Rodriguez_Felipe_DSC530_8.2Exercise

February 5, 2023

[164]:
```
# Used from code to download scripts
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")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/nsfg.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/first.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
 ↪2002FemPreg.dct")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
 ↪2002FemPreg.dat.gz")
```

[165]:
```
# Import Scripts
import first
import thinkstats2
import thinkplot
import numpy as np
import pandas as pd
```

9-1 **Exercise:** As sample size increases, the power of a hypothesis test increases, which means it is more likely to be positive if the effect is real. Conversely, as sample size decreases, the test is less likely to be positive even if the effect is real.

To investigate this behavior, run the tests in this chapter with different subsets of the NSFG data. You can use `thinkstats2.SampleRows` to select a random subset of the rows in a DataFrame.

What happens to the p-values of these tests as sample size decreases? What is the smallest sample

size that yields a positive test?

[166]:
```
# Used from Chapter 9
class DiffMeansPermute(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):
        group1, group2 = data
        test_stat = abs(group1.mean() - group2.mean())
        return test_stat

    def MakeModel(self):
        group1, group2 = self.data
        self.n, self.m = len(group1), len(group2)
        self.pool = np.hstack((group1, group2))

    def RunModel(self):
        np.random.shuffle(self.pool)
        data = self.pool[:self.n], self.pool[self.n:]
        return data
```

[167]:
```
# Used from Chapter 9
class CorrelationPermute(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):
        xs, ys = data
        test_stat = abs(thinkstats2.Corr(xs, ys))
        return test_stat

    def RunModel(self):
        xs, ys = self.data
        xs = np.random.permutation(xs)
        return xs, ys
```

[168]:
```
# Used from Chapter 9
class PregLengthTest(thinkstats2.HypothesisTest):

    def MakeModel(self):
        firsts, others = self.data
        self.n = len(firsts)
        self.pool = np.hstack((firsts, others))

        pmf = thinkstats2.Pmf(self.pool)
        self.values = range(35, 44)
        self.expected_probs = np.array(pmf.Probs(self.values))

    def RunModel(self):
        np.random.shuffle(self.pool)
```

```
        data = self.pool[:self.n], self.pool[self.n:]
        return data

    def TestStatistic(self, data):
        firsts, others = data
        stat = self.ChiSquared(firsts) + self.ChiSquared(others)
        return stat

    def ChiSquared(self, lengths):
        hist = thinkstats2.Hist(lengths)
        observed = np.array(hist.Freqs(self.values))
        expected = self.expected_probs * len(lengths)
        stat = sum((observed - expected)**2 / expected)
        return stat
```

[169]:
```
# Solution

def RunTests(live, iters=1000):
    # Counts live births
    n = len(live)
    # Creates first bron data set
    firsts = live[live.birthord == 1]
    # Creates other data set
    others = live[live.birthord != 1]

    # Comparison of Pregnancy Lengths
    data = firsts.prglngth.values, others.prglngth.values
    ht = DiffMeansPermute(data)
    p1 = ht.PValue(iters=iters)

    # Comparison of Total Weight
    data = (firsts.totalwgt_lb.dropna().values,
            others.totalwgt_lb.dropna().values)
    ht = DiffMeansPermute(data)
    p2 = ht.PValue(iters=iters)

    # Correlation of Age Pregnant and Total Weight
    live2 = live.dropna(subset=['agepreg', 'totalwgt_lb'])
    data = live2.agepreg.values, live2.totalwgt_lb.values
    ht = CorrelationPermute(data)
    p3 = ht.PValue(iters=iters)

    # Chi-Squared of Pregnancy Length
    data = firsts.prglngth.values, others.prglngth.values
    ht = PregLengthTest(data)
    p4 = ht.PValue(iters=iters)
```

```
    print('%d\t\t\t%0.2f\t\t\t%0.2f\t\t%0.2f\t\t\t%0.2f' % (n, p1, p2, p3, p4))
```

