9 Running Monte Carlo simulations

This chapter covers

- Monte Carlo simulations
- Mathematical foundations
- Applications to discrete random variables
- Applications to continuous random variables
- Generating and interpreting simulation results

Monte Carlo simulations are a powerful and versatile tool used to model and analyze complex systems and processes across many fields. They are named after the Monte Carlo Casino in Monaco, reflecting the reliance on random sampling methods, which are reminiscent of the inherent randomness in gambling. This connection stems from early applications of these methods in probabilistic studies during the mid-20th century. The basic idea behind Monte Carlo simulations is to use random sampling to generate a range of possible outcomes for a given problem. These outcomes can then be analyzed to understand the probability distribution and potential variability of the results. This approach is particularly useful when analytical solutions are impractical or impossible, such as in systems with high complexity, nonlinear relationships, or numerous interacting variables. Monte Carlo simulations provide a way to approximate solutions in these challenging scenarios.

Our goal is to equip you with the knowledge and practical skills to effectively run Monte Carlo simulations and understand their mathematical foundations. This chapter begins with simulations on discrete random variables, where outcomes are finite and distinct (e.g., the number of absentee employees). It then extends to continuous random variables, where outcomes can take on any value within a range (e.g., the closing prices of a public stock). Understanding the nuances between these two types of data is crucial for effectively applying Monte Carlo methods to real-world problems. By the end of this chapter, you will understand not only how to implement these simulations but also how to interpret their results to make informed decisions in the face of uncertainty. By combining theoretical explanations with hands-on examples, this chapter bridges the gap between abstract concepts and their practical applications. This dual approach ensures that you will gain both a foundational understanding and the practical skills necessary to confidently run Monte Carlo simulations.

9.1 Applications and benefits of Monte Carlo simulations

Monte Carlo simulations are versatile tools with widespread applications across various fields. In finance, they model portfolio performance under extreme market conditions by simulating asset price behaviors under diverse scenarios, helping assess risk and uncertainty. Engineers use Monte Carlo methods to predict the structural integrity of systems, such as bridges under varying loads, by modeling the effects of uncertain parameters. In project management, these simulations are invaluable for forecasting timelines and budgets, accounting for uncertainties in task durations and costs. Scientists employ Monte Carlo techniques to simulate complex physical processes, such as particle diffusion or molecular behavior in gases, offering insights into phenomena that cannot be directly observed.

One of the key strengths of Monte Carlo simulations is their ability to handle both discrete and continuous random variables (see chapter 2). Discrete random variables, such as the number of defective items in a batch, take on specific values. In contrast, continuous random variables, such as the time required to complete a task, can take on any value within a defined range. By incorporating randomness and variability into the modeling process, Monte Carlo simulations create realistic representations of real-world systems, capturing the inherent uncertainties that deterministic models can miss.

Compared to traditional deterministic models—which provide single-point estimates based on fixed inputs—Monte Carlo simulations generate a distribution of outcomes based on variability in those inputs. For example, rather than predicting that a project will take exactly 20 days, a Monte Carlo simulation might reveal a 70% chance of completion within 22 days and a 10% chance of exceeding 25 days. This probabilistic view gives stakeholders a clearer understanding of risks and tail events. Moreover, unlike closed-form analytical models, Monte Carlo can accommodate nonlinear relationships, irregular distributions, and interaction effects between variables—making it especially valuable in complex, real-world scenarios where exact solutions are either unavailable or unreliable.

Monte Carlo simulations are particularly useful when systems exhibit significant uncertainty or variability. Traditional deterministic models rely on fixed input values and provide single-point predictions, which fail to account for the full range of possible outcomes in complex systems. Monte Carlo simulations overcome this limitation by generating a wide range of scenarios based on random sampling. This allows for deeper insights into the probabilities of various outcomes and the factors driving these changes, making them an ideal tool for systems that evolve over time or respond to varying conditions.

The process of conducting a Monte Carlo simulation involves several essential steps. First, a mathematical model of the system is developed, incorporating the relevant variables and their relationships. Next, probability distributions for the uncertain variables are defined. These distributions describe the possible range of values each variable can take and the likelihood of each value. The simulation then samples values from these distributions to generate a large number of scenarios or trials, each representing a possible outcome based on the sampled inputs. Finally, the results of these trials are aggregated and analyzed to produce a probability distribution of outcomes, offering insights into the variability and uncertainty inherent in the system.

For example, when modeling stock prices, historical data is used to create a probability distribution for price changes. By simulating multiple scenarios, analysts can visualize the potential range of future prices and calculate key summary statistics, such as the mean, standard deviation, and price range. This provides a clearer understanding of the likely behavior of the stock over time, empowering investors to make data-driven decisions.

Monte Carlo simulations enable decision-makers to navigate uncertainty and make strategic choices with confidence. By understanding the likelihood of potential outcomes, stakeholders can tailor their actions to mitigate risks or capitalize on opportunities. For instance, if a simulation reveals a high probability of significant stock price fluctuations, investors may adopt a more cautious strategy. Conversely, if the simulation indicates stability, they might invest more aggressively. Ultimately, Monte Carlo simulations are robust analytical tools that help individuals and organizations address uncertainty in complex environments effectively.

9.2 Step-by-step process

Monte Carlo simulations involve a systematic approach to model and analyze uncertainty and variability in complex systems. This section breaks down the process into six essential steps, thereby offering a clear roadmap for implementing these simulations effectively:

1 *Establishing a probability distribution*—Define the probability distribution for each uncertain variable in the system. These distributions describe the possible values a variable can take and the likelihood of each value occurring. For example, a normal distribution might model daily stock returns, whereas a Poisson distribution might be used to offset employee absenteeism. Selecting the appropriate distribution—and specifying its parameters, such as mean and variance—is critical and often informed by historical data, domain knowledge, or assumptions about the behavior of the variable.

2 *Computing a cumulative probability distribution*—Transform the probability distribution into a cumulative distribution function (CDF). The CDF accumulates the probabilities of all values up to a given point, making it easier to map random numbers to outcomes. For instance, if a variable has probabilities of 0.2, 0.3, and 0.5 for three possible outcomes, its cumulative probabilities will be 0.2, 0.5, and 1.0. The CDF ensures every possible outcome corresponds to a specific range of cumulative probabilities.

3 *Establishing an interval of random numbers*—Using the cumulative probability distribution, assign intervals of random numbers to each possible outcome. These intervals match the cumulative probabilities, such that a random number generated within an interval corresponds to its associated outcome. For example, if a variable has a cumulative probability of 0.4 for its first outcome, the interval [0.00, 0.40] maps to that outcome. This mapping ensures the simulation aligns with the defined probabilities.

4 *Generating random numbers*—Generate a series of random numbers to simulate variability in the system. Each random number is assigned to one of the intervals established in the previous step, determining the corresponding outcome. Python's pandas library provides robust random number generators that support a variety of probability distributions. Setting a random seed ensures reproducibility of the simulation results.

5 *Simulating a series of trials*—Run the simulation by conducting multiple trials. In each trial, random values are sampled for all variables, representing one possible outcome of the system. The greater the number of trials, the more robust the simulation results, as they reduce the influence of anomalies or extreme values. For example, a simulation with 10,000 trials provides a more reliable approximation of the system's behavior than one with only 100 trials.

6 *Analyzing the results*—Aggregate and analyze the outcomes from all trials to generate insights. The results can be visualized using histograms, cumulative distribution plots, or scatterplots. Key summary statistics, such as the mean, standard deviation, and range, help describe the variability and central tendencies of the simulated outcomes. For instance, if simulating project completion times, the mean represents the expected duration, whereas the range shows the best- and worst-case scenarios. Analyzing these results allows decision-makers to evaluate risks and make informed choices.

We will begin by manually performing a small number of Monte Carlo simulations on discrete data. This hands-on approach will break down the fundamental concepts and steps, providing a solid foundation for understanding the methodology. Next, we will scale up the process to run hundreds of simulations on the same discrete data, demonstrating how increased iterations capture a more comprehensive range of outcomes and improve the reliability of results. Finally, we will apply Monte Carlo simulations to continuous data, showcasing their flexibility and applicability to a broader set of problems. This progression will not only highlight the versatility and power of Monte Carlo simulations but also equip you with the skills to apply them effectively across various scenarios.

9.3 Hands-on approach

To demonstrate the practical application of Monte Carlo simulations, we begin with a relatable example: optimizing staffing levels for a small call center. Staffing is a critical operational challenge: too few employees lead to service-level violations—such as excessive wait times or dropped calls—whereas overstaffing results in idle workers and wasted payroll expenses. The inherent variability in daily employee attendance and call volume makes deterministic models inadequate. Monte Carlo simulations, by incorporating randomness and exploring a range of scenarios, offer a robust solution for balancing competing risks.

In this example, we will focus on modeling the daily variability in staffing caused by employee absenteeism. Specifically, we will use discrete random variables, where the outcomes (e.g., the number of absent employees) take on countable, distinct values. This is a practical starting point, as working with discrete data simplifies the simulation process while laying the groundwork for more advanced applications with continuous data.

Our approach involves manually performing a small number of Monte Carlo simulations to illustrate the fundamental steps and concepts. By starting with a manageable number of trials, you can see how each simulation contributes to a larger picture of potential outcomes. This hands-on process ensures that you understand the relationship between random sampling, probability distributions, and the resulting insights. It also reinforces a key modeling principle: always verify how the process behaves with known inputs before automating, as even flawed or incomplete data can produce seemingly valid outputs. In this scenario, the call center manager aims to determine the optimal level of overstaffing needed to maintain service standards while minimizing idle time.

9.3.1 Establishing a probability distribution (step 1)

The first step in performing a Monte Carlo simulation is to define a probability distribution for the uncertain variable in question. For our call center example, the variable of interest is the number of absentee employees on a given day. Based on historical data, the manager observes that absenteeism closely follows a Poisson distribution (see chapter 3).

The Poisson distribution is particularly suited for modeling count-based data, like the number of absentees per day, where events occur independently and the average rate of occurrence, denoted as lambda (λ), remains constant. Here, the average absentee rate is set to two employees per day. Using this distribution, we can calculate the probabilities of observing zero, one, two, or more absentees, providing a realistic foundation for staffing decisions.

Our first snippet of Python code calculates these probabilities and organizes them into a pandas data frame for easy interpretation, where `k` represents the number of absentee employees and `Probability (%)` represents the respective probabilities:

```
>>> import numpy as np
>>> import pandas as pd
>>> from scipy.stats import poisson
>>> lam = 2
>>> k_values = np.arange(0, 9)
>>> probabilities = poisson.pmf(k_values, lam)
>>> probabilities_percentage = \
>>>     np.round(probabilities * 100, 2)
>>> poisson_table = pd.DataFrame({'k': k_values, \
>>>   'Probability (%)': probabilities_percentage})
>>> print(poisson_table)
   k  Probability (%)
0  0            13.53
1  1            27.07
2  2            27.07
3  3            18.04
4  4             9.02
5  5             3.61
6  6             1.20
7  7             0.34
8  8             0.09
```

Imports the numpy library

Imports the pandas library

Imports the poisson module from the scipy library

Sets the rate parameter for a Poisson distribution to 2, indicating an average of two absentee employees per day

Generates an array containing integers, or discrete random variables, from 0 to 8, inclusive

Calculates the Poisson probabilities for each random variable using the given rate parameter

Converts the Poisson probabilities to percentages and rounds the results to two decimal places

Creates a data frame with two columns; contains probabilities for each random variable

Prints the data frame

Of course, it's often helpful to display the same data in graphical format. The Matplotlib bar plot in figure 9.1 shows the percentage probabilities for each discrete random variable in the `poisson_table` data frame.

The displayed distribution represents the Poisson probabilities for the number of absentee employees (k) in a call center, based on a rate parameter (λ) of 2. According to the distribution, the likelihood of zero absentees is 13.53%, whereas the probabilities of having one or two absentees are each 27.07%. Beyond this point, the probabilities decrease steadily as the number of absentees increases. This right-skewed, or positively skewed, distribution emphasizes that smaller numbers of absentee employees are more likely, reflecting the expected daily average of two absentees. Such a pattern provides a realistic foundation for simulating and planning staffing needs in the face of variability.

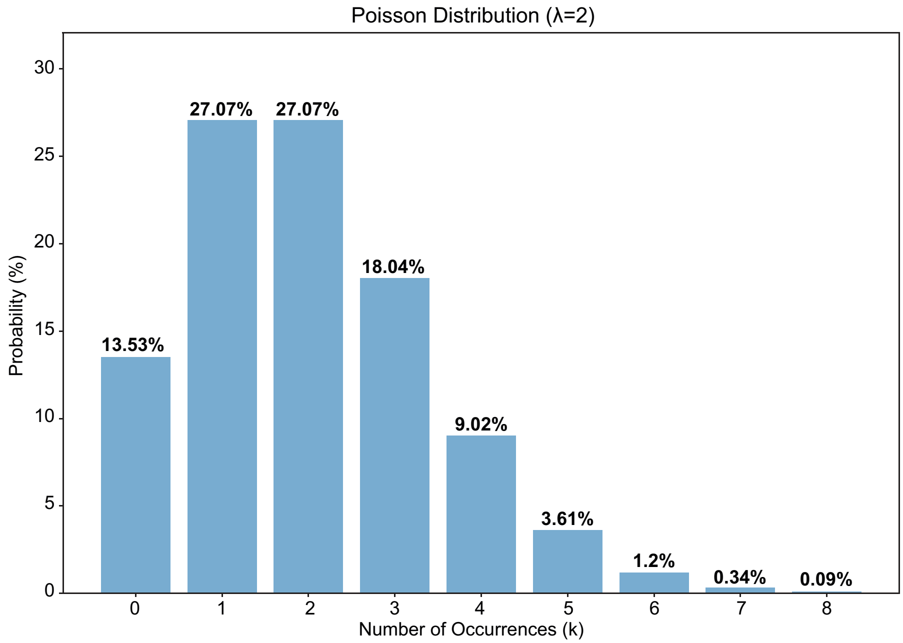

Figure 9.1 A Poisson distribution with a rate parameter (λ) of 2. At lower rate parameters, the distribution is right-skewed, indicating that smaller outcomes are more likely, whereas larger outcomes become increasingly rare.

9.3.2 Computing a cumulative probability distribution (step 2)

The second step in performing a Monte Carlo simulation involves computing a cumulative probability distribution. This distribution provides a cumulative view of the likelihood that the random variable will take a value less than or equal to a specific outcome. For instance, in the context of our call center example, the cumulative probability of having up to two absentee employees includes the probabilities for zero, one, and two absentees summed together. Or, put differently, the cumulative probability at a given point *k* is the sum of the probabilities of all outcomes from 0 to *k*.

Although individual probabilities are useful for understanding the likelihood of specific outcomes, cumulative probabilities allow us to map random numbers to outcomes efficiently, a critical step in generating random scenarios. By summing the probabilities for all outcomes up to a given value, we can establish intervals that align with the cumulative probabilities, thereby simplifying the simulation process.

To manually compute a cumulative probability distribution, we do the following:

1. Start with the individual probabilities for each outcome.
2. Add each probability to the total of all preceding probabilities to compute the cumulative probability for that outcome.
3. Repeat this process for all outcomes until the cumulative probability for the final value equals 1 (or 100%).

With respect to our Poisson example, where λ equals 2,

- The probability of zero absentee employees is 13.53%.
- The cumulative probability of having one or fewer absentee employees is the sum of 13.53% and 27.07%, or 40.60%.
- The cumulative probability of having two or fewer absentee employees is the sum of 40.60% and 27.07%, or 67.67%, and so on.

We can get the same results programmatically by making a call to the `poisson.cdf()` method. The following snippet of code calculates the cumulative probabilities for each value of `k` and stores the results in a column called `Cumulative Probability (%)` appended to the end of `poisson_table`:

```
>>> cumulative_probabilities = \
>>>     poisson.cdf(k_values, lam)
>>> cumulative_percentage = \
>>>  np.round(cumulative_probabilities * 100, 2)
>>> poisson_table = pd.DataFrame({'k': k_values, \
>>>                     'Probability (%)': probabilities_percentage, \
>>>                     'Cumulative Probability (%)': \
>>>                 cumulative_percentage})
>>> print(poisson_table)
   k  Probability (%)  Cumulative Probability (%)
0  0            13.53                       13.53
1  1            27.07                       40.60
2  2            27.07                       67.67
3  3            18.04                       85.71
4  4             9.02                       94.73
5  5             3.61                       98.34
6  6             1.20                       99.55
7  7             0.34                       99.89
8  8             0.09                       99.98
```

Calculates the cumulative Poisson probabilities for each random variable

Converts the probabilities to percentages and rounds the results to two decimal places

Returns a data frame called poisson_table that now includes three, rather than two, columns

Prints the data frame

Cumulative probabilities are often best visualized using a bar line plot (see figure 9.2). The bars represent the Poisson probabilities for each discrete random variable, whereas the line with circular markers represents their cumulative probabilities. The bars correspond to the primary *y* axis on the left, whereas the line corresponds to the secondary *y* axis on the right. The cumulative probability curve starts at 0% for the smallest outcome (`k = 0`) and gradually approaches 100% as larger outcomes are included. For example, the cumulative probability at `k = 3` is 85.71%, meaning there is an 85.71% chance of having three or fewer absentees on any given day.

The cumulative distribution provides the foundation for the next step in the simulation process: mapping random numbers to specific outcomes. By establishing cumulative intervals, we can efficiently assign random values to the corresponding outcomes, ensuring that the generated scenarios reflect the defined probabilities. This step enhances the realism and reliability of the Monte Carlo simulation by accurately representing the variability inherent in the system.

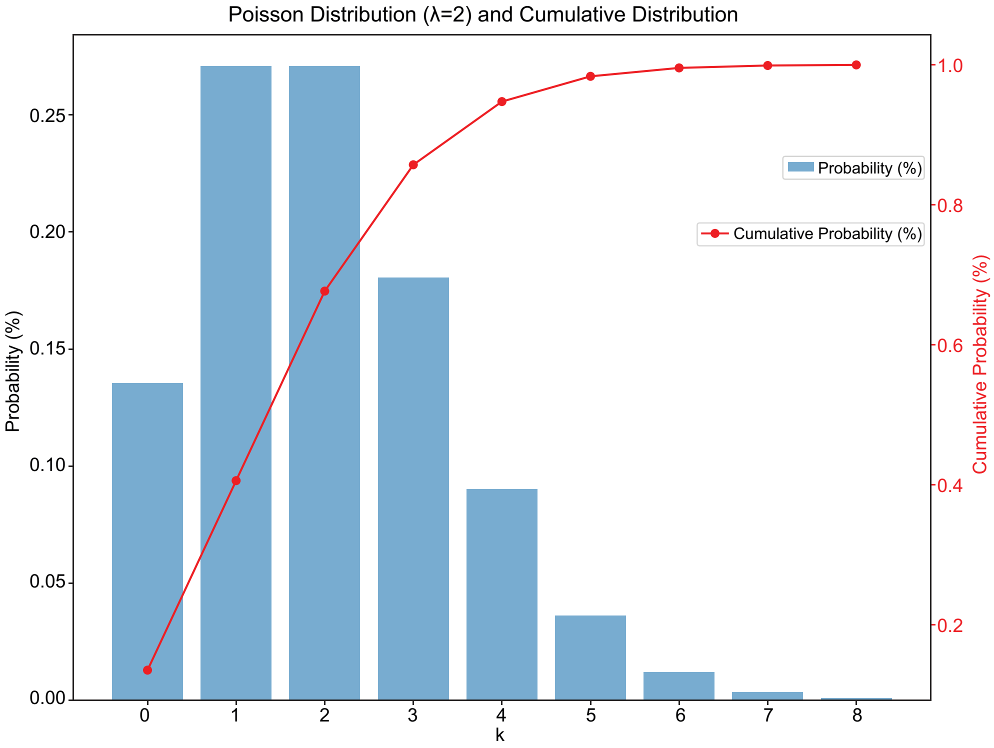

Figure 9.2 A Poisson distribution and its cumulative distribution, with the bars representing the Poisson probabilities and the line representing the cumulative probabilities. The bars correspond to the primary *y* axis on the left, whereas the line with circular markers corresponds to the secondary *y* axis on the right.

9.3.3 Establishing an interval of random numbers for each variable (step 3)

The third step in performing a Monte Carlo simulation is translating the probability distributions of discrete random variables into intervals of random numbers. These intervals provide a practical way to simulate outcomes by mapping random numbers to specific values based on their probabilities.

To achieve this, a list of random digits from 00 to 99 is generated to represent all possible outcomes in the simulation. Each discrete random variable (k) is then assigned a portion of these random digits proportional to its probability. For example, if a particular k value has a probability of 27%, it will be assigned approximately 27 random digits from the list.

The assignment process begins at the start of the list and proceeds sequentially. The number of random digits allocated to each k value is determined based on its probability percentage, ensuring that every k value receives at least one digit. These digits are grouped into ranges, which are then assigned to the corresponding k values.

If the range reaches the end of the list, it wraps back to the beginning to ensure all 100 digits are used effectively. This process provides a clear mechanism for generating outcomes that align with the probability distribution.

The following snippet of Python code performs this assignment programmatically:

```
>>> random_digits = [f'{i:02}' for i in range(0, 100)]
>>> random_digits_ranges = []
>>> current_index = 0
>>> total_digits = len(random_digits)
>>> for i, row in poisson_table.iterrows():
>>>     count = max(1, int(row['Probability (%)']))
>>>     end_index = (current_index + count - 1) % total_digits
>>>     if current_index <= end_index:
>>>         start = random_digits[current_index]
>>>         end = random_digits[end_index]
>>>         random_digits_ranges.append(f’{start}-‘ \
>>>                                     f’{end}’)
>>>     else:
>>>         start = random_digits[current_index]
>>>         end = random_digits[end_index]
>>>         random_digits_ranges.append(f’{start}-99, ‘ \
>>>                                     f’00-{end}')
>>>     current_index = \
>>>         (end_index + 1) % total_digits
>>> poisson_table['Random Digits'] = \
>>>.    random_digits_ranges
>>> print(poisson_table)
```

Generates a list of formatted strings representing the numbers 00 through 99

Initializes an empty list named random_digits_ranges

Sets the starting index for the assignment process

Calculates the total number of random digits available

Iterates through each row in the poisson_ table data frame

Ensures that at least one random digit is assigned to each discrete random variable

Calculates the end index for the current k value’s range

Checks whether the range wraps around to the start of the list

Assigns the start of the range when no wrapping is needed

Assigns the end of the range when no wrapping is needed

Appends the range to random_digits_ranges

Handles cases where wrapping occurs

Assigns the start of the range

Assigns the end of the range

Appends the wrapped range

Updates the current index for the next iteration

Assigns the random digit ranges to a new poisson_table column (Random Digits)

Prints the data frame

```
   k  Probability (%)  Cumulative Probability (%) Random Digits
0  0            13.53                       13.53         00-12
1  1            27.07                       40.60         13-39
2  2            27.07                       67.67         40-66
3  3            18.04                       85.71         67-84
4  4             9.02                       94.73         85-93
5  5             3.61                       98.34         94-96
6  6             1.20                       99.55         97-97
7  7             0.34                       99.89         98-98
8  8             0.09                       99.98         99-99
```

This method ensures that random sampling aligns closely with the defined probability distribution. For instance,

- The `k = 1` value, with a probability of 27.07%, is assigned a range of random digits from `13` to `39`. This range encompasses 27 digits, reflecting the higher likelihood of this outcome.

- Conversely, `k = 6`, with a probability of just 1.20%, is assigned a single digit, `97`, illustrating its lower probability.

By creating these intervals, we effectively bridge the gap between abstract probability distributions and the practical process of generating random outcomes. This structured approach forms the backbone of Monte Carlo simulations, ensuring that the generated scenarios accurately represent the underlying probabilities. As we move to the next step, these intervals will be used to simulate numerous scenarios, enhancing our understanding of potential outcomes and supporting data-driven decision-making.

9.3.4 Generating random numbers (step 4)

The fourth step in a Monte Carlo simulation involves generating random numbers, which serve as the foundation for simulating real-world scenarios. These random numbers will be mapped to the intervals established in the previous step to generate corresponding outcomes. Random numbers between 00 and 99 can be generated using various methods, ranging from traditional random digit tables to modern programming techniques.

In Python, random number generation is both flexible and efficient, offering tools to automate the process. Instead of relying on manual lookup tables, we can programmatically generate a sequence of random digits tailored to the requirements of our simulation. Additionally, setting a seed ensures reproducibility, allowing the same sequence of random numbers to be generated in future runs—a critical feature for testing and validation.

The following code demonstrates how to generate 100 random digits and randomly select 10 of them, formatted as two-digit strings:

```
>>> np.random.seed(1)
>>> random_digits = \
>>>     np.random.choice(100, 10, replace = False)
>>> formatted_digits = \
>>>     [f'{digit:02}' for digit in random_digits]
```

Sets a random seed to ensure that the same random digits are generated across multiple runs, thereby enabling reproducibility

Randomly selects 10 digits from the range 00 to 99 without replacement, ensuring that no digit is repeated in this selection

Formats each digit as a two-digit string (e.g., 00, 24) to maintain consistency with earlier steps

If the simulation requires more iterations, such as 500 random numbers, we can modify the code to sample with replacement, ensuring sufficient variability, like so:

```
>>> random_digits = np.random.choice(100, 500, replace = True)
```

In this case, allowing replacement ensures that digits can be selected multiple times, a necessity for larger simulations that require independent random samples.

Generating random numbers programmatically eliminates manual effort and reduces the chance of human error. It also ensures that the random digits align with the intervals defined in step 3, allowing seamless integration into the simulation process. By using Python's random number generation tools, we can efficiently create inputs for any Monte Carlo simulation or similar analysis.

9.3.5 Simulating a series of trials (step 5)

The fifth step in performing a Monte Carlo simulation involves running a series of trials to generate a distribution of outcomes. Each trial maps a randomly selected digit to its corresponding `k` value, based on the intervals established earlier. This step demonstrates how randomness in the simulation leads to variability in outcomes, offering insights beyond those provided by deterministic models.

To illustrate, suppose one of the randomly selected digits is 73. Based on the mapping established in step 3, this digit falls into the range assigned to `k = 3`. Although the expected value based on the rate parameter (λ) is two absentee employees, this trial suggests the call center manager should prepare for three absentee employees instead. This single trial highlights how Monte Carlo simulations account for the randomness inherent in real-world scenarios and provide alternative staffing recommendations compared to deterministic models.

This is just one outcome from a single trial. In this example, we are running 10 simulations, each using one of the randomly selected digits from the previous step. Each digit is mapped to a corresponding k value, generating a distribution of possible outcomes. These outcomes provide the call center manager with a range of staffing scenarios, allowing them to better understand variability and make more informed decisions. By analyzing this range, the manager can balance the risk of understaffing with the cost implications of overstaffing.

The following line of code displays the list of formatted random digits generated earlier. These digits are used to simulate outcomes in the trials:

```
>>> print('Random Digits:', formatted_digits)
Random Digits: ['80', '84', '33', '81', '93', '17', '36', '82', '69', '65']
```

Table 9.1 shows how each random digit maps to its corresponding `k` value based on the Poisson distribution defined earlier. Each `k` value represents the number of additional associates required for the given trial. Summing these `k` values and calculating their mean provides an estimate of the average number of additional staff needed across all simulations.

Table 9.1 Random numbers aligned to `k` values

Random number	`k` value
80	3
84	3
33	1
81	3
93	4
17	1
36	1

Table 9.1 Random numbers aligned to k values *(continued)*

Random number	k value
82	3
69	3
65	2
Sum	24
Mean	2.4

The calculated mean (2.4) provides the call center manager with a realistic estimate of the average number of additional staff needed, accounting for variability in absenteeism. This approach goes beyond deterministic averages by considering a range of possible scenarios and their associated probabilities.

How should the call center manager use these results in making staffing decisions? By simulating multiple scenarios, the manager gains insights into the potential variability in staffing needs. For example,

- *Understaffing risks*—If the mean suggests 2.4 additional staff, the manager might choose to overstaff slightly to minimize service disruptions during peak absenteeism periods.
- *Cost optimization*—Conversely, if the simulations consistently indicate lower absenteeism, the manager might avoid unnecessary overstaffing, thereby reducing labor costs.

This process highlights how Monte Carlo simulations empower decision-makers to plan more effectively in uncertain environments, balancing risks and costs to achieve operational efficiency.

9.3.6 Analyzing the results (step 6)

Based on the results of the Monte Carlo simulations, the call center manager can determine an optimal staffing strategy by weighing cost efficiency against service levels:

- *Cost efficiency*—If the primary concern is managing payroll costs, the simulations suggest overstaffing by an average of two employees. This approach balances adequate staffing with minimizing unnecessary expenses.
- *Service levels*—Conversely, if the manager prioritizes maintaining high service levels and minimizing risks like service disruptions, customer complaints, or churn, overstaffing by three employees is more prudent. This recommendation provides a buffer to accommodate variability in staff attendance, ensuring operations remain smooth even on days with above-average absenteeism.

By relying on Monte Carlo simulation results instead of solely on the historical rate parameter (λ), the manager can make more informed and nuanced decisions that align with the organization's priorities.

EXPECTED VALUE CALCULATION

The results of the Monte Carlo simulation not only differ from the historical rate parameter ($\lambda = 2$) but also deviate slightly from the expected staffing needs, which can be calculated as the expected value:

$$\text{Expected Overstaff} = \sum (\text{Probability of } k) \times (k)$$

or

$$\begin{aligned}\text{Expected Overstaff} = {} & (0.1353)\,(0) + (0.2707)\,(1) + (0.2707)\,(2) \\ & + (0.1804)\,(3) + (0.0902)\,(4) + (0.0361)\,(5) \\ & + (0.0120)\,(6) + (0.0034)\,(7) + (0.0009)\,(8)\end{aligned}$$

This equals 1.9976. Interestingly, this value closely matches the rate parameter ($\lambda = 2$), suggesting a relationship between the two. However, Monte Carlo simulations capture variability and uncertainty, offering additional insights that deterministic calculations cannot provide.

It is essential to recognize the limitations of small-scale simulations. With only 10 simulations, the results may not fully represent the system's true variability. Anomalous outcomes can arise, skewing the conclusions. Larger numbers of simulations tend to normalize over successive trials, providing more consistent and reliable results. This principle underscores the importance of running sufficient simulations to capture the full range of potential scenarios.

To avoid drawing conclusions from potentially skewed data, the next section demonstrates how to automatically run 500 Monte Carlo simulations using the same data and random digit assignments. By using Python, we will efficiently generate a more extensive set of results, providing a clearer and more dependable understanding of staffing requirements.

9.4 Automating simulations on discrete data

Automating the process to run 500 Monte Carlo simulations on the same discrete data set allows us to follow the same fundamental steps demonstrated earlier but with significant scalability. These steps include defining the probability distribution, computing the cumulative distribution, establishing intervals for random variables, generating random numbers, simulating a series of trials, and analyzing the results. The primary advantage of automation is the ability to substantially increase the number of trials, providing more reliable and robust insights.

By increasing the number of trials from 10 to 500, we enhance the statistical power of our simulations. A larger sample size smooths out anomalies that might appear in smaller trials, yielding a clearer and more accurate representation of the underlying probability distribution. This greater precision allows us to better understand the variability and uncertainty within the system, enabling more informed and reliable decision-making.

The following code snippet demonstrates how to automate 500 Monte Carlo simulations. It ensures reproducibility by setting a seed for the random number generator, normalizes the probabilities to sum to exactly `1`, and calculates the frequency and percentage of each outcome:

```
>>> np.random.seed(1)
>>> num_trials = 500
>>> normalized_probabilities = \
>>>     probabilities / probabilities.sum()
>>> simulated_values = np.random.choice(k_values,
>>>                          size = num_trials,
>>>                          p = normalized_probabilities)
>>> unique, counts = \
>>>     np.unique(simulated_values,
>>>               return_counts = True)
>>> simulated_counts = \
>>>     dict(zip(unique, counts))
>>> simulated_percentages = \
>>>     {k: (v / num_trials) * 100 for k, \
>>>     v in simulated_counts.items()}
```

Sets a seed to ensure reproducibility of the random numbers generated across runs

Specifies the number of Monte Carlo simulations to be run (500 trials in this case)

Normalizes the probabilities so they sum to exactly 1, ensuring valid inputs for sampling

Samples values from the array k_values, representing the possible outcomes of the random variable

Specifies the total number of samples to draw (500)

Applies the normalized probabilities to the sample according to the Poisson distribution

Returns an array of unique outcomes observed in the simulations

Counts the occurrences of each unique outcome

Creates a dictionary mapping unique outcomes to their counts

Converts the counts into percentages of the total number of trials

A few clarifications are in order:

- The process can easily scale to 1,000 or even 10,000 simulations by simply adjusting the `num_trials` parameter. However, increasing the number of trials beyond a certain point offers diminishing returns, as results stabilize when the sample size becomes sufficiently large.
- Ensuring that probabilities sum to exactly 1 is essential in Monte Carlo simulations. Initially, the probabilities for `k > 8` were excluded due to their negligible values in a Poisson distribution with a rate parameter of 2. Normalizing the probabilities accounts for these omissions, providing an accurate foundation for simulations. For example, the probability of `k = 0` increased slightly from 13.53% to 13.54%, and the probability of `k = 3` increased from 18.04% to 18.05% after normalization.
- Despite running 500 simulations, no outcomes resulted in `k = 8`, consistent with the Poisson distribution's low probability for higher values when $\lambda = 2$. This highlights the robustness of the simulations in capturing more likely outcomes while demonstrating the rarity of extreme values.

