# **Hotel Overbooking – Flatiron Course**

## Objectives for this course

* Define a real world problem to solve with Python
* Understand the Bernoulli experiment
* Learn about a binomial distribution and apply it to the problem
* Use the above information to help determine the optimal number of rooms to overbook

The travel and hospitality industries have become reliant on technology to help them make and keep their operations profitable. For hotel owners and managers one thing they work towards, everyday, is to fill as many rooms as possible while getting the best price. In this course, we will explore one method to help maximize profitability by optimizing the booking of rooms in a hotel using Python.

# **THE PROBLEM**

You are the owner of a hotel with 100 rooms, and you would like to maximize your revenue for these rooms. In order to determine the maximum revenue, you first need to establish what information you know about your hotel. You take a look at the books and discovered the following information:

* You have 100 rooms in the hotel that can be sold.
* Each room costs $220 USD per night
* On an average night, you know that 8% of your hotel guests who book a room don’t show up.
* If overbooked, we have to place customers at another hotel, which costs us $400 USD.

Using all of this information, we need to determine how many rooms we should overbook in order to maximize expected revenue.

## Develop a game plan

In order to successfully solve this problem, there are a few things we’ll need to learn about. First of all, we’ll need to understand what it means numerically that “on average, 8% of our hotel guests don’t show up”.

It turns out that you can model this with a **binomial distribution**. To understand **the binomial distribution**, an important concept in probability theory which lies at the foundation of data science, you first need to learn a little bit more about the Bernoulli experiment\*.

# **THE BERNOULLI EXPERIMENT**

A Bernoulli experiment is an experiment for which the probability a certain event occurs is  or ; or in other words, the event has two possible outcomes: one event occurring with probability  and the other one with probability .

**EXAMPLE 1**: A classical example of the Bernoulli experiment is flipping a coin. When flipping a coin,  is equal to 0.5, and the probability of heads is 0.5, as well as the probability of tails, which is the other possible event, is .

**EXAMPLE 2**: The probability of scoring a point when being granted a penalty kick in soccer is 0.8. In this case the Bernoulli experiment is whether someone scores or not:  is equal to 0.8, and the probability of not scoring is .

We can use Python to design a Bernoulli experiment. Let’s look at the code we use to generate the Bernoulli experiment that equals tossing a coin:

**import** numpy **as** np

print(np.random.binomial(1, 0.5, 1))

The first line of code imports the Python NumPy library. A library is essentially an open-source reusable chunk of code that you may want to include in your programs / projects. All you need to know is that NumPy is a library widely used in Python, and will help you solve some of the problems you’ll see in this course.

The second line of code uses the random.binomial function in the NumPy library to run a Bernoulli experiment. The function random.binomial takes in three so-called “arguments” here:

* The first argument represents the number of trials - or how many coin flips we’re doing in each experiment
* The second argument represents  or the probability of “success”.
* The third argument represents how many experiments you’re running (if the difference between argument 1 and 3 is not entirely clear at this time, don’t worry - it will become clear later!)

Let’s get into running some code!

**IN JUPYTER NB!**

You’ll see that you either see “1” (= success, say heads) as an output, or “0” (no success, say tails). Let’s say that we consider heads as success, that means running this experiment (or running this code cell) is exactly the same as tossing a coin. Run the code a few times and see how you’ll sometimes get a 1 and sometimes a 0!

