Bilenkin530Week3_Exercise_1.1

December 14, 2024

1 Chapter 1

2 Page 11: (Exercises 1-1)

[240]:
```
import warnings
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)

warnings.filterwarnings('ignore', category = FutureWarning)


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

[241]:
```
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"
)
```

2.1 Examples from Chapter 1

Read NSFG data into a Pandas DataFrame.

[242]:
```
# Importing nsfg data
import nsfg
```

[243]:
```
# Loading the data and displaying the first few rows of the data
preg = nsfg.ReadFemPreg()
preg.head()
```

[243]:

	caseid	pregordr	howpreg_n	howpreg_p	moscurrp	…	finalwgt	\
0	1	1	NaN	NaN	NaN	…	6448.271112	
1	1	2	NaN	NaN	NaN	…	6448.271112	
2	2	1	NaN	NaN	NaN	…	12999.542264	
3	2	2	NaN	NaN	NaN	…	12999.542264	
4	2	3	NaN	NaN	NaN	…	12999.542264	

	secu_p	sest	cmintvw	totalwgt_lb
0	2	9	NaN	8.8125
1	2	9	NaN	7.8750
2	2	12	NaN	9.1250
3	2	12	NaN	7.0000
4	2	12	NaN	6.1875

[5 rows x 244 columns]

Print the column names.

[244]:
```
print(preg.columns)
```

```
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.

[245]:
```
preg.columns[1]
```

[245]:
```
'pregordr'
```

Select a column and check what type it is.

[246]:
```
pregordr = preg['pregordr']
type(pregordr)
```

[246]:
```
pandas.core.series.Series
```

Print a column.

[247]:
```
print(pregordr)
```

```
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.

[248]:
```
pregordr[0]
```

[248]:
```
1
```

Select a slice from a column.

[249]:
```
pregordr[2:5]
```

[249]:
```
2    1
3    2
4    3
Name: pregordr, dtype: int64
```

Select a column using dot notation.

[250]:
```
pregordr = preg.pregordr
pregordr
```

[250]:
```
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
```

Count the number of times each value occurs.

[251]:
```
preg.outcome.value_counts().sort_index()
```

[251]:
```
outcome
1    9148
2    1862
3     120
4    1921
5     190
6     352
Name: count, dtype: int64
```

Check the values of another variable.

[252]:
```
preg.birthwgt_lb.value_counts().sort_index()
```

[252]:
```
birthwgt_lb
0.0       8
1.0      40
2.0      53
3.0      98
4.0     229
       ...
11.0     26
12.0     10
13.0      3
14.0      3
15.0      1
Name: count, Length: 16, 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.

[253]:
```
caseid = 10229
preg_map = nsfg.MakePregMap(preg)
indices = preg_map[caseid]
preg.outcome[indices].values
```

[253]:
```
array([4, 4, 4, 4, 4, 4, 1], dtype=int64)
```

2.2 Exercises

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

[254]:
```
# Counting how many times each value occurs in the birthord column and␣
 ↪assigning to as new variable birthord_counts.
birthord_counts_of_each_value_occurs = preg.birthord.value_counts()

# Printing result.
print(birthord_counts_of_each_value_occurs)
```

```
birthord
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: count, dtype: int64
```

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

[255]:
```
preg.birthord.isnull().sum()
```

[255]:
```
4445
```

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

[256]:
```
import pandas as pd
# The codebook provides answer using ranges of weeks instead of giving␣
 ↪individually each number for each week.
# Thus, will create the same ranges as in the codebook and run the code so the␣
 ↪answer can be compared easier.
bins = [0, 13, 26, 50]
labels = ['0-13', '14-26', '27-50']

# Creating new column for the ranges.
preg['prglngth_range'] = pd.cut(preg.prglngth, bins=bins, labels=labels,␣
 ↪right=True)

# Counting the values in each range.
prglngth_range_counts = preg.prglngth_range.value_counts().sort_index()

# Printing results
print(prglngth_range_counts)
```

```
prglngth_range
0-13     3507
14-26     793
27-50    9278
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:

[257]:
```
preg.totalwgt_lb.mean()
```

[257]:
```
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.