The automated simulations provide a detailed distribution of results, which can be further analyzed to gain deeper insights. For example, plotting the outcomes of the 500 Monte Carlo simulations enables us to visually assess the distribution and identify patterns in the data. This analysis offers valuable input for decision-making, helping to address variability and uncertainty in real-world scenarios.

9.4.1 Plotting and analyzing the results

The resulting plot in figure 9.3 shows the outcomes of 500 Monte Carlo simulations for a Poisson distribution with a rate parameter (λ) of 2 (see figure 9.3). Each bar represents the percentage of trials that resulted in a specific k value, where k ranges from 0 to 7. The *y* axis represents the percentage of trials corresponding to each k value, and the *x* axis lists the k values. Labels atop each bar display the percentage values, making the frequencies clear and easy to interpret. This visualization highlights the relative likelihood of each outcome, aligning closely with the underlying theoretical distribution while reflecting the variability introduced by random sampling.

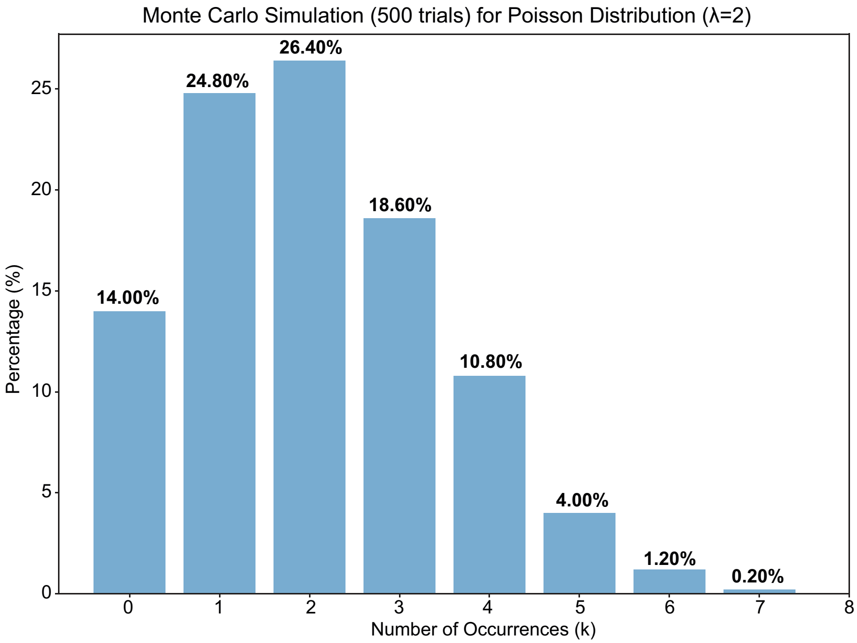

Figure 9.3 Probability distribution from 500 Monte Carlo simulations closely resembling a theoretical Poisson distribution with a rate parameter of 2. The labels atop the bars represent the percentage of trials that resulted in each unique random variable between 0 and 7. No trials result in a k value of 8, consistent with the low theoretical probability of such an outcome.

The results of the 500 simulations show a distribution of k values ranging from 0 to 7, with the percentage frequencies for each value clearly illustrated. When compared to a theoretical Poisson distribution with $\lambda = 2$, the simulated results align closely. As expected, slight variations occur due to the inherent randomness of the simulation, but these differences are minimal and well within acceptable bounds. Otherwise, none of our 500 simulations resulted in k = 8, reflecting the near-zero probability of this outcome in the Poisson distribution.

The call center manager now faces a decision: whether to rely solely on the rate parameter and assume an overstaffing need of two or to consider the Monte Carlo results. The simulation suggests that `k = 3` or `k = 4` may be slightly more likely than predicted by the theoretical Poisson distribution, potentially supporting a decision to overstaff by three instead of two. This choice would mitigate the risks of understaffing—such as service disruptions, delayed responses, and customer dissatisfaction—but at the cost of additional payroll expenditures.

Monte Carlo simulations offer several advantages that make them a robust alternative to purely theoretical approaches. These benefits include the following:

- *Realistic modeling of uncertainty*—Monte Carlo simulations incorporate randomness and variability directly into the modeling process. This allows for a more realistic representation of real-world scenarios, capturing the inherent uncertainty and fluctuations that may not be fully captured by theoretical distributions.
- *Flexibility and adaptability*—Monte Carlo simulations can be adapted to a wide range of problems and conditions. They can model complex systems with multiple variables and dependencies, providing insights into scenarios that are difficult or impossible to address using purely theoretical approaches.
- *Handling of non-normal distributions*—Many real-world phenomena do not follow standard distributions like the Poisson or normal distribution. Monte Carlo simulations can accommodate any distribution, whether it is normal, skewed, or entirely irregular, ensuring that the unique characteristics of the data are accurately represented.
- *Data-driven insights*—Although theoretical distributions are often based on historical observations, they can become outdated or fail to reflect recent changes. Monte Carlo simulations use current data and can be continually updated to provide the most relevant and timely insights.
- *Robustness to outliers*—Real-world data often contains outliers or anomalies that can skew theoretical models. Monte Carlo simulations, through repeated random sampling, can mitigate the effect of these outliers, leading to more reliable and robust conclusions.
- *Scenario analysis*—Monte Carlo simulations allow for the exploration of a wide range of potential outcomes by running numerous scenarios. This helps in understanding the full spectrum of possible results, identifying worst-case scenarios, and planning for contingencies.
- *Quantification of risk*—By simulating a large number of trials, Monte Carlo simulations provide a detailed probability distribution of outcomes. This quantification of risk helps in making more informed decisions by highlighting the likelihood and potential effects of different events.
- *Better decision support*—The ability to simulate various scenarios and analyze their outcomes provides decision-makers with a comprehensive view of potential risks and rewards. This leads to more informed and confident decision-making, as it is based on a broader range of data and scenarios.

This example demonstrates how Monte Carlo simulations complement theoretical distributions by providing a practical framework for decision-making under uncertainty. By simulating a wide range of possible scenarios, the call center manager can make informed choices that balance cost efficiency with service quality. These advantages highlight the versatility of Monte Carlo methods in handling both discrete and continuous data, as we will explore in the next section.

9.5 Automating simulations on continuous data

As we transition from working with illustrative discrete data to real continuous data, we will explore one of the most popular applications of Monte Carlo simulations: predicting future stock prices. Previously, we demonstrated how to fit time series models to forecast future closing prices. Now, we will use Monte Carlo methods to simulate a range of potential outcomes, providing a probability distribution of future prices.

Monte Carlo simulations provide a robust approach to predicting stock prices by incorporating the inherent uncertainty and variability of financial markets. Instead of relying solely on historical trends and deterministic models, Monte Carlo methods use random sampling to generate a range of possible future outcomes. This approach provides a deeper understanding of potential price movements and helps assess the risk and uncertainty associated with various investment strategies.

We will demonstrate how to apply Monte Carlo simulations to forecast future closing prices of a stock. By using real historical stock data, defining the necessary parameters, and running numerous simulations, we will generate a probability distribution of future prices. This process highlights the flexibility and power of Monte Carlo methods in handling real-world continuous data, particularly in quantifying risk and uncertainty. Through this demonstration, we will gain a comprehensive view of potential outcomes, enhancing our ability to make informed and strategic investment decisions.

9.5.1 Predicting stock prices with Monte Carlo simulations

When working with continuous data, such as predicting closing stock prices using Monte Carlo simulations, the process involves several distinct steps compared to working with discrete data. Following is a step-by-step breakdown of how Monte Carlo simulations work for continuous data and how we derive closing prices *per day* for each trial:

1. *Analyzing historical data*—First we gather historical stock price data. This data helps us understand the past behavior of the stock and estimate key statistical parameters, such as the mean return and standard deviation. Previously, when working with discrete data, we used historical outcomes to establish a probability distribution. Here, historical stock prices serve as the foundation for determining future price movements.
2. *Calculating log returns*—We then calculate the daily log returns of the stock. Log returns are preferred because they allow for easier compounding over multiple periods and offer a better statistical representation of price movements. The daily log return is calculated as

$$\text{Log Return} = \ln\left(\frac{P_t}{P_{t-1}}\right)$$

where P_t is the closing price on day t and P_{t-1} is the closing price on the previous day. In contrast to the discrete data approach, where we computed a probability distribution for each random variable, this step focuses on quantifying the daily percentage changes in stock prices.

3. *Computing statistical parameters*—Using the calculated log returns, we determine the mean (μ) and standard deviation (σ) of the daily returns. These statistical parameters model the randomness of future price movements. Previously, when working with discrete data, we used the probability distribution to create proportional intervals of random numbers. Here, μ and σ serve as inputs for generating future random returns.
4. *Generating random daily returns*—For each simulation, instead of generating random numbers and assigning them to intervals as we did for discrete data, we generate random daily returns directly from a normal distribution. This distribution is used because historical log returns usually approximate a normal distribution, and it is characterized by the mean and standard deviation calculated from the historical log returns. This step effectively simulates the randomness of stock price movements, capturing their variability.
5. *Simulating prices*—Starting from the last known closing price, we calculate future prices using the randomly generated daily returns to simulate future prices. The closing price for each day is calculated using the following formula:

   $$P_{t+1} = P_t \times e^r$$

   where e is the base of the natural logarithm, equal to approximately 2.72, and r is the randomly generated log return for that day. This process reflects the compounding nature of stock price returns over time. In contrast to the discrete data approach, where trials were based on random number intervals, this step generates a continuous path for stock prices.
6. *Simulating multiple trials*—We repeat these steps for a large number of trials (e.g., 500 simulations). Each trial represents one potential future trajectory of the stock price over the specified time horizon. By running numerous simulations, we create a robust data set representing a wide range of possible outcomes.
7. *Analyzing the results*—Finally, after completing all the simulations, we analyze the resulting distribution of simulated closing prices. This analysis provides insights into the range of possible future prices, their probabilities, and associated risks. Such analysis aids in making more informed financial decisions by quantifying potential variability and uncertainty in stock price movements.

By automating the process, we can efficiently perform a large number of Monte Carlo simulations, providing more reliable and robust results. This approach allows us to model the inherent randomness of stock price movements, offering valuable insights for financial forecasting and risk management. Monte Carlo simulations are a powerful tool for understanding the range of possible future outcomes and making strategic investment decisions based on data-driven insights.

9.5.2 Analyzing historical data (step 1)

In this first step, we prepare the historical stock data needed for our Monte Carlo simulations. Following a approach similar to that in chapter 7, where we worked with Apple stock data, we will now focus on General Motors (GM), the automaker behind brands such as Chevrolet, Cadillac, GMC, and Buick. This involves retrieving historical stock prices and processing them for use in our simulation. Specifically, we will gather daily closing prices for two periods: July 1, 2023, to December 31, 2023, and January 1, 2024, to January 31, 2024. The data from the latter half of 2023 will serve as the basis for our simulations, and the data from January 2024 will allow us to evaluate the accuracy of our forecasts.

Here's the Python code used to fetch and process the data:

```
>>> import yfinance as yf
>>> ticker_symbol = 'GM'
>>> ticker_data = yf.Ticker(ticker_symbol)
>>> stock_data_23 = ticker_data.history(start = '2023-07-01',
>>>                         end = '2023-12-31',
>>>                         interval = '1d')
>>> stock_data_24 = ticker_data.history(start = '2024-01-01',
>>>                         end = '2024-01-31',
>>>                         interval = '1d')
>>> close_prices_23 = stock_data_23[['Close']]
>>> close_prices_24 = stock_data_24[['Close']]
>>> close_prices_23 = \
>>>     close_prices_23.reset_index(drop = True)
>>> close_prices_24 = \
>>>     close_prices_24.reset_index(drop = True)
```

Imports the yfinance library, used for fetching financial market data from Yahoo Finance

Sets the variable ticker_symbol to the string 'GM', for General Motors

Creates a Ticker object to interact with Yahoo Finance's API for the GM stock

Retrieves historical daily stock data using dates in the ISO 8601 format

Retrieves historical daily stock data; same stock, different start and end dates

Subsets stock_data_23 to include just the Close column and creates a new data frame

Subsets stock_data_24 to include just the Close column and creates a new data frame

Resets the index; same method, different data frame

Resets the index by removing the original and replacing it with a default integer index

We use the `describe()` method to generate key descriptive statistics for both periods. This provides a clear picture of the stock's behavior before running simulations:

```
>>> stats_23 = close_prices_23.describe()
>>> print(stats_23)
            Close
count  126.000000
mean    32.932516
std      3.654504
min     26.425901
```

```
25%     30.057603
50%     32.831583
75%     35.848412
max     40.051197

>>> stats_24 = close_prices_24.describe()
>>> print(stats_24)
           Close
count  20.000000
mean   35.455665
std     0.809152
min    34.387157
25%    35.021101
50%    35.192638
75%    35.804214
max    37.937252
```

The statistical summary for the second half of 2023 (July–December) shows an average closing price of approximately $32.93, with a standard deviation of $3.65, indicating moderate variability. Prices ranged from $26.43 to $40.05. In contrast, the summary for January 2024 reveals a higher average price of $35.46 but a much lower standard deviation of $0.81, indicating greater price stability. The January prices ranged from $34.39 to $37.94.

The historical data from 2023 serves as the foundation for generating random paths of future prices, allowing us to simulate potential price movements. The January 2024 data, on the other hand, acts as a benchmark for evaluating the accuracy of our simulations. The differences in variability between the two periods underscore the importance of incorporating realistic volatility into the Monte Carlo process to account for market dynamics effectively. By establishing this solid data foundation, we are prepared to move forward with the simulation steps, using historical insights to predict and evaluate future stock price behavior.

9.5.3 Calculating log returns (step 2)

The second step in our Monte Carlo simulation involves calculating the daily log returns from the closing prices in the `close_prices_23` data frame. Log returns are preferred in financial modeling because they offer a better statistical representation of relative price changes and are easily compounded over multiple periods.

To compute log returns, we calculate the natural logarithm of the ratio of each day's closing price to the previous day's closing price. This operation therefore captures the proportional change in price while accounting for compounding effects. The computed log returns are then stored in a new column called `Log Return`:

```
>>> close_prices_23['Log Return'] = \
>>>     np.log(close_prices_23['Close'] /
>>>            close_prices_23['Close'].shift(1))
>>> log_returns = close_prices_23['Log Return'].dropna()
```

Additional explanation of this code snippet is in order:

- The `shift(1)` method offsets the `Close` column by one row, aligning each day's closing price with the previous day's closing price for the ratio calculation.
- The `np.log()` function computes the natural logarithm of these ratios to derive the log returns.
- The `dropna()` method removes the missing value generated for the first row (as no prior closing price exists), ensuring a clean series of log returns.

The variable `log_returns` is then passed to a snippet of Matplotlib and seaborn code to create a density plot of the log returns for GM stock between July and December 2023 (see figure 9.4). This plot provides a visual representation of daily price changes during this period.

