 전환한다.

```
>>> o = np.exp(log_o)
[  0.223   0.670   0.670  11.02  ]

>>> p = o / (o+1)
[ 0.182  0.401  0.401  0.916 ]
```

log_o가 0 보다 클 때, o는 1 보다 크고 p는 0.5 보다 크다는 것을 주목한다.

결과 우도는 y==1일 때 p, y==0일 때 1-p다. 예를 들어, 남아가 태어날 확률이 0.8이고, 결과가 남아라고 생각한다면, 우도는 0.8이다; 만약 결과가 여아라면, 우도는 0.2다. 다음과 같이 계산할 수 있다.

```
>>> likes = y * p + (1-y) * (1-p)
[ 0.817  0.401  0.598  0.916 ]
```

데이터 전체 우도는 likes 곱이다.:

```
>>> like = np.prod(likes)
0.18
```

beta 값에 대해, 데이터 우도는 0.18이다. 로지스틱 회귀 목표는 우도를 최대화하는 모수를 찾아내는 것이다. 이를 위해서, 대부분의 통계 팩키지는 뉴톤 방법(Newton's Method) 같은 반복 해결사(iterative solver)를 사용한다.(https://en.wikipedia.org/wiki/Logistic_regression#Model_fitting 참조).

제 8 절 구현 (Implementation)

StatsModels에는 로지스틱 회귀 기능을 제공하는 확률을 로그 오즈(log odds)로 전환하는 함수 이름을 따서 logit이라고 부른다. 사용법을 시연하기 위해서, 성비(sex ratio)에 영향을 주는 변수를 찾는다.

다시, NSFG 데이터를 적재하고 임신 30 주차 이상된 정보만 선택한다.

```
live, firsts, others = first.MakeFrames()
df = live[live.prglngth>30]
```

logit에 종속변수는 (부울 자료형 보다) 이진(binary)이 되어야 한다. 그래서 이진 정수(binary integer)로 전환하는 astype(int) 메쏘드를 사용해서 새로운 칼럼(열) boy를 생성한다.

```
df['boy'] = (df.babysex==1).astype(int)
```

성비에 영향을 주는 것을 밝혀진 요인은 부모 연령, 출생 순서, 인종, 사회적 지위가 포함된다. 로지스틱 회귀를 사용해서 이러한 효과가 NSFG 데이터에 나타나는지 살펴보자. 산모 연령부터 시작하자.

```
import statsmodels.formula.api as smf

model = smf.logit('boy ~ agepreg', data=df)
results = model.fit()
SummarizeResults(results)
```

logit은 ols와 동일한 인자, Patsy 구문 공식(formula)과 데이터프레임을 받는다. 결과는 모형을 표현하는 Logit 객체다. 객체 내부에는 endog와 exog 속성이 포함된다; 종속변수를 부르는 다른 이름 **내생 변수 (endogenous variable)**와 설명변수를 부르는 다른 이름 **외생 변수 (exogenous variables)**가 있다. 넘파이(NumPy) 배열이기 때문에, 때때로 배열을 데이터프레임으로 변환하는 것이 편리하다.

```
endog = pandas.DataFrame(model.endog, columns=[model.endog_names])
exog = pandas.DataFrame(model.exog, columns=model.exog_names)
```

model.fit 결과는 BinaryResults 객체로, ols 적합에서 얻은 RegressionResults 객체와 유사하다. 다음에 요약결과가 있다.

```
Intercept   0.00579   (0.953)
agepreg     0.00105   (0.783)
R^2 6.144e-06
```

agepreg 모수는 양수다. 나이가 더 많은 산모가 남아를 더 선호하는 것같지만, p-값은 0.783으로 외관효과는 우연에 의한 것으로 볼 수 있다.

결정계수, R^2는 로지스틱 회귀에 적용되지 않는다. 하지만, "의사 R^2 값 (pseudo R^2 values)"으로 사용될 수 있는 대안이 몇개 있는데 모형을 비교하는데 유용하다. 예를 들어, 성비와 연관되어 있다고 믿어지는 요인 몇개를 포함한 모형이 다음에 있다.

```
formula = 'boy ~ agepreg + hpagelb + birthord + C(race)'
model = smf.logit(formula, data=df)
results = model.fit()
```

산모 연령과 함께, 모형에는 출생시점에 아버지 연령 (hpagelb), 출생 순서 (birthord), 범주형 변수로 인종이 포함된다. 다음에 결과가 있다.

```
Intercept      -0.0301     (0.772)
C(race)[T.2]   -0.0224     (0.66)
C(race)[T.3]   -0.000457   (0.996)
agepreg        -0.00267    (0.629)
hpagelb         0.0047     (0.266)
birthord        0.00501    (0.821)
R^2 0.000144
```

추정 모수 어떤 것도 통계적 유의성이 없다. 의사-R^2 값은 약간 더 높다. 하지만 우연적인 요인 때문일 수 있다.

제 9 절 정밀도 (Accuracy)

복권판매소 시나리오에서, 모형 정밀도(accuracy of the model)에 더 관심이 있다: 우연한 기대와 비교하여 성공적으로 예측한 횟수.

NSFG 데이터에는 여아보다 남아가 더 많다. 그래서 기본 전략은 매번 "남아"로 추측한다. 이 전략 정확도는 단지 남아 비율이 된다.

```
actual = endog['boy']
baseline = actual.mean()
```

`actual`이 이진 정수로 부호화되어 있어서, 평균은 남아 비율, 0.507이 된다.

다음에 모형 정확도를 계산한 방법이 나와있다.

```
predict = (results.predict() >= 0.5)
true_pos = predict * actual
true_neg = (1 - predict) * (1 - actual)
```

`results.predict`는 확률 넘파이(NumPy) 배열을 반환하는데, 0 혹은 1로 반올림한다. `actual`을 곱해서 만약 남아를 예측하고 맞다면 1, 그렇지 않다면 0 을 산출한다. 그래서 `true_pos` 는 "참양성(true positives)"을 나타낸다.

마찬가지로 `true_neg`는 "여아"라고 추측하고 맞춘 경우를 나타낸다. 정밀도는 추측이 적중한 비율이다.

```
acc = (sum(true_pos) + sum(true_neg)) / len(actual)
```

결과값이 0.512 로 기본 전략 0.507 보다 다소 높게 나온다. 하지만, 이 결과를 너무 심각하게 받아들이면 안된다. 동일한 데이터를 사용해서 모형을 구축했고 검정했다. 그래서 모형이 새로운 데이터에 예측력을 갖추지 못할 수 있다.

그럼에도 불구하고, 모형을 사용해서 복권판매소에 대한 예측을 해보자. 친구 나이가 35세이고, 백인, 친구 남편 나이는 39, 그리고 세번째 아이를 임신하고 있다고 가정하자.

```
columns = ['agepreg', 'hpagelb', 'birthord', 'race']
new = pandas.DataFrame([[35, 39, 3, 2]], columns=columns)
y = results.predict(new)
```

신규 사례에 대해서 `results.predict`을 호출하기 위해서, 모형에 각 변수에 대한 칼럼을 갖는 데이터프레임을 작성해야 한다. 이 경우에 0.52가 나와서 "남아"로 추정해야 한다. 하지만, 만약 모형이 승리 가능성을 향상시키지만, 차이는 매우 작다.

제 10 절 연습 문제

My solution to these exercises is in `chap11soln.ipynb`.

Exercise 11.1 Suppose one of your co-workers is expecting a baby and you are participating in an office pool to predict the date of birth. Assuming that bets are placed during the 30th week of pregnancy, what variables could you use to make the best prediction? You should limit yourself to variables that are known before the birth, and likely to be available to the people in the pool.

Exercise 11.2 The Trivers-Willard hypothesis suggests that for many mammals the sex ratio depends on "maternal condition"; that is, factors like the mother's age, size, health, and social status. See `https://en.wikipedia.org/wiki/Trivers-Willard_hypothesis`

Some studies have shown this effect among humans, but results are mixed. In this chapter we tested some variables related to these factors, but didn't find any with a statistically significant effect on sex ratio.

As an exercise, use a data mining approach to test the other variables in the pregnancy and respondent files. Can you find any factors with a substantial effect?

Exercise 11.3 If the quantity you want to predict is a count, you can use Poisson regression, which is implemented in StatsModels with a function called `poisson`. It works the same way as `ols` and `logit`. As an exercise, let's use it to predict how many children a woman has born; in the NSFG dataset, this variable is called `numbabes`.

Suppose you meet a woman who is 35 years old, black, and a college graduate whose annual household income exceeds $75,000. How many children would you predict she has born?

Exercise 11.4 If the quantity you want to predict is categorical, you can use multinomial logistic regression, which is implemented in StatsModels with a function called `mnlogit`. As an exercise, let's use it to guess whether a woman is married, cohabitating, widowed, divorced, separated, or never married; in the NSFG dataset, marital status is encoded in a variable called `rmarital`.

Suppose you meet a woman who is 25 years old, white, and a high school graduate whose annual household income is about $45,000. What is the probability that she is married, cohabitating, etc?

제 11 절 용어 사전

- 회귀 (regression): 데이터에 모형을 적합하는 모형을 추정하기 위한 몇가지 관련된 과정 중의 하나.

- 종속변수 (dependent variables): 회귀모형에서 예측하려는 변수. 또한, 내생변수로도 알려져있다.
- 설명변수 (explanatory variables): 종속변수를 예측하거나 설명하는데 사용되는 변수. 또한, 독립변수 혹은 외생변수로도 알려져 있다.
- 단순회귀 (simple regression): 단지 하나의 종속변수, 하나의 독립변수만을 갖는 회귀.
- 다중회귀 (multiple regression): 설명변수 다수를 갖는 회귀, 하지만 종속변수는 하나다.
- 선형회귀 (linear regression): 선형 모형에 기반한 회귀.
- 보통최소제곱 (ordinary least squares): 잔차 제곱 오차를 최소화함으로써 모수를 추정하는 선형회귀.
- 거짓 관계 (spurious relationship): 모형에 포함되지 않는 통계적 산출물 혹은 요인으로 발생하여 두 변수가 연관된 두 변수 사이 관계.
- 제어변수 (control variable): 거짓관계를 "제어 목적으로" 혹은 제거하는데 회귀에 포함되는 변수.
- 대리변수 (proxy variable): 다른 요인과 관계때문에 간접적으로 회귀 모형에 정보를 기여하는 변수. 그래서 그 요인에 대한 대리(proxy) 역할을 한다.

- 범주형 변수 (categorical variable): 이산형 순서없는 값을 갖는 변수.
- 결합 (join): 두 프레임에 행을 매칭하는 키(key)르 사용해서 두 데이터프레임에 데이터를 결합하는 연산.
- 데이터마이닝 (data mining): 많은 모형을 검정함으로써 변수 사이 관계를 찾아내는 접근법.
- 로지스틱 회귀 (logistic regression): 종속변수가 부울(boolean) 자료형식일 때 사용되는 회귀 형태.
- 포아송 회귀 (Poisson regression): 종속변수가 음수가 아닌 정수, 통상 갯수일 때, 사용되는 회귀 형식.
- 오즈 (odds): 확률 p를 확률과 확률보(complement) 비율로 $p/(1-p)$로 나타내는 대안적인 방법.

제 12 장

시계열 분석

시계열(time series)은 시스템에서 시간에 따라 변화하는 측정값 시퀀스다. 유명한 사례는 "하키스틱 그래프 (hockey stick graph)"로 시간에 따른 글로벌 평균 기온을 보여준다.(`https://en.wikipedia.org/wiki/Hockey_stick_graph` 참조).

이번 장에서 작업할 예제는 정치과학 연구자 Zachary M. Jones에게서 왔다. Zachary는 미국 대마초(마리화나)에 대한 암시장(black market)을 연구한다 (`http://zmjones.com/marijuana`). "담배 가격 (Price of Weed)"으로 불리는 웹사이트에서 데이터를 수집했다. 이 웹사이트는 대마초 거래장소, 가격, 품질, 수량을 참여자에게 물어 시장정보를 클라우드 소싱(crowd sourcing)했다 (`http://www.priceofweed.com/`). 이 프로젝트의 목적은 사장에 대한 합법화(legalization)같은 정책결정 효과를 조사하는 것이다. 이 프로젝트에 매력을 발견했는데 이유는 데이터를 사용해서 중요한 정치적 질문 예를 들면, 약물정책(drug policy)같이 다루기 깨문이다.

이번 장에서 흥미를 찾았으면하는 희망이 있지만, 분석에 전문가적인 태도를 유지하는 중요성을 반복하는 기회가 되었으면 한다. 약물(drug)가 불법 혹은 어느 약물이 불법이 되어야 하느냐는 중요하고 어려운 공공정책질문이다; 정직하게 보고된 정확한 데이터로 우리의 결정을 통보해야 한다.

이번 장에서 사용되는 코드는 `timeseries.py`에 있다. 코드를 다운로드하고 작업하는 것에 대한 정보는 2을 참조한다.

제 1 절 가져오기(Importing)와 정제하기(cleaning)

Mr. Jones 사이트에서 다운로드한 데이터는 이책 저장소에 있다. 다음 코드가 데이터를 읽어 판다스 데이터프레임으로 저장한다.

```
    transactions = pandas.read_csv('mj-clean.csv', parse_dates=[5])
```

`parse_dates`는 5번째 칼럼(열)값을 날짜 자료형으로 해석하고, 넘파이(NumPy) `datetime64` 객체로 전환한다.

데이터프레임에는 각각 보고된 거래건에 대한 행과 다음 칼럼(열)이 있다.

- city (도시): 문자열 도시 이름.
- state (주): 알파벳 두 글자로된 미국 주명.
- price (가격): 달러로 지불된 가격.
- amount (수량): 그램으로 구입한 수량.
- quality (품질): 구매자가 보고한 고급, 보통, 저급 품질.
- date (날짜): 보고날짜, 추측컨데 구매일 직후 날짜.
- ppg: 달러 표기된 그램당 가격 (price per gram)
- state.name: 문자열 미국 주 이름
- lat: 도시 이름에 기반한 거래가 발생한 근사치 위도 정보.
- lon: 거래가 발생한 근사치 위도 정보.

