

Exercise from chap01ex.ipynb, exclude the prefilled book demo code

```
from os.path import basename, exists

def download(url):
    filename = basename(url)
    if 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")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
nsfg.py")

Downloaded thinkstats2.py
Downloaded thinkplot.py
Downloaded nsfg.py

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

"https://github.com/AllenDowney/ThinkStats2/raw/master/code/2002FemPre
g.dat.gz"
)

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

Read NSFG data into a Pandas DataFrame.

```
import nsfg

preg = nsfg.ReadFemPreg()

C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:68:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
```

Series through chained assignment using an inplace method. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df.birthwgt_lb.replace(na_vals, np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:69:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.
```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df.birthwgt_oz.replace(na_vals, np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:70:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.
```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df.hpagelb.replace(na_vals, np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:72:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.
```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the

original object.

```
df.babysex.replace([7, 9], np.nan, inplace=True)
```

C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:73:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df.nbrnaliv.replace([9], np.nan, inplace=True)
```

exercise 1.1 start from here:

Select the `birthord` column, print the value counts, and compare to results published in the [codebook](#)

```
birthord = preg['birthord']
```

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

```
preg.birthord.isnull().sum()
```

4445

Select the `prglngth` column, print the value counts, and compare to results published in the [codebook](#)

```
preg.prglngth.value_counts().sort_index()
```

```
prglngth
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: count, 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:

```
preg.totalwgt_lb.mean()
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.

```
preg['totalwgt_kg'] = preg.totalwgt_lb*0.45359237
preg.totalwgt_kg.mean()

3.295633631632828
```

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

```
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

resp = nsfg.ReadFemResp()
```

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

```
resp.head()
```

| | caseid | rscrinf | rdormres | rostscrn | rscreenhisp | rscreenrace |
|---------|--------|---------|----------|----------|-------------|-------------|
| age_a \ | | | | | | |
| 0 | 2298 | 1 | 5 | 5 | 1 | 5.0 |
| 27 | | | | | | |
| 1 | 5012 | 1 | 5 | 1 | 5 | 5.0 |
| 42 | | | | | | |
| 2 | 11586 | 1 | 5 | 1 | 5 | 5.0 |
| 43 | | | | | | |
| 3 | 6794 | 5 | 5 | 4 | 1 | 5.0 |
| 15 | | | | | | |
| 4 | 616 | 1 | 5 | 4 | 1 | 5.0 |
| 20 | | | | | | |

| | age_r | cmbirth | agescrn | ... | pubassis_i | basewgt |
|-------------------|-------|---------|---------|-----|------------|-------------|
| adj_mod_basewgt \ | | | | | | |
| 0 | 27 | 902 | 27 | ... | 0 | 3247.916977 |
| 5123.759559 | | | | | | |
| 1 | 42 | 718 | 42 | ... | 0 | 2335.279149 |
| 2846.799490 | | | | | | |
| 2 | 43 | 708 | 43 | ... | 0 | 2335.279149 |
| 2846.799490 | | | | | | |
| 3 | 15 | 1042 | 15 | ... | 0 | 3783.152221 |
| 5071.464231 | | | | | | |
| 4 | 20 | 991 | 20 | ... | 0 | 5341.329968 |

6437.335772

| | finalwgt | secu_r | sest | cmintvw | cmlstyr | screen time | intvlngth |
|---|-------------|--------|------|---------|---------|-------------|------------|
| 0 | 5556.717241 | 2 | 18 | 1234 | 1222 | 18:26:36 | 110.492667 |
| 1 | 4744.191350 | 2 | 18 | 1233 | 1221 | 16:30:59 | 64.294000 |
| 2 | 4744.191350 | 2 | 18 | 1234 | 1222 | 18:19:09 | 75.149167 |
| 3 | 5923.977368 | 2 | 18 | 1234 | 1222 | 15:54:43 | 28.642833 |
| 4 | 7229.128072 | 2 | 18 | 1233 | 1221 | 14:19:44 | 69.502667 |

[5 rows x 3087 columns]

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