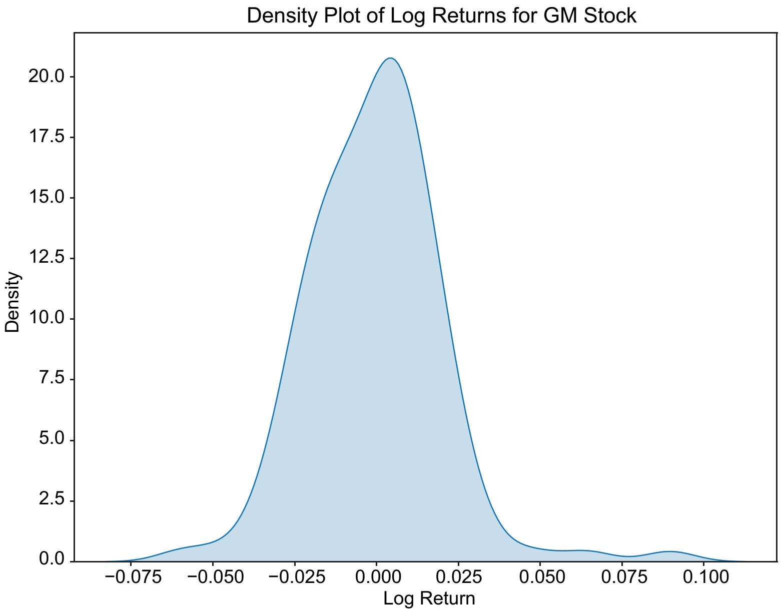

Figure 9.4 Density plot of the log returns for GM stock between July and December 2023. The distribution is approximately normal, centered around a mean close to zero, with most log returns clustered near zero but with some larger positive and negative values in the tails.

The density plot highlights the distribution of daily log returns. Key observations include the following:

- *Normality*—The log returns are approximately normally distributed, with most values clustered near zero. This suggests that daily price changes are small on average, with no extreme skewness in the data.

- *Tails*—The presence of longer tails indicates that although most daily changes are modest, there are occasional larger positive or negative returns. These tails represent days of higher market volatility.
- *Mean and standard deviation*—The plot's central tendency and spread provide the mean and standard deviation of the log returns, which are essential inputs for simulating future price movements.

This analysis of historical log returns serves as a foundational step in the Monte Carlo process. The distribution of log returns, which closely approximates a normal distribution, allows us to make the key assumption that future log returns will follow a similar pattern. This enables us to do the following:

- Generate realistic random samples based on the calculated mean and standard deviation.
- Incorporate the inherent randomness and variability observed in historical stock prices.
- Create simulations that reflect potential real-world scenarios with greater accuracy.

By using these statistical properties, we ensure that our Monte Carlo simulations are robust and provide meaningful insights into the range of possible future stock price movements.

9.5.4 Computing statistical parameters (step 3)

The third step in our Monte Carlo simulation involves calculating the key statistical parameters—that is, the mean and standard deviation—of the log returns. These parameters are critical for modeling future stock price movements as they define the central tendency and variability of daily returns:

- The mean (μ) represents the average daily return and indicates the overall trend of the stock price during the historical period.
- The standard deviation (σ) measures the level of volatility, reflecting the extent to which daily returns deviate from the mean.

The following code calculates and prints these parameters from the `log_returns` data:

```
>>> mu = log_returns.mean()
>>> print(mu)
-0.000605520424324896

>>> sigma = log_returns.std()
>>> print(sigma)
0.020045660849343817
```

We can conclude the following:

- The mean log return is approximately –0.00061, suggesting a slight downward trend in GM's stock price over the analyzed period.

- The standard deviation of 0.02005 indicates a moderate level of daily price fluctuations. This value reflects the stock's historical volatility, a critical factor in forecasting future movements.

These statistical parameters are essential inputs for generating random daily returns in the simulation process. By incorporating the calculated `mu` and `sigma`, we ensure that

- The simulated returns realistically reflect the historical behavior of the stock.
- The randomness in price movements accounts for both the trend (mean) and volatility (standard deviation) observed in the data.

Incorporating these parameters into our Monte Carlo model allows us to generate a wide range of potential outcomes, capturing the inherent uncertainty and variability of stock price movements. This step lays the foundation for creating accurate and reliable forecasts, enabling better decision-making in financial planning and risk assessment.

9.5.5 Generating random daily returns (step 4)

The fourth step in the Monte Carlo process involves defining the parameters for the simulation: the number of simulations to run and the forecast horizon. These parameters are crucial for ensuring the robustness and relevance of the predictions.

The following code snippet sets these two parameters:

```
>>> num_simulations = 500
>>> num_days = 20
```

Let's elaborate:

- *Number of simulations* (`num_simulations`)—Set to 500 to provide a comprehensive range of possible outcomes. Running a high number of simulations ensures that the model accounts for variability and provides robust predictions, reducing the effect of anomalies or outliers on the results. However, running significantly more simulations may lead to diminishing returns, as the results stabilize beyond a certain threshold, offering little additional insight.
- *Number of days* (`num_days`)—Set to 20, representing the actual number of trading days in January 2024. This corresponds to the record count in the `close_prices_24` data frame, aligning the simulation period with the actual stock price data for comparison.

Choosing an appropriate number of simulations and forecast days is vital:

- A higher number of simulations increases the reliability of the results by capturing a wider range of potential outcomes. However, it is essential to balance this with computational efficiency and recognize that beyond a certain point, additional simulations may yield diminishing returns as the results stabilize.
- Setting the forecast horizon to match real-world periods (such as a month of trading days) ensures that the predictions are relevant and actionable for decision-making.

With these parameters defined, we are ready to simulate 500 potential stock price trajectories over the 20 trading days in January 2024, providing a detailed and realistic forecast for future stock price movements.

9.5.6 Simulating prices (step 5)

The fifth step in the Monte Carlo process involves generating simulated stock prices using the parameters defined in the previous steps: the mean (`mu`), standard deviation (`sigma`), number of simulations (`num_simulations`), and forecast horizon (`num_days`). This step creates a distribution of potential future stock prices by incorporating the randomness and variability observed in historical log returns.

The following code snippet performs the simulation:

Initializes an empty data frame called simulation_df to store the simulated stock price trajectories. Each column will correspond to a single simulation.

Generates a series of random daily returns for each simulation using a normal distribution. The distribution is defined by the historical log returns' mean (mu) and standard deviation (sigma).

```
>>> simulation_df = pd.DataFrame()
>>> for i in range(num_simulations):
>>>     sampled_returns = \
>>>         np.random.normal(mu, sigma, num_days)
>>>     price_list = [last_price_23]
>>>     for r in sampled_returns:
>>>         price_list.append(price_list[-1] * \
>>>                           np.exp(r))
>>>     simulation_df[i] = price_list
```

Initializes a price_list starting with the last closing price of 2023 (last_price_23), which serves as the baseline for the simulated stock prices

Stores the simulated stock prices for each trial in the simulation_df data frame, with each column representing the trajectory of stock prices for one simulation

Iteratively calculates and appends the simulated daily stock prices, using the exponential of the randomly generated log returns

Let's explain further:

- For each simulation, a sequence of daily returns is randomly sampled from a normal distribution defined by the historical mean and standard deviation of log returns. This models the inherent randomness and variability in stock price movements.
- Starting from the last observed price in 2023, the sampled daily returns are compounded iteratively to calculate the stock price for each subsequent day. This compounding process reflects the geometric nature of stock price movements.
- Each simulated trajectory is stored in the data frame, with rows representing days and columns representing individual simulations. The result is a comprehensive view of 500 potential future stock price paths over the forecast period.

The resulting `simulation_df` data frame contains 500 columns, each representing a unique simulation of stock prices over 20 trading days in January 2024. This data frame serves as the foundation for visualizing and analyzing the distribution of potential future prices, enabling a deeper understanding of the range of outcomes and associated probabilities.

This approach uses historical volatility and trends to provide realistic and actionable predictions, helping stakeholders assess potential risks and make informed decisions. In the next section, we will analyze these results using visualization techniques to gain further insights into the simulated price trajectories.

9.5.7 Simulating multiple trials (step 6)

The plot in figure 9.5 displays the results of our 500 Monte Carlo simulations for GM's closing stock price over the 20 trading days in January 2024. Each line represents the trajectory of stock prices for a single simulation trial, providing a visual summary of the range of possible outcomes.