거래 각각은 시간에 따라 발생한 사건(event)으로, 이 데이터셋을 시계열 자료로 처리할 수 있다. 하지만, 사건이 시간에 균등하게 발생하는 것은 아니다; 각 날짜별로 보고된 거래 숫자는 0건 에서 수백건으로 다양한다. 시계열을 분석하는데 사용되는 많은 방법은 측정값이 균등하게 간격으로 되어있어야 한다. 혹은 만약 데이터가 동일 간격이라면 더 간단하다.

이 방법을 시연하기 위해서, 데이터셋을 대마초 품질을 보고한 집단으로 나눈다. 그리고 나서 그램당 일별 평균 가격을 계산해서 각 집단을 동일 가격 계열(equally spaced series)로 변환한다.

```
def GroupByQualityAndDay(transactions):
    groups = transactions.groupby('quality')
    dailies = {}
    for name, group in groups:
        dailies[name] = GroupByDay(group)

    return dailies
```

groupby는 GroupBy 객체를 반환하는 데이터프레임 메쏘드다; for 루프에 사용되서, 집단 이름과 집단을 표현하는 데이터프레임을 반복한다. quality가 low, medium, high이기 때문에 이 이름을 갖는 집단 세개를 얻는다.

루프는 집단을 반복하는데 GroupByDay를 호출해서 일별 평균 가격을 계산하고, 새로운 데이터프레임을 반환한다.

```
def GroupByDay(transactions, func=np.mean):
    grouped = transactions[['date', 'ppg']].groupby('date')
    daily = grouped.aggregate(func)

    daily['date'] = daily.index
    start = daily.date[0]
    one_year = np.timedelta64(1, 'Y')
    daily['years'] = (daily.date - start) / one_year

    return daily
```

모수 transactions은 데이터프레임으로 date와 ppg칼럼을 포함하는 데이터프레임이다. 칼럼을 두개 선택하고 나서, date 별로 묶는다.

grouped 결과는 날짜별로 데이터프레임으로 매핑하는데, 그 날짜에 보고된 가격을 담고 있다. aggregate는 GroupBy 메쏘드로 집단을 반복하고, 함수를 집단 각 칼럼에 적용한다; 이 경우 칼럼은 ppg 하나다. 그래서 aggregate 결과는 각 날짜에 대해 행 하나와 한 칼럼 ppg을 갖는 데이터프레임이다.

데이터프레임에 날짜는 넘파이(NumPy) datetime64 객체로 저장되어 있는데 10억분의 1초(nanoseconds) 64-비트 정수로 표현된다. 다음 분석을 위해서, 년도처럼 사람이 읽기 쉬운 단위로 시간 정보를 작업하는 것이 편리할 것이다. 그래서, GroupByDay는 index를 복사해서 date라는 칼럼을 추가하고 나서, years를 추가하는데 부동소수점 형식으로 첫 거래 이후로 년도 숫자를 담고 있다.

최종결과 데이터프레임에는 ppg, date, years 칼럼이 있다.

제 2 절 플롯 그리기 (Plotting)

GroupByQualityAndDay 결과는 각 대마초 품질로부터 일별 가격 데이터프레임으로 매핑이다. 다음에 시계열 세개를 플롯으로 그리는 코드가 있다.

```
    thinkplot.PrePlot(rows=3)
    for i, (name, daily) in enumerate(dailies.items()):
        thinkplot.SubPlot(i+1)
        title = 'price per gram ($)' if i==0 else ''
        thinkplot.Config(ylim=[0, 20], title=title)
```

그림 12.1: Time series of daily price per gram for high, medium, and low quality cannabis.

```
        thinkplot.Scatter(daily.index, daily.ppg, s=10, label=name)
        if i == 2:
            pyplot.xticks(rotation=30)
        else:
            thinkplot.Config(xticks=[])
```

rows=3을 갖는 PrePlot은 세개 하위그림(subplot)을 3행에 걸쳐 플롯을 그릴려고 한다는 것을 의미한다. 루프가 데이터프레임을 반복하면서 각각에 대해 산점도를 생성한다. 시계열 데이터를 플롯 그림으로 그릴 때 점들 사이를 선분(line segment)으로 표현하는 것이 일반적이다. 하지만, 이 경우에는 데이터 점이 많고, 가격 변동성이 크기 때문에, 선분을 추가는 것이 그다지 도움이 되지 못한다.

x-축에 라벨이 날짜라서, 가독성을 좋게 하려고 pyplot.xticks 을 사용해서 "ticks"을 30도 회전한다.

그림 12.1에 결과가 나와있다. 이 플롯 그림을 통해서 한가지 명백한 특징은 2013년 11월 틈이 있다는 것이다. 데이터 수집이 이 기간동안 활발하지 못하거나, 데이터가 이용가능하지 않았다는 것도 가능하다. 나중에 결측 데이터를 처리하는 방법을 생각해볼 것이다.

시각적으로 고품질 대마초 가격이 이 기간동안 하락하는 것처럼 보인다. 저품질 대마초 가격은 상승하는 것처럼 보이지만, 식별하기는 어렵다. 왜냐하면 변동성이 더 크기 때문이다. 품질 데이터는 자발적으로 보고한 것으로 시간에 따른 경향에는 참여자가 어떻게 품질 라벨을 적용했는지에 대한변동이 반영된다는 것을 기억하라.

제 3 절 선형 회귀 (Linear regression)

시계열분석에 특화된 방법론이 있지만, 많은 문제에 대해서, 시작하기 좋은 단순한 방법은 선형 회귀같은 범용 도구를 적용해보는 것이다. 다음 함수는 일별 가격 정보 데이터프레임을 받아서, 최소제곱 적합을 계산한다. StatsModels에서 모형과 결과 객체를 반환한다.

```
def RunLinearModel(daily):
    model = smf.ols('ppg ~ years', data=daily)
    results = model.fit()
    return model, results
```

그리고 나서, 수량을 반복하고 각각 모형에 적합한다.

그림 12.2: Time series of daily price per gram for high quality cannabis, and a linear least squares fit.

```
    for name, daily in dailies.items():
        model, results = RunLinearModel(daily)
        print(name)
        regression.SummarizeResults(results)
```

다음에 결과가 나와있다.

quality	intercept	slope	R^2
high	13.450	-0.708	0.444
medium	8.879	0.283	0.050
low	5.362	0.568	0.030

추정된 기울기는 고품질 대마초 가격이 관측구간에서 매년 약 71 센트 하락했다; 중간 품질 대마초에 대해서는 매년 약 28 센트 증가했고, 저급 품질 대마초에 대해서는 매년 약 57 센트 증가했다. 이러한 추정값은 매우 작은 p-값으로 모두 통계적으로 유의하다.

고품질 대마초에 대한 R^2 값은 0.44로 설명변수로서 시간이 관측된 가격 변동성의 44%를 설명한다. 다른 대마초 품질에 대해서는 가격 변동이 더 작고, 가격에 변동성이 더 커서, R^2 값이 더 작다(하지만, 여전히 통계적 유의성이 있다.)

다음 코드는 관측 가격과 적합값(fitted value)을 플롯으로 그린다.

```
def PlotFittedValues(model, results, label=''):
    years = model.exog[:,1]
    values = model.endog
    thinkplot.Scatter(years, values, s=15, label=label)
    thinkplot.Plot(years, results.fittedvalues, label='model')
```

8 절에서 보았듯이, `model`에는 `exog`, `endog`, 넘파이(NumPy)배열이 외생(설명), 내생(종속) 변수로 포함된다.

`PlotFittedValues`은 데이터 점을 산점도로, 적합값을 선형 플롯 그림으로 만든다. 그림 12.2에는 고품질 대마초에 대해 플롯을 그린 결과가 있다. 모형이 데이터에 대한 좋은 선형 적합처럼 보인다; 하지만, 선형회귀는 이러한 데이터에 대해서 가장 적절한 선택은 아니다.

- 첫째로, 장기 추세를 선형 혹은 다른 간단한 함수로 예측할 이유가 없다. 일반적으로, 가격은 수요와 공급에 의해 결정된다. 모두 시간에 따라 예측 불가한 방향으로 변화한다.

- 둘째, 선형회귀모형은 모든 데이터 (최근 혹은 과거)에 대해 동일한 가중치를 둔다. 예측 목적으로, 아마도 최근 데이터에 더 많은 가중치를 두어야 한다.

- 마지막으로, 선형회귀 가정중의 하나는 잔차가 상관되지 않는 잡음이라는 것이다. 시계열 데이터에서 연속된 값이 상관되기 때문에 종종 이런 가정은 틀리다.

다음 절에서 대안을 제시하는 시계열 데이터에 대해서 더 적절하다.

제 4 절 이동 평균 (Moving averages)

대부분 시계열 분석은 관측된 계열이 다음 세가지 요소 합이라는 모형화 가정에 기반한다.

- 추세 (Trend): 지속되는 변경사항을 잡아내는 평활 함수 (smooth function).
- 계절성 (Seasonality): 주기적 변동, 아마도 일별, 주별, 월별, 혹은 년별 사이클.
- 잡음 (Noise): 장기 추세 주위 확률 변동.

앞절에서 살펴봤듯이, 회귀는 계열에서 추세를 뽑아내는 한 방법이다. 하지만, 추세는 단순한 함수가 아니다. 훌륭한 대안은 **이동평균(moving average)**이다. 이동평균은 계열을 **윈도우(windows)**로 불리는 겹치지 않는 지역으로 나누고 각 윈도우에 있는 값을 평균낸다.

가장 단순한 이동평균 중의 하나는 **이동평균 (rolling mean)**이다. 각 윈도우에 있는 값을 평균낸다. 예를 들어, 만약 윈도우 크기가 3이라면, 이동 평균은 0에서 2까지, 1에서 3까지, 2에서 4까지 등등 사이에 있는 값을 평균낸다.

판다스는에는 `rolling_mean`가 있는데, 시리즈와 윈도우 크기를 인자로 받아 새로운 시리즈를 반환한다.

```
>>> series = np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> pandas.rolling_mean(series, 3)
array([ nan,  nan,   1,   2,   3,   4,   5,   6,   7,   8])
```

첫 두값은 `nan`이다; 다음 값은 첫 세 요소(0,1,2) 평균이다. 다음 값은 1,2,3의 평균. 등등

대마초 데이터에 `rolling_mean`을 적용하기 전에, 결측값을 다루어야 한다. 하나 혹은 그 이상 대마초 품질 범주에 대해서, 관측 구간에 보고되지 않은 거래가 있는 몇일이 있고 데이터 수집이 활성화되지 않은 2013년 특정 기간이 있다.

그림 12.3: Daily price and a rolling mean (left) and exponentially-weighted moving average (right).

지금까지 사용한 데이터프레임에 상기 날짜는 비워져있다; 인덱스는 데이터에 없는 날은 건너뛴다. 다음 분석에 대해서, 결측 데이터를 명시적으로 표기할 필요가 있다. 데이터프레임을 "다시 인덱싱(reindexing)"해서 이 작업을 수행한다.

```
dates = pandas.date_range(daily.index.min(), daily.index.max())
reindexed = daily.reindex(dates)
```

첫번째 행은 관측 구간 처음부터 끝까지 모든 날을 포함하는 날짜 범위를 계산한다. 두번째 행은 daily로부터 모든 데이터를 갖는 새로운 데이터프레임을 생성하지만, nan으로 채워진 모든 날째 행 열을 포함한다.

이제, 다음과 같이 이동평균 플롯을 그릴 수 있다.

```
roll_mean = pandas.rolling_mean(reindexed.ppg, 30)
thinkplot.Plot(roll_mean.index, roll_mean)
```

윈도우 크기는 30이고, roll_mean에 있는 각 값은 reindexed.ppg으로부터 30개값의 평균이다.

그림 12.3 (왼편)에 결과가 그려져 있다. 이동평균은 잡음을 평활하고 추세를 추출하는 작업을 잘 하는 것처럼 보인다. 첫 29개 값은 nan 이고, 결측값이 있는 곳에, 또다른 nan 29개가 다음에 있다. 이런 틈을 매울 수 있는 방법이 몇개 있지만, 성가신 작은 일이다.

대안은 **지수가중 이동평균 (exponentially-weighted moving average)**으로 두가지 장점이 있다. 첫째, 이름에서 암시하듯이, 가중평균을 계산하는데 가장 최근 값에 가장 높은 가중치를 두고 이전 값의 가중치는 지수적으로 하락한다. 두번째, EWMA 판다스 구현이 결측값을 더 잘 처리한다.

```
ewma = pandas.ewma(reindexed.ppg, span=30)
thinkplot.Plot(ewma.index, ewma)
```

span 모수는 대략 이동평균 윈도우 크기에 상응한다; 가중치가 얼마나 빨리 감쇄하는지 제어한다. 그래서 각 평균값에 무시못할 기여를 하는 점의 갯수를 결정한다.

그림 12.3 (오른편)에 동일한 데이터에 대한 EWMA가 그려져 있다. 둘다 정의된 지점에서는 이동평균과 비슷하다. 하지만, 결측값이 없는데 작업하기 더 수월하게 한다. 시계열 시작부근에서 값들이 잡음이 있어 보이는데, 좀더 적은 데이터 점에 모형이 기반하기 때문이다.

그림 12.4: Daily price with filled data.

제 5 절 결측값 (Missing values)

시계열 데이터 추세를 특성화했기 때문에, 다음 단계는 주기적인 행동인 계절성을 조사할 것이다. 인간 행동에 기반한 시계열 데이터는 종종 일별, 주별, 월별, 년별 주기(cycle)를 나타낸다. 다음 절에서, 계절성을 검정하는 방법을 제시한다. 하지만, 결측값에는 잘 동작하지 않는다. 그래서, 이 문제를 먼저 해결해야 한다.

