DSC530-302 Data Exploration and Analysis

Author: Chitramoy Mukherjee

Date: 05/04/2023

Title: "DSC530-302 Week-05 Assignment-9.1 and 10.1"

In [6]:
```
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")

import numpy as np

import random

import thinkstats2
import thinkplot
```

Exercise 9.1 : 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?

In [7]:
```
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"
)

import first

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

In [10]:
```
class DiffMeansPermute(thinkstats2.HypothesisTest):

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

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

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

In [14]:
```
# define class CorrelationPermute

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
```

In [17]:
```
# define class PregLengthTest

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
```

In [18]:
```
def RunTests(live, iters=1000):
    n = len(live)
```

```
# Identify first and others from live dataset

    firsts = live[live.birthord == 1]
    others = live[live.birthord != 1]

    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))
```

In [21]:

```
n = len(live)
for _ in range(7):
    sample = thinkstats2.SampleRows(live, n)
    RunTests(sample)
    n //= 2
```

```
9148    0.16    0.00    0.00    0.00
4574    0.75    0.01    0.00    0.00
2287    0.77    0.00    0.00    0.01
1143    0.39    0.01    0.02    0.00
571     0.34    0.16    0.07    0.16
285     0.86    0.34    0.54    0.16
142     0.70    0.17    0.01    0.83
```

In [ ]:

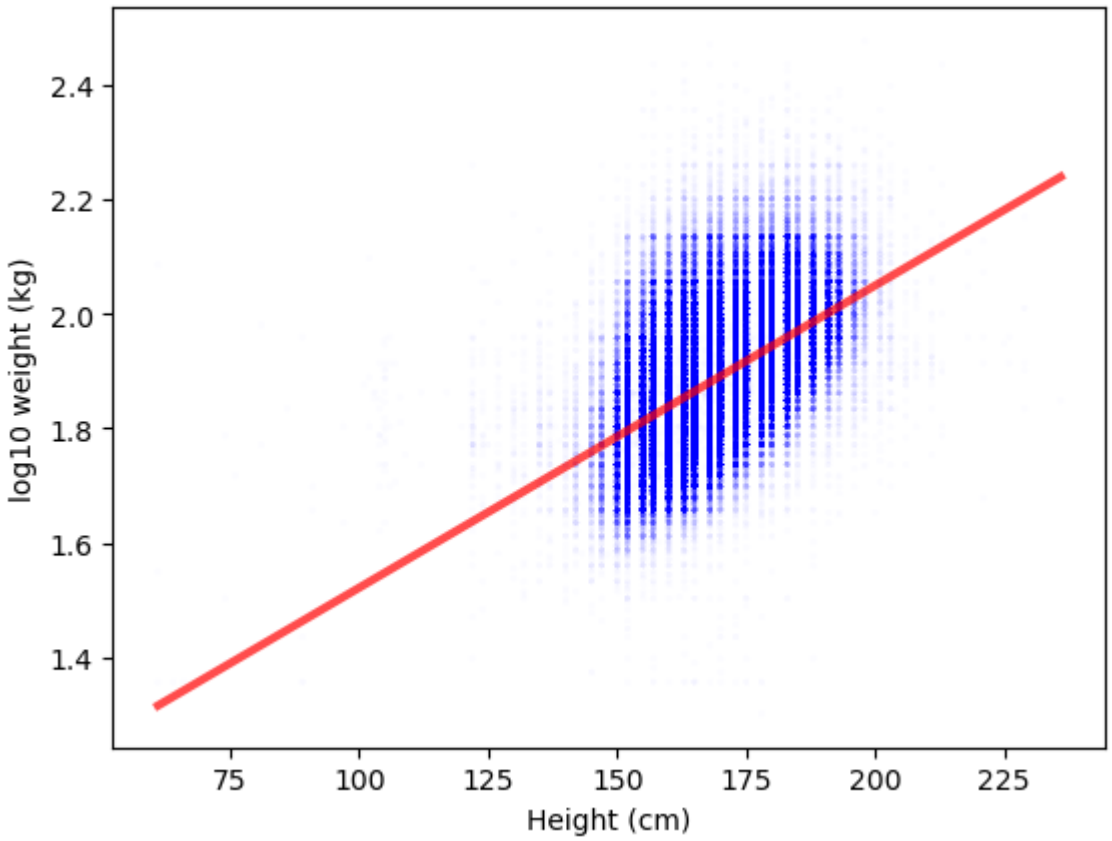
```
# Solution

# Results :

# 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  0.16    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 we reduce the data volume, the test result become negative from positive.
# Pattern is erratic, with some positive tests even small sample size.
```

