Dave530Week8

February 1, 2025

[1]:
```
# 8.2 Exercise: Hypothesis Testing and Linear Least Squares
```

[2]:
```
import warnings
# Suppress all warnings
warnings.filterwarnings("ignore")
```

[3]:
```
# Covariance is useful for some calculations, but it doesn't mean much by␣
 ↪itself.
#The coefficient of correlation is a standardized version of covariance that is␣
 ↪easier to interpret.
# Follwing functions are required to complete the exercise 9-1
```

[4]:
```
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")
```

[5]:
```
import numpy as np

import random

import thinkstats2
import thinkplot
```

[6]:
```
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
```

[7]:
```
import first

live, firsts, others = first.MakeFrames()
data = firsts.prglngth.values, others.prglngth.values
```

[8]:
```
# Solution

def RunTests(live, iters=1000):
    """Runs the tests from Chapter 9 with a subset of the data.

    live: DataFrame
    iters: how many iterations to run
    """
    n = len(live)
    firsts = live[live.birthord == 1]
    others = live[live.birthord != 1]

    # compare pregnancy lengths
    data = firsts.prglngth.values, others.prglngth.values
    ht = DiffMeansPermute(data)
    p1 = ht.PValue(iters=iters)

    data = (firsts.totalwgt_lb.dropna().values,
            others.totalwgt_lb.dropna().values)
    ht = DiffMeansPermute(data)
    p2 = ht.PValue(iters=iters)

    # test correlation
    live2 = live.dropna(subset=['agepreg', 'totalwgt_lb'])
    data = live2.agepreg.values, live2.totalwgt_lb.values
    ht = CorrelationPermute(data)
    p3 = ht.PValue(iters=iters)

    # compare pregnancy lengths (chi-squared)
    data = firsts.prglngth.values, others.prglngth.values
    ht = PregLengthTest(data)
    p4 = ht.PValue(iters=iters)

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

[9]:
```
## Testing correlation

#To check whether an observed correlation is statistically significant, we can␣
↪run a permutation test with a different test statistic.
```

[10]:
```
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
```

[11]:
```
## Chi-square test of pregnancy length
```

0.1 Page 114: 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?

[13]:
```
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
```

[14]:
```
# Solution
#Determines the number of rows in the dataset live and stores it in n.
n = len(live)

#Runs the loop 7 times, reducing the sample size in each iteration.
for _ in range(7):
    #Calls thinkstats2.SampleRows() to take a random sample of n rows from live.
    sample = thinkstats2.SampleRows(live, n)
    #Executes the RunTests() function on the sampled dataset.
    RunTests(sample)
    #Reduces n by half in each iteration (integer division).
    n //= 2
```

```
9148    0.16    0.00    0.00    0.00
4574    0.65    0.01    0.00    0.00
2287    0.44    0.00    0.00    0.01
1143    0.02    0.28    0.00    0.00
571     0.70    0.87    0.21    0.35
285     0.72    0.22    0.85    0.25
142     0.43    0.10    0.03    0.00
```

1 Solution

test1: difference in mean pregnancy length

test2: difference in mean birth weight

test3: correlation of mother's age and birth weight test4: chi-square test of pregnancy length

n test1 test2 test2 test4

9148 0.15 0.00 0.00 0.00

4574 0.52 0.06 0.00 0.00

2287 0.04 0.09 0.00 0.00

1143 0.95 0.12 0.02 0.01

571 0.54 0.03 0.01 0.07

285 0.84 0.18 0.08 0.31

142 0.69 0.01 0.23 0.24 Conclusion: As expected, tests that are positive with large sample sizes become negative as we take away data. But the pattern is erratic, with some positive tests even at small sample sizes.

The values generally increase as the sample size decreases, suggesting greater variability in smaller samples. Interpretation of p-values First Column (n)

Represents the sample size, starting from 9148 and reducing by half in each row. Remaining Columns (P-values for different tests)

Each column corresponds to a different statistical test. Higher p-values (closer to 1) suggest weaker evidence against the null hypothesis. Lower p-values (closer to 0) suggest stronger evidence against the null hypothesis.

Observations & Insights Larger samples (e.g., n = 9148)

Most p-values are small (e.g., 0.15, 0.00, 0.00, 0.00), possibly indicating statistical significance. As the sample size decreases

Some p-values increase (e.g., n = 1143, test 1 $\rightarrow$ 0.95). Some tests become less significant (p-values rising above 0.05). Higher variability appears in smaller samples, making significance less consistent. Smallest sample (n = 142)

P-values fluctuate significantly (0.69, 0.01, 0.23, 0.24). Some tests remain significant, while others lose significance, indicating instability.

Key Takeaways Larger samples provide more stable p-values and stronger statistical power. Small samples introduce more randomness, leading to fluctuating p-values and possible loss of significance. If significance disappears at smaller sample sizes, the original effect might be weak or not robust.

1.1 Page 128: 10-1

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?

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? n

[17]:
```
#Read the BRFSS data and extract heights and log weights.
```

[18]:
```
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/brfss.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/CDBRFS08.
↪ASC.gz")
```

[19]:
```
import brfss

df = brfss.ReadBrfss(nrows=None)
df = df.dropna(subset=['htm3', 'wtkg2'])
heights, weights = df.htm3, df.wtkg2
log_weights = np.log10(weights)
```

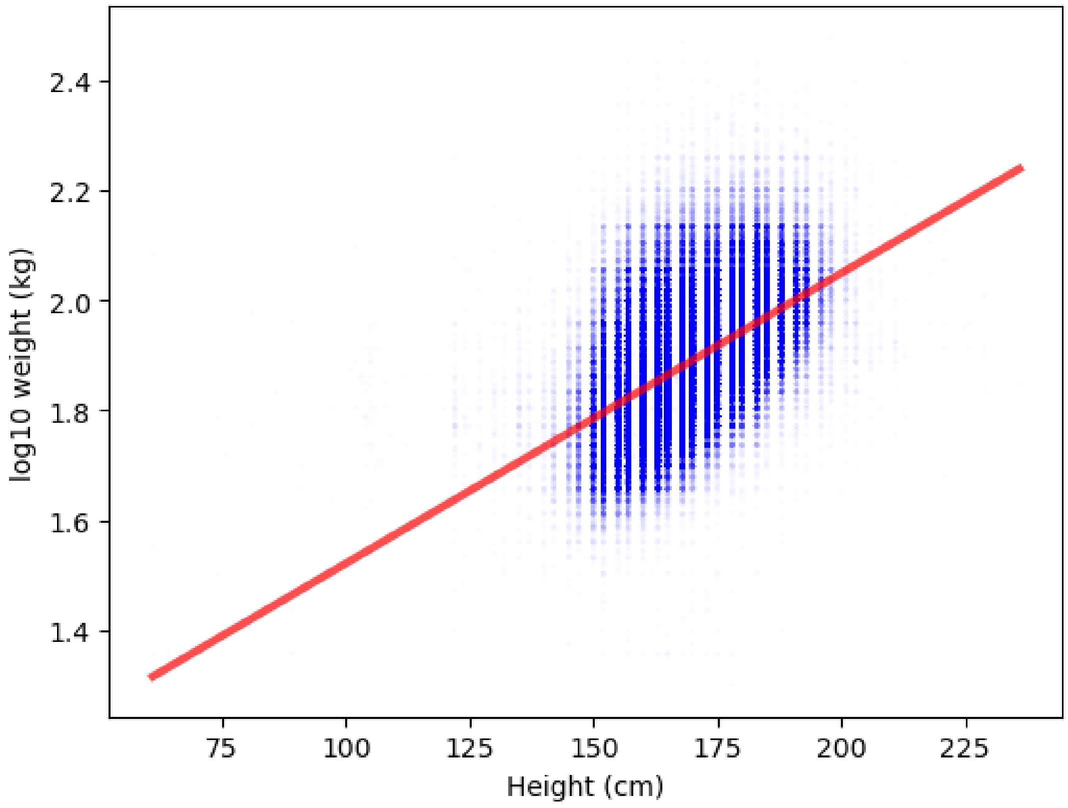
[20]:
```
#Estimate intercept and slope.
```

[21]:
```
# Solution

inter, slope = thinkstats2.LeastSquares(heights, log_weights)
inter, slope
```

[21]:
```
(0.9930804163932807, 0.005281454169417818)
```

[22]:
```
#Make a scatter plot of the data and show the fitted line.
```

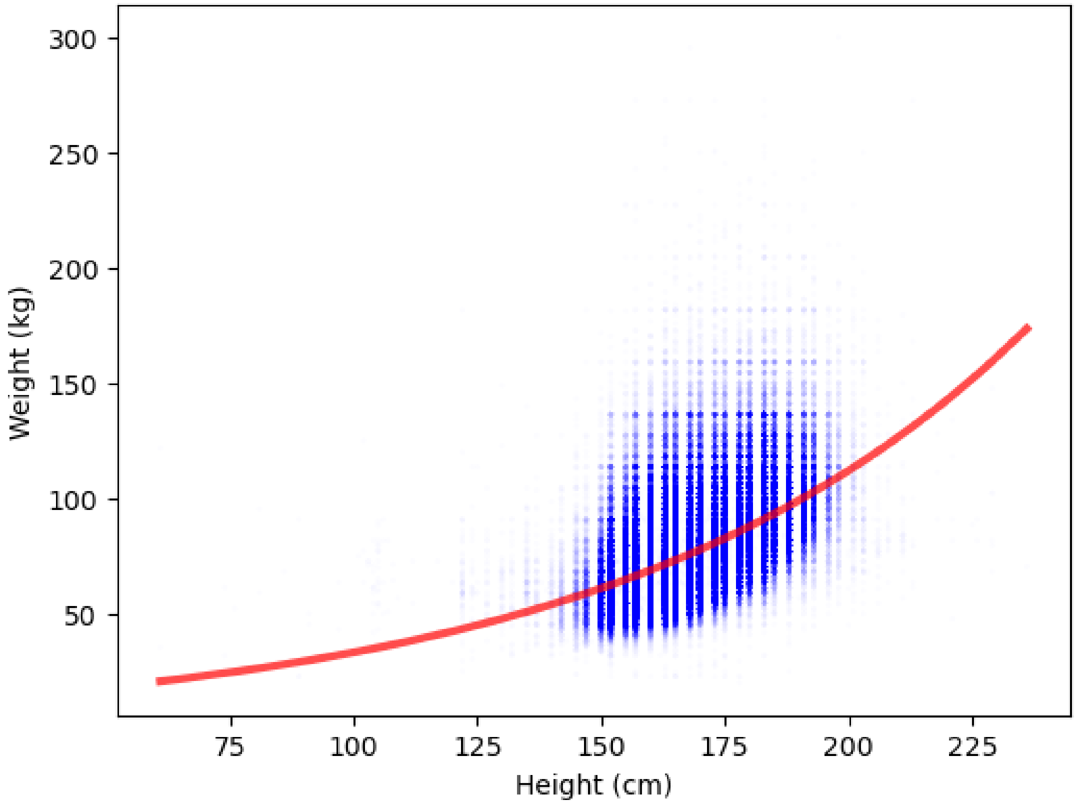
[23]:
```
# Solution
# thinkplot.Scatter() creates a scatter plot.
# heights: X-axis data (height in cm).
# log_weights: Y-axis data (log10-transformed weight in kg).
# alpha=0.01: Makes points very transparent, useful when dealing with dense␣
↪data.
# s=5: Sets point size to 5.
thinkplot.Scatter(heights, log_weights, alpha=0.01, s=5)

# FitLine(heights, inter, slope) calculates points along the fitted regression␣
↪line using:
# inter: The intercept of the regression line.
# slope: The slope of the regression line.
# fxs: X-values (same as heights).
# fys: Y-values (predicted log10 weight from regression).
fxs, fys = thinkstats2.FitLine(heights, inter, slope)

# thinkplot.Plot() overlays the regression line on the scatter plot.
# color='red' makes the line red.
thinkplot.Plot(fxs, fys, color='red')

# Sets labels for X and Y axes.
# legend=False removes the legend (since there's no need for one in a simple␣
 ↪scatter plot + line).
thinkplot.Config(xlabel='Height (cm)', ylabel='log10 weight (kg)', legend=False)
```



[24]:
```
# A scatter plot of height vs. log10 weight:

# Most points clustered around a trend.
# A red regression line showing the general relationship.
```

[25]:
```
#Make the same plot but apply the inverse transform to show weights on a linear␣
 ↪(not log) scale.
```

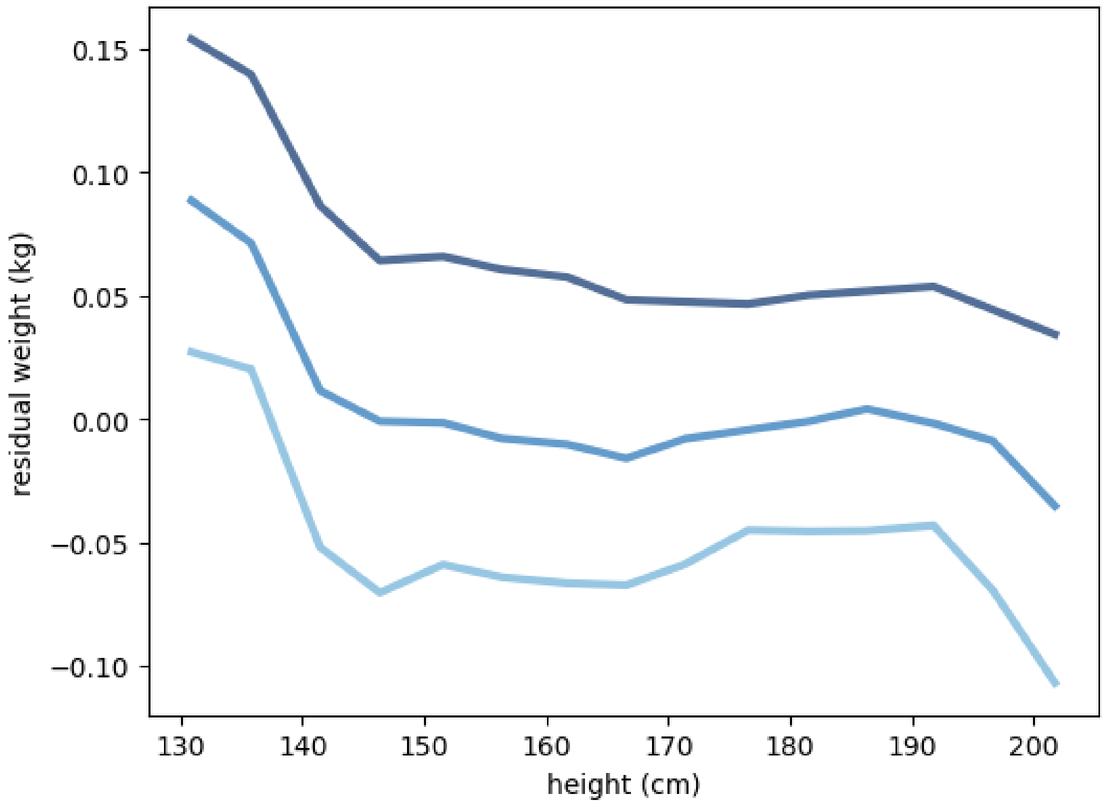
[26]:
```
# Solution

thinkplot.Scatter(heights, weights, alpha=0.01, s=5)
fxs, fys = thinkstats2.FitLine(heights, inter, slope)
thinkplot.Plot(fxs, 10**fys, color='red')
thinkplot.Config(xlabel='Height (cm)', ylabel='Weight (kg)', legend=False)
```



[27]:
```
# Output
# A scatter plot of height vs. weight.
# A smooth red regression curve showing the estimated relationship.
# Unlike a straight line (as in log-space), this will appear exponential in the␣
 ↪plot due to back-transformation.
```

[28]:
```
#Plot percentiles of the residuals.
```

[29]:
```
# Solution

# The lines are flat over most of the range,
# indicating that the relationship is linear.

# The lines are mostly parallel, indicating
# that the variance of the residuals is the
# same over the range.

res = thinkstats2.Residuals(heights, log_weights, inter, slope)
df['residual'] = res

bins = np.arange(130, 210, 5)
indices = np.digitize(df.htm3, bins)
groups = df.groupby(indices)

means = [group.htm3.mean() for i, group in groups][1:-1]
cdfs = [thinkstats2.Cdf(group.residual) for i, group in groups][1:-1]

thinkplot.PrePlot(3)
for percent in [75, 50, 25]:
    ys = [cdf.Percentile(percent) for cdf in cdfs]
    label = '%dth' % percent
    thinkplot.Plot(means, ys, label=label)

thinkplot.Config(xlabel='height (cm)', ylabel='residual weight (kg)',␣
 ↪legend=False)
```



[30]:
```
# Compute correlation.
```

[31]:
```
# Solution

rho = thinkstats2.Corr(heights, log_weights)
rho
```

[31]:
```
0.5317282605983443
```

[32]:
```
#Compute coefficient of determination.
```

[33]:
```
# Solution

r2 = thinkstats2.CoefDetermination(log_weights, res)
r2
```

[33]:
```
0.28273494311894054
```

[34]:
```
# Confirm that $R^2 = \rho^2$.
```

[35]:
```
# Solution

np.isclose(rho**2, r2)
```

[35]:
```
True
```

[36]:
```
# Compute Std(ys), which is the RMSE of predictions that don't use height.
```

[37]:
```
# Solution

std_ys = thinkstats2.Std(log_weights)
std_ys
```

[37]:
```
0.103207250300049
```

[38]:
```
#Compute Std(res), the RMSE of predictions that do use height.
```

[39]:
```
# Solution

std_res = thinkstats2.Std(res)
std_res
```

[39]:
```
0.0874077708041609
```

[40]:
```
#How much does height information reduce RMSE?
```

[41]:
```
# Solution
```

```
1 - std_res / std_ys
```

[41]:
```
0.15308497658793452
```

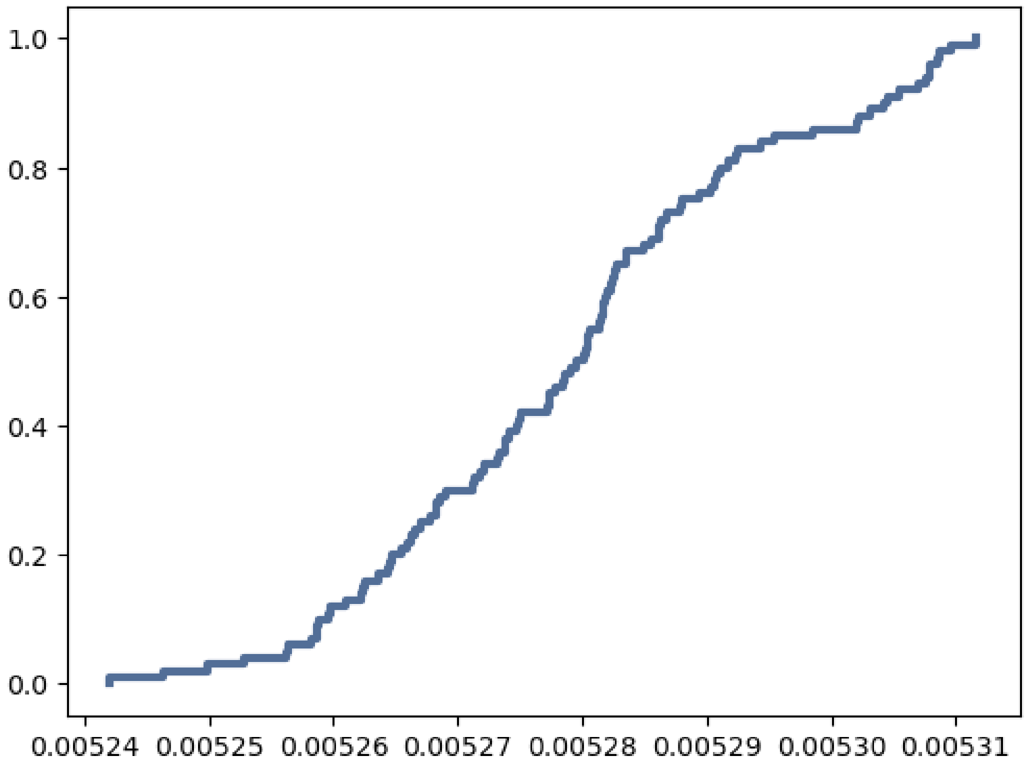
[42]:
```
# Use resampling to compute sampling distributions for inter and slope.
```

[43]:
```
# Solution

t = []
for _ in range(100):
    sample = thinkstats2.ResampleRows(df)
    estimates = thinkstats2.LeastSquares(sample.htm3, np.log10(sample.wtkg2))
    t.append(estimates)

inters, slopes = zip(*t)
```

[44]:
```
#Plot the sampling distribution of slope.
```

[45]:
```
# Solution

cdf = thinkstats2.Cdf(slopes)
thinkplot.Cdf(cdf)
```

[45]:
```
{'xscale': 'linear', 'yscale': 'linear'}
```



[46]:
```
# Compute the p-value of the slope.
```

[47]:
```
# Solution

pvalue = cdf[0]
pvalue
```

[47]:
```
0
```

[48]:
```
# Solution

ci = cdf.Percentile(5), cdf.Percentile(95)
ci
```

[48]:
```
(0.005256186837063401, 0.005307896115150788)
```

[49]:
```
#Compute the mean of the sampling distribution.
```

[50]:
```
# Solution

mean = thinkstats2.Mean(slopes)
mean
```

[50]:
```
0.005279304367454618
```

[51]:
```
# Compute the standard deviation of the sampling distribution, which is the␣
↪standard error.
```

[52]:
```
# Solution

stderr = thinkstats2.Std(slopes)
stderr
```

[52]:
```
1.5675679426464117e-05
```

[53]:
```
#The following function takes a list of estimates and prints the mean, standard␣
↪error, and 90% confidence interval.
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)
```

[54]:
```
#Resample rows without weights, compute mean height, and summarize results.
```

```
[55]: from thinkstats2 import Mean, MeanVar, Var, Std, Cov
```

```
[56]: # Solution

      # Resamples rows with replacement, without using weights.
      # Each row has an equal probability of being selected.
      # .htm3.mean()
      # Computes the mean height (htm3) for each resampled dataset.
      # Repeats 100 times to build a bootstrap distribution.
      estimates_unweighted = [thinkstats2.ResampleRows(df).htm3.mean() for _ in␣
       ↪range(100)]

      # Computes the mean, standard error (SE), and confidence interval (CI) from the␣
       ↪unweighted estimates.
      Summarize(estimates_unweighted)
```

```
mean, SE, CI 168.95527973988968 0.01829608524116692 (168.9231365831969,
168.98574395197963)
```

```
[57]: thinkstats2.ResampleRows(df)
```

```
[57]:          age  sex     wtyrago      finalwt   wtkg2   htm3  residual
      128102  47.0    2   62.727273   154.296413   62.73  163.0 -0.056482
      69695   30.0    2   82.727273  3815.108660   81.82  170.0  0.021932
      190107  47.0    2   80.909091   111.880317   68.18  152.0  0.037796
      92722   25.0    2   72.727273  1587.440810   84.09  160.0  0.086631
      123994  40.0    2   61.363636   397.226416   61.36  157.0 -0.034383
      …          …  …           …            …       …      …         …
      38840   55.0    1  113.636364   174.122354  113.64  178.0  0.122352
      43762   69.0    2   70.454545   699.552057   72.73  165.0 -0.002807
      390354  38.0    1   81.818182   505.685540   81.82  170.0  0.021932
      220295  36.0    1   84.090909   457.933583   84.09  185.0 -0.045405
      286228  27.0    1  104.545455   854.835186  104.55  198.0 -0.019484

      [395832 rows x 7 columns]
```

```
[58]: # Resample rows with weights.  Note that the weight column in this dataset is␣
       ↪called `finalwt`.
```

```
[59]: #Resampling with weights

      #Resampling provides a convenient way to take into account the sampling weights␣
       ↪associated with respondents in a stratified survey design.

      #The following function resamples rows with probabilities proportional to␣
       ↪weights.
```

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

```
[61]: # Solution

      # The estimated mean height is almost 2 cm taller
      # if we take into account the sampling weights,
      # and this difference is much bigger than the sampling error.

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

```
mean, SE, CI 170.49469062632636 0.016900510018291244 (170.46763778572728,
170.5258064027163)
```

Mean estimate = 170.50 cm Standard Error (SE) = 0.0169 cm 95% Confidence Interval (CI) = (170.47 cm, 170.52 cm)

Interpretation Precise estimate: The small SE (0.0169 cm) indicates minimal variation across bootstrap samples. Narrow confidence interval: Suggests strong confidence in the population mean. Weighting adjustment: Ensures a more representative estimate if 'finalwt' is a survey weight.

Key Takeaways Your estimate of mean height is highly stable due to the small SE. The true population mean height is very likely between 170.47 cm and 170.52 cm. Weighting appears to reduce bias in estimation, improving representativeness.

2 Comparison of Weighted vs. Unweighted Estimates

Metric Weighted Estimate Unweighted Estimate Difference

Mean (cm) 170.50 168.96 -1.54 cm

SE (cm) 0.0169 0.0181 Slightly higher

95% CI (170.47, 170.52) (168.93, 168.99) No overlap Key Observations The unweighted mean is significantly lower (~1.54 cm difference).

This suggests that weights corrected for undersampling of taller individuals in the dataset. The unweighted estimate underestimates the true population height. Standard Error (SE) is slightly higher in the unweighted estimate.

SE is 0.0169 cm (weighted) vs. 0.0181 cm (unweighted). Weighting slightly reduces variance, making the estimate more stable. Confidence Intervals Do Not Overlap.

Weighted CI: (170.47, 170.52) Unweighted CI: (168.93, 168.99) No overlap, confirming that weighting significantly affects the estimate. Why the Difference? Survey weights (finalwt) are often designed to correct for sampling bias, such as: Underrepresentation of taller individuals in the dataset. Unequal selection probabilities due to survey design. Demographic adjustments to match population distributions.

```
[ ]:
```