Week 3 Exercises (1-1, 1-2, 2-1, 2-4)

Kausik Chattapadhyay

Exercise 1-1

In [1]:
```python
from os.path import basename, exists


def download(url):
    filename = basename(url)
    if not exists(filename):
        from urllib.request import urlretrieve

        local, _ = urlretrieve(url, filename)
        print("Downloaded " + local)


download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/thinkstats2.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/thinkplot.py")
```

```
Downloaded thinkstats2.py
Downloaded thinkplot.py
```

In [2]:
```python
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/nsfg.py")

download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/2002FemPreg.dct")
download(
    "https://github.com/AllenDowney/ThinkStats2/raw/master/code/2002FemPreg.dat.gz"
)
```

```
Downloaded nsfg.py
Downloaded 2002FemPreg.dct
Downloaded 2002FemPreg.dat.gz
```

Examples from Chapter 1

Read NSFG data into a Pandas DataFrame.

In [3]:
```python
import nsfg
```

In [4]:
```python
preg = nsfg.ReadFemPreg()
preg.head()
```

Out[4]:

	caseid	pregordr	howpreg_n	howpreg_p	moscurrp	nowprgdk	pregend1	pregend2	nbrnaliv	multbrth	...	laborfor_i	religion_i	metro_i	basewgt	adj_mod_basewgt	finalwgt	secu_p	sest
0	1	1	NaN	NaN	NaN	NaN	6.0	NaN	1.0	NaN	...	0	0	0	3410.389399	3869.349602	6448.271112	2	9
1	1	2	NaN	NaN	NaN	NaN	6.0	NaN	1.0	NaN	...	0	0	0	3410.389399	3869.349602	6448.271112	2	9
2	2	1	NaN	NaN	NaN	NaN	5.0	NaN	3.0	5.0	...	0	0	0	7226.301740	8567.549110	12999.542264	2	12
3	2	2	NaN	NaN	NaN	NaN	6.0	NaN	1.0	NaN	...	0	0	0	7226.301740	8567.549110	12999.542264	2	12
4	2	3	NaN	NaN	NaN	NaN	6.0	NaN	1.0	NaN	...	0	0	0	7226.301740	8567.549110	12999.542264	2	12

5 rows × 244 columns

Print the column names.

In [5]:
```python
preg.columns
```

Out[5]:
```
Index(['caseid', 'pregordr', 'howpreg_n', 'howpreg_p', 'moscurrp', 'nowprgdk',
       'pregend1', 'pregend2', 'nbrnaliv', 'multbrth',
       ...
       'laborfor_i', 'religion_i', 'metro_i', 'basewgt', 'adj_mod_basewgt',
       'finalwgt', 'secu_p', 'sest', 'cmintvw', 'totalwgt_lb'],
      dtype='object', length=244)
```

Select a single column name.

In [6]:
```python
preg.columns[1]
```

Out[6]:
```
'pregordr'
```

Select a column and check what type it is.

In [7]:
```python
pregordr = preg['pregordr']
type(pregordr)
```

Out[7]:
```
pandas.core.series.Series
```

Print a column.

In [8]:
```python
pregordr
```

Out[8]:
```
0        1
1        2
2        1
3        2
4        3
        ..
13588    1
13589    2
13590    3
13591    4
13592    5
Name: pregordr, Length: 13593, dtype: int64
```

Select a single element from a column.

In [9]:
```python
pregordr[0]
```

Out[9]:
```
1
```

Select a slice from a column.

In [10]:
```python
pregordr[2:5]
```

Out[10]:
```
2    1
3    2
4    3
Name: pregordr, dtype: int64
```

Select a column using dot notation.

In [11]:
```python
pregordr = preg.pregordr
```

Count the number of times each value occurs.

In [12]:
```python
preg.outcome.value_counts().sort_index()
```

Out[12]:
```
1    9148
2    1862
3     120
4    1921
5     190
6     352
Name: outcome, dtype: int64
```

Check the values of another variable.

