Assignment 8.2

Exercises 9 - 1 and 10 - 1

http://thinkstats2.com

Copyright 2016 Allen B. Downey

MIT License: https://opensource.org/licenses/MIT

In [1]:
```
# Imports
import numpy as np
import matplotlib.pyplot as plt
import thinkstats2
import first
```

In [2]:
```
from IPython.core.display import HTML
table_css = 'table {align:left;display:block} '
HTML('<style>{}</style>'.format(table_css))
```

Out[2]:

Exercise 9 - 1

As sample size increase, the power of a hypotheses test increase, which means it is more likely
positive if the effect is real. Conversely, as sample size decreases, the test is less likely to be pos
even if the effect is real.

To investigage this behavior, run the tests in this chapter with different subsets of the NSFG data
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 the sample size decreases? What is the smallest
size that yields a positive test?

In [3]:
```
# Read NSFG data
live_df, firsts_df, others_df = first.MakeFrames()
```

```
In [4]: def PregLengths(sample, iters):
            '''
            Pregnancy lengths: Mean Difference, P-Value

            args:
                sample (df)
                iters (int)

            returns:
                none
            '''
            # Get first borns and others from the sample
            firsts = sample[sample.birthord == 1]
            others = sample[sample.birthord != 1]

            # Get pregnancy lengths from each df and drop any empty rows
            first_prglngths = firsts.prglngth.dropna().values
            others_prglngths = others.prglngth.dropna().values

            # get the mean difference
            first_mean = first_prglngths.mean()
            others_mean = others_prglngths.mean()
            mean_diff = first_mean - others_mean

            print('\nMean Difference in Preg Length: {} weeks'.format(mean_diff))

            # see probablilty of this randomly happening
            count = 0

            # loop from 1 to number of trials
            for _ in range(iters):
                # get length of pregnancy lengths
                first_len = len(first_prglngths)
                others_len = len(others_prglngths)

                # get data from sample pregnancy lengths
                all_prglngths = sample.prglngth.values
                # shuffle the contents of all pregnancy lengths
                np.random.shuffle(all_prglngths)

                # get pregnancy length from beginning to length of first borns
                new_first = all_prglngths[:first_len]
                # get pregnancy length from length of first borns to end
                new_others = all_prglngths[first_len:]
                # calculate new mean difference
                new_mean_diff = new_first.mean() - new_others.mean()

                # if new mean difference / mean difference >= 1 add 1 to count
                if new_mean_diff / mean_diff >= 1:
                    count += 1

            # calculate p-value
            pValue = count / iters

            print(f'P Value: {pValue}')


        def BirthWeights(sample, iters):
            '''
            Birth Weights: Mean Difference, P-Value
```

```
In [5]: # Statistical tests - 100 rows
        rows = 100
        trials = 1000

        sample = thinkstats2.SampleRows(live_df, rows)

        print('Sample Size: 100')

        PregLengths(sample,trials)
        BirthWeights(sample,trials)
        Correlate(sample, trials)
```

```
Sample Size: 100

Mean Difference in Preg Length: 0.20488195278110766 weeks
P Value: 0.308

Mean Difference in Birth Weights: 0.30334793491864787 lbs
P Value: 0.0

Correlation Coefficient between Age and Weight: 0.0848341795247848
P Value: 0.196
```

```
In [6]: # Statistical tests - 50 rows
        rows = 50
        trials = 1000

        sample = thinkstats2.SampleRows(live_df, rows)

        print('Sample Size: 50')

        PregLengths(sample,trials)
        BirthWeights(sample,trials)
        Correlate(sample, trials)
```

```
Sample Size: 50

Mean Difference in Preg Length: 0.4614121510673286 weeks
P Value: 0.289

Mean Difference in Birth Weights: 0.5591133004926112 lbs
P Value: 0.069

Correlation Coefficient between Age and Weight: -0.004576981667129779
P Value: 0.474
```