결측값을 채우는 간단하고 흔한 방법이 이동평균이다. 시리즈 메쏘드 `fillna`가 원하는 것이다.

```
reindexed.ppg.fillna(ewma, inplace=True)
```

`reindexed.ppg`에 `nan`가 있는 곳 어디에서나, `fillna`가 결측값을 `ewma`에서 상응하는 값으로 교체한다. `inplace` 옵션 플래그(flag)는 `fillna`에 새로 생성하는 대신에 기존 시리즈를 변경하게 한다.

이 방식의 결점은 시리즈에 잡음(noise)을 과소추정하는 것이다. 재표본추출된 잔차(resampled residual)를 추가해서 이 문제를 해결할 수 있다.

```
resid = (reindexed.ppg - ewma).dropna()
fake_data = ewma + thinkstats2.Resample(resid, len(reindexed))
reindexed.ppg.fillna(fake_data, inplace=True)
```

`resid`에는 `ppg`가 `nan`일 때 포함되지 않은 날 잔차값이 포함되어있다. `fake_data`에는 이동평균 합과 잔차 확률표본(random sample)이 담겨있다. 마지막으로, `fillna`는 `nan`를 `fake_data` 값으로 바꾼다.

그림 12.4에 결과가 나와있다. 채워진 데이터는 시각적으로 실제값과 비슷하다. 재표본추출한 잔차가 랜덤(random)으로 결과는 매번 달라진다; 나중에, 결측값에서 생성된 오차를 어떻게 특성화하는지 살펴볼 것이다.

제 6 절 계열 상관 (Serial correlation)

가격은 매일 매일 변화하는데, 패턴을 보고 싶을지 모른다. 만약 월요일에 가격이 높다면, 다음 몇일동안 가격이 높을 것을 예상할 수 있다; 그리고, 만약 낮다면, 낮게 유지될 것을 예상할 수 있다. 이와 같은 패턴이 **계열 상관 (serial correlation)**이라고 부른다. 왜냐하면 각 값이 계열에 다음 값과 상관되기 때문이다.

계열 상관을 계산하기 위해서, 시계열을 **시차(lag)**로 불리는 구간만큼 이동할 수 있다. 그리고 나서, 원시계열과 이동된 시계열 사이 상관을 계산한다.

```
def SerialCorr(series, lag=1):
    xs = series[lag:]
    ys = series.shift(lag)[lag:]
    corr = thinkstats2.Corr(xs, ys)
    return corr
```

이동한 후에, 첫 시차 (lag) 값이 nan이라, 슬라이스를 사용해서 Corr를 계산하기 전에 제거한다.

만약 SerialCorr을 시차 1을 가진 원가격데이터에 적용하면, 계열상관이 고품질 대마초에 대해 0.48, 중간품질에는 0.16, 저품질에는 0.10이 된다. 장기 추세를 갖는 어떤 시계열에 대해서도, 강한 계열상관을 예상한다; 예를 들어, 만약 가격이 떨어지고 있다면, 계열 절반 전반부에는 평균보다 상위 값을, 계열 절반 후반부에는 평균보다 하위 값을 예상한다.

만약 추세를 제거한다면, 상관이 지속하는지 살펴보는 것이 더 흥미롭다. 예를 들어, EWMA 잔차를 계산하고 나서 계열 상관을 계산한다.

```
ewma = pandas.ewma(reindexed.ppg, span=30)
resid = reindexed.ppg - ewma
corr = SerialCorr(resid, 1)
```

lag=1으로 추세 제거된 데이터에 대한 계열 상관은 고품질에는 -0.022, 중간품질에는 -0.015, 저품질에는 0.036이다. 계열 상관값이 작아서, 이 시계열 데이터에는 하루 계열상관이 매우 작거나 없다.

주별, 월별, 년별 계절성을 점검하기 위해서, 다시 다른 시차를 갖는 분석을 실행한다. 다음에 결과가 나와있다.

lag	high	medium	low
1	-0.029	-0.014	0.034
7	0.02	-0.042	-0.0097
30	0.014	-0.0064	-0.013
365	0.045	0.015	0.033

다음절에서, 이러한 상관이 통계적 유의성(통계적 유의성은 없다)이 있는지 검정할 것이다. 하지만, 이 지점에서 잠정적으로 이들 시계열 데이터에 상당한 계절적 패턴이, 적어도 이런 시차에는 없다고 결론내릴 수 있다.

제 7 절 자기상관 (Autocorrelation)

만약 시계열이 계열상관을 갖고 있다고 생각하지만 어떤 시차를 검정할지 모른다면, 모든 시차를 검정할 수 있다!!! **자기상관 함수 (autocorrelation function)**는 시차에서 주어진 시차를 갖는 계열 상관에 매핑하는 함수다. "자기상관(Autocorrelation)"은 계열상관의 또 다른 이름으로, 시차가 1이 아닐 때 더 많이 사용된다.

그림 12.5: Autocorrelation function for daily prices (left), and daily prices with a simulated weekly seasonality (right).

StatsModels는 1절에 선형회귀에 사용했는데, 자기상관 함수를 계산하는 acf를 포함하여 시계열분석에 함수도 제공한다.

```
import statsmodels.tsa.stattools as smtsa
acf = smtsa.acf(filled.resid, nlags=365, unbiased=True)
```

acf는 계열상관을 0 시차부터 nlags까지 계산한다. unbiased 플래그옵션은 acf에 표본크기에 대한 추정값을 보정하게 한다. 결과는 상관 배열이다. 만약 고품질 대마초에 대한 일별 가격을 선택하고, 1, 7, 30, 365 시차에 대한 상관을 추출한다면, acf와 SerialCorr는 근사적으로 거의 동일한 결과를 산출한다.

```
>>> acf[0], acf[1], acf[7], acf[30], acf[365]
1.000, -0.029, 0.020, 0.014, 0.044
```

lag=0으로, acf 본인과 계열 상관을 계산하는데, 항상 1 이다.

그림 12.5 (왼편)에는 nlags=40 시차로 3개 품질 범주에 대한 자기상관함수를 보여준다. 회색지역에는 만약 실제 자기상관이 없다면 예상되는 정규 변동성이 나타나 있다; 이 범위 밖에 있는 어느 것이나 p-값이 5%보다 작아서 통계적 유의성이 있다. 거짓양성(false positive)가 5% 이고 120개 상관(3개 시계열에 대해서 시차 40)을 계산하기 때문에, 이 구역 박에 약 6개 점이 예상된다. 사실 7개 점이 있다. 이들 시계열에는 우연으로 설명될 수 없는 자기상관이 없다고 결론낸다.

잔차를 재표본추출해서 회식 지역을 계산했다. 코드는 timeseries.py 파일에 있다; 함수는 SimulateAutocorrelation다.

계절적 성분이 있을 때, 자기상관 함수가 어떤 느낌인지 살펴보기 위해서, 주별 주기를 더해서 모의 시험 데이터를 생성했다. 대마초 수요가 주말에 더 높다고 가정하면, 가격이 더 높을 것으로 예상할 수 있다. 효과를 모의 시험하기 위해서, 금요일 혹은 토요일 날짜를 선택하고, 가격에 $0 에서 $2 까지 균등분포에서 추출한 임의 금액을 더한다.

```
def AddWeeklySeasonality(daily):
    frisat = (daily.index.dayofweek==4) | (daily.index.dayofweek==5)
    fake = daily.copy()
    fake.ppg[frisat] += np.random.uniform(0, 2, frisat.sum())
    return fake
```

frisat는 부울 시리즈로, 만약 여일이 금요일 혹은 토요일이면, 참(True)이다. fake는 새 데이터프레임으로 원래 daily 복사본으로 ppg에 확률값(random value)을 더해서 변경한다. frisat.sum()은 금요일과 토요일 전체 갯수로 생성해야하는 확률값 갯수가 된다.

그림 12.5 (오른편)에는 모의시험 계절성을 갖는 가격에 대한 자기상관 함수가 나타나 있다. 예상했듯이, 시차가 7의 곱이 될 때 가장 높다. 고급과 중급 품질에 대해 신규 상관이 통계적 유의성이 있다. 저급 품질에 대서는 유의적이지 않은데 이유는 이 범주에 잔차가 크기 때문이다; 잡음을 뚫고 보여지기 위해서는 효과가 더 커야한다.

제 8 절 예측 (Prediction)

시계열 분석은 시간에 변화하는 시스템 행동을 조사하고 때때로 설명하는데 사용될 수 있다. 또한 예측도 할 수 있다.

3절에서 사용한 선형회귀는 예측에 사용될 수 있다. RegressionResults 클래스에 `predict`를 사용해서 설명변수를 포함하는 데이터프레임을 인자로 받아 예측 시퀀스를 반환한다. 다음에 코드가 있다.

```
def GenerateSimplePrediction(results, years):
    n = len(years)
    inter = np.ones(n)
    d = dict(Intercept=inter, years=years)
    predict_df = pandas.DataFrame(d)
    predict = results.predict(predict_df)
    return predict
```

`results`는 RegressionResults 객체다; `years`는 예측하려는 시간 값 시퀀스다. 함수가 데이터프레임을 생성하고 `predict`에 전달하고 결과를 반환한다.

만약 원하는 모든 것이, 하나의 최선을 다한 예측이라면, 완료했다. 하지만, 대부분 목적에 대해서, 오차를 정량화하는 것이 중요하다. 다른 말로, 예측이 얼마의 정확성이 있는지 알고싶다.

고려해야 하는 오류 원천이 3가지 있다.

- 표집오차 (Sampling error): 예측은 추정 모수에 기반하는데, 추정 모수는 표본 확률변동에 의존한다. 만약 실험을 다시 실시한다면, 추정값이 변화할 것으로 예상된다.

- 확률변동 (Random variation): 설사 추정 모수가 완벽할지라도, 관측 데이터는 장기추세주변에서 확률적으로 변동하고 이러한 변동은 미래에도 계속될 것으로 예상된다.

- 모형화 오차 (Modeling error): 장기 추세가 선형이 아니라는 증거를 이미 봤다. 그래서 선형 모형에 기반한 예측은 결국 실패할 것이다.

고려할 또 다른 오류 원천은 예상치 못한 미래 사건이다. 농산물 가격은 날씨에 영향을 받고, 모든 가격은 정치와 법에 영향하에 있다. 이 책을 집필하고 있을 때, 대마초는 미국 두주에서 합법이고, 의료목적으로 20개 이상 주에서 합법이다. 만약 더 많은 주가 합법화한다면, 가격은 내려갈 것이다. 하지만, 연방정부가 단속한다면, 가격이 치솟을지 모른다.

모형화 오차 (modeling error)와 예상치 못한 미래 사건은 정량화하기 어렵니다. 표집오차와 확률변동은 다루기 더 쉽다. 그래서 먼저 이것을 다뤄보자.

표집오차를 정량화하기 위해서, 4절에서 했던 것처럼 재표본추출법을 사용한다. 늘 그렇듯이, 목적은 만약 실험을 다시 실행한다면 무엇이 발생할 것인지 모의 시험하는데 실제 관측점을 사용하는 것이다. 모의시험은 추정 모수가 맞지만 확률 잔차가 다를 수 있다는 가정에 기반한다. 여기 모의 시험을 수행하는 함수가 있다.

```
def SimulateResults(daily, iters=101):
    model, results = RunLinearModel(daily)
    fake = daily.copy()

    result_seq = []
    for i in range(iters):
        fake.ppg = results.fittedvalues + Resample(results.resid)
        _, fake_results = RunLinearModel(fake)
        result_seq.append(fake_results)

    return result_seq
```

`daily`는 관측 가격을 포함하는 데이터프레임이다; `iters`는 모의 시험할 횟수다.

`SimulateResults`는 3절로부터 `RunLinearModel`을 사용해서 관측값의 기울기와 절편을 추정한다.

루프를 반복할 때마다, 잔차를 재표본추출하고 적합값에 더해서 "fake" 데이터 셋을 생성한다. 그리고 나서, "fake" 데이터에 선형모형을 실행하고 RegressionResults 객체에 저장한다.

다음 단계로 예측을 생성하는데 모의 시험 결과를 사용한다.

```
def GeneratePredictions(result_seq, years, add_resid=False):
    n = len(years)
    d = dict(Intercept=np.ones(n), years=years, years2=years**2)
    predict_df = pandas.DataFrame(d)

    predict_seq = []
    for fake_results in result_seq:
```

그림 12.6: Predictions based on linear fits, showing variation due to sampling error and prediction error.

```
            predict = fake_results.predict(predict_df)
            if add_resid:
                predict += thinkstats2.Resample(fake_results.resid, n)
            predict_seq.append(predict)

    return predict_seq
```

GeneratePredictions는 이전 단계에서 결과 시퀀스 뿐만 아니라 예측을 생성할 구간을 명시하는 부동소수점 시퀀스 years, 그리고 재표본추출 잔차가 직선 예측에 추가해야 하는지를 말지를 나타내는 add_resid을 인자로 받는다. GeneratePredictions은 RegressionResults 시퀀스를 반복해서 예측 시퀀스를 생성한다.

마지막으로, 다음에 예측으로 90% 신뢰구간을 플롯으로 그리는 코드가 있다.

```
def PlotPredictions(daily, years, iters=101, percent=90):
    result_seq = SimulateResults(daily, iters=iters)
    p = (100 - percent) / 2
    percents = p, 100-p

    predict_seq = GeneratePredictions(result_seq, years, True)
    low, high = thinkstats2.PercentileRows(predict_seq, percents)
    thinkplot.FillBetween(years, low, high, alpha=0.3, color='gray')

    predict_seq = GeneratePredictions(result_seq, years, False)
    low, high = thinkstats2.PercentileRows(predict_seq, percents)
    thinkplot.FillBetween(years, low, high, alpha=0.5, color='gray')
```

PlotPredictions이 GeneratePredictions을 두번 호출한다; 한번은 add_resid=True이고, 다른 한번은 add_resid=False이다. PercentileRows를 사용해서 각 년도에 대해서 5번째와 95번째 백분위수를 선택한다. 그리고 나서, 경계 사이 회색지역을 플롯으로 그린다..

