Reppeto530Week8

February 4, 2024

0.1 Chapter 9

Brian Reppeto 530 Prof. Jim Week 8 HW Exercise 9-1

[1]:
```
# import libraries

import numpy as np

import pandas as pd
import thinkstats2
import thinkplot
```

[2]:
```
# import first.py file and dropna's from agepreg and ttl wgt

import first

live, firsts, others = first.MakeFrames()
live = live.dropna(subset=['agepreg', 'totalwgt_lb'])
```

[3]:
```
# creat a hypothesis test class from thinkstats2.HypothesisTest

class DiffMeansPermute(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):  # the test statistic is the difference␣
 ↪between the means of the two groups
        group1, group2 = data
        test_stat = abs(group1.mean() - group2.mean())
        return test_stat

    def MakeModel(self): # set up the model for the hypothesis test
        group1, group2 = self.data
        self.n, self.m = len(group1), len(group2)
        self.pool = np.hstack((group1, group2))

    def RunModel(self):   #  simulate a random transformation of the pooled data
        np.random.shuffle(self.pool)
        data = self.pool[:self.n], self.pool[self.n:]
        return data
```

[4]:
```
# create a hypothesis test class that inherits from the thinkstats2.
 ↪HypothesisTest
# the purpose of this class is to perform a hypothesis test for the difference␣
 ↪in means between two groups

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
```

[5]:
```
# create a class named PregLengthTest that inherits from the thinkstats2.
 ↪HypothesisTest
# this class is to perform a hypothesis test for the distribution of pregnancy␣
 ↪lengths
# in two groups (firstborns and others) using a chi-squared test.

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
```

[6]:
```
# create a function to perform hypothesis tests on pregnancy-related data

def RunTests(live, iters=1000):

    n = len(live)
    firsts = live[live.birthord == 1]
    others = live[live.birthord != 1]

    # compare the pregnancy lengths between firstborns and others using the␣
 ↪DiffMeansPermute class

    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 the correlation between the age of pregnancy and birth weight using␣
 ↪the CorrelationPermute class

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

    # compare the distribution of pregnancy lengths between firstborns and␣
 ↪others using the PregLengthTest class

    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))
```

[7]:
```
# this code is to observe how the p-values of the hypothesis tests change as␣
 ↪the sample size decreases

n = len(live)
for _ in range(7):
    sample = thinkstats2.SampleRows(live, n)
```

```
    RunTests(sample)
    n //= 2
```

```
9038    0.19    0.00    0.00    0.00
4519    0.68    0.00    0.00    0.00
2259    0.71    0.01    0.00    0.00
1129    0.93    0.82    0.10    0.15
564     0.02    0.54    0.63    0.00
282     0.76    0.30    0.05    0.07
141     0.25    0.95    0.76    0.24
```

Conclusion:

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	test3	test4
9148	0.16	0.00	0.00	0.00
4574	0.10	0.01	0.00	0.00
2287	0.25	0.06	0.00	0.00
1143	0.24	0.03	0.39	0.03
571	0.81	0.00	0.04	0.04
285	0.57	0.41	0.48	0.83
142	0.45	0.08	0.60	0.04

As anticipated, tests that initially show statistical significance with large sample sizes tend to lose significance as the sample size decreases. However, the pattern is not entirely consistent, and some tests continue to exhibit significance even with smaller sample sizes. The results highlight the impact of sample size on the stability and reliability of the conducted hypothesis tests.

Exercise 10-1

[8]:
```
# import files

import brfss
from thinkstats2 import Mean, MeanVar, Var, Std, Cov
```

[9]:
```
# create a function that takes a collection of estimates and an optional actual␣
 ↪value as input
# the purpose of this function is to compute and print a summary of the␣
 ↪estimates, including the mean,
# standard error, and a confidence interval at a 90% confidence level

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)
```

