E1_binning

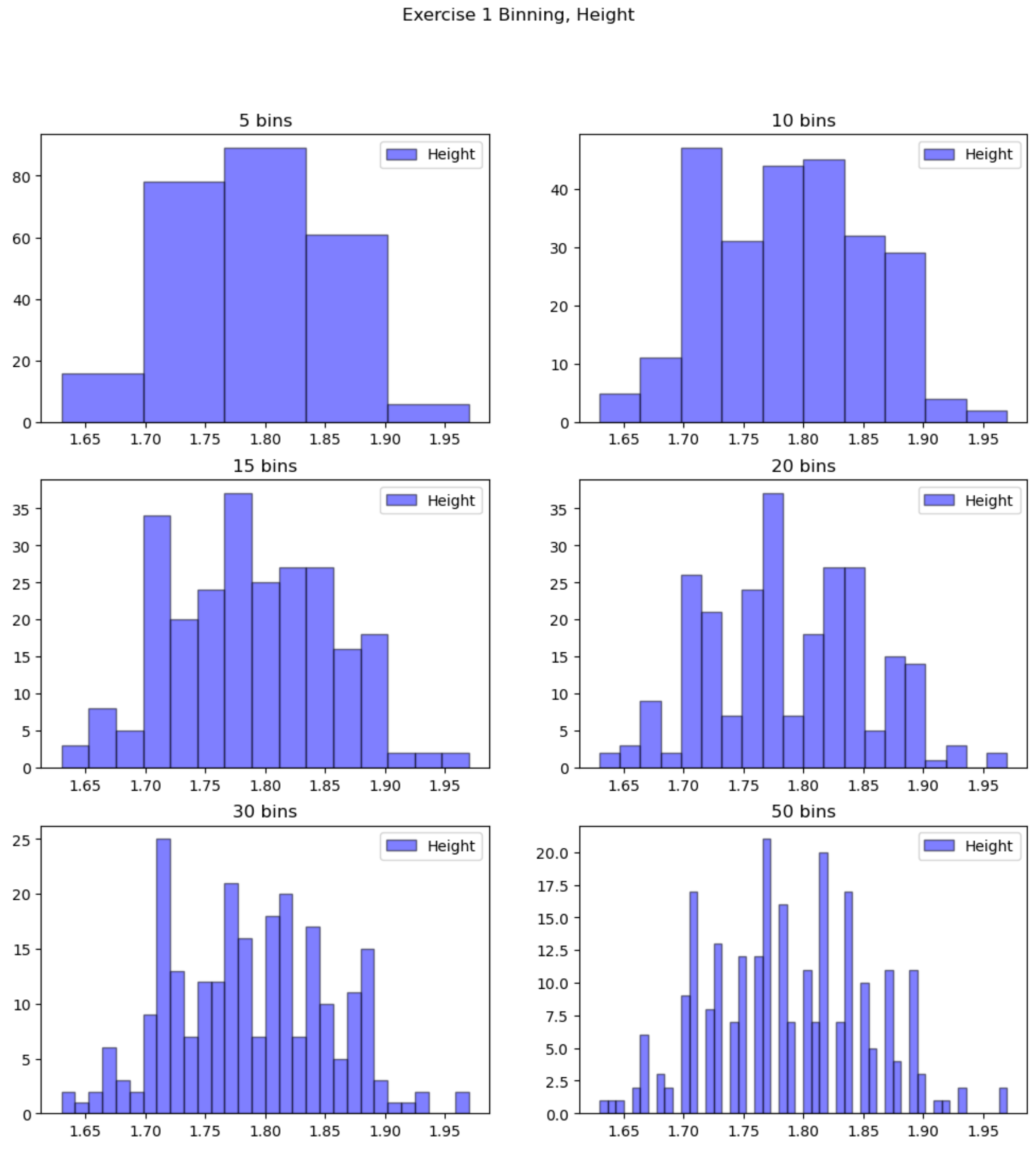
April 14, 2023

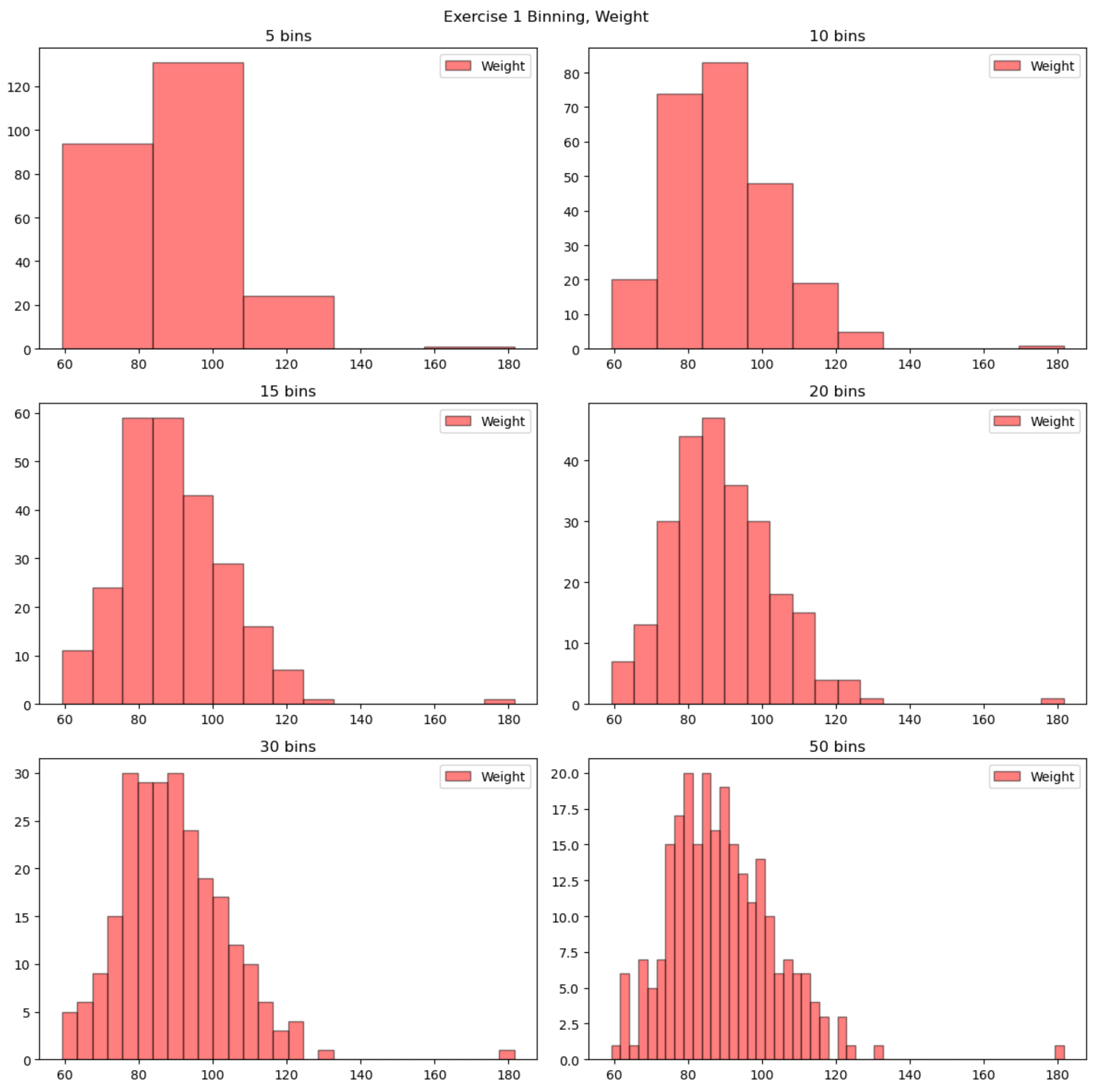
1 1 a)

```
[ ]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
```

```
[ ]: # Read the data
     data = pd.read_csv("height_weight.csv")

     # Extract height and weight
     heights = data["height"]
     weights = data["weight"]

     # Bins
     bins_list = [5, 10, 15, 20, 30, 50]

     # Create the figure and axes
     fig, axes = plt.subplots(3, 2, figsize=(12, 12))
     fig.suptitle("Exercise 1 Binning, Height")

     # Loop through bins and create subplots
     for idx, bins in enumerate(bins_list):
         row = idx // 2
         col = idx % 2

         ax = axes[row, col]
         ax.hist(heights, bins, alpha=0.5, label="Height", color="blue",␣
       ↪edgecolor="black")
         ax.set_title(f"{bins} bins")
         ax.legend(loc="upper right")

     # Create the figure and axes
     fig2, axes2 = plt.subplots(3, 2, figsize=(12, 12))
     fig2.suptitle("Exercise 1 Binning, Weight")

     # Loop through bins and create subplots
     for idx, bins in enumerate(bins_list):
```

```
    row = idx // 2
    col = idx % 2

    ax2 = axes2[row, col]
    ax2.hist(weights, bins, alpha=0.5, label="Weight", color="red",␣
↪edgecolor="black")
    ax2.set_title(f"{bins} bins")
    ax2.legend(loc="upper right")

plt.tight_layout()
plt.show()
```

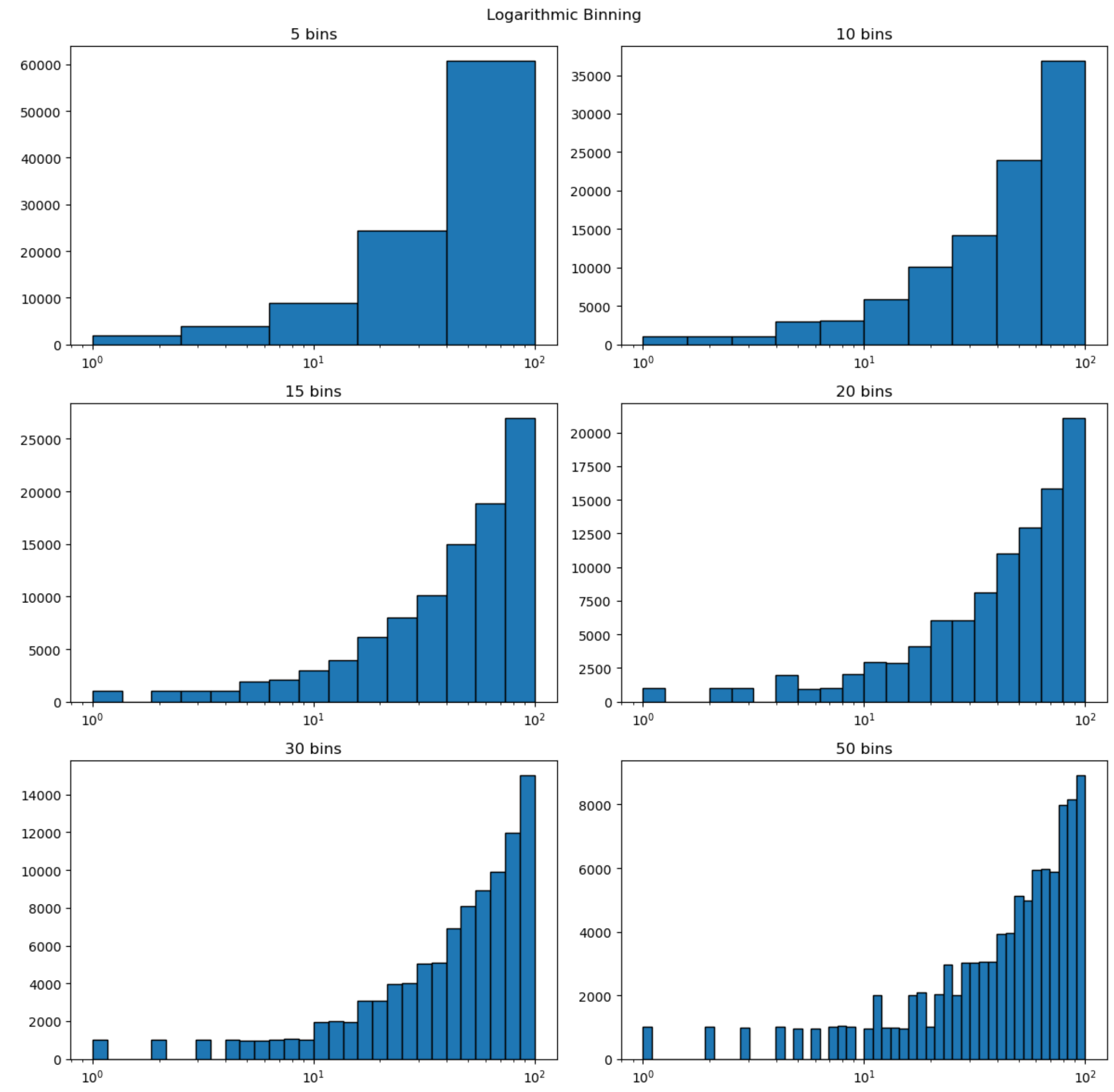




2 c)

```
[ ]: # Draw uniformly distributed integers
     random_integers = np.random.randint(1, 101, size=int(1e5))

     # Bins
     bins_list = [5, 10, 15, 20, 30, 50]

     # Create the figure and axes
     fig, axes = plt.subplots(3, 2, figsize=(12, 12))
```

```
fig.suptitle("Logarithmic Binning")

# Loop through bins and create subplots
for idx, bins in enumerate(bins_list):
    row = idx // 2
    col = idx % 2

    # Create logarithmic bins
    log_bins = np.logspace(np.log10(1), np.log10(100), bins+1)

    ax = axes[row, col]
    ax.hist(random_integers, bins=log_bins, edgecolor="black")
    ax.set_title(f"{bins} bins")
    ax.set_xscale("log")

plt.tight_layout()
plt.show()
```



High-resolution binning can result in “holes” or noisy representations of the data, which might misrepresent the underlying distribution. There are several ways to tackle these issues:

- Kernel Density Estimation (KDE): KDE is a non-parametric method that can be used to smooth the histogram by estimating a continuous probability density function from the data. It involves placing a kernel (e.g., Gaussian) at each data point and summing these individual kernels to obtain the overall distribution.
- Adaptive binning: Rather than using a fixed bin width, adaptive binning adjusts the bin widths depending on the data density. Bins are wider in regions with sparse data and narrower in regions with denser data. This approach can help minimize the impact of noise and holes in the histogram.
- Bayesian Blocks: This is a more advanced technique that involves partitioning the data into blocks based on the Bayesian information criterion (BIC). This method can better represent

the data by finding the optimal binning based on the data itself, rather than arbitrarily choosing the number of bins.

- Data smoothing: You can apply various data smoothing techniques, like moving averages or Savitzky-Golay filtering, to the histogram to reduce noise and better represent the underlying distribution.
- Selecting an appropriate number of bins: You can use different rules of thumb to determine the optimal number of bins, like the Freedman-Diaconis rule, Sturges' rule, or the Square-root rule. These rules take into account the sample size and data characteristics to suggest a suitable number of bins.

Keep in mind that each technique has its assumptions and limitations, and the best choice depends on the specific characteristics of your data and the goals of your analysis.