Homework #2 by Olivia Lewandowski Final

January 25, 2023

1 DS4E: Homework 2

[1]:
```
# import libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

1.1 Question 1

1(a)

[2]:
```
myarray = np.array([12, 9, 1, 3, 1, 2, 7, 6])
print(myarray)
```

```
[12  9  1  3  1  2  7  6]
```

[3]:
```
median = np.median(myarray)
print('Median: ' + str(median))
```

```
Median: 4.5
```

[4]:
```
mean = np.mean(myarray)
print('Mean: ' + str(mean))
```

```
Mean: 5.125
```

[5]:
```
sorted_array = np.sort(myarray)

#can also do without sorted array
quartile1 = np.median(sorted_array[:4])
quartile3 = np.median(sorted_array[4:])
interqr = quartile3 - quartile1
print('IQR: ' + str(interqr))
```

```
IQR: 6.5
```

[6]:
```
std = np.std(myarray)
print('Standard Deviation: ' + str(std))
```

```
Standard Deviation: 3.7893765978060294
```

1(b)

```
[7]: mis_array = np.array([21, 9, 1, 3, 1, 2, 7, 6])
     print(mis_array)
```

```
[21  9  1  3  1  2  7  6]
```

```
[8]: median2 = np.median(mis_array)
     print('Median: ' + str(median2))
```

```
Median: 4.5
```

```
[9]: mean2 = np.mean(mis_array)
     print('Mean: ' + str(mean2))
```

```
Mean: 6.25
```

```
[10]: std2 = np.std(mis_array)
      print('Standard Deviation: ' + str(std2))
```

```
Standard Deviation: 6.219927652312364
```

1(c)

I noticed that in this case, the relative robustness of the median is greater than the mean. This is because the median is not responsive to outliers, while the mean can be heavily altered by extreme high or low values. The median is more resilient to extreme data, making it useful in certain situations, and therefore it is more robust.

1(d)

Standard deviation is a measure of the distance between values and the mean of a data set. A larger standard deviation indicates that the values are more spread out around the mean; the higher the standard deviation, the farther the values are from the mean on average. Therefore it makes sense that substituting a larger value (further from the mean) into the data set for a smaller one (closer to the mean) would cause the standard deviation to grow/be larger, because that new misrecorded value is farther from the mean, therefore increasing the average distance of the values from the mean (which is the def. of SD).

1.2 Question 2

2(a)

```
[11]: #general function, with replacement specified in the print statement, not␣
       ↪parameter
      def sample(myarray, replace):
        output = np.random.choice(myarray,
                                  size = len(myarray),
                                  replace = replace)
```

```
  return(output)

print(sample(myarray, True))
print(sample(myarray, False))
```

```
[1 6 3 2 9 1 3 7]
[ 1 12  9  3  6  7  2  1]
```

```
[12]: #function with replacement set to True as the parameter; both variations work
       ↪though, but this one is hard code/worse
      def sample(myarray, replace = True):
        output = np.random.choice(myarray,
                                  size = len(myarray),
                                  replace = replace)
        return(output)

      print(sample(myarray))
```

```
[3 1 1 1 2 1 7 2]
```

2(b)

```
[13]: for i in range(1,4):
            print(np.mean(sample(myarray, True)))
```

```
5.0
3.625
7.75
```

2(c)

```
[14]: for i in range(1,4):
            print(np.mean(sample(myarray, False)))
```