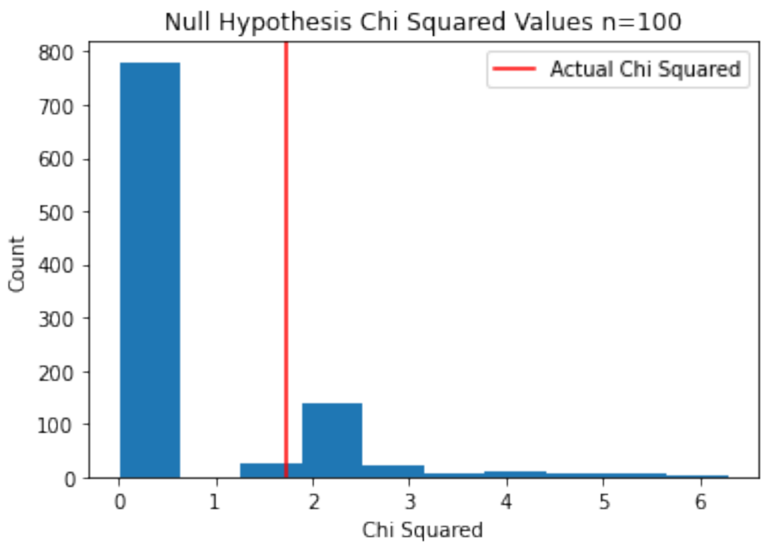
```
In [7]: # Statistical tests - 10 rows
        rows = 10
        trials = 1000

        sample = thinkstats2.SampleRows(live_df, rows)

        print('Sample Size: 10')

        PregLengths(sample,trials)
        BirthWeights(sample,trials)
        Correlate(sample, trials)
```

```
Sample Size: 10

Mean Difference in Preg Length: 1.3333333333333357 weeks
P Value: 0.306

Mean Difference in Birth Weights: -0.30208333333333304 lbs
P Value: 0.0

Correlation Coefficient between Age and Weight: -0.2719457369136887
P Value: 0.202
```

test	Sample Size	p-value
Mean Diff in Preg Length	100	0.008
Mean Diff in Preg Length	50	0.006
Mean Diff in Preg Length	10	0.706
Mean Diff in Birth Weight	100	0.442
Mean Diff in Birth Weight	50	0.0
Mean Diff in Birth Weight	10	0.0
Corr Coef: Age and Weight	100	0.211
Corr Coef: Age and Weight	50	0.1
Corr Coef: Age and Weight	10	0.355

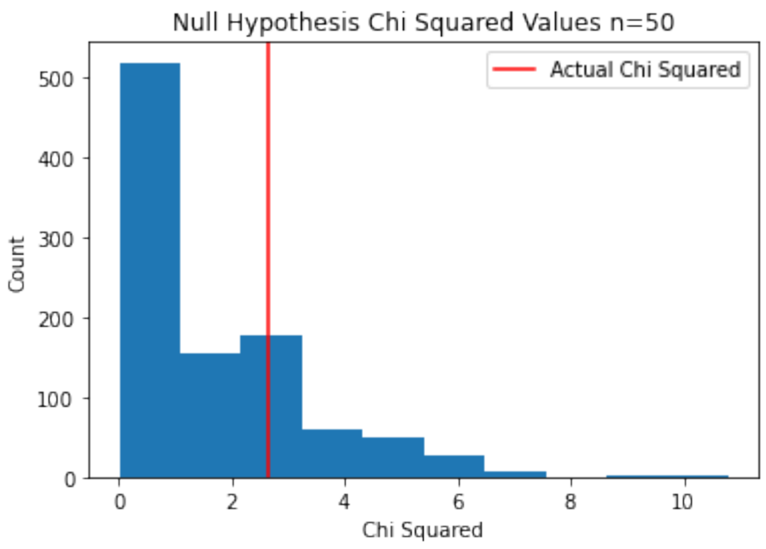
The p-values dont really move in a pattern except birth weight which gets down towards 0.

```
In [8]: # chi squared test
        n = 100
        sample = thinkstats2.SampleRows(live_df, n)

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

        chi_sq = ChiSquared(live_df, firsts)

        # generate null hypothesis data
        iters = 1000
        chis = []

        for _ in range(iters):
            samp = thinkstats2.SampleRows(live_df, n)
            chis.append(ChiSquared(live_df, samp))

        p_val = len(np.array(chis)[np.where(chis >= chi_sq)]) / len(chis)

        plt.hist(chis)
        plt.axvline(chi_sq, label='Actual Chi Squared', color='r')
        plt.title('Null Hypothesis Chi Squared Values n={}'.format(n))
        plt.ylabel('Count')
        plt.xlabel('Chi Squared')
        plt.legend()
        plt.show()

        print(f'Max Value Under Null Hypothesis: {max(chis)}')
        print(f'Actual Chi Squared: {chi_sq}')
        print(f'p-value: {p_val}')
```