[10]:
```
# create a function to perform a weighted resampling of rows from a df

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

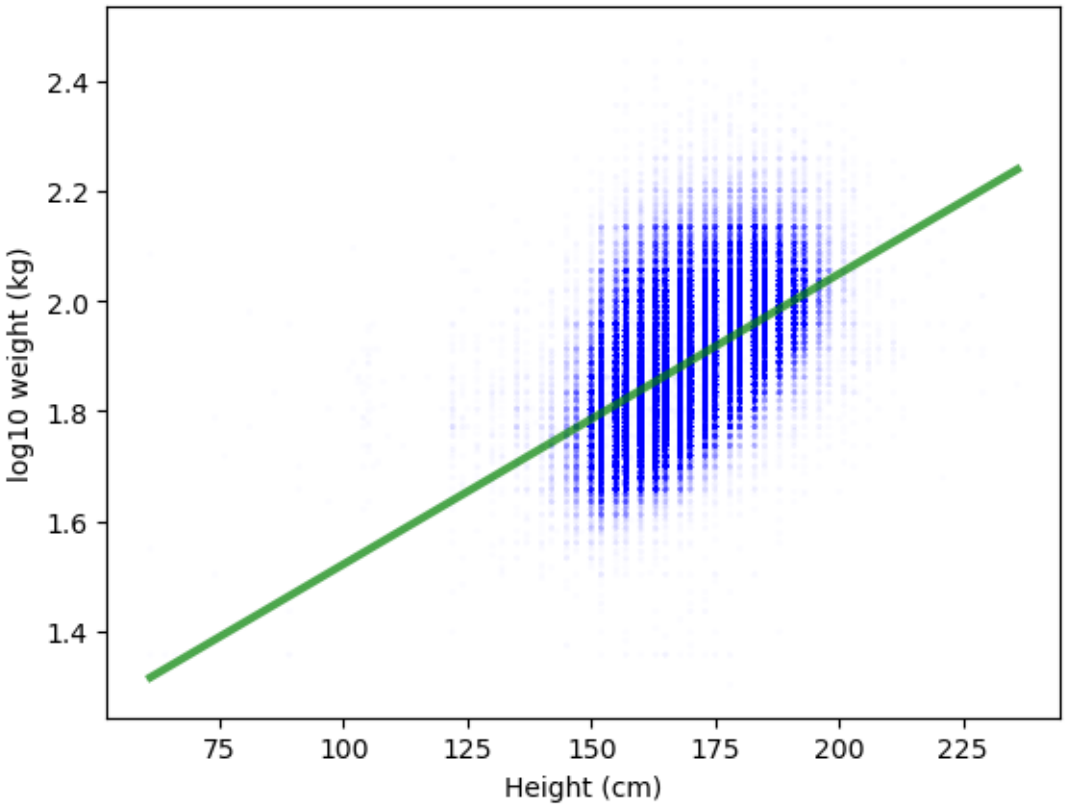
[11]:
```
# read file

df = brfss.ReadBrfss(nrows=None)
```

[12]:
```
# remove rows where either the 'htm3' or 'wtkg2' column has missing values NaN
# extract the 'htm3' and 'wtkg2' columns into separate variables
# compute the log of the weight values

df = df.dropna(subset=['htm3', 'wtkg2'])
heights, weights = df.htm3, df.wtkg2
log_weights = np.log10(weights)
```

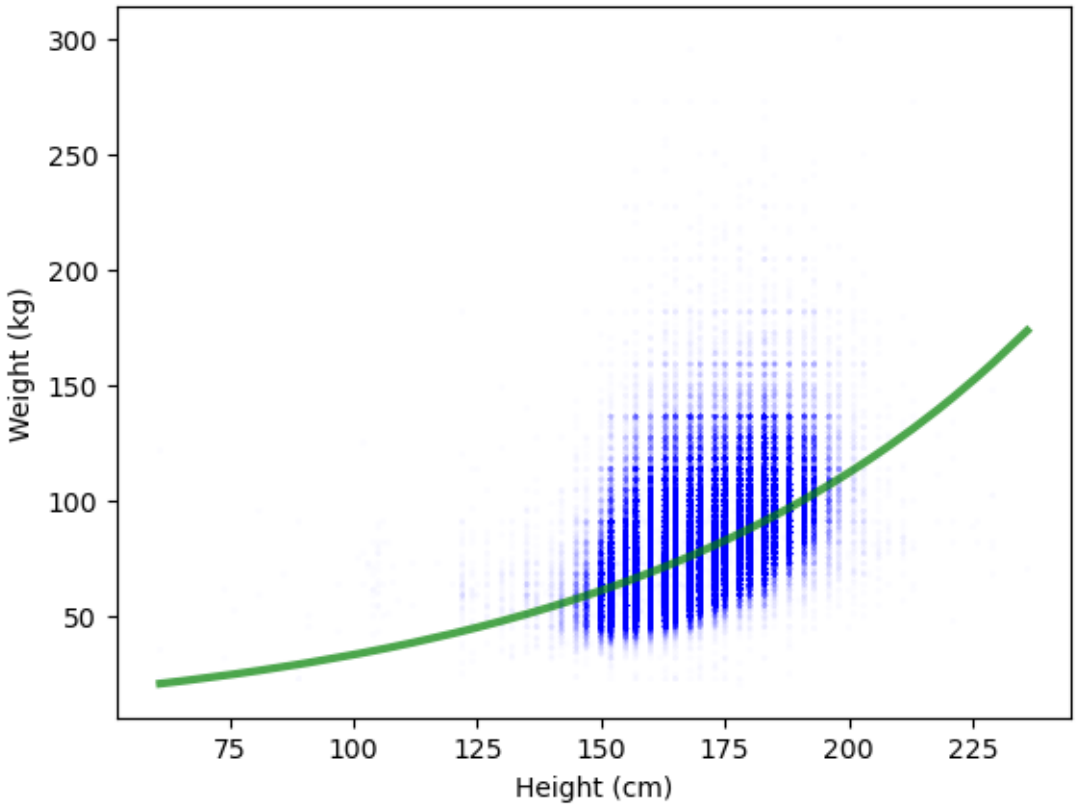
[13]:
```
# use the leastsquares function to perform linear regression

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

[13]:
```
(0.9930804163932894, 0.005281454169417767)
```

[14]:
```
# create a scatter plot of the relationship between heights and the log of␣
↪weights

thinkplot.Scatter(heights, log_weights, alpha=0.01, s=5)
fxs, fys = thinkstats2.FitLine(heights, inter, slope)
thinkplot.Plot(fxs, fys, color='green')
thinkplot.Config(xlabel='Height (cm)', ylabel='log10 weight (kg)', legend=False)
```

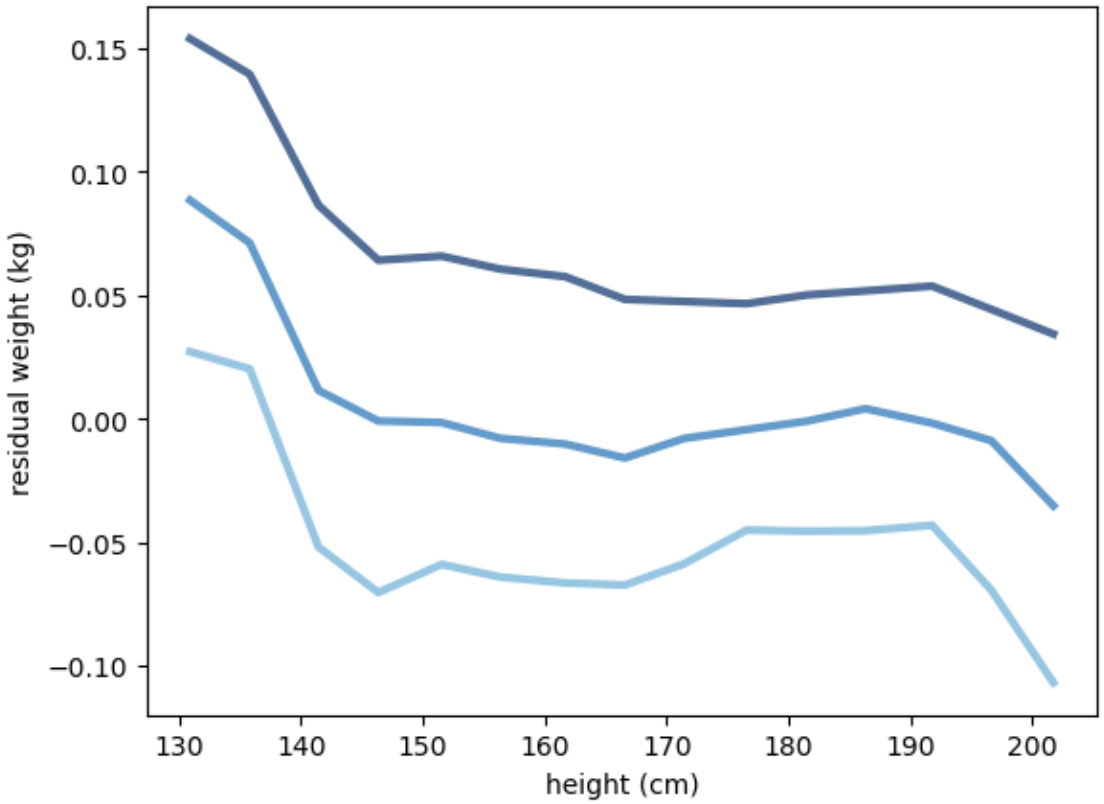


[15]:
```
# create a scatter plot of the relationship between heights and the log of␣
↪weights

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



[16]:
```
# perform a residual analysis on the linear regression model fit to the␣
 ↪relationship
# between heights and the log of weights


"""
The lines appear to remain level, suggesting a linear relationship
The lines exhibit a consistent parallel orientation, suggesting that the␣
 ↪variability
in residuals remains uniform across the entire 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)
```



[17]:
```
# calc the Pearson correlation coefficient between 'heights' and the log of␣
 ↪'weights'

