Question 1

Question 2

In order to perform statistical computations and data visualization (also to load the data in the `.mat` format into the `Python` environment) some libraries were necessary. The `Python` solution code for this question, assumes that the following lines are executed:

```
# Necassary imports
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat # to be able to use .mat file in the python environment
```

The `loadmat` function in the `scipy.io` package was necessary to load the dataset, this can be done by executing the lines:

```
data = loadmat('XData.mat')
x_data = data['x'].flatten()
print('The dataset: ', x_data, '\n')
```

a) Mean and median tells about where the data is centered. Depending on the specifics of a dataset such as the frequency of the outliers, which are essentially the data points that are distant from others, one of these measures may be more preferable. Overall, these measures establish a solid baseline for the underlying distribution that generated the data.

To compute the sample mean and the sample median of the dataset, numpy functions `np.mean()` and `np.median()` are used:

```
mean = np.mean(x_data)
median = np.median(x_data)
print('Sample mean: %f \nSample median: %f\n' % (mean, median))
```

The output of the script gives the results:

```
Sample mean: 98.436667
Sample median: 97.850000
```

b) One can also analyze the dispersion of a dataset to gain further understanding about the underlying distribution. The standard deviation and the inter-quantile range are descriptive measures that are used to quantify the dispersion of the data points.

Similarly the sample standard deviation is computed using `np.std()`. For the sample inter-quantile range computation, the first and the third quantiles are found with the help of the function `np.percentile()` and their difference is computed. The script is provided below:

```
std = np.std(x_data)
q25, q75 = np.percentile(x_data, [25, 75])
iqr = q75 - q25
print('Sample standard deviation: %f \nSample inter-quartile range: %f\n' % (std, iqr))
```

The output is:

```
Sample standard deviation: 10.056523
Sample inter-quartile range: 9.950000
```

c) An histogram is a type of plot that visually represents the frequency distribution of data points. It correlates to the likelihood function since frequently observed data points are likelier to occur in the future.

`Matplotlib`, a 2D plotting library in `Python` is used to draw the histograms. The function calls in the library are self-explanatory. The code that plots the asked histograms is given below:

```
x_range = [70, 130]
num_bins = [3, 6, 12]
colors = ['g', 'r', 'b']

print('The histograms:')
for i in range(3):
    plt.figure(figure_num)
    figure_num += 1
    plt.title('Histogram of the aggragated responses')
    plt.ylabel('Counts')
    plt.xlabel('Aggragated responses')
    plt.hist(x_data.flatten(), bins=num_bins[i], range=x_range, color=colors[i], edgecolor='black', linewidth
        =1.2)
    plt.show(block=False)
```

Note that the very last line `plt.show(block=False)` has the `block=False` argument, because when `block=True` (the default value) the execution stops and the program waits for the user to press the `Enter` key. To prevent this behavior, `plt.show()` line is added at the very end of the solution code of Question 2; this way the program executes completely and the plots are displayed together afterwards.

The histograms drawn by the script above follow. Notice that as the number of bins increases the histograms become more detailed; also they give the impression that the data is normally distributed. Under the assumption that we have enough data, it looks like the histogram will converge to a Gaussian as the number of bins approaches infinity.

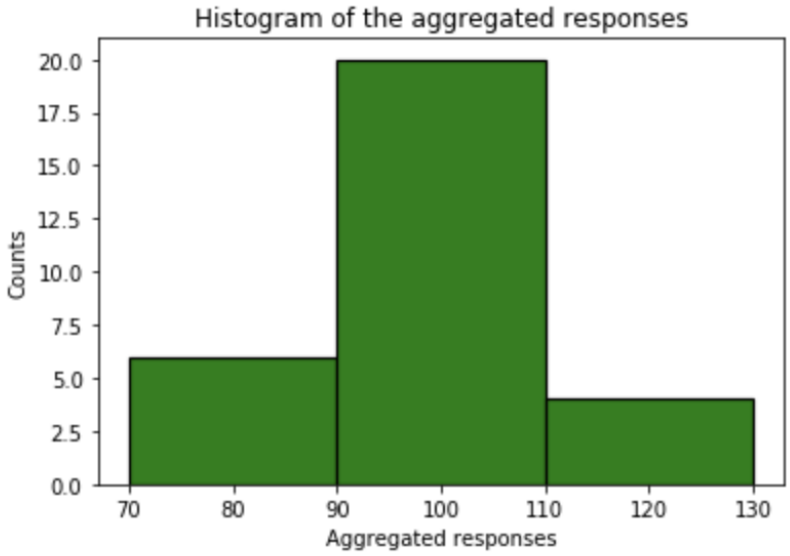

Figure 1: Histogram of the aggregated responses with 3 bins.

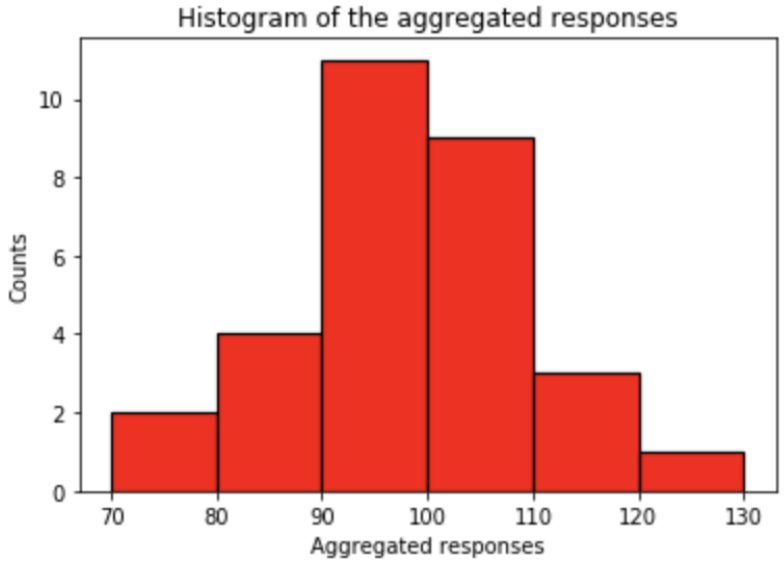

Figure 2: Histogram of the aggregated responses with 6 bins.

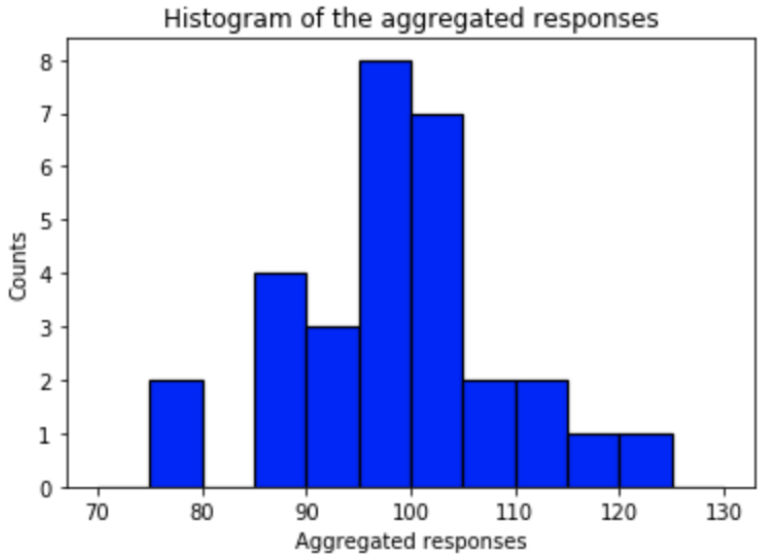

Figure 3: Histogram of the aggregated responses with 12 bins.

d) The Normal Quantile Plot is basically a procedure to visually check whether the data came from the Normal distribution or not. It simply sorts the sample data in ascending order and plots the quantiles against the quantiles drawn from the Normal distribution. If the resulting scatter plot forms a roughly straight line, it means that the both set of quantiles came from the Normal distribution. [1]

The `Matlab` Normal Quantile Plot code given in the assignment can be translated into `Python` as:

```
# Python equivalent of the Matlab Normal Quantile Plot code given in the assignment
y = np.sort(x_data)
n = np.size(x_data)
f = (np.arange(1, n + 1) − 3/ 8) / (n + 1 / 4)
q = 4.91 ∗ (f ∗∗ 0.14 − (1 − f) ∗∗ 0.14)
plt.grid()
plt.plot(q, y, '∗−')
```

The plot produced by this script is given in the next page. This plot conforms with the expectation that the data is normally distributed since the points form a roughly straight line.

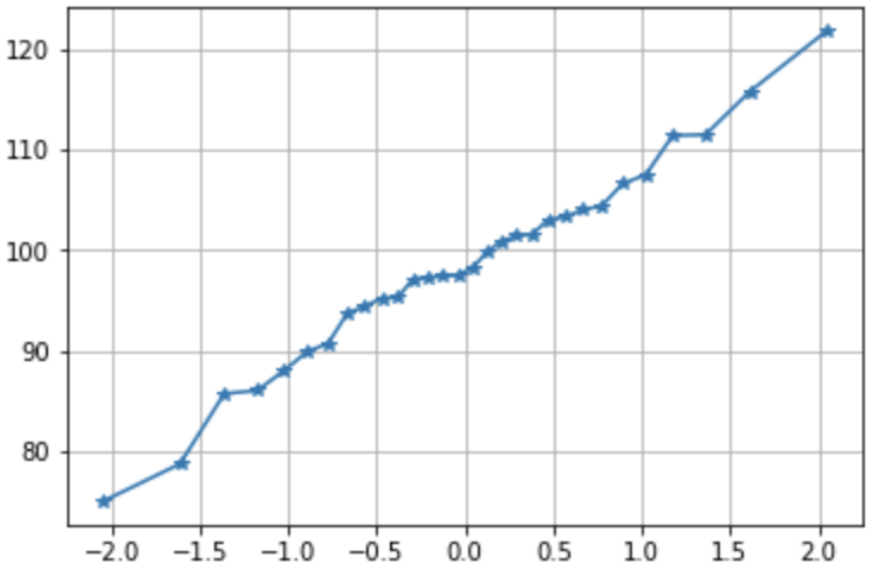

Figure 4: Histogram of the aggregated responses with 12 bins.

e) Bootstrap resampling is a method to approximate the properties of a distribution using a generalized approximation of the dataset. The method begins by randomly picking samples from the dataset with replacement. Then the properties of interest are computed again for these resamples. Later, the means of the properties together with their standard errors can be reported as valid approximations. This method is quite useful in practice since it can artificially produce up to n^n resamples, where n is the number of samples. [2]

A function to perform bootstrap resampling can be implemented as follows:

```
# Function to perform bootstrap resampling
    def bootstrap_resampling(data, bootstraps, stat_func):
    """
    Given the data, number of bootstrapping iterations and the statistic of
    interest; this function returns a list of statistics obtained from each
    bootstrapped sample of the original data.
    Args:
        data: The original data
        bootstraps: The number of bootstrapping iterations
        stat_func: The function to compute the statistic of interest
            (e.g.: np.mean, np.std, etc.)
    Returns:
        bootstrap_replicate: The mean of all bootstrap replicates
        standard_error: The standard error of the bootstrap replicate
        bootstrap_replicates: The list of statistics obtained from the
            bootstrapped samples
        """
```

```
        bootstrap_replicates = []
            sample_size = np.size(data)
            for _ in range(bootstraps):
                bootstrap_sample = np.random.choice(data, sample_size, replace=True)
                bootstrap_replicates.append(stat_func(bootstrap_sample))
            return np.mean(bootstrap_replicates), np.std(bootstrap_replicates), bootstrap_replicates
```

Another important statistical concept is confidence interval. Confidence interval refers to the range of values that is likely to include an unseen data point generated by the underlying distribution. In particular, 95% confidence interval refers to the range in which you can be 95% percent sure that your statistics are sound. In other words, it is the range where your results will lie in 95% of your experiments.

An implementation of a function that computes the confidence interval for a dataset is given:

```
# Function to compute the confidence interval of data samples
def compute_confidence_interval(data, confidence):
    """
    Given the data and the confidence level, computes the confidence interval
    of the data samples.
    Args:
        data: The given data
        confidence: The confidence level, known as alpha (between 0 and 100)
    Returns:
        lower: The lowerbound of the confidence interval
        upper: The upperbound of the confidence interval
    """
    sorted_data = np.sort(data)
    lower = np.percentile(sorted_data, (100 - confidence) / 2)
    upper = np.percentile(sorted_data, confidence + (100 - confidence) / 2)
    return lower, upper
```

The mean of the distribution can be approximated from 1000 bootsrap resamples using the functions above as:

```
# Performing the bootstrap sampling and computing the bootstrap replicate
# of the mean together with its standard error
bootstrap_mean, standard_error_mean, bootstrap_means = bootstrap_resampling(x_data, 1000, np.mean)
print('The sample mean computed from %s bootstrap samples: %f' % (1000, bootstrap_mean))
print('The bootstrapped estimate for the standard error of the mean:', standard_error_mean)

# Computing the 95% confidence interval
lower, upper = compute_confidence_interval(bootstrap_means, 95)
confidence_interval = (lower, upper)
print('The 95% confidence interval of the mean:', confidence_interval)

print('Overall, mean of the dataset can be reported as: %.3f +- %.3f, (%.3f, %.3f)\n' %
      (bootstrap_mean, standard_error_mean, lower, upper))
```

`np.std()` is used to compute the standard error, since it is simply the standard deviation of the sampling distribution. The output of the script is:

```
The sample mean computed from 1000 bootstrap samples: 98.438220
The bootstrapped estimate for the standard error of the mean: 1.798037377080295
The 95% confidence interval of the mean: (95.12241666666664, 102.17050000000002)
Overall, mean of the dataset can be reported as: 98.438 +- 1.798, (95.122, 102.171)
```

An histogram of the means of the bootstrap resamples can be plotted using the same procedures used in the previous parts. Such an histogram with 50 bins is shown below together with the lines to plot it:

```
1 plt.title('Histogram of the bootstrapped means')
2 plt.ylabel('Counts')
3 plt.xlabel('Means')
4 plt.hist(bootstrap_means, bins=50, edgecolor='black', linewidth=1.2)
5 plt.show(block=False)
```

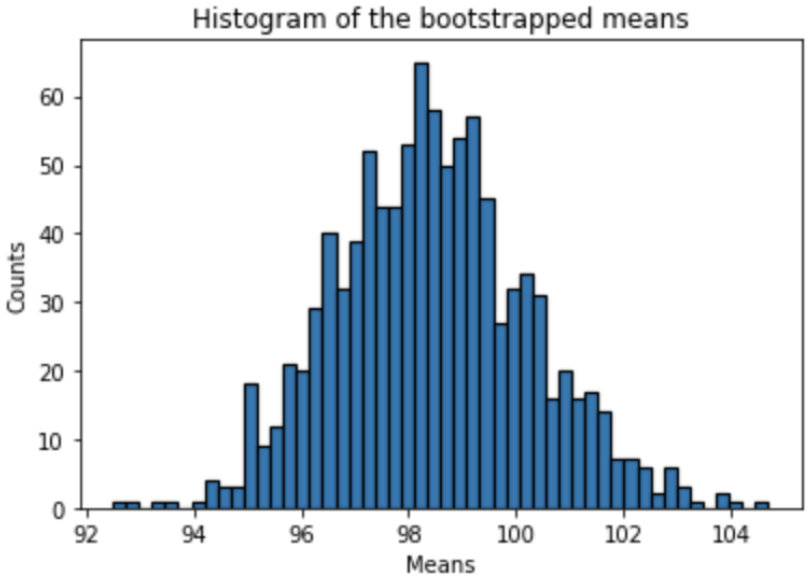

Figure 5: Histogram of the means of the bootstrap resamples with 50 bins.

f) The exact same steps are fallowed to estimate the standard deviation of the underlying distribution. The only change is that the statistic of interest is now the standard deviation instead of the mean. The corresponding script follows:

```
1 # Performing the bootstrap sampling and computing the bootstrap replicate of the
2 # standard deviation together with its standard error
```

```
bootstrap_std, standard_error_std, bootstrap_stds = bootstrap_resampling(x_data, 1000, np.std)
print('The sample standard deviation computed from %s bootstrap samples: %f' % (1000, bootstrap_std))
print('The bootstrapped estimate for the standard error of the standard deviation:', standard_error_std)

# Computing the 95% confidence interval
lower, upper = compute_confidence_interval(bootstrap_stds, 95)
confidence_interval = (lower, upper)
print('The 95% confidence interval of the standard deviation:', confidence_interval)

print('Overall, standard deviation of the dataset can be reported as: %.3f +- %.3f, (%.3f, %.3f)\n'
      % (bootstrap_std, standard_error_std, lower, upper))
```

The output of this script and the histogram of the standard deviations of the bootstrap resamples with 50 bins are:

```
The sample standard deviation computed from 1000 bootstrap samples: 9.833668
The bootstrapped estimate for the standard error of the standard deviation: 1.3571556300969863
The 95% confidence interval of the standard deviation: (7.173218592238309, 12.5811392567476)
Overall, standard deviation of the dataset can be reported as: 9.834 +- 1.357, (7.173, 12.581)
```

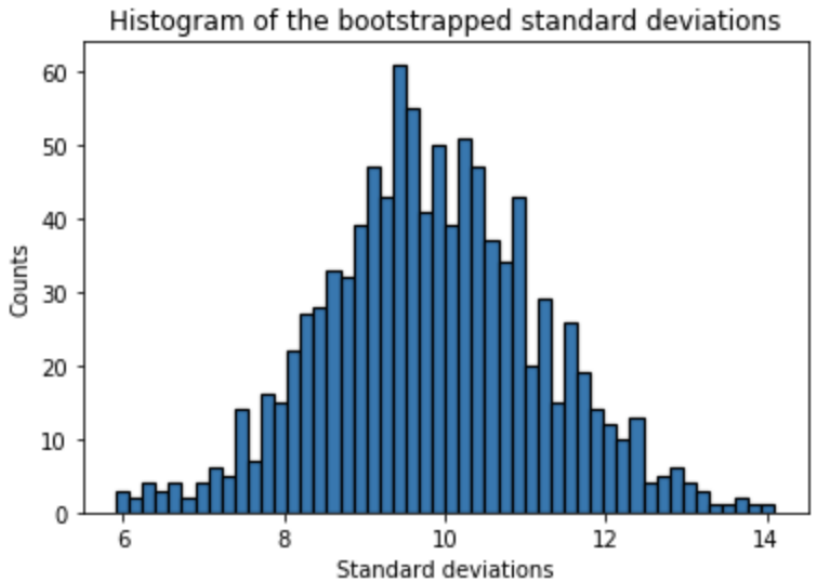

Figure 6: Histogram of the standard deviations of the bootstrap resamples with 50 bins.

g) Jackknife resampling is yet another resampling procedure to obtain valid approximations for statistics of interest. It pre-dates most of the resampling methods and it is relatively simpler. It operates by systematically leaving out one sample and considering the resulting subsample as a resample. Then, it computes the statistics of interest on these resamples. The number of resamples it can produce has an upper bound of n, where n is again the number of samples. [3]

A function that performs jackknife resampling is provided below:

```
# Function to perform Jacknife resampling
def jackknife_resampling(data, stat_func):
    """
    Given the data and the statistic of interest; this function returns a
    list of statistics obtained from each jacknifed sample of the original data.
    Args:
        data: The original data
        stat_func: The function to compute the statistic of interest
            (e.g.: np.mean, np.std, etc.)
    Returns:
        jackknife_replicate: The mean of all jackknife replicates
        standard_error: The standard error of the jackknife replicate
        jackknife_replicates: The list of statistics obtained from the
            jackknifed samples
    """
    jackknife_replicates = []
    jackknifes = np.size(data)
    for i in range(jackknifes):
        jackknife_data = [elem for j, elem in enumerate(data) if i != j]
        jackknife_replicate_i = stat_func(jackknife_data)
        jackknife_replicates.append(jackknife_replicate_i)
        return np.mean(jackknife_replicates), np.std(jackknife_replicates), jackknife_replicates
```

The jackknife approximation of the sample mean can be neatly formulated as:

$$\bar{x}_i = \frac{1}{n-1} \sum_{j=1, j \neq i}^{n} x_j, \qquad i = 1, \ldots, n.$$

The previous parts in which mean and standard deviation are approximated can be repeated again with the little change where `jackknife_resampling` method is called instead of `bootstrap_resampling`.

This code approximates the mean of the underlying distribution using jackknife resampling:

```
# Performing the jackknife sampling and computing the jackknife replicate
# of the mean togerther with its standard error
jackknife_mean, standard_error_mean, jackknife_means = jackknife_resampling(x_data, np.mean)
print('The sample mean computed from jackknife samples:', jackknife_mean)
print('The jackknifed estimate for the standard error of the mean:', standard_error_mean)

# Computing the 95% confidence interval
lower, upper = compute_confidence_interval(jackknife_means, 95)
confidence_interval = (lower, upper)
print('The 95% confidence interval of the mean:', confidence_interval)

print('Overall, mean of the dataset can be reported as: %.3f +- %.3f, (%.3f, %.3f)\n' %
      (jackknife_mean, standard_error_mean, lower, upper))
```

The output of this code and the histogram of the means of jackknife resamples with 5 bins are:

```
The sample mean computed from jackknife samples: 98.43666666666667
The jackknifed estimate for the standard error of the mean: 0.3467766563828598
The 95% confidence interval of the mean: (97.78103448275863, 99.15232758620692)
Overall, mean of the dataset can be reported as: 98.437 +− 0.347, (97.781, 99.152)
```

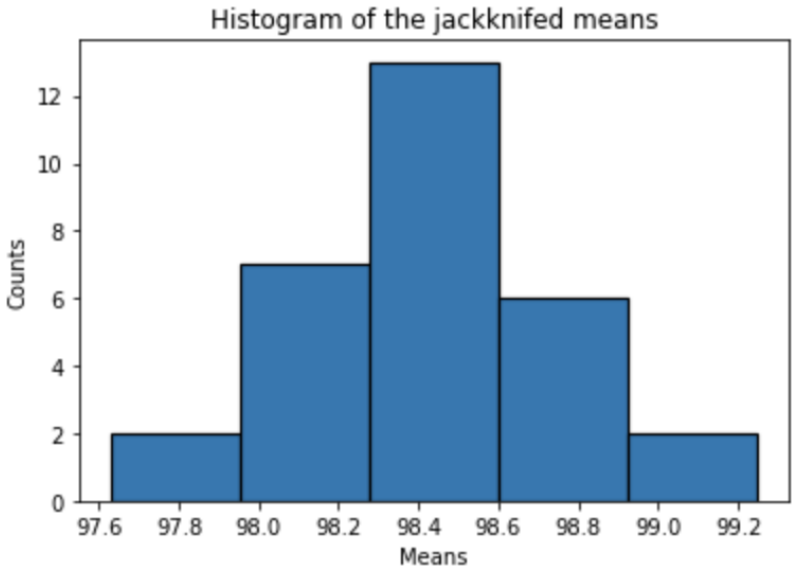

Figure 7: Histogram of the means of the jackknife resamples with 5 bins.

One can follow the same steps to approximate the standard deviation using jackknife resampling, the only change is to pass `np.std` as the function to compute the statistic of interest to the `jackknife_resampling` function:

```
# Performing the jackknife sampling and computing the jackknife replicate
# of the standard deviation togerther with its standard error
jackknife_std, standard_error_std, jackknife_stds = jackknife_resampling(x_data, np.std)
print('The sample standard deviation computed from jackknife samples:', jackknife_std)
print('The jackknifed estimate for the standard error of the standard deviation:', standard_error_std)

# Computing the 95% confidence interval
lower, upper = compute_confidence_interval(jackknife_stds, 95)
confidence_interval = (lower, upper)
print('The 95% confidence interval of the standard deviation:', confidence_interval)

print('Overall, standard deviation of the dataset can be reported as: %.3f +− %.3f, (%.3f, %.3f)\n' %
      (jackknife_std, standard_error_std, lower, upper))
```

The output of these lines and the histogram of the standard deviations of the jackknife resamples with 5 bins follow:

```
The sample standard deviation computed from jackknife samples: 10.046791101023341
The jackknifed estimate for the standard error of the standard deviation: 0.27457253785040076
The 95% confidence interval of the standard deviation: (9.225836041965325, 10.22730576358885)
Overall, standard deviation of the dataset can be reported as: 10.047 +− 0.275, (9.226, 10.227)
```

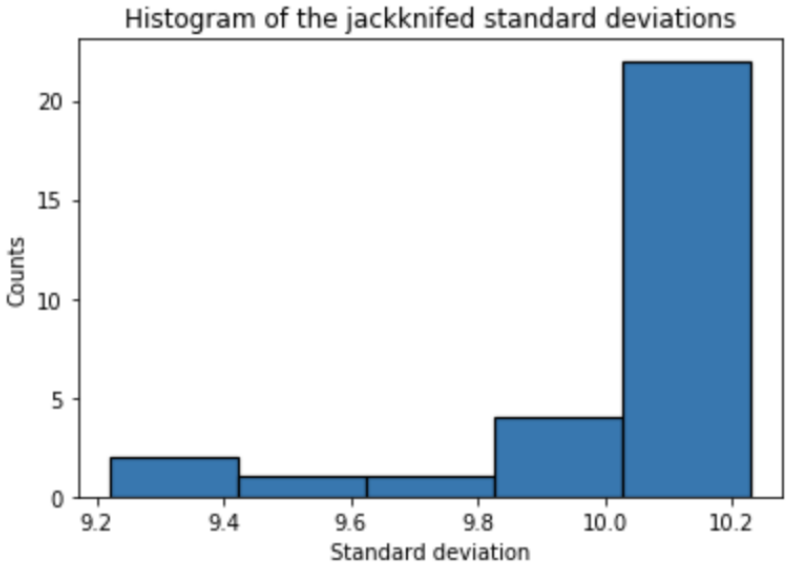

Figure 8: Histogram of the standard deviations of the jackknife resamples with 5 bins.

Question 3

Question 4

References

[1] "Understanding q-q plots — university of virginia library research data services + sciences," Data.library.virginia.edu, 2019, [Accessed: 18- Feb- 2019]. [Online]. Available: https://data.library.virginia.edu/understanding-q-q-plots/

[2] "Bootstrapping (statistics)," En.wikipedia.org, 2019, [Accessed: 18- Feb- 2019]. [Online]. Available: https://en.wikipedia.org/wiki/Bootstrapping_(statistics)

[3] "Jackknife resampling," En.wikipedia.org, 2019, [Accessed: 18- Feb- 2019]. [Online]. Available: https://en.wikipedia.org/wiki/Jackknife_resampling