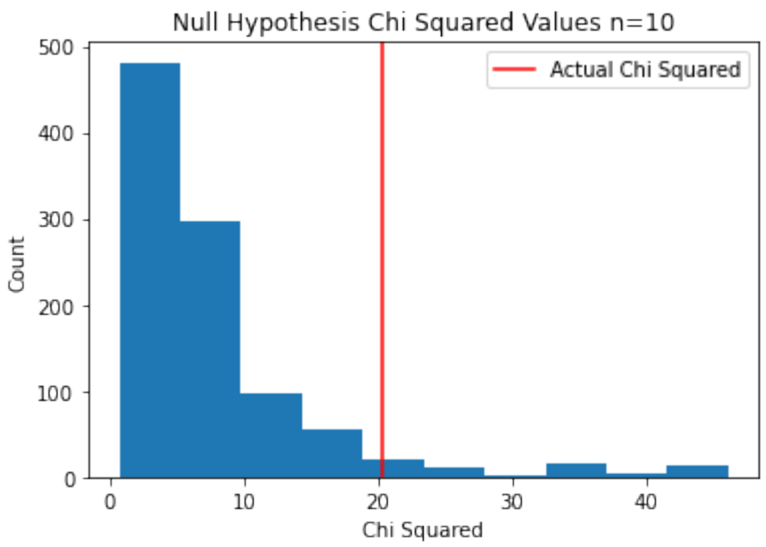
```
Max Value Under Null Hypothesis: 6.293512417161693
Actual Chi Squared: 1.7261786610411627
p-value: 0.214
```

```
In [9]: n = 50
        sample = thinkstats2.SampleRows(live_df, n)

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

        chi_sq = ChiSquared(live_df, firsts)

        # generate null hypothesis data
        iters = 1000
        chis = []

        for _ in range(iters):
            samp = thinkstats2.SampleRows(live_df, n)
            chis.append(ChiSquared(live_df, samp))

        p_val = len(np.array(chis)[np.where(chis >= chi_sq)]) / len(chis)

        plt.hist(chis)
        plt.axvline(chi_sq, label='Actual Chi Squared', color='r')
        plt.title('Null Hypothesis Chi Squared Values n={}'.format(n))
        plt.ylabel('Count')
        plt.xlabel('Chi Squared')
        plt.legend()
        plt.show()

        print(f'Max Value Under Null Hypothesis: {max(chis)}')
        print(f'Actual Chi Squared: {chi_sq}')
        print(f'p-value: {p_val}')
```



```
Max Value Under Null Hypothesis: 10.793459600498872
Actual Chi Squared: 2.6623909883909604
p-value: 0.203
```

```
In [10]: n = 10
         sample = thinkstats2.SampleRows(live_df, n)

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

         chi_sq = ChiSquared(live_df, firsts)

         # generate null hypothesis data
         iters = 1000
         chis = []

         for _ in range(iters):
             samp = thinkstats2.SampleRows(live_df, n)
             chis.append(ChiSquared(live_df, samp))

         p_val = len(np.array(chis)[np.where(chis >= chi_sq)]) / len(chis)

         plt.hist(chis)
         plt.axvline(chi_sq, label='Actual Chi Squared', color='r')
         plt.title('Null Hypothesis Chi Squared Values n={}'.format(n))
         plt.ylabel('Count')
         plt.xlabel('Chi Squared')
         plt.legend()
         plt.show()

         print(f'Max Value Under Null Hypothesis: {max(chis)}')
         print(f'Actual Chi Squared: {chi_sq}')
         print(f'p-value: {p_val}')
```



```
Max Value Under Null Hypothesis: 46.15692057008778
Actual Chi Squared: 20.35544668462675
p-value: 0.067
```

The p-values for chi-squared are also not following a pattern so it is hard to predict the lowest sa
size that yields a positive test.

Exercise 10-1

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

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, without weights, to estimate the mean height of respondents in the BRFSS, the standard error of mean, and a 90% confidence interval. How much does correct weighting affet the estimates?

```
In [11]: import brfss
         import scipy
```

```
In [12]: # Read BRFSS data
         brfss_df = brfss.ReadBrfss(nrows=None)
         brfss_df.head()
```

Out[12]:

	age	sex	wtyrago	finalwt	wtkg2	htm3
0	82.0	2	76.363636	185.870345	70.91	157.0
1	65.0	2	72.727273	126.603027	72.73	163.0
2	48.0	2	NaN	181.063210	NaN	165.0
3	61.0	1	73.636364	517.926275	73.64	170.0
4	26.0	1	88.636364	1252.624630	88.64	185.0