[258]:
```
# Creating new column 'totalwgt_kg' and converting from pounds to kilograms.
preg['totalwgt_kg'] = preg['totalwgt_lb'] * 0.453592

# Computing the mean of the 'totalwgt_kg'.
mean_totalwgt_in_kilograms = preg['totalwgt_kg'].mean()

# Printing result.
print(mean_totalwgt_in_kilograms)
```

```
3.2956309433502984
```

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

[259]:
```
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
↪2002FemResp.dct")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
↪2002FemResp.dat.gz")
```

[260]:
```
resp = nsfg.ReadFemResp()
```

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

[261]:
```
# Displaying the first five rows. The head() is set by default to display the␣
↪first five.
resp.head()
```

[261]:

	caseid	rscrinf	rdormres	rostscrn	rscreenhisp	…	sest	cmintvw	\
0	2298	1	5	5	1	…	18	1234	
1	5012	1	5	1	5	…	18	1233	
2	11586	1	5	1	5	…	18	1234	
3	6794	5	5	4	1	…	18	1234	
4	616	1	5	4	1	…	18	1233	

	cmlstyr	screentime	intvlngth
0	1222	18:26:36	110.492667
1	1221	16:30:59	64.294000
2	1222	18:19:09	75.149167
3	1222	15:54:43	28.642833
4	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?

[262]:
```
# Extracting the age_r column from resp DataFrame.
age_r_column = resp.age_r

# Printing the column to see result.
print(age_r_column)

# Counting the occurrences of each age.
each_age_occurance_count = age_r_column.value_counts()
print(each_age_occurance_count)

# Finding the youngest and oldest respondets.
youngest_respondent = age_r_column.min()
oldest_respondent = age_r_column.max()

# Printing results for youngest and oldgerst ages.
print(f"The youngest respondent is {youngest_respondent} years old.")
print(f"The oldest respondent is {oldest_respondent} years old.")
```

```
0       27
1       42
2       43
3       15
4       20
        ..
7638    34
7639    17
7640    29
7641    16
7642    28
Name: age_r, Length: 7643, dtype: int64
age_r
30    292
22    287
23    282
31    278
32    273
     …
17    234
16    223
15    217
42    215
39    215
Name: count, Length: 30, dtype: int64
```

```
The youngest respondent is 15 years old.
The oldest respondent is 44 years old.
```

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:

[263]:
```
resp[resp.caseid==2298]
```

[263]:
```
   caseid  rscrinf  rdormres  rostscrn  rscreenhisp  …  sest  cmintvw  \
0    2298        1         5         5            1  …    18     1234

   cmlstyr  screentime   intvlngth
0     1222    18:26:36  110.492667

[1 rows x 3087 columns]
```

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

[264]:
```
preg[preg.caseid==2298]
```

[264]:
```
      caseid  pregordr  howpreg_n  howpreg_p  moscurrp  …  sest  cmintvw  \
2610    2298         1        NaN        NaN       NaN  …    18      NaN
2611    2298         2        NaN        NaN       NaN  …    18      NaN
2612    2298         3        NaN        NaN       NaN  …    18      NaN
2613    2298         4        NaN        NaN       NaN  …    18      NaN

      totalwgt_lb  prglngth_range  totalwgt_kg
2610       6.8750           27-50     3.118445
2611       5.5000           27-50     2.494756
2612       4.1875           27-50     1.899417
2613       6.8750           27-50     3.118445

[4 rows x 246 columns]
```

How old is the respondent with `caseid` 1?

[265]:
```
# Selecting the row from 'resp' where 'caseid' is 1.
respondent_row_with_caseid_1 = resp[resp['caseid'] == 1]

# Printing result.
respondent_row_with_caseid_1 = respondent_row_with_caseid_1['age_r'].values[0]
print(f"The respondent with caseid 1 is {respondent_row_with_caseid_1} years␣
 ↪old.")
```

```
The respondent with caseid 1 is 44 years old.
```

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

[266]:
```
# Selecting the rows from 'preg' with 'caseid' 2298.
pregnancy_rows = preg[preg['caseid'] == 2298]
```

```
# Extracting the 'prglngth' column that contains pregnacy lengths data.
pregnancy_duration = pregnancy_rows['prglngth']

# Printing result with the pregnancy lenghts.
print(pregnancy_duration)
```

```
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` 5013?

[267]:
```
# Selecting the rows from 'preg' with 'caseid' 5013.
pregnancy_rows = preg[preg['caseid'] == 5013]

# Extracting the 'birthweight' column with birth weights.
birthweights = pregnancy_rows['birthwgt_lb']

# Printing result.
first_baby_birthweight = birthweights.iloc[0]
print(f"The birthweight of the first baby born to the respondent with caseid␣
 ↪5013 is {first_baby_birthweight} pounds.")
```