```
resp.age_r.value_counts().sort_index()
```

```
age_r
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: count, 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:

```

resp[resp.caseid==2298]

```

| | caseid | rscrinf | rdormres | rostscrn | rscreenhisp | rscreenrace |
|---------|--------|---------|----------|----------|-------------|-------------|
| age_a \ | | | | | | |
| 0 | 2298 | 1 | 5 | 5 | 1 | 5.0 |
| 27 | | | | | | |

| | age_r | cmbirth | agescrn | ... | pubassis_i | basewgt |
|-------------------|-------|---------|---------|-----|------------|-------------|
| adj_mod_basewgt \ | | | | | | |
| 0 | 27 | 902 | 27 | ... | 0 | 3247.916977 |
| 5123.759559 | | | | | | |

| | finalwgt | secu_r | sest | cmintvw | cmlstyr | screen time | intvlngth |
|---|-------------|--------|------|---------|---------|-------------|------------|
| 0 | 5556.717241 | 2 | 18 | 1234 | 1222 | 18:26:36 | 110.492667 |


```

[1 rows x 3087 columns]

```

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

```

preg[preg.caseid==2298]

```

| | caseid | pregordr | howpreg_n | howpreg_p | moscurrp | nowprgdk |
|------------|--------|----------|-----------|-----------|----------|----------|
| pregendl \ | | | | | | |
| 2610 | 2298 | 1 | NaN | NaN | NaN | NaN |
| 6.0 | | | | | | |
| 2611 | 2298 | 2 | NaN | NaN | NaN | NaN |
| 6.0 | | | | | | |
| 2612 | 2298 | 3 | NaN | NaN | NaN | NaN |
| 6.0 | | | | | | |
| 2613 | 2298 | 4 | NaN | NaN | NaN | NaN |
| 6.0 | | | | | | |

| | pregend2 | nbrnaliv | multbrth | ... | religion_i | metro_i |
|-------------|----------|----------|----------|-----|------------|---------|
| basewgt \ | | | | | | |
| 2610 | NaN | 1.0 | NaN | ... | 0 | 0 |
| 3247.916977 | | | | | | |
| 2611 | NaN | 1.0 | NaN | ... | 0 | 0 |
| 3247.916977 | | | | | | |
| 2612 | NaN | 1.0 | NaN | ... | 0 | 0 |
| 3247.916977 | | | | | | |

```

2613      NaN      1.0      NaN ...      0      0
3247.916977

      adj_mod_basewgt      finalwgt      secu_p      sest      cmintvw      totalwgt_lb
\
2610      5123.759559      5556.717241      2      18      NaN      6.8750
2611      5123.759559      5556.717241      2      18      NaN      5.5000
2612      5123.759559      5556.717241      2      18      NaN      4.1875
2613      5123.759559      5556.717241      2      18      NaN      6.8750

      totalwgt_kg
2610      3.118448
2611      2.494758
2612      1.899418
2613      3.118448

[4 rows x 245 columns]

```

How old is the respondent with `caseid` 1?

```

resp[resp.caseid==1].age_r

1069      44
Name: age_r, dtype: int64

```

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

```

preg[preg.caseid==2298].prglngth

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?

```

preg[preg.caseid==5012]

      caseid      pregordr      howpreg_n      howpreg_p      moscurrp      nowprgdk
pregend1 \
5515      5012      1      NaN      NaN      NaN      NaN
6.0

      pregend2      nbrnaliv      multbrth ...      religion_i      metro_i
basewgt \

```



```
5515      NaN      1.0      NaN ...      0      0
2335.279149
```

```
      adj_mod_basewgt      finalwgt      secu_p      sest      cmintvw      totalwgt_lb
\
5515      2846.79949      4744.19135      2      18      NaN      6.0
```

```
      totalwgt_kg
5515      2.721554
```

```
[1 rows x 245 columns]
```

```
#preg[preg.caseid==5012][preg.pregordr==1].totalwgt_lb
preg[(preg.caseid==5012) & (preg.pregordr==1)].totalwgt_lb
```