그림 12.6에 결과가 나와 있다. 짙은 회색 지역이 표집오차에 대한 90% 신뢰구간을 나타낸다; 즉, 표집때문에 추정 기울기와 절편에 대한 불확실성.

밝은 지역은 예측 오차에 대한 90% 신뢰구간을 보여주는데, 표집오차와 확률변동 합이다.

이들 지역이 표집오차, 확률변동을 정량화하지만, 모형화 오류는 정량화하지 않는다. 일반적으로 모형화 오류는 정량화하기 어렵다. 하지만, 이경우에 최소한 오차 원천 하나(예측치 못한 외부 사건)는 다룰 수 있다.

그림 12.7: Predictions based on linear fits, showing variation due to the interval of observation.

회귀 모형은 시스템이 정상성(stationary, 시불변)이 있다는 가정에 기반한다; 즉, 모형 모수는 시간에 따라 변하지 않는다. 좀더 구체적으로, 기울기와 절편 뿐만 아니라 잔차 분포도 일정하다고 가정한다.

하지만, 그림 12.3에 있는 이동 평균을 살펴보면, 최소한 관측 기간동안 한번 기울기는 변하고 잔차 분산이 후반부보다 전반부에서 더 큰 것처럼 보인다.

결과로, 추정한 모수는 관측 기간에 의존성이 있다. 이것이 예측에 얼마나 많은 효과를 갖는지 살펴보기 위해서, 다른 시작과 종료 날짜를 갖는 관측 구간을 사용하도록 `SimulateResults`를 연장한다. 구현은 `timeseries.py` 파일에 있다.

그림 12.7에 중간 품질 대마초에 대한 결과가 나와 있다. 옅은 회색 구역이 신뢰구간을 보여주는데 표집오차, 확률변동, 관측구간 변동으로 인한 불확실성이 포함한다.

전체 구간에 기반한 모형이 양수 기울기를 갖고 있는데, 가격이 오르고 있다는 것을 나타낸다. 하지만, 가장 최신 구간은 하락하는 가격 신호를 보여준다. 그래서 가장 최신 데이터에 기반한 모형은 음수 기울기를 갖는다. 결과로, 가장 폭이 넓은 구간은 내년에 하락하는 가격 가능성을 포함한다.

제 9 절 추가 읽기

시계열 분석은 큰 주제다; 이번 장에서 단지 표면만 긁었을 뿐이다. 시계열 데이터로 작업하는데 중요한 도구는 자기회귀로 여기서는 다루지 않았다. 왜냐하면, 작업한 예제 데이터에 사용하기에 적합하지 않았기 때문이다.

하지만, 이장에 있는 내용를 학습하면, 자기회귀를 학습할 준비가 된 것이다. 추천하는 한 교재는 Philipp Janert가 저술한 책, *Data Analysis with Open Source Tools* O'Reilly Media, 2011. 시계열에 대한 장에서 여기서 다루지 않는 내용을 학습할 수 있다.

제 10 절 연습문제

My solution to these exercises is in `chap12soln.py`.

Exercise 12.1 The linear model I used in this chapter has the obvious drawback that it is linear, and there is no reason to expect prices to change linearly over time. We can add flexibility to the model by adding a quadratic term, as we did in Section 3.

Use a quadratic model to fit the time series of daily prices, and use the model to generate predictions. You will have to write a version of `RunLinearModel` that runs that quadratic model, but after that you should be able to reuse code in `timeseries.py` to generate predictions.

Exercise 12.2 Write a definition for a class named `SerialCorrelationTest` that extends `HypothesisTest` from Section 2. It should take a series and a lag as data, compute the serial correlation of the series with the given lag, and then compute the p-value of the observed correlation.

Use this class to test whether the serial correlation in raw price data is statistically significant. Also test the residuals of the linear model and (if you did the previous exercise), the quadratic model.

Exercise 12.3 There are several ways to extend the EWMA model to generate predictions. One of the simplest is something like this:

1. Compute the EWMA of the time series and use the last point as an intercept, `inter`.
2. Compute the EWMA of differences between successive elements in the time series and use the last point as a slope, `slope`.
3. To predict values at future times, compute `inter + slope * dt`, where `dt` is the difference between the time of the prediction and the time of the last observation.

Use this method to generate predictions for a year after the last observation. A few hints:

- Use `timeseries.FillMissing` to fill in missing values before running this analysis. That way the time between consecutive elements is consistent.
- Use `Series.diff` to compute differences between successive elements.
- Use `reindex` to extend the DataFrame index into the future.
- Use `fillna` to put your predicted values into the DataFrame.

제 11 절 용어 사전

- 시계열(time series): 각 값이 시간도장(timestamp)과 연관된 데이터셋. 종종 측정값과 추집된 시점 계열.
- 윈도우 (window): 이동 평균을 계산하는 종종 사용되는 시계열에 연속값 시퀀스.
- 이동평균 (moving average): 겹쳐지지 않는 일련의 윈도우에 대한 평균을 계산함으로써 시계열에 잠재하는 추세를 추정하는데 사용되는 여러 통계량 중의 하나.
- 이동평균 (rolling mean):각 윈도우 평균값에 기반한 이동평균.
- 지수가중이동평균 (exponentially-weighted moving average, EWMA): 가중평균에 기반한 이동평균으로 가장 최근 값에 가장 높은 가중치를 두고, 이전 값에 대해서는 지수적으로 줄어드는 가중치를 둔다.
- 스팬 (span): 가중치가 얼마나 빨리 줄어드는지를 결정하는 EWMA 모수.
- 계열상관 (serial correlation): 한 시계열과 이동된 혹은 시차이동한 자신 시계열과 상관.
- 시차 (lag): 계열 상관 혹은 자기상관에서 이동 크기.
- 자기상관 (autocorrelation): 임의 시차를 갖는 계열상관에 대한 좀더 일반적인 용어.
- 자기상관 함수 (autocorrelation function): 시차에서 계열상관으로 매핑하는 함수.
- 정상성 (stationary): 만약 모수와 잔차 분포가 시간에 따라 변화하지 않는다면, 모형이 정상성이 있다.

제 13 장

생존분석

생존분석(Survival analysis)은 무언가 얼마나 지속하는지를 기술하는 방법이다. 종종 사람 생명 연구에 사용되지만, 또한 기계나 전자 부품의 "생존(survial)" 혹은 좀더 일반적으로 사건 전 시간 간격에도 적용된다.

만약 여러분이 알고 있는 누군가 생명을 위협하는 질병을 진단받았다면, "5년 생존율 (5-year survival rate)"을 들어봤을지도 모른다. 진단 후에 5년을 생존할 확률이다. 이 추정값과 관련된 통계량이 생존분석 결과다.

이번 장에서 사용되는 코드는 `survival.py`에 있다. 코드를 다운로드하고 작업하는 것에 대한 정보는 2을 참조한다.

제 1 절 생존곡선 (Survival curves)

생존분석에 기본 개념은 **생존곡선 (survival curve)** $S(t)$로, 존속 t를 t보다 더 오래 생존할 확률로 매핑하는 함수다. 만약 존속(duration) 분포 즉, "수명(lifetimes)"을 알고 있다면, 생존곡선을 찾는 것은 쉽다; CDF의 여분포(complement)가 된다.

$$S(t) = 1 - \text{CDF}(t)$$

여기서, $CDF(t)$는 t보다 적거나 같은 수명 확률이다.

예를 드어, NSFG 데이터셋에서 11189 출산 존속기간을 알고 있다. 이 데이터를 읽어서 CDF를 계산할 수 있다.

```
preg = nsfg.ReadFemPreg()
complete = preg.query('outcome in [1, 3, 4]').prglngth
cdf = thinkstats2.Cdf(complete, label='cdf')
```

그림 13.1: Cdf and survival function for pregnancy length (top), hazard function (bottom).

결과 코드(outcome code) 1, 3, 4은 정상출산, 사산, 유산을 각각 나타낸다. 분석을 위해서 유발유산(induced abortion), 자궁외 임신(ectopic pregnancy), 그리고 응답자와 인터뷰중에 임신상태인 경우는 제외한다.

데이터프레임 메쏘드 query는 부울 표현식을 인자로 받아서 각 행마다 평가하고 참(True)을 산출하는 행을 선택한다.

그림 13.1 (상단)에 임신기간 CDF와 생존함수인 상보 CDF가 보여진다. 생존함수를 표현하기 위해서, 객체를 정의해서 Cdf를 래핑(wrapping) 인터페이스를 조정하여 맞춘다(adapt).

```
class SurvivalFunction(object):
    def __init__(self, cdf, label=''):
        self.cdf = cdf
        self.label = label or cdf.label

    @property
    def ts(self):
        return self.cdf.xs

    @property
    def ss(self):
        return 1 - self.cdf.ps
```

SurvivalFunction는 프로퍼티(property)를 두개 제공한다; ts는 수명 시퀀스고, ss는 생존함수다. 파이썬에서 "프로퍼티(property)"는 마치 변수처럼 호출할 수 있는 메쏘드다.

수명 CDF를 인자로 넘김으로써 SurvivalFunction를 인스터스화할 수 있다.

```
    sf = SurvivalFunction(cdf)
```

또한 SurvivalFunction는 __getitem__와 Prob을 제공하는데 생존함수를 평가한다.

```
# class SurvivalFunction

    def __getitem__(self, t):
        return self.Prob(t)

    def Prob(self, t):
        return 1 - self.cdf.Prob(t)
```

예를 들어, sf[13] 는 임신초기 3개월을 지난 임신 비율이다.

```
>>> sf[13]
0.86022
>>> cdf[13]
0.13978
```

약 86% 임신이 첫 3개월을 지났다; 약 14% 그렇지 못하다.

SurvivalFunction는 Render 기능있다. 그래서, thinkplot에 함수를 사용해서 sf 플롯을 그릴 수 있다.

```
    thinkplot.Plot(sf)
```

그림 13.1 (상단)에 결과가 나와 있다.

곡선은 거의 13주차에서 26주차는 거의 평평해서, 임신 중기에는 임신 몇몇사례만 중단되는 것을 보여준다. 그리고, 곡선이 39주차에 가장 가파른데 가장 흔한 임신기간이다.

제 2 절 위험 함수 (Hazard function)

생존함수에서 **위험함수(hazard function)**를 도출할 수 있다; 임신기간에 대해서, 위험함수는 시간 t에서 t까지 계속되고 나서 t에서 끝나는 임신 비율이다. 좀더 정확하게는 다음과 같다.

$$\lambda(t) = \frac{S(t) - S(t+1)}{S(t)}$$

분자는 PMF(t)로 t에서 끝나는 수명 비율이다.

SurvivalFunction는 MakeHazard 함수를 제공하는데, 위험 함수를 계산한다.

```
# class SurvivalFunction

    def MakeHazard(self, label=''):
        ss = self.ss
        lams = {}
        for i, t in enumerate(self.ts[:-1]):
            hazard = (ss[i] - ss[i+1]) / ss[i]
            lams[t] = hazard

        return HazardFunction(lams, label=label)
```

HazardFunction 객체는 판다스 시리즈 에 대한 랩퍼다.

```
class HazardFunction(object):

    def __init__(self, d, label=''):
        self.series = pandas.Series(d)
        self.label = label
```

d는 딕셔너리 혹은 또다른 시리즈를 포함해서 시리즈를 초기화할 수 있는 다른 어떤 자료형도 될 수 있다. label은 플롯으로 그림을 그렸을 때, HazardFunction을 식별하는데 사용되는 문자열이다.

HazardFunction은 __getitem__을 제공해서, 다음과 같이 평가할 수 있다.

```
>>> hf = sf.MakeHazard()
>>> hf[39]
0.49689
```

So of all pregnancies that proceed until week 39, about 50% end in week 39.

그림 13.1 (아래)에 임신 기간에 대한 위험함수가 보여진다. 임신 42주차 이후 시간에 대해, 위험함수가 불규칙한데, 적은 사례에 기초하기 때문이다. 그 부분을 제외하고, 곡선 모양은 예상한 것과 같다; 약 39주차에 가장 높고, 임신 중기보다 초기에 약간 더 높다.

위험함수는 그자체로 유용하지만, 또한 다음 절에서 살펴보듯이, 생존곡선을 추정하는데 중요한 도구가 된다.

제 3 절 생존곡선 추정하기

만약 누군가 여러분에게 수명 CDF를 준다면, 생존함수와 위험함수를 계산하기는 쉽다. 하지만, 많은 현실 시나리오에서, 직접 수명분포를 측정할 수는 없다. 유추해야만 한다.

예를 들어, 진단뒤에 얼마나 오랜기간동안 환자가 생존하는지 살펴보기 위해서 한 환자집단을 추적조사한다고 가정하자. 모든 환자가 동일한 날에 진단받은 것은 아니다. 그래서 시점 아무때나 일부 환자가 다른 환자보다도 더 오래 생존한다. 만약 환자 일부가 죽는다면, 죽은 환자 생존시간을 알게된다. 여전히 살아 있는 환자에 대해서, 생존시간을 알지 못하지만, 생존시간 하한에 대한 정보는 갖게 된다.

만약 모든 환자가 죽을 때까지 기다린다면, 생존곡선을 계산할 수 있다. 하지만, 새로운 처리(treatment)에 대한 효과성을 평가한다면, 그렇게 오래 기다릴 수는 없다. 불완전한 정보 (incomplete information)를 사용해서 생존곡선을 추정할 방법이 필요하다.

좀더 명량한 예제로, NSFG 데이터를 사용해서 응답자가 초혼 때까지 얼마나 오래 "생존(survive)"하는지 정량화할 것이다. 응답자 연령 범위는 14세에서 44세까지다. 그래서, 데이터셋이 일생에서 서로다른 단계에 있는 여성의 스탭샷(snapshot) 정보를 제공한다.

결혼한 여성에 대해서, 데이터셋에는 초혼 날짜와 그 당시 연령 정보가 포함되어 있다. 결혼하지 않은 여성에 대해서는 인터뷰했을 당시 연령정보가 있지만, 언제 혹은 결혼을 할 것인지 알 수 있는 방법이 없다.