In [13]:
```python
preg.birthwgt_lb.value_counts().sort_index()
```

Out[13]:
```
0.0        8
1.0       40
2.0       53
3.0       98
4.0      229
5.0      697
6.0     2223
7.0     3049
8.0     1889
9.0      623
10.0     132
11.0      26
12.0      10
13.0       3
14.0       3
15.0       1
Name: birthwgt_lb, dtype: int64
```

Make a dictionary that maps from each respondent's `caseid` to a list of indices into the pregnancy `DataFrame`. Use it to select the pregnancy outcomes for a single respondent.

In [14]:
```python
caseid = 10229
preg_map = nsfg.MakePregMap(preg)
indices = preg_map[caseid]
preg.outcome[indices].values
```

Out[14]:
```
array([4, 4, 4, 4, 4, 4, 1])
```

Exercises

Select the `birthord` column, print the value counts, and compare to results published in the codebook

In [15]:
```python
preg.birthord.value_counts().sort_index()
```

Out[15]:
```
1.0     4413
2.0     2874
3.0     1234
4.0      421
5.0      126
6.0       50
7.0       20
8.0        7
9.0        2
10.0       1
Name: birthord, dtype: int64
```

We can also use `isnull` to count the number of nans.

In [16]:
```python
preg.birthord.isnull().sum()
```

Out[16]:
```
4445
```

Select the `prglngth` column, print the value counts, and compare to results published in the codebook

In [17]:
```python
preg.prglngth.value_counts().sort_index()
```

Out[17]:
```
0       15
1        9
2       78
3      151
4      412
5      181
6      543
7      175
8      409
9      594
10     137
11     202
12     170
13     446
14      29
15      39
16      44
17     253
18      17
19      34
20      18
21      37
22     147
23      12
24      31
25      15
26     117
27       8
28      38
29      23
30     198
31      29
32     122
33      50
34      60
35     357
36     329
37     457
38     609
39    4744
40    1120
41     591
42     328
43     148
44      46
45      10
46       1
47       1
48       7
50       2
Name: prglngth, dtype: int64
```

To compute the mean of a column, you can invoke the `mean` method on a Series. For example, here is the mean birthweight in pounds:

In [18]:
```python
preg.totalwgt_lb.mean()
```

Out[18]:
```
7.265628457623368
```

Create a new column named `totalwgt_kg` that contains birth weight in kilograms. Compute its mean. Remember that when you create a new column, you have to use dictionary syntax, not dot notation.

In [19]:
```python
preg['totalwgt_kg'] = preg.totalwgt_lb / 2.2
preg.totalwgt_kg.mean()
```

Out[19]:
```
3.302558389828807
```

`nsfg.py` also provides `ReadFemResp`, which reads the female respondents file and returns a `DataFrame`:

In [20]:
```python
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/2002FemResp.dct")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/2002FemResp.dat.gz")
```

```
Downloaded 2002FemResp.dct
Downloaded 2002FemResp.dat.gz
```

In [21]:
```python
resp = nsfg.ReadFemResp()
```

`DataFrame` provides a method `head` that displays the first five rows:

In [22]:
```python
resp.head()
```

Out[22]:

	caseid	rscrinf	rdormres	rostscrn	rscreenhisp	rscreenrace	age_a	age_r	cmbirth	agescrn	...	pubassis_i	basewgt	adj_mod_basewgt	finalwgt	secu_r	sest	cmintvw	cmlstyr	screentime
0	2298	1	5	5	1	5.0	27	27	902	27	...	0	3247.916977	5123.759559	5556.717241	2	18	1234	1222	18:26:36
1	5012	1	5	1	5	5.0	42	42	718	42	...	0	2335.279149	2846.799490	4744.191350	2	18	1233	1221	16:30:59
2	11586	1	5	1	5	5.0	43	43	708	43	...	0	2335.279149	2846.799490	4744.191350	2	18	1234	1222	18:19:09
3	6794	5	5	4	1	5.0	15	15	1042	15	...	0	3783.152221	5071.464231	5923.977368	2	18	1234	1222	15:54:43
4	616	1	5	4	1	5.0	20	20	991	20	...	0	5341.329968	6437.335772	7229.128072	2	18	1233	1221	14:19:44

5 rows × 3087 columns

Select the `age_r` column from `resp` and print the value counts. How old are the youngest and oldest respondents?

