Lab_01

January 27, 2025

Probability for Data Science

UC Berkeley, Spring 2025

Michael Xiao and Ani Adhikari

1 Lab 1: Birthday Attack (Due Monday, January 27th at 5 PM)

Welcome to lab in Data 140! In this first lab you will get acquainted with the computing environment of the course and explore an application of the birthday problem. Specifically, you will:

- Review the code used in the textbook to study the birthday problem on Planet Earth
- Study the "birthday paradox" on Mars (or, if you have a more practical outlook, in hashing collisions), with an exact computation as well as an approximation
- See how the title of this lab is actually a thing, not just a cutesy way of getting your attention

1.0.1 Instructions

Similar to your homeworks, your labs will generally have two components: a written portion and a portion that also involves code.
- Written work should be completed on paper, and coding questions should be done in the notebook.
- Start the work for the written portions of each section on a new page. - You are welcome to LaTeX your answers to the written portions, but staff will not be able to assist you with LaTeX related issues. - Show your work. Give reasoning. The question isn't always going to ask for it, because we assume that you will provide justification for your answers. Every answer should contain a calculation, reasoning, or diagrams that are clearly labeled to show what's going on. - It is your responsibility to ensure that both components of the lab are submitted completely and properly to Gradescope. **Make sure to assign each page of your pdf to the correct question. - Refer to the bottom of the notebook for submission instructions.**

First run the Setup cell below. You can ignore its contents for now if you aren't interested. In future labs, this cell will appear before the start of the lab.

[64]:
```
# SETUP

# These lines make warnings go away
```

```python
import warnings
warnings.filterwarnings('ignore')

# The main libraries
import numpy as np
from datascience import *
from prob140 import *

# These lines do some fancy plotting magic
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

1.0.2 Useful Methods

Because you have only recently returned to school after the break, we are providing a list of some methods that will be useful in the lab. Please consult the one-page **code reference sheet** in Resources if you need a reminder of the syntax.

For today's lab you will need some or all of: - Array operations and NumPy functions including `item`, `diff`, and `append` - Defining functions: `def` - Conditional statements: `if/else` - Iteration: `for` (or any other Python method for iteration) - `Table` methods from the `datascience` library: - Creation: `with_columns` - Accessing rows that satisfy a condition: `where` - Accessing and using values as inputs to a function: `apply` - Scatter plots: `scatter` - Distribution methods from the `prob140` library, operating on `Table()`: - Specifying the possible values: `values` - Specifying the probabilities: `probabilities` or `probability_function` - Visualization methods from the `prob140` library: - Probability histogram of an integer-valued random variable: `Plot` - Overlaid probability histograms of two integer-valued random variables: `Plots`

1.1 Identify Your Lab Partner

This is a multiple choice question. Please select **ONE** of following options that best describes how you complete Lab 1.

- I am doing this lab by myself and I don't have a partner.
- My partner for this lab is [PARTNER'S NAME] with email [berkeley.edu email address]. [SUBMITTER'S NAME] will submit to Gradescope and add the other partner to the group on Gradescope after submission.

Please copy and paste **ONE** of above statements and fill in blanks if needed. If you work with a partner, make sure only one of you submit on Gradescope and that the other member of the group is added to the submission on Gradescope. Refer to the bottom of the notebook for submission instructions.

I am doing this lab by myself and I don't have a partner.

1.2 We will not grade assignments which do not have pages selected for each question.

1.3 Section 1: Birthday Paradox on Earth

In this part you will review the code used in the textbook to study the birthday problem on Planet Earth.

As you have seen, the setting of the birthday problem is a special case of n draws made at random with replacement from the set of integers $1, 2, 3, \ldots, N$.

In the context of birthdays on Earth, this corresponds to two simplifying assumptions about n people: - Each year has 365 days; $N = 365$ - Each person is equally likely to be born on each of the 365 days, regardless of the birthdays of all others

The main question is, "What is the chance that among the n people there is at least one pair whose birthdays are the same?"

In the context of hashing, the question is, "If you assign each of n individuals one of N hash values chosen at random, what is the chance of at least one collision?"

Start out by setting $N = 365$. Run the cell or it won't get set. After this, we won't keep reminding you to run cells.

```
[65]: N = 365
```

1.3.1 1a) A No Matches Calculation

Let D_n be the event that all n birthdays are different, that is, the event that there is no match. You have seen that

$$P(D_n) = \prod_{i=0}^{n-1} \frac{365 - i}{365}$$

Notice that $P(D_1) = 1$. There is certain to be no match if there is only one birthday, because there is nothing it can be matched with.

Notice also that the computation labels the n people as $0, 1, 2, \ldots, n-1$, which makes Python's 0-origin indexing rather convenient.

Let's brush off any cobwebs that might have gathered on your programming skills over the holidays. Run the cells below and observe the output carefully.