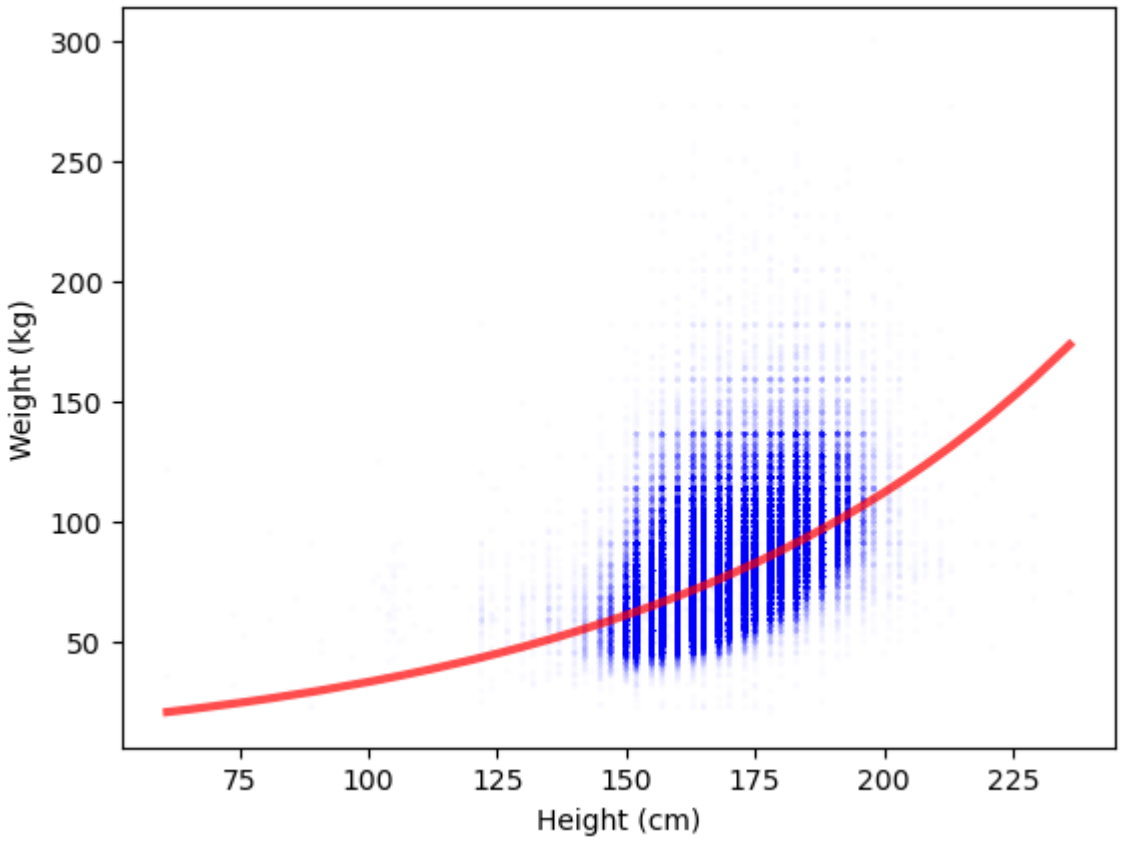
Exercise 10.1 Using the data from the BRFSS, compute the linear least squares fit for log(weight) versus height. How would you best present theestimated parameters for a model like this where one of the variables is log transformed? If you were trying to guess someone's weight, how much wouldit help to know their height? Like the NSFG, the BRFSS oversamples some groups and provides a samplingweight for each respondent. In the BRFSS data, the variable name for these weights is finalwt. Use resampling, with and without weights, to estimatethe mean height of respondents in the BRFSS, the standard error of the mean, and a 90% confidence interval. How much does correct weightingaffect the estimates?