```
5.125
5.125
5.125
```

2(d)

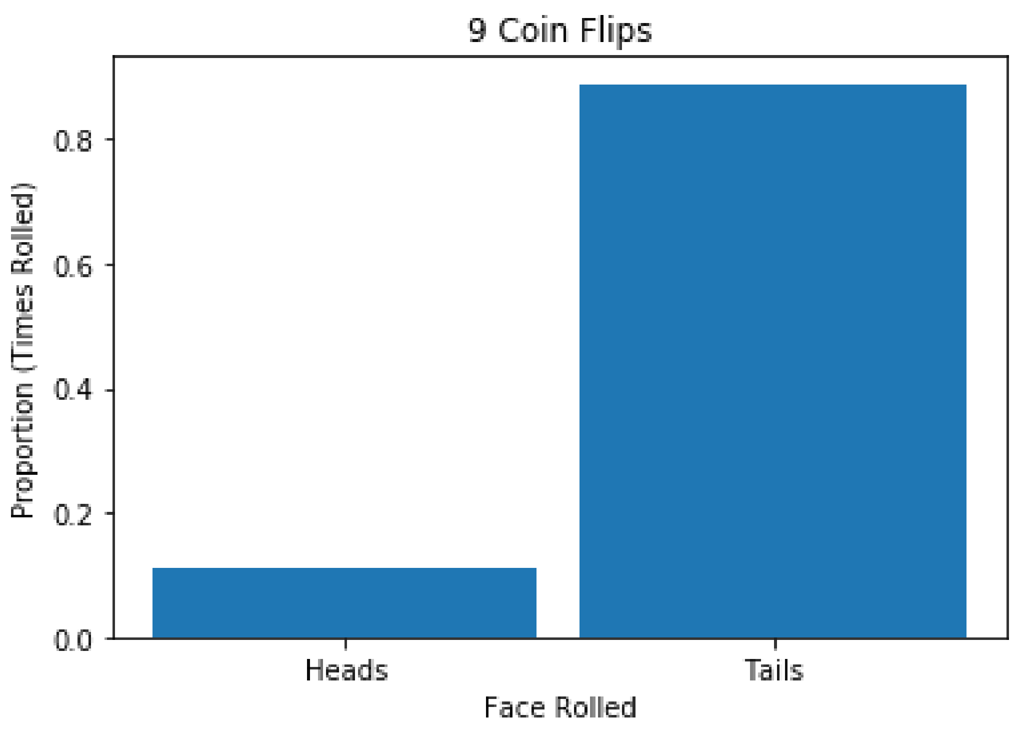
When sampling with replacement, we get different means, but when sampling without replacement, we get the same mean each time. This is because when sampling with replacement, we can get different combinations and amounts of the numbers in the array, allowing for a larger or smaller sum than the initial array, and therefore a different mean. However, when sampling without replacement, we are guaranteed to get the same set of numbers every time (we are sampling 8 #s from an 8 # array), just in a different order, therefore the mean is still the same because the numbers (and sum) are the same.

1.3 Question 3

3(a)

[63]:
```python
rolls = np.random.choice(range(1,3),
                           size=9,
                           replace=True)

print(rolls)
x_labels = ['Heads','Tails']

plt.hist(rolls,
         bins=np.arange(.5, 2.75),
         rwidth=.9,
         weights = np.ones(len(rolls)) / len(rolls)
        )
plt.title('9 Coin Flips')
plt.xlabel('Face Rolled')
plt.ylabel('Proportion (Times Rolled)')
plt.xticks(range(1,3), x_labels)

plt.show()
```

```
[2 2 2 2 2 2 2 1 2]
```



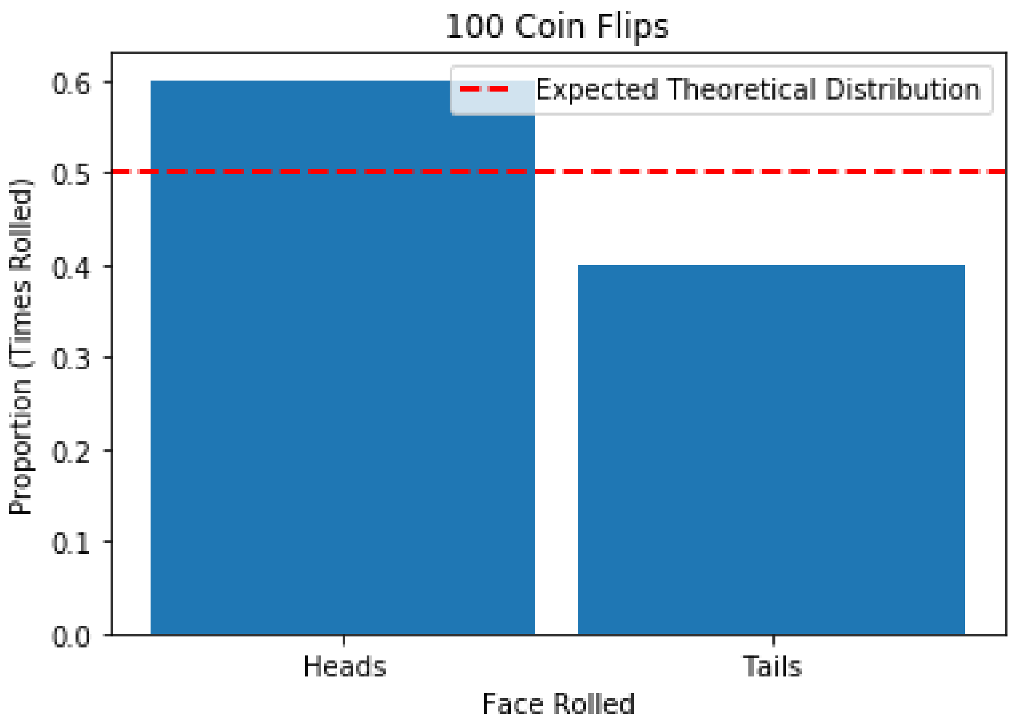
3(b)

Of my 9 flips, 1 came up heads, so a 1/9 or 0.11 proportion. Theoretically, we would expect 4.5 of the flips to be heads, and 4.5 to be tails, due to the fact that there is a 0.5 (equal) probability of landing on either side, and 50% of 9 is 4.5, however this isn't empirically possible because half-flips don't exist.

3(c)

[16]:
```
rolls = np.random.choice(range(1,3),
                           size=100,
                           replace=True)

