Bilenkin530Week7_Exercises_7.2

January 26, 2025

0.0.1 Chapter 7 Exercise 7.1 (Page 89)

```
[156]: import warnings
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)

warnings.filterwarnings('ignore', category=FutureWarning)

download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
 ↪thinkstats2.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/thinkplot.
 ↪py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/nsfg.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
 ↪2002FemResp.dct")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
 ↪2002FemResp.dat.gz")
```

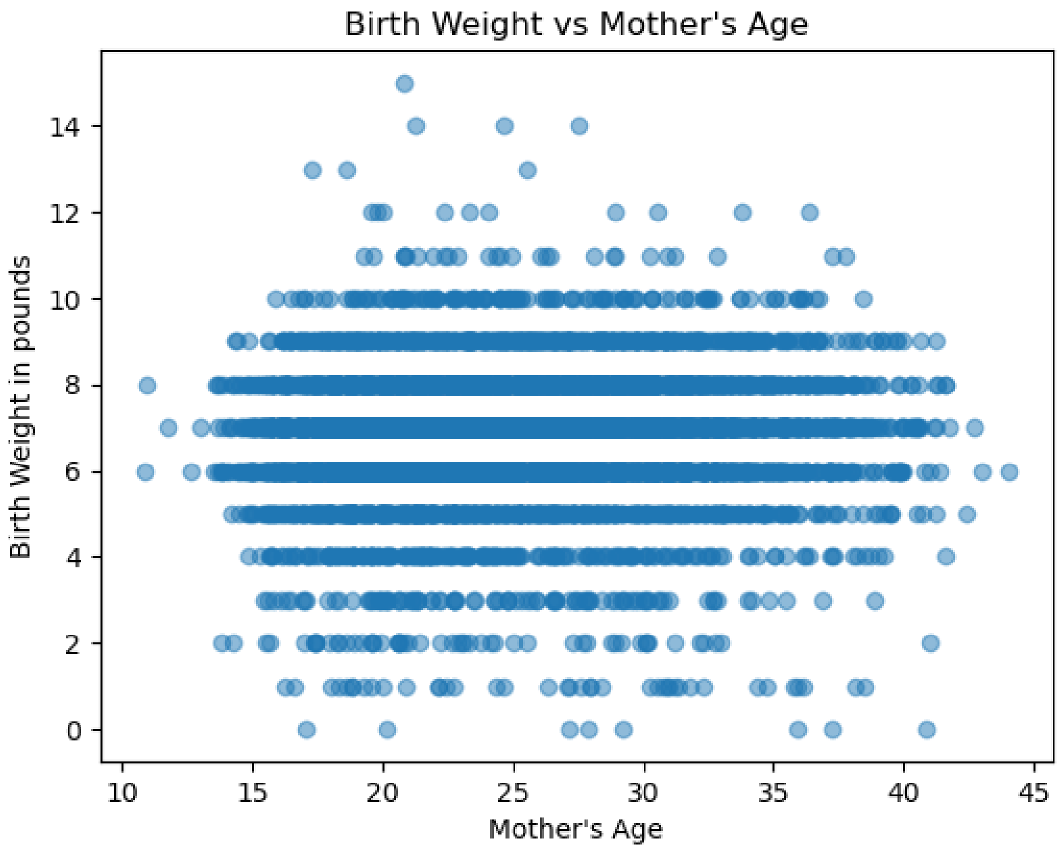
0.0.2 Loading, cleaning and preparing the data for further processing

```
[157]: import thinkstats2
import thinkplot
import nsfg
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings

# Ignoring warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

```
# Loading the NSFG data file
pregnancy = nsfg.ReadFemPreg()

# Cleaning data
pregnancy['birthwgt_lb'] = pregnancy['birthwgt_lb'].replace([97, 98, 99], np.
  ↪nan)
pregnancy['birthwgt_oz'] = pregnancy['birthwgt_oz'].replace([97, 98, 99], np.
  ↪nan)
pregnancy['agepreg'] = pregnancy['agepreg'].replace([97, 98, 99], np.nan)

