Theodore Koby-Hercsky
DSC530-T302

7.2 Exercises

Exercises 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?

In [147]:
```
# Solution goes here

def Preg_solution(live, iters=2000):

    #First we are using the live dataframe and setting firsts and others to birth order.
    n = len(live)
    firsts = live[live.birthord == 1]
    others = live[live.birthord != 1]

    # As seen below in the data output we see that in the first
    # column has the 'n' which represents our random sample size.

    # Next test the correlation between the birth legth and monthers age.
    cor_live = live.dropna(subset=['prglngth', 'agepreg'])
    data = cor_live.prglngth.values, cor_live.agepreg.values
    ht = CorrelationPermute(data)
    p_value1 = ht.PValue(iters=iters)

    # As seen below in the second column the p-values for the
    # correlation between the birth legth and monthers age which shows
    # a positive output throughout each random test but changes from
    # higher to lower throughout the sample sizes.

    # Next I compared the age of the mother during pregnancy of first and others.
    data = firsts.agepreg.values, others.agepreg.values
    ht = DiffMeansPermute(data)
    p_value2 = ht.PValue(iters=iters)

    # In our third column we have compared the age of the mother
    # during pregnancy of first and others babbies born which shows a
    # 0.00 out put until sample size 35 which shows a p-value of 0.08
    # then last sample row shows 0.33.

    # Next I use the Chi-square test of pregnancy length
    # to compare the pregnancy lengths with chi-squared.
    data = firsts.prglngth.values, others.prglngth.values
    ht = PregLengthTest(data)
    p_value3 = ht.PValue(iters=iters)

    # In our last we have the p-value of the Chi-square test of
    # pregnancy length which gathers Chi-square data from IN[94] which
    # shows a p-value of 0.00 until we reach random sample 1143 which
    # shows 0.01 and goes up and down till our ninth random sample of
    # 35 which is back at 0.00.

    print('%d\t%0.2f\t%0.2f\t%0.2f' % (n, p_value1, p_value2, p_value3))
```

In [148]:
```
# Solution goes here
n = len(live)
for _ in range(10):
    think_rows = thinkstats2.SampleRows(live, n)
    Preg_solution(think_rows)
    n //= 2
```

```
9148    0.29    0.00    0.00
4574    0.27    0.00    0.00
2287    0.90    0.00    0.00
1143    0.54    0.00    0.01
571     0.23    0.00    0.11
285     1.00    0.00    0.29
142     0.57    0.00    0.31
71      0.15    0.00    0.23
```

```
<ipython-input-94-7ea16e8fdd92>:26: RuntimeWarning: invalid value encountered in true_divid
e
  stat = sum((observed - expected)**2 / expected)
```

```
35      0.36    0.08    0.00
17      0.44    0.33    0.00
```

In [26]:
```
# What happens to the p-values of these tests as sample size decreases?
""" As for the p-values of these tests as sample size decreases we see
that the correlation p-value shows for the most part excluding random
sample 285 we have a downward positive p-value as the sample size goes
down. But in regards to the last two columns we see alot of p-values
at 0.00 with positive valuse from random sample 1143 to 35 in either
last two columns.
"""
# What is the smallest sample size that yields a positive test?
""" The smallest sample size we see that yields a positive test is
sample size 17 which shows a positive 0.07 for the correlation
between the birth legth and monthers age.
"""
```

Exercises 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?

Read the BRFSS data and extract heights and log weights.

In [40]:
```
import brfss

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

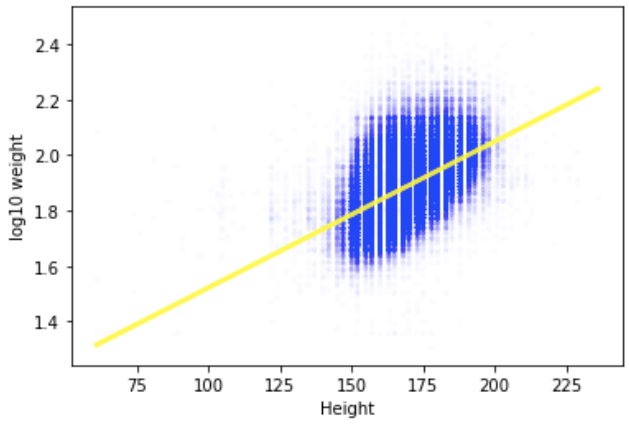
Estimate intercept and slope.

In [48]:
```
# First I will estimate the inercept and slope of the log(weight)
# versus height by setting the intercept and slope equal to the
# thinkstats2. leastsquares for our datas heights and log weight as
# seen in line In [5] above which was similar to this line of code.