```
[66]: n = 5
      individuals = np.arange(n)
      individuals
```

```
[66]: array([0, 1, 2, 3, 4])
```

```
[67]: N - individuals
```

```
[67]: array([365, 364, 363, 362, 361])
```

```
[68]: (N - individuals)/N
```

```
[68]: array([ 1.        ,  0.99726027,  0.99452055,  0.99178082,  0.9890411 ])
```

```
[69]: np.prod( (N - individuals)/N )
```

```
[69]: 0.97286442630020653
```

1.3.2 1b) The Probability of No Matches

Use the sequence of steps in 1a to define a function `p_no_match` that takes n as its argument and returns $P(D_n)$, assuming the fixed value $N = 365$.

```
[70]: def p_no_match(n):
          individuals = np.arange(n)
          return np.prod((N - individuals) / N)
```

1.3.3 1c) No Matches Among 23 People

Based on what you saw in class about the birthdays of 23 people, $P(D_{23})$ should be

(i) a bit less than 1/2 (ii) 1/2 (iii) a bit more than 1/2

Pick the right option and explain without computation.

The probability should be a bit less than 1/2 because at n = 23 people, the chance that there is at least one match exceeds 50%, making the chance that there is no match less than 50%.

Confirm your answer in the cell below.

```
[71]: p_no_match(23)
```

```
[71]: 0.4927027656760144
```

1.3.4 1d) No Matches Table

The birthday problem is only interesting for $1 \le n \le N$, because for larger n it's clear that $P(D_n)$ must be 0. Use a table to display your value of $P(D_n)$ for every n in the range 1 through N, in the following steps.

The next cell sets up a table of all the possible numbers of people.

```
[72]: birthday_probs = Table().with_columns('People', np.arange(1, N+1))
      birthday_probs
```

```
[72]: People
      1
      2
      3
      4
      5
      6
      7
      8
```

```
9
10
… (355 rows omitted)
```

Create an array called `all_different` that contains $P(D_n)$ for each n in the range 1 through N.

You will need the Table method `apply` that applies a function to a specified column in each row of a table.

[73]:
```
all_different = birthday_probs.apply(p_no_match, 'People')
```

Which item of `all_different` corresponds to 23 people? That item of `all_different` should agree with 1c; check this!

[74]:
```
all_different.item(22)
```

[74]:
```
0.4927027656760144
```

Augment `birthday_probs` with two columns - one labeled `P(no match)`; in the row corresponding to n people, the column should contain $P(D_n)$ - one labeled `P(at least one match)`; in the row corresponding to n people, the column should contain $P(\text{at least one match among the } n \text{ birthdays})$

[75]:
```
birthday_probs = birthday_probs.with_columns(
    "P(no match)", all_different,
    "P(at least one match)", 1 - all_different
)