# Extracting relevant columns and dropping NaNs at the same time
data_clean = pregnancy[['agepreg', 'birthwgt_lb']].dropna()

mother_age = data_clean['agepreg']
birth_weight = data_clean['birthwgt_lb']
```

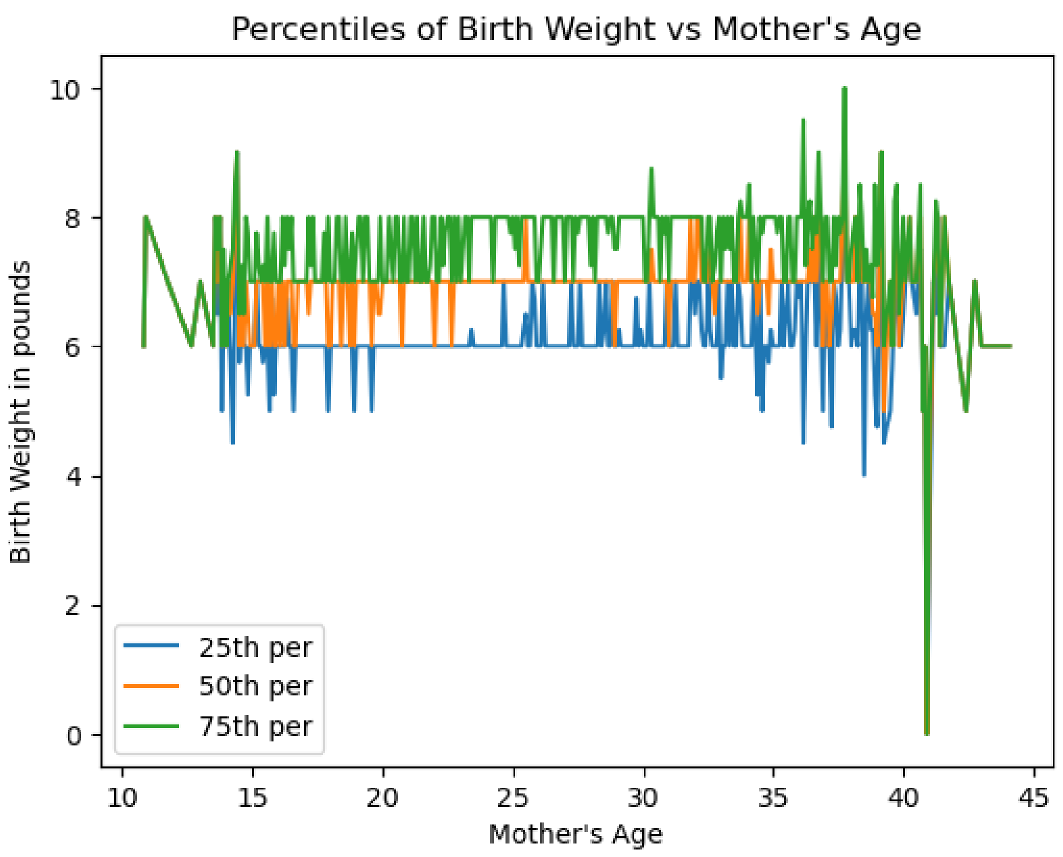
0.0.3 Using data from the NSFG, make a scatter plot of birth weight versus mother's age.

[158]:
```
# Creating Scatter plot of birth weitght and mother's age
plt.scatter(mother_age, birth_weight, alpha=0.5)
plt.xlabel('Mother\'s Age')
plt.ylabel('Birth Weight in pounds')
plt.title('Birth Weight vs Mother\'s Age')
plt.show()
```



0.0.4 Plot percentiles of birth weight versus mother's age.

[159]:
```
# Grouping data by mother's age and calculating the percentiles
age_groups = data_clean.groupby('agepreg')['birthwgt_lb']
p25 = age_groups.quantile(0.25)
p50 = age_groups.quantile(0.50)
p75 = age_groups.quantile(0.75)

# Creating percentiles plot
plt.plot(p25.index, p25, label='25th per')
plt.plot(p50.index, p50, label='50th per')
plt.plot(p75.index, p75, label='75th per')
plt.xlabel('Mother\'s Age')
plt.ylabel('Birth Weight in pounds')
plt.legend()
plt.title('Percentiles of Birth Weight vs Mother\'s Age')
plt.show()
```



0.0.5 Compute Pearson's and Spearman's correlations.

[160]:
```
from scipy.stats import pearsonr, spearmanr

# Calculating Pearson's correlation
pearson_corr, _ = pearsonr(mother_age, birth_weight)
print('Pearson correlation: {:.4f}'.format(pearson_corr))

# Calculating Spearman's correlation
spearman_corr, _ = spearmanr(mother_age, birth_weight)
print('Spearman correlation: {:.4f}'.format(spearman_corr))
```

```
Pearson correlation: 0.0690
Spearman correlation: 0.0946
```

0.0.6 How would you characterize the relationship between these variables?

Looking at the scatter plot it seems like there is a very little trend. However, the data points are very dispersed. By looking at the percentiles plot we can see a better trend. We can clearly see how the birth weight changes with mother's age. Overall, we can see that the birth weight remains consistent across different mothers' age groups with little change in some age ranges. With the calculated Pearson's correlation value of 0.0690, we can conclude that there is a very low correlation between mother's age and birth weight. Spearman's correlation of 0.0946 indicates that there is a little more correlation between the two variables. However, overall, the correlation is still weak. Finally, we can conclude that the relationship between the two variables is weak. Most likely there are other more important factors that can have more correlation and impact on prediction.

0.0.7 Chapter 8 Exercise 8.1 (Page 99)

In this chapter we used x̄ and median to estimate , and found that x̄ yields lower MSE. Also, we used S^2 and S^2n-1 to estimate , and found that S2 is biased and S^2n-1 unbiased. Run similar experiments to see if x̄ and median are biased estimates of . Also check whether S2 or S^2n-1 yields a lower MSE.

0.0.8 Calculating mean and median biases

[161]:
```python
# Function to calculate the mean and median biases
def experiment_mean_median(n, num_simulations):
    means = []
    medians = []

    for _ in range(num_simulations):
        data = np.random.normal(0, 1, n)
        means.append(np.mean(data))
        medians.append(np.median(data))

    population_mean = 0

    mean_bias = np.mean(means) - population_mean
    median_bias = np.mean(medians) - population_mean

    return mean_bias, median_bias

# Parameters for the experiment
n = 100  # Sample size
num_simulations = 1000

# Displaying calculated results
mean_bias, median_bias = experiment_mean_median(n, num_simulations)
print(f"Mean bias: {mean_bias:.4f}")
print(f"Median bias: {median_bias:.4f}")
```

```
Mean bias: 0.0043
Median bias: 0.0037
```

0.0.9 Calculating biased and unbiased variance estimators

[162]:
```python
# Function to calculate the MSE for biased and unbiased variance estimators
def experiment_variance(n, num_simulations):
    biased_var = []
    unbiased_var = []

    for _ in range(num_simulations):
        data = np.random.normal(0, 1, n)
        biased_var.append(np.var(data, ddof=0))
        unbiased_var.append(np.var(data, ddof=1))

    population_variance = 1

    mse_biased = np.mean((np.array(biased_var) - population_variance)**2)
    mse_unbiased = np.mean((np.array(unbiased_var) - population_variance)**2)

    return mse_biased, mse_unbiased

# Displaying calculated results
mse_biased, mse_unbiased = experiment_variance(n, num_simulations)
print(f"MSE of biased variance: {mse_biased:.4f}")
print(f"MSE of unbiased variance: {mse_unbiased:.4f}")
```

```
MSE of biased variance: 0.0193
MSE of unbiased variance: 0.0197
```

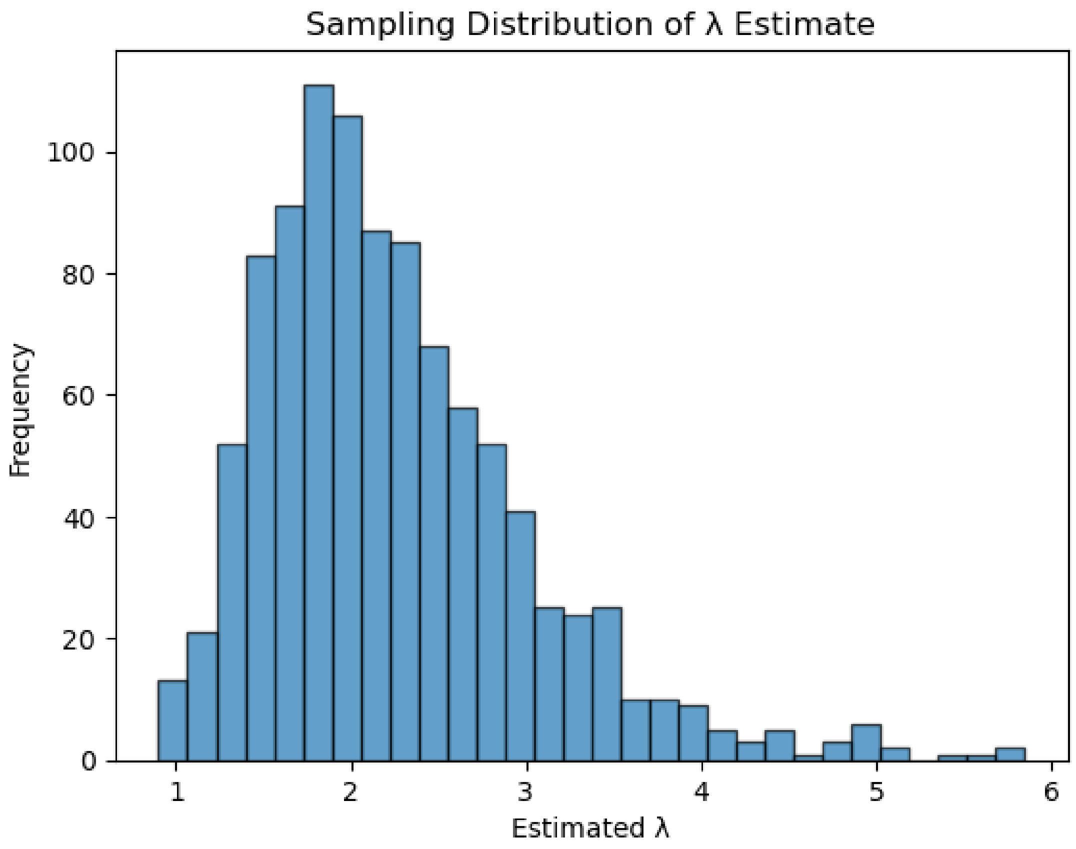
In both experiments we can see that the results for biased and ubised means are close to zero which means both mean and median are unbised estimators of the mean. The same goes for MSE of biased and unbised variances. Both calculated values are very close to zero which means they won't impact the predictability of the mean and median of the population mean.

0.0.10 Chapter 8 Exercise 8.2 (Page 99)

Suppose you draw a sample with size n=10 from an exponential distribution with =2. Simulate this experiment 1000 times and plot the sampling distribution of the estimate L. Compute the standard error of the estimate and the 90% confidence interval.

0.0.11 Simulation experiment for sample of size n = 10 for 1000 times

[163]:
```python
# Parameters of the exponential distribution
lambda_true = 2
n = 10
num_simulations = 1000

# Simulating the experiment
estimates = []
for _ in range(num_simulations):
    sample = np.random.exponential(1 / lambda_true, n)
    lambda_hat = 1 / np.mean(sample)
    estimates.append(lambda_hat)

# Plot the sampling distribution of the estimate
plt.hist(estimates, bins=30, alpha=0.7, edgecolor='black')
plt.xlabel('Estimated  ')
plt.ylabel('Frequency')
plt.title('Sampling Distribution of   Estimate')
plt.show()

# Convert estimates to a numpy array for easier calculation
estimates = np.array(estimates)

# Calculate the standard error of the estimates
standard_error = np.std(estimates)
print(f'Standard Error: {standard_error:.4f}')