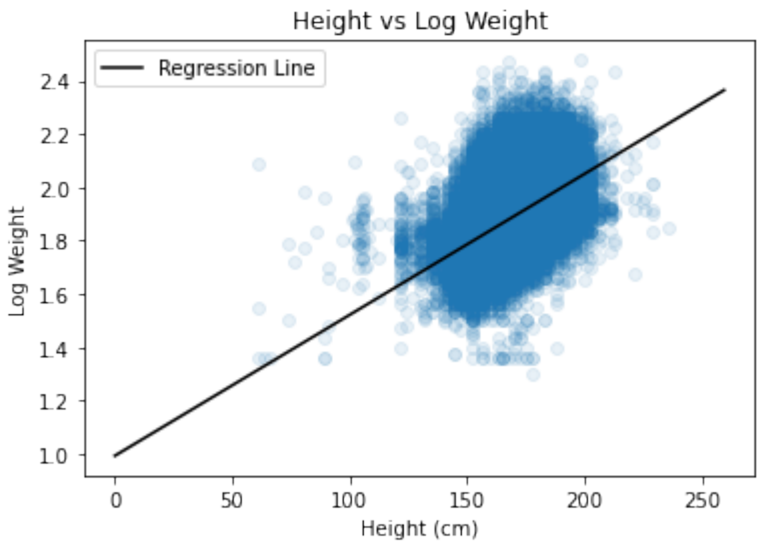
```
In [13]: # drop the empty rows from the htm3 and wtkg2 columns
         brfss_df = brfss_df.dropna(subset=['htm3', 'wtkg2'])

         # move the columns into their own df
         heights, weights = brfss_df.htm3, brfss_df.wtkg2
```

```
In [14]: # convert weights to log 10
         log_weights = np.log10(weights)
```

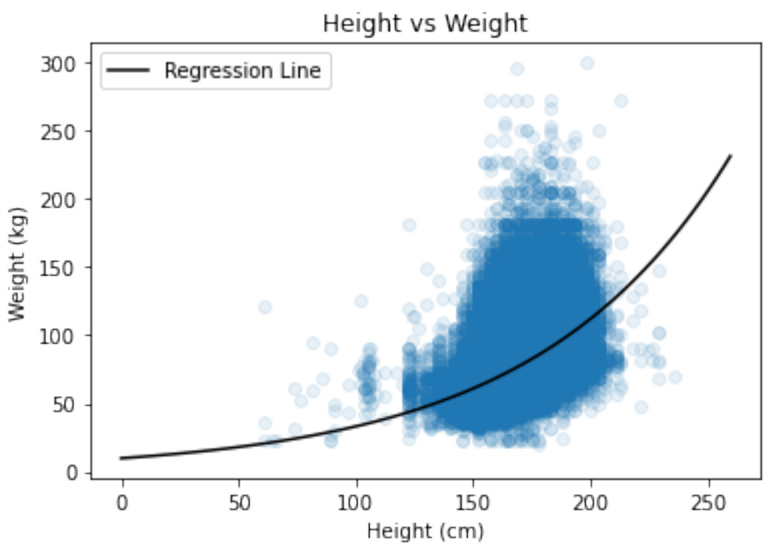
```
In [15]: # calculate the linear regression for heights and weights
         regress = scipy.stats.linregress(heights, log_weights)
         slope, intercept, r_value, p_value, std_err = regress

         print(f"Slope: {slope}")
         print(f"Intercept: {intercept}")
         print(f"R Value - Pearson: {r_value}")
         print(f"P Value: {p_value}")
         print(f"Std Error: {std_err}")
```

```
Slope: 0.005281454169417775
Intercept: 0.9930804163932879
R Value - Pearson: 0.5317282605983431
P Value: 0.0
Std Error: 1.3370549498879916e-05
```

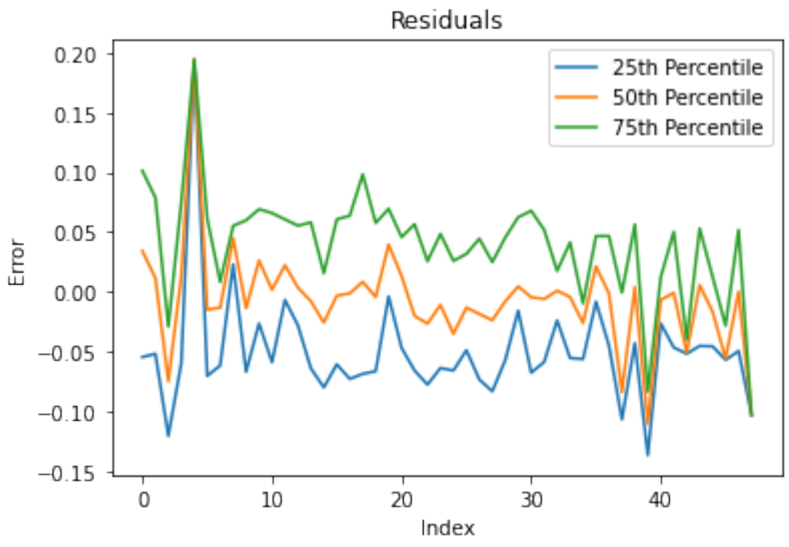
```
In [16]: # Plot regression with log

         plt.scatter(heights, log_weights, alpha = 0.1)

         x_fit = np.linspace(0, max(heights)*1.1, 100)
         y_fit = slope * x_fit + intercept

         plt.plot(x_fit, y_fit, color = 'k', label = 'Regression Line')
         plt.title('Height vs Log Weight')
         plt.xlabel('Height (cm)')
         plt.ylabel('Log Weight')

         plt.legend()
         plt.show()
```



```
In [17]: # Plot regression without log

         plt.scatter(heights, weights, alpha = 0.1)

         x_fit = np.linspace(0, max(heights)*1.1, 100)
         y_fit = slope * x_fit + intercept

         plt.plot(x_fit, 10**y_fit, color = 'k', label = 'Regression Line')
         plt.title('Height vs Weight')
         plt.xlabel('Height (cm)')
         plt.ylabel('Weight (kg)')

         plt.legend()
         plt.show()
```



```
In [18]: # Calculate the residuals
         residuals = []

         for h, w in zip(heights, log_weights):
             r = w - (h * slope + intercept)
             residuals.append(r)

In [19]: # height range
         height_range = range(140, 190)

In [20]: # calculate percentiles
         r_25 = []
         r_50 = []
         r_75 = []

         residuals = np.array(residuals)

         for h in height_range:
             r = residuals[np.where(heights == h)]

             if r.size != 0:
                 r_25.append(np.percentile(residuals[np.where(heights == h)], 25))
                 r_50.append(np.percentile(residuals[np.where(heights == h)], 50))
                 r_75.append(np.percentile(residuals[np.where(heights == h)], 75))

In [21]: # Plot percentiles
         plt.plot(r_25, label='25th Percentile')
         plt.plot(r_50, label='50th Percentile')
         plt.plot(r_75, label='75th Percentile')

         plt.title('Residuals')
         plt.xlabel('Index')
         plt.ylabel('Error')

         plt.legend()
         plt.show()
```



```
In [22]: # calculate coefficient of determination

         r2 = thinkstats2.CoefDetermination(log_weights, residuals)
         print(f"r2: {r2}")

        r2: 0.282734943118939

In [23]: # Confirm that R^2 = p^2
         np.isclose(r_value**2, r2)

Out[23]:  True

In [24]: # unweighted resampling

         # resample rows
         estimates_unweighted = [thinkstats2.ResampleRows(brfss_df).htm3.mean()
                                 for _ in range(100)]

         mean = np.mean(estimates_unweighted)
         stderr = np.std(estimates_unweighted)
         cdf = thinkstats2.Cdf(estimates_unweighted)
         ci = cdf.ConfidenceInterval(90)

         print(f'Mean: {mean}')
         print(f'Std Error: {stderr}')
         print(f'CI: {ci}')

        Mean: 168.9567373279573
        Std Error: 0.015794354242453464
        CI: (168.93093787263282, 168.9810879362962)
```

```
In [25]: # weighted resampling
         def ResampleRowsWeighted(df):
             weights = df['finalwt']
             cdf = thinkstats2.Cdf(dict(weights))
             indices = cdf.Sample(len(weights))
             sample = df.loc[indices]
             return sample

         # resample rows
         estimates_weighted = [ResampleRowsWeighted(brfss_df).htm3.mean()
                               for _ in range(100)]

         mean = np.mean(estimates_weighted)
         stderr = np.std(estimates_weighted)
         cdf = thinkstats2.Cdf(estimates_weighted)
         ci = cdf.ConfidenceInterval(90)

         print(f'Mean: {mean}')
         print(f'Std Error: {stderr}')
         print(f'CI: {ci}')
```

```
Mean: 170.4965447209928
Std Error: 0.016775810759808673
CI: (170.46839315669274, 170.52121101881608)
```

With weighting the mean difference is only about 1.5 kg with a standard error difference of only a 0.004 which implies that the difference is not due to chance.