Contents

Week 5: Simulation Modeling

By the end of this week, you will be able to create simulation models by combining algorithmic thinking with probabilistic sampling. In the context of prescriptive analytics, simulation helps decision makers to rigorously quantify the trade-offs between possible decisions in the presence of uncertainties. It is especially important if large-scale experimentation is prohibitively costly.

Session 9: Probabilistic Sampling

Sampling from a Simple Distribution

```
[1]: from numpy.random import default_rng
     rng=default_rng()
     rng.normal(20,10)
```

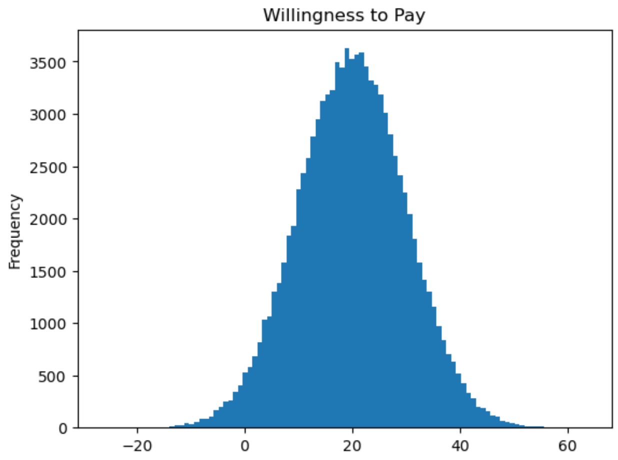
```
13.895520346781248
```

```
[2]: import pandas as pd
     valueList=pd.Series(rng.normal(20,10,size=100000))
     valueList
```

```
0        27.659515
1        24.049746
2        25.167474
3        18.524263
4        10.731987
           ...
99995    17.515881
99996    27.667804
99997    11.621479
99998    12.147279
99999    20.448597
Length: 100000, dtype: float64
```

```
[3]: valueList.plot(kind='hist',bins=100,title='Willingness to Pay')
```

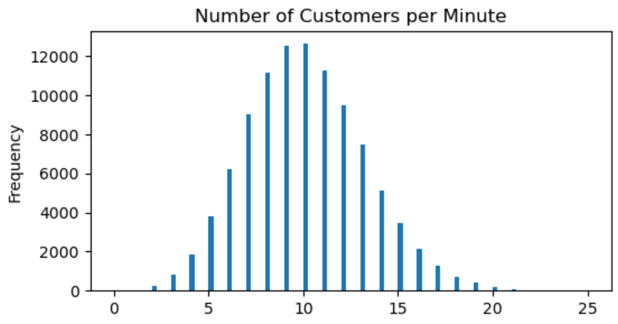
```
<Axes: title={'center': 'Willingness to Pay'}, ylabel='Frequency'>
```



```
[4]: demand=pd.Series(rng.poisson(10,size=100000))
     demand
```

```
0         4
1        12
2         8
3        12
4         8
         ..
99995     8
99996     6
99997    12
99998    15
99999    12
Length: 100000, dtype: int64
```

```
[5]: demand.plot(kind='hist',bins=100,title='Number of Customers per Minute',figsize=(6,3))
     import matplotlib.pyplot as plt
     plt.show()
```



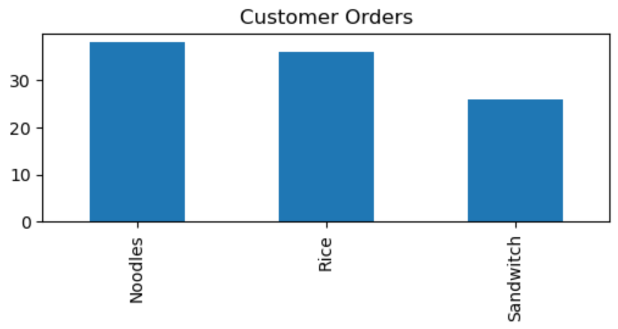
```
[6]: coin=pd.Series(rng.binomial(1,0.5,size=10))
     coin
```

```
0    1
1    0
2    0
3    0
4    0
5    0
6    0
7    0
8    1
9    1
dtype: int64
```

```
[7]: order=pd.Series(rng.choice(['Noodles','Rice','Sandwitch'],p=[0.4,0.3,0.3],size=100))
     order.head()
```

```
0    Noodles
1       Rice
2       Rice
3       Rice
4    Noodles
dtype: object
```

```
[8]: order.value_counts().plot(kind='bar',title='Customer Orders',figsize=(6,2))
     plt.show()
```



```
[9]: # Sampling with replacement
     rng.choice(range(10),size=9)
```

```
array([1, 1, 8, 9, 5, 2, 6, 4, 0])
```

```
[10]: # Sampling without replacement
      rng.choice(range(10),size=9,replace=False)
```

```
array([5, 7, 8, 6, 9, 4, 1, 3, 2])
```

Common Probability Distributions

`rng` function	Parameters	Description
`normal`	`loc`(mean) , `scale` (std)	Bell curve with given mean and standard deviation.
`binomial`	`n`, `p`	# of heads among n independent coin tosses. (Probability of heads is p.)
`uniform`	`low`, `high`	Equally likely to be any decimal number in this range.
`poisson`	`lam` (mean)	# of independent events within a period; each event occurs at a random time.
`geometric`	`p`	# of coin tosses before landing heads. (Probability of heads is p.)

In addition, `rng.choice` can be used to sample from any given list at the given probabilities.

Note on Vectorized Operations

This week, we will often work with a whole array or Series of numbers. It is helpful to learn to use vectorized operations, which operates on the entire list of numbers at the same time.

```
[11]: import pandas as pd
      a=pd.Series([5,3,2.5])
      a
```

```
0    5.0
1    3.0
2    2.5
dtype: float64
```

```
[12]: a*2
```

```
0    10.0
1     6.0
2     5.0
dtype: float64
```

```
[13]: a*2>=6
```

```
0     True
1     True
2    False
dtype: bool
```

```
[14]: (a*2>=6) & (a<5)
```

```
0    False
1     True
2    False
dtype: bool
```

```
[15]: # Number of entries satisfying this condition
      ((a*2>=6) & (a<5)).sum()
```

```
1
```

```
[16]: # Proportion of entries satisfying this condition
      ((a*2>=6) & (a<5)).mean()
```

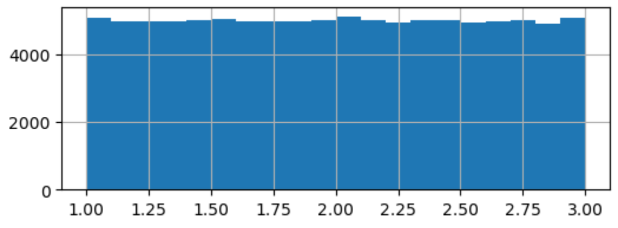
```
0.3333333333333333
```

```
[17]: a[a*2>=6]
```

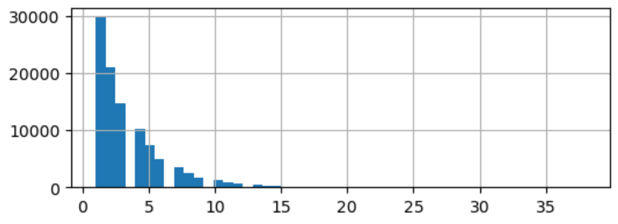
```
0    5.0
1    3.0
dtype: float64
```

Exercise 5.1: Sampling from Simple Distributions

a) Generate 100,000 samples from a uniform distribution between 1 and 3, and plot the histogram with 20 bins, as below.



b) Generate 100,000 samples from a geometric probability with $p = 0.3$, and plot the histogram with 50 bins, as below.



c) Suppose that the number of customers arriving to a grocery store every 10 minutes is Poisson distributed with mean 8.5. Estimate the chance that the number of arrivals in a given 10 minute interval is 15 or higher.

Hint: Generate 1,000,000 samples from the Poisson distribution and calculate the proportion of samples that are at least 15 using vectorized operations.

```
0.027572
```

d) Suppose there are two products, and a customer's willingness to pay for each of the two products are drawn independently from the following distributions:

- Product 1: Normal(25,10)
- Product 2: Normal(20,15)

Estimate the proportion of customers whose willingness to pay is higher for the second product than the first, AND whose willingness to pay for the second product is at least zero.

Hint: Draw 1,000,000 samples from each distribution and compare the two Series using vectorized operations based on the above logic.

```
0.390713
```

Sampling from Complex Distributions using Algorithmic Thinking

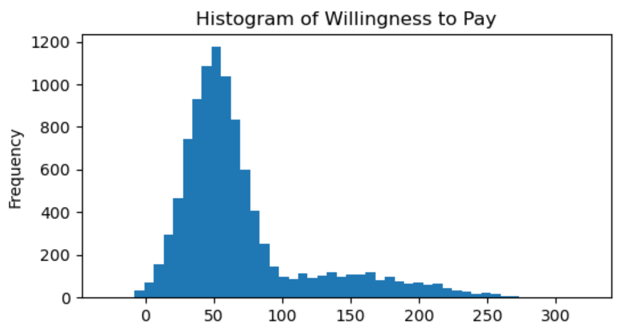
Example 1: Mixture of Distributions

Suppose that there are two market segments, "A" and "B", and the willingness to pay for a product depends on the segment:

- Segment A: Normal(150,50)
- Segment B: Normal(50,20)

Moreover, each customer has a 0.2 chance of being in segment A and 0.8 chance of being in segment B. The following code generates a simulated dataset of the willingness to pay of 10,000 customers. Similar code logic can be used to sample from any mixture of simple distributions.

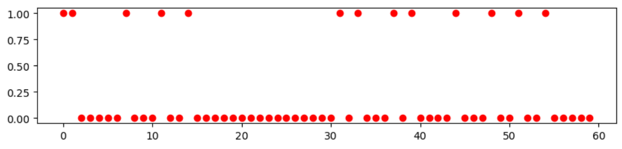
```
[22]: import pandas as pd
      import matplotlib.pyplot as plt
      from numpy.random import default_rng
      rng=default_rng()
      data=[]
      for i in range(10000):
          segment=rng.choice(['A','B'],p=[0.2,0.8])
          if segment=='A':
              data.append(rng.normal(150,50))
          else:
              data.append(rng.normal(50,20))
      valuations=pd.Series(data)
      valuations.plot(kind='hist',bins=50,title='Histogram of Willingness to␣
 ↪Pay',figsize=(6,3))
      plt.show()
```



Example 2: Serial Correlation

Suppose we want to simulate whether it will rain for 60 days. Knowing that the overall proportion of rainy days is 0.2, the first attempt might be as follows.

```
[23]: rain=pd.Series(rng.binomial(1,0.2,size=60))
      rain.plot(style='ro',figsize=(10,2))
      plt.show()
```

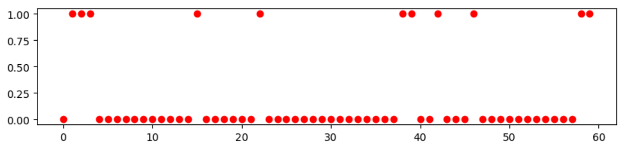


However, it might be more realistic to take into account the correlation in whether between two consecutive days. Suppose that the probability whether it will rain today is:

- 0.6 if it rained yesterday.
- 0.1 if it did not rain yesterday.

The following code takes into account this correlation.

```
[24]: data=[rng.binomial(1,0.2)]
      for i in range(1,60):
          if data[-1]==1:
              data.append(rng.binomial(1,0.6))
          else:
              data.append(rng.binomial(1,0.1))
      rain=pd.Series(data)
      rain.plot(style='ro',figsize=(10,2))
      plt.show()
```



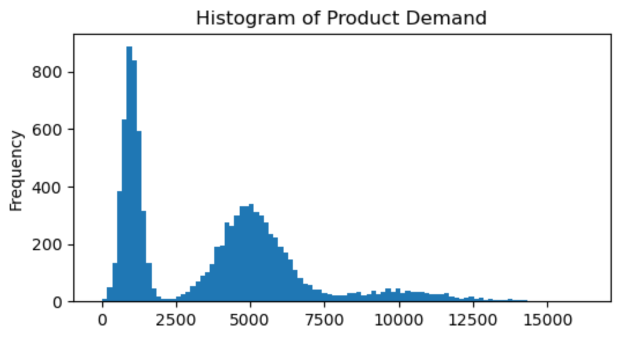
Exercise 5.2: Accounting for Uncertain Product Quality when Forecasting Demand

You are tasked with forecasting demand for a new product. Based on past data and your knowledge of the product, you estimate that the product quality will be Amazing with probability 0.1, Mediocre with probability 0.5, and Terrible with probability 0.4. You model the demand as normally distributed, with mean and standard deviation dependent on the product quality as follows.

Prod. Quality:	Amazing	Mediocre	Terrible
mean	10000	5000	1000
standard deviation	2000	1000	300

Create a `Series` called "forecast" with 10,000 samples of the demand forecast, and compute the mean and standard deviation of the samples, as well as the probability that demand is more than 6000. Finally, plot a histogram of the samples with 100 bins. (For this question, you can ignore the constraint that demand should be integer and non-negative.) Due to randomness, your estimates and plot may be slightly different, but they should be roughly the same as the sample output below.

```
Sample mean: 3875.9049161561243
Sample standard deviation: 2922.1347999073378
Probability demand more than 6000: 0.1771
```



Exercise 5.3: Simulating Serially Correlated Demand

A logistic company would like to simulate demand for a given product. Assume that there are Good or Bad weeks. On a Good week, the demand is Normally distributed with mean 300 and standard deviation 50. On a Bad week, demand is Normally distributed with mean 100 and standard deviation 30. Moreover, you should round the demand to the nearest integer and set it to zero if it is ever negative.

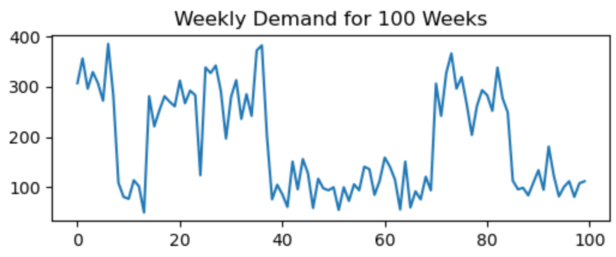
Whether a week is Good or Bad is not independent across time. Conditional on a given week being Good, the next week remains Good with probability 0.95. Similarly, conditional on a given week being Bad, the next week remains Bad with probability 0.95.

Write a function called `generate_demand` with one input argument:

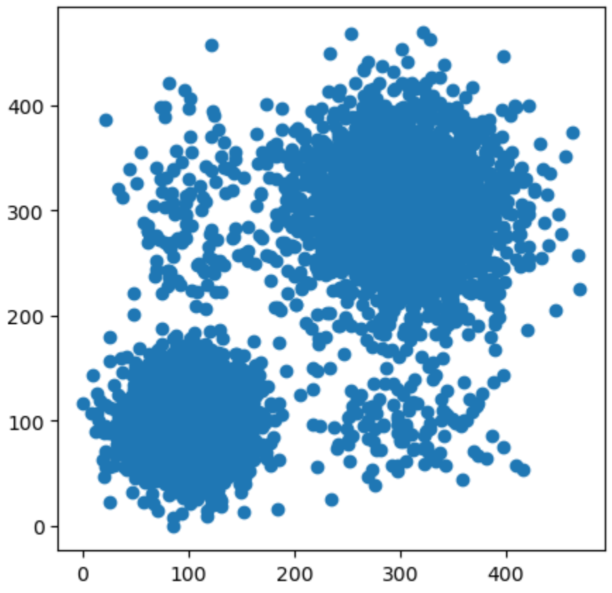
- **n**: the number of weeks to simulate.

The function should return a list of simulated demand for n weeks, assuming that the first week is Good.

```
[28]: # Sanity check. Your plot will be different. Don't worry about reproducing this.
      import pandas as pd
      import matplotlib.pyplot as plt
      s=pd.Series(generate_demand(100))
      s.plot(title='Weekly Demand for 100 Weeks',figsize=(6,2))
      plt.show()
```



```
[29]: # Test code (your plot should look similar)
      import matplotlib.pyplot as plt
      d=generate_demand(5000)
      plt.figure(figsize=(5,5))
      plt.scatter(d[:-1],d[1:])
      plt.show()
```



Exercise 5.4: Simulating Willingness to Pay for Two Products

A firm sells two styles of headphones, which we refer to as Model 0 and Model 1. Based on a clustering analysis using historic data, the firm estimates that customers will come from three segments (A, B or C). In each segment,

- the willingness to pay for Model 0 is normally distributed with mean μ_0 and standard deviation σ_0;
- the willingness to pay for Model 1 is normally distributed with mean μ_1 and standard deviation σ_1, independent from the willingness to pay for Model 0.

The following table summarizes these parameters for each segment, as well as the probability that a randomly drawn customer comes from each of the three segments:

Segment	μ_0	σ_0	μ_1	σ_1	Probability of Segment
A	30	30	70	30	0.1
B	80	20	20	10	0.3
C	-10	20	-10	20	0.6

As in the above table, a randomly chosen customer will be from segment A with 10% probability, segment B with 30% probability and segment C with 60% probability. Segment A customers have high valuations for model 1, while segment B customers have high valuations for model 0. Segment C customers, which make up the majority, do not on average value either products.

Write a function called "generateValuations" with one input argument:

- n: the number of customers to generate.

The function should return a pandas DataFrame representing the simulated valuations of n randomly chosen customers. Each row represents a customer. There are three columns:

- **segment**: The segment of the customer, being "A", "B" or "C".
- **model_0**: The customer's maximum willingness to pay for Model 0.
- **model_1**: The customer's maximum willingness to pay for Model 1.

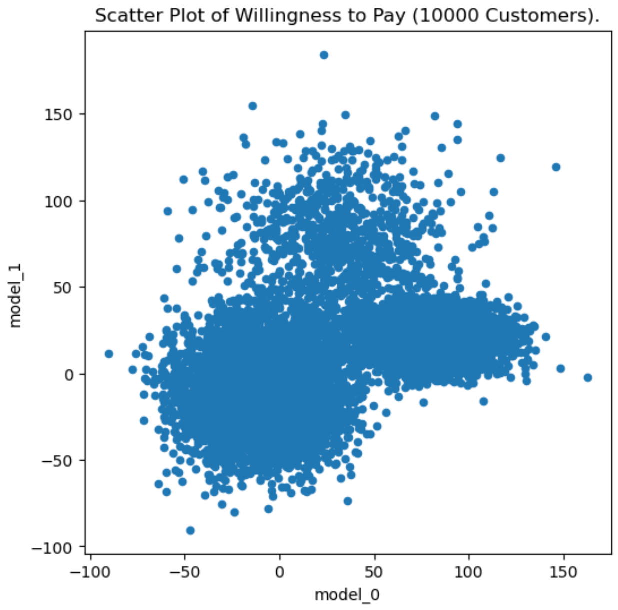
See the sample runs below for examples.

```
[31]: # Showing the format of the generated table. Your numbers will be different.
      generateValuations(6)
```

```
  segment     model_0    model_1
0       B  100.698656  25.029819
1       B   74.571015  25.779851
2       B   80.916389   7.435396
3       C   -7.011228 -10.925133
4       B   63.881735  16.589810
5       C  -33.858308  13.847102
```

```
[32]: # Test code (your plot should look similar)
      generateValuations(10000)\
      .plot(x='model_0',y='model_1',kind='scatter',figsize=(6,6),\
           title='Scatter Plot of Willingness to Pay (10000 Customers).')
```

```
<Axes: title={'center': 'Scatter Plot of Willingness to Pay (10000 Customers).'}, xlabel='model_0',␣
↪ylabel='model_1'>
```



Session 10: Case Study in Epidemic Modeling

Exercise 5.5: Epidemics Modeling

A new virus has broken out in a city and has an incubation period of $d = 5$ days. Starting from $a = 2$ days after infection to the last day of the incubation period, each infected patient has close contact with $n = 4$ uninfected people per day, and infects each of them with probability $p = 0.2$ independently from others. At the end of the last day of incubation, each infected person reports to the hospital and enters isolation, which means that they stop infecting others.

Create a function that simulates the number of patients who report to the hospital at the end of Day 1 through Day m**.** The function should be called `simulateNewCases` and has the following input arguments:

- `m`: the number of days to simulate.
- `initial` (default value 1): the number of individuals who are newly infected on Day 0.
- `a` (default value 2): the first day after infection when a patient becomes contagious.
- `d` (default value 5): the last day after infection when a patient may infect others (not in isolation).
- `n` (default value 4): the number of uninfected individuals a person has close contact with each day.
- `p` (default value 0.2): the probability of infecting each uninfected individual during a close contact.

The function should return a list of m integers, representing the number of infected individuals who exit incubation and report to the hospital at the end of Day 1 through Day m.

To illustrate the timeline. Suppose that a person is infected on Day 5, then the person becomes contagious on Day $5+a = 7$ and starting on that day, has the capacity to infect up to n people per day. At the end Day $5 + d = 10$, after possibly infecting new people on that day, the patient reports to the hospital and enters into isolation.

Solve this problem by applying the four steps of algorithmic thinking.

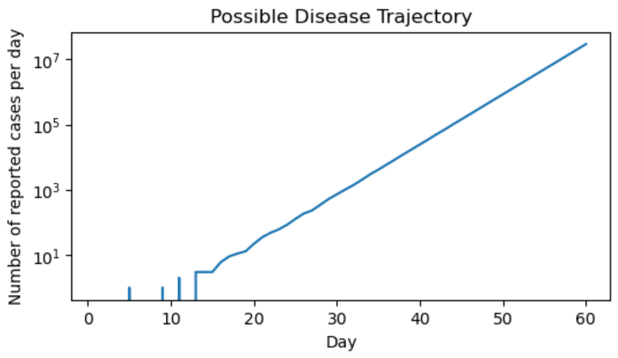
Step 1. Understand

Step 2. Decompose

Step 3. Analyze

Step 4. Synthesize

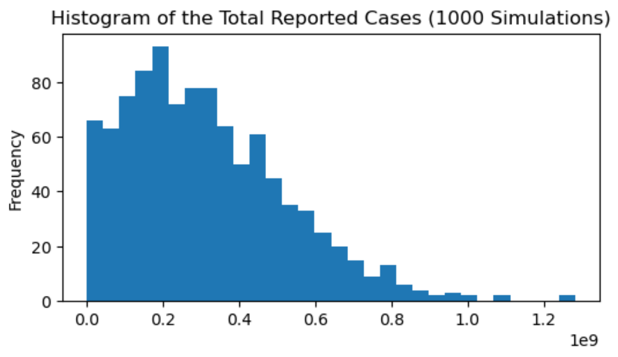
```
[40]: # Sanity check of one trajectory. Your plot will look different. Re-run to see a␣
 ↪different trajectory.
      import pandas as pd
      import matplotlib.pyplot as plt
      m=60
      s=pd.Series(simulateNewCases(m),index=range(1,m+1))
      s.plot(logy=True,title='Possible Disease Trajectory',figsize=(6,3))
      plt.xlabel('Day')
      plt.ylabel('Number of reported cases per day')
      plt.show()
```



The following set of testing code uses your function to forecast the total number of reported cases within 60 days. Due to randomness, your output would not be exactly the same.

```
[41]: # Test code
      import matplotlib.pyplot as plt
      totalReports=pd.Series([sum(simulateNewCases(60)) for i in range(1000)])
      print('Number of total reported cases in 60 days.')
      print(f'\tForecast mean: {totalReports.mean():.0f}')
      print(f'\tForecast standard deviation: {totalReports.std():.0f}')
      totalReports.plot(kind='hist',bins=30,figsize=(6,3),\
                        title="Histogram of the Total Reported Cases (1000 Simulations)")
      plt.show()
```

```
Number of total reported cases in 60 days.
        Forecast mean: 313908061
        Forecast standard deviation: 212706064
```

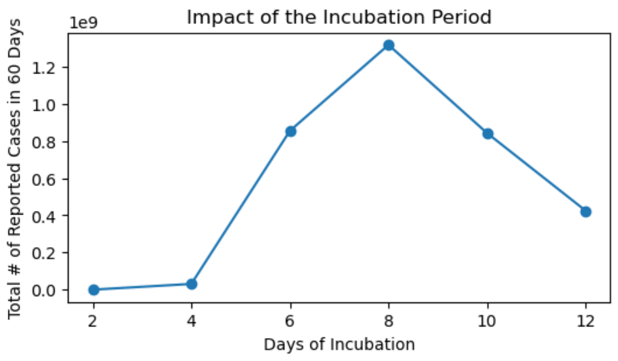


Additional Test Cases

Here are some examples of potentially interesting analyses that uses the simulation model above.

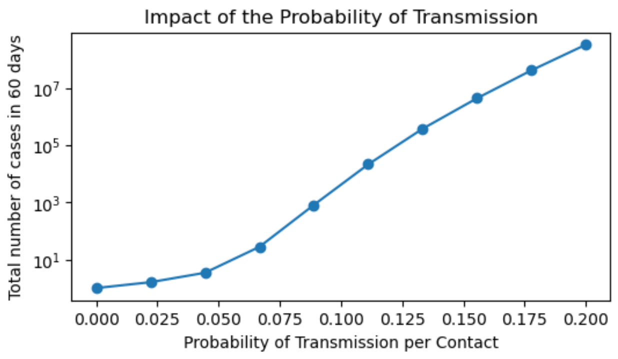
i) Impact of the Incubation Period What is the impact of the incubation period on the number of cases in the first 60 days?

```
[42]: import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
      numCases=pd.Series(dtype=float)
      for d in range(2,13,2):
          meanPrediction=np.mean([sum(simulateNewCases(60,d=d)) for i in range(1000)])
          numCases.loc[d]=meanPrediction
      numCases.plot(style='o-',title='Impact of the Incubation Period',figsize=(6,3))
      plt.xlabel('Days of Incubation')
      plt.ylabel('Total # of Reported Cases in 60 Days')
      plt.show()
```



ii) Impact of Better Hygiene How does the expected total number of cases change based on p?

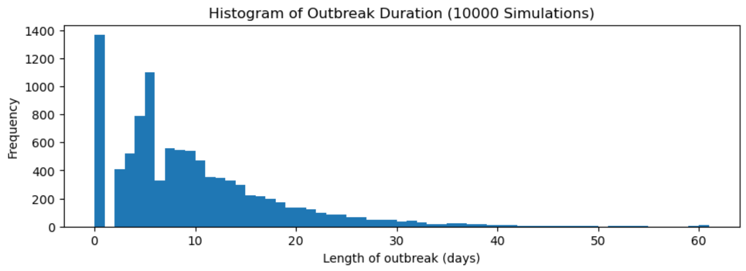
```
[43]: import numpy as np
      pList=np.linspace(0,0.2,10)
      numCases=pd.Series(dtype=float)
      for p in pList:
          numCases.loc[p]=np.mean([sum(simulateNewCases(60,p=p)) for i in range(1000)])
      numCases.plot(style='o-',title='Impact of the Probability of␣
 ↪Transmission',figsize=(6,3))
      plt.xlabel('Probability of Transmission per Contact')
      plt.ylabel('Total number of cases in 60 days')
      plt.yscale('log')
      plt.show()
```



iii) Length of an Outbreak after Herd Immunity Suppose that most people in the city have been vaccinated, so that the parameter p is reduced by 5 fold to 0.04. Moreover, the city has been case free for a while. However, on day 0, 3 new people become infected. How many days would this outbreak last? For simplicity, we say that the outbreak has ended if there are no hospital reports for 14 days.

```
[44]: import pandas as pd
      import numpy as np
      length=[]
      for i in range(10000):
          reports=simulateNewCases(78,initial=3,p=0.04)
          noCaseDays=0
          endDay=61
          day=0
          for cases in reports:
              if cases==0:
                  noCaseDays+=1
```

```
                if noCaseDays>=14:
                    endDay=day-14-4
                    break
            else:
                noCaseDays=0
            day+=1
        length.append(endDay)
    pd.Series(length).plot(kind='hist',bins=61,\
            title='Histogram of Outbreak Duration (10000 Simulations)',figsize=(10,3))
    plt.xlabel('Length of outbreak (days)')
    plt.show()
```



(Optional) Additional Analysis

What is another interesting question that might be answered using the simulation model above? How would you go about doing the analysis? I encourage you to do the analysis in your own time for practice, but you don't have to hand in anything.

Exercise 5.6: Forecasting Bonus Earnings

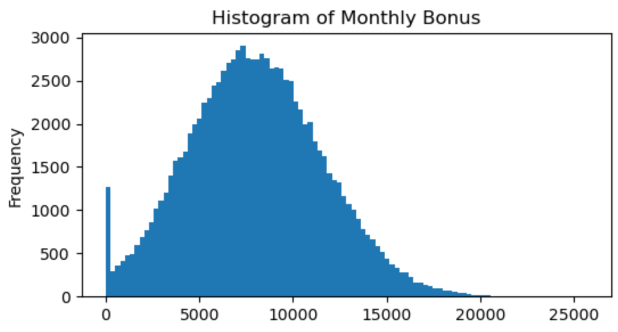
Nadeem is a car salesperson who faces the following incentive scheme at the dealership where he works. For each month, there is a "target profit" of 80,000 that the dealership sets for the month. If he makes more profit for the dealership that month than the target, then he receives a 20% bonus on the amount of profit over the target. However, if he does not meet the target, he receives no bonus. For example, if he makes 100,000 of profit, then he receives a 4,000 bonus that month: $(100000 - 80000) \times 0.2 = 4000$. However, if he makes 70,000, then he receives zero bonus that month. Nadeem would like to understand the distribution of his monthly bonus.

Nadeem estimates that the number of cars he sells is binomial distributed with $n = 200$ and $p = 0.2$. On every car he sells, the amount of profit he makes for the dealership is normally distributed with $\mu = 3000$ and $\sigma = 1000$, and the profit from each car is **independent** of another. This means that the profit for two different cars should be two independent draws of the normal distribution.

Create a `Series` called "monthlyBonus" with 100,000 samples of his monthly bonus. As in the sample outputs below, print the mean and the standard deviation rounded to the nearest integer, as well as the probability the bonus is less than 5000, with the probability being rounded to three decimal places. Finally, plot a histogram of simulated earnings with 100 bins.

Note: Given the inherent randomness with simulation, even if your cord is perfectly correct, the estimates will be slightly different from those shown below.

```
Mean bonus: 8009
Standard deviation: 3595
Probability that bonus is less than 5000: 0.207
```



Exercise 5.7: Simulating Stock Prices

This question asks you to simulate the weekly price of a certain stock, given the initial price, the expected change in price per week, the standard deviation per week, and the number of weeks to simulate.

Write a function called "simulatePrice" with the following input arguments:

- **initial**: a positive number representing the price in Week 0.
- **mu**: the expected change in price from one week to the next.
- **sigma**: the standard deviation in the change in price.
- **N**: the number of weeks to simulate. This does not include Week 0, for which the price is already given.

The function should return a Pandas Series containing the simulated price of the stock from Week 0 through N. (Note that the initial price is the first entry and is not simulated.)

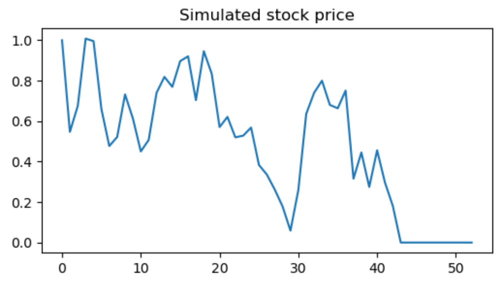
You should assume the following probabilistic model. Suppose the price in week $t-1$ is p_{t-1}, then the price in week t is given by

$$p_t = \begin{cases} 0 & \text{if } p_{t-1} = 0, \\ \max(0, p_{t-1} + \epsilon_t) & \text{if } p_{t-1} > 0, \end{cases}$$

where ϵ_t is independently and Normally distributed with mean **mu** and standard deviation **sigma**, as given by the input arguments.

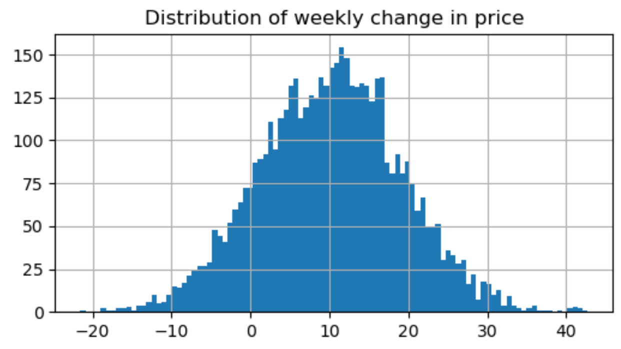
The following plots one random sequence of 52 weeks.

```
[47]: # Sanity check. Don't worry about exactly reproducing this plot. Re-run for a␣
↪different sequence.
      import matplotlib.pyplot as plt
      simulatePrice(1,0.01,0.2,52).plot(title='Simulated stock price',figsize=(6,3))
      plt.show()
```

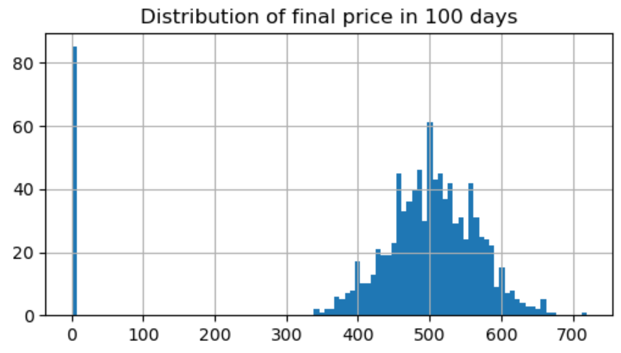


```
[48]: # Test code 1
      import matplotlib.pyplot as plt
      import numpy as np
```

```
s=simulatePrice(10000,10,9,5000)
s.diff(1).hist(bins=100,figsize=(6,3))
plt.title('Distribution of weekly change in price')
plt.show()
```



```
[49]: # Test code 2
      import matplotlib.pyplot as plt
      import numpy as np
      end=pd.Series(simulatePrice(5,5,6,100).iloc[-1] for i in range(1000))
      end.hist(bins=100,figsize=(6,3))
      plt.title('Distribution of final price in 100 days')
      plt.show()
```



Exercise 5.8: Simulating a Strategy for Stock Trading

This question asks you to evaluate an algorithmic trading strategy using a time series of stock prices, similar to the output in Q2.

Write a function called "simulateTrades" with the following input arguments:

- **prices**: a pandas Series of stock prices, with each entry representing the price of the stock in a given week. You can assume that all given prices are non-negative (i.e. ≥ 0).
- **cash**: a positive decimal number designating the total amount of money you have at the very beginning, before any trades occur in Week 0. You use up cash when buying stocks and gain cash when selling stocks.
- **p_L**: a threshold on the price to purchase. As soon as the stock price is less than or equal to this price, assuming that the price is non-zero, you would purchase as many shares as you can using all the cash you have. (**Note: you can only purchase an integer number of shares.** You can round down a number n using the code `int(n)`. i.e., `int(3.6)` would yield 3.) If the price is equal to zero, then you would not purchase.

- **p_H**: a threshold on the price to sell. As soon as the stock price is greater than or equal to this price, you would sell all the shares you have and convert it to cash.

Assume that at the beginning, you have zero shares of the stock. Moreover, in the last week, you have to sell all the shares you have regardless of the price. **The function should return (not print) the total amount of cash you have at the end of all periods.**

For example, suppose that `prices=pd.Series([1.1,1.3,0.8,1.3,1.6,0.4])`. Then

- `simulateTrades(prices, 8, 0.8, 1.6)` returns 16.0 because you would buy 10 shares when the price hits 0.8 and sell then for 1.6 each when the price reaches 1.6. In the last period, you buy 40 shares at 0.4 but sell everything again at that price, so your cash doesn't change.
- `simulateTrades(prices, 8, 0.8, 1.7)` returns 4.0 because while the sell price of 1.7 is never reached, you are stuck with 10 shares, which you are forced to sell in the last period for 0.4 each.
- `simulateTrades(prices, 10, 0.8, 1.6)` returns 19.6 because you buy 12 shares when the price reaches 0.8, with 0.4 cash left, and you sell all 12 shares at a price of 1.6. Your ending cash is $0.4 + 1.6 \times 12 = 19.6$.
- `simulateTrades(prices, 8, 0.7, 1.6)` returns 8.0 because the price is never low enough for you to buy, so you end up with the same amount of cash you had at the beginning.
- `simulateTrades(prices, 8, 0.8, 1.2)` returns 13.0 because you buy the 10 shares when the price hits 0.8 and sell everything when the price hits 1.3.
- `simulateTrades(prices, 11, 1.2, 1.3)` returns 21.0 because you immediately buy 10 shares since the opening price of 1.1 is lower than 1.2. You sell these shares when the price hits 1.3, and now you have 13 dollars of cash. Once the price goes down to 0.8, you buy 16 shares, and have 0.2 in cash left. You sell those 16 shares when the price hits 1.3, so the final cash is $0.2 + 16 \times 1.3 = 21.0$.

```
[51]: # Sample runs
      import pandas as pd
      prices=pd.Series([1.1,1.3,0.8,1.3,1.6,0.4])
      simulateTrades(prices, 8, 0.8, 1.6)

16.0

[52]: simulateTrades(prices, 8, 0.8, 1.7)

4.0

[53]: simulateTrades(prices, 10, 0.8, 1.6)

19.6

[54]: simulateTrades(prices, 8, 0.7, 1.6)

8.0

[55]: print(simulateTrades(prices, 8, 0.8, 1.2))

13.0

[56]: print(simulateTrades(prices, 11, 1.2, 1.3))

21.0

[57]: simulateTrades([5.0,3,8,10,3,10,6,8,11,8,6,3,1,8], 100,9, 4)

1224.0

[58]: simulateTrades([5.0,3,8,10,3,10,6,8,11,8,6,3,1,8,1], 100,9, 4)

153.0

[59]: simulateTrades([5.0,3,8,10,3,10,6,8,11,8,6,3,1,8,1,10], 100,9, 4)

1530.0
```

Additional Test Code: Combining Exercise 5.7 and 5.8 into One Simulation Model

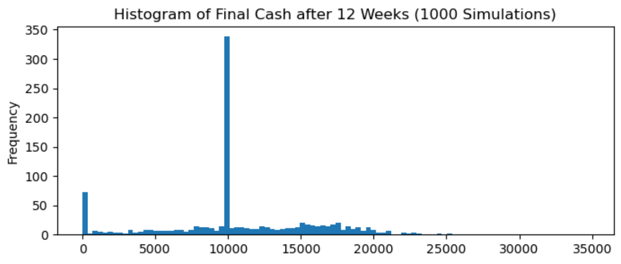
Once you have finshed Exercise 5.7 and 5.8, the following code illustrates how you can combine them to forecast the performance of the trading strategy, given the parameters initial, mu, sigma, N, cash, p_L and p_H. Your graph would not be exactly as follows but the shape should be similar, otherwise there is probably an error.

```
[60]: import matplotlib.pyplot as plt

      initial=1
      mu=0.01
      sigma=0.2
      N=12
      cash=10000
      p_L=0.9
      p_H=1.3

      # Combining Exercises 5.7 and 5.8
      finalCash=pd.Series(dtype=float)
      for i in range(1000):
          prices=simulatePrice(initial,mu,sigma,N)
          finalCash.loc[i]=simulateTrades(prices,cash,p_L,p_H)
      finalCash.plot(kind='hist',title='Histogram of Final Cash after 12 Weeks (1000␣
↪Simulations)',bins=100,figsize=(8,3))
      print('Mean:', finalCash.mean())
      print('Standard deviation:', finalCash.std())
      plt.show()
```

```
Mean: 10844.993015911929
Standard deviation: 5437.075200859241
```



Information about the Midterm Exam

As described in the Syllabus, the midterm will be held during class time on the Tuesday after Week 6. It tests your mastery of skills taught in Weeks 1-5, which culminates in creating simulation models using Python and algorithmic thinking. The exam is 80 minutes, and will span the entire class session. There are three questions, worth a total of 24 points. See the four sample midterms posted on Blackboard for examples of question formats and difficulties.

The exam is open notes but closed computer. You can bring paper notes or books of any kind, but no computers, tablets, or cell phones are allowed. Before the midterm is graded, do not share the questions or your solutions with anyone else, including students of other sections. When you turn in the exam, you must also hand in all scrap paper that you wrote on. Do not use a cell phone, tablet or computer in the classroom before all of the exams are handed in.

Week 6 will be dedicated to practicing for the exam. **We will discuss Sample Midterm A on Tuesday and Sample Midterm B on Thursday.** You will have a little bit of time to work on these sample midterms during class, but it's best if you attempt them beforehand and be ready to discuss your solutions. For the other two sample midterms, I will post videos to go over the solutions on Blackboard, but you should attempt them yourself under test conditions before watching the videos.