In [23]:
```python
resp.age_r.value_counts().sort_index()

# Youngest is 15 years and oldest is 44 years old.
```

Out[23]:
```
15    217
16    223
17    234
18    235
19    241
20    258
21    267
22    287
23    282
24    269
25    267
26    260
27    255
28    252
29    262
30    292
31    278
32    273
33    257
34    255
35    262
36    266
37    271
38    256
39    215
40    256
41    250
42    215
43    253
44    235
Name: age_r, dtype: int64
```

We can use the `caseid` to match up rows from `resp` and `preg`. For example, we can select the row from `resp` for `caseid` 2298 like this:

In [24]:
```python
resp[resp.caseid==2298]
```

Out[24]:

	caseid	rscrinf	rdormres	rostscrn	rscreenhisp	rscreenrace	age_a	age_r	cmbirth	agescrn	...	pubassis_i	basewgt	adj_mod_basewgt	finalwgt	secu_r	sest	cmintvw	cmlstyr	screentime
0	2298	1	5	5	1	5.0	27	27	902	27	...	0	3247.916977	5123.759559	5556.717241	2	18	1234	1222	18:26:36

1 rows × 3087 columns

And we can get the corresponding rows from `preg` like this:

In [25]:
```python
preg[preg.caseid==2298]
```

Out[25]:

	caseid	pregordr	howpreg_n	howpreg_p	moscurrp	nowprgdk	pregend1	pregend2	nbrnaliv	multbrth	...	religion_i	metro_i	basewgt	adj_mod_basewgt	finalwgt	secu_p	sest	cmintvw
2610	2298	1	NaN	NaN	NaN	NaN	6.0	NaN	1.0	NaN	...	0	0	3247.916977	5123.759559	5556.717241	2	18	NaN
2611	2298	2	NaN	NaN	NaN	NaN	6.0	NaN	1.0	NaN	...	0	0	3247.916977	5123.759559	5556.717241	2	18	NaN
2612	2298	3	NaN	NaN	NaN	NaN	6.0	NaN	1.0	NaN	...	0	0	3247.916977	5123.759559	5556.717241	2	18	NaN
2613	2298	4	NaN	NaN	NaN	NaN	6.0	NaN	1.0	NaN	...	0	0	3247.916977	5123.759559	5556.717241	2	18	NaN

4 rows × 245 columns

How old is the respondent with `caseid` 1?

In [26]:
```python
resp[resp.caseid==1].age_r

# 44 Years
```

Out[26]:
```
1069    44
Name: age_r, dtype: int64
```

What are the pregnancy lengths for the respondent with `caseid` 2298?

In [27]:
```python
preg[preg.caseid==2298].prglngth
```

Out[27]:
```
2610    40
2611    36
2612    30
2613    40
Name: prglngth, dtype: int64
```

What was the birthweight of the first baby born to the respondent with `caseid` 5012?

In [28]:
```python
preg[preg.caseid==5012].birthwgt_lb

# 6 lb
```

Out[28]:
```
5515    6.0
Name: birthwgt_lb, dtype: float64
```

Exercise 1-2

In [34]:
```python
from __future__ import print_function, division

import numpy as np
import sys

import nsfg
import thinkstats2


def ReadFemResp(dct_file='2002FemResp.dct',
                dat_file='2002FemResp.dat.gz',
                nrows=None):
    """Reads the NSFG respondent data.
    dct_file: string file name
    dat_file: string file name
    returns: DataFrame
    """
    dct = thinkstats2.ReadStataDct(dct_file)
    df = dct.ReadFixedWidth(dat_file, compression='gzip', nrows=nrows)
    CleanFemResp(df)
    return df


def CleanFemResp(df):
    """Recodes variables from the respondent frame.
    df: DataFrame
    """
    pass


def ValidatePregnum(resp):
    """Validate pregnum in the respondent file.
    resp: respondent DataFrame
    """
    # read the pregnancy frame
    preg = nsfg.ReadFemPreg()

    # make the map from caseid to list of pregnancy indices
    preg_map = nsfg.MakePregMap(preg)

    # iterate through the respondent pregnum series
    for index, pregnum in resp.pregnum.items():
        caseid = resp.caseid[index]
        indices = preg_map[caseid]

        # check that pregnum from the respondent file equals
        # the number of records in the pregnancy file
        if len(indices) != pregnum:
            print(caseid, len(indices), pregnum)
            return False

    return True
```

In [35]:
```python
def main(script):
    """Tests the functions in this module.
    script: string script name
    """
    resp = ReadFemResp()

    assert(len(resp) == 7643)
    print("pregnum value_counts")
    print(resp.pregnum.value_counts().sort_index())
    assert(resp.pregnum.value_counts()[1] == 1267)
    assert(ValidatePregnum(resp))
    print("Validated results against NSFG codebook")


main("script1")
```

```
pregnum value_counts
0     2610
1     1267
2     1432
3     1110
4      611
5      305
6      150
7       80
8       40
9       21
10       9
11       3
12       2
14       2
19       1
Name: pregnum, dtype: int64
Validated results against NSFG codebook
```

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.

If I wanted to get a story on the evening news, I would use mean as summary statistics.

If I wanted to reassure an anxious patient, I would use variance to check the result.

In my opinion, I don't think the first baby will arrive late. Below are my reasons,

```
1. First, the conditions at which both the groups were kept is not clear

  For example, their daily routine, food consumed, existing body state and health conditions. If the      conditions of the sample groups are not
the same, we cannot get the fair output.

2. Second, selection bias

 People who join a discussion of this question might be interested because their first babies were late.

3. 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.

4. The mean between the first baby and others is only 0.078

From the mean, we cannot find the difference between them.

5. The CohenEffectSize is only 0.029
```

Execise 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?

In [46]:
```python
import nsfg
import thinkplot
```

In [40]:
```python
preg = nsfg.ReadFemPreg()
live = preg[preg.outcome == 1]
firsts = live[live.birthord == 1]
others = live[live.birthord != 1]
```

In [42]:
```python
firsts.totalwgt_lb.mean(), others.totalwgt_lb.mean()

#First babies are lighter than others
```

Out[42]:
```
(7.201094430437772, 7.325855614973262)
```

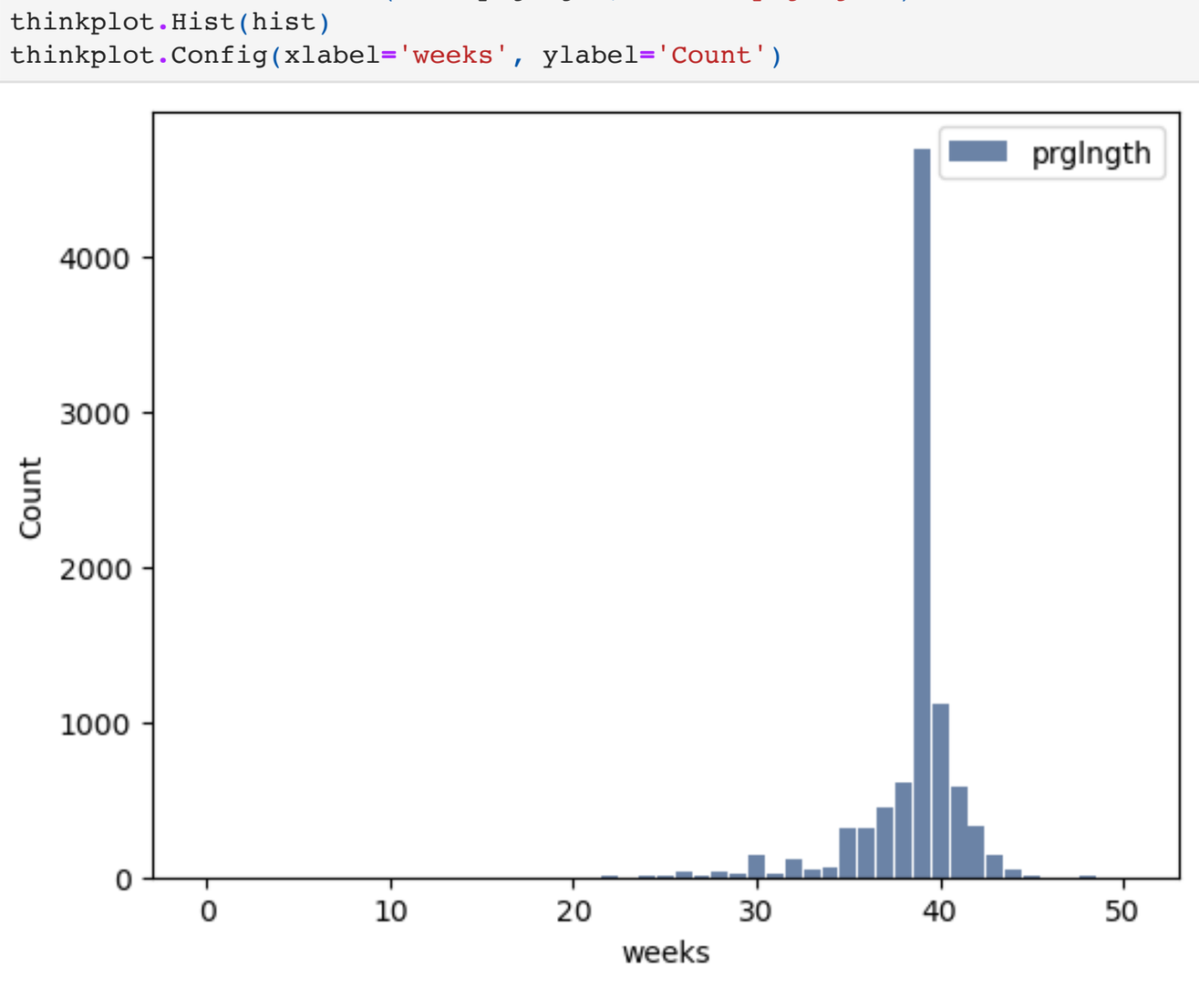
This functon computes the Cohen effect size, which is the difference in means expressed in number of standard deviations:

In [43]:
```python
def CohenEffectSize(group1, group2):
    """Computes Cohen's effect size for two groups.

    group1: Series or DataFrame
    group2: Series or DataFrame

    returns: float if the arguments are Series;
             Series if the arguments are DataFrames
    """
    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 / np.sqrt(pooled_var)
    return d
```