```
The birthweight of the first baby born to the respondent with caseid 5013 is 7.0
pounds.
```

[ ]:
```
```

3 Chapter 1

4 Page 11: (Exercises 1-2)

[268]:
```
## from __future__ import print_function, division
import numpy as np
import sys
import nsfg
import thinkstats2

# Limiting display of the DataFrame rows and columns to 10 because the file too␣
 ↪big.
pd.set_option('display.max_rows', 10)
pd.set_option('display.max_columns', 10)

# Creating two variables for two directory paths to the files.
dct_file = 'C:\\Users\\maxim\\ThinkStats2\\code\\2002FemResp.dct'
```

```
dat_gz_file = 'C:\\Users\\maxim\\ThinkStats2\\code\\2002FemResp.dat.gz'

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

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

# Creating method to read pregnancy file.
def ReadFemPreg(dct_file = dct_file, dat_file = dat_gz_file, nrows = None):
    dct = thinkstats2.ReadStataDct(dct_file)
    df = dct.ReadFixedWidth(dat_file, compression='gzip', nrows=nrows)
    return df

# Creating method to map caseid to a list of indices into the DataFrame.
def MakePregMap(df):
    d = {}
    for index, caseid in df.caseid.items():
        if caseid not in d:
            d[caseid] = []
        d[caseid].append(index)
    return d

# Creating method for cross-validate the respondent and pregnancy files.
def Cross_Validate(resp, preg, limit = 10):
    pregnancy_map = nsfg.MakePregMap(preg)
    discrepancies = []
    for index, row in resp.iterrows():
        caseid = row.caseid
        pregnum = row.pregnum
        indices = pregnancy_map.get(caseid, [])
        number_records = len(indices)
        if number_records != pregnum:
            discrepancies.append((caseid, pregnum, number_records))
        if len(discrepancies) >= limit:
            break
    for caseid, pregnum, number_records in discrepancies:
            print(f'CaseId {caseid}: pregnum {pregnum} does not match number of␣
 ↪records {number_records}')

def main(script):

    # Reading and storing respondent data into variable.
    resp = ReadFemResp()
```

```
    # Printing the first 10 records because the file is too big.
    print(resp.sample(10))

    # Printing many how times each respondent has been pregnant.
    print('Value how many times each respondent has been pregnant:')
    print(resp.pregnum.value_counts().sort_index())

    # Extracting pregnancy data set and assigning to a variable.
    preg = ReadFemPreg()

    # Cross-validating the respondent and pregnancy files by comparing pregnum
 ↪for each respondent.
    Cross_Validate(resp, preg)

    print('%s: All tests passed.' % script)

# Simulating the command line.
args = ['notebook_name']

# Calling the main function.
main(*args)
```

```
      caseid  rscrinf  rdormres  rostscrn  rscreenhisp  …  sest  cmintvw  \
3509    4972        1         5         3            1  …    56     1234
558     2406        1         5         4            5  …    28     1232
5262    7639        5         5         8            5  …    78     1232
7081    6747        1         5         3            5  …    16     1231
2645    5693        1         5         4            5  …    53     1228
1264    1893        5         1         3            5  …    84     1231
944     8865        5         5         4            5  …    48     1237
3408   11483        1         5         1            5  …    69     1229
3421    3008        5         5         6            5  …    65     1228
3647    4291        5         5         4            5  …    62     1232

      cmlstyr  screentime   intvlngth
3509     1222    15:50:05  106.205667
558      1220    12:18:46   71.265833
5262     1220    20:03:32  147.557333
7081     1219    18:32:34   61.325333
2645     1216    16:47:34   69.090833
1264     1219    14:14:14   46.158500
944      1225    16:18:25  126.048000
3408     1217    15:21:45   89.913833
3421     1216    16:06:03   81.332833
3647     1220    18:44:21  101.806333

[10 rows x 3087 columns]
Value how many times each respondent has been pregnant:
```

```
pregnum
0     2610
1     1267
2     1432
3     1110
4      611
      …
10       9
11       3
12       2
14       2
19       1
Name: count, Length: 15, dtype: int64
CaseId 2298: pregnum 4 does not match number of records 1
CaseId 6794: pregnum 0 does not match number of records 1
CaseId 616: pregnum 0 does not match number of records 1
CaseId 845: pregnum 8 does not match number of records 1
CaseId 10333: pregnum 0 does not match number of records 1
CaseId 855: pregnum 0 does not match number of records 1
CaseId 8656: pregnum 3 does not match number of records 1
CaseId 3566: pregnum 0 does not match number of records 1
CaseId 5917: pregnum 2 does not match number of records 1
CaseId 6320: pregnum 2 does not match number of records 1
notebook_name: All tests passed.
```

[ ]:

5 Chapter 2

6 Page 25: (Exercises 2-1)

Based on the results from this chapter we learned that the first babies arrive on time. The easiest summary statistics to use are mean and median. This would be easy for patients and prospective parents to understand the study. In addition, it would be helpful to include both histograms for the anxious patients so they can visually see it.

As we learned from chapter 2, the distribution in Figure 2.4 is almost normal with slightly negative skewness. Thus, we can avoid using median because there is no significant skewness for our data distribution. From the graph we can see that the most pregnancy length appears to be 39 weeks or approximately 9 months. It would be easy to explain to the anxious patients that most first babies arrive on time. There are some babies delivered earlier than 39 weeks but not that many. There are very few pregnancies that go beyond 43 weeks but those are extremely rare cases. As it was mentioned in this chapter in those cases doctors intervene. There is another histogram presented in Figure 2.5 that shows “first babies” and “other” babies that are delivered from 27-46 weeks length of time. We can see from the histogram that “first babies” delivered slightly earlier than “others”. However, judging by the mean for first babies of 38.601 weeks and the mean for “others” is 38.523. It does show that first babies arrive a little later than others but still not beyond 39 weeks or 9 months. Thus, we can conclude statistically that the majority of first babies don’t arrive late. So,

the question “Do first babies arrive late?” can be easily answered as” No”!

[ ]:

7 Chapter 2

8 Page 25: (Exercises 2-4)

[269]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# Define the path to the file
dat_gz_file = 'C:\\Users\\maxim\\ThinkStats2\\code\\2002FemResp.dat.gz'

# Defining the field widths based on the data structure and load the data
colspecs = [(0, 12), (13, 16), (17, 19), (20, 22), (23, 25), (26, 28), (29,␣
 ↪31), (32, 34), (35, 37), (38, 40), (41, 43)]
df = pd.read_fwf(dat_gz_file, compression='gzip', colspecs=colspecs,␣
 ↪header=None)

# Defining columns and apply correction to weights
df.columns = ['caseid', 'pregordr', 'howpregend', 'nbrnaliv', 'babysex',␣
 ↪'birthwgt_lb', 'birthwgt_oz', 'prglngth', 'outcome', 'birthord',␣
 ↪'totalwgt_lb']
df['birthwgt_lb'] = pd.to_numeric(df['birthwgt_lb'], errors='coerce') / 10
df['birthwgt_oz'] = pd.to_numeric(df['birthwgt_oz'], errors='coerce')
df['totalwgt_lb'] = df['birthwgt_lb'] + (df['birthwgt_oz'] / 16.0)

# Filter to include plausible weight values
filtered_df = df[(df['totalwgt_lb'] > 1) & (df['totalwgt_lb'] < 15)]

# Ensure required columns exist and analyze weights
if 'birthord' in filtered_df.columns and 'totalwgt_lb' in filtered_df.columns:
    first_babies_weights = filtered_df[filtered_df['birthord'] ==␣
 ↪1]['totalwgt_lb']
    subsequent_babies_weights = filtered_df[filtered_df['birthord'] !=␣
 ↪1]['totalwgt_lb']

    # Verify the number of samples
    if len(first_babies_weights) > 0 and len(subsequent_babies_weights) > 0:
        mean_first = first_babies_weights.mean()
        mean_subsequent = subsequent_babies_weights.mean()
        std_first = first_babies_weights.std()
        std_subsequent = subsequent_babies_weights.std()

        # Computing Cohen's d
```

```
        cohens_d = (mean_first - mean_subsequent) / np.sqrt((std_first**2 +␣
↪std_subsequent**2) / 2)

        # Printing statistics and plot histograms
        print(f"Mean weight of first babies: {mean_first:.2f} lbs")
        print(f"Mean weight of subsequent babies: {mean_subsequent:.2f} lbs")
        print(f"Cohen's d: {cohens_d:.2f}")

        plt.hist(first_babies_weights, bins=30, alpha=0.5, label='First Babies')
        plt.hist(subsequent_babies_weights, bins=30, alpha=0.5,␣
↪label='Subsequent Babies')
        plt.xlabel('Weight (lbs)')
        plt.ylabel('Frequency')
        plt.legend(loc='upper right')
        plt.title('Weight Distribution of First and Subsequent Babies')
        plt.show()
```

```
Mean weight of first babies: 7.08 lbs
Mean weight of subsequent babies: 7.02 lbs
Cohen's d: 0.02
```