몇몇 여성에 대한 초혼 연령을 알고 있기 때문에, 나머지를 제외하고 정보가 있는 데이터 CDF를 계산하고 싶은 유혹이 있다. 이것은 매우 바람직하지 못하다. 결과가 두가지 방향에서 잘못 도출될 수 있다; (1) 좀더 나이가 많은 여성이 더 많이 대표 표본으로 되는데, 인터뷰 당시 좀더 결혼할 것 같기 때문이다. (2) 결혼한 여성이 더 많이 대표 표본이 될 것이다. 사실, 분석 결과 모든 여성은 결혼한다는 결론이 도출될 것이다. 하지만 분명하게 틀렸다.

제 4 절 캐플란-마이어 추정 (Kaplan-Meier estimation)

이번 예제에서, 결혼하지 않은 여성 관측정보를 포함하는 것은 바람직할 뿐만 아니라 필요한데 이유는 생존분석에서 중심적인 알고리즘 중의 하나인 **캐플란-마이어 추정(Kaplan-Meier estimation)**과 연결되기 때문이다.

일반적인 생각은 데이터를 사용해서, 위험함수를 추정하고 나서, 위험함수를 생존함수로 전환한다. 위험함수를 추정하기 위해서, 각 연령별로 다음을 고려한다. (1) 그 연령에 결혼한 여성 숫자, (2) 결혼 "위험 상태(at risk)"에 있는 여성 숫자로 이전 연령에서 결혼하지 않은 모든 여성이 포함된다.

다음에 코드가 있다.

```
def EstimateHazardFunction(complete, ongoing, label=''):

    n = len(complete)
    hist_complete = thinkstats2.Hist(complete)
    sf_complete = SurvivalFunction(thinkstats2.Cdf(complete))

    m = len(ongoing)
    sf_ongoing = SurvivalFunction(thinkstats2.Cdf(ongoing))

    lams = {}
    for t, ended in sorted(hist_complete.Items()):
        at_risk = ended + n * sf_complete[t] + m * sf_ongoing[t]
```

```
        lams[t] = ended / at_risk

    return HazardFunction(lams, label=label)
```

complete는 완벽한 관측 집단이다; 이 경우에 응답자가 결혼했을 때 연령이 된다. ongoing은 완벽하지 못한 관측 집단이다; 이 경우에 인터뷰를 했을 때 미혼인 여성 연령이 된다.

먼저, 응답자가 결혼했을 때 연령 Hist인 hist_complete, 결혼한 여성에 대한 생존함수, sf_complete, 미혼 여성에 대한 생존함수 sf_ongoing를 미리 계산한다.

응답자가 결혼했을 때 루프는 연령을 반복한다. t 각 값에 대해, t 연령에서 결혼한 여성 숫자인 ended가 있다. 그리고 나서 다음의 합으로 "위험 상태(at risk)" 여성 숫자를 계산한다.

- ended, 연령 t에서 결혼한 응답자 숫자.
- n * sf_complete[t], 연령 t 뒤에 결혼한 응답자 숫자.
- m * sf_ongoing[t], 연령 t 후에 인터뷰한 미혼 응답자 숫자, 그러므로 t 시점 혹은 이전에 결혼했는지 알려지지 않았다.

시점 t에 위험함수 추정값은 at_risk에 대한 ended의 비율이다.

lams은 딕셔너리로 t에서 $\lambda(t)$으로 매핑한다. 결과는 HazardFunction 객체다.

제 5 절 결혼 곡선 (marriage curve)

이 함수를 검정하기 위해서, 데이터 정제와 변환을 수행해야 한다. 필요한 NSFG 변수는 다음과 같다.

- cmbirth: 모든 응답자에 대해서 알려진, 응답자 생년월일.
- cmintvw: 모든 응답자에 대해 알려진, 응답자가 인터뷰한 날짜.
- cmmarrhx: 알려져있고 해당되면, 응답자가 첫 혼인한 날짜.
- evrmarry: 만약 인터뷰 날짜 이전에 결혼했다면 1, 그렇지 않으면 0.

첫 변수 세개는 "세기-월(century-months)" 방식으로 부호화되었다; 즉, 1899년 12월 이후 정수형 개월 숫자. 그래서, 세기-월(century-month) 1은 1900년 1월이 된다.

첫째, 응답자 파일을 읽고, cmmarrhx에 있는 타당하지 않는 값 (invalid value)을 교체한다.

```
    resp = chap01soln.ReadFemResp()
    resp.cmmarrhx.replace([9997, 9998, 9999], np.nan, inplace=True)
```

그리고 나서, 혼인 당시에 각 응답자 연령과 인터뷰 당시 연령을 계산한다.

```
    resp['agemarry'] = (resp.cmmarrhx - resp.cmbirth) / 12.0
    resp['age'] = (resp.cmintvw - resp.cmbirth) / 12.0
```

다음에, complete 변수에 결혼한 여성에 대해서 결혼 당시 연령 정보를 뽑아낸다. 그리고, ongoing 변수에 결혼하지 않은 여성에 대한 인터뷰 당시 연령 정보를 대입한다.

```
    complete = resp[resp.evrmarry==1].agemarry
    ongoing = resp[resp.evrmarry==0].age
```

마지막으로, 위험함수를 계산한다.

```
    hf = EstimateHazardFunction(complete, ongoing)
```

그림 13.2 (위)에 추정 위험함수가 그려져 있다; 10대에는 낮고, 20대에는 더 높고, 30대에는 감소한다. 40대에 다시 증가한다. 하지만, 이것은 추정 과정의 산출물이다; "위험 상황(at risk)" 응답자 숫자가 감소함에 따라, 혼인한 적은 여성이 높은 추정 위험을 산출한다. 생존함수는 이러한 잡음을 부드럽게 평활한다.

제 6 절 생존함수 추정하기

위험함수를 갖게 되면, 생존함수를 추정할 수 있다. t 시점을 지나 생존할 가망성은 줄곧 살아서 t 시점까지 생존할 가망성이 되는데, 보수 위험함수(complementary hazard function)의 누적 곱이 된다.

$$[1 - \lambda(0)][1 - \lambda(1)]...[1 - \lambda(t)]$$

HazardFunction 클래스는 상기 누적곱을 계산하는 MakeSurvival 함수를 제공한다.

```
# class HazardFunction:

    def MakeSurvival(self):
        ts = self.series.index
        ss = (1 - self.series).cumprod()
        cdf = thinkstats2.Cdf(ts, 1-ss)
        sf = SurvivalFunction(cdf)
        return sf
```

그림 13.2: Hazard function for age at first marriage (top) and survival function (bottom).

ts는 위험 함수가 추정되는 시점 시퀀스다. ss는 보수 위험함수 누적곱이다. 따라서, 생존함수가 된다.

SurvivalFunction가 구현되는 방식 때문에, ss 보수를 계산하고, Cdf를 만들고 나서, SurvivalFunction 객체를 인스턴스화 한다.

그림 13.2 (아래) 에 결과가 그려져 있다. 생존곡선은 대부분의 여성이 결혼하는 25세에서 35세 사이가 가장 가파르다. 35세에서 45세 사이는 거의 평평하다. 35세 전에 결혼하지 않은 여성이 혼일할 것 같지 않다는 것을 나타낸다.

이와 같은 곡선이 1986년 유명한 잡지기사의 기초가 된다; *뉴스위크(Newsweek)* 에 따르면, 40세된 미혼 여성이 혼인보다도 "테러리스트에 의해 더 살해될 가능성"이 있다. 이러한 통계량은 널리 보도되고 대중적인 문화 일부분이 되었다. 하지만, 그러고 나서 잘못되었다(왜냐하면, 오류가 있는 분석에 기반했기 때문이다) 그리고, 심지어 더 잘못된 것으로 밝혀졌다(이미 진행중이고, 지속되는 문화 변화 때문이다). 2006년 *뉴스위크(Newsweek)*는 보도가 잘못되었다고 시인하는 또다른 기사를 실었다.

이 기사, 기사의 기반이 된 통계량, 그리고 반응에 관해 더 읽어보기를 격려한다. 신중히 통계분석을 수행하고, 적절한 의심(appropriate skepticism)를 가지고 결과를 해석하고, 공공에게 정확하고 정직하게 제시하는데 있어, 윤리적 책임의 중요성을 상기시켰으면 한다.

제 7 절 신뢰구간 (Confidence intervals)

캐플란-마이어 분석(Kaplan-Meier analysis)은 생존곡선에 대한 단 하나의 추정값을 산출한다. 하지만, 추정값의 불확실성을 정량화하는 것도 또한 중요하다. 늘 그렇듯이, 세가지 오차 원천이 있다; 측정오차, 표집오차, 모형화 오차.

이번 예제에서, 측정오차는 대략 작다. 일반적으로, 사람의 출생년월, 혼인여부, 혼인 시점은 알고 있다. 그리고, 이러한 정보를 정확하게 보고할 것으로 예상할 수 있다.

재표본추출(resampling)을 통해서 표집오차를 정량화할 수 있다. 다음에 코드가 있다.

```
def ResampleSurvival(resp, iters=101):
    low, high = resp.agemarry.min(), resp.agemarry.max()
    ts = np.arange(low, high, 1/12.0)
```

그림 13.3: Survival function for age at first marriage and a 90% confidence interval based on weighted resampling.

```
ss_seq = []
for i in range(iters):
    sample = thinkstats2.ResampleRowsWeighted(resp)
    hf, sf = EstimateSurvival(sample)
    ss_seq.append(sf.Probs(ts))

low, high = thinkstats2.PercentileRows(ss_seq, [5, 95])
thinkplot.FillBetween(ts, low, high)
```

`ResampleSurvival` 함수는 응답자 데이터프레임 `resp`와 재표본추출 횟수 `iters`을 인자로 받는다. `ts`를 계산하는데 생존함수를 평가하는 연령 시퀀스다.

루프 내부에, `ResampleSurvival`는 다음과 같다:

- 7절에서 살펴본 `ResampleRowsWeighted`을 사용해서 응답자를 재표본추출한다.
- `EstimateSurvival` 호출하는데, 위험곡선과 생존곡선을 추정하는데 앞절 과정을 사용한다.
- `ts`에 각 연령별로 생존곡선을 평가한다.

`ss_seq`는 평가된 생존곡선 시퀀스다. `PercentileRows`는 이 시퀀스를 받아 5번째와 95번째 백분위수를 계산하고, 생존곡선 90% 신뢰구간을 반환한다.

그림 13.3에 앞절에서 추정한 생존함수와 함께 결과가 나와 있다. 신뢰구간은 추정곡선과 달리 표집 가중치(sampling weight)를 고려한다. 둘 사이에 불일치는 표집 가중치가 추정값에 상당한 효과가 있음을 나타낸다—이 사실을 유념해야 한다.

제 8 절 코호트 효과 (Cohort effects)

생존분석 도전중의 하나는 추정 곡선의 다른 부분이 응답자의 다른 집단에 기반한다는 것이다. 시점 `t`에 곡선 부분은 인터뷰 당시에 적어도 응답자 연령이 적어도 `t`인 응답자에 기반한다. 그래서, 곡선의 가장 왼쪽 부분은 모든 응답자로부터 데이터가 포함되어 있고, 가장 오른쪽에는 가장 나이든 응답자만 포함된다.

만약 응답자의 관련 특성이 시간에 따라 변화하지 않는다면, 문제가 되지 않고 좋다. 하지만, 이 경우에 다른 세대에 태어난 여성에 대해서 혼인 패턴은 다를 것 같다. 출생별 10년을 단위로 응답자를 집단화해서 효과를 조사할 수있다. 출생 혹은 비슷한 사건으로 정의되는 이와 같은 집단을 **코호트(cohorts)**라고 부른다. 그리고, 집단 간 차이를 **코호트 효과 (cohort effects)**라고 부른다.

NSFG 혼인 데이터에서 코호트 효과를 조사하기 위해서, 이책 전체적으로 사용된 2002년 조사 사이클 6 데이터; 11절에서 사용된 2006–2010 조사 사이클 6 데이터; 1995년 사이클 5 데이터를 수집했다. 모두 합쳐 데이터셋에는 30,769 응답자가 있다.

```
resp5 = ReadFemResp1995()
resp6 = ReadFemResp2002()
resp7 = ReadFemResp2010()
resps = [resp5, resp6, resp7]
```

resp 각 데이터프레임에 대해서, cmbirth를 사용해서 각 응답자에 대한 출생 십년을 계산한다.

```
month0 = pandas.to_datetime('1899-12-15')
dates = [month0 + pandas.DateOffset(months=cm)
         for cm in resp.cmbirth]
resp['decade'] = (pandas.DatetimeIndex(dates).year - 1900) // 10
```

cmbirth은 1899년 12월 이후 정수형 개월수로 부호화된다; month0는 Timestamp 객체로 날짜를 표현한다. 각 출생일에 대해 DateOffset를 인스턴스화하고, 세기-월을 포함하고 month0에 더한다; 결과는 Timestamps 시퀀스로 DateTimeIndex로 전환된다. 마지막으로 year를 추출하고 십년단위로 계산한다.

표집 가중치를 고려하고, 표집오차 때문에 변동성을 보여주기 위해, 데이터를 재표본추출하고, 십년 단위로 응답자를 집단화하고, 생존곡선을 플롯으로 그린다.

```
for i in range(iters):
    samples = [thinkstats2.ResampleRowsWeighted(resp)
               for resp in resps]
    sample = pandas.concat(samples, ignore_index=True)
    groups = sample.groupby('decade')

    EstimateSurvivalByDecade(groups, alpha=0.2)
```

세개 NSFG 시이클 데이터는 서로 다른 표집 가중치를 사용한다. 그래서, 개별적으로 재표본추출하고 나서, concat를 사용해서, 하나의 데이터프레임으로 병합한다. 모수 ignore_index는 concat에게 인덱스로 응답자를 매칭하지 못하게 한다; 대신에 0에서 30768까지 새로운 인덱스를 생성한다.