In [22]:
```
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/brfss.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/CDBRFS08.ASC.gz")

import brfss

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

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

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

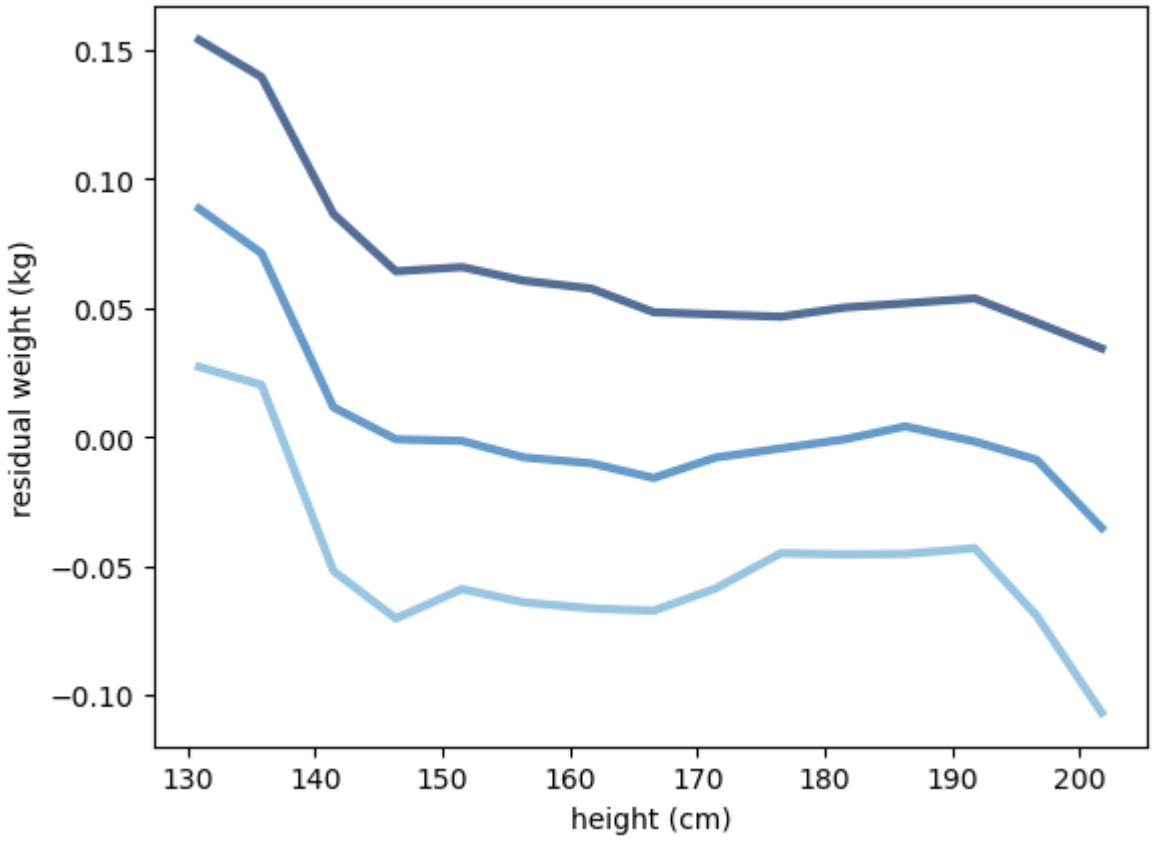
```
Downloaded brfss.py
Downloaded CDBRFS08.ASC.gz
```



In [23]:
```
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)
```



In [24]:
```
# Solution

# As the lines are flat over most of the rnage it indicates the ralationship linear.

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)
```



In [25]:
```
# Compute correlation

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

Out[25]:
```
0.5317282605983439
```

In [26]:
```
# Compute coefficient of determination.
```

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

Out[26]:
```
0.2827349431189401
```

In [27]:
```
# Compare the rho with r2

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

Out[27]:
```
True
```

In [28]:
```
# Compute Std(ys) which is the RMSE of Predications without using height

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

Out[28]:
```
0.10320725030004894
```

In [29]:
```
# Compute Std(ys) which is the RMSE of Predications using height

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

Out[29]:
```
0.08740777080416089
```

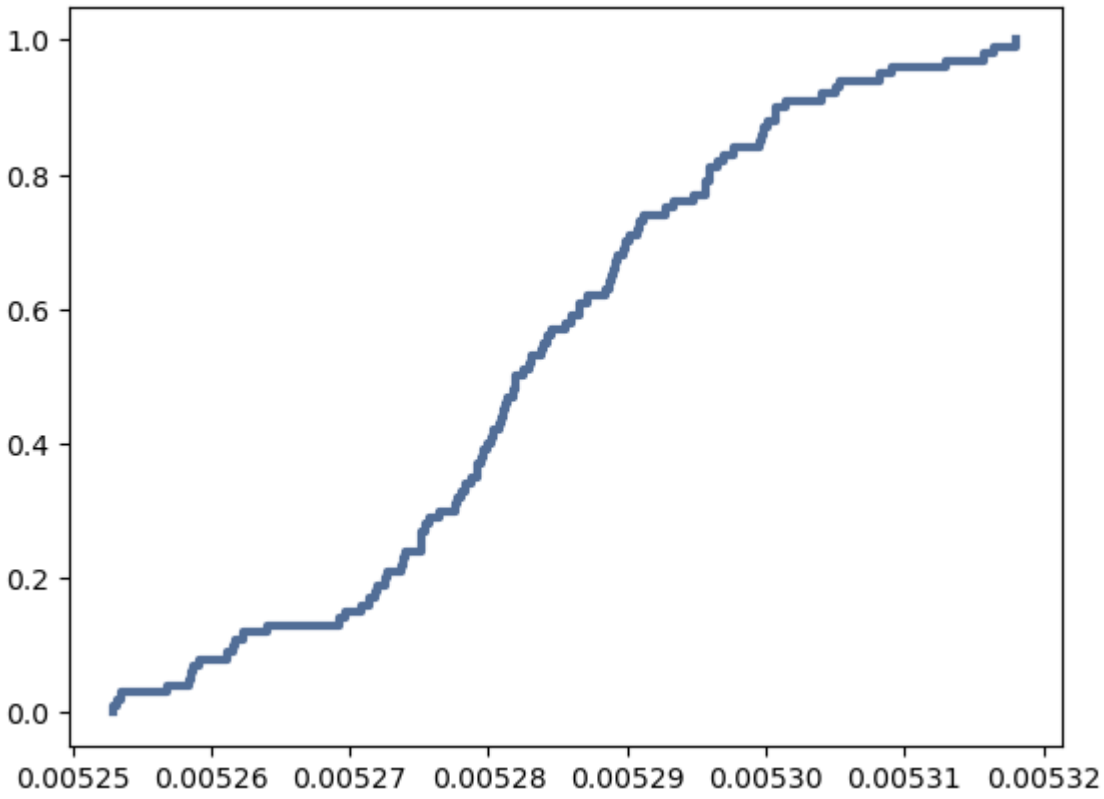
In [30]:
```
# How much height info rediuces RMSE

1 - std_res / std_ys
```

Out[30]:
```
0.15308497658793419
```

In [31]:
```
# Resampling to compute sampling distributions for inter and slope.

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)
```

In [32]:
```
# Sampling plot for distribution of slope

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

Out[32]:
```
{'xscale': 'linear', 'yscale': 'linear'}
```



In [33]:
```
# Compute p value

pvalue = cdf[0]
pvalue
```

Out[33]:
```
0
```

In [34]:
```
# Compute the 90% confidence interval of slope

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

Out[34]:
```
(0.005258367180278052, 0.005308233535690939)
```

In [36]:
```
# Mean of sd

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

Out[36]:
```
0.0052835293886798395
```

In [37]:
```
# Compute standard deviation of sd

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

Out[37]:
```
1.4531878772626838e-05
```

In [41]:
```
#define summarize

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)

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

In [42]:
```
# Resample rows without weights, compute mean height, and summarize results

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

```
mean, SE, CI 168.95833899229976 0.017392388656227505 (168.93123850522443, 168.9851982
65931)
```

In [44]:
```
# define ResampleRowsWeighted

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

In [45]:
```
# Solution

# Estimated mean height is almost 2 cm taller if we consider the sampling weights and
# bigger than the sampling error.

estimates_weighted = [ResampleRowsWeighted(df, 'finalwt').htm3.mean() for _ in range(1
Summarize(estimates_weighted)
```

```
mean, SE, CI 170.49718446209502 0.017711801793396924 (170.4694365286283, 170.52614745
64967)
```