# Calculate the 90% confidence interval
confidence_interval = np.percentile(estimates, [5, 95])
print(f'90% Confidence Interval: {confidence_interval[0]:.4f} to␣
 ↪{confidence_interval[1]:.4f}')
```

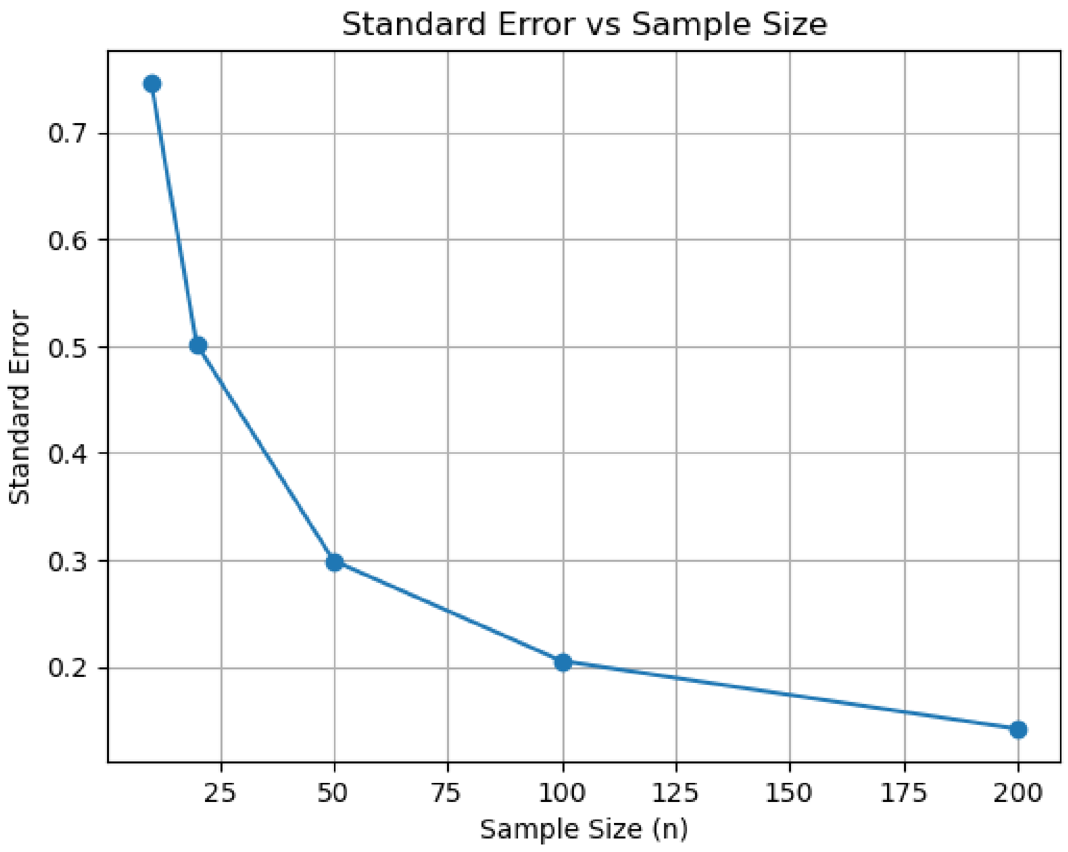


```
Standard Error: 0.7680
90% Confidence Interval: 1.3053 to 3.6566
```

Repeat the experiment with a few different values of n and make a plot of standard error versus n.

0.0.12 Repeating the experiment with different sample sizes of n and plot standard error vs n

[164]:
```
num_simulations = 1000
sample_sizes = [10, 20, 50, 100, 200]
standard_errors = []

for n in sample_sizes:
    estimates = []
    for _ in range(num_simulations):
        sample = np.random.exponential(1 / lambda_true, n)
        lambda_hat = 1 / np.mean(sample)
        estimates.append(lambda_hat)

    standard_error = np.std(estimates)
    standard_errors.append(standard_error)

# Plotting standard error vs sample size
plt.plot(sample_sizes, standard_errors, marker='o')
plt.xlabel('Sample Size (n)')
plt.ylabel('Standard Error')
plt.title('Standard Error vs Sample Size')
plt.grid(True)
plt.show()
```



We can see from the plot; the Standard error tends to decrease as Sample Size increases because larger sample size can capture more characteristics of population. That's the reason why larger sample sizes are preferable than smaller.