birthday_probs
```

[75]:
```
People | P(no match) | P(at least one match)
1      | 1           | 0
2      | 0.99726     | 0.00273973
3      | 0.991796    | 0.00820417
4      | 0.983644    | 0.0163559
5      | 0.972864    | 0.0271356
6      | 0.959538    | 0.0404625
7      | 0.943764    | 0.0562357
8      | 0.925665    | 0.0743353
9      | 0.905376    | 0.0946238
10     | 0.883052    | 0.116948
… (355 rows omitted)
```

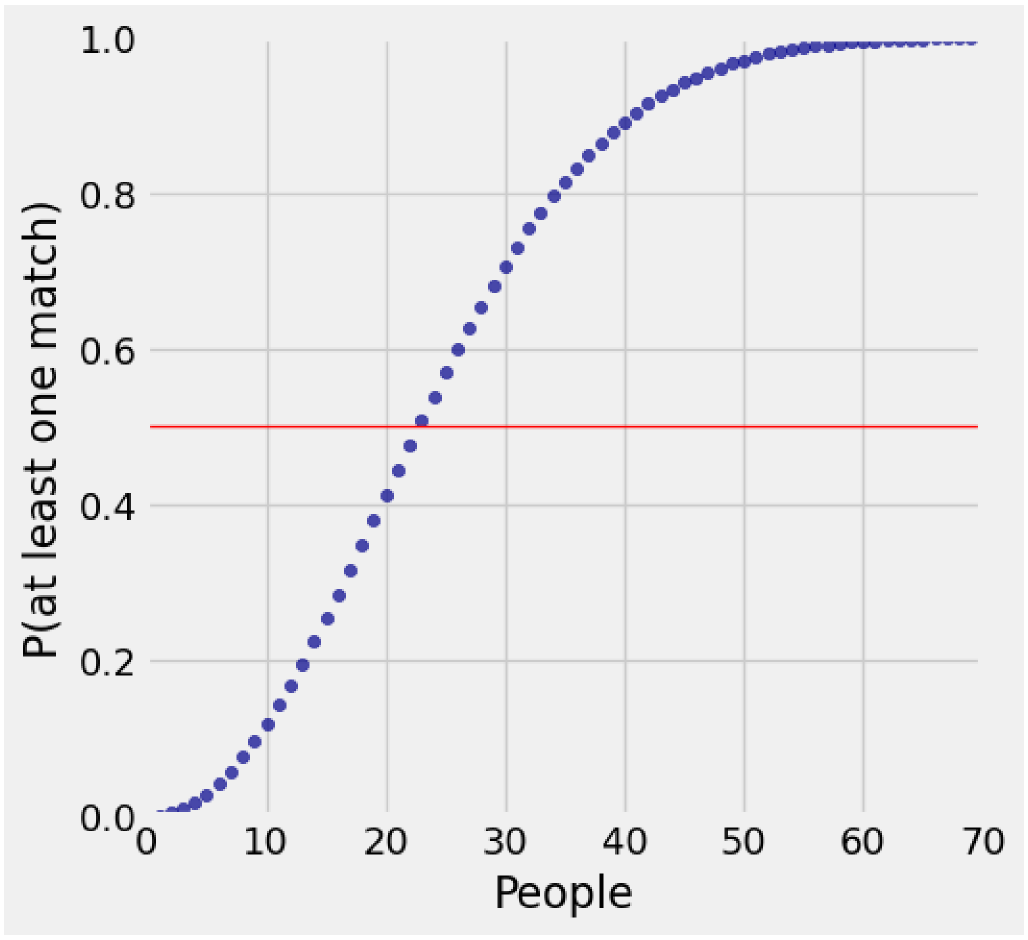
Compare with the `results` table in Section 1.4 of the Data 140 textbook. It should be the same apart from column labels.

1.3.5 1e) Visualizing the Birthday Paradox

Now visualize the "birthday paradox". Draw the scatter plot of `P(at least one match)` versus `People`.

In the cell below, we have restricted the values of `People` to just the range where there is visible change in the probability being plotted. And we have drawn a horizontal line at the level 1/2. The code for the graphics is briefly explained in the comments. The final semi-colon prevents some unnecessary text output from `matplotlib`.

```
[76]: birthday_probs.scatter("People", "P(at least one match)")

# Everything below this line is for fine-tuning the graphics.
# There is nothing for you to enter below this line.

# plt is short for matplotlib.pyplot; see the import cell at the top

plt.xlim(0, 70)     # restrict trials to at most 70

plt.ylim(0, 1)      # use the probability scale on the vertical axis

# Draw a red horizontal line at level 1/2
# plt.plot joins the dots between the two points (x_1, y_1) and (x_2, y_2)
# Arguments: [x_1, x_2], [y_1, y_2], color=, and lw=
# That last one is line width. Bigger values produce thicker lines.

plt.plot([0, 70], [0.5, 0.5], color='red', lw=1);
```



Use the graph to indentify the smallest number of people at which P(at least one match) exceeds 1/2.

The smallest number of people at which P(at least one match exceeds 1/2) is 23 people.

1.4 Section 2: Birthday Paradox (Martian Edition)

Now suppose you're a Martian. Then one year on your planet is 687 days long. Or in general, suppose you're from a planet whose year has N days. The goal of this part is to answer the following question:

For a year of length N, what is the smallest number of individuals so that the chance of a match among those individuals is at least half?

In terms of a hash table with N values, you are trying to find the smallest number of individuals so that a collision is more likely than not.

We'll call this number the *tipping point* corresponding to a hash table with N values.

In this part you will write a function to find the tipping point as a function of N. To gain some efficiency in the code, the steps below avoid Table methods; those work better for display, visualization, and so on.

1.4.1 2a) A Formula for D_n [On Paper]

Based on the tipping point for the Earth year (23, with $N = 365$), you should have the sense that the tipping point is going to be small relative to N. Here are a couple of observations that will help you write your function.

First review the formula for $P(D_n)$ in Part **1a**. This is the probability of no match among n people.

Fill in the blank with the correct factor:

For $2 \le n \le N$,

$$P(D_n) \;=\; P(D_{n-1}) \cdot \underline{\hspace{3cm}}$$

If you have trouble, start with $N = 365$ and refer to **1a**. What will happen if you replace n by $n - 1$?

In what follows, keep in mind that:

- $P(D_1) = 1$
- The smallest n for which $P(\text{at least one match among } n \text{ birthdays}) > 1/2$ is also the smallest n for which $P(D_n) < 1/2$.

1.4.2 2b) Calculating the Tipping Point

Use 2a to define a function called `tipping_point` that takes N as its argument and returns the tipping point when there are N days in the year.

Do **not** compute $P(D_n)$ for all n. The vast majority of that computation will be unnecessary. Rather, set up your computation in the steps indicated by 2a:

- Start with one individual and probability 1 of no match.
- As long as the probability is greater than 1/2, keep increasing the number of individuals by 1 and updating the probability of no match by the product formula you came up with in 2a.
- Once the probability is less than 1/2, return (carefully!) the number of individuals corresponding to the tipping point.

There are numerous ways to write this function. Use any efficient way; this implies avoiding Table here.

```
[77]: def tipping_point(N):
          n = 1
          prob = 1

          while prob >= 1/2:
              n += 1
              prob = prob * ((N - (n - 1)) / N)

          return n
```

Run the cell below as a check of whether your function is working correctly.

```
[78]: tipping_point(365)
```

```
[78]: 23
```

1.4.3 2c) The Tipping Point on Mars

What's the tipping point on Mars?

```
[79]: tipping_point(687)
```

```
[79]: 32
```

1.4.4 2d) The Tipping Point of a 16-Bit Hash

If you use a 16-bit hash, there are $2^{16} = 65536$ hash values. What is the tipping point for a 16-bit hash?

```
[80]: tipping_point(65536)
```

```
[80]: 302
```

Optional: If you use a 32-bit hash there are $2^{32} \approx 4.3 \times 10^9$ hash values. If you wrote your function reasonably efficiently, it should give you the tipping point for a 32-bit hash after some chugging, without crashing your system. But please only try it after saving your progress. It's not a required element of the lab.

```
[81]: # OPTIONAL
      ...
```

1.5 Section 3: The Attack

Hash functions play a major role in cryptography and computer security. For example, servers verify passwords by comparing the hash value of an entered string with the hash value stored in a database. *Birthday attacks* take advantage of the birthday paradox to generate hash collisions and trick computers into accepting a malicious file in place of a previously-encountered safe file.

The key idea behind birthday attacks is what you have observed in this lab: by making a relatively small number (the tipping point!) of random attempts, the attacker has more than an even chance of matching a hash value that has been assigned.

Note that the attackers don't aim to collide with a *specific* hash value—they just try for a collision with any assigned value. In other words, the goal is to get any match, just as in the birthday problem.

1.5.1 3a) An Exponential Approximation

If you use a 64-bit hash there are approximately 1.8×10^{19} hash values. It might be cruel to ask a computer to crank out your `tipping_point` value for such a large N.

But you know an exponential approximation to the chance of no match, which you can now use to approximate the tipping point.

Review the approximation in Step 4 of Section 1.5 of the Data 140 textbook. You should review the math that went into the approximation, but for this lab it is enough just to know the final answer:

For large n,

$$P(D_n) \approx e^{-\frac{1}{2N}n^2}$$

To get a rough approximation of the tipping point, set $P(D_n)$ to $1/2$ in the approximation above and solve for n in terms of N. Do this on scratch paper. Then write a function called `approx_tipping_point` that takes N as its argument and returns the approximate tipping point.

Use `np.log(x)` for $\log(x)$ and the ceiling function `np.ceil(x)` for the smallest integer greater than or equal to x.

[82]:
```python
def approx_tipping_point(N):
    return np.ceil((2 * N * np.log(2)) ** (1/2))
```

1.5.2 3b) Approximate Tipping Points

Run the cell below and compare with the exact answers you got in Part 2. Keep in mind that all the calculations might be affected by floating point accuracy issues.

[83]:
```python
approx_tipping_point(365), approx_tipping_point(687),␣
↪approx_tipping_point(2**16)
```

[83]:
```
(23.0, 31.0, 302.0)
```

1.5.3

3c) The Approximate Tipping Point of a 128-Bit Hash ### What's the approximate tipping point for a 128-bit hash?

```
[84]: approx_tipping_point(2**128)
```

```
[84]: 2.1719381355163562e+19
```

1.5.4 3d) Checking Your Work

Take a look at the Wikipedia page on birthday attacks. Scroll down to the table in the middle of the article.

The notation H in the second column is what we are calling N.

We have been looking for an essentially 50% chance of getting a match, so you can just focus on the 50% column under Desired Probability of a Random Collision.

Are your approximations in **3b** and **3c** consistent with the entries in this column?

1.5.5 My approximations for 3b and 3c are consistent with the entries in this column.

Further Reading on the Birthday Attack To test whether their cryptographic hash functions are safe from attack, tech companies have to try to break them. Security researchers at Google generated a hash collision using SHA-1, a widely used hashing algorithm that generates 160-bit hash values. If you are interested in learning more, take a look at Google's technical report. Their recommendation after generating the collision: "For the tech community, our findings emphasize the necessity of sunsetting SHA-1 usage."

1.6 Section 4: The Time Required for a Collision

Suppose you start out with N hash values and assign them to individuals one by one at random with replacement as we have assumed throughout.

Let M_1 be the number of individuals who are assigned hash values until the first time the value assigned has been assigned before. For example, if the sequence of assigned values starts out as 212, 41, 7, 90, 41, then $M_1 = 5$.

Important Note: In this course "until" will mean "up to and including" as you can see in the example above.

The random variable M_1 is called *the time of the first match*. In this part of the lab you will find its probability distribution.

1.6.1 4a) The Possible Values

When you are trying to identify a distribution, **always** start with the possible values. In terms of N, what are the possible values of M_1?

The possible values of M1 are 1,2,3,... all the way up to N.

1.6.2 4b) [On Paper] D_n and M_1

Suppose there are n individuals in all. As in Part **1a** let D_n be the event that all n individuals are assigned different values. In Part 1 you have the algebraic formula for $P(D_n)$ as well as numerical values when the total number of hash values is $N = 365$.

Fill in the blank with one of the symbols $=, >, \geq, <, \leq$, and **explain your logic**.

For $n > 1$, the event D_n is the same as the event $\{M_1$ ______ $n\}$.

1.6.3 4c) [On Paper] The Distribution of M_1

For $n \geq 1$ define the event $G_n = \{M_1 > n\}$.

Fill in the blanks. - The first blank should be filled in with set operations (such as union, intersection, difference, complement, etc) performed on some or all of the events $G_1, G_2, \ldots$. - The second blank should be filled in with arithmetic operations performed on some or all of $P(G_1), P(G_2), \ldots$.

For a possible value n in the range identified in Part **a** above, the event $\{M_1 = n\}$ is the same as the event __________________, and hence $P(M_1 = n) =$ __________________________________.

Be sure to show that your answer is correct for the values of n that are at the edges of your answer to Part **a**.

1.6.4 4d) The Distribution of M_1, Numerically

Complete the cell below to find probability distribution of M_1 numerically in the case $N = 365$.

- The array `possible_vals_M1` should contain the possible values of M_1 found in Part **a** above.
- The probabilities of those values should be placed in the array `matching_time_probs`, calculated using:
  - Your work in Parts **a-c** above

- The array `all_different` from **Part 1**
- NumPy functions; consult the list provided at the start of the lab

Be careful with `matching_time_probs`. Keep track of signs and lengths, and check that the first and last values are correct.

[85]:
```
N = 365

""" Distribution of M_1 """

# Array of possible values, in terms of N
possible_vals_M1 = np.arange(1, N + 1)

# Carefully ... An array of the corresponding probabilities, in terms of N and
# the array all_different
matching_time_probs = np.zeros(N)

for k in possible_vals_M1:
    matching_time_probs[k - 1] = all_different[k - 1] - all_different[k]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[85], line 13
     10 matching_time_probs = np.zeros(N)
     12 for k in possible_vals_M1:
---> 13     matching_time_probs[k - 1] = all_different[k - 1] - all_different[k

