Simulation Group Project

SYSC 4005 - Discrete Simulation / Modeling
Deliverable 2

Authors
Connor Judd, 101109256
Marko Majkic, 101109409
Millan Wang, 101114457

Due : 23h00 March 11th 2022

Input Data Modeling

To proceed with this project, a random number and variate generation process for the individual component inspection and workstation build times must be determined. This will be done by analyzing the input data and then creating a distribution model for generation of new data points. This will be done with the following steps:

1. Generate a histogram for the simulation entity from the associated dat file
2. Compare the histogram to default distribution graphs.
   See which is the most similar for next step choice
3. Choose a distribution to test if the input data fits to the default graphs.
   QQ and/or Chi-squared testing
   If the tests conclude that the input data does not match the
   hypothesized distribution, try a different distribution
4. Select a method for random variate generation based on the distribution

The above process will be repeated for all 6 .dat files representing the different service times for the component inspection and workstation build processes.

Stat Manager

The **`Stat_Manager`** class provides functionality to select the data files to then make comparisons to different distributions. The method reads data from the selected files, makes calculations, and sends the required information to method calls in the **`Grapher`** class. When a **`Stat_Manager`** object is instantiated, it calls the **`compute_stats`** method and will save the data for the selected data set. The **`compute_stats`** method takes in an input to select which data file is going to be analyzed. The data file is then located, read line by line, and then converted into a Python list of floats. The input data has a precision of 3 decimals, and to prevent issues with float numbers, all values in the list are converted to integers by multiplying by 1000 to represent milliseconds instead of seconds. Using Python's built in list operations, the sum, count, sample mean, and sample variance are determined, and a sorted copy of the list is also stored.

The next set of methods are used for doing operations on the data for visualization and hypothesis testing. The first is **generate_histogram**, which interfaces with the grapher class to generate histograms. The next method is **qq_plot** which also interfaces with the grapher for QQ Plot generation. The next methods are in the **chi_squared** family and conduct chi squared tests on the sample data to test if the sample data is consistent with a hypothesized distribution. The subsequent methods are used as helper methods to the above procedures to generate quantiles and cdf values. The above functionality inside SMSM will be used in the Appendix where the distribution fitting will happen for the different simulation entities.

Graphing Process

The **Grapher** class serves as a helper for the **stat_manager** class. The **Grapher** class provides the ability to create a multitude of graphs by providing the functions with the proper parameters. There are 7 different graphs that can be graphed using the grapher: Histogram, Q-Q plot, Poisson Distribution Plot, Normal Distribution Plot, Exponential Distribution Plot, Weibull Distribution Plot, and a Log Normal Distribution Plot.
The functions and their specifications are as follow:

The **build_histogram** function allows the user to provide an array of data points, and a title. It will then calculate the appropriate number of bins using the data and the Freedman–Diaconis formula, create the histogram using Matplotlib's histogram function, then save it to a file with the title and a date/time stamp.

The **build_qq_plot** function allows the user to provide two lists of data points and a title. It will then create the q-q plot by plotting the lists on the x and y-axis respectively, then save it to a file with the title and a date/time stamp.

The **build_poisson** function allows the user to provide an array of lambdas (1 or more) and a title. It will then use Scipy's poisson library to create the poisson PMF and plot the function using Matplotlib. It will do this for each lambda in the array and plot them on the same figure, then save it to a file with the title and a date/time stamp.

The **build_normal** function allows the user to provide a mean, a standard deviation, and a title. It will then calculate the distribution's range, and using Scipy's Norm library, calculate the PDF and plot the function using Matplotlib, then save it to a file with the title and a date/time stamp.

The **build_exponential** function allows the user to provide an array of betas (1 or more), and a title. It will then use the Seaborn library's kdeplot function to plot an exponential relationship using the provided beta. It will do this for each beta in the array and plot them on the same figure, then save it to a file with the title and a date/time stamp.

The **build_weibull_constant_beta** function allows the user to provide an array of alphas (1 or more), and a title. It will then use Reliability statistics library's **Weibull_distribtuion** function to create a distribution and a corresponding PDF and plot it using Matplotlib. It will do this for each alpha in the array and plot them on the same figure, then save it to a file with the title and a date/time stamp.

The build **build_weibull_double_param** function, similarly to the **build_weibull_constant_beta** will create a Weibull plot in the same way, however, -**build_weibull_double_param** allows the user to provide an alpha and beta value, and it only can create one plot.

The **build_log_normal** function allows the user to provide a mean, standard deviation, and title. It will then create a log normal distribution using Numpy, and plot the histogram, with the overlaid PDF, then save it to a file with the title and a date/time stamp.

Input Data Modeling - Distribution Fit Hypothesis Testing

To test if a collection of input data fits a hypothesized distribution, QQ tests and Chi-Squared tests will be conducted. For the simulation entities in this project, the distribution fit hypothesis testing will occur in the hypothesis.

QQ Testing

A Q-Q or quantile-quantile plot is a graphical method of comparing two probability distributions by plotting their quantiles against each other. It provides a method to evaluate how well a distribution represents a set of data. In the code we currently have implemented QQ testing for Exponential and Weibull distributions because of the initial observations of the histograms. For a Weibull distribution the following equation is used to find the inverse.

$$Q(p; k, \lambda) = \lambda(-\ln(1-p))^{1/k}$$

for $0 \leq p < 1$.

In this function p is varied from 0 to 0.99 and uses constants for k and lambda to control the fit. For the exponential distribution the following equation.

$$F^{-1}(p; \lambda) = \frac{-\ln(1-p)}{\lambda}, \qquad 0 \leq p < 1$$

In this function p is varied from 0 to 0.99 using a constant value for lambda to control the fit. By changing the constants in the equation, the best possible fit for the distributions can be found. The fit of the distribution is evaluated by how close data points are to a straight line on the graph. The closer that this fit is to a straight line, the better the fit of the distribution to the data, and if through visual inspection we determine that the line on the QQ plot is sufficiently straight, then we will proceed to test the goodness of fit with Chi-Squared testing.

Chi Squared Testing

A Chi-Squared test presents a null hypothesis stating that the distribution of a set of sample data points is the same as a specified hypothesized distribution, and the alternate hypothesis states that they are different. Testing if the input data fits a distribution using a chi squared test is done by first sorting the input data from smallest to largest and then determining how many bins of what size to use to categorize the sample data. Once the bin count and sizes are known, then the next step is to tabulate the observed frequencies in each bin in the same way that it is done for making histograms. Then for each bin, an expected frequency number will need to be determined. This expected frequency number will be calculated by determining the difference between the cdf's of the upper and lower bounds of the current bin, and multiplying the difference by the size of the dataset. Then for each bin the result of…

((observedFrequency-expectedFrequency)^2)/expectedFrequency

…will be calculated and then summed into a final variable called Chi-Squared. This Chi-Squared value will be compared to a reference Chi-Squared value from a table whose value will be dependent on the number of bins, and the alpha level of significance. If the table value is smaller than the calculated value, then the null hypothesis is is rejected and the distribution is not the same as the sample data, and if the table value is larger than the calculated value, the null hypothesis will not be rejected and we can proceed with the assumption that the selected distribution is the same as the input data.

Random Number Generation Process

To generate randomized variates to use for the simulation's component inspection times and workstation build times, a random number generation system was manually implemented and tested. After determining what distribution best models an input dataset, random numbers will be generated following the below steps

1. Seeded pseudo random generation with multiplicative congruential model
2. Use 2 multiplicative congruential models with L'Ecuyer's algorithm to get a better pseudo random number with a much longer generator period
3. Use the L'Ecuyer's generated number to generate a value along a specified distribution with the inverse-transformation or acceptance-rejection technique

Multiplicative Congruential Model

Random number generation with the multiplicative congruential model is done with a recursive method as shown below.

$$X_{current} = (aX_{previous}) \% m$$

Where X represents a randomly selected number, a is the multiplier parameter, m is the modulus parameter, and % is the modulus operator. Since the current random number to generate depends on the previous generated number, the user must define an initial seed value to set as the first Xprevious so that subsequent numbers can be generated. Once a number is generated, the value for Xprevious will be overwritten by the current Xcurrent value.

L'Ecuyer's Algorithm

To increase the period of the random number generation to be sufficiently large to avoid a period cycle, two multiplicative congruential model instances will be combined to make a generator with a period of approximately 2x10^18. This will be done using a slightly modified version of L'Ecuyer's algorithm as shown below.

$SubGenerator\ 1: m = 2147483563, a = 40014, seed = first\ generated\ int\ from\ SubGenerator2$
$SubGenerator\ 2: m = 2147483399, a = 40692, seed = user\ defined\ in\ range[1, 2147483398]$

$$L'Ecuyer's\ Generator: X_{Current} = (X_{SubGen1,\ current} - X_{SubGen2,\ current}) \% 2147483562$$

Xcurrent represents the current random integer generated from the L'Ecuyer process and the sub-generator specific Xcurrent values will come from their individual multiplicative congruential model generators. To convert the random integer to a decimal number in the range [0,1], denoted as R, the below translation will occur.

$$R_{current} = if\ (X_{current} > 0)\ \{\frac{X_{Current}}{2147483563}\}\ else\ \{\frac{2147483562}{2147483563}\}$$

Random Number Generation Testing

In order to ensure that the randomly generated numbers are truly random, we shall test for uniformity and independence with 95% confidence. Uniformity will be tested using a Kolmogorov-Smirnov test and independence will be tested using an Autocorrelation test

Kolmogorov-Smirnov Testing

To determine if the random number generation process produces uniform values, a Kolmogorov-Smirnov test will be conducted comparing a set of randomly generated numbers to a uniform distribution. The null hypothesis of this test states that there is no difference between the distribution of the random numbers and a uniform distribution, and the alternate hypothesis states that they are different.

This test will compare the continuous cumulative distribution function of a uniform distribution to the empirical cumulative distribution function observed from a set of random samples.

The first step is to generate a set of random numbers. 100 random numbers will be generated for the testing in this project, and this set of random numbers shall be sorted from smallest to largest.

Another set of 100 numbers will be created to represent the expected cumulative distribution function values, also ordered from smallest to largest. Given that there are 100 random samples to match, this list will also have 100 values, starting at 0.01 with each element being 0.01 larger than the previous as shown below

```
[0.01, 0.02, 0.03, . . . 0.98, 0.99, 1.0]
```

Given these sorted lists of random numbers and expected cumulative distributions, two more lists will be made taking in corresponding values from the two lists. The first list will be called the D_plus list and each element of the list will be the calculation result of the (currentExpectedValue) - (currentSortedRandomNumber). For example the first element of the D_plus list will be the result of (0.01)-(smallestRandomNumber)

The second list will be called D_minus and each element of the list will be the calculation result of (currentRandomNumber)-(previousExpectedValue). The first element of D-minus will use previousExpectedValue=0, and the second element in D_minus will be (2ndSmallestRandomNumber)-(0.01=previousExpectedValue)

Once the D_plus and D_minus lists are made, the largest contained value between both of them will be retrieved and will be referred to as D_max. This D_max value will then be compared to the table based D_alpha value, and using alpha=0.05, D_alpha=1.36/sqrt(100)=0.136

If the D_max value is less than the D_alpha table value, then the null hypothesis will not be rejected and it can be assumed that the generated numbers are uniformly distributed. Otherwise, the null hypothesis will be rejected and it can be concluded that the randomly generated values are not uniformly distributed. This test is done in the code in **`Lecuyer_Generator.py`** in the **`run_kolmogorov_smirnov_test()`** function and after multiple repeated automated tests, the null hypothesis was not rejected.

Autocorrelation Testing

To determine if the random number generation process produces independent numbers, an autocorrelation test will be conducted. The null hypothesis of this test states that the numbers are independent, and the alternate hypothesis is that they are not.

The test assesses if subsequently selected random numbers tend to follow a pattern, i.e if low numbers tend to be followed by higher numbers, or if high numbers tend to be followed by high numbers, etc.. This first involves setting & calculating the below variables, and the values used for testing this project are shown below

$N = 1000 = Total\ number\ of\ random\ numbers\ to\ generate$
$\alpha = 0.05 = confidence.\ This\ value\ halved\ will\ get\ ZTable\ value\ to\ beat$
$i = 1 = starting\ index\ in\ the\ set\ of\ random\ numbers$
$m = 1 = lag,\ how\ many\ numbers\ to\ skip\ between\ each\ choice$
$M = 998 = Largest\ integer\ that\ makes\ i + (M + 1)m \leq N\ true$

Given the above variables calculate the following values

$$\rho = \frac{1}{M+1}\left(\sum_{k=0}^{M} R_{i+km}R_{1+(k+1)m}\right) - 0.25$$

$$\sigma = \frac{\sqrt{13M+7}}{12(M+1)}$$

$$\frac{\rho}{\sigma} \leq Z_{table} = 1.96$$

If the above condition is false, then the null hypothesis is rejected and we conclude that the numbers are dependent. If the above condition is true, then we cannot reject the null hypothesis, and can proceed with the assumption that the numbers are independent with 95% confidence. This test is done in the code in **`Lecuyer_Generator.py`** in the **`run_autocorrelation_test()`** function and after many repeated automated tests, the null hypothesis was not rejected

Random Variate Generation

Using the above uniform random number generator, random variates along a specified distribution will be generated with the following methods

Inverse-Transform Technique

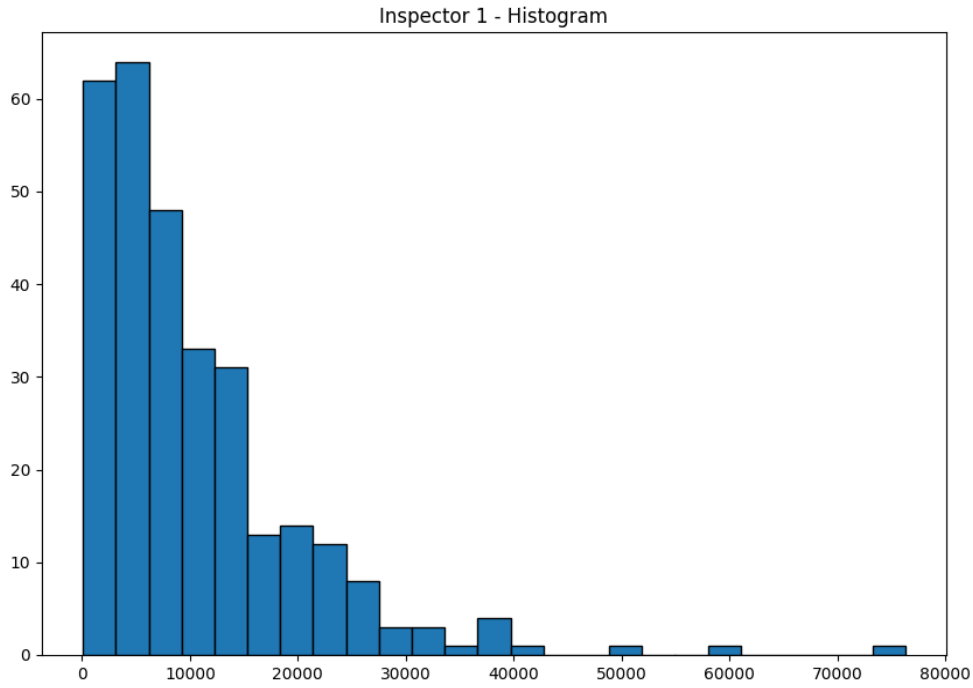
For generating random variates, the preferred method is the inverse-transform technique. The inverse-transform technique requires an inversible cumulative distribution function for the distribution to generate from. Once the inverse of a distribution's cumulative distribution function is determined, then a randomly selected number from a uniform distribution will become the input parameter, and the output of the calculation will be a random variate. This technique will be used for uniform, exponential, Weibull, and triangular distributions.

Acceptance-Rejection Technique

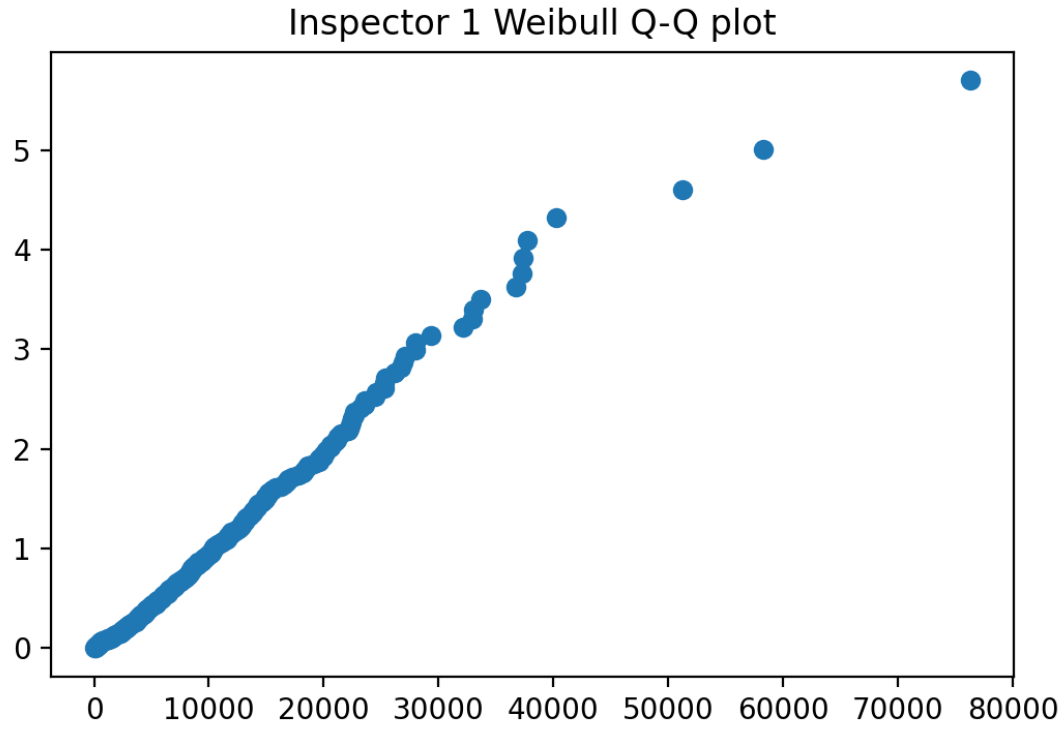
When the inverse of a distribution's cumulative distribution function is impossible or prohibitively difficult to determine, the Acceptance-Rejection technique for generating random variates will be done instead of the Inverse-Transform technique.

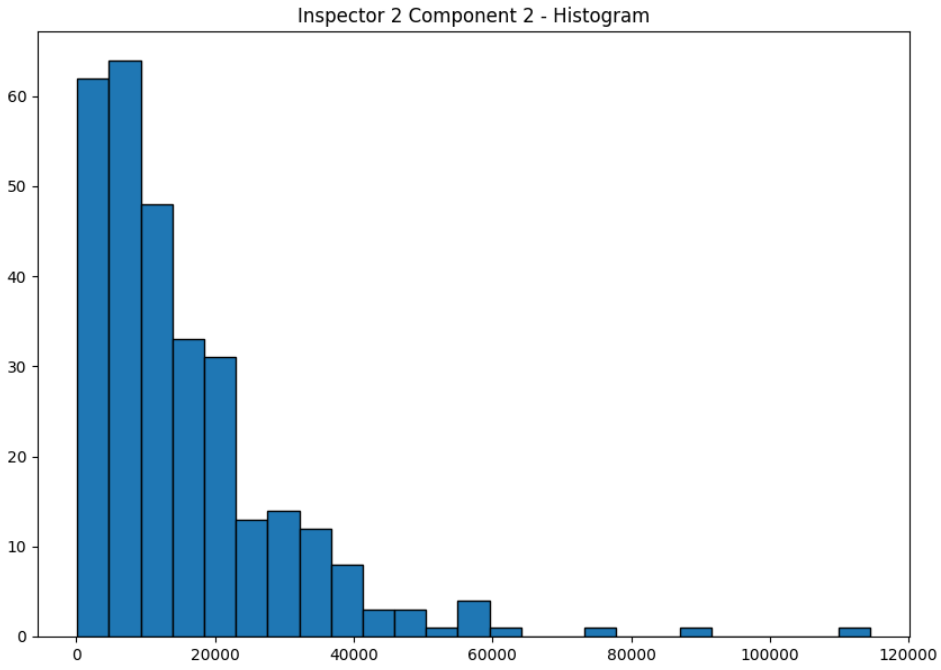
Before generating a random variate with the acceptance-rejection technique, one must first determine a condition in which random variates will be accepted. Every time a number is generated it will be checked against this condition and if the condition is false, the number will be discarded and re-generated. If a generated number makes the condition true, then it will be forwarded to be used in the simulation

Appendix A - Analysis of Sample Data for Simulation Entities

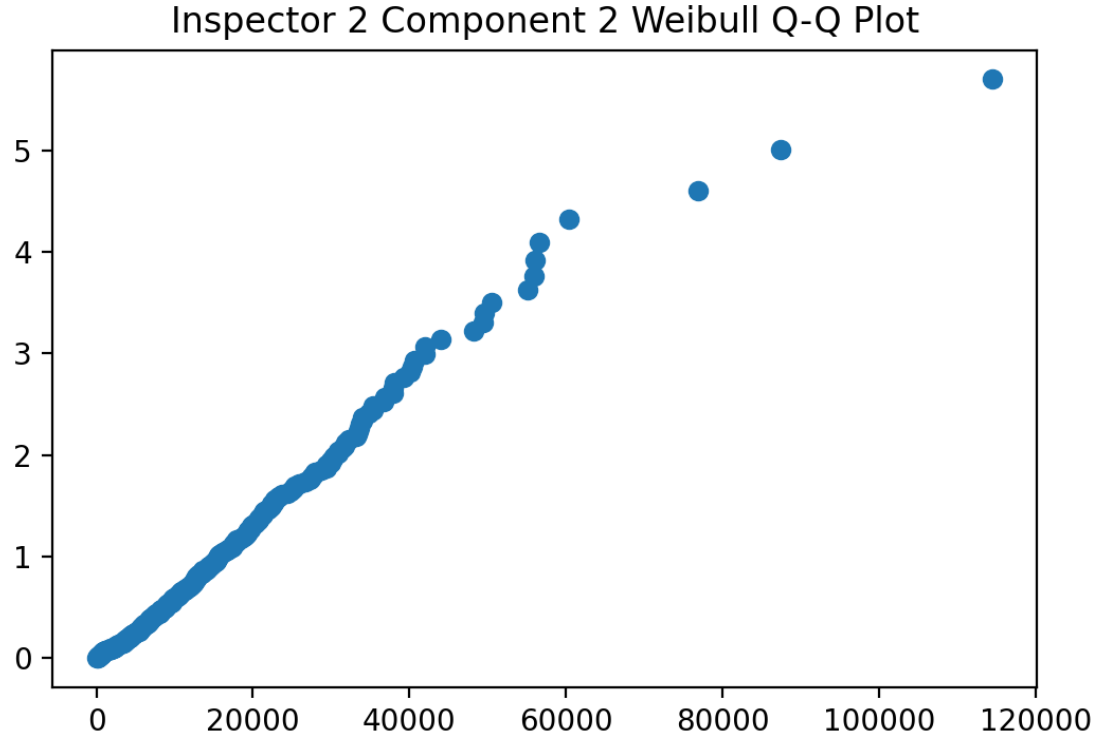


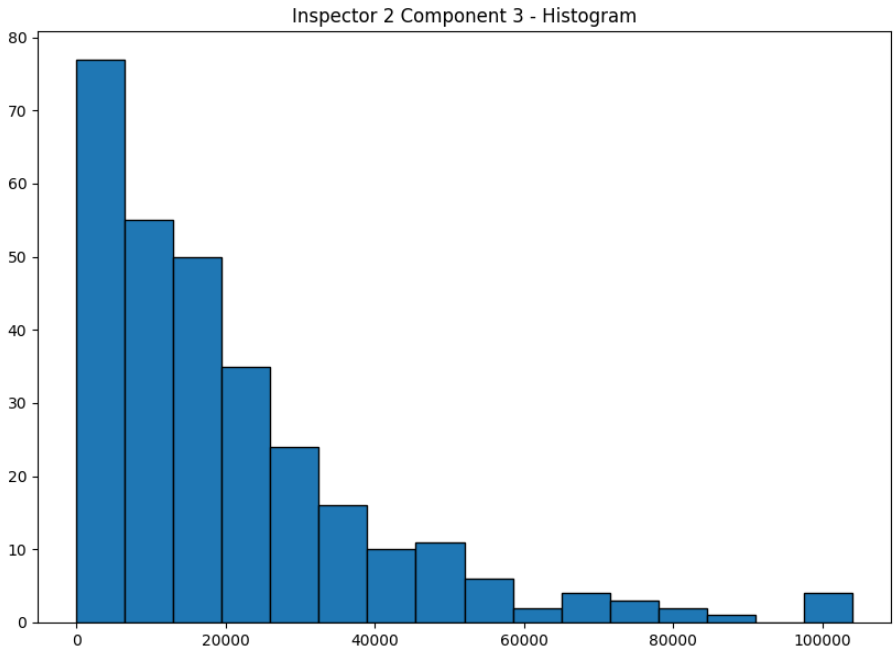
The distribution displayed above seems to follow a Weibull distribution with the first bin being smaller than the second, and then a downward trend following. Using a Weibull distribution, QQ and Chi-squared testing will be conducted to test the fit. If the Weibull distribution is unsuccessful a different distribution will be attempted. Then the inverse transform method of random variate generation will be conducted on the distribution. Below is a QQ plot for this sample data set. This QQ plot is a straight line showing that the correct distribution was selected.



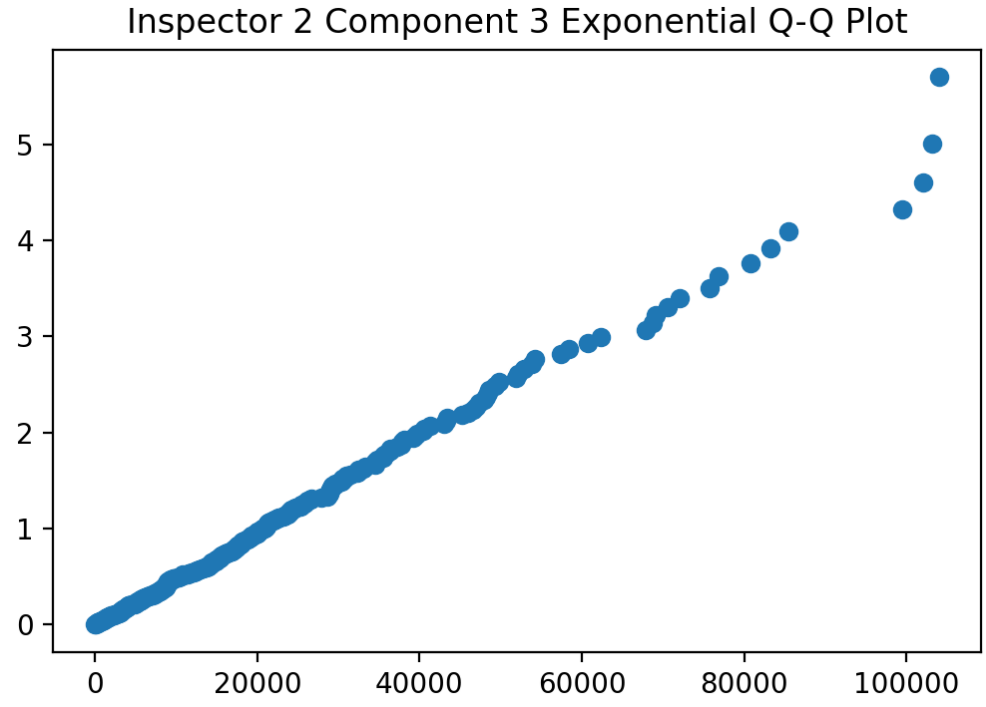


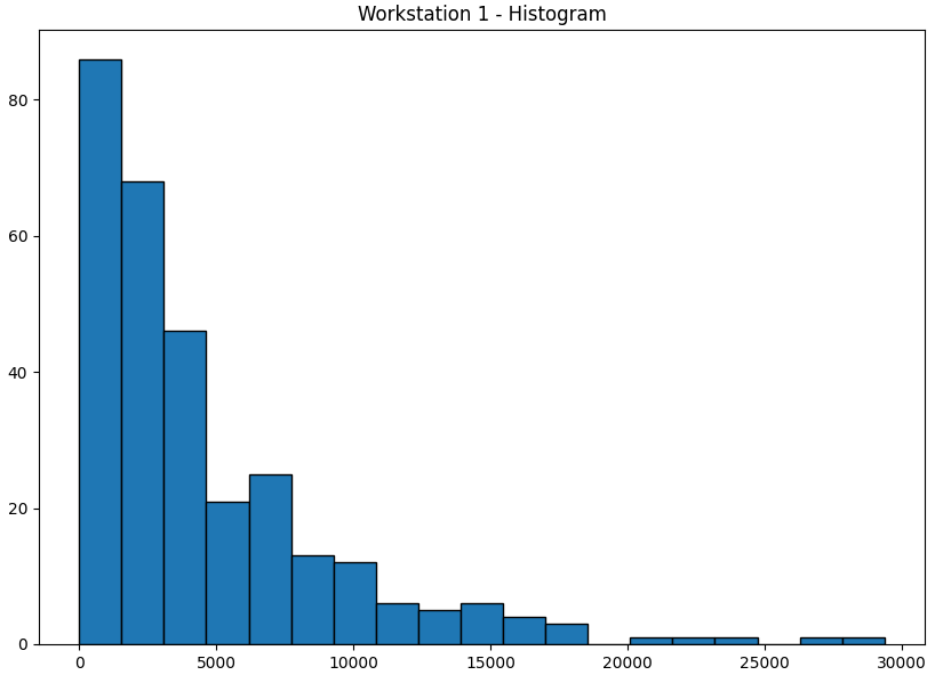
The distribution displayed above seems to follow a Weibull distribution with the first bin being smaller than the second, and then a downward trend following. Using a Weibull distribution, QQ and Chi-squared testing will be conducted to test the fit. If the Weibull distribution is unsuccessful a different distribution will be attempted. Then the inverse transform method of random variate generation will be conducted on the distribution. Below is a QQ plot of the sample data compared to a Weibull distribution. This QQ plot is a straight line showing that the correct distribution was selected.



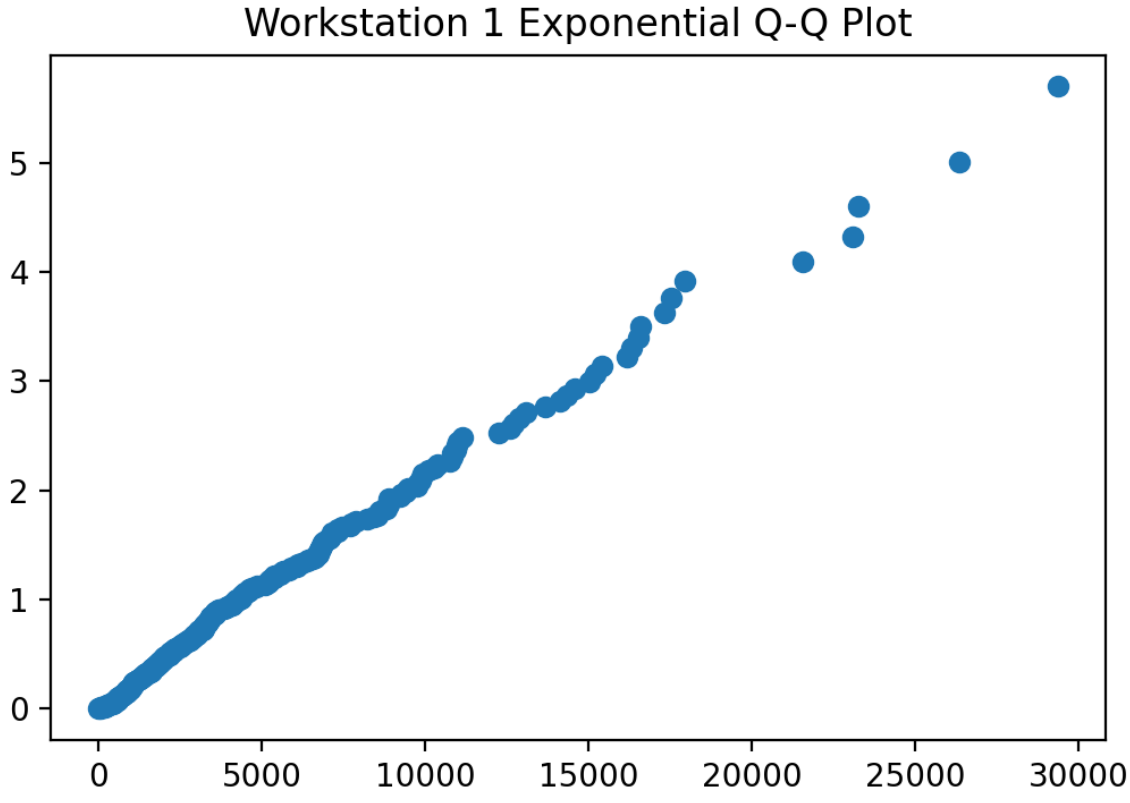


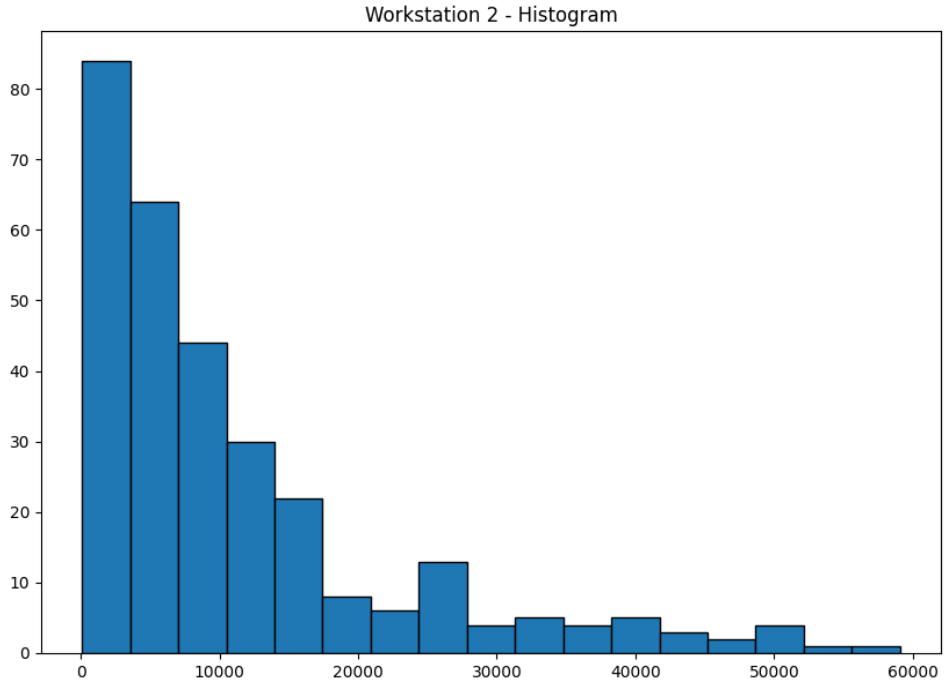
The distribution displayed above seems to follow an exponential distribution with a continuous downwards trend. Using an exponential distribution, QQ and Chi-squared testing will be conducted to test the fit. If the exponential distribution is unsuccessful a different distribution will be attempted. Then the inverse transform method of random variate generation will be conducted on the distribution. Below is a QQ plot of the sample data compared to an exponential distribution. This QQ plot is a straight line with a slight tail showing that the correct distribution was selected.



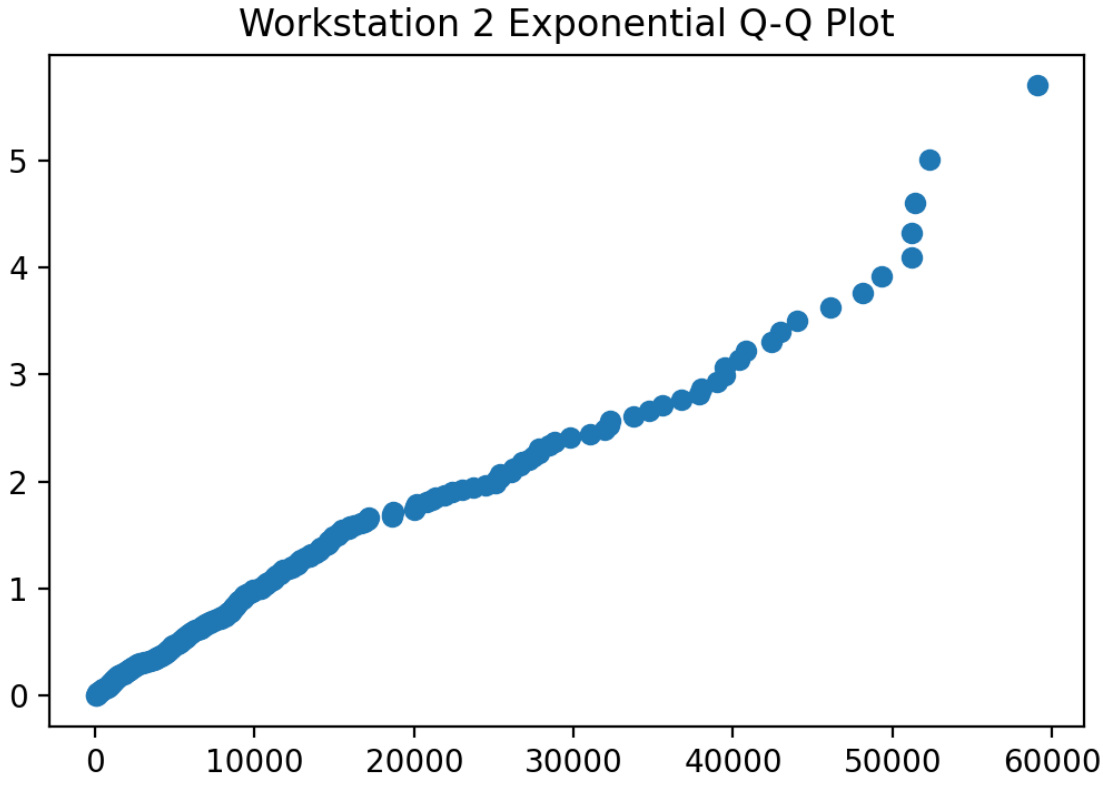


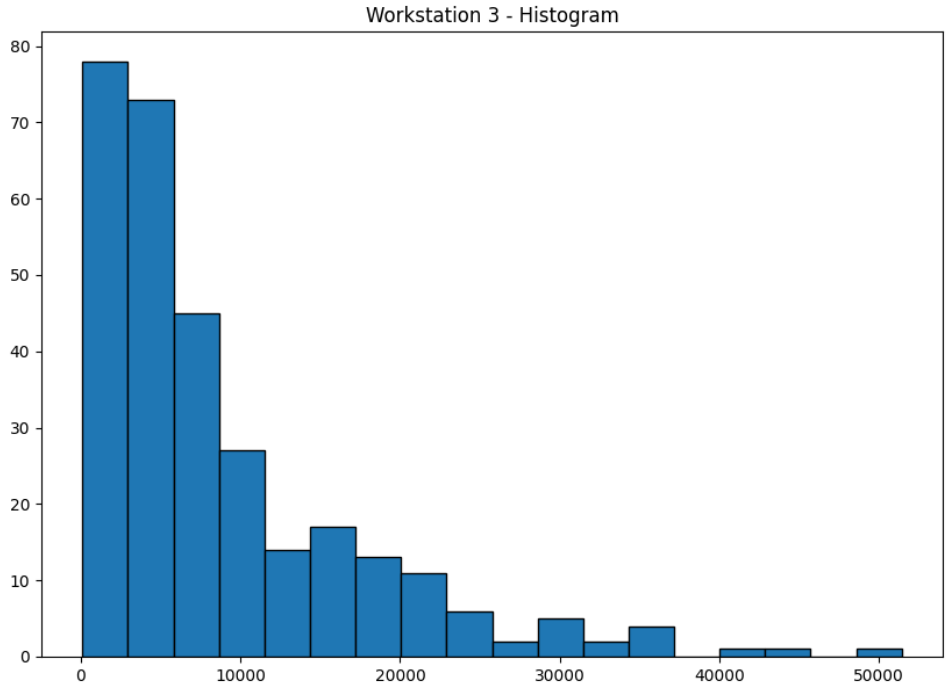
The distribution displayed above seems to follow an exponential distribution with a continuous downwards trend. Using an exponential distribution, QQ and Chi-squared testing will be conducted to test the fit. If the exponential distribution is unsuccessful a different distribution will be attempted. Then the inverse transform method of random variate generation will be conducted on the distribution. Below is a QQ plot of the sample data compared to an exponential distribution. This QQ plot is a straight line showing that the correct distribution was selected.



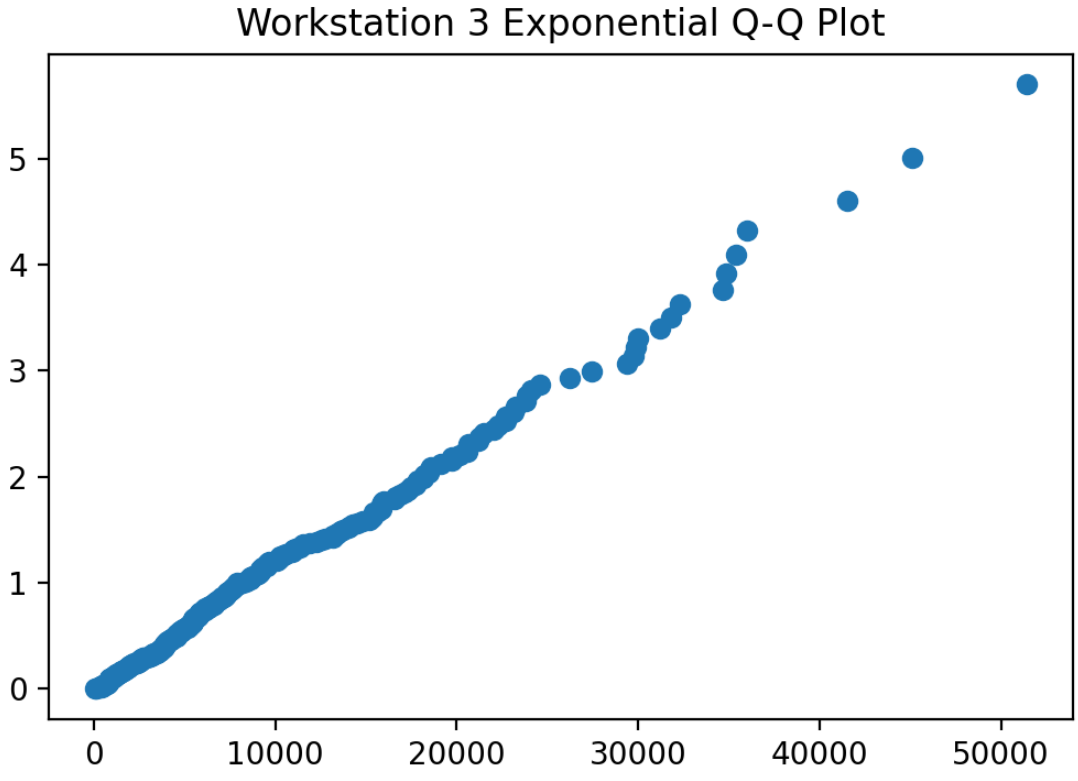


The distribution displayed above seems to follow an exponential distribution with a continuous downwards trend. Using an exponential distribution, QQ and Chi-squared testing will be conducted to test the fit. If the exponential distribution is unsuccessful a different distribution will be attempted. Then the inverse transform method of random variate generation will be conducted on the distribution. Below is a QQ plot of the sample data compared to an exponential distribution. This QQ plot is a straight line with a slight tail showing that the correct distribution was selected.

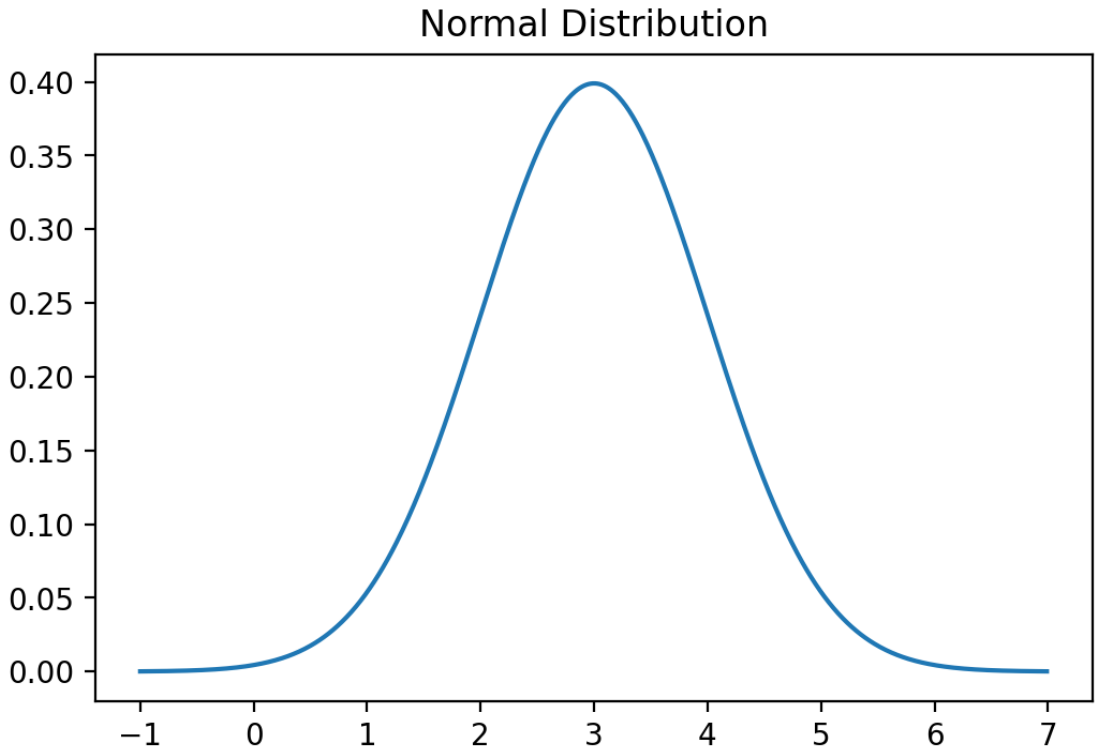


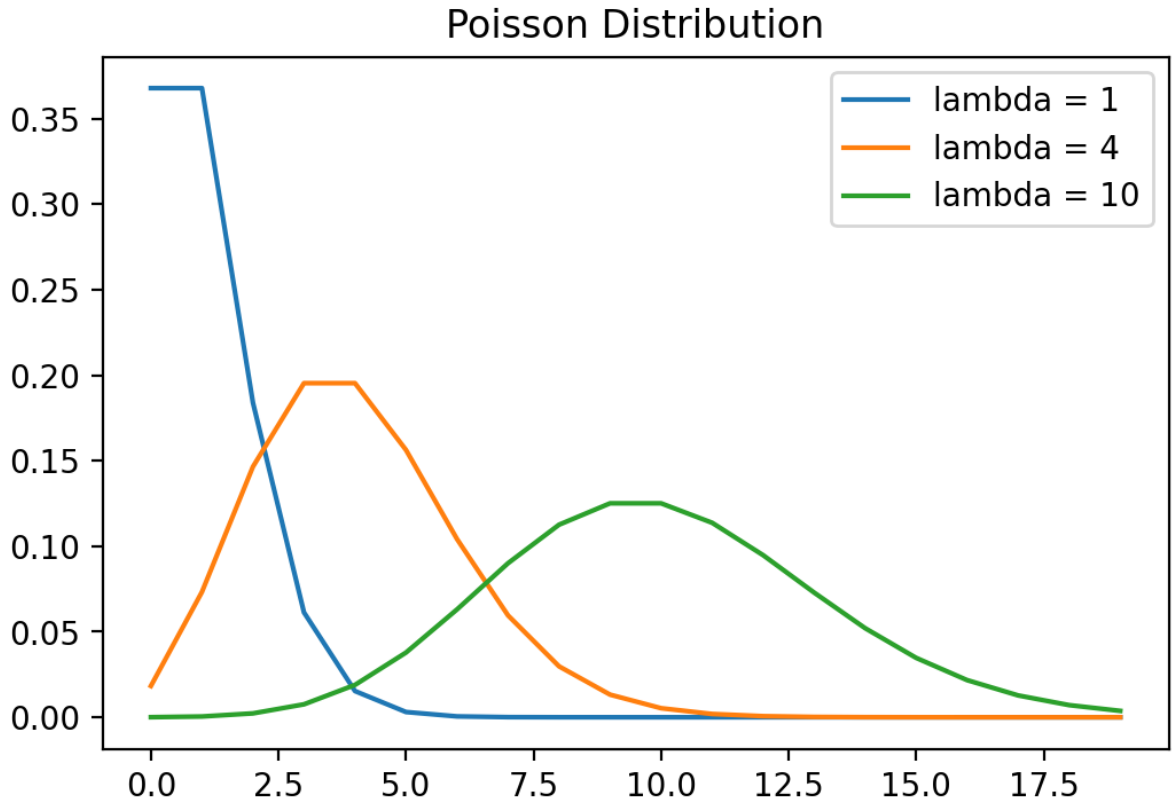


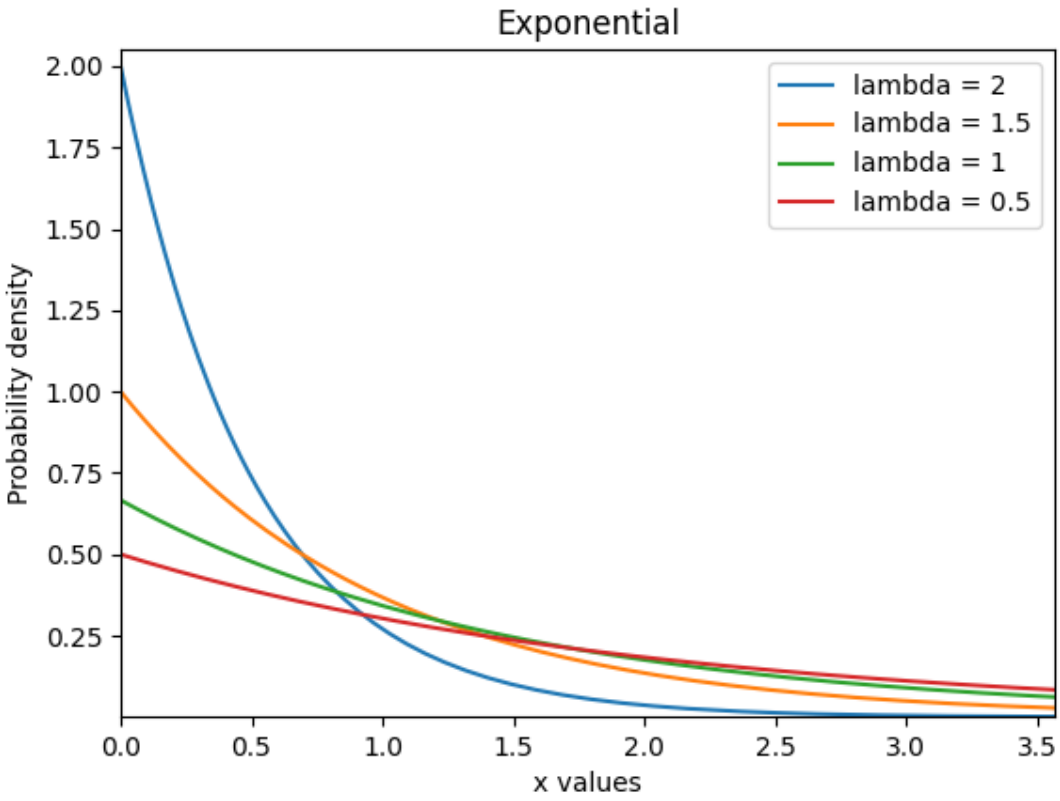
The distribution displayed above seems to follow an exponential distribution with a continuous downwards trend. Using an exponential distribution, QQ and Chi-squared testing will be conducted to test the fit. If the exponential distribution is unsuccessful a different distribution will be attempted. Then the inverse transform method of random variate generation will be conducted on the distribution. Below is a QQ plot of the sample data compared to an exponential distribution. This QQ plot is a straight line showing that the correct distribution was selected.

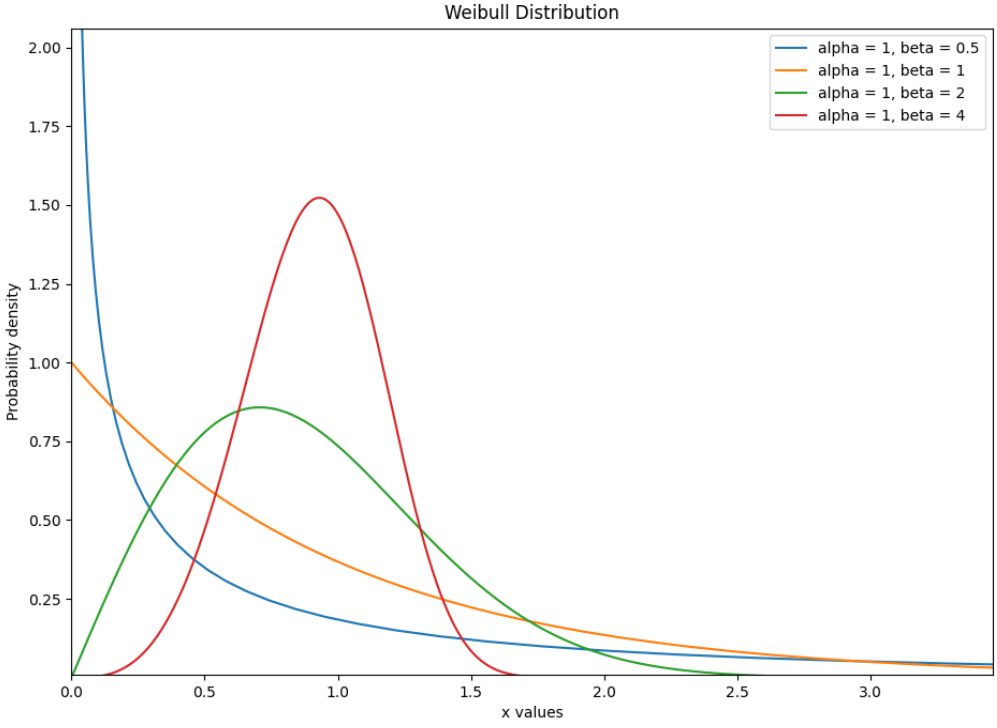


Appendix B - Shapes of Distributions for Reference






Exponential
lambda = 2
lambda = 1.5
lambda = 1
lambda = 0.5
Probability density
x values
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5


Weibull Distribution
alpha = 1, beta = 0.5
alpha = 1, beta = 1
alpha = 1, beta = 2
alpha = 1, beta = 4
Probability density
x values
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.0
0.5
1.0
1.5
2.0
2.5
3.0