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

Out[48]: (0.993080416391812, 0.005281454169417809)

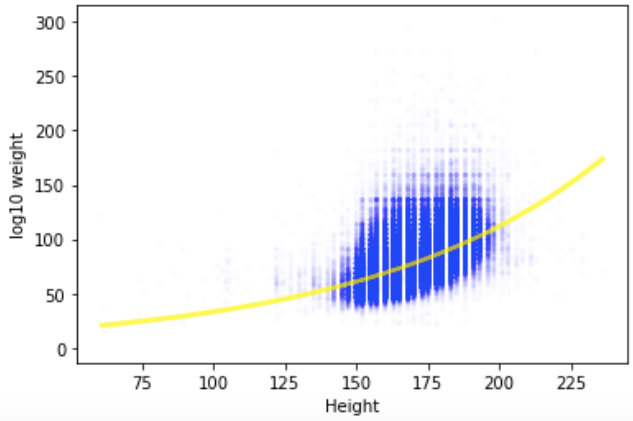
Make a scatter plot of the data and show the fitted line.

In [68]:
```
# First we will define our fit the line which will allow us to
#fit the line for our heights and log_weights then create a scatter
# plot from our findings

def FitLine(heights, inter, slope):
    fxs = np.sort(heights)
    fys = inter + slope * fxs
    return fxs, fys

fxs, fys = FitLine(heights, inter, slope)

thinkplot.Scatter(heights, log_weights, alpha=0.01, s=10)
thinkplot.Plot(fxs, fys, color='white', linewidth=3)
thinkplot.Plot(fxs, fys, color='yellow')
thinkplot.Config(xlabel='Height',
                 ylabel='log10 weight',
                 legend=False)
```



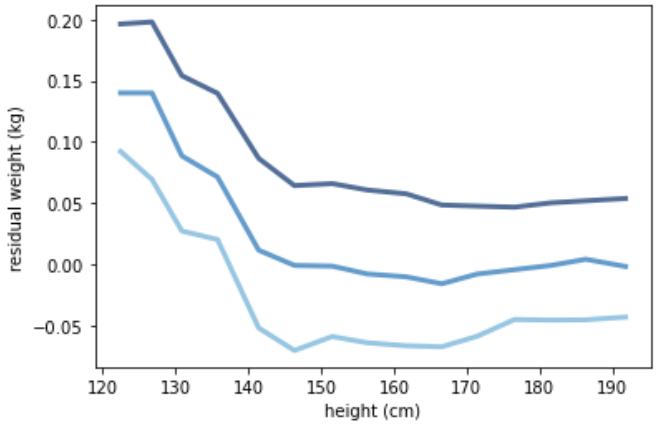
Make the same plot but apply the inverse transform to show weights on a linear (not log) scale.

In [72]:
```
# First we will define our fit the line which will allow us to
#fit the line for our heights and weights  and set the 10**fys then crea
# plot from our findings as seen below

def FitLine(heights, inter, slope):
    fxs = np.sort(heights)
    fys = inter + slope * fxs
    return fxs, fys

fxs, fys = FitLine(heights, inter, slope)

thinkplot.Scatter(heights, weights, alpha=0.01, s=10)
thinkplot.Plot(fxs, fys, color='white', linewidth=3)
thinkplot.Plot(fxs, 10**fys, color='yellow')
thinkplot.Config(xlabel='Height',
                 ylabel='log10 weight',
                 legend=False)
```



Plot percentiles of the residuals.

In [79]:
```
# As seeen below I definded the residual and plotted the heights and
# log-weight.
def Residuals(xs, ys, inter, slope):
    xs = np.asarray(xs)
    ys = np.asarray(ys)
    res = ys - (inter + slope * xs)
    return res

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

bins = np.arange(120, 200, 5)
indices = np.digitize(df.htm3, bins)
groups = df.groupby(indices)

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

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

PlotPercentiles(means, cdfs)

thinkplot.Config(xlabel="height (cm)",
                 ylabel='residual weight (kg)',
                 legend=False)
# As seen below we see that the lines take a fast dip and then level off
# which shows a slight linear relationship as the hight goes up and weig
# goes down
```



Compute correlation.

In [80]:
```
# We will calculate the rho as seen in the examples above by using the
# thinkstats2.Corr and using the heights and log weight.

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

Out[80]: 0.5317282605983437

Compute coefficient of determination.

In [81]:
```
# Solution goes here
r2 = rho**2
r2
```

Out[81]:
```
0.28273494311894004
```

Confirm that $R^2 = \rho^2$.

In [82]:
```
# We will confirm that r2 is equal to p2 by setting r2 minus the rho**2
p2_confirm = r2-rho**2
p2_confirm
```

Out[82]:
```
0.0
```

Compute Std(ys), which is the RMSE of predictions that don't use height.

In [83]:
```
# When computing the Std(ys) we can use log_weights and the
# thinkstat2 std to receive our calculation

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

Out[83]:
```
0.103207250300049
```

Compute Std(res), the RMSE of predictions that do use height.

In [84]:
```
# Next the Std(res) can be easily calculated by useing the
#thinkstats2.std and setting it to the res

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

Out[84]:
```
1.404875428785783
```

How much does height information reduce RMSE?

In [85]:
```
# Next we can determine how much height information reduces RMSE by
# taking 1 and minusing it by the stdres and the dividing it by stdys.

RMSEreduce = 1 - stdres / stdys
RMSEreduce
```

Out[85]:
```
-12.612177678423393
```

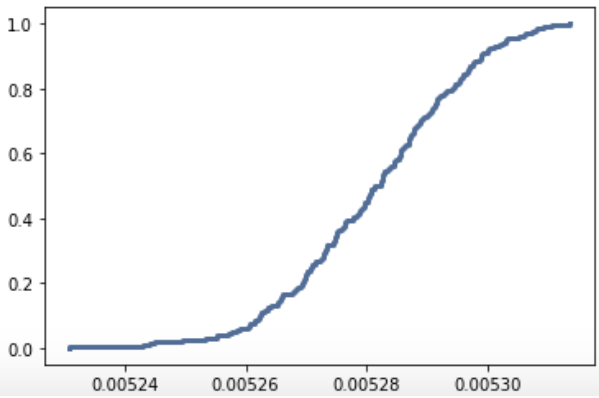
Use resampling to compute sampling distributions for inter and slope.

In [90]:
```
# First we will create a for loop that will resample and compute the
# distributions for the inters and the slope of the distribution
dist = []
for _ in range(200):
    re_samp = thinkstats2.ResampleRows(df)
    least_est = thinkstats2.LeastSquares(re_samp.htm3,
                                         np.log10(re_samp.wtkg2))
    dist.append(least_est)

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

Plot the sampling distribution of slope.

In [91]:
```
# Next we will plot the sampling distribution of slope by taking the
# thinkstats2.CDF and setting it to the slopes
sampling_cdf = thinkstats2.Cdf(slopes)
thinkplot.Cdf(sampling_cdf)
```

Out[91]:
```
{'xscale': 'linear', 'yscale': 'linear'}
```



Compute the p-value of the slope.

In [92]:
```
# The p-value of the slope can be calculate as done at In[36] by
# setting the sampling_cdf to zero as done above and now below

solution_pvalue = sampling_cdf[0]
solution_pvalue
```

Out[92]:
```
0
```

Compute the 90% confidence interval of slope.

In [94]:
```
# Next I will plot the confidence interval of the slope by setting
#the sampling_cdf.percentile to 90
coninter = cdf.Percentile(90)
coninter
```

Out[94]:
```
0.005303222468758684
```

Compute the mean of the sampling distribution.

In [95]:
```
# The mean of the samppling distribution can be calculated by setting th
# thinkstats2.Mean to the slopes which will give us the mean of the
# sampling distribution


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

Out[95]:
```
0.005281035326024636
```

Compute the standard deviation of the sampling distribution, which is the standard error.

In [96]:
```
# Next we can compute the standard deviation of the sampling distributio
# be calculated by setting the thinkstats2.Std to the slopes which will
# give us the standard deviation or also known as the standard error.

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

Out[96]:
```
1.4403043590925084e-05
```

Resample rows without weights, compute mean height, and summarize results.

In [98]:
```
# We can resample rows without the weights and copute the mean height by
# setting without weights equal to the thinkstats2.resamplerows to the
# data frame and htm3 to the mean with a for loop and them summarize.
without_weights = [thinkstats2.ResampleRows(df).htm3.mean()
                   for _ in range(200)]
Summarize(without_weights)
```

```
mean, SE, CI 168.956228842034 0.015064700861272883 (168.92866418076355,
168.97934729885407)
```

Resample rows with weights. Note that the weight column in this dataset is called `finalwt`.

In [99]:
```
# Next we will resample with weights by using the same format but with
# weighted and the finalwt
with_weights = [ResampleRowsWeighted(df, 'finalwt').htm3.mean()
                for _ in range(100)]
Summarize(with_weights)
```

```
mean, SE, CI 170.4963182865458 0.017301482416706118 (170.46516704056268
, 170.5239722912751)
```