```
5515      6.0
Name: totalwgt_lb, dtype: float64
```

```
preg.describe()
```

```
      caseid      pregordr      howpreg_n      howpreg_p
moscurrp \
count      13593.000000      13593.000000      352.000000      349.000000      352.000000
mean      6216.526595      2.349150      15.144886      1.34384      4.647727
std      3645.417341      1.577807      13.922211      0.47567      2.527523
min      1.000000      1.000000      0.000000      1.000000      0.000000
25%      3022.000000      1.000000      5.000000      1.000000      2.000000
50%      6161.000000      2.000000      9.000000      1.000000      5.000000
75%      9423.000000      3.000000      23.000000      2.000000      7.000000
max      12571.000000      19.000000      99.000000      2.000000      9.000000
```

```
      nowprgdk      pregend1      pregend2      nbrnaliv      multbrth ...
\
count      3.000000      13241.000000      18.000000      9144.000000      163.000000 ...
mean      3.666667      4.650177      4.055556      1.022419      1.834356 ...
std      4.618802      1.849790      1.696787      0.190098      1.630208 ...
min      1.000000      1.000000      1.000000      1.000000      1.000000 ...
25%      1.000000      3.000000      3.000000      1.000000      1.000000 ...
```

| | | | | | | |
|-----|----------|----------|----------|----------|----------|-----|
| 50% | 1.000000 | 6.000000 | 4.000000 | 1.000000 | 1.000000 | ... |
| 75% | 5.000000 | 6.000000 | 6.000000 | 1.000000 | 1.000000 | ... |
| max | 9.000000 | 9.000000 | 6.000000 | 5.000000 | 5.000000 | ... |

| | religion_i | metro_i | basewgt | adj_mod_basewgt |
|------------|--------------|---------|--------------|-----------------|
| finalwgt \ | | | | |
| count | 13593.000000 | 13593.0 | 13593.000000 | 13593.000000 |
| mean | 0.003016 | 0.0 | 4216.271164 | 5383.982581 |
| std | 0.058727 | 0.0 | 3982.680473 | 5640.499431 |
| min | 0.000000 | 0.0 | 64.577101 | 71.201194 |
| 25% | 0.000000 | 0.0 | 2335.445237 | 2798.048902 |
| 50% | 0.000000 | 0.0 | 3409.648504 | 4127.220642 |
| 75% | 0.000000 | 0.0 | 4869.941451 | 5795.692880 |
| max | 2.000000 | 0.0 | 99707.832014 | 157143.686687 |

| | secu_p | sest | cmintvw | totalwgt_lb | totalwgt_kg |
|-------|--------------|--------------|---------|-------------|-------------|
| count | 13593.000000 | 13593.000000 | 0.0 | 9038.000000 | 9038.000000 |
| mean | 1.487310 | 44.083352 | NaN | 7.265628 | 3.295634 |
| std | 0.499857 | 24.110403 | NaN | 1.408293 | 0.638791 |
| min | 1.000000 | 1.000000 | NaN | 0.125000 | 0.056699 |
| 25% | 1.000000 | 25.000000 | NaN | 6.500000 | 2.948350 |
| 50% | 1.000000 | 45.000000 | NaN | 7.375000 | 3.345244 |
| 75% | 2.000000 | 65.000000 | NaN | 8.125000 | 3.685438 |
| max | 2.000000 | 84.000000 | NaN | 15.437500 | 7.002332 |

[8 rows x 245 columns]

End of exercise 1.1

Exercise 1.2: Create a file named chap01ex.py and write code that reads the respondent file, 2002FemResp.dat.gz

```
import thinkstats2

from collections import defaultdict

# read respondent file
def ReadFemResp(dct_file='2002FemResp.dct',
                dat_file='2002FemResp.dat.gz',
                nrows=None):

    dct = thinkstats2.ReadStataDct(dct_file)
    df = dct.ReadFixedWidth(dat_file, compression='gzip', nrows=nrows)
    #CleanFemResp(df)
    return df

#read preg file
def ReadFemPreg(dct_file='2002FemPreg.dct',
                dat_file='2002FemPreg.dat.gz'):

    dct = thinkstats2.ReadStataDct(dct_file)
    df = dct.ReadFixedWidth(dat_file, compression='gzip')
    nsfg.CleanFemPreg(df)
    return df

def ValidatePregnum(resp, preg):
    # make the map from caseid to list of pregnancy indices
    preg_map = 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
```

```

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

def main():
    # read and validate the respondent file

    resp = ReadFemResp()

    assert(len(resp) == 7643)
    assert(resp.pregnum.value_counts()[1] == 1267)

    print('All tests passed.')

if __name__ == '__main__':
    main()

```

All tests passed.

End of exercise 1.2

2-1 (Based on the results in this chapter, suppose you were asked to summarize what you learned about whether first babies arrive late...)

Mean pregnancy length for first babies is 38.601;

for other babies it is 38.523.

the standard deviation is 2.7 weeks

(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?)

It would be the standard deviation.

End of exercise 2-1

2-4 (Using the variable totalwgt_lb, investigate whether first babies are lighter or heavier than others...)

Mean

First babies 7.201094430437772

Others 7.325855614973262

Variance

First babies 2.0180273009157768

Others 1.9437810258964572

Difference in lbs -0.12476118453549034

Difference in oz -1.9961789525678455

Difference relative to mean (%age points) -1.7171423678372415

Cohen d -0.088672927072602

first babies is slightly lighter (~2 oz) than others, but the difference is just a fraction of the first babies mean (less than 2%). However, Based on the Cohen-D value, the variation on baby weight is bigger than the variation in pregnancy length

the code for exercise 2-4 is below:

```
import math
import matplotlib.pyplot as plt
import nsfg

def read_data():
    preg = nsfg.ReadFemPreg()

    live = preg[preg.outcome == 1]
    firsts = live[live.birthord == 1]
```

```

    others = live[live.birthord != 1]

    return live, firsts, others

def calculate(live, firsts, others):

    mean1 = firsts.totalwgt_lb.mean()
    mean2 = others.totalwgt_lb.mean()
    print(f'Mean of First {mean1=} , others {mean2=}')

    var1 = firsts.totalwgt_lb.var()
    var2 = others.totalwgt_lb.var()
    print(f'Variance of First {var1=} , others {var2=}')

    # calculate Choen D
    diff = mean1 - mean2
    n1, n2 = len(firsts.totalwgt_lb), len(others.totalwgt_lb)
    pooled_var = (n1 * var1 + n2 * var2) / (n1 + n2)
    d = diff / math.sqrt(pooled_var)
    print('Cohen d', d)

    plt.hist(firsts.totalwgt_lb, label='first baby', alpha=0.5)
    plt.hist(others.totalwgt_lb, label='other babies', alpha=0.5)
    plt.legend(loc='best')
    plt.show()

def main(script):
    live, firsts, others = read_data()
    calculate(live, firsts, others)

if __name__ == '__main__':
    main('test')

```

```

Mean of First mean1=7.201094430437772 , others 7.325855614973262
Variance of First var1=2.018027300915786 , others 1.9437810258964716
Cohen d -0.08864367587767717

```

C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:68:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df.birthwgt_lb.replace(na_vals, np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:69:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.
```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df.birthwgt_oz.replace(na_vals, np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:70:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.
```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df.hpagelb.replace(na_vals, np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:72:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.
```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df.babysex.replace([7, 9], np.nan, inplace=True)
C:\Users\gyanr\gyan-python-workspace\DSC-530\nsfg.py:73:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
```

always behaves as a copy.

For example, when doing `df[col].method(value, inplace=True)`, try using `df.method({col: value}, inplace=True)` or `df[col] = df[col].method(value)` instead, to perform the operation inplace on the original object.

```
df.nbrnaliv.replace([9], np.nan, inplace=True)
```