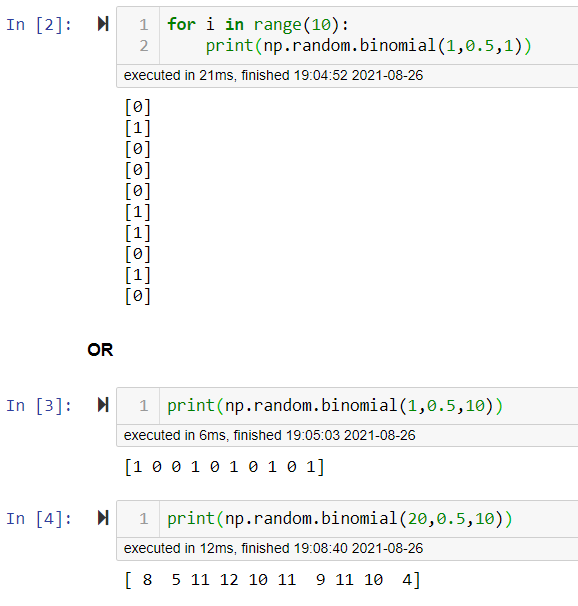
# **BUILDING THE EXPERIMENT**

## Designing the experiment

In the previous lesson we learned how to simulate the toss of a coin using NumPy’s random.binomial function. Flipping a coin once is not much of an experiment, so in this lesson we will look at using that function to perform multiple coin tosses and look at the results which is known as a Bernoulli Experiment.

We saw in our first experiment that the outcome of the coin flip is **random**, but if you run it enough times you’ll notice that out of all of the coin flips, you’ll get 1 (heads) about half of the time, and 0 (tails) about half of the time. Let’s take a look at what it looks like when we run the experiment 10 times.

Next, let’s look at what happens if we change the first parameter from 1 to 20.



This output looks as if we threw 20 coins at a time and counted the number of heads. To summarize, each number in the resulting array represents the **total number of “successes”** (say, heads) experienced during 10 experiments of flipping 20 coins (performing 20 trials) per experiment.

# **REPEATED BERNOULLI EXPERIMENTS**

## The Binomial Distribution

What you’ve been seeing here is the groundwork of what is called the **binomial distribution**. The binomial distribution is essentially a probability distribution that represents the likelihood of obtaining a predetermined number of successes (say, r) in a predetermined number of trials (say, n). Like in the Bernoulli experiment, you also need to know ahead of time what the probability of success is (say,  ).

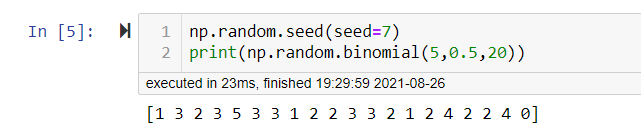
Example: What is the probability of obtaining “heads” 4 times, when flipping 5 coins?

In this example,  ,  and (because we’re talking about a coin flip)  .

The answer to this question (and Binomial distribution as a whole) can be derived mathematically, but in this lesson we’ll be using **Python** to get to the answer by repeating the experiment a very high number of times (say, 1000 or even 10000). If it doesn’t immediately click, don’t worry, you’ll see what we mean in a bit!

Let’s use np.random.binomial and run the experiment “flipping 5 coins” 20 times.

If you use the function correctly, the output will for each of the 20 experiments, simulate how many times “heads” was obtained.

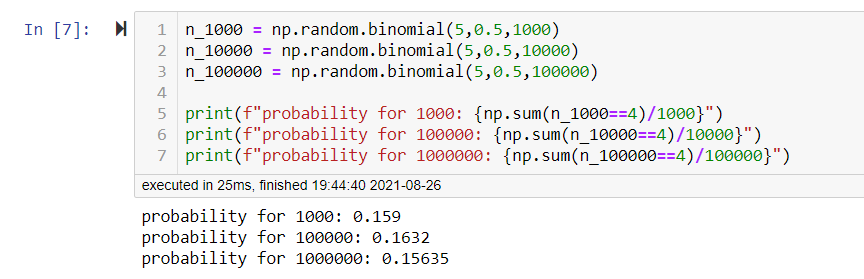


Let’s take a moment to understand what the output of this function is. When we run this function the result is a list of numbers. Each of these numbers represents the result of an experiment of flipping 5 coins. It tells us the total number of successes, or in our case, the number of heads from the 5 flips.

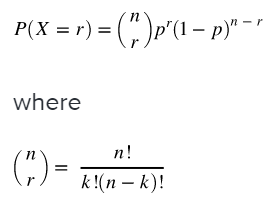
Out of these 20 experiments, there were *two* instances of “obtaining heads 4 times”. So here, you obtained the desired outcome in 4 experiments out of the 20 experiments, so 10% of the time.