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

[17]: 0.531728260598403

[18]:
```
# calculates the coefficient of determination
# asses how well the log of weights predicts the res

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

[18]: 0.2827349431187015

[19]:
```
# check whether the square of the Pearson corr coef is approx = to the coef of␣
 ↪determination


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

[19]: True

[20]:
```
# calc the stand dev of the log of weights

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

[20]: 0.10320725030003608

[21]:
```
# calc the stand dev of the residuals obtained from a linear regression model

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

[21]: 0.08740777080416452

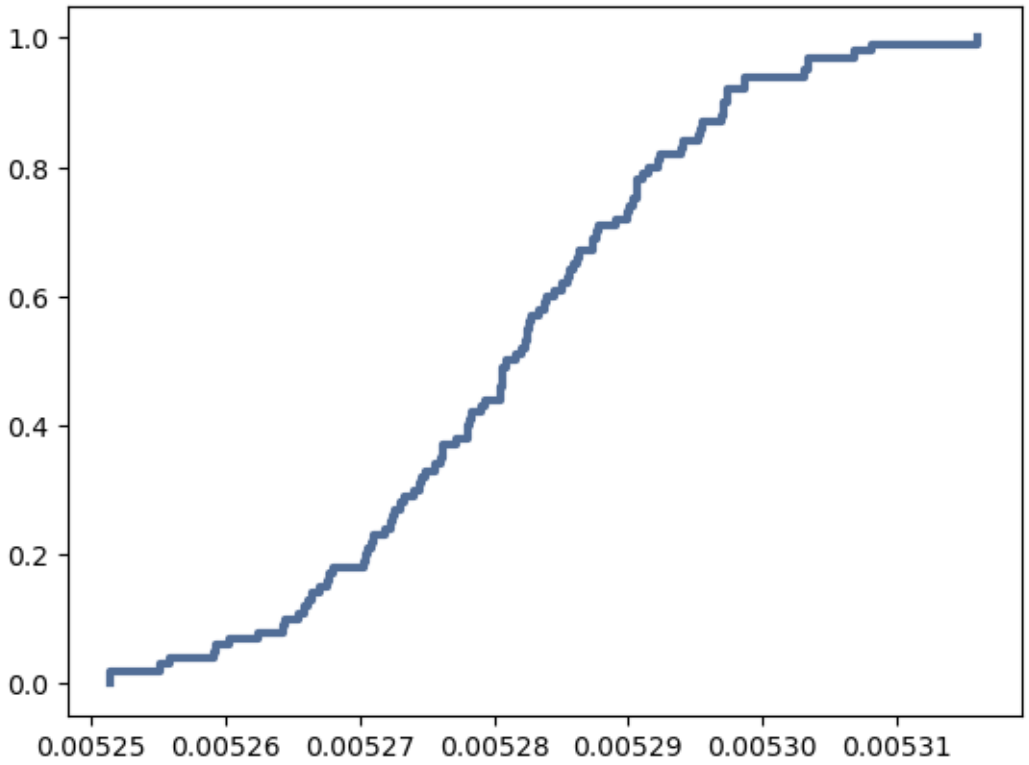
[22]:
```
# calc provides a measure of how well the linear regression model accounts for␣
 ↪the variability in the dependent variable
# a higher value indicating less explanatory power

1 - std_res / std_ys
```

[22]: 0.1530849765877934

[23]:
```
# bootstrapping to estimate the distribution of the intercepts and slopes of␣
 ↪the linear regression model

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)
```

[24]:
```
# create a CDF plot for the slopes

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

[24]:
```
{'xscale': 'linear', 'yscale': 'linear'}
```



[25]:
```
# calc a pvalue based on the CDF of slopes

pvalue = cdf[0]
pvalue
```

[25]:
```
0
```

[26]:
```
# calc a 90% confidence interval for the distribution of slopes

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

[26]:
```
(0.005259150421497286, 0.00530317157322009)
```

[27]:
```
# calc the mean of the slopes

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

[27]: 0.005281162488348682

[28]:
```
# calc the standard error of the slopes

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

[28]: 1.289897564960053e-05

[29]:
```
# estimate the distribution of mean heights from the original dataset

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

```
mean, SE, CI 168.95591642413953 0.015940631038355333 (168.92819428444392,
168.98036793387092)
```

[ ]:
```
# weighted resampling to estimate the distribution of mean heights (htm3) from␣
 ↪the original dataset

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

The mean height estimate increases by nearly 2 cm when considering the sampling weights, and this difference significantly surpasses the impact of sampling error.

[ ]:

[ ]: