Milestone 2 Report

Ghassan Arnouk
101078550

Kareem El-Assad
101107739

Evan Lloyd
101108670

SYSC 4005A
Winter 2023
Group 49

Instructor: Prof. Changcheng Huang

Day Submitted: March 19, 2023

Contents

List of Tables

List of Figures

List of Listings

1 Data Collection & Input Modelling

1.1 Distributions

The histograms of the provided data are shown in Fig. 1.

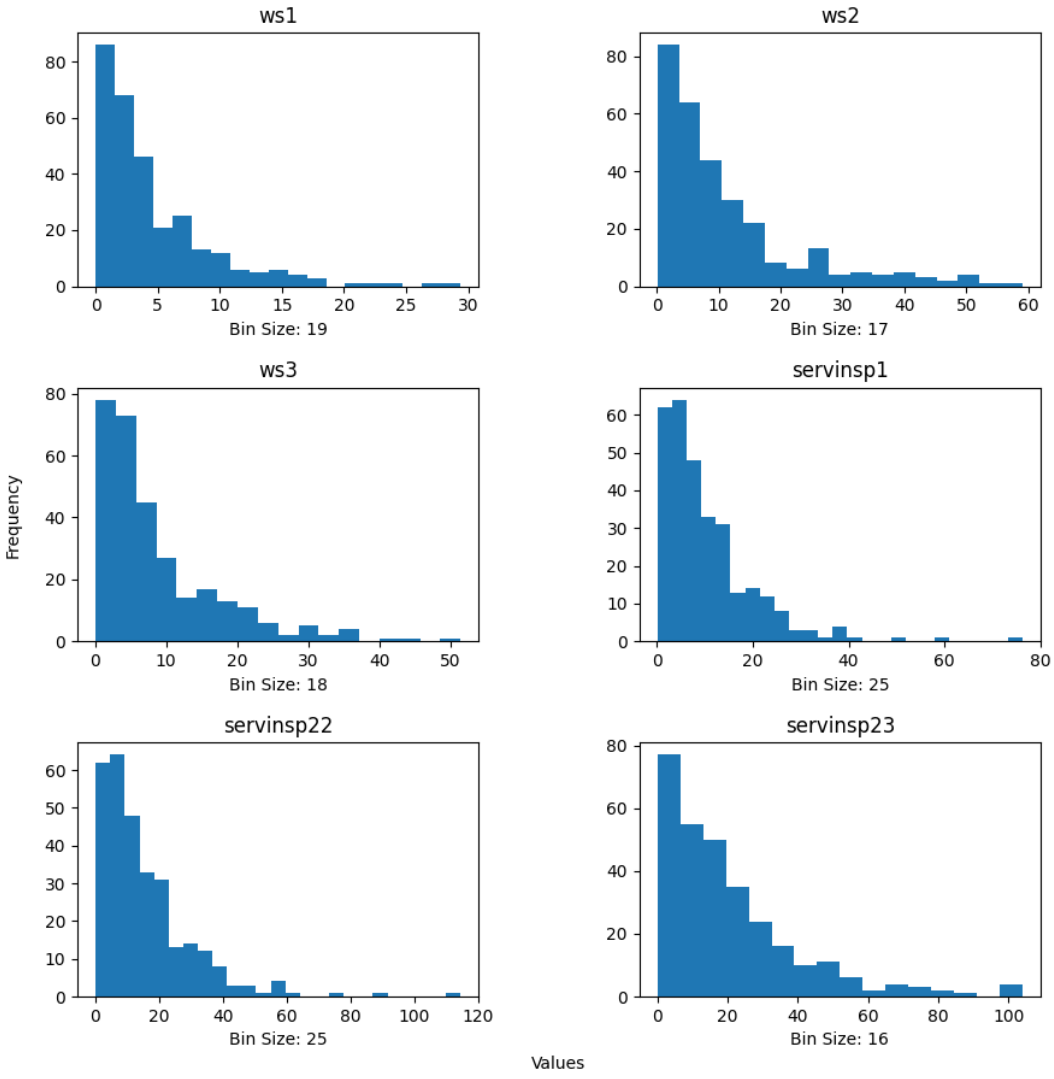

Figure 1: Histograms of the provided data

1.1.1 Bin Size

As shown in listing 2, the bin size is calculated using the Freedman-Diaconis rule with the following equation [1]:

$$W = \frac{2IQR}{n^{\frac{1}{3}}} \tag{1}$$

The IQR is calculated by subtracting the First Quartile (Q1) value from the Third Quartile (Q3) value using the following equation:

$$IQR = Q3 - Q1 \tag{2}$$

The bin size used on the provided data, to generate the histograms, is as follows:

- $ws_1 \rightarrow 19$
- $ws_2 \rightarrow 17$
- $ws_3 \rightarrow 18$
- $servinsp_1 \rightarrow 25$
- $servinsp_2 \rightarrow 25$
- $servinsp_3 \rightarrow 16$

1.1.2 Choice of Distribution

Using the plots shown in Figure 1, the histograms for the 6 data sets are identified as follows:

- $ws_1 \rightarrow$ Exponential
- $ws_2 \rightarrow$ Exponential
- $ws_3 \rightarrow$ Exponential
- $servinsp_1 \rightarrow$ Exponential
- $servinsp_2 \rightarrow$ Exponential
- $servinsp_3 \rightarrow$ Exponential

1.1.3 Discussion

Choosing a proper bin size is an essential key to creating histograms. The bin size also plays a big role in visualizing and interpreting the data. A good bin size for a histogram requires a balance between accurate representation of the data as well as clear and informative visual display.

The ideal bin size varies depending on several factors such as: size of the data set, range of the data, distribution of the data, and lastly, the purpose of the histogram.

For the size of the data set, a larger data set have more variation and may require a smaller bin size to capture the detail and structure of the data. Considering the range of the data, if the range is very large, a larger bin size may be required to avoid oversimplification of the data. In addition, if the goal of the histogram is to compare trends or how best the data fit a model, then a smaller bin size may be required to detect small differences.

Using the shapes of the generated histograms, shown in Fig. 1, it is observed that ws_1, ws_2, and ws_3 follow an exponential distribution. The exponential distribution is often used in reliability analysis, where it used to model the time until a failure occurs. Generally, it is used to model waiting times. For instance, modelling the time between generating different products (P1, P2, P3) assuming the events of generating those products occur randomly and independently at a constant rate.

Similarly, $servinsp_1$, $servinsp_2$, and $servinsp_3$ also follow an exponential distribution, as shown in Fig. 1. The main advantage of using the exponential distribution for the service time of inspectors is that the probability of the inspection occurring and passing the inspection is independent of how much time has gone by since the last occurrence of the inspection.

1.2 Q-Q Plotting

1.2.1 Histogram is not enough

An important factor that is a limitation to histograms is the sensitivity to the choice of the bin size. If the bin size is too large, important details about the histogram may be lost due to oversimplification of the data. On the other hand, if the bind size is too small, the histogram can become more complex and difficult to comprehend.

1.2.2 Theory Behind Q-Q Plotting

The objective of Quantile-Quantile (Q-Q) plot is to determine if two data sets come from the same distribution [2]. Furthermore, constructing a Q-Q plot is done by plotting the first data set's quantiles against the second data set's quantiles. When observing a generated Q-Q plot, it is seen that the points match up a straight line indicating that the quantifies match [2]. While the straight line plotted is not an essential component of a Q-Q plot, it allows a visual representation of where the points should line up [2].

1.2.3 Q-Q Plots

Sample Calculations

Note that all the calculations were completed using listing 3. However, the steps below outlined to demonstrate the process and the calculations performed.

Step 1: Ordering the data set from smallest to largest

Step 2: Drawing a normal distribution curve

Step 3: Finding the z-value (cuttoff point) for each segment

Step 4: Plotting the data set values (from step 1) against the normal distribution cuttoff points (from step 3)

The Q-Q plots of the provided data are shown in Fig. 2.

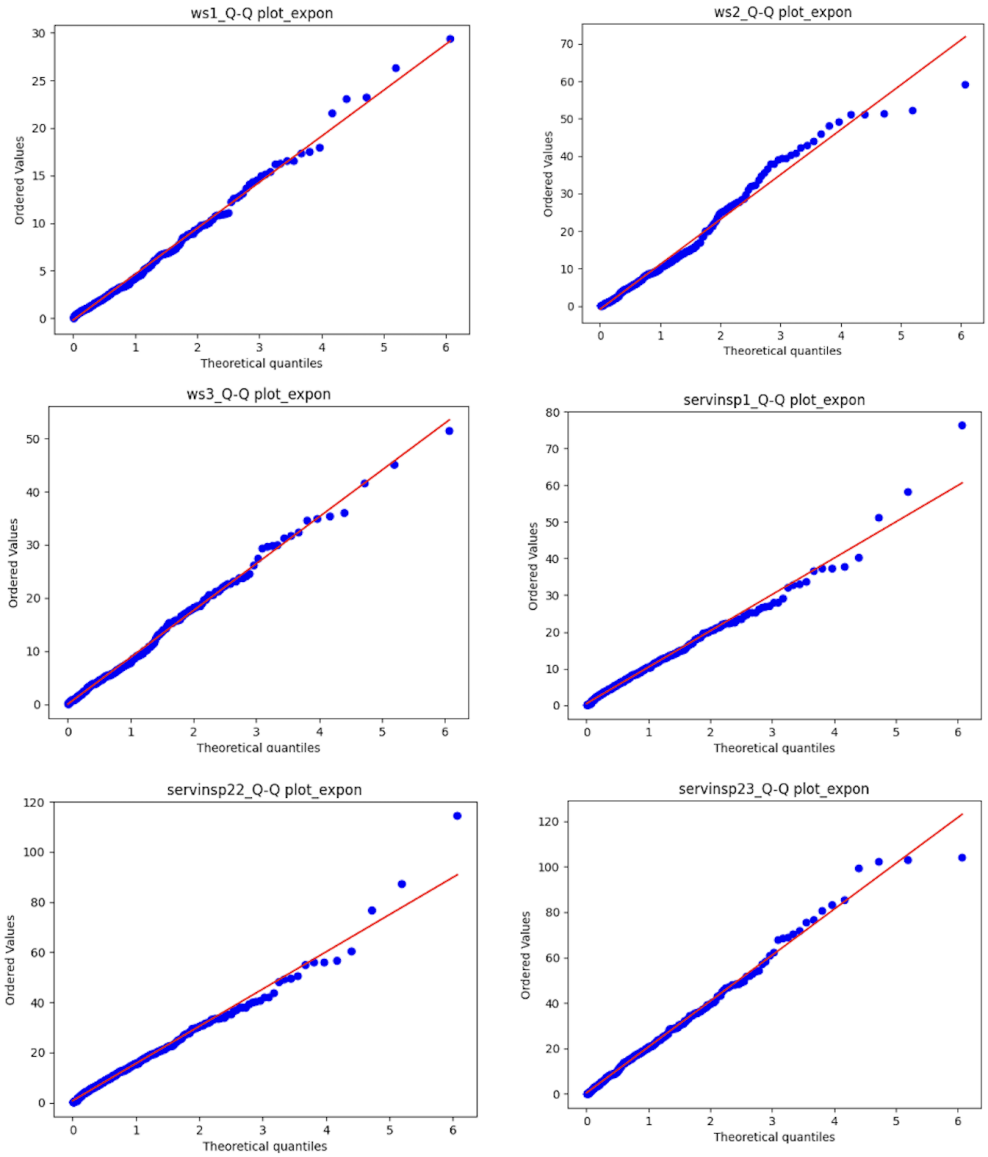

Figure 2: Q-Q plots of the provided data

1.2.4 Non-linearity at Certain Points

The non-linear points of on the Q-Q plot indicates the differences in the shape of the distributions (how not similar the distributions are). However, as shown in Fig. 2, the majority of the individual Q-Q plots are linear and therefore, the non-linearity is correlated to the differences in the spread of the data as well as the presence of outliers in the data sets.

1.2.5 Conclusion

A bin size for the histogram was chosen and calculated using the Freedman-Diaconis rule due its relevance and easy of use. Appropriate bin size values were generated aiding in generating various histograms. The histograms followed an exponential distribution matching context of the data (the purpose the data serves). Additionally, Q-Q plots were generated which was linear for the most part verifying the results obtained from the histograms. The Q-Q plots showed non-linearity at certain points due to the spread of the data and the presence of outliers. The exponential distribution was a good choice for the distribution of the data sets as it was verified by the resulted Q-Q plots.

1.3 Chi-Square

1.3.1 Theory Behind Chi-Square Test

The Chi-Square test is a hypothesis test that is used to determine whether there is a relationship between two categorical variables or not [3]. The objective of the Chi-Square test is to compare the observed frequency distribution of a categorical variable with the expected frequency distribution of the same variable [3].

2 Input Generation

2.1 Creating and Testing RNG

2.1.1 Discussion of Theory

To run our simulation accurately, it is required for wait times to be generated. For our simulation to be accurate these wait times must be based on historical data from the actual system that we are simulating. In the case of our project, we are completing input modeling on the time it takes for our inspections to complete and the time it takes for our workstations to completing assembly. We then use the data collected during our input modeling in combination with a random number generator to generate wait times for our simulation. For the simulation to accurately reflect the actual system these values must be values that could be recorded by the actual system and in a similar distribution to the historical data. Using arbitrary random values for our wait times would make it so that our simulation does not reflect the actual system.

The random numbers we generate should be uniform and independent. For random numbers to be uniform the generated values should be evenly distributed between 0 and our max value, usually 1. If our values were not uniform, the numbers would be skewed to certain values. This would result in our simulation being inaccurate since the generated wait times would not reflect the actual wait times of the system. For random numbers to be independent, the probability of generating a value should be independent of previously generated values. This ensures that the values being generated are truly random and not just based on previous values.

An issue we face with simulation is a computer is unable to generate truly random values. A computer is a completely deterministic system meaning there is no randomness involved. Using certain methods, computers are able to generate pseudo random numbers.

Two methods for pseudo random number generation are the linear congruential method and combined linear congruential generators. Combined linear congruential generators use multiple linear congruential methods that don't have an increment value, also known as multiplicative congruential generators. The advantage of combined linear congruential generators is that they have a longer period (the number of values generated before the generator repeats). For our purpose the linear congruential method will provide us with a long enough period.

To generate a sequence of pseudo random number using the linear congruential method, the following formula is used:

$$x_{i+1} = (aX_i + c) mod\ m \tag{3}$$

To ensure that our function has maximum density and period, we must choose appropriate values for a, X_i, c and m. The values used were L'Ecuyer's suggested values for an

RNG [4]. The values that were chosen are m = 32749, a = 9515, c = 0 and x.

As shown in 4, the function generateValue uses the chosen seed, a, c and m values to generate a series of pseudo random variables. The main function runs it 200 time and prints the results. To test for independence an autocorrelation test can be used. I will

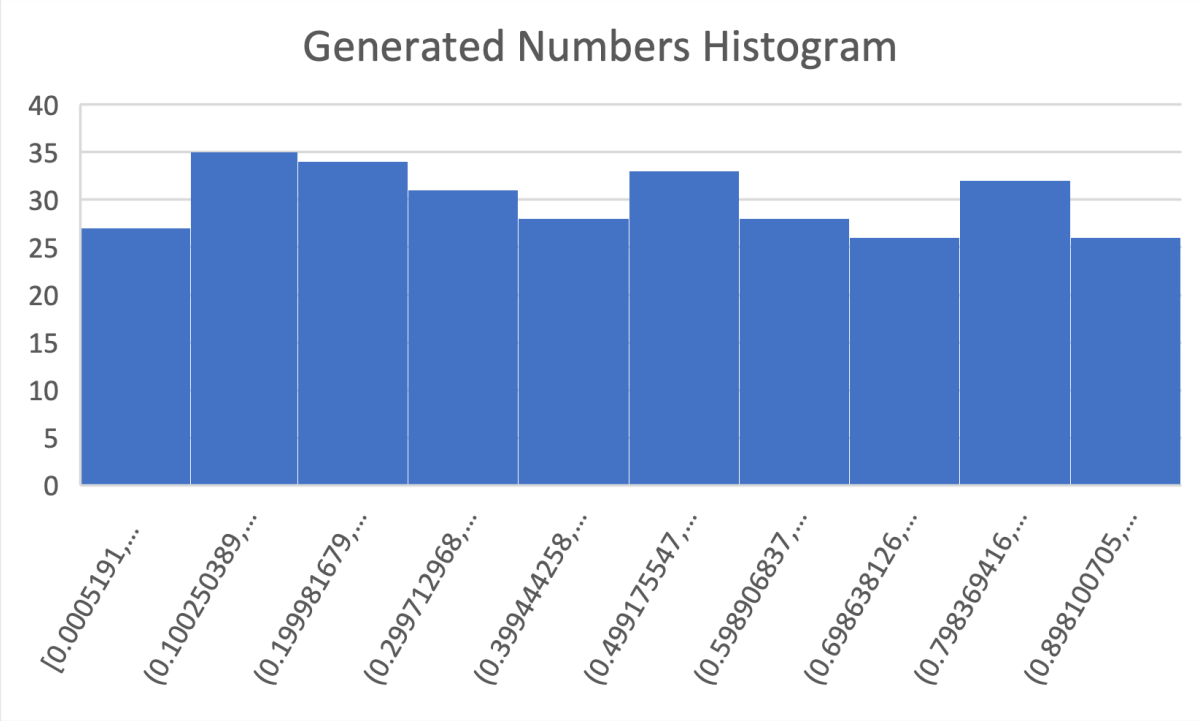

Figure 3: Generated Number Histogram

only complete a single autocorrelation test, but more tests should be done to ensure the generated values are independent. The following formula is used to test for autocorrelation. The value i is the first value tested and the value of m is the lag (autocorrelation between every m number of values generated).

Acronyms

IQR InterQuartile Range. 2

n Total Number of Data Points. 2

Q-Q Quantile-Quantile. 4–6

Q1 First Quartile. 2

Q3 Third Quartile. 2

References

[1] J. D. Triveri, "Determining Histogram Bin Width using the Freedman-Diaconis Rule," *Pleasure of Finding Things Out*, Mar. 2019. [Online]. Available: http://www.jtrive.com/determining-histogram-bin-width-using-the-freedman-diaconis-rule.html.

[2] *Understanding Q-Q Plots | University of Virginia Library Research Data Services + Sciences*, [Online; accessed 17. Mar. 2023], Mar. 2023. [Online]. Available: https://data.library.virginia.edu/understanding-q-q-plots.

[3] , *Chi-Square Test [Simply explained]*, [Online; accessed 17. Mar. 2023], Mar. 2023. [Online]. Available: https://www.youtube.com/watch?v=rpKzq64GA9Y&themeRefresh=1.

[4] P. L'Ecuyer, "Tables of linear congruential generators of different sizes and good lattice structure," *Math. Comput.*, vol. 68, no. 225, pp. 249–260, 1999, ISSN: 0025-5718. DOI: 10.1090/S0025-5718-99-00996-5.

Appendices

A Extract Data Class

```python
import pandas as pd


class Extract_Data:
    # inspector service time : servinsp1.dat
    # inspector 2 service time FOR C2 : servinsp22.dat
    # inspector 2 service time FOR C3 : servinsp23.dat

    # Workstation 1 service time : ws1.dat
    # Workstation 2 service time : ws2.dat
    # Workstation 3 service time : ws3.dat

    def __init__(self) -> None:
        self.servinsp1 = pd.read_csv("./data/servinsp1.dat", header=None)
        self.servinsp22 = pd.read_csv("./data/servinsp22.dat",
        ↪  header=None)
        self.servinsp33 = pd.read_csv("./data/servinsp23.dat",
        ↪  header=None)
        self.ws1 = pd.read_csv("./data/ws1.dat", header=None)
        self.ws2 = pd.read_csv("./data/ws2.dat", header=None)
        self.ws3 = pd.read_csv("./data/ws3.dat", header=None)

    def get_servinsp1(self):
        return self.servinsp1

    def get_servinsp22(self):
        return self.servinsp22

    def get_servinsp23(self):
        return self.servinsp33

    def get_ws1(self):
        return self.ws1

    def get_ws2(self):
        return self.ws2

    def get_ws3(self):
        return self.ws3

    def get_sim_duration(self):
        return 50


# def main():
#     extract_data = Extract_Data()
```

```
#     # print(extract_data.get_servinsp1().columns)
#     print(extract_data.get_servinsp1().iloc[0].values[0])
#     # drop the first row in place
#     extract_data.get_servinsp1().drop(
#         extract_data.get_servinsp1().index[0], inplace=True
#     )
#     print(extract_data.get_servinsp1().iloc[0].values)
#     extract_data.get_servinsp1().drop(
#         extract_data.get_servinsp1().index[0], inplace=True
#     )
#     print(extract_data.get_servinsp1().iloc[0].values)
#     # print(extract_data.get_servinsp22().head())
#     # print(extract_data.get_servinsp33().head())
#     # print(extract_data.get_ws1().head())
#     # print(extract_data.get_ws2().head())
#     # print(extract_data.get_ws3().head())


# if __name__ == "__main__":
#     main()
```

Listing 1: Extract Data Class

B Plot Histograms Class

```
import Extract_Data
import pandas as pd
import matplotlib.pyplot as plt


def calculate_iqr(data: pd.DataFrame) -> pd.Series:
    q1 = data.quantile(0.25)
    q3 = data.quantile(0.75)
    iqr = q3 - q1
    return iqr.rename("iqr")


def calculate_bin_width(data: pd.DataFrame, iqr):
    n = len(data)
    bin_width = 2 * iqr / n ** (1 / 3)
    return int(round((data.max() - data.min()) / bin_width))


# import data from Extract_Data
Extract_Data = Extract_Data.Extract_Data()
ws1 = Extract_Data.get_ws1()
ws2 = Extract_Data.get_ws2()
ws3 = Extract_Data.get_ws3()
servinsp1 = Extract_Data.get_servinsp1()
```

```
servinsp22 = Extract_Data.get_servinsp22()
servinsp23 = Extract_Data.get_servinsp23()

# calculate iqr for each data set
ws1_iqr = calculate_iqr(ws1).iloc[0]
ws2_iqr = calculate_iqr(ws2).iloc[0]
ws3_iqr = calculate_iqr(ws3).iloc[0]
servinsp1_iqr = calculate_iqr(servinsp1).iloc[0]
servinsp22_iqr = calculate_iqr(servinsp22).iloc[0]
servinsp23_iqr = calculate_iqr(servinsp23).iloc[0]

# concatenate the iqr series into a single dataframe
# iqr_df = pd.concat(
#     [ws1_iqr, ws2_iqr, ws3_iqr, servinsp1_iqr, servinsp22_iqr, servinsp23_iqr], axis=1
# )

# ws1_n = len(ws1)
# ws1_bin_width = 2 * ws1_iqr * (ws1_n ** (-1 / 3))
# print(f"ws1_iqr: {ws1_iqr}, ws1_n: {ws1_n}, ws1_bin_width: {ws1_bin_width}")

# plot the data using histogram and calculated width
bins_ws1 = calculate_bin_width(ws1, ws1_iqr)
bins_ws2 = calculate_bin_width(ws2, ws2_iqr)
bins_ws3 = calculate_bin_width(ws3, ws3_iqr)
bins_servinsp1 = calculate_bin_width(servinsp1, servinsp1_iqr)
bins_servinsp22 = calculate_bin_width(servinsp22, servinsp22_iqr)
bins_servinsp23 = calculate_bin_width(servinsp23, servinsp23_iqr)

# print all bins
print(
    f"bins_ws1: {bins_ws1}, bins_ws2: {bins_ws2}, bins_ws3: {bins_ws3}, bins_servinsp1: {bins_servinsp1}, bins_servinsp22: {bins_servinsp22}, bins_servinsp23: {bins_servinsp23}"
)

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(10, 10))

# plot each histogram on its own subplot
axes[0, 0].hist(ws1, bins=bins_ws1)
axes[0, 0].set_title("ws1")
axes[0, 0].set_xlabel(f"Bin Size: {bins_ws1}")

axes[0, 1].hist(ws2, bins=bins_ws2)
axes[0, 1].set_title("ws2")
axes[0, 1].set_xlabel(f"Bin Size: {bins_ws2}")

axes[1, 0].hist(ws3, bins=bins_ws3)
axes[1, 0].set_title("ws3")
axes[1, 0].set_xlabel(f"Bin Size: {bins_ws3}")

```

```
axes[1, 1].hist(servinsp1, bins=bins_servinsp1)
axes[1, 1].set_title("servinsp1")
axes[1, 1].set_xlabel(f"Bin Size: {bins_servinsp1}")

axes[2, 0].hist(servinsp22, bins=bins_servinsp22)
axes[2, 0].set_title("servinsp22")
axes[2, 0].set_xlabel(f"Bin Size: {bins_servinsp22}")

axes[2, 1].hist(servinsp23, bins=bins_servinsp23)
axes[2, 1].set_title("servinsp23")
axes[2, 1].set_xlabel(f"Bin Size: {bins_servinsp23}")

# adjust layout and show the plot
# set common labels
fig.text(0.5, 0.04, "Values", ha="center")
fig.text(0.04, 0.5, "Frequency", va="center", rotation="vertical")

# adjust spacing between subplots
plt.subplots_adjust(left=0.1, bottom=0.1, right=0.9, top=0.9, wspace=0.4,
↪  hspace=0.4)

# show the plot
plt.show()

# save the plots
fig.savefig("./m2/plots/data_histogram_plots.png")

# # plot all 6 histograms
# plt.hist(ws1, bins=bins_ws1, label="ws1")
# plt.hist(ws2, bins=bins_ws2, label="ws2")
# plt.hist(ws3, bins=bins_ws3, label="ws3")
# plt.hist(servinsp1, bins=bins_servinsp1, label="servinsp1")
# plt.hist(servinsp22, bins=bins_servinsp22, label="servinsp22")
# plt.hist(servinsp23, bins=bins_servinsp23, label="servinsp23")

# # show the plot
# plt.legend()
# plt.show()
```

Listing 2: Plot Histograms Class

C Q-Q Plots Code

```
import Extract_Data
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
import numpy as np

```

```
7
8   def build_qq_plot(data):
9       """Builds a Q-Q plot for the input data."""
10      stats.probplot(data.values.ravel(), dist="norm", plot=plt)
11      plt.title("Q-Q plot")
12      plt.show()
13
14
15  def build_poisson(data):
16      """Builds a Poisson distribution plot for the input data."""
17      mu = np.mean(data)
18      x = np.arange(stats.poisson.ppf(0.01, mu), stats.poisson.ppf(0.99,
        ↪  mu))
19      pmf = stats.poisson.pmf(x, mu)
20      plt.plot(x, pmf, "bo", ms=8)
21      plt.vlines(x, 0, pmf, colors="b", lw=5, alpha=0.5)
22      plt.title("Poisson distribution plot")
23      plt.show()
24
25
26  def build_normal(data):
27      """Builds a normal distribution plot for the input data."""
28      mu, std = stats.norm.fit(data)
29      xmin, xmax = plt.xlim()
30      x = np.linspace(xmin, xmax, 100)
31      p = stats.norm.pdf(x, mu, std)
32      plt.plot(x, p, "k", linewidth=2)
33      plt.title("Normal distribution plot")
34      plt.show()
35
36
37  def build_exponential(data):
38      """Builds an exponential distribution plot for the input data."""
39      loc, scale = stats.expon.fit(data)
40      xmin, xmax = plt.xlim()
41      x = np.linspace(xmin, xmax, 100)
42      p = stats.expon.pdf(x, loc, scale)
43      plt.plot(x, p, "k", linewidth=2)
44      plt.title("Exponential distribution plot")
45      plt.show()
46
47
48  def build_weibull_constant_beta(data):
49      """Builds a Weibull distribution plot with constant beta for the input
        ↪  data."""
50      shape, loc, scale = stats.weibull_min.fit(data, floc=0)
51      xmin, xmax = plt.xlim()
52      x = np.linspace(xmin, xmax, 100)
53      p = stats.weibull_min.pdf(x, shape, loc, scale)
54      plt.plot(x, p, "k", linewidth=2)
55      plt.title("Weibull distribution plot with constant beta")
56      plt.show()
```

```

def build_weibull_double_param(data):
    """Builds a Weibull distribution plot with double parameters for the
    ↪  input data."""
    shape, loc, scale = stats.weibull_min.fit(data)
    xmin, xmax = plt.xlim()
    x = np.linspace(xmin, xmax, 100)
    p = stats.weibull_min.pdf(x, shape, loc, scale)
    plt.plot(x, p, "k", linewidth=2)
    plt.title("Weibull distribution plot with double parameters")
    plt.show()


def build_log_normal(data):
    """Builds a log-normal distribution plot for the input data."""
    s, mu, _ = stats.lognorm.fit(data)
    xmin, xmax = plt.xlim()
    x = np.linspace(xmin, xmax, 100)
    p = stats.lognorm.pdf(x, s=s, loc=0, scale=np.exp(mu))
    plt.plot(x, p, "k", linewidth=2)
    plt.title("Log-normal distribution")
    plt.show()


# import data from Extract_Data
Extract_Data = Extract_Data.Extract_Data()
ws1 = Extract_Data.get_ws1()
ws2 = Extract_Data.get_ws2()
ws3 = Extract_Data.get_ws3()
servinsp1 = Extract_Data.get_servinsp1()
servinsp22 = Extract_Data.get_servinsp22()
servinsp23 = Extract_Data.get_servinsp23()

# build_qq_plot(ws1)
# build_poisson(ws1)
# build_normal(ws1)
# build_exponential(ws1)
# build_weibull_constant_beta(ws1)
# build_weibull_double_param(ws1)
build_log_normal(ws1)
```

Listing 3: Q-Q Plot Code

D Random Number Generation Code

```
seed = 27
a = 9515
c = 0
```

```
m = 32749

def generateValue():
    global seed
    x = (a * seed + c) % m
    seed = x
    return x/m

def main():
    for i in range(300):
        print(generateValue())

if __name__ == "__main__":
    main()
```

Listing 4: Random Number Generation Code