[170]:
```
# Solution

live, firsts, others = first.MakeFrames()

n = len(live)
print('Live Births\t', 'Pregnancy Length\t', 'Total Weight\t', 'Age vs. Total␣
 ↪Weight\t', 'Chi-Quared Preg Length')
# Sets up loop and iterations of the loop
for _ in range(7):
    # Creates data
    sample = thinkstats2.SampleRows(live, n)
    # Runs test function
    RunTests(sample)
    # Reduces sample size each time
    n //= 2
```

```
Live Births      Pregnancy Length        Total Weight    Age vs. Total Weight
Chi-Quared Preg Length
9148                    0.18                    0.00            0.00
0.00
4574                    0.40                    0.00            0.00
0.00
2287                    0.40                    0.00            0.00
0.00
1143                    0.61                    0.84            0.02
0.09
571                     0.33                    0.27            0.43
0.30
285                     0.57                    0.45            0.18
0.21
142                     1.00                    0.40            0.34
0.68
```

As sample size decreases, p values become negative. However, for some examples even though size has been decreased. the p value sometimes yields a positive test.

10-1

1 Exercises

Exercise: Using the data from the BRFSS, compute the linear least squares fit for log(weight) versus height. How would you best present the estimated parameters for a model like this where one of the variables is log-transformed? If you were trying to guess someone's weight, how much would it help to know their height?

[171]:
```
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/brfss.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/CDBRFS08.
 ↪ASC.gz")
```

[172]:
```
import brfss

# Creates dataframe
df = brfss.ReadBrfss(nrows=None)
# Creates dataframe drops nulls and only selects two columns
df = df.dropna(subset=['htm3', 'wtkg2'])
# Selects height and weight
heights, weights = df.htm3, df.wtkg2
# Makes weight variable log-transformed
log_weights = np.log10(weights)
```

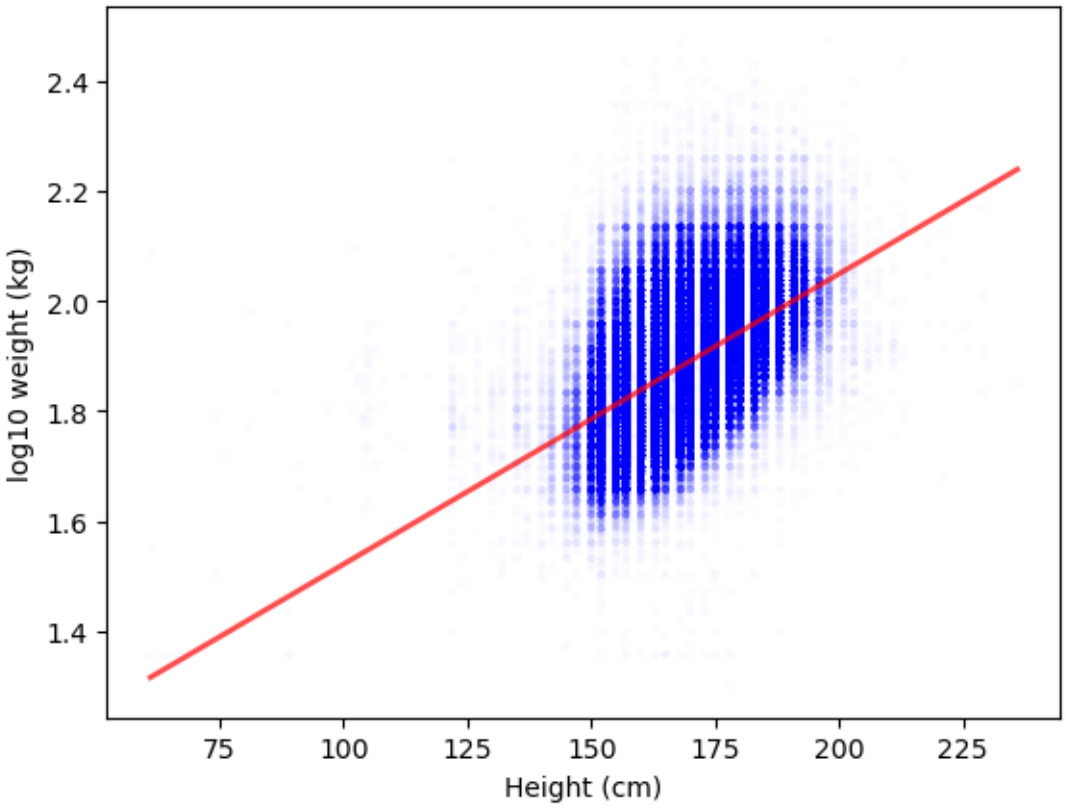
[173]:
```
# Used from chapter 10 code
from thinkstats2 import Mean, MeanVar, Var, Std, Cov

def LeastSquares(xs, ys):
    meanx, varx = MeanVar(xs)
    meany = Mean(ys)

    slope = Cov(xs, ys, meanx, meany) / varx
    inter = meany - slope * meanx

    return inter, slope
```

