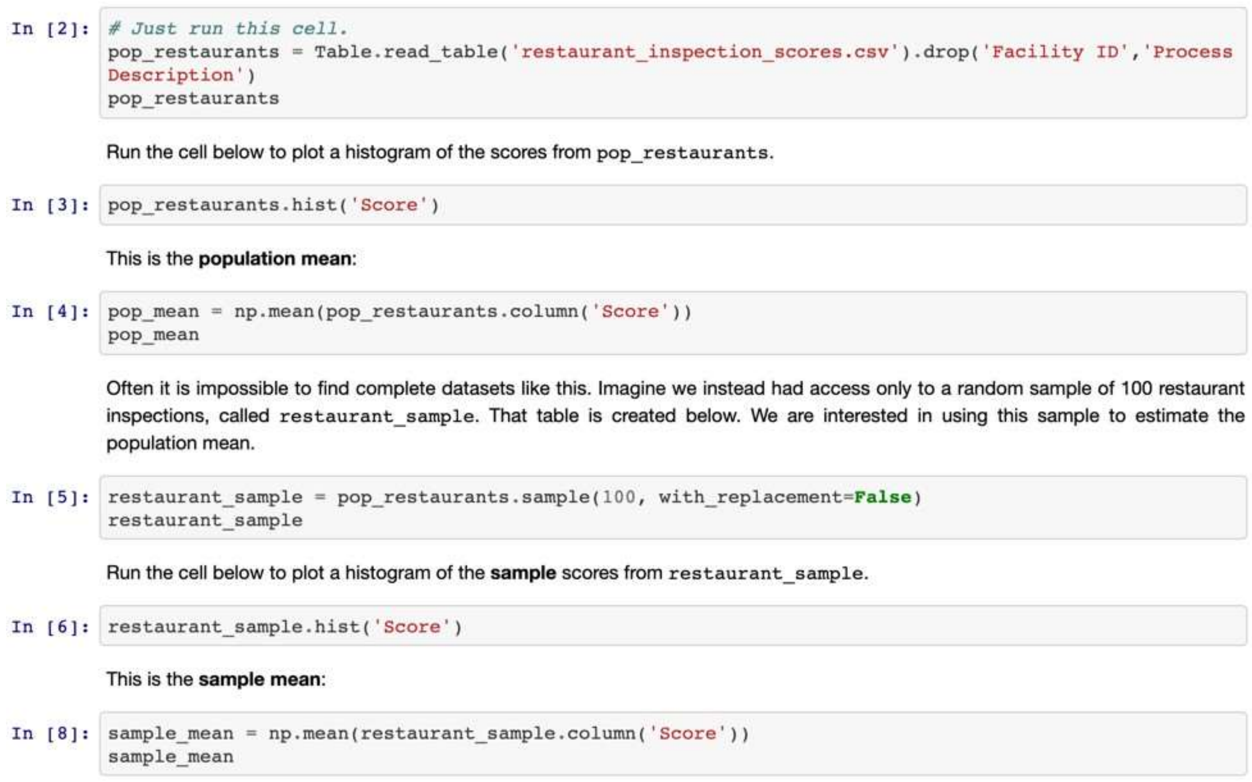
INFO5502 Assignment 6

Introduction

This assignment is copied from Data8.org, the original assignment can be found at github.com at: https://github.com/data-8/materials-su19/tree/master/materials/su19/hw/hw10. You may check the website for details. In this assignment, you will investigate Central Limit Theorem. 1 point for each question and there are total 10 points.

1 The Bootstrap and The Normal Curve

In this exercise, we will explore a dataset that includes the safety inspection scores for restaurants in the city of Austin, Texas. We will be interested in determining the average restaurant score for the city from a random sample of the scores; the average restaurant score is out of 100. We'll compare two methods for computing a confidence interval for that quantity: the bootstrap resampling method, and an approximation based on the Central Limit Theorem.



```
In [2]: # Just run this cell.
        pop_restaurants = Table.read_table('restaurant_inspection_scores.csv').drop('Facility ID','Process
        Description')
        pop_restaurants
```

Run the cell below to plot a histogram of the scores from `pop_restaurants`.

```
In [3]: pop_restaurants.hist('Score')
```

This is the **population mean**:

```
In [4]: pop_mean = np.mean(pop_restaurants.column('Score'))
        pop_mean
```

Often it is impossible to find complete datasets like this. Imagine we instead had access only to a random sample of 100 restaurant inspections, called `restaurant_sample`. That table is created below. We are interested in using this sample to estimate the population mean.

```
In [5]: restaurant_sample = pop_restaurants.sample(100, with_replacement=False)
        restaurant_sample
```

Run the cell below to plot a histogram of the **sample** scores from `restaurant_sample`.

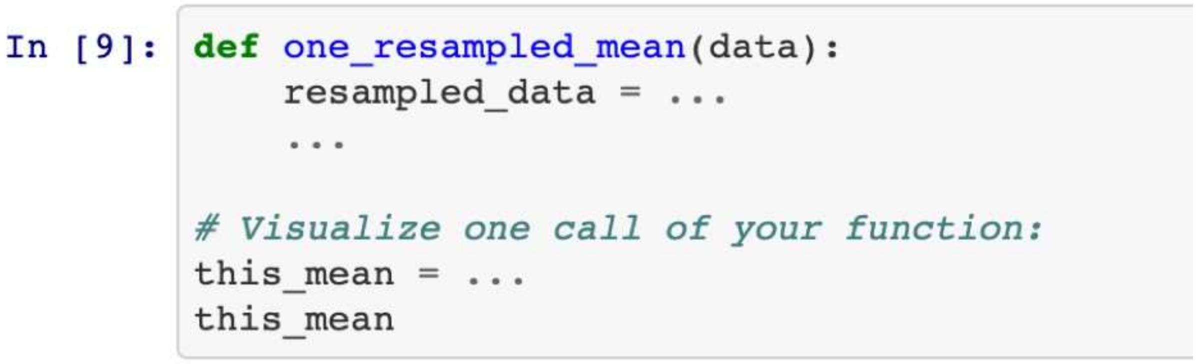
```
In [6]: restaurant_sample.hist('Score')
```

This is the **sample mean**:

```
In [8]: sample_mean = np.mean(restaurant_sample.column('Score'))
        sample_mean
```

1.1 Question 1

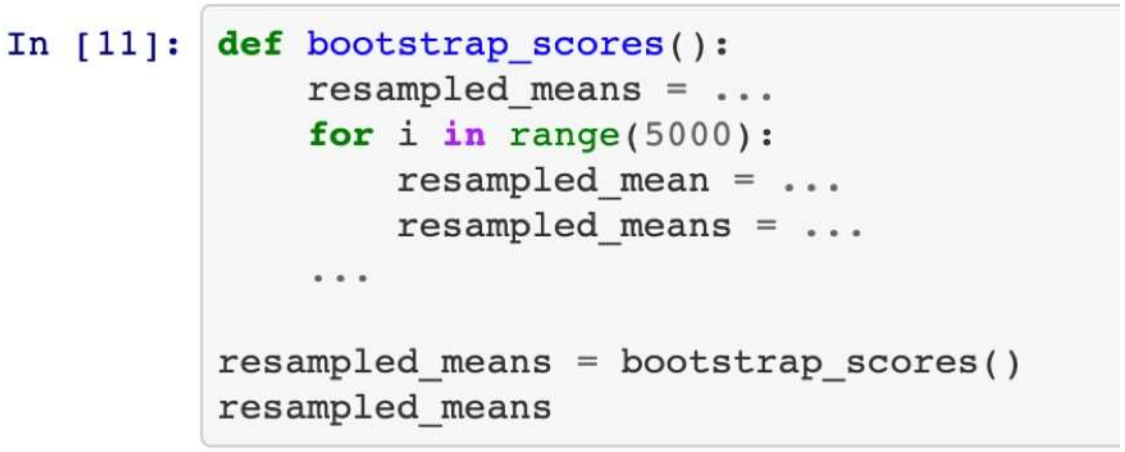
Complete the function *one_resampled_mean* below. It should take in an original table data, with a column Score, and return the mean score of one resampling from data.



```
In [9]: def one_resampled_mean(data):
            resampled_data = ...
            ...

        # Visualize one call of your function:
        this_mean = ...
        this_mean
```

1.2 Question 2

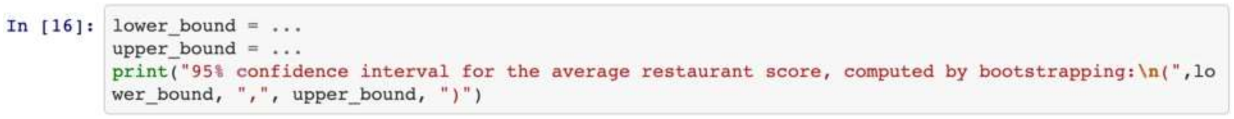
Complete the function *bootstrap_scores* below. It should take no arguments. It should simulate drawing 5000 resamples from *restaurant_sample* and compute the mean restaurant score in each resample. It should return an array of those 5000 resampled means.



```
In [11]: def bootstrap_scores():
             resampled_means = ...
             for i in range(5000):
                 resampled_mean = ...
                 resampled_means = ...
             ...

         resampled_means = bootstrap_scores()
         resampled_means
```

1.3 Question 3

Compute a 95 percent confidence interval for the average restaurant score using the array *resampled_means*.



```
In [16]: lower_bound = ...
         upper_bound = ...
         print("95% confidence interval for the average restaurant score, computed by bootstrapping:\n(",lo
         wer_bound, ",", upper_bound, ")")
```

1.4 Question 4

What distribution is the histogram between question 2 and 3 displaying (that is, what data are plotted), and why does it have that shape?

1.5 Question 5

Does the distribution of the sampled scores look normally distributed? State "yes" or "no" and describe in one sentence why you should expect this result.

1.6 Question 6

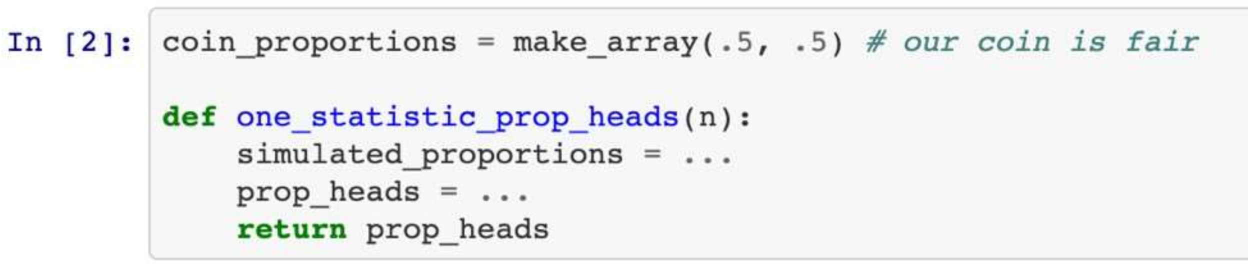
Without referencing the array *resampled_means* or performing any new simulations, calculate an interval around the *sample_mean* that covers approximately 95% of the numbers in the *resampled_means* array. This confidence interval should look very similar to the one you computed in Question 3.

2 Testing the Central Limit Theorem

To recap the properties we just saw: The Central Limit Theorem tells us that the probability distribution of the sum or average of a large random sample drawn with replacement will be roughly normal, regardless of the distribution of the population from which the sample is drawn.

2.1 Question 1

Define the function *one_statistic_prop_heads* which should return exactly one simulated statistic of the proportion of heads from *n* coinflips.



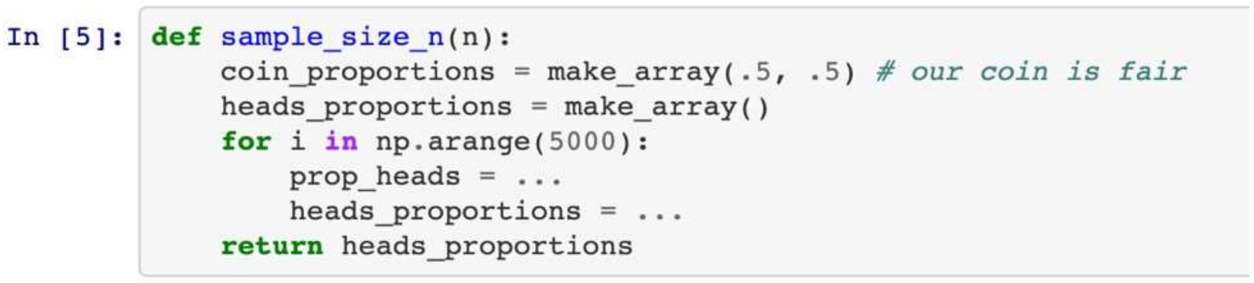
```
In [2]: coin_proportions = make_array(.5, .5) # our coin is fair

        def one_statistic_prop_heads(n):
            simulated_proportions = ...
            prop_heads = ...
            return prop_heads
```

2.2 Question 2

The CLT only applies when sample sizes are "sufficiently large." This isn't a very precise statement. Is 10 large? How about 50? The truth is that it depends both on the original population distribution and just how "normal" you want the result to look.

Consider a coin flip. If we say *Heads* is 1 and *Tails* is 0, then there's a 50% chance of getting a 1 and a 50% chance of getting a 0, which definitely doesn't match our definition of a normal distribution. The average of several coin tosses, where *Heads* is 1 and *Tails* is 0, is equal to the proportion of heads in those coin tosses (which is equivalent to the mean value of the coin tosses), so the CLT should hold true if we compute the sample proportion of heads many times.

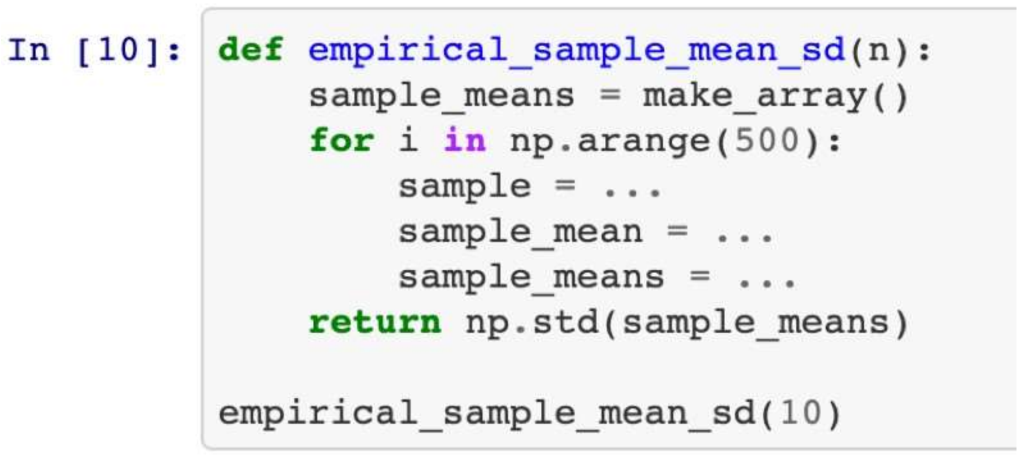
Write a function called *sample_size_n* that takes in a sample size *n*. It should return an array that contains 5000 sample proportions of heads, each from *n* coin flips.



```
In [5]: def sample_size_n(n):
            coin_proportions = make_array(.5, .5) # our coin is fair
            heads_proportions = make_array()
            for i in np.arange(5000):
                prop_heads = ...
                heads_proportions = ...
            return heads_proportions
```

2.3 Question 3

Write a function called *empirical_sample_mean_sd* that takes a sample size *n* as its argument. The function should simulate 500 samples with replacement of size *n* from the flight delays dataset, and it should return the standard deviation of the means of those 500 samples.


```
In [10]: def empirical_sample_mean_sd(n):
             sample_means = make_array()
             for i in np.arange(500):
                 sample = ...
                 sample_mean = ...
                 sample_means = ...
             return np.std(sample_means)

         empirical_sample_mean_sd(10)
```

2.4 Question 4

Now, write a function called *predict_sample_mean_sd* to find the predicted value of the standard deviation of means according to the relationship between the standard deviation of the sample mean and sample size that is discussed in the textbook. It takes a sample size *n* (a number) as its argument. It returns the predicted value of the standard deviation of the mean delay time for samples of size *n* from the flight delays (represented in the table *united*).