IndexError: index 365 is out of bounds for axis 0 with size 365
```

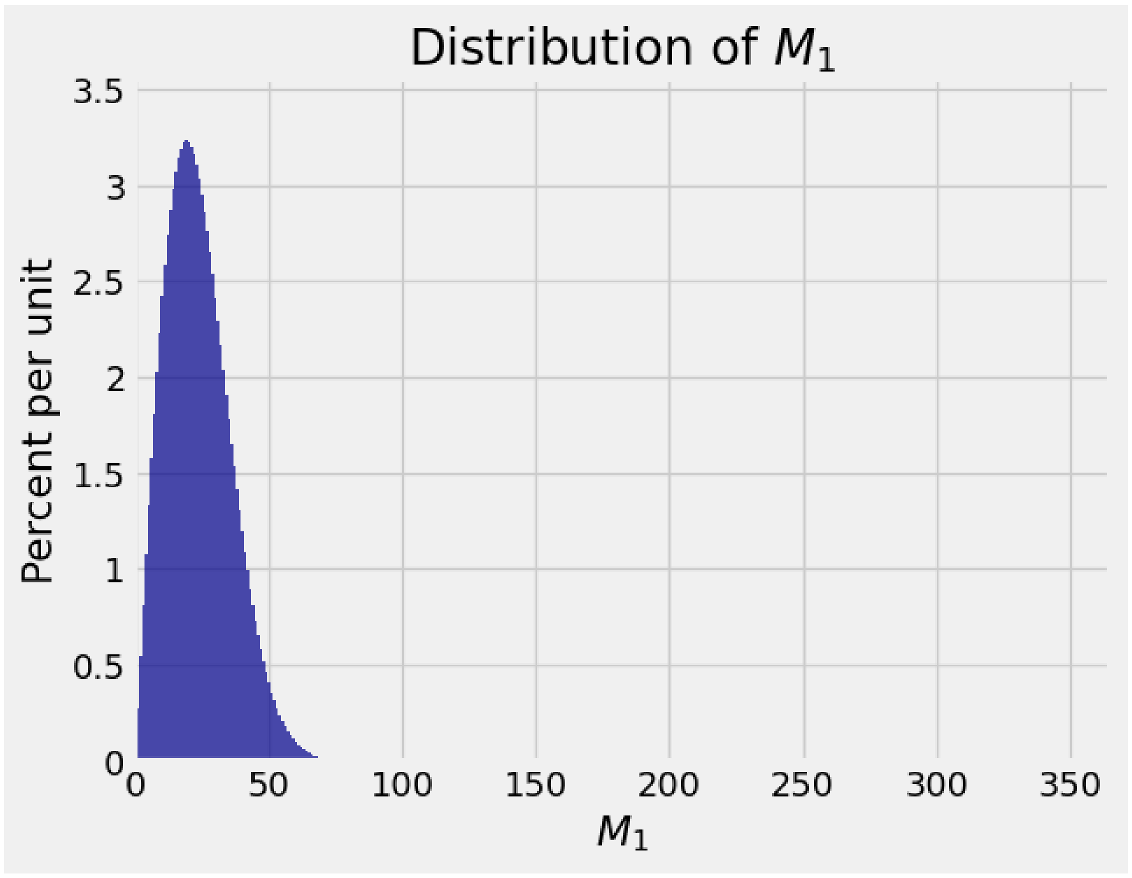
Run the cell below for a ball-park check of whether your calculations are correct. The two lengths should match, and you know what the sum should be.

[86]:
```
len(possible_vals_M1), len(matching_time_probs), sum(matching_time_probs)
```

[86]:
```
(365, 365, 0.9999999999999989)
```

Run the cell below to display the probability histogram of $M_1$.

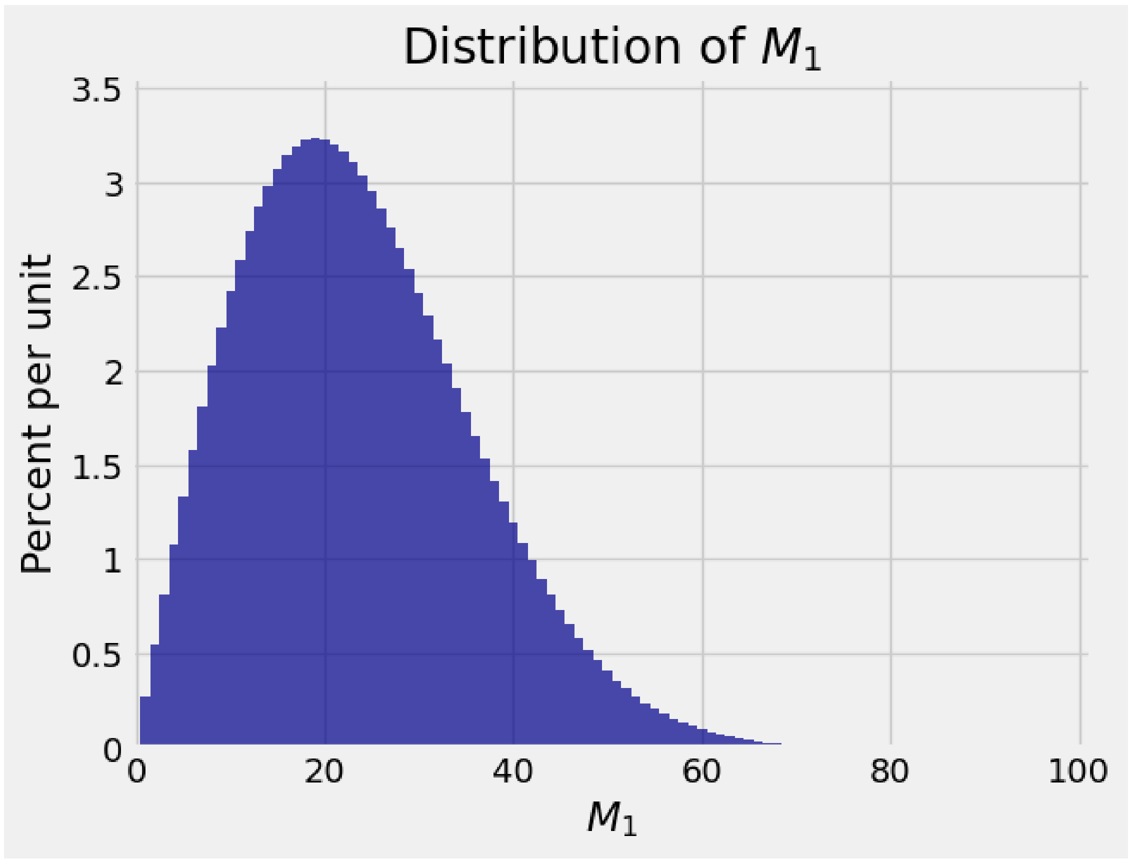
[87]:
```
dist_M1 = Table().values(possible_vals_M1).probabilities(matching_time_probs)

Plot(dist_M1)

# labels
plt.xlabel('$M_1$')
plt.title('Distribution of $M_1$');
```



If you have calculated the distribution correctly, the histogram should be bunched up on the left. For the other values, the chances are so small that the bars are invisible.

Run the cell below to zoom in on the main action.