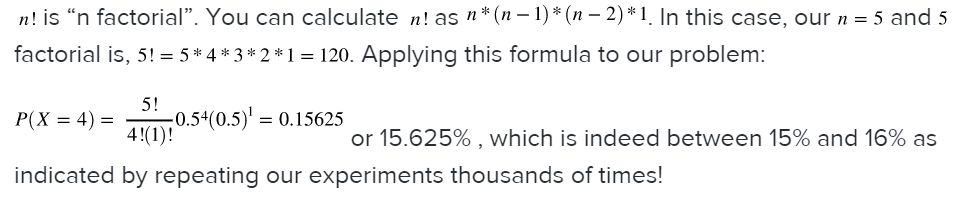
## Calculating the probability

Now, to get an accurate approximation of exactly what the probability is of getting 4 times heads, we need to repeat this experiment far more than 20 times. Below, we’ll run the experiment 1000 times, 10,000 times, and 100,000 times. We’ll store the list of outcomes in variables that we will call: n\_1000, n\_10000 and n\_100000.



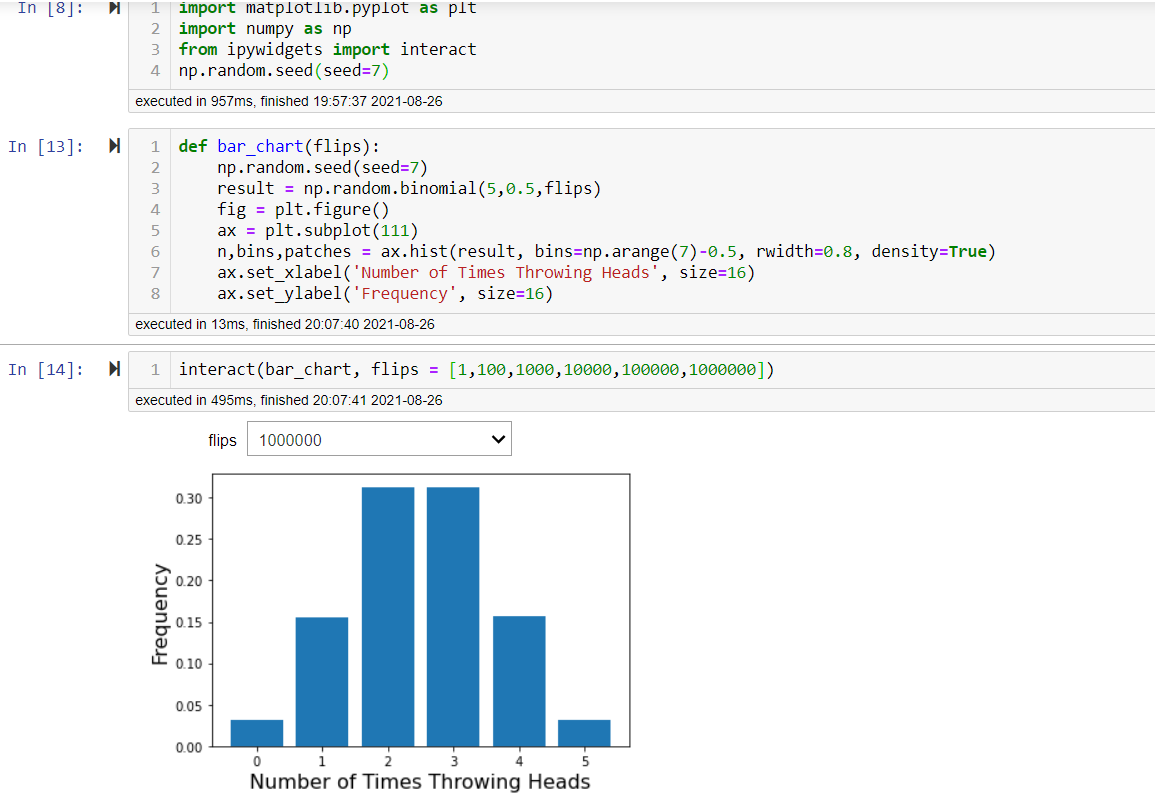
When looking at the results, it looks like the probability of getting 4 heads when flipping a coin 5 times is somewhere between 15% and 16%. For those of you who like to dive into the mathematics, this probability can actually be calculated mathematically as well! The “binomial probability” can be calculated as follows (don’t worry if what follows doesn’t immediately click - you’ll get really familiar with all this in Flatiron School’s Data Science Program):