In [44]:
```python
CohenEffectSize(firsts.totalwgt_lb, others.totalwgt_lb)
```

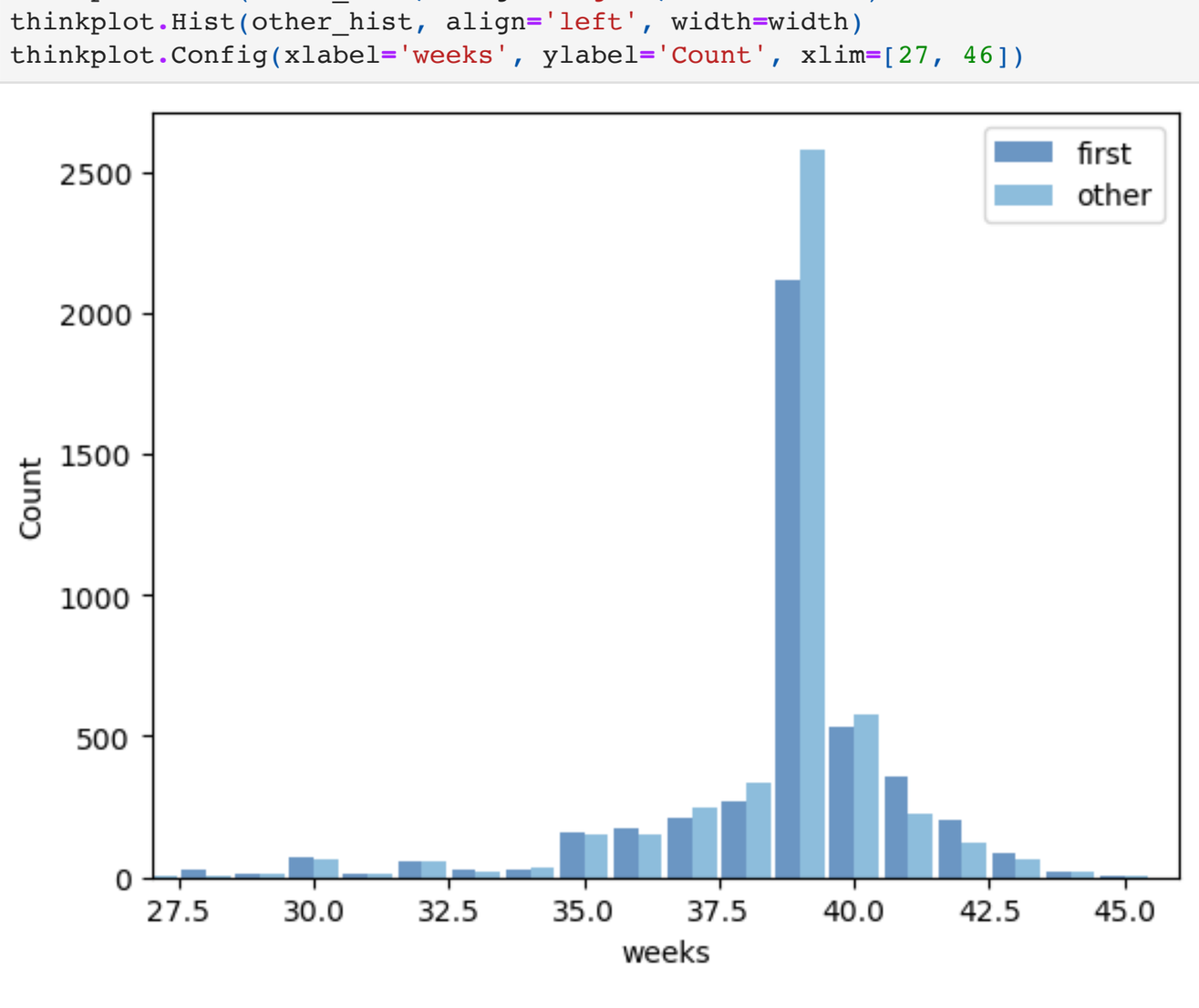
Out[44]:
```
-0.088672927072602
```

In [47]:
```python
hist = thinkstats2.Hist(live.prglngth, label='prglngth')
thinkplot.Hist(hist)
thinkplot.Config(xlabel='weeks', ylabel='Count')
```



In [48]:
```python
first_hist = thinkstats2.Hist(firsts.prglngth, label='first')
other_hist = thinkstats2.Hist(others.prglngth, label='other')
```

In [49]:
```python
width = 0.45
thinkplot.PrePlot(2)
thinkplot.Hist(first_hist, align='right', width=width)
thinkplot.Hist(other_hist, align='left', width=width)
thinkplot.Config(xlabel='weeks', ylabel='Count', xlim=[27, 46])
```



Here's are the mean pregnancy lengths for first babies and others:

In [50]:
```python
firsts.prglngth.mean(), others.prglngth.mean()
```

Out[50]:
```
(38.60095173351461, 38.52291446673706)
```

And here's the difference (in weeks):

In [51]:
```python
firsts.prglngth.mean() - others.prglngth.mean()
```

Out[51]:
```
0.07803726677754952
```

Compute the Cohen effect size for the difference in pregnancy length for first babies and others.

In [52]:
```python
CohenEffectSize(firsts.prglngth, others.prglngth)
```

Out[52]:
```
0.028879044654449883
```

In [ ]: