4.9 Exercises

2. Why do the different Gaussians have different peak heights? It's clearly related to their shape parameter (σ) because that's the key variable that you manipulated. Think of an answer

[11] I'm pretty sure it's a *statistically significant* improvement, although I admit I have no data on this.

before reading on.

Compute the sum over all values for each Gaussian. Make sure you sum across the correct dimension of the matrix of Gaussians: The number of sums must be equal to the number of Gaussians, not the number of x-axis values.

Are you surprised at the result? To help this make sense, instead of computing a sum, compute the discrete integral, which is obtained by multiplying the Gaussian by the discretization of the grid over which the Gaussian was evaluated, that is, the distance between successive x-axis values. Does this help you to understand the result?

The answer to the question at the outset of this exercise is that the Gaussians are designed to integrate to 1. But why do only the first few Gaussians sum to 1 while the later Gaussians sum to less than 1? The answer is that a true Gaussian is defined from $x = -\infty$ to $x = +\infty$; a restricted domain is not guaranteed to integrate to 1. You can see this by inspecting the middle panel of Figure 4.29: Many Gaussians do not taper down to zero at $x = |3|$.

Now re-run the code for the previous exercise but increase the domain of x, e.g., to ± 5. The larger the domain, the closer all the integrals get to 1.

Finally, remove the multiplicative factor in the beginning of the equation, and reproduce Figure 4.29 and the integral above. Now the sums don't equal 1, but all Gaussians have a peak value of 1. There's nothing wrong with this result; there are different ways to normalize data and functions, and different normalizations have different implications and are appropriate for different situations. That is a general theme — and a source of confusion — when working with data. More on this topic in Chapter 6.

4. In the previous exercise you saw that the `ddof` parameter in `np.var()` needed to be adjusted to compute the sample variance. Does it really matter if you use the population or the sample variance (that is, if you divide by N or $N-1$)? Intuitively, it should make sense that it matters more with smaller sample sizes (consider, for example, that the proportional difference between 4 and 5 is much larger than the proportional difference between 999 and 1000).

Let's run an empirical experiment to explore this. Generate random integers between -100 and +100, compute their variances twice, setting the `ddof` parameter to 0 and to 1, and compute their difference (set the subtraction such that the difference is positive). Repeat the above procedure for sample sizes ranging from 5 to 100.

If you are using R, you need to adjust the variance explicitly, because the `var()` function always scales by $N-1$. Thus, to calculate the population variance of variable `x`, use `var(x)*(N-1)/N`.

Because we are using random numbers, we should repeat the experiment multiple times and average the results. Therefore, run the experiment described above 25 times, each time generating a new random dataset. Create an errorbar plot that shows the average over 25 runs, and their standard deviation, for each sample size. My results are shown in Figure 4.30; yours should look similar.

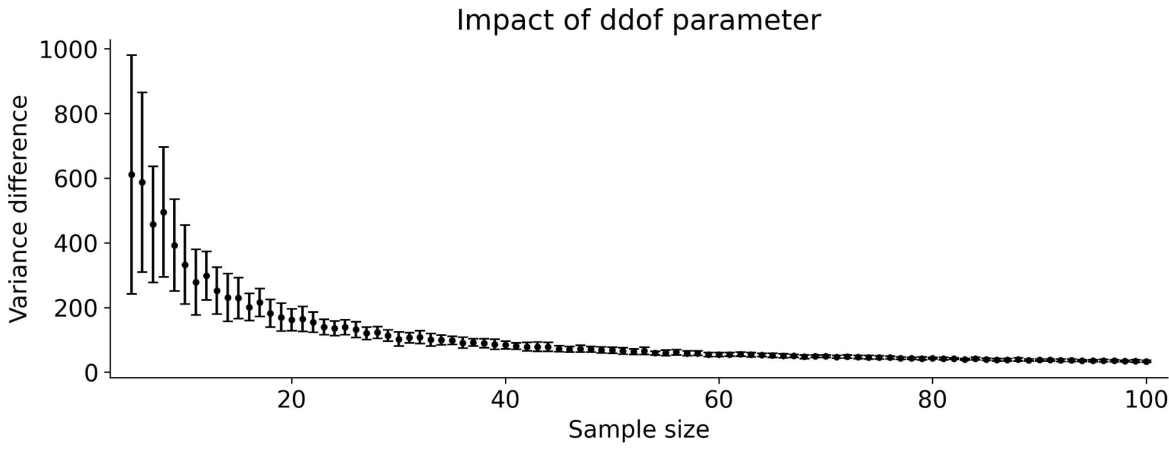

Figure 4.30: Visualization for Exercise 4.

Three observations from this result. First, the impact of the `ddof` parameter decreases with increasing sample size. Second, the error bars indicate that different repetitions of the same experiment give different results, especially with smaller sample sizes. Increased variability and uncertainty in small sample sizes is an annoyance in statistics that you will see many times in this book, and in your post-education applications.

Third, the differences in variances are in the 200-600 range for relatively small sample sizes. Is that a "big" difference? Repeat the experiment but use numbers between -10 and +10. The y-axis values are now much smaller. In other words, it is difficult to draw conclusions based on the numerical values because they depend on the range of the data values. Perhaps we could gain a deeper understanding of the plot if we could apply some normalization (segue to the next exercise)...

9. IQR and standard deviation are conceptually similar — both relate to the spread of data around their central tendency (median for IQR; mean for standard deviation). In fact, for a normal distribution, IQR is approximately 1.35σ.

Confirm this by creating 10,000 random numbers drawn from a normal distribution, and compute the standard deviation and IQR. You already know that a normal distribution has a theoretical standard deviation of 1, and the empirical standard deviation should be fairly close to that value. This means that the empirical IQR should be close to 1.35.

[12] I used the `annotate` function in `matplotlib` to plot the arrows; in general in these exercises, you should focus on implementing the statistical concepts and don't stress about plotting details.

Repeat the calculations using the data pushed through the natural exponential function, that is, e^X where X is the normally distributed data. Is the IQR still approximately 1.35σ?

Next, write code to produce a visualization like Figure 4.34. This visualization shows a histogram using bins defined by the Friedman-Diaconis rule, quartiles 1 and 3 in the solid black vertical lines, and one standard deviation below and above the mean in the dashed line, scaled by 1.35, which should match the quartiles. Panel A shows the data generated from a normal distribution and panel B shows the data pushed through the natural exponential function.

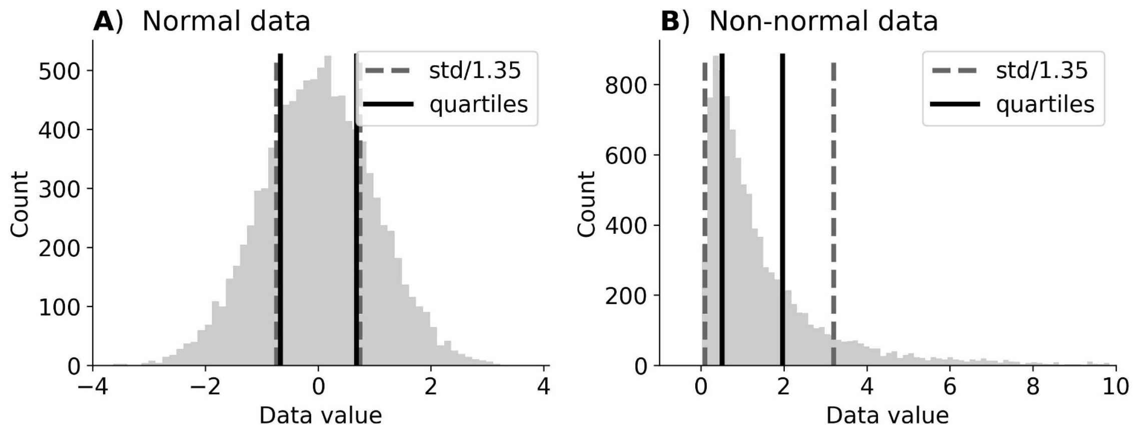

Figure 4.34: Visualization for Exercise 9.

Some observations: (**1**) For a normal distribution (panel A), the quartiles and standard deviation are comparably far away from the center of the distribution (not surprising, considering that the mean and median are nearly the same, and that the distribution is symmetric). (**2**) For a non-normal distribution (panel B), the quartiles and standard deviation are more dissimilar, because the mean and median are further from each other[13]. (**3**) For the non-normal distribution, the standard deviation is difficult to interpret — indeed, the standard deviation below the mean is *negative* despite the dataset being entirely positive-valued.

More generally, this exercise highlights the importance of understanding the shape of the data distribution when interpret-

[13]If you would like to expand on this exercise, you can additionally indicate the locations of the means and medians.

ing descriptive statistics.

10. The goal of this exercise is to develop an algorithm to compute the empirical FWHM of a Gaussian-like distribution. Write a function that follows the following algorithm (refer back to Figure 4.17 on page 127).

1. **Define the function**. I called mine `empFWHM`. The function takes two arguments, `x` and `y`, corresponding to arrays that represent the x and y coordinates of a set of points on a curve.
2. **Normalize the data**: The y-values should be in the range of [0,1]. This is called "min-max scaling," and I'll discuss it in Chapter 6. For now, simply apply the formula
   $\widetilde{y} = (y - \min(y))/(\max(y) - \min(y))$
3. **Find the peak**: Find the index of the maximum y-value.
4. **Find the pre-peak half-maximum point**: Because the function is normalized, the pre-peak half-maximum point is the x-value where $y \approx .5$. Because these are empirical data, there probably won't be a value that *exactly equals* .5, so you will need to find the value *closest to* .5.
5. **Find the post-peak half-maximum point**: Same as above but for the half-maximum *after* the peak. Be mindful of indexing.
6. **Compute and return the FWHM**: The FWHM is the span on the x-axis from the pre-peak to the post-peak half-maxima. It's also handy to export the x-axis half-maxima values for subsequent plotting.

Test your algorithm on a pure Gaussian function using Equation 4.10, where you can also compute the analytical FWHM using Equation 4.11. If you use $\sigma = 1.9$ and x ranging from -8 to +8 in 1001 steps, you should get that FWHM=4.47 for the analytical solution and 4.46 for the empirical estimate.

Next, test your function over a range of σ values from .1 to 5.

My results are shown in Figure 4.35. It seems like the empirical and analytical estimates matched closely until around $\sigma = 3.5$ and then diverged. What's the problem? My explanation and fix are in the online code.

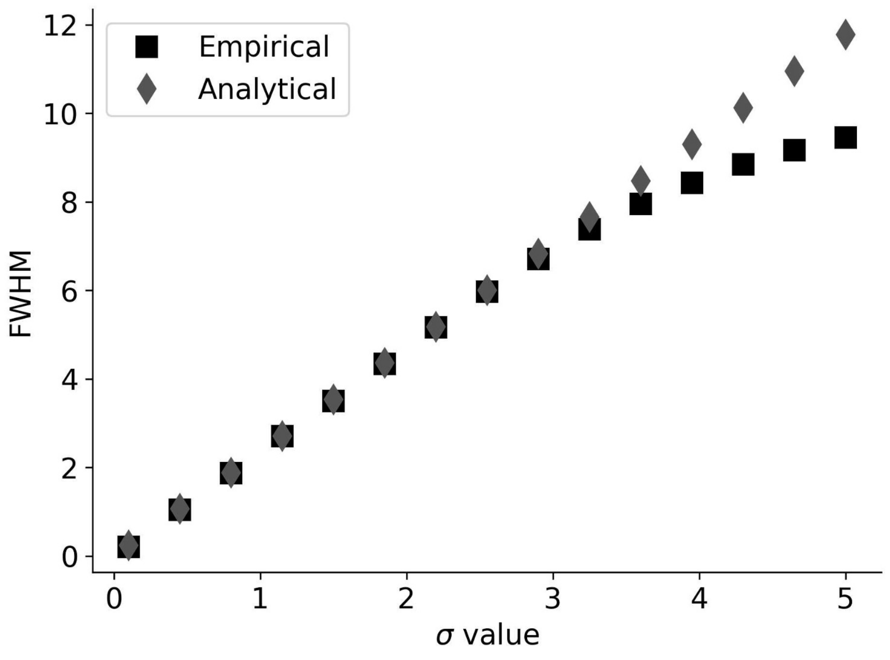

Figure 4.35: Visualization for Exercise 10.

Now that you have confirmed that the function works as expected, compute the empirical FWHM of a histogram of sampled data. To generate Figure 4.36, I used 12,345 data points randomly sampled from a Gaussian distribution and a histogram with 100 bins. Show the histogram with a dashed line indicating the FWHM, and report the empirical FWHM, as in 4.35.

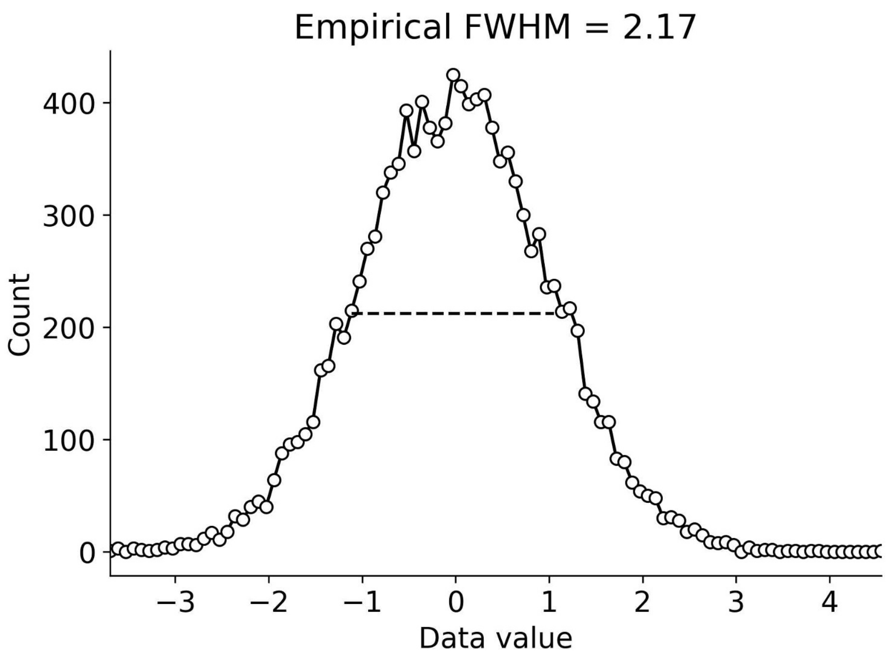

Figure 4.36: Visualization for Exercise 10.

11. The purpose of this exercise is to explore the implications of different histogram bin rules on the visual appearance of a distribution.

Create a dataset of 1000 random numbers drawn from a normal distribution. Extract the histogram using four binning rules: Arbitrary (set to 40 bins), FD, Sturges, and Scott[14]. Plot all histograms in one graph, as in Figure 4.37.

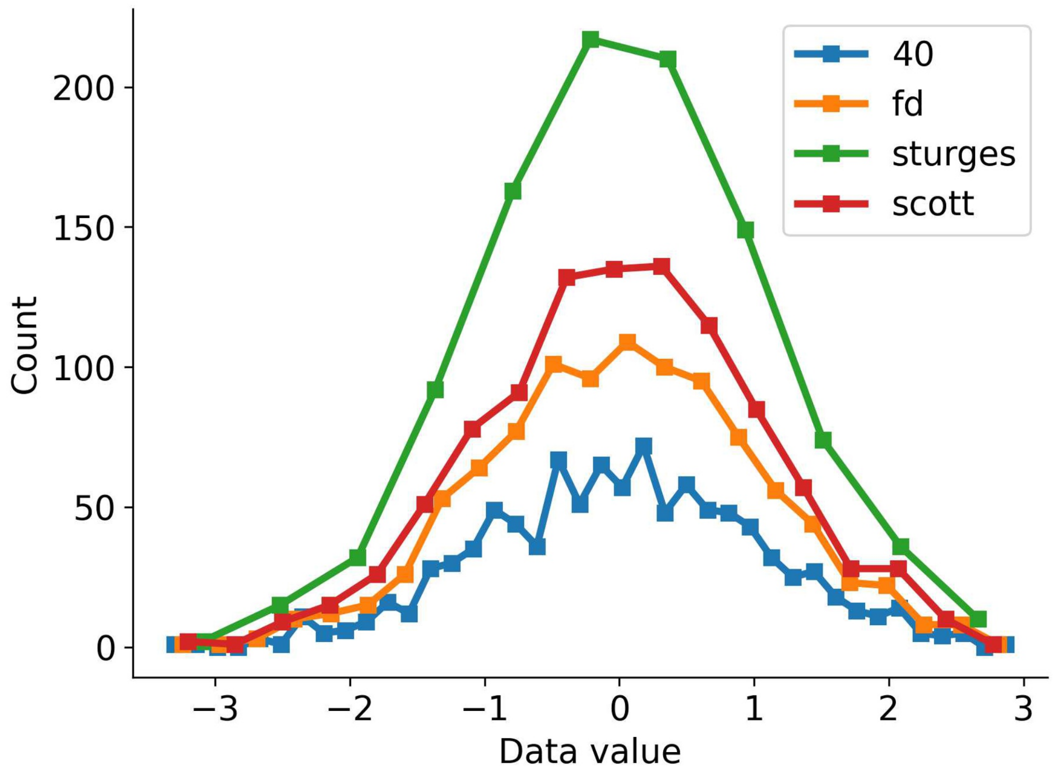

Figure 4.37: Visualization for Exercise 11. The lines are in color, which will look better on your screen than in the printed version of this book (assuming that you have a color screen).

What is your opinion about the results? Why are some histograms taller than others, although they are generated from the same dataset? Answers to these questions are in the online code.

Note about R: For small sample sizes, R may use different algorithms than what you specified. If your plot appears to

[14]I didn't discuss Scott's rule earlier, but it is comparable to the FD rule in that it defines bin width based on the standard deviation and sample size.

have fewer than four lines, then it is likely that, e.g., 40 bins and FD produce identical results with overlapping lines.

Once you have code for this exercise, it is easy to explore different distributions, for example, using uniform-distributed data. You can also explore other bin-defining rules, other sample sizes, and the impact of normalizing each distribution to a max of 1.

One important take-home message from this exercise is that the method you choose to select the number of histogram bins usually doesn't make a qualitative impact on the results. That is, the different rules have a numerical impact, but would not change the qualitative interpretation of the data.

6. The goal here is to compute an empirical cdf of a pdf that is not supplied by Python or R. Create the pdf by summing together two Gaussian pdfs with an x-axis grid of 1001 points linearly spaced between -6 and +6. One pdf has its x variable shifted by -2.7 while the other has its x variable shifted by +2.7. This shifting creates a bimodal distribution, which you can see in Figure 8.15A.

Compute the cdf of this distribution by computing the dx-scaled cumulative sum of the pdf. You will see in Figure 8.15B that this is not a proper cdf — it extends above 1. Figure out why this happens and how to fix it. My answers are in the next paragraph.

The problem begins with the pdf. Because we've summed two individual pdfs, the result is not a true pdf even when scaled by dx. One solution is to apply the unit sum normalization introduced in Exercise 2. This creates a more accurate estimate of a pdf, and its cumulative sum (without any additional scaling) gives a sensible result shown in Figure 6C. Another option is to leave the pdf unnormalized and then divide the cdf by its final value. I show this solution in the online code.

7. One more exercise on cdfs. The goal here is to demonstrate that the pdf can be computed as the derivative of the cdf. Use `scipy` or R to create a cdf of the lognormal distribution, using x-axis spacing from 0 to 10 in 200 steps, and location and scale parameters of 1 and .5.

Then create two pdfs, one using `stats.lognorm.pdf` (`dlnorm` in R) and one taking the difference (the discrete derivative function) of the cdf. Create a plot like Figure 8.16. What normalizations are necessary for which pdf-creation methods?

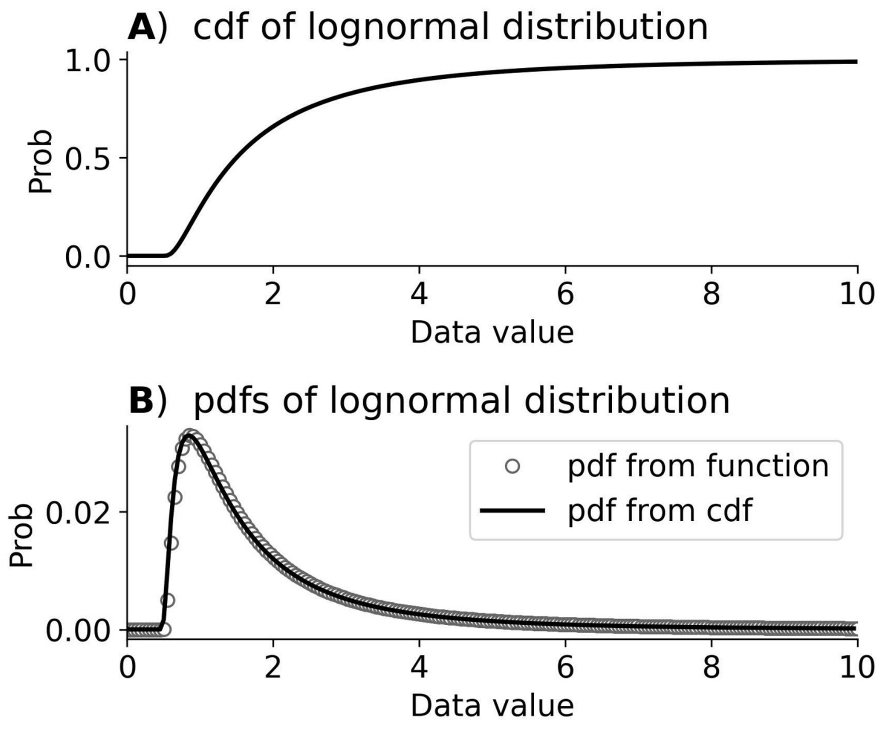

Figure 8.16: Visualization for Exercise 7.