[174]:
```
inter, slope = thinkstats2.LeastSquares(heights, log_weights)
inter, slope
```

[174]:
```
(0.9930804163917826, 0.005281454169417984)
```

[175]:
```
# Taken from chapter 10 to create line for sequence
def FitLine(xs, inter, slope):
    fit_xs = np.sort(xs)
    fit_ys = inter + slope * fit_xs
    return fit_xs, fit_ys
```

[176]:
```
# Creates scatter plot
thinkplot.Scatter(heights, log_weights, alpha=0.01, s=10)
# Creats line
fxs, fys = thinkstats2.FitLine(heights, inter, slope)
# Displays plot
thinkplot.Plot(fxs, fys, color='red', linewidth=2)
thinkplot.Config(xlabel='Height (cm)', ylabel='log10 weight (kg)', legend=True)
```



These parameters can be shown using a scatter plot to show the data and a fitted line. With this line, you can see the intercept points and estimate the log weight. By having height, you can calculate log weight by doing the formula log_weight = inter + slope * height. An example can be seen below if we know the height to be 150 cm, we can enter it in the formula and now we know the log weight is 1.78 kg.

[177]:
```
log_weights_amount = inter + slope * 150
log_weights_amount
```

[177]:
```
1.78529854180448
```

Like the NSFG, the BRFSS oversamples some groups and provides a sampling weight for each respondent. In the BRFSS data, the variable name for these weights is totalwt. Use resampling, with and without weights, to estimate the mean height of respondents in the BRFSS, the standard error of the mean, and a 90% confidence interval. How much does correct weighting affect the estimates?

[178]:
```
# Used from chapter 10 code
def Summarize(estimates, actual=None):
    mean = Mean(estimates)
    stderr = Std(estimates, mu=actual)
    cdf = thinkstats2.Cdf(estimates)
```

```
    ci = cdf.ConfidenceInterval(90)
    print('mean, SE, CI', mean, stderr, ci)
```

[179]:
```
estimates_unweighted = [thinkstats2.ResampleRows(df).htm3.mean() for _ in␣
↪range(100)]
Summarize(estimates_unweighted)
```

```
mean, SE, CI 168.95604471088743 0.015989720899106562 (168.92717364942703,
168.98185341255888)
```

[180]:
```
def ResampleRowsWeighted(df, column='finalwgt'):
    weights = df[column]
    cdf = thinkstats2.Cdf(dict(weights))
    indices = cdf.Sample(len(weights))
    sample = df.loc[indices]
    return sample
```

[181]:
```
estimates_weighted = [ResampleRowsWeighted(df, 'finalwt').htm3.mean() for _ in␣
↪range(100)]
Summarize(estimates_weighted)
```

```
mean, SE, CI 170.49739877018536 0.01748051355905787 (170.47095737585641,
170.52708219648738)
```