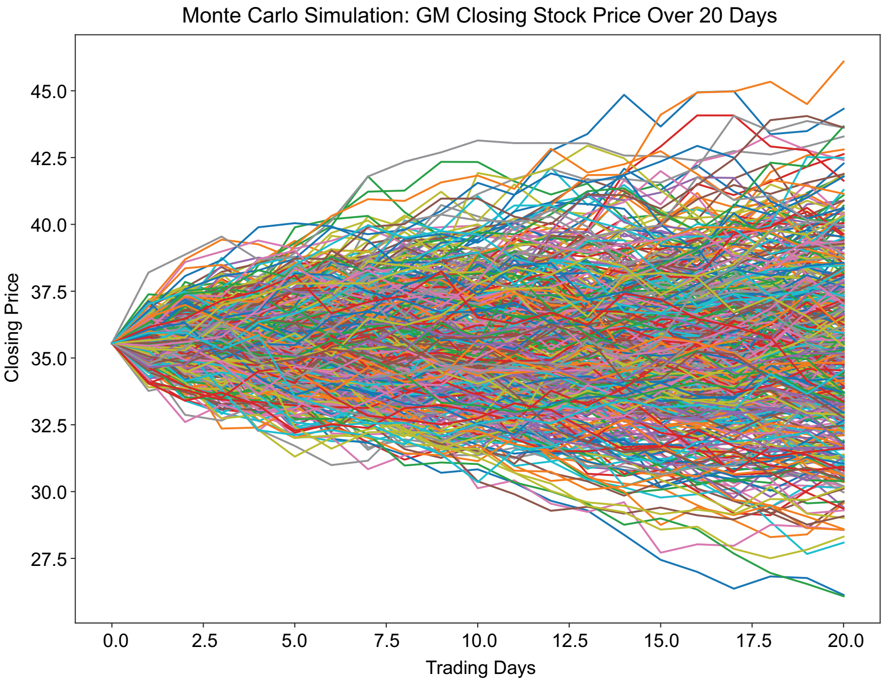

Figure 9.5 500 Monte Carlo simulations of GM's closing stock price for January 2024. Each line corresponds to a single simulation, depicting potential stock price trajectories based on the historical mean and volatility of log returns.

This visualization highlights the variability and uncertainty inherent in stock price movements. Key observations include the following:

- *Clustered trajectories*—Although some simulations show significant increases or decreases in stock price, the majority of trajectories remain clustered around the average trend. This clustering provides an indication of the most probable price range.

- *Divergent scenarios*—The presence of both extreme upward and downward movements in some trials reflects the stochastic nature of financial markets, where unexpected events can lead to substantial deviations.
- *Uncertainty across time*—The increasing spread of trajectories over the 20-day forecast horizon demonstrates how uncertainty accumulates as predictions extend further into the future, which highlights a critical limitation of long-term forecasts, as their utility diminishes due to compounding uncertainty, making them less reliable for precise decision-making. Caution is advised when relying on such forecasts for long-term planning, as they are better suited for understanding general trends rather than exact outcomes.

This plot provides several actionable insights:

- *Risk assessment*—The variability in trajectories can help investors assess the level of risk associated with the stock, identifying both the potential upside and downside.
- *Decision support*—By understanding the range of possible outcomes, decision-makers can plan for best-case and worst-case scenarios, tailoring investment strategies to align with their risk tolerance and objectives.
- *Market behavior*—The clustering of trajectories around the mean provides a visual approximation of the expected price behavior, useful for benchmarking expectations.

Monte Carlo simulations, as shown here, provide a robust framework for exploring potential future price paths and their associated probabilities. This enables a deeper understanding of market dynamics and empowers investors and analysts to make informed financial decisions. In the next section, we will delve further into the analysis of these simulated results, calculating key summary statistics to quantify the range and likelihood of outcomes.

9.5.8 Analyzing the results (step 7)

To analyze the outcomes of our 500 Monte Carlo simulations, we start by comparing the ending prices of the trials to the starting price. This analysis provides a broad view of the potential direction of stock prices and the likelihood of specific outcomes. Specifically, we count the number of simulations where the ending price is greater than or less than the starting price.

The following snippet of code performs this analysis by identifying the starting price and comparing it to the ending prices across all simulations:

```
>>> start_price = simulation_df.iloc[0, 0]
>>> ending_prices = simulation_df.iloc[-1, :]
>>> num_greater = \
>>>     (ending_prices > start_price).sum()
>>> num_less = \
```

Extracts the starting price from the first row and first column of the simulation_df data frame

Extracts the ending prices from the last row of simulation_df

Counts the number of simulations where the ending price is greater than the starting price

```
>>>     (ending_prices < start_price).sum()
>>> print(num_greater)
219
>>> print(num_less)
281
```

Counts the number of simulations where the ending price is less than the starting price

Prints the simulation count where the end of January 2024 stock price is greater than the starting price: equals 219

Prints the simulation count where the end of January 2024 stock price is less than the starting price: equals 281

The results show that 219 simulations (43.8%) resulted in a higher ending price than the starting price, and 281 simulations (56.2%) ended lower. This suggests a slightly higher likelihood of the stock price decreasing over the simulated period. These probabilities provide insights into potential trends and highlight the uncertainty inherent in financial markets.

To gain a deeper understanding of the distribution of simulated stock prices, we calculate summary statistics across all simulations. Flattening the data frame allows us to analyze all simulated prices as a single data set:

```
>>> all_simulations = \
>>>     simulation_df.values.flatten()
>>> overall_stats = pd.DataFrame(all_simulations,
>>>         columns = ['Simulated Prices']).describe()
>>> print(overall_stats)
       Simulated Prices
count      10500.000000
mean          35.559536
std            2.204029
min           25.996207
25%           34.261598
50%           35.691632
75%           36.821081
max           47.983021
```

Flattens the 2D array of simulated prices into a 1D array, combining all simulations into one list

Prints the results

Creates a data frame from the flattened simulation results and computes summary statistics

The descriptive statistics reveal the following:

- *Mean*—Simulated prices average $35.56, closely matching the actual mean price of $35.46 observed in January 2024, which suggests that the simulations effectively capture the central tendency of stock prices.
- *Standard deviation*—Simulated prices have a standard deviation of $2.20, significantly higher than the actual value of $0.81, reflecting the simulations' broader exploration of potential variability in stock prices.
- *Range*—Simulated prices range from $25.99 to $47.98, compared to the observed range of $34.39 to $37.94, indicating the inclusion of extreme scenarios in the simulations.

The Monte Carlo simulations successfully approximate the mean and median of the actual January 2024 prices, reflecting the method's ability to capture central tendencies. However, the higher variability in the simulations highlights the method's purpose: to explore a wide range of potential outcomes, including extreme scenarios, rather than to match observed data precisely.

This variability underscores a pair of key insights:

- *Risk assessment*—The broader range of simulated prices helps investors prepare for worst-case and best-case scenarios, providing a comprehensive view of potential risks and rewards.
- *Decision-making support*—The simulations' clustering around the mean and median aligns with expected trends, offering confidence in planning strategies while acknowledging potential volatility.

Monte Carlo simulations remain a valuable tool for modeling uncertainty and assessing variability in complex systems. Although they may not precisely replicate observed outcomes, their ability to account for randomness and generate a spectrum of scenarios makes them indispensable for risk management and strategic planning. In the next chapter, we will build on this foundation by exploring decision trees to derive expected values and assess alternatives, further enhancing our capacity for data-driven decision-making.

Summary

- Monte Carlo simulations provide a powerful tool for modeling and analyzing complex systems with inherent uncertainty, offering insights into the range of possible outcomes and their probabilities. This chapter demonstrated how to apply Monte Carlo simulations to both discrete and continuous data, highlighting the differences in approach and the specific steps involved in each case.
- For discrete data, such as employee absenteeism, simulations relied on historical frequencies and discrete probability distributions to generate potential scenarios. For continuous data, such as stock price movements, the process involved calculating key statistical parameters like the mean and standard deviation to model the variability of outcomes.
- The simulations illustrated the importance of capturing inherent randomness and variability. Discrete simulations provided clarity in scenarios with distinct outcomes, whereas continuous simulations accounted for the fluidity of real-world processes. The chapter also emphasized the role of visual tools, such as density plots and trajectory graphs, in interpreting simulation results and understanding the implications of variability.
- Running a large number of simulations was highlighted as essential for achieving robust results, smoothing out anomalies, and providing reliable insights. By generating a range of potential future outcomes, Monte Carlo simulations empower decision-makers to assess risks, plan for contingencies, and optimize strategies in uncertain environments.
- The hands-on approach demonstrated in this chapter underscored the versatility of Monte Carlo methods in addressing both discrete and continuous uncertainties. Looking ahead, the next chapter builds on these methods, exploring decision trees to derive expected values and assess alternatives, further enhancing our capacity for data-driven decision-making.