그림 13.4: Survival functions for respondents born during different decades.

`EstimateSurvivalByDecade`는 각 코호트에 대해서 생존곡선을 플롯으로 그린다.

```
def EstimateSurvivalByDecade(resp):
    for name, group in groups:
        hf, sf = EstimateSurvival(group)
        thinkplot.Plot(sf)
```

그림 13.4에 결과가 나와있다. 패턴 몇개가 눈에 보인다.

- 50년대 여성이 가장 일찍 결혼했고, 연속 코호트 혼인은 점점 늦어지고, 적어도 30대 연령까지 그렇다.
- 60년에 태어난 여성에는 놀라운 패턴이 있다. 25세 이전에 그전 세대보다 혼인 속도가 더 늦었다. 25세 이후에는 혼인 속도가 더 빠르다. 32세경에는 50년대 코호트를 따라잡고, 44세경에는 상당히 더 결혼상태에 있을 것 같다.

  60년대 태어난 여성은 1985년에서 1995년 사이 25세를 넘어선다. 앞서 언급한 *뉴스위크 (Newsweek)* 기사가 1986년에 보도된 것을 기억하면, 이 기사가 결혼붐에 단초를 제공했다고 상상할 마음이 생긴다. 이런 설명이 너무 쉬운 것이지만, 기사와 기사에 대응이 이 코호트 행동에 영향을 주었다는 기분을 나타낸다는 것도 가능하다.
- 70년대 패턴도 비슷하다. 25세 이전에 혼인하는 것이 그 전세대만 못하다. 하지만, 35세경에 이전 두세대 코호트를 모두 따라잡는다.
- 80년생 여성은 25세전에 훨씬 덜 혼인할 것 같다. 이후에 일어난 것은 명분명하지 않다; NSFG 다음 사이틀 데이터를 기다려야만 한다.

그동안 예측도 할 수 있다.

제 9 절 외삽법 (Extrapolation)

70년대생 코호트 생존곡선은 약 38세에서 끝난다; 80년대생 코호트는 약 28세에서 끝나고, 90년대생 코호트는 거의 어떤 자료도 없다.

이전 코호트에서 데이터를 "빌려옴(borrowing)"으로써 이들 곡선을 외삽(extrapolate)할 수 있다. HazardFunction에는 `Extend` 메쏘드를 제공하는데, 또다른 더 긴 HazardFunction에서 꼬리를 복사한다.

그림 13.5: Survival functions for respondents born during different decades, with predictions for the later cohorts.

```
# class HazardFunction

    def Extend(self, other):
        last = self.series.index[-1]
        more = other.series[other.series.index > last]
        self.series = pandas.concat([self.series, more])
```

2절에서 살펴봤듯이, HazardFunction은 t에서 $\lambda(t)$로 매핑하는 시리즈를 담고 있다. `Extend`는 `self.series`에 마지막 인덱스인 `last`를 찾고, `last` 다음에 오는 값을 `other`에서 선택하고, `self.series`에 덧붙인다.

이제 각 코호트에 대해서 이전 것에서 가져온 값을 사용해서 HazardFunction을 연장할 수 있다.

```
def PlotPredictionsByDecade(groups):
    hfs = []
    for name, group in groups:
        hf, sf = EstimateSurvival(group)
        hfs.append(hf)

    thinkplot.PrePlot(len(hfs))
    for i, hf in enumerate(hfs):
        if i > 0:
            hf.Extend(hfs[i-1])
        sf = hf.MakeSurvival()
        thinkplot.Plot(sf)
```

`groups`는 출생을 십년 단위로 나눈 응답자를 갖는 GroupBy 객체다. 첫번째 루프가 각 집단에 대한 HazardFunction을 계산한다.

두번째 루프가 이전 세대로부터 값(이전 집단으로부터 값을 포함할 수 있다)으로 각 HazardFunction를 연장한다. 그리고 나서, 각 HazardFunction을 SurvivalFunction로 전환하고 플롯을 그린다.

그림 13.5에 결과가 나와있다; 50년대생 코호트를 제거해서 예측값을 좀더 가시적으로 만들었다. 이 결과가 시사하는 바는, 40세경에 가장 최신 코호트는 60대 코호트에 수렴하는데, 결코 혼인하지 않는 비율은 20% 보다 적다.

제 10 절 기대 잔존 수명

생존곡선이 주어지면, 현재 연령 함수로 기대잔존수명을 계산할 수 있다. 예를 들어, 1절에서 임신기간 생존함수가 주어지면 출산까지 예상시간을 계산할 수 있다.

첫단계는 수명 PMF를 추출한다. SurvivalFunction가 이를 수행하는 메쏘드를 제공한다.

```
# class SurvivalFunction

    def MakePmf(self, filler=None):
        pmf = thinkstats2.Pmf()
        for val, prob in self.cdf.Items():
            pmf.Set(val, prob)

        cutoff = self.cdf.ps[-1]
        if filler is not None:
            pmf[filler] = 1-cutoff

        return pmf
```

SurvivalFunction는 수명 Cdf를 담고 있다는 것을 기억하라. 루프가 Cdf에서 Pmf로 값과 확률을 복사한다.

cutoff는 Cdf에 가장 높은 확률로, 만약 Cdf가 완전하다면 1이고, 그렇지 않다면 1보다 작다. 만약 Cdf가 불완전하다면, 완료하기 위해 제공되는 값 filler를 꽂아 넣는다,

임신기간 Cdf는 완전하기 때문에, 아직은 이러한 세부사항까지 걱정할 필요는 없다.

다음 단계는 기대잔존수명을 계산하는데, 여기서 "기대(expected)"는 평균을 의미한다. SurvivalFunction은 또한 이것을 수행하는 메쏘드를 제공한다.

```
# class SurvivalFunction

    def RemainingLifetime(self, filler=None, func=thinkstats2.Pmf.Mean):
        pmf = self.MakePmf(filler=filler)
        d = {}
        for t in sorted(pmf.Values())[:-1]:
            pmf[t] = 0
            pmf.Normalize()
            d[t] = func(pmf) - t

        return pandas.Series(d)
```

그림 13.6: Expected remaining lifetime for pregnancy length (left) and years until first marriage (right).

`RemainingLifetime`는 인자로 `MakePmf`에 전달되는 `filler`와, 잔존수명 분포를 요약하는데 사용되는 함수 `func`을 인자로 받는다.

`pmf`는 SurvivalFunction에서 추출된 수명 Pmf다. `d`는 현재 연령 `t`에서 기대잔존수명으로 매핑 결과를 담고 있다.

루프가 Pmf에 값을 반복 돌린다. `t` 각 값에 대해, 수명이 `t`를 초과한 것을 둔 상태에서, 수명 조건부분포를 계산한다. Pmf에서 값을 한번에 하나씩 제거하고, 남은 값을 다시 정규화함으로써 이것을 수행한다.

그리고 나서, `func`를 사용해서, 조건부분포를 요약한다. 상기 예제에서, 기간이 `t`를 초과했을 때 결과는 평균임신기간이 된다. `t`를 빼서, 평균잔존임신기간을 얻는다.

그림 13.6 (왼편)에 현재 지속기간의 함수로 기대잔존임신기간이 나와 있다. 예를 들어, 0주차에는 기대잔존기간이 약 34주가 된다. 이것이 만삭(39주)보다 짧은데 이유는 임신 초기에 임신중절이 평균을 낮추기 때문이다.

임신초기 기간에 곡선이 천천히 떨어진다. 13주차 뒤에는 기대잔존수명이 25주로 단지 9주만 떨어진다. 그 후에, 곡선은 주마다 약 1주씩 더 빨리 떨어진다.

37주에서 42주 사이, 곡선은 1주와 2주 사이 수평을 유지한다. 이 기간동안 아무 시점이나 평균잔존기대수명은 같다; 매주 지나감에 따라 목표가 더이상 가까와지지 않는다. 이와 같은 특성을 가진 과정을 **무기억성(memoryless)**이라고 부른다. 이유는 과거가 예측에 아무런 효과가 없기 때문이다. ㅇ 행동이 산부인과 간호사의 격노하는 만트라의 수학적 기반이 된다: “곧 지금이라도 (any day now)”

그림 13.6 (오른쪽)에는 연령의 함수로 초혼까지 중위수 잔존시간이 나와있다. 11살 소녀에게 초혼까지 중위수 시간은 약 14년이다. 곡선은 중위수 잔존시간이 약 7년일때 22세까지 감소한다. 그 후에 다시 올라간다: 나이 30에 시작한 연령 14년으로 다시 증가한다.

이 데이터에 기반해서, 젊은 여성은 감소하는 잔존 “수명”을 갖는다. 이와 같은 성질을 가진 기계부품을 **NBUE**(“new better than used in expectation")로 부른다. 새부품이 더 오래 갈 것으로 예상된다는 의미다.

22세 이상되는 여성은 초혼까지 증가하는 잔존시간을 갖는다. 이와 같은 성질을 갖는 부품을 **UBNE**(“used better than new in expectation”)로 부른다. 즉, 부품이 오래될수록, 더 오래 갈 것으로 예상된다. 신생아와 암환자가 또한 UBNE다; 더 오래 살수록 이들의 기대수명은 증가한다.

이 예제에서 Cdf가 불와전해서, 평균대신에 중위수를 계산했다; 생존곡선이 약 20% 응답자가 44세 이전에 결혼하지 않을 것으로 추산한다. 이 여성에 대한 초혼 연령은 알려져있지 않고, 존재하지 않을지도 모른다. 그래서 평균을 계산할 수 없다.

미지값(unknown value)을 무한대를 나타내는 특수값 `np.inf`로 바꿔서 미지값을 처리한다. 이렇게 하면 모든 연령에 대해서 평균이 무한대가 된다. 하지만, 잔존수명 50% 이상이 유한(연령 30세까지 사실이다)하기만 하면 중위수는 잘 정의된다. 이후에 유의미한 기대잔존수명을 정의하는 것은 어렵다.

다음에 이들 함수를 계산하고 플롯으로 그리는 코드가 있다.

```
rem_life1 = sf1.RemainingLifetime()
thinkplot.Plot(rem_life1)

func = lambda pmf: pmf.Percentile(50)
rem_life2 = sf2.RemainingLifetime(filler=np.inf, func=func)
thinkplot.Plot(rem_life2)
```

`sf1`는 임신 기간에 대한 생존함수다; 이 경우에 `RemainingLifetime`에 초기설정값을 사용할 수 있다.

`sf2`는 초혼 연령에 대한 생존함수다; `func`는 Pmf를 인자로 받아 중위수(50번째 백분위수)를 계산하는 함수다.

제 11 절 연습문제

My solution to this exercise is in `chap13soln.py`.

Exercise 13.1 In NSFG Cycles 6 and 7, the variable `cmdivorcx` contains the date of divorce for the respondent's first marriage, if applicable, encoded in century-months.

Compute the duration of marriages that have ended in divorce, and the duration, so far, of marriages that are ongoing. Estimate the hazard and survival function for the duration of marriage.

Use resampling to take into account sampling weights, and plot data from several resamples to visualize sampling error.

Consider dividing the respondents into groups by decade of birth, and possibly by age at first marriage.

제 12 절 용어 사전

- 생존분석 (survival analysis): 수명, 좀더 일반적으로 사건이 일어나기까지 시간을 기술하고 예측하는 방법론 집합.
- 생존곡선 (survival curve): 시점 t에 t를 지나 생존 확률로 매핑하는 함수.
- 위험함수 (hazard function): 시점 t 에 t 시점까지 생존한 사람이 t 시점에 사망한 비율을 매핑하는 함수.
- 캐플란-마이어 추정 (Kaplan-Meier estimation): 위험함수와 생존함수를 추정하는 알고리즘.
- 코호트 (cohort): 특정 시간 구간에 생년월일 같은 사건으로 정의된 개체 집단.
- 코호트 효과 (cohort effect): 코호트 사이 차이.
- NBUE: 기대잔존수명 성질, "새 것이 예상하기에 오래된 것보다 좋은 것 (New better than used in expectation)"
- UBNE: 기대잔존수명 성질, "오래된 부품이 예상하기에 새 것보다 좋은 것 (Used better than new in expectation)"

제 14 장

해석적 방법 (Analytic methods)

이책은 모의시험이나 재표본추출같은 수치해석적 방법(computational methods)에 집중했지만, 해결한 문제중 일부는 훨씬더 빠르게 해결할 수 있는 해석적 해(analytic solution)를 가지고 있다.

이번 장에서 해석적 방법 일부를 제시하고, 어떻게 동작하는지 설명한다. 이장말미에 탐색적 데이터 분석을 위해서 수치해석적 방법과 해석적 방법 통합에 대한 제언을 한다.

이번 장에서 사용되는 코드는 `normal.py`에 있다. 코드를 다운로드하고 작업하는 것에 대한 정보는 2을 참조한다.

제 1 절 정규분포

동기부여를 위한 사례로, 3 절에 있던 문제를 검토하자.

> 야생동물 보호구에서 고릴라를 연구하는 과학자가 있다고 가정하자. 고릴라 9마리 체중을 재서, 표본평균 $\bar{x} = 90$ kg와 표본 표준편차 $S = 7.5$ kg을 얻었다. 만약 $\bar{x}$를 모집단 평균으로 추정한다면, 추정값의 표준오차는 얼마나 될까?

이 질문에 대답하기 위해서, $\bar{x}$ 표집 분포가 필요하다. 3절에서, (고릴라 9마리 체중을 재는) 실험을 모의시험함으로써 분포를 근사했고, 각 모의시험 실험에 대해서 $\bar{x}$를 계산하고, 추정값 분포를 축적했다.

결과는 표집분포를 근사했다. 그리고 나서, 표집분포를 사용해서 표준오차와 신뢰구간을 계산했다.

1. 표집분포 표준편차는 추정값의 표준오차다; 이 경우 약 2.5 kg이 된다.

2. 5번째와 95번째 백분위수 표집분포 구간이 90% 신뢰구간이 된다. 만약 실험을 많이 수행한다면, 추정값이 90% 신뢰구간에 떨어질 것으로 예상한다. 이 경우 90% CI는 $(86, 94)$ kg이 된다.

이제 해석적으로 동일한 계산을 수행한다. 성인 여성 고릴라 체중이 대략 정규분포한다는 사실을 이용한다. 정규분포는 분석을 용이하게 하는 성질을 두개 갖고 있다; 선형 변환과 덧셈에 "닫혀(closed)"있다. 이것이 의미하는 바를 설명하기 위해서, 약간의 표기가 필요하다.

어떤 양(quantity)의 분포 X가 모수 μ와 σ을 갖는 정규분포라면, 다음과 같이 표현할 수 있다.

$$X \sim \mathcal{N}\left(\mu, \sigma^2\right)$$

여기서, 기호 $\sim$ 는 "분포한다(is distributed)"를 의미하고, 스크립트 문자 $\mathcal{N}$는 "정규(normal)"를 나타낸다.

X의 선형변환은 $X' = aX + b$와 같은 것으로, 여기서 a와 b는 실수다. 만약, X'이 X와 같은 모임(족, family)이면, 분포 모임이 선형변환에 닫혀있다. 정규분포는 이 성질을 갖고 있다; 만약 $X \sim \mathcal{N}\left(\mu, \sigma^2\right)$이면,

$$X' \sim \mathcal{N}\left(a\mu + b, a^2\sigma^2\right) \tag{1}$$

정규분포는 또한 덧셈에도 닫혀있다. 만약 $Z = X + Y$ 이고, $X \sim \mathcal{N}\left(\mu_X, \sigma_X^2\right)$, $Y \sim \mathcal{N}\left(\mu_Y, \sigma_Y^2\right)$이면,

$$Z \sim \mathcal{N}\left(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2\right) \tag{2}$$

특별한 경우 $Z = X + X$이면, 다음을 만족한다.

$$Z \sim \mathcal{N}\left(2\mu_X, 2\sigma_X^2\right)$$

그리고, 일반적으로 X에서 n개 값을 추출한다면, 다음을 갖게 된다.

$$Z \sim \mathcal{N}\left(n\mu_X, n\sigma_X^2\right) \tag{3}$$

제 2 절 표집분포

이제 $\bar{x}$ 표집분포를 계산하는데 필요한 모든 것을 갖췄다. $\bar{x}$를 계산하는데 n개 고릴라 체중을 재고, 더해서 전체 체중값을 얻고 나서, n으로 나눈다는 것을 기억하라.

고릴라 체중 X 분포가 근사적으로 정규분포라고 가정한다.

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

만약 n개 고릴라 체중을 잰다면, 전체체중 Y는 3번 방정식을 사용해서 다음과 같이 분포한다.

$$Y \sim \mathcal{N}(n\mu, n\sigma^2)$$

그리고 만약 n으로 나눈다면, $a = 1/n$으로 1번 방정식을 사용해서 표본평균 Z는 다음과 같이 분포한다.

$$Z \sim \mathcal{N}(\mu, \sigma^2/n)$$

Z 분포는 $\bar{x}$의 표집분포가 된다. Z 평균은 μ으로, $\bar{x}$가 μ의 불편 추정값이 된다는 것을 보여준다. 표집분포 분산은 σ^2/n이다.

그래서, 표집분포 표준편차, 즉 추정값의 표준오차는 $\sigma/\sqrt{n}$이 된다. 예제에서, σ는 7.5 kg 이고, n은 9가 되서, 표준오차는 2.5 kg가 된다. 결과는 모의시험으로 추정한 것과 일치하지만, 계산하기는 훨씬 더 빠르다!

또한, 표집분포를 사용해서 신뢰구간도 계산할 수 있다. $\bar{x}$에 대한 90% 신뢰구간은 Z의 5번째와 95번째 백분위수 간격이다. Z이 정규분포를 따르기 때문에, 역 CDF(inverse CDF)를 평가해서 백분위수를 계산할 수 있다.

정규분포 CDF와 역 CDF에 대한 닫힌 형태(closed form)가 존재하지 않지만, 빠른 수치해석 방법이 존재하고 2절에서 살펴본 바와 같이 SciPy에 구현되어 있다. `thinkstats2`에 SciPy 함수를 좀더 쉽게 사용할 수 있게 하는 랩퍼(wrapper) 함수가 제공된다.

```
def EvalNormalCdfInverse(p, mu=0, sigma=1):
    return scipy.stats.norm.ppf(p, loc=mu, scale=sigma)
```

확률 `p`가 주어졌을 대, 모수 `mu`와 `sigma`를 갖는 정규분포에서 대응되는 백분위수를 반환한다. $\bar{x}$의 90% 신뢰구간에 대해서, 다음과 같이 5번째와 95번째 백분위수를 계산한다.

```
>>> thinkstats2.EvalNormalCdfInverse(0.05, mu=90, sigma=2.5)
85.888

>>> thinkstats2.EvalNormalCdfInverse(0.95, mu=90, sigma=2.5)
94.112
```

그래서, 만약 실험을 많이 반복한다면, 추정값 $\bar{x}$가 약 90%로 범위 $(85.9, 94.1)$에 떨어질 것이다. 다시 이값은 모의시험으로 얻은 결과와 일치한다.

제 3 절 정규분포 표현하기

상기 계산을 좀더 쉽게하기 위해서, Normal 클래스를 정의하는데 정규분포를 표현하고 앞절 방정식을 부호화한다. 다음에 코드가 있다.

```
class Normal(object):

    def __init__(self, mu, sigma2):
        self.mu = mu
        self.sigma2 = sigma2

    def __str__(self):
        return 'N(%g, %g)' % (self.mu, self.sigma2)
```

그래서, 고릴라 체중 분포를 표현하는 Normal을 인스턴스화할 수 있다.

```
>>> dist = Normal(90, 7.5**2)
>>> dist
N(90, 56.25)
```

Normal 클래스에는 Sum 메쏘드가 제공되는데, 표본크기 n을 인다로 받아, 방정식 3을 사용해서 n개 값의 합계 분포를 반환한다.

```
    def Sum(self, n):
        return Normal(n * self.mu, n * self.sigma2)
```

Normal은 또한 방정식 1을 사용해서 곱셈과 나눗셈도 수행한다.

```
    def __mul__(self, factor):
        return Normal(factor * self.mu, factor**2 * self.sigma2)

    def __div__(self, divisor):
        return 1 / divisor * self
```

그래서, 표본크기 9를 가지고 평균의 표집분포를 계산할 수 있다.

```
>>> dist_xbar = dist.Sum(9) / 9
>>> dist_xbar.sigma
2.5
```

으로 앞절에서 살펴보았듯이, 표집분포 표준편차는 2.5 kg이다. 마지막으로, Normal 클래스에는 Percentile 메쏘드가 있어서 신뢰구간을 계산하는데 사용할 수 있다.

```
>>> dist_xbar.Percentile(5), dist_xbar.Percentile(95)
85.888 94.113
```

그리고, 이것은 앞에서 얻은 답과 같다. 나중에 Normal 클래스를 다시 사용할 것이다. 하지만, 더 진도를 나가기 전에, 분석 한가지가 더 필요하다.

제 4 절 중심극한정리 (Central limit theorem)

앞절에서 살펴봤듯이, 만약 정규분포에서 추출한 값을 더하면, 합 분포도 정규분포다. 다른 대부분의 분포는 이런 성질을 갖지 못하다; 만약 다른 분포에서 추출한 값을 더한다면, 합은 일반적으로 해석적 분포를 갖지 못한다.

하지만, 거의 모든 분포에서 n 값을 더하면, 합 분포가 n이 증가함에 따라 정규분포로 수렴한다.

좀더 구체적으로, 만약 값들의 분포가 평균과 표준편차 μ와 σ를 갖는다면, 합 분포는 근사적으로 $\mathcal{N}(n\mu, n\sigma^2)$이 된다.

이 결과가 중심극한정리(Central Limit Theorem, CLT)이다. 통계분석을 위한 가장 유용한 도구 중 하나다. 하지만, 몇가지 주의점이 있다.

- 값들이 독립적으로 추출되어야 한다. 만약 상관된다면, CLT을 적용할 수 없다.(설사 이것이 실무에서 문제가 결코되지는 않을지만)
- 값들이 동일한 분포에서 나와야 한다 (설사 이런 요구사항은 완화될 수도 있지만)
- 유한 평균과 분산을 갖는 분포에서 값들이 나와야 한다. 그래서 대부분 파레토 분포는 해당되지 않는다.
- 수렴 속도는 분포 왜도에 의존한다. 지수분포에서 나온 값들의 합은 작은 n에 대해서 수렴한다. 로그 정규분포에서 나온 값들의 합은 더 커다란 크기가 필요하다.

중심극한정리는 자연 세계에 정규분포가 널리 퍼짐을 설명한다. 생물체의 많은 특징이 유전적 환경적 요인에 영향을 받는데, 이들 효과는 가법(additive)적이다. 측정하는 특징은 많은 작은 효과의 합이다. 그래서, 분포가 정규분포화 되는 경향이 있다.

제 5 절 CLT 검정

중심극한정리가 어떻게 동작하는지와 언제 동작하지 않는지를 살펴보기 위새허, 실험을 몇가지 시도해보자. 먼저, 지수분포로 시도해보자.

```
def MakeExpoSamples(beta=2.0, iters=1000):
    samples = []
    for n in [1, 10, 100]:
        sample = [np.sum(np.random.exponential(beta, n))
                  for _ in range(iters)]
        samples.append((n, sample))
    return samples
```

그림 14.1: Distributions of sums of exponential values (top row) and lognormal values (bottom row).

그림 14.2: Distributions of sums of Pareto values (top row) and correlated exponential values (bottom row).

MakeExpoSamples는 지수분포값 합을 생성한다("지수분포로부터 추출된 값들"을 줄여 "지수분포값"을 사용한다). beta는 분포 모수다; iters는 생성할 합 갯수다.

상기 함수를 설명하기 위해서, 내부에서 시작해서 밖으로 나가면서 마무리한다. np.random.exponential을 매번 호출할 때마다, n개 지수분포값 시퀀스를 얻어서 합을 계산한다. sample은 길이 iters를 갖는 합계 리스트가 된다.

n과 iters를 혼동하기 쉽다: n은 각 합에 대한 항의 갯수다; iters는 합분포를 특성화하기 위해서 계산하는 합 갯수다.

반환되는 값은 (n, sample) 쌍(pair) 리스트다. 각 쌍에 대해 정규확률그림을 만들 수 있다.

```
def NormalPlotSamples(samples, plot=1, ylabel=''):
    for n, sample in samples:
        thinkplot.SubPlot(plot)
        thinkstats2.NormalProbabilityPlot(sample)

        thinkplot.Config(title='n=%d' % n, ylabel=ylabel)
        plot += 1
```

NormalPlotSamples은 MakeExpoSamples에서 리스트 짝(pair)를 인자로 받아 정규확률그림 행을 생성한다.

그림 14.1 (위쪽 행)에 결과가 나와있다. n=1일 때, 합분포는 여전히 지수분포라서 정규확률그림은 직선이 아니다. 하지만, n=10일 때, 합분포는 근사 정규분포가 되고, n=100일 때, 정규분포와 거의 구별되지 않는다.

그림 14.1 (아래 행)에 로그정규분포에 대한 비슷한 결과가 나와 있다. 로그정규분포는 일반적으로 지수분포보다 기울어짐이 더 심해서, 합분포는 수렴하는데 더 오래 걸린다. n=10일 때, 정규확률그림은 거의 직선인 곳이 없다. 하지만, n=100일 때, 근사적으로 정규분포다.

파레토 분포는 로그정규분포보다 더 기울어짐이 심하다. 모수에 따라서, 많은 파레토 분포는 유한 평균과 분산을 갖지 못하다. 결과로, 중심극한정리가 적용되지 않는다. 그림 14.2 (상단 행)에 파레토 값들의 합 분포가 나와 있다. 심지어 n=100일 때, 정규확률그림은 직선에서 거리가 멀다.

또한 CLT는 만약 값들이 상관된다면 적용되지 않는다고 언급했다. 이것을 검정하기 위해서, 지수분포에서 상관된 값들을 생성했다. 상관된 값을 재생하는데 사용된 알고리즘은 (1) 상관된 정규분포 값을 발생시킨다. (2) 정규분포 CDF를 사용해서 값을 균등분포로 변환한다. (3) 역 지수분포 CDF를 사용해서 균등분포 값을 지수분포로 변환한다.

`GenerateCorrelated`는 계열상관 `rho`를 갖는 `n`개 정규분포 값 반복자를 반환한다.

```
def GenerateCorrelated(rho, n):
    x = random.gauss(0, 1)
    yield x

    sigma = math.sqrt(1 - rho**2)
    for _ in range(n-1):
        x = random.gauss(x*rho, sigma)
        yield x
```

첫값은 표준정규분포 값이다. 각 후속값은 이전 값에 의존한다: 만약 이전 값이 `x`, 다음값은 평균 `x*rho`, 분산 `1-rho**2`이다. `random.gauss`는 두번째 인자로 분산이 아닌 표준편차를 받는 것을 주목한다.

`GenerateExpoCorrelated`는 결과 시퀀스를 인자로 받아 지수분포 값으로 변환한다.

```
def GenerateExpoCorrelated(rho, n):
    normal = list(GenerateCorrelated(rho, n))
    uniform = scipy.stats.norm.cdf(normal)
    expo = scipy.stats.expon.ppf(uniform)
    return expo
```

`normal`은 상관된 정규분포 값 리스트다. `uniform`은 0과 1사이 균등분포 값 시퀀스다. `expo`는 상관된 지수분포 값 시퀀스다. `ppf`는 "퍼센트점 함수 (percent point function)"의 약어로 역CDF에 대한 또다른 이름이다.

