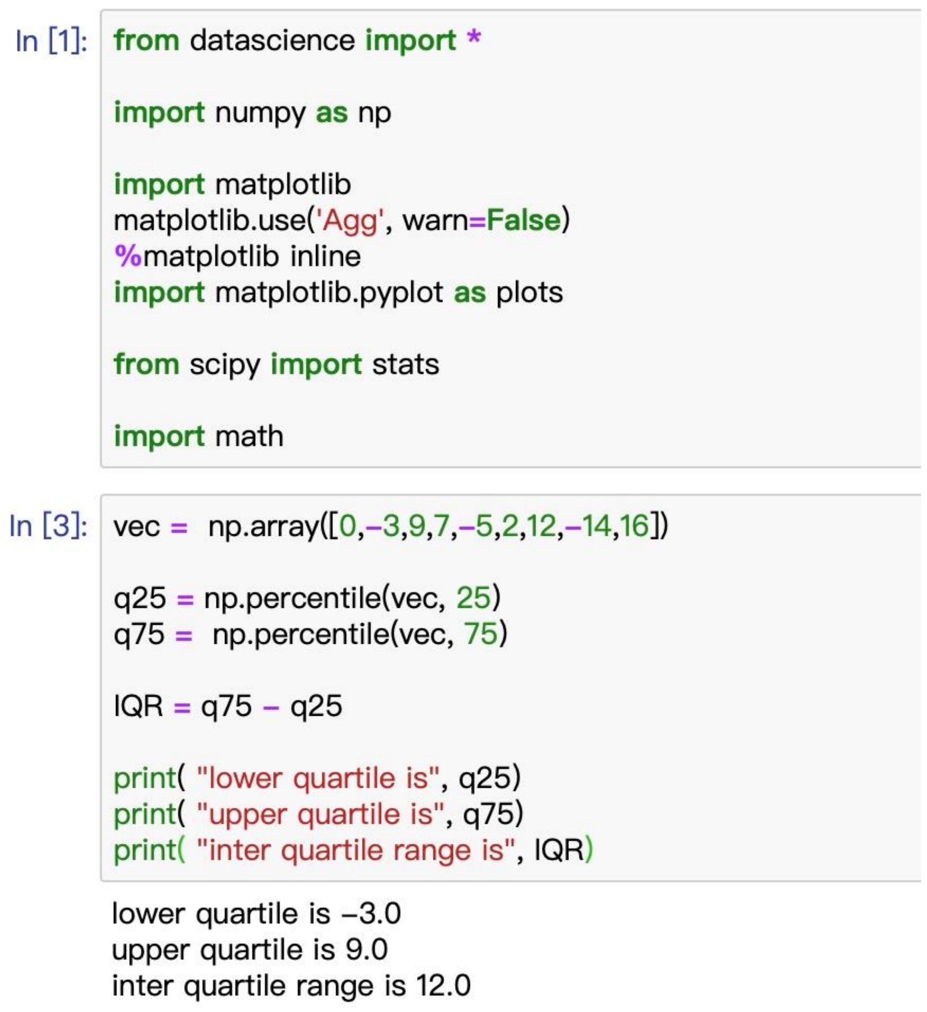
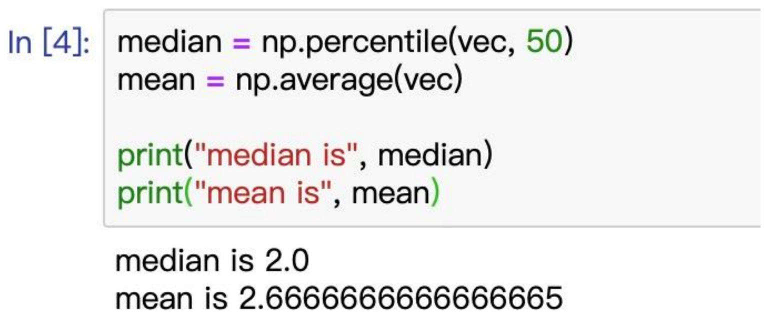
1. a)



```
In [1]: from datascience import *

        import numpy as np

        import matplotlib
        matplotlib.use('Agg', warn=False)
        %matplotlib inline
        import matplotlib.pyplot as plots

        from scipy import stats

        import math
```

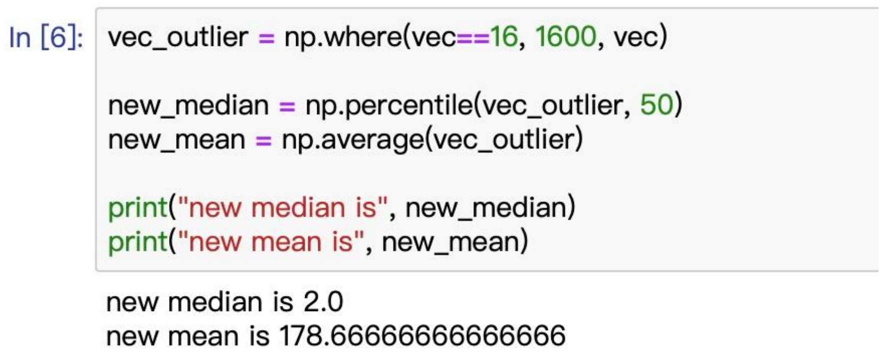
```
In [3]: vec =  np.array([0,-3,9,7,-5,2,12,-14,16])

        q25 = np.percentile(vec, 25)
        q75 =  np.percentile(vec, 75)

        IQR = q75 - q25

        print( "lower quartile is", q25)
        print( "upper quartile is", q75)
        print( "inter quartile range is", IQR)

        lower quartile is -3.0
        upper quartile is 9.0
        inter quartile range is 12.0
```

b)



```
In [4]: median = np.percentile(vec, 50)
        mean = np.average(vec)

        print("median is", median)
        print("mean is", mean)

        median is 2.0
        mean is 2.6666666666666665
```

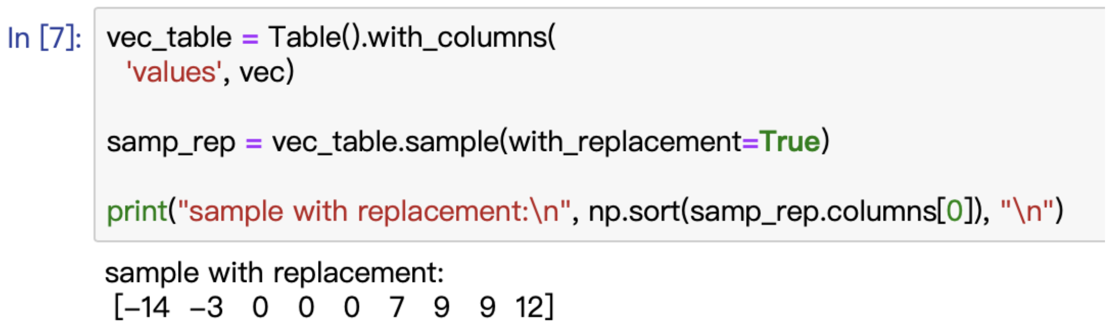
c)



```
In [6]: vec_outlier = np.where(vec==16, 1600, vec)

        new_median = np.percentile(vec_outlier, 50)
        new_mean = np.average(vec_outlier)

        print("new median is", new_median)
        print("new mean is", new_mean)

        new median is 2.0
        new mean is 178.66666666666666
```

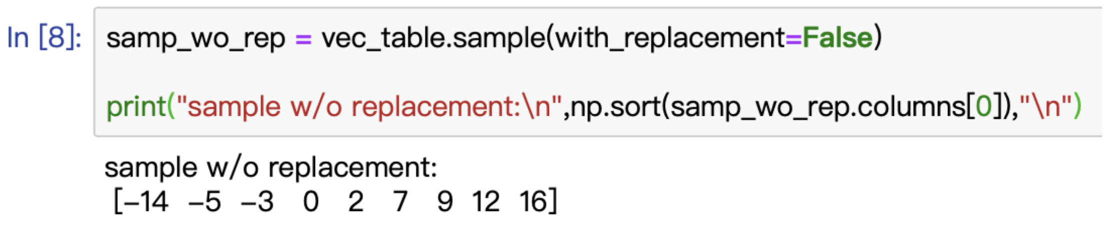
d) This means that median of an array is more sensitive to outliers. It would be less likely to report great error if one or two great outlier arrives in the sample.

2. a)



```
In [7]: vec_table = Table().with_columns(
          'values', vec)

        samp_rep = vec_table.sample(with_replacement=True)

        print("sample with replacement:\n", np.sort(samp_rep.columns[0]), "\n")

        sample with replacement:
         [-14  -3   0   0   0   7   9   9  12]
```

b)



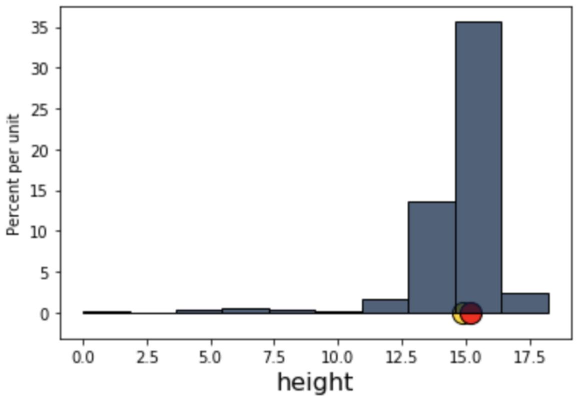
```
In [8]: samp_wo_rep = vec_table.sample(with_replacement=False)

        print("sample w/o replacement:\n",np.sort(samp_wo_rep.columns[0]),"\n")

        sample w/o replacement:
         [-14  -5  -3   0   2   7   9  12  16]
```

It’s absolutely the same as the original sample. Because the original sample contains only 9 elements, and we should take 9 of them without replacement, we have to take all of them once into the sample.

3. a)

```
In [12]: horses = Table.read_table('horses.csv')

         horses.select('height').hist()

         plots.scatter(np.average(horses.column("height")), 0, color='gold', s=200, edgecolors="black");
         plots.scatter(percentile(50, horses.column("height")), 0, color= 'red', s=200,  edgecolors="black")

         print("median is ", percentile(50, horses.column("height")))
         print("mean is" ,np.average(horses.column("height")))
```

```
median is  15.182
mean is 14.860920750782064
```

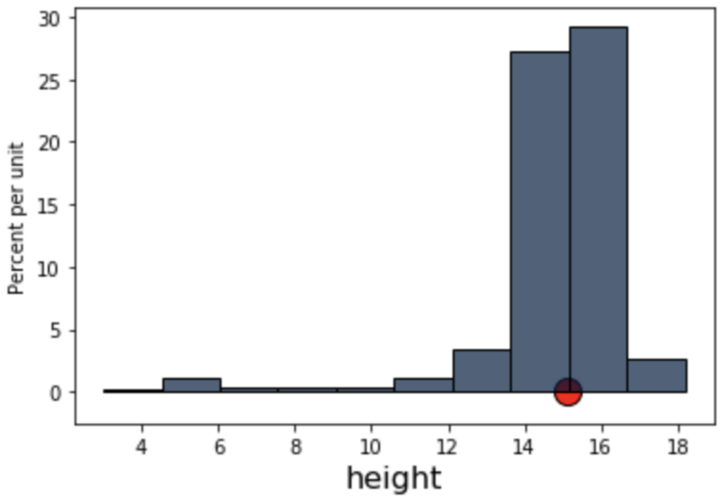


b) The distribution is skewed to the left.

c)

```
In [15]: a_resample = horses.sample(with_replacement=True)

         a_resample.select('height').hist()

         samp_median= percentile(50, a_resample.column('height'))
         print("resample median is:",samp_median)
         plots.scatter(samp_median, 0, color='red', s=200, edgecolors="black");
```

```
resample median is: 15.112
```

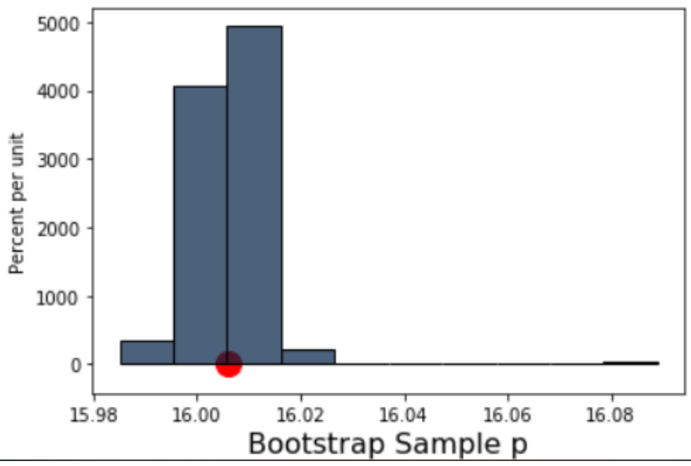


d)

```
In [29]: def bootstrap_per(perc, original_sample, label, replications):

             just_one_column = original_sample.select(label)
             pers = make_array()
             for i in np.arange(replications):
                 bootstrap_sample = just_one_column.sample()
                 resampled_per = percentile(perc, bootstrap_sample.column(0))
                 pers = np.append(pers, resampled_per)

             return pers
```

```
In [13]: bstrap_ps = bootstrap_per(75, horses, 'height', 10000)

         resampledps = Table().with_column('Bootstrap Sample p', bstrap_ps)
```

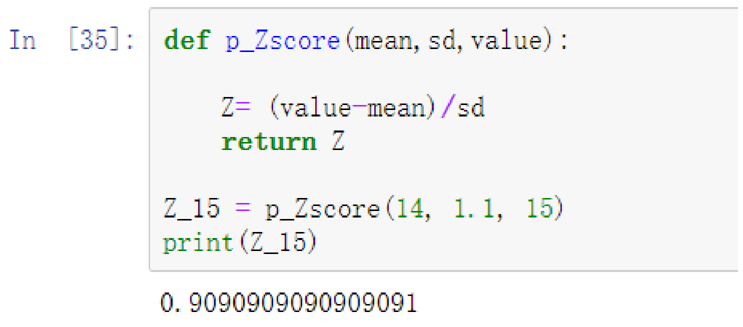
```
In [14]: pop_p = percentile(75, horses.column("height"))

         resampledps.hist()
         plots.scatter(pop_p, 0, color='red', s=200);
```



4.

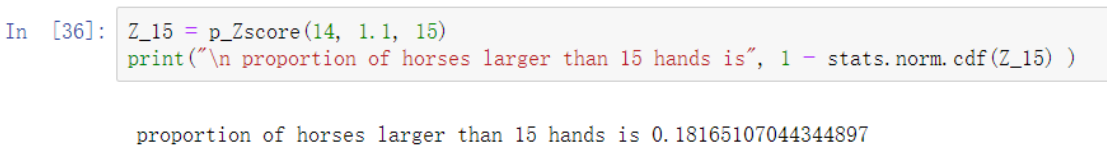
```
In [20]: print("Standard deviation of height is", np.std(horses.column("height")), "hands")

         print("\nVariance of height is", np.std(horses.column("height"))**2, "hands squared")

         Standard deviation of height is 1.8354099745673877 hands

         Variance of height is 3.368729774741459 hands squared
```

5. a)



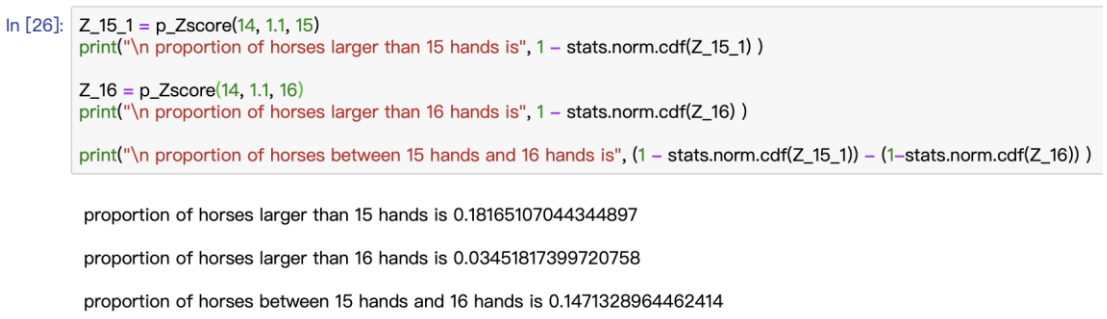
```
In [35]: def p_Zscore(mean, sd, value):

             Z= (value-mean)/sd
             return Z

         Z_15 = p_Zscore(14, 1.1, 15)
         print(Z_15)

         0.9090909090909091
```

b)



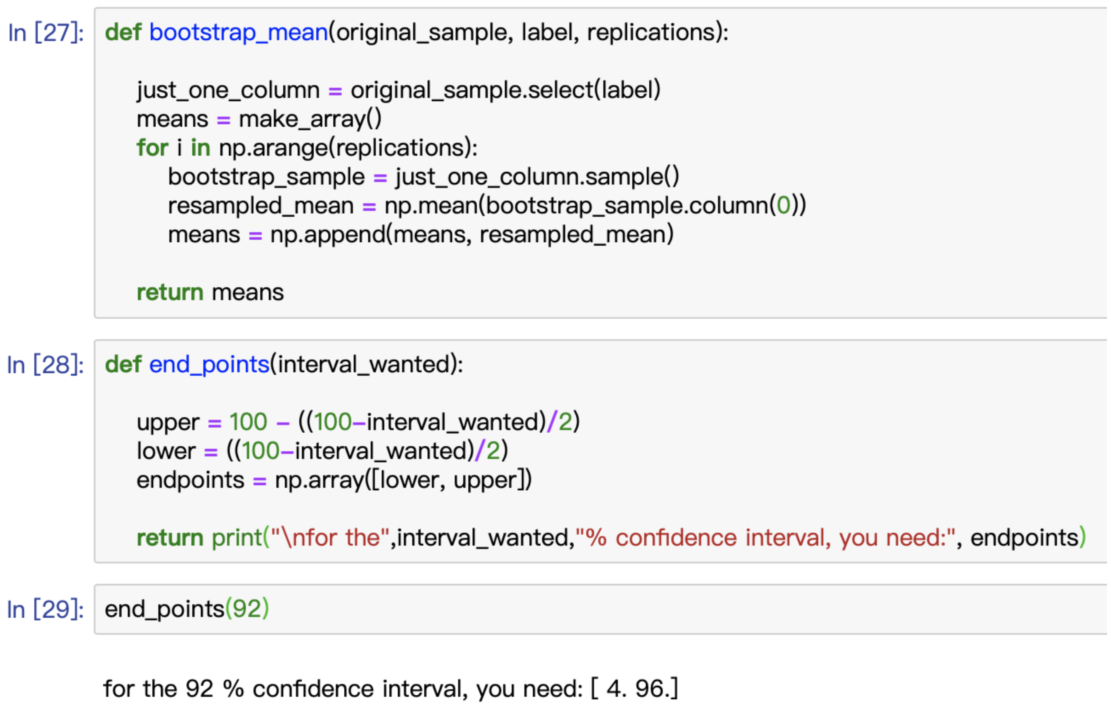
```
In [36]: Z_15 = p_Zscore(14, 1.1, 15)
         print("\n proportion of horses larger than 15 hands is", 1 - stats.norm.cdf(Z_15) )


          proportion of horses larger than 15 hands is 0.18165107044344897
```

c)



```
In [26]: Z_15_1 = p_Zscore(14, 1.1, 15)
         print("\n proportion of horses larger than 15 hands is", 1 - stats.norm.cdf(Z_15_1) )

         Z_16 = p_Zscore(14, 1.1, 16)
         print("\n proportion of horses larger than 16 hands is", 1 - stats.norm.cdf(Z_16) )

         print("\n proportion of horses between 15 hands and 16 hands is", (1 - stats.norm.cdf(Z_15_1)) - (1-stats.norm.cdf(Z_16)) )


          proportion of horses larger than 15 hands is 0.18165107044344897

          proportion of horses larger than 16 hands is 0.03451817399720758

          proportion of horses between 15 hands and 16 hands is 0.1471328964462414
```

6. a)

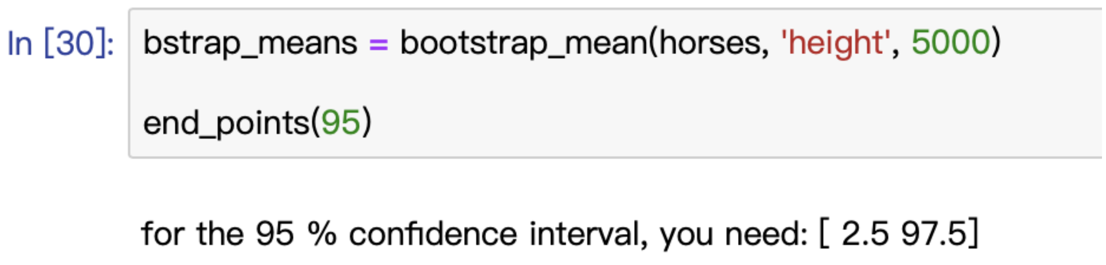


```
In [27]: def bootstrap_mean(original_sample, label, replications):

             just_one_column = original_sample.select(label)
             means = make_array()
             for i in np.arange(replications):
                 bootstrap_sample = just_one_column.sample()
                 resampled_mean = np.mean(bootstrap_sample.column(0))
                 means = np.append(means, resampled_mean)

             return means
```

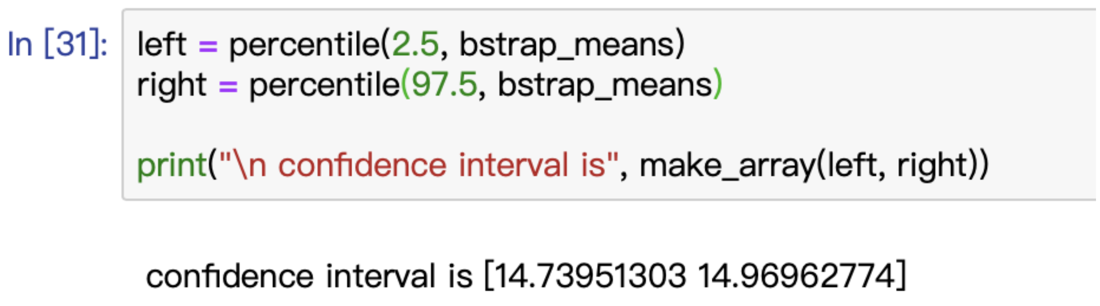
```
In [28]: def end_points(interval_wanted):

             upper = 100 - ((100-interval_wanted)/2)
             lower = ((100-interval_wanted)/2)
             endpoints = np.array([lower, upper])

             return print("\nfor the",interval_wanted,"% confidence interval, you need:", endpoints)
```

```
In [29]: end_points(92)

         for the 92 % confidence interval, you need: [ 4. 96.]
```

b)



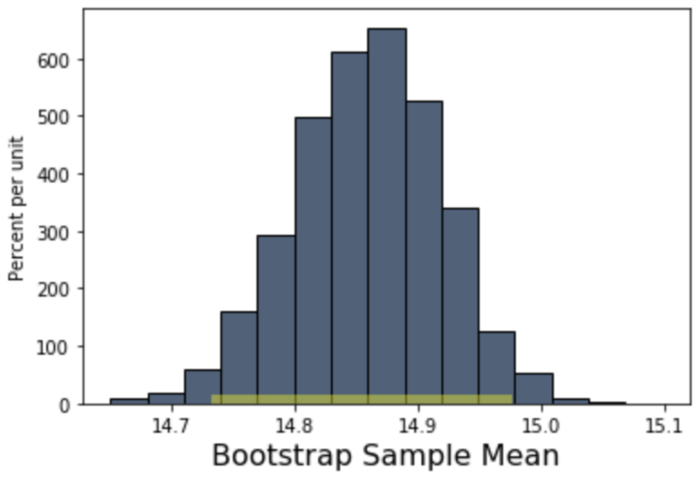
```
In [30]: bstrap_means = bootstrap_mean(horses, 'height', 5000)

         end_points(95)

         for the 95 % confidence interval, you need: [ 2.5 97.5]
```



```
In [31]: left = percentile(2.5, bstrap_means)
         right = percentile(97.5, bstrap_means)

         print("\n confidence interval is", make_array(left, right))

          confidence interval is [14.73951303 14.96962774]
```

c)

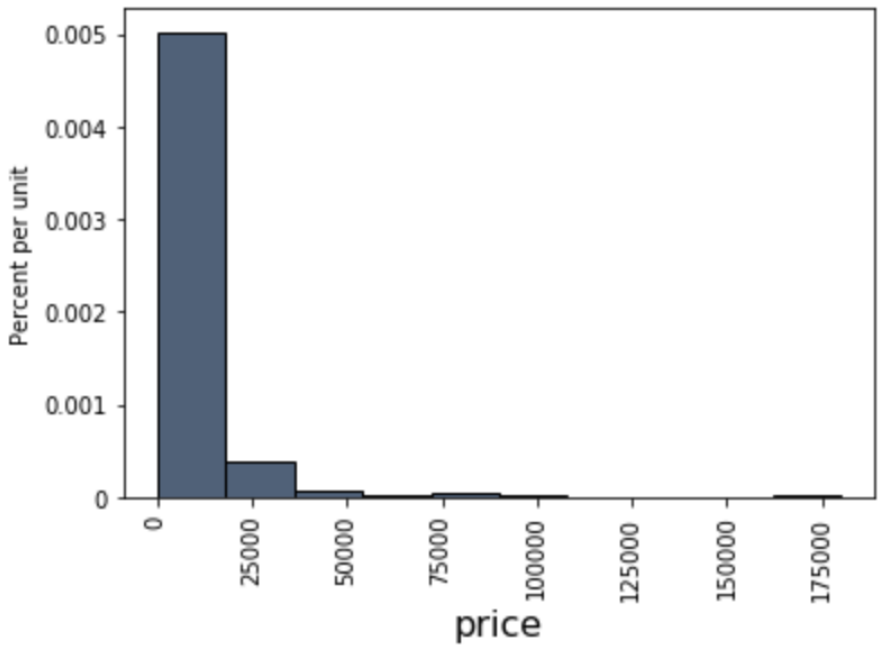
```
In [32]: resampled_means = Table().with_column(
             'Bootstrap Sample Mean', bstrap_means
         )
         resampled_means.hist(bins=15)

         plots.plot(make_array(left, right), make_array(0, 0), color='yellow', lw=10, alpha=.4)

Out[32]: [<matplotlib.lines.Line2D at 0x2b3a2cf012e8>]
```



d) I reject the null hypothesis because 14 is not included in the 95% confidence interval which is [14.73951303 14.96962774].

7. a)

```
In [33]: horses.select('price').hist()
```



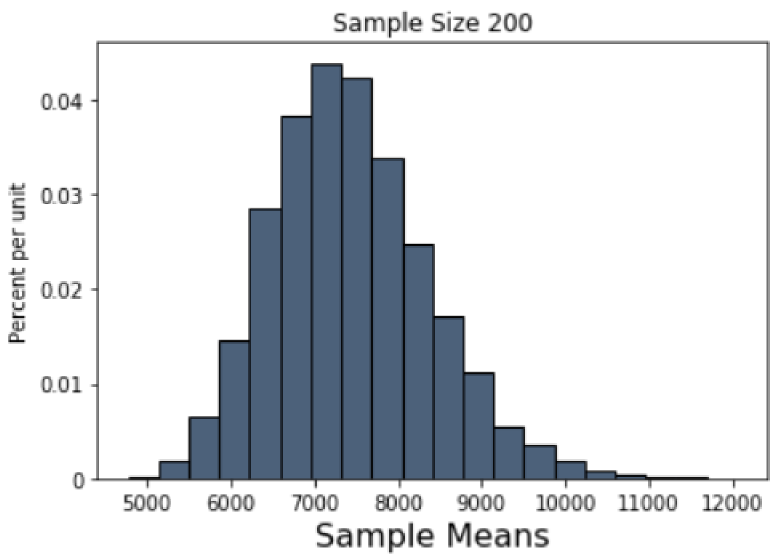
It's not normally distributed. It's because the sample size is not large enough.

b)

```
In  [34]: def simulate_sample_mean(table, label, sample_size, repetitions):

              means = make_array()

              for i in range(repetitions):
                  new_sample = table.sample(sample_size)
                  new_sample_mean = np.mean(new_sample.column(label))
                  means = np.append(means, new_sample_mean)

              sample_means = Table().with_column('Sample Means', means)

              # Display empirical histogram and print all relevant quantities
              sample_means.hist(bins=20)
              plots.xlabel('Sample Means')
              plots.title('Sample Size ' + str(sample_size))
              print("Sample size: ", sample_size)
              print("Population mean:", np.mean(table.column(label)))
              print("Average of sample means: ", np.mean(means))
              print("Population SD:", np.std(table.column(label)))
              print("SD of sample means:", np.std(means))
```

```
In  [35]: simulate_sample_mean(horses, 'price', 200, 10000)
```

```
Sample size:  200
Population mean: 7439.958289885297
Average of sample means:  7427.2755755
Population SD: 13271.68966481892
SD of sample means: 936.9190883258309
```



c) The mean of sampling distribution is close to the population mean, but not the same.

d) It's approximately normal. Because according to the central limit theorem, the sample size is large enough that the sampling distribution is approximately normal.

e)

```
In  [28]: standard_error = np.std(horses.column("price"))/math.sqrt(200)

          print("analytical Standard Error is",standard_error)
```

```
analytical Standard Error is 938.4501759796875
```

8. a) The margin of error of the poll is 4.3%.

   b) The confidence interval around Kamala Harris's percentage of vote is [1.7pts 10.3pts].

   c) The confidence interval around Pete Buttigieg's percentage of vote is [2.7pts 11.3pts].

   d) Because Harris's "true" performance is within the confidence interval of Buttigieg's percentage of vote, and Buttigieg's "true" performance is within the confidence interval of Harris's percentage of vote, it is not clear who is doing better in the country as a whole.