```
[88]: truncated_dist_M1 = dist_M1.take(np.arange(100))

      Plot(truncated_dist_M1)

      plt.xlabel('$M_1$')
      plt.title('Distribution of $M_1$');
```



Remember this shape. You'll see a continuous version of it later in the course when we study the Rayleigh distribution. Scroll down till you see a graph with a familiar shape.

1.6.5 4e) The Median of M_1

The *median* of M_1 is the smallest value of n such that $P(M_1 \leq n) \geq 0.5$.

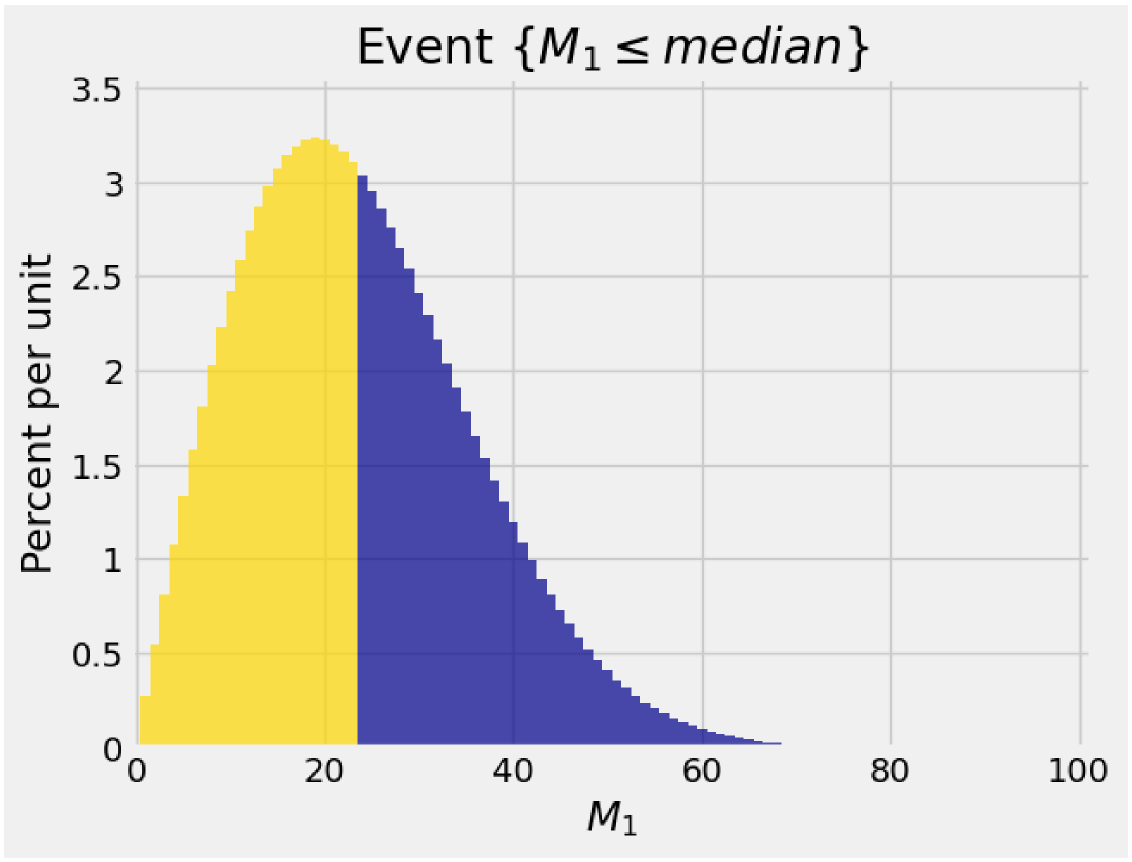
Without calculation, complete the cell below with the numerical value of the median in the case $N = 365$, and provide your reasoning in the comment.

[89]:
```
"""
Use as many lines of comment as you need to explain your logic fully.

The median is approximately 23. Looking at the distribution, I observe that the␣
 ↪peak 50% at least
half of the probability is found at or below the value of 23. This means that␣
 ↪23 is the number of
individuals when at least one collision will occur.
"""

# This should just be an number; no arithmetic operations, please
```

```
median_365 = 23
```

Now run the two cells below to visualize the event $\{M_1 \leq \text{median}\}$ and obtain its probability.

```
[90]: event_4e = np.arange(median_365 + 1)

      Plot(truncated_dist_M1, event=event_4e)

      plt.xlabel('$M_1$')
      plt.title('Event {$M_1 \leq median$}');
```



```
[91]: dist_M1.prob_event(event_4e)
```

```
[91]: 0.538344257914529
```

1.7 Conclusion

You now know: - What a birthday attack is, at least in a rough sense - Roughly how many random attempts have to be made for a collision to be more likely than not, as a function of the number of values in the hash table - That problems involving artificial settings such as birthdays can have considerably wider practical applications - That exponential approximations can be great as approximations and also as ways to get a quick sense of how a complicated function is behaving

1.7.1 You have completed your first Data 140 lab. Congratulations!

Please follow the submission instructions below to ensure that you have submitted the lab properly.

1.8 Submission Instructions

Many assignments throughout the course will have a written portion and a code portion. Please follow the directions below to properly submit both portions.

1.8.1 Written Portion

- Scan all the pages into a PDF. You can use any scanner or a phone using applications such as CamScanner. Please **DO NOT** simply take pictures using your phone.
- Please start a new page for each question. If you have already written multiple questions on the same page, you can crop the image in CamScanner or fold your page over (the old-fashioned way). This helps expedite grading.
- It is your responsibility to check that all the work on all the scanned pages is legible.
- If you used $\LaTeX$ to do the written portions, you do not need to do any scanning; you can just download the whole notebook as a PDF via LaTeX.

1.8.2 Code Portion

- Save your notebook using `File > Save Notebook`.
- Generate a PDF file using `File > Save and Export Notebook As > PDF`. This might take a few seconds and will automatically download a PDF version of this notebook.
  - If you have issues, please post a follow-up on the general Lab 1 Ed thread.

1.8.3 Submitting

- Combine the PDFs from the written and code portions into one PDF. Here is a useful tool for doing so.
- Submit the assignment to Lab 1 on Gradescope.
- **Make sure to assign each page of your pdf to the correct question.**
- **It is your responsibility to verify that all of your work shows up in your final PDF submission.**

If you are having difficulties scanning, uploading, or submitting your work, please read the Ed Thread on this topic and post a follow-up on the general Lab 1 Ed thread.

```
[ ]:
```

Lab 1 : Birthday Attack

2a. $P(D_n) = P(D_{n-1}) \cdot \underline{\quad\quad}$

$$\prod_{i=0}^{n-1} \frac{365-i}{365} = \prod_{i=0}^{n-2} \frac{365-i}{365} \cdot \underline{\quad\quad}$$

ie. $n = 3$

$$P(D_3) = \frac{(365-(1-1)) \times (365-(2-1)) \times (365-(3-1))}{365^3}$$

$$= \frac{365 \times 364 \times 363}{365^3}$$

$$P(D_2) = \frac{(365-(1-1)) \times (365-(2-1))}{365^2}$$

$$= \frac{365 \times 364}{365^2}$$

$P(D_3) = P(D_2) \cdot \underline{\quad\quad}$

$$\frac{\cancel{365} \times \cancel{364} \times 363}{365^{\cancel{3}}} = \frac{\cancel{365} \times \cancel{364}}{\cancel{365^2}} \cdot \underline{\quad\quad}$$

$$\frac{363}{365} = \underbrace{\frac{365-(3-1)}{365}}$$

$$\boxed{= \frac{N-(n-1)}{N}}$$

Ex 3a

$$P(D_n) \approx e^{-\frac{1}{2N}n^2}$$

$$\frac{1}{2} \approx e^{-\frac{1}{2N}n^2}$$

$$\ln\left(\frac{1}{2}\right) \approx \ln\left(e^{-\frac{1}{2N}n^2}\right)$$

$$\ln\left(\frac{1}{2}\right) \approx -\frac{n^2}{2N}$$

$$n^2 \approx -2N\ln\left(\frac{1}{2}\right)$$

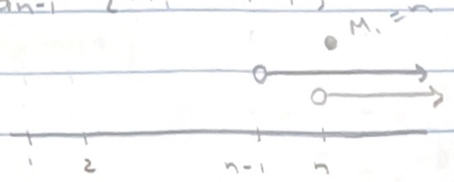
$$n^2 \approx 2N\ln(2)$$

$$n \approx \sqrt{2N\ln(2)}$$

Ex 4b. For $n > 1$, the event D_n is the same as the event $\{M_1 > n\}$ For all n individuals to be assigned different values, there can't be any duplicates among the first n individuals.

So, D_n is the same as the event $\{M_1 > n\}$, as the first match must occur after n individuals.

Ex 4c. $G_{n-1} = \{M_1 > n-1\}$



$G_{n-1} - G$ think of them as sets

the event $\{M_1 = n\}$ is the same as the event $G_{n-1} \setminus G_n$ or $G_{n-1} \cap G_n^c$, and hence

$$P(M_1 = n) = P(G_{n-1} \setminus G_n)$$
$$= P(G_{n-1}) - P(G_n)$$