print(rolls)
x_labels = ['Heads', 'Tails']

plt.hist(rolls,
         bins=np.arange(.5, 2.745),
         rwidth=.9,
         weights = np.ones(len(rolls)) / len(rolls)
        )
plt.title('100 Coin Flips')
plt.xlabel('Face Rolled')
plt.ylabel('Proportion (Times Rolled)')
plt.axhline(1/2, color='red', lw = 2, ls = '--', label='Expected Theoretical␣
 ↪Distribution')
plt.legend()
plt.xticks(range(1,3), x_labels)

plt.show()

heads_100 = ((rolls == 1).sum())
```

```
[1 1 2 2 2 1 1 2 1 2 1 1 1 2 2 2 1 2 1 1 2 1 2 1 1 2 1 1 1 2 1 1 1 2 1 1 1
 1 2 2 1 2 2 2 2 1 1 2 1 1 1 1 1 2 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 2 2 2
 1 1 1 1 2 1 1 1 2 2 1 1 2 2 2 1 2 1 1 1 2 2 1 2 2 1]
```

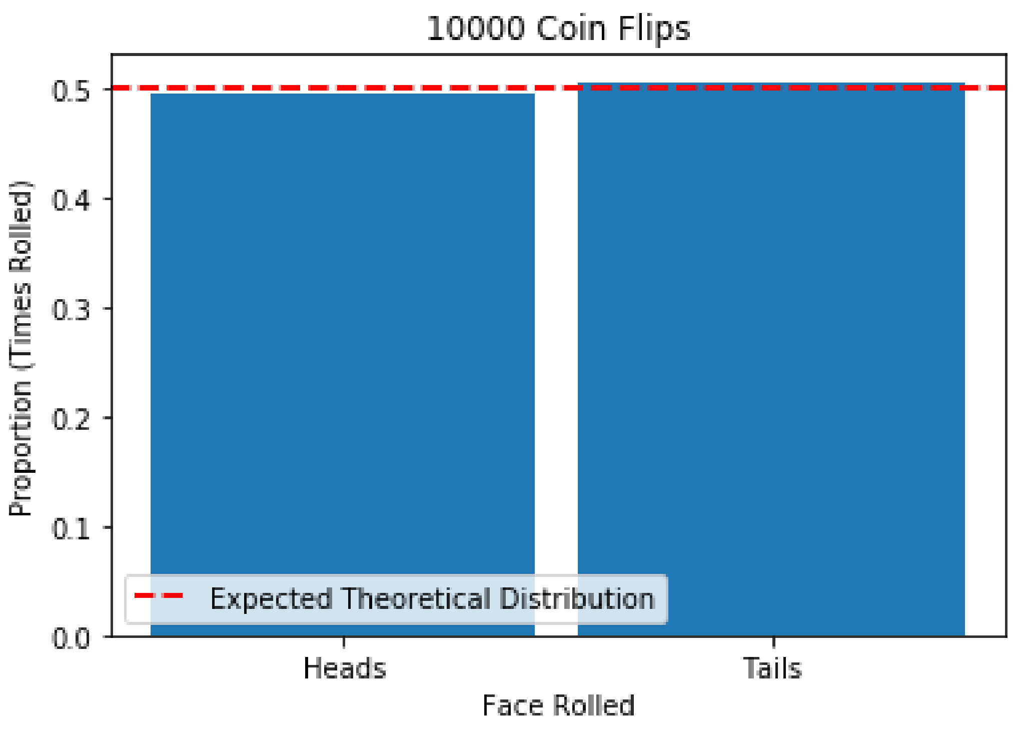


3(d)

[17]:
```
rolls = np.random.choice(range(1,3),
                           size=10000,
                           replace=True)

print(rolls)
x_labels = ['Heads', 'Tails']

plt.hist(rolls,
         bins=np.arange(.5, 2.745),
         rwidth=.9,
         weights = np.ones(len(rolls)) / len(rolls)
        )
plt.title('10000 Coin Flips')
plt.xlabel('Face Rolled')
plt.ylabel('Proportion (Times Rolled)')
plt.axhline(1/2, color='red', lw = 2, ls = '--', label='Expected Theoretical␣
 ↪Distribution')
plt.legend()
plt.xticks(range(1,3), x_labels)

plt.show()
```

```python
heads_10000 = ((rolls == 1).sum())

print('Proportion of heads in 100 rolls: ' + str(heads_100/100))
print('Proportion of heads in 10000 rolls: ' + str(heads_10000/10000))
```

```
[2 2 1 … 2 1 1]
```



```
Proportion of heads in 100 rolls: 0.6
Proportion of heads in 10000 rolls: 0.4949
```

The proportion of heads in 10000 flips is much closer to the theoretical probatility of heads flipped, 0.50, than the proportion of heads in 100 flips. This is due to the Law of Large Numbers, which states that if we increase the sample size, it will eventually converge on the expected distribution/theoretical value. With more flips, the empirical proportional value gets closer to the expected proportional value, therefore 10000 flips will produce a probability much closer to the theoretical than 100 flips will.

1.4 Question 4

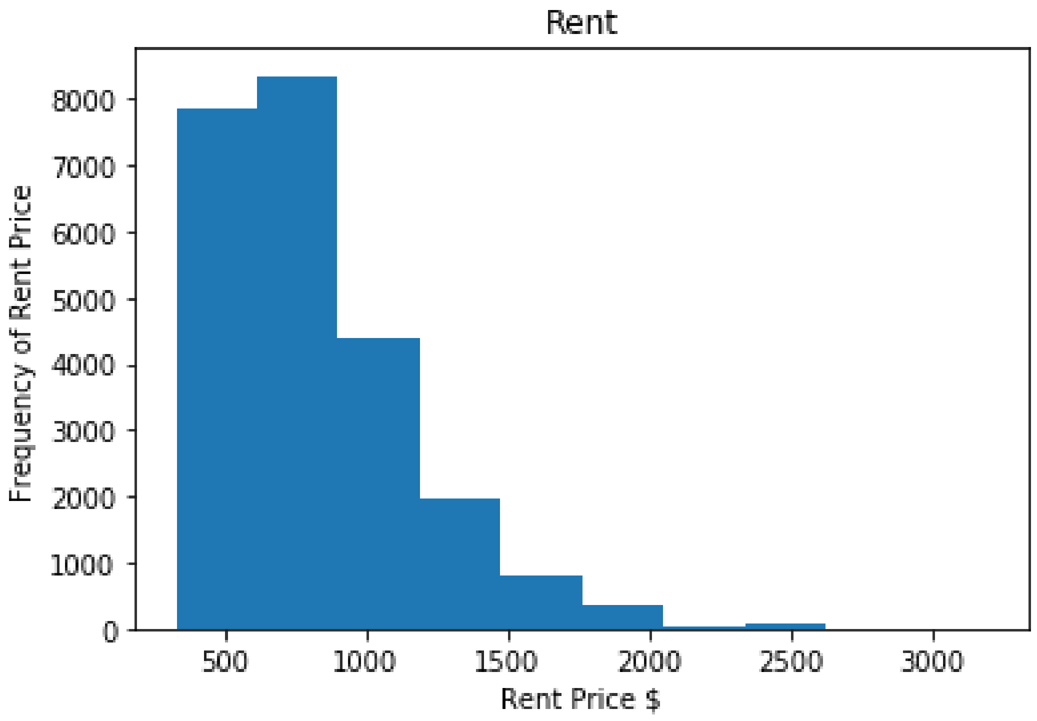
4(a)

```python
[21]: data = pd.read_csv('rents_county.csv')
      data.head()
```

```
[21]:    year       fips  state  county  bedrooms  rent
      0  2009  100199999      1       1         0   580
      1  2009  100399999      1       3         0   524
      2  2009  100599999      1       5         0   453
      3  2009  100799999      1       7         0   590
      4  2009  100999999      1       9         0   590
```

4(b)

```
[22]: plt.hist(data['rent'])
      plt.title('Rent')
      plt.xlabel('Rent Price $')
      plt.ylabel('Frequency of Rent Price')

      plt.show()
```



In this histogram, there is a right, or positive skew; this is where the values (rents) seem to pile up on the left (cheaper side), pulling the rest of the distribution of the sample out to the right. Furthermore, this indicates that the mean is at a higher value than the median, showing that there are fewer rents at very high prices, which are able to influence the less-robust mean.

4(c)

[23]:
```python
rent = data['rent']

length = len(rent)
print('Number of Observations: ' + str(length))

mean = np.mean(rent)
print('Mean: ' + str(round(mean, 1)))

maximum = max(rent)
print('Maximum: ' + str(maximum))

std = np.std(rent)
print('Standard Deviation: ' + str(round(std, 1)))
```

```
Number of Observations: 23815
Mean: 819.2
Maximum: 3198
Standard Deviation: 343.3
```

4(d)

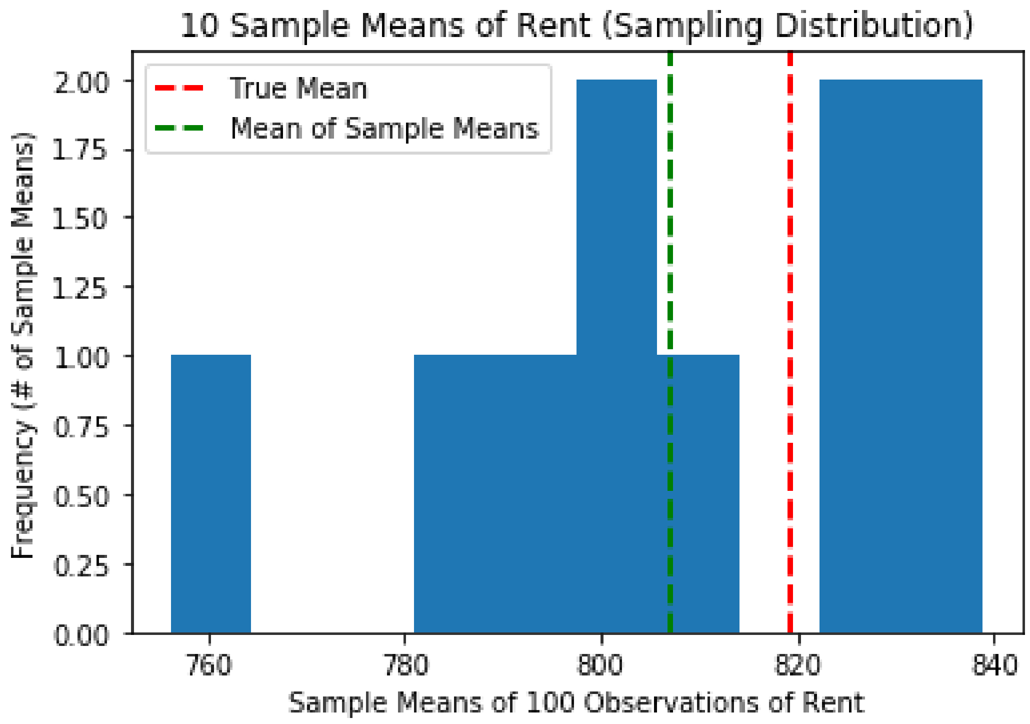
[24]:
```python
random_sample100 = rent.sample(n=100)
print(random_sample100)
print('Mean: ' + str(np.mean(random_sample100)))
```

```
1607      497
9792      891
3828      615
5434      827
10198     576
        …
16131    1007
19349     877
19161     766
20334     708
7438      418
Name: rent, Length: 100, dtype: int64
Mean: 807.36
```

4(e)

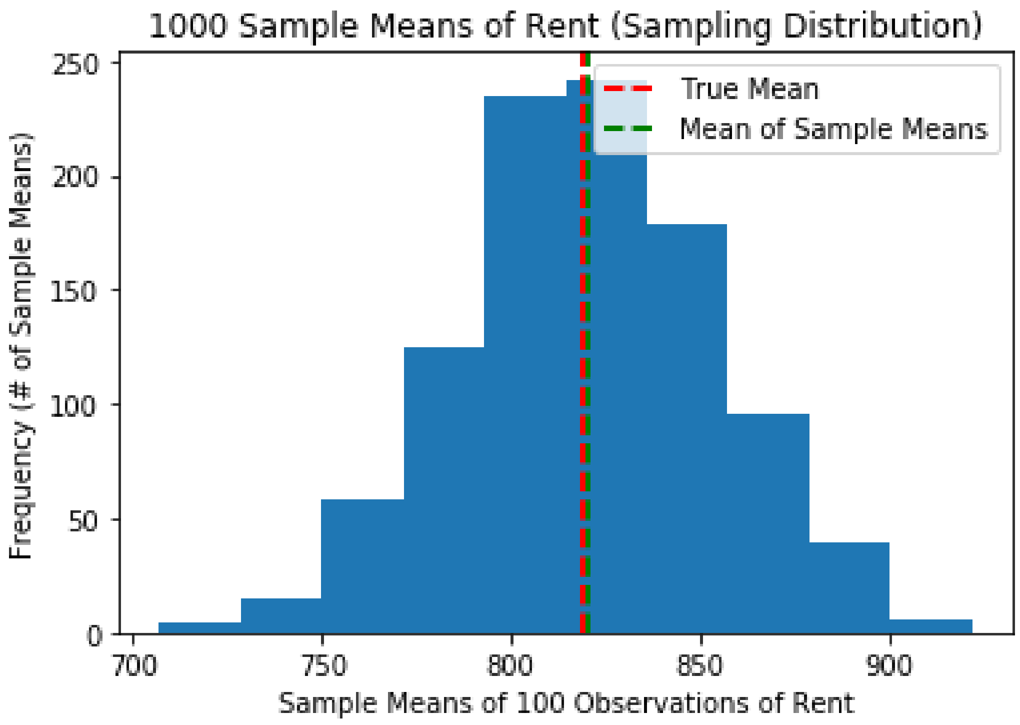
[25]:
```python
means_array = np.array([])

for outcome in range(10):
    means_array = np.append(means_array, (np.mean(rent.sample(n=100))))

print(means_array)

plt.hist(means_array)
plt.title('10 Sample Means of Rent (Sampling Distribution)')
plt.xlabel('Sample Means of 100 Observations of Rent')
plt.ylabel('Frequency (# of Sample Means)')
plt.axvline(819.2, color='red', lw = 2, ls = '--', label= 'True Mean')
plt.axvline(int(np.mean(means_array)), color='green', lw = 2, ls = '--', label=␣
  ↪'Mean of Sample Means')
plt.legend()


plt.show()
```

```
[838.11 798.63 799.31 787.97 756.23 827.55 792.04 810.8  838.86 828.45]
```



4(f)

[26]:
```
means_array = np.array([])

for outcome in range(1000):
    means_array = np.append(means_array, (np.mean(rent.sample(n=100))))

plt.hist(means_array)
plt.title('1000 Sample Means of Rent (Sampling Distribution)')
plt.xlabel('Sample Means of 100 Observations of Rent')
plt.ylabel('Frequency (# of Sample Means)')
plt.axvline(819.2, color='red', lw = 2, ls = '--', label= 'True Mean')
plt.axvline(int(np.mean(means_array)), color='green', lw = 2, ls = '--', label=␣
  ↪'Mean of Sample Means')
plt.legend()


plt.show()
```



4(g) As the number of samples increases, the mean of the sampling distribution (sampled means) gets closer and closer to the true mean, or the mean of the entire population. **4(h)**

As the number of samples increases, the shape of the sampling distribution gets closer and closer to a normal distribution where the mean, median, mode are all the same, and it is unimodal.

4(i)

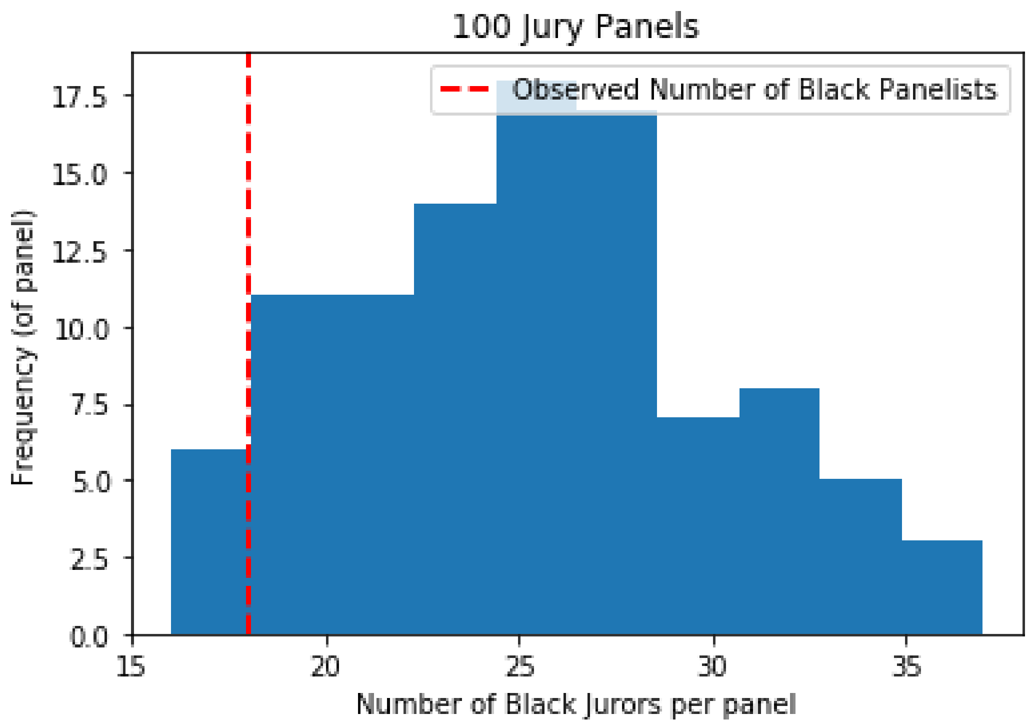
The Central Limit Theorem is the principle that predicts the outcomes regarding mean and shape of the sampling distribution as the number of samples increases. It states that if we take a reasonably large amount of samples, the sample size of the mean will end up to be normally distributed. It explains how the sampling distribution of the sample mean will be normal, and the average of sample means will be closer and closer to the population mean. We can see this in our sampling distribution of 10000 samples, because the data looks much more normally distributed and the av. of sample means is much closer to the pop. mean than in the distribution of 100 samples.

1.5 Question 5

5(a)

```
[18]: def simulate_panel(runs):
        '''
        Simulate a panel of 100 citizens, based on a demographic distribution.
        Return the count of Black panelists each time.
        '''

        counts = np.zeros(runs)

        for i in range(runs):
          panel = np.random.choice(["Black", "Non-Black"],
                                   size = 100,
                                   p = [0.25, 0.75])
          options, count = np.unique(panel, return_counts=True)
          counts[i] = count[0]

        return(counts)

      panel100 = simulate_panel(100)

      print(panel100)

      plt.hist(panel100)
      plt.title('100 Jury Panels')
      plt.xlabel('Number of Black Jurors per panel')
      plt.ylabel('Frequency (of panel)')
      plt.axvline(18, color='red', lw = 2, ls = '--', label= 'Observed Number of␣
        ↪Black Panelists')
      plt.legend()
      plt.show()
```

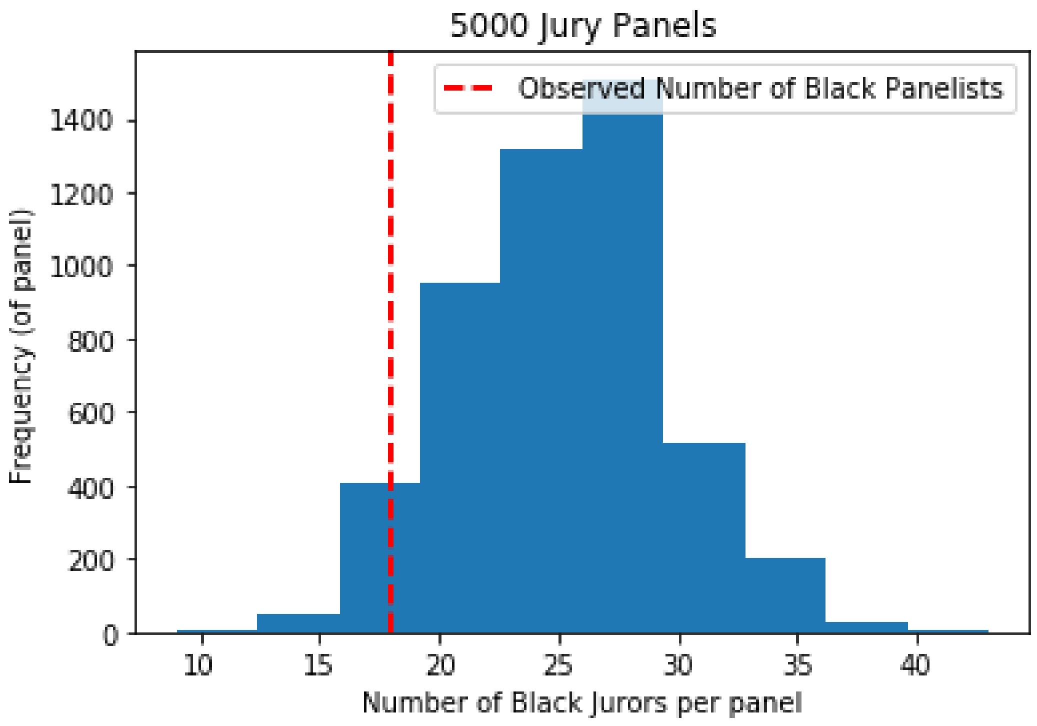
```
[27. 28. 20. 19. 32. 30. 26. 26. 21. 28. 22. 31. 26. 25. 18. 20. 29. 23.
 22. 22. 29. 24. 26. 30. 23. 23. 19. 19. 21. 23. 33. 33. 25. 26. 24. 17.
 25. 25. 23. 26. 26. 25. 35. 28. 31. 27. 27. 31. 33. 22. 28. 21. 23. 25.
 17. 27. 28. 24. 28. 37. 31. 32. 24. 30. 33. 27. 23. 17. 16. 31. 22. 24.
 21. 31. 29. 30. 24. 28. 36. 34. 20. 26. 18. 27. 27. 23. 25. 22. 20. 26.
 27. 28. 25. 20. 20. 20. 20. 26. 21. 28.]
```



5(b)

```
[19]: def simulate_panel(runs):
        '''
        Simulate a panel of 5000 citizens, based on a demographic distribution.
        Return the count of Black panelists each time.
        '''

        counts = np.zeros(runs)

        for i in range(runs):
          panel = np.random.choice(["Black", "Non-Black"],
                                   size = 100,
                                   p = [0.25, 0.75])
          options, count = np.unique(panel, return_counts=True)
          counts[i] = count[0]

        return(counts)

      panel5000 = simulate_panel(5000)

      print(panel5000)

      plt.hist(panel5000)
      plt.title('5000 Jury Panels')
      plt.xlabel('Number of Black Jurors per panel')
      plt.ylabel('Frequency (of panel)')
      plt.axvline(18, color='red', lw = 2, ls = '--', label= 'Observed Number of␣
        ↪Black Panelists')
      plt.legend()
      plt.show()
```

```
[34. 21. 23. … 27. 25. 29.]
```



5(c)

```
[20]: print('Number of panels with 18 or fewer Black panelists: ' + str((panel5000 <=␣
  ↪18).sum()))
print('Proportion of panels with <18 Black panelists: ' + str((panel5000 <= 18).
  ↪sum()/5000))
```

```
Number of panels with 18 or fewer Black panelists: 295
Proportion of panels with <18 Black panelists: 0.059
```

5(d)

Based on the histogram above and the proportion of randomized panels with 18 or fewer black panelists, the outcome is definitely possible, but the plausibility of it occuring is very rare/low considering only 5.9% of the panels had this outcome in our random simulation of 5,000 panels.

1.6 Question 6

6(a)

The dependent variable in this study is the opinion of NYC residents on budget allocations to public transportation.

6(b)

The independent variables in this study are subway stop and borough (specifically five busiest subway stops & in Manhattan/Brooklyn.

6(c)

This researcher conceptualizes “opinions of NYC residents on budget allocations to public transportation” by categorizing the different aspects of public transportation - subways, buses, bike lanes, and “other” - and asks the participants to choose one which they think more money should be spent on, in order to get their opinions on budget allocations to public transportation in an efficient way.

6(d)

This researcher operationalizes “opinions of NYC residents on budget allocations to public transportation” by going to the five busiest subway stops in the boroughs of both Manhattan and Brooklyn, and spending a specified time of 2 hours at each stop to ask people who walk by their opinions. Furthermore, they make the question relatively a “choose one of the options” question, which automatically measures/quantifies the opinions of allocating budget to public transportation; they make an individual choose which of the options they would want to spend money on (under the realm of public transportation).

6(e)

A plausible source of random error in the resulting dataset could be if researcher misheared someone in a subway station, and accidentally recording down the wrong answer/opinion, (ex. buses instead of bike lanes) and then inputted this wrong answer into the dataset. Another example of a random error could be if a researcher incorrectly inputted data into a dataset that was initially recorded correctly, such as putting in “buses” instead of “bike lanes”. For these minor errors, they will often cancel out over time.

6(f)

A plausible source of selection bias in this study is a sampling issue; in this study, the researchers are asking participants in the subways, who are in the act of using one of the specific modes of public transportation referenced in the study, therefore their answers are likely to be skewed in favor of subways because they themselves use them. If the researchers wanted to poll opinions of NYC residents (population of interest), which they claim they are doing, it would be best for them to do so in a more randomized, general way that captures more of the individuals in the entire population; here they are basically only getting the opinions of NYC subway users, which is likely biased.