# **VISUALIZING FLIPPING A COIN**

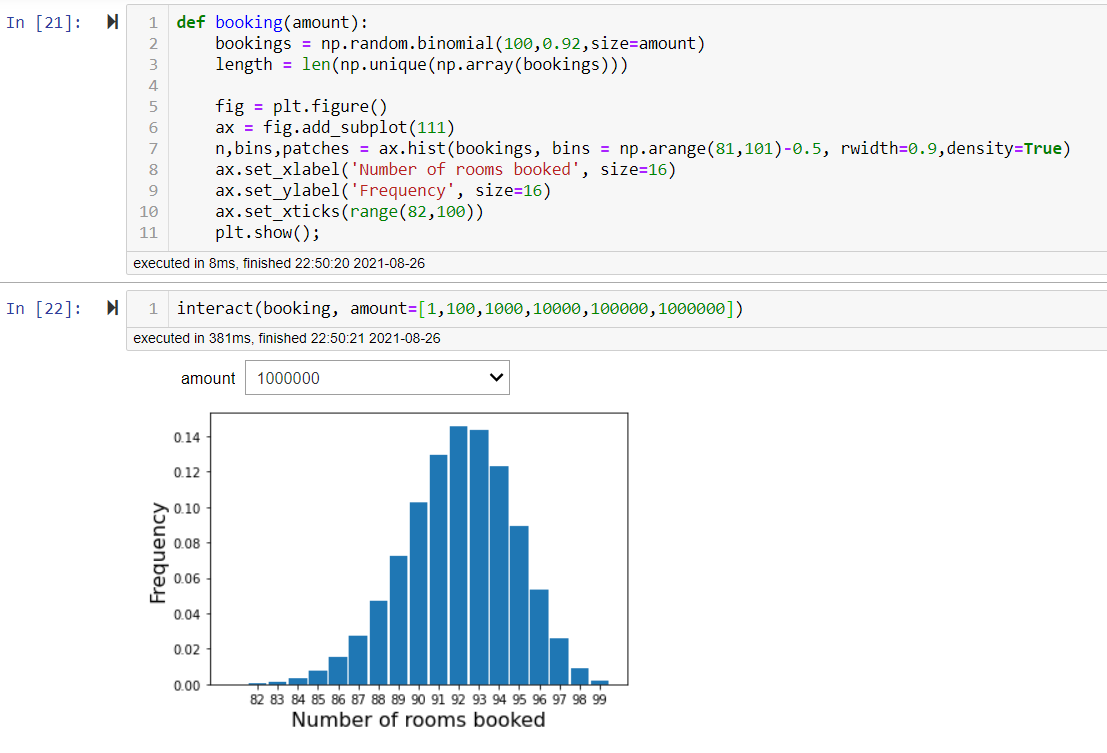
To get a full picture on how likely it is to throw heads a certain number of times, you can visualize this in a plot. In the plot below, you can see for each possible outcome (ranging from “throwing heads 0 times” to “throwing heads 5 times” how likely it is: for 4 heads, you see it’s about 16% chance. For 3 heads, it’s about 31%. In the widget below, you can choose the number of experiments being performed and see the effect it has on the graph.



## Bernoulli experiments to model room bookings

If you have 100 hotel rooms, in an ideal world, you have all of your 100 rooms booked every night. However, a room booking still holds uncertainty in itself. When someone booked a room at the hotel, the hotel guest showing up is still not 100% certain. In that sense, the “guest showing up” is a probability event at itself, except (hopefully!) with a much higher chance than 50% in the event of a coin toss. in our hotel, on average, 8% of the customers who book a room don’t end up staying on a given night.

Let’s check in Python what we can expect in terms of room bookings. We’ll use NumPy again, and run the “experiment” so we can create a plot like the one before.



As you can see in the plot above, if you select 100,000, the probability that exactly all guests who booked rooms showing up is virtually 0. Out of the 100,000 experiments we ran, only 20 times all 100 rooms are booked (you can’t even see that with the naked eye!). This is one night out of every ~14 years!

# **CALCULATING PROFIT ON ANY GIVEN NIGHT**

From what you’ve seen before, it looks like overbooking your hotel seems like a potential solution to use the full capacity of your hotel in an optimal way. But how many rooms should you overbook? The answer depends on the cost structure of overbooking.

For our hotel, we have the following information:

* Our customers pay a price of $220 USD per night for our hotel.
* If on any night we have fewer rooms available than hotel guests, we have to place customers at a competitor hotel, which costs us $400 USD.

This means that we end up losing $180 USD per room that is overbooked, when more guests show up than number of rooms available.

The total profit on any given night is equal to:



Let’s say that in total 104 guests booked a room on a given night, and only 99 show up. Our expected profit that night is then equal to:



Note how the hotel benefits from overbooking because the customers fully prepay!

On the other hand, if we overbooked and 104 guests and all of them show up on one given night, the expected is then equal to:



**Expected Profit when overbooking at 104 rooms**

Previously, you saw how overbooking at 104 rooms can give very different profits depending on how many guests actually end up staying in the hotel. Overbooking is a great tool to make more money, but can end up being costly if more hotel guests end up staying than rooms are available.

In this section, we’ll look into what our hotel’s expected profit is when overbooking 104 rooms.

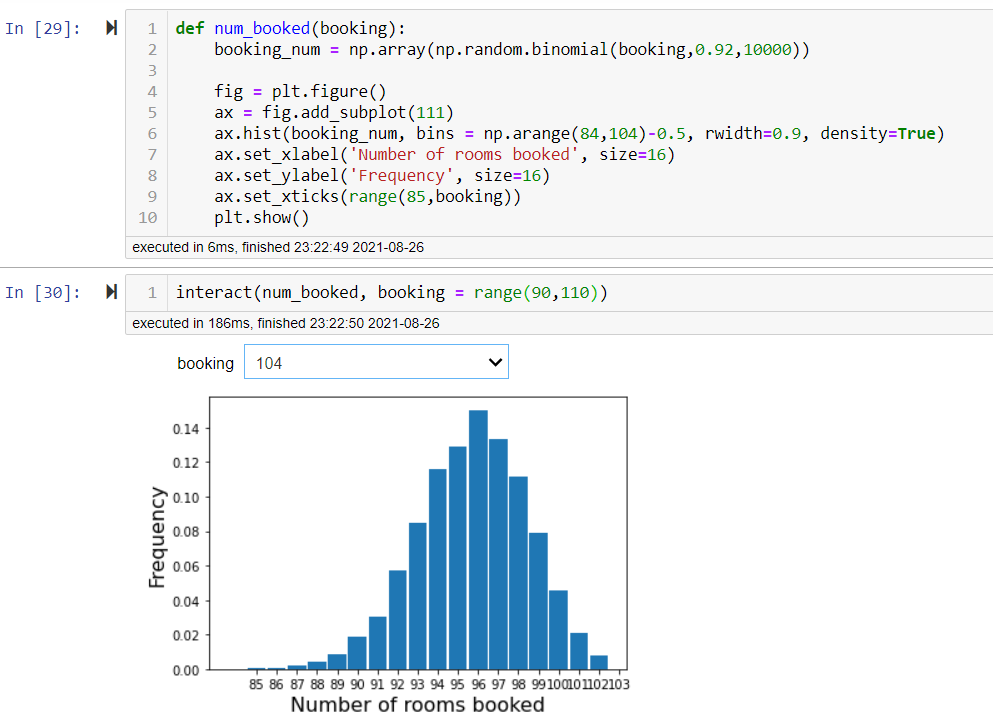
We can achieve getting the expected profit by taking the following 3 steps:

* Run a Bernoulli experiment like you did before, this time with n=104 instead of n=100. Let’s run the experiment 10,000 times.
* For each of the potential outcomes, we’ll calculate the profit.
* Next, we’ll calculate the *expected profit*. You’ll learn how to do that in a bit!

### **1) The Bernoulli Experiment when booking 104 rooms**

Let’s run a Bernoulli experiment of what happens if we allow 104 rooms to be booked, knowing that on average, there is a 92% “success rate”, in other words, 8% of hotel customers don’t end up staying at the hotel. We’ll run the experiment 10,000 times.

Let’s create a plot to analyze what we’re seeing. Use the dropdown in the widget below to see how the number of bookings can affect the outcome.



As you can see here, the majority of the nights, around 94 to 98 rooms end up being occupied. Next, let's create a similar plot but looking at the profit on the y-axis. We'll have to use some of the financial information to get there, but we'll explain the code step by step

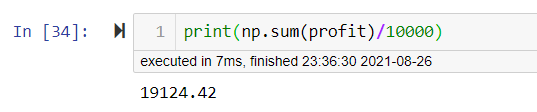
### **2) The Profit Distribution when booking 104 rooms**

Next, let’s use our array that stores 10,000 random experiments of 104 room bookings and eventual turnout, and let’s use that array to calculate the profit that the hotel would make in each of the 10,000 cases. To get there, we need to calculate:

* The revenue
* The cost
* Profit = Revenue - Cost

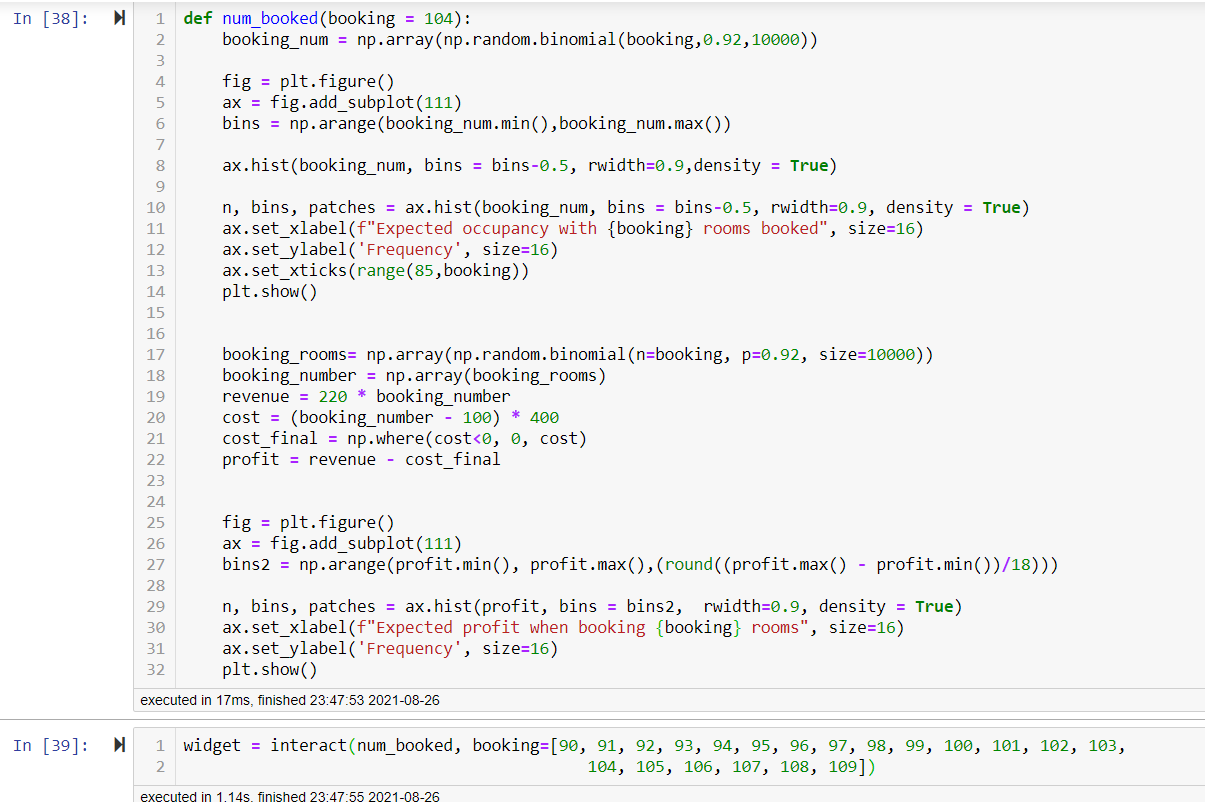
### **3) The Expected Profit**