그림 14.2 (하단 행)에 `rho=0.9` 상관을 갖는 지수분포 값 합의 분포가 나와 있다. 상관된 경우 수렴속도가 느리다; 그럼에도 불구, `n=100`일 때, 정규확률그림은 거의 직선이다. 그래서 값들이 상관되었을 때, CLT가 엄격하게 적용되지는 않지만, 적당한 상관은 실무에서 결코 문제가 되지 않는다.

중심극한정리가 어떻게 동작하는지와 중심극한정리가 동작하지 않을 때 무슨일이 발생하는지 보여주기 위해서 이들 실험이 고안되었다. 이제 중심극한정리를 어떻게 사용하는지 살펴보자.

제 6 절 CLT 적용하기

왜 중심극한정리가 유용한지 살펴보기 위해서, 3절에 예제로 돌아가자: 첫째아이와 첫째가 아닌 아이에 대한 평균 임신기간 외관효과 검정. 앞에서 살펴봤듯이, 외관 차이는 약 0.078주다.

```
>>> live, firsts, others = first.MakeFrames()
>>> delta = firsts.prglngth.mean() - others.prglngth.mean()
0.078
```

가설검정 로직을 기억하라: p-값을 계산하는데, 귀무가설 아래에서 관측 차이 확률이다; 만약 작다면, 관측 차이는 우연에 의한 것이 아닐 것으로 결론낸다.

이 예제애서, 귀무가설은 임신기간 분포가 첫째 아이와 첫째가 아닌 아이들에 대해서 같다. 그래서, 다음과 같이 평균 표집분포를 계산할 수 있다.

```
    dist1 = SamplingDistMean(live.prglngth, len(firsts))
    dist2 = SamplingDistMean(live.prglngth, len(others))
```

두 표집분포는 동일 모집단에 기반하는데 모든 정상 출산을 합동(pool)한 것이다. SamplingDistMean는 이 값 시퀀스와 표본크기를 인자로 받아 표집분포를 표현하는 Normal 객체를 반환한다.

```
def SamplingDistMean(data, n):
    mean, var = data.mean(), data.var()
    dist = Normal(mean, var)
    return dist.Sum(n) / n
```

mean와 var는 data 평균과 분산이다. 정규분포 dist로 데이터 분포를 근사한다.

이 예제애서 데이터는 정규분포되어 있지 않다. 그래서 이러한 근사가 그다지 좋지 않다. 하지만 그리고 나서 dist.Sum(n) / n을 계산하는데 n개 값 평균의 표집분포다. 데이터가 정규분포되어 있지 않지만, 평균 표집분포는 중심극한정리에 의해서 정규분포된다.

다음, 평균에 차이 표집분포를 계산한다. Normal 클래스는 방정식 2를 사용해서 뺄셈을 어떻게 수행하는지 알고 있다.

```
    def __sub__(self, other):
        return Normal(self.mu - other.mu,
                      self.sigma2 + other.sigma2)
```

그래서, 다음과 같이 차이 표집분포를 계산할 수 있다.

```
>>> dist = dist1 - dist2
N(0, 0.0032)
```

평균은 0 인데, 일리가 있다. 왜냐하면 평균적으로 동일 분포에서 추출된 두 표본은 같은 평균을 갖을 것으로 예상하기 때문이다. 표집분포 분산은 0.0032가 된다.

Normal 클래스는 Prob메쏘드를 제공하는데 정규분포 CDF를 평가한다. Prob를 사용해서 귀무가설 아래에서 차이 확률을 delta 만큼 계산할 수 있다.

```
>>> 1 - dist.Prob(delta)
0.084
```

의미하는 바는 단측검정에 대한 p-값이 0.084다. 양측검정에 대해서 또한 다음과 같이 계산한다.

```
>>> dist.Prob(-delta)
0.084
```

정규분포는 대칭이기 때문에 동일하다. 꼬리 합은 0.168로, 3절 추정값과 일치한다; 값이 0.17이였다.

제 7 절 상관검정 (Correlation test)

5절에서 출생 체중과 산모 연령 사이 상관을 계산한느데 ㅅ누열 검정(permutation test)을 사용했고, p-값이 0.001보다 작아 통계적으로 유의적이라는 것을 발견했다.

이제 해석적으로 같은 것을 수행할 수 있다. 메쏘드는 다음 수학적 결과에 기반한다: 정규분포를 따르고 상관되지 않는 두변수가 주어지고, 만약 n개 크기 표본을 생성하고, 피어슨 상관 r을 계산하고 나서, 변환된 상관을 계산하면,

$$t = r\sqrt{\frac{n-2}{1-r^2}}$$

t분포는 모수 $n-2$을 갖는 스튜던트 t-분포다. t-분포는 해석적 분포다; CDF는 감마 함수를 사용해서 효율적으로 계산될 수 있다.

이 결과를 사용해서 귀무가설 아래에서 상관 표집분포를 계산할 수 있다; 즉, 만약 상관되지 않은 정규분포 값 시퀀스를 생성한다면, 두 시퀀스 상관 분포는 무엇일까요? StudentCdf가 표본크기 n을 인자로 받아 상관 표집분포를 반환한다.

```
def StudentCdf(n):
    ts = np.linspace(-3, 3, 101)
    ps = scipy.stats.t.cdf(ts, df=n-2)
    rs = ts / np.sqrt(n - 2 + ts**2)
    return thinkstats2.Cdf(rs, ps)
```

ts는 t에 대한 값 넘파이(NumPy) 배열, 변환 상관이다. ps는 대응되는 확률을 담고 있는데, SciPy로 구현된 스튜던트 t-분포 CDF를 사용해서 계산된다. t-분포 모수, df는 "자유도 (degrees of freedom)"의 두문자(acronym)다. 이 용어를 설명하지는 않지만, 웹사이트에서 자세한 내용 참조한다. http://en.wikipedia.org/wiki/Degrees_of_freedom_(statistics).

그림 14.3: Sampling distribution of correlations for uncorrelated normal variables.

그림 14.4: Sampling distribution of chi-squared statistics for a fair six-sided die.

ts로부터 상관계수 rs를 얻기 위해서, 역변환을 적용한다.

$$r = t/\sqrt{n-2+t^2}$$

결과는 귀무가설 아래에서 r의 표집분포가 된다. 그림 14.3에 재표본추출로 5절에서 생성한 분포를 따라 이 분포를 함께 보여준다. 두 분포가 거의 동일(identical)하다. 실제 분포가 정규분포는 아니지만, 피어슨 상관계수는 표본 평균과 분산에 기반한다. 중심극한정리에 의해서, 설사 데이터는 그렇지 않지만, 적률기반 통계량은 정규분포한다.

그림 14.3로부터, 만약 변수가 실제로 상관되지 않는다면, 관측 상관 0.07은 일어날 것 같지 않다는 것을 볼 수 있다. 해석 분포를 사용해서, 얼마나 그럴것 같지 않은지를 계산할 수 있다.

```
t = r * math.sqrt((n-2) / (1-r))
p_value = 1 - scipy.stats.t.cdf(t, df=n-2)
```

r=0.07에 상응하는 t 값을 계산한다. 그리고 나서 t에 t-분포를 평가한다. 결과는 6.4e-12가 나온다. 이 예제는 해석적 방법의 장점을 시연한다: 매우 작은 p-값을 계산할 수 있다. 하지만, 실무에서 보통 문제가 되는 않는다.

제 8 절 카이제곱 검정 (Chi-squared test)

7절에서 카이제곱 통계량을 사용해서 주사위가 비뚤었는지 검정했다. 카이제곱 통계량은 표(table)에 기대값과 정규화된 총 편차를 측정한다:

$$\chi^2 = \sum_i \frac{(O_i - E_i)^2}{E_i}$$

카이제곱 통계량이 널리 사용되는 이유는 귀무가설 아래에서 표집분포가 해석적(analytic)이기 때문이다; 놀라운 우연의 일치[1]로, 카이제곱 분포로 불린다. t-분포처럼 카이제곱 CDF는 감마 함수를 사용해서 효율적으로 계산될 수 있다.

SciPy는 카이제곱 분포 기능을 제공한다. 이것을 사용해서 카이제곱 통계량 표집분포를 계산한다.

[1] 정말.(Not really.)

```
def ChiSquaredCdf(n):
    xs = np.linspace(0, 25, 101)
    ps = scipy.stats.chi2.cdf(xs, df=n-1)
    return thinkstats2.Cdf(xs, ps)
```

그림 14.4에 재표본추출로 얻은 분포와 함께 해석적 결과가 함께 나와 있다. 둘다 모두 매우 비슷한데, 특히 꼬리부분에 그렇다. 꼬리부분은 통상 가장 관심을 두는 부분이다.

이분포를 사용해서 관측 검정 통계량, chi2의 p-값을 계산할 수 있다:

```
    p_value = 1 - scipy.stats.chi2.cdf(chi2, df=n-1)
```

결과는 0.041로 7절 결과와 일치한다.

카이제곱 분포 모수는 다시 "자유도(degrees of freedom)"가 된다. 이 경우 맞는 모수는 n-1ㅇ 되는데, 여기서 n은 표(table) 크기로 6이다. 이 모수를 선택한 것이 다소 까다로울 수 있다; 정직하게 해석결과와 재표본추출 결과를 비교하기 위해서, 그림 14.4같은 것을 생성할 때가지 저자는 그것이 맞는지 확신할 수 없었다.

제 9 절 토의 (Discussion)

이책은 재표본추출과 순열 같은 수치해석적 방법(computational method)에 집중했다. 이 방법이 해석적인 방법에 대해서 몇가지 장점이 있다.

- 수치해석적 방법이 설명하고 이해하기 더 쉽다. 예를 들어, 기초 통계에서 가장 어려운 주제가 가설검정이다. 많은 학생이 p-값이 무엇인지 정말 이해하지 못한다. 9장에서 제시한 접근법—귀무가설을 모의시험하고 검정통계량을 계산하는 것—이 근본개념을 좀더 명확하게 한다고 믿는다.

- 수치해석적 방법이 강건하고 다목적이다. 해석적 방법은 실무에서 지지할 수 없는 가정에 종종 기반한다. 수치해석적 방법은 더 적은 가정을 요구하고 좀더 쉽게 확장과 개조를 할 수 있다.

- 수치해석적 방법은 디버그(debuggable)할 수 있다. 해석적 방법은 종종 블랙박스다: 숫자를 집어 넣으면 결과가 튀어나온다. 하지만 미묘한 실수를 하기 쉽고, 결과가 맞는지 확신하기 어렵다. 그리고 만약 문제가 없다면 문제를 찾기가 어렵다. 수치해석적 방법은 점증 개발 및 테스트(incremental development and testing)를 할 수 있게 해서 결과에 신뢰감을 함양한다.

하지만, 한가지 단점이 있다: 수치해석적 방법은 느릴 수 있다. 장점과 단점을 고려해서, 다음 과정(process)를 추천한다.

1. 탐색과정에서 수치해석적 방법을 사용하라. 만약 만족스러운 답을 찾고 실행시간이 수용가능하면, 멈출 수 있다.

2. 만약 실행시간을 받아들일 수 없다면, 최적화할 기회를 찾아봐라. 해석적 방법을 사용하는 것은 몇가지 최적화 방법중 하나다.

3. 만약 수치해석적 방법을 해석적 방법으로 바꾸는 것이 적절하다면, 수치해석적 방법을 비교 기반으로 사용하라. 왜냐하면 수치해석적 결과와 해석적 결과 사이에 상호 타당성 검증기능을 제공하기 때문이다.

저자가 해결하려고 노력한 방대한 문제에 대해서, 1단계를 지나갈 필요는 없다.

제 10 절 연습 문제

A solution to these exercises is in `chap14soln.py`

Exercise 14.1 In Section 4, we saw that the distribution of adult weights is approximately lognormal. One possible explanation is that the weight a person gains each year is proportional to their current weight. In that case, adult weight is the product of a large number of multiplicative factors:

$$w = w_0 f_1 f_2 ... f_n$$

where w is adult weight, w_0 is birth weight, and f_i is the weight gain factor for year i.

The log of a product is the sum of the logs of the factors:

$$\log w = \log w_0 + \log f_1 + \log f_2 + ... + \log f_n$$

So by the Central Limit Theorem, the distribution of $\log w$ is approximately normal for large n, which implies that the distribution of w is lognormal.

To model this phenomenon, choose a distribution for f that seems reasonable, then generate a sample of adult weights by choosing a random value from the distribution of birth weights, choosing a sequence of factors from the distribution of f, and computing the product. What value of n is needed to converge to a lognormal distribution?

Exercise 14.2 In Section 6 we used the Central Limit Theorem to find the sampling distribution of the difference in means, δ, under the null hypothesis that both samples are drawn from the same population.

We can also use this distribution to find the standard error of the estimate and confidence intervals, but that would only be approximately correct. To be more precise, we should compute the sampling distribution of δ under the alternate hypothesis that the samples are drawn from different populations.

Compute this distribution and use it to calculate the standard error and a 90% confidence interval for the difference in means.

Exercise 14.3 In a recent paper[2], Stein et al. investigate the effects of an intervention intended to mitigate gender-stereotypical task allocation within student engineering teams.

Before and after the intervention, students responded to a survey that asked them to rate their contribution to each aspect of class projects on a 7-point scale.

Before the intervention, male students reported higher scores for the programming aspect of the project than female students; on average men reported a score of 3.57 with standard error 0.28. Women reported 1.91, on average, with standard error 0.32.

Compute the sampling distribution of the gender gap (the difference in means), and test whether it is statistically significant. Because you are given standard errors for the estimated means, you don't need to know the sample size to figure out the sampling distributions.

After the intervention, the gender gap was smaller: the average score for men was 3.44 (SE 0.16); the average score for women was 3.18 (SE 0.16). Again, compute the sampling distribution of the gender gap and test it.

Finally, estimate the change in gender gap; what is the sampling distribution of this change, and is it statistically significant?

[2] "Evidence for the persistent effects of an intervention to mitigate gender-sterotypical task allocation within student engineering teams," Proceedings of the IEEE Frontiers in Education Conference, 2014.