Think Stats

Exploratory Data Analysis in Python

Version 2.0.25

Think Stats

Exploratory Data Analysis in Python

Version 2.0.25

Allen B. Downey
Victor KC Lee

Green Tea Press

Needham, Massachusetts

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`.

Preface

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.

0.1 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`.

0.2 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.

Contents

Chapter 1

Exploratory data analysis

The thesis of this book is that data combined with practical methods can answer questions and guide decisions under uncertainty.

As an example, I present a case study motivated by a question I heard when my wife and I were expecting our first child: do first babies tend to arrive late?

If you Google this question, you will find plenty of discussion. Some people claim it's true, others say it's a myth, and some people say it's the other way around: first babies come early.

In many of these discussions, people provide data to support their claims. I found many examples like these:

> "My two friends that have given birth recently to their first babies, BOTH went almost 2 weeks overdue before going into labour or being induced."
>
> "My first one came 2 weeks late and now I think the second one is going to come out two weeks early!!"
>
> "I don't think that can be true because my sister was my mother's first and she was early, as with many of my cousins."

Reports like these are called **anecdotal evidence** because they are based on data that is unpublished and usually personal. In casual conversation, there is nothing wrong with anecdotes, so I don't mean to pick on the people I quoted.

But we might want evidence that is more persuasive and an answer that is more reliable. By those standards, anecdotal evidence usually fails, because:

- Small number of observations: If pregnancy length is longer for first babies, the difference is probably small compared to natural variation. In that case, we might have to compare a large number of pregnancies to be sure that a difference exists.
- Selection bias: People who join a discussion of this question might be interested because their first babies were late. In that case the process of selecting data would bias the results.
- Confirmation bias: People who believe the claim might be more likely to contribute examples that confirm it. People who doubt the claim are more likely to cite counterexamples.
- Inaccuracy: Anecdotes are often personal stories, and often misremembered, misrepresented, repeated inaccurately, etc.

So how can we do better?

1.1 A statistical approach

To address the limitations of anecdotes, we will use the tools of statistics, which include:

- Data collection: We will use data from a large national survey that was designed explicitly with the goal of generating statistically valid inferences about the U.S. population.
- Descriptive statistics: We will generate statistics that summarize the data concisely, and evaluate different ways to visualize data.
- Exploratory data analysis: We will look for patterns, differences, and other features that address the questions we are interested in. At the same time we will check for inconsistencies and identify limitations.
- Estimation: We will use data from a sample to estimate characteristics of the general population.
- Hypothesis testing: Where we see apparent effects, like a difference between two groups, we will evaluate whether the effect might have happened by chance.

By performing these steps with care to avoid pitfalls, we can reach conclusions that are more justifiable and more likely to be correct.

1.2 The National Survey of Family Growth

Since 1973 the U.S. Centers for Disease Control and Prevention (CDC) have conducted the National Survey of Family Growth (NSFG), which is intended to gather "information on family life, marriage and divorce, pregnancy, infertility, use of contraception, and men's and women's health. The survey results are used ... to plan health services and health education programs, and to do statistical studies of families, fertility, and health." See `http://cdc.gov/nchs/nsfg.htm`.

We will use data collected by this survey to investigate whether first babies tend to come late, and other questions. In order to use this data effectively, we have to understand the design of the study.

The NSFG is a **cross-sectional** study, which means that it captures a snapshot of a group at a point in time. The most common alternative is a **longitudinal** study, which observes a group repeatedly over a period of time.

The NSFG has been conducted seven times; each deployment is called a **cycle**. We will use data from Cycle 6, which was conducted from January 2002 to March 2003.

The goal of the survey is to draw conclusions about a **population**; the target population of the NSFG is people in the United States aged 15-44. Ideally surveys would collect data from every member of the population, but that's seldom possible. Instead we collect data from a subset of the population called a **sample**. The people who participate in a survey are called **respondents**.

In general, cross-sectional studies are meant to be **representative**, which means that every member of the target population has an equal chance of participating. That ideal is hard to achieve in practice, but people who conduct surveys come as close as they can.

The NSFG is not representative; instead it is deliberately **oversampled**. The designers of the study recruited three groups—Hispanics, African-Americans and teenagers—at rates higher than their representation in the U.S. population, in order to make sure that the number of respondents in each of these groups is large enough to draw valid statistical inferences.

Of course, the drawback of oversampling is that it is not as easy to draw conclusions about the general population based on statistics from the survey. We will come back to this point later.

When working with this kind of data, it is important to be familiar with the **codebook**, which documents the design of the study, the survey questions, and the encoding of the responses. The codebook and user's guide for the NSFG data are available from `http://www.cdc.gov/nchs/nsfg/nsfg_cycle6.htm`

1.3 Importing the data

The code and data used in this book are available from `https://github.com/AllenDowney/ThinkStats2`. For information about downloading and working with this code, see Section 0.2.

Once you download the code, you should have a file called `ThinkStats2/code/nsfg.py`. If you run it, it should read a data file, run some tests, and print a message like, "All tests passed."

Let's see what it does. Pregnancy data from Cycle 6 of the NSFG is in a file called `2002FemPreg.dat.gz`; it is a gzip-compressed data file in plain text (ASCII), with fixed width columns. Each line in the file is a **record** that contains data about one pregnancy.

The format of the file is documented in `2002FemPreg.dct`, which is a Stata dictionary file. Stata is a statistical software system; a "dictionary" in this context is a list of variable names, types, and indices that identify where in each line to find each variable.

For example, here are a few lines from `2002FemPreg.dct`:

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

This dictionary describes two variables: `caseid` is a 12-character string that represents the respondent ID; `pregorder` is a one-byte integer that indicates which pregnancy this record describes for this respondent.

The code you downloaded includes `thinkstats2.py`, which is a Python module that contains many classes and functions used in this book, including functions that read the Stata dictionary and the NSFG data file. Here's how they are used in `nsfg.py`:

```
def ReadFemPreg(dct_file='2002FemPreg.dct',
```

```
                dat_file='2002FemPreg.dat.gz'):
    dct = thinkstats2.ReadStataDct(dct_file)
    df = dct.ReadFixedWidth(dat_file, compression='gzip')
    CleanFemPreg(df)
    return df
```

`ReadStataDct` takes the name of the dictionary file and returns `dct`, a `FixedWidthVariables` object that contains the information from the dictionary file. `dct` provides `ReadFixedWidth`, which reads the data file.

1.4 DataFrames

The result of `ReadFixedWidth` is a DataFrame, which is the fundamental data structure provided by pandas, which is a Python data and statistics package we'll use throughout this book. A DataFrame contains a row for each record, in this case one row per pregnancy, and a column for each variable.

In addition to the data, a DataFrame also contains the variable names and their types, and it provides methods for accessing and modifying the data.

If you print `df` you get a truncated view of the rows and columns, and the shape of the DataFrame, which is 13593 rows/records and 244 columns/variables.

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

The attribute `columns` returns a sequence of column names as Unicode strings:

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

The result is an Index, which is another pandas data structure. We'll learn more about Index later, but for now we'll treat it like a list:

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

To access a column from a DataFrame, you can use the column name as a key:

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

The result is a Series, yet another pandas data structure. A Series is like a Python list with some additional features. When you print a Series, you get the indices and the corresponding values:

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

In this example the indices are integers from 0 to 13592, but in general they can be any sortable type. The elements are also integers, but they can be any type.

The last line includes the variable name, Series length, and data type; `int64` is one of the types provided by NumPy. If you run this example on a 32-bit machine you might see `int32`.

You can access the elements of a Series using integer indices and slices:

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

The result of the index operator is an `int64`; the result of the slice is another Series.

You can also access the columns of a DataFrame using dot notation:

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

This notation only works if the column name is a valid Python identifier, so it has to begin with a letter, can't contain spaces, etc.

1.5 Variables

We have already seen two variables in the NSFG dataset, `caseid` and `pregordr`, and we have seen that there are 244 variables in total. For the explorations in this book, I use the following variables:

- `caseid` is the integer ID of the respondent.
- `prglngth` is the integer duration of the pregnancy in weeks.
- `outcome` is an integer code for the outcome of the pregnancy. The code 1 indicates a live birth.
- `pregordr` is a pregnancy serial number; for example, the code for a respondent's first pregnancy is 1, for the second pregnancy is 2, and so on.
- `birthord` is a serial number for live births; the code for a respondent's first child is 1, and so on. For outcomes other than live birth, this field is blank.
- `birthwgt_lb` and `birthwgt_oz` contain the pounds and ounces parts of the birth weight of the baby.
- `agepreg` is the mother's age at the end of the pregnancy.
- `finalwgt` is the statistical weight associated with the respondent. It is a floating-point value that indicates the number of people in the U.S. population this respondent represents.

If you read the codebook carefully, you will see that many of the variables are **recodes**, which means that they are not part of the **raw data** collected by the survey; they are calculated using the raw data.

For example, `prglngth` for live births is equal to the raw variable `wksgest` (weeks of gestation) if it is available; otherwise it is estimated using `mosgest * 4.33` (months of gestation times the average number of weeks in a month).

Recodes are often based on logic that checks the consistency and accuracy of the data. In general it is a good idea to use recodes when they are available, unless there is a compelling reason to process the raw data yourself.

1.6 Transformation

When you import data like this, you often have to check for errors, deal with special values, convert data into different formats, and perform calculations. These operations are called **data cleaning**.

`nsfg.py` includes `CleanFemPreg`, a function that cleans the variables I am planning to use.

```
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` contains the mother's age at the end of the pregnancy. In the data file, `agepreg` is encoded as an integer number of centiyears. So the first line divides each element of `agepreg` by 100, yielding a floating-point value in years.

`birthwgt_lb` and `birthwgt_oz` contain the weight of the baby, in pounds and ounces, for pregnancies that end in live birth. In addition it uses several special codes:

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

Special values encoded as numbers are *dangerous* because if they are not handled properly, they can generate bogus results, like a 99-pound baby. The `replace` method replaces these values with `np.nan`, a special floating-point value that represents "not a number." The `inplace` flag tells `replace` to modify the existing Series rather than create a new one.

As part of the IEEE floating-point standard, all mathematical operations return `nan` if either argument is `nan`:

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

So computations with `nan` tend to do the right thing, and most pandas functions handle `nan` appropriately. But dealing with missing data will be a recurring issue.

The last line of `CleanFemPreg` creates a new column `totalwgt_lb` that combines pounds and ounces into a single quantity, in pounds.

One important note: when you add a new column to a DataFrame, you must use dictionary syntax, like this

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

Not dot notation, like this:

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

The version with dot notation adds an attribute to the DataFrame object, but that attribute is not treated as a new column.

1.7 Validation

When data is exported from one software environment and imported into another, errors might be introduced. And when you are getting familiar with a new dataset, you might interpret data incorrectly or introduce other misunderstandings. If you take time to validate the data, you can save time later and avoid errors.

One way to validate data is to compute basic statistics and compare them with published results. For example, the NSFG codebook includes tables that summarize each variable. Here is the table for `outcome`, which encodes the outcome of each pregnancy:

```
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
```

The Series class provides a method, `value_counts`, that counts the number of times each value appears. If we select the `outcome` Series from the DataFrame, we can use `value_counts` to compare with the published data:

```
>>> df.outcome.value_counts().sort_index()
1    9148
2    1862
3     120
```

```
4    1921
5     190
6     352
```

The result of `value_counts` is a Series; `sort_index` sorts the Series by index, so the values appear in order.

Comparing the results with the published table, it looks like the values in `outcome` are correct. Similarly, here is the published table for `birthwgt_lb`

```
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
```

And here are the value counts:

```
>>> 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
```

The counts for 6, 7, and 8 pounds check out, and if you add up the counts for 0-5 and 9-95, they check out, too. But if you look more closely, you will notice one value that has to be an error, a 51 pound baby!

To deal with this error, I added a line to `CleanFemPreg`:

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

This statement replaces invalid values with `np.nan`. The expression in brackets yields a Series of type `bool`, where True indicates that the condition is true. When a boolean Series is used as an index, it selects only the elements that satisfy the condition.

1.8 Interpretation

To work with data effectively, you have to think on two levels at the same time: the level of statistics and the level of context.

As an example, let's look at the sequence of outcomes for a few respondents. Because of the way the data files are organized, we have to do some processing to collect the pregnancy data for each respondent. Here's a function that does that:

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

`df` is the DataFrame with pregnancy data. The `iteritems` method enumerates the index (row number) and `caseid` for each pregnancy.

`d` is a dictionary that maps from each case ID to a list of indices. If you are not familiar with `defaultdict`, it is in the Python `collections` module. Using `d`, we can look up a respondent and get the indices of that respondent's pregnancies.

This example looks up one respondent and prints a list of outcomes for her pregnancies:

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

`indices` is the list of indices for pregnancies corresponding to respondent `10229`.

Using this list as an index into `df.outcome` selects the indicated rows and yields a Series. Instead of printing the whole Series, I selected the `values` attribute, which is a NumPy array.

The outcome code 1 indicates a live birth. Code 4 indicates a miscarriage; that is, a pregnancy that ended spontaneously, usually with no known medical cause.

Statistically this respondent is not unusual. Miscarriages are common and there are other respondents who reported as many or more.

But remembering the context, this data tells the story of a woman who was pregnant six times, each time ending in miscarriage. Her seventh and most recent pregnancy ended in a live birth. If we consider this data with empathy, it is natural to be moved by the story it tells.

Each record in the NSFG dataset represents a person who provided honest answers to many personal and difficult questions. We can use this data to answer statistical questions about family life, reproduction, and health. At the same time, we have an obligation to consider the people represented by the data, and to afford them respect and gratitude.

1.9 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!

1.10 Glossary

- **anecdotal evidence**: Evidence, often personal, that is collected casually rather than by a well-designed study.
- **population**: A group we are interested in studying. "Population" often refers to a group of people, but the term is used for other subjects, too.
- **cross-sectional study**: A study that collects data about a population at a particular point in time.
- **cycle**: In a repeated cross-sectional study, each repetition of the study is called a cycle.
- **longitudinal study**: A study that follows a population over time, collecting data from the same group repeatedly.
- **record**: In a dataset, a collection of information about a single person or other subject.
- **respondent**: A person who responds to a survey.
- **sample**: The subset of a population used to collect data.
- **representative**: A sample is representative if every member of the population has the same chance of being in the sample.
- **oversampling**: The technique of increasing the representation of a sub-population in order to avoid errors due to small sample sizes.
- **raw data**: Values collected and recorded with little or no checking, calculation or interpretation.
- **recode**: A value that is generated by calculation and other logic applied to raw data.
- **data cleaning**: Processes that include validating data, identifying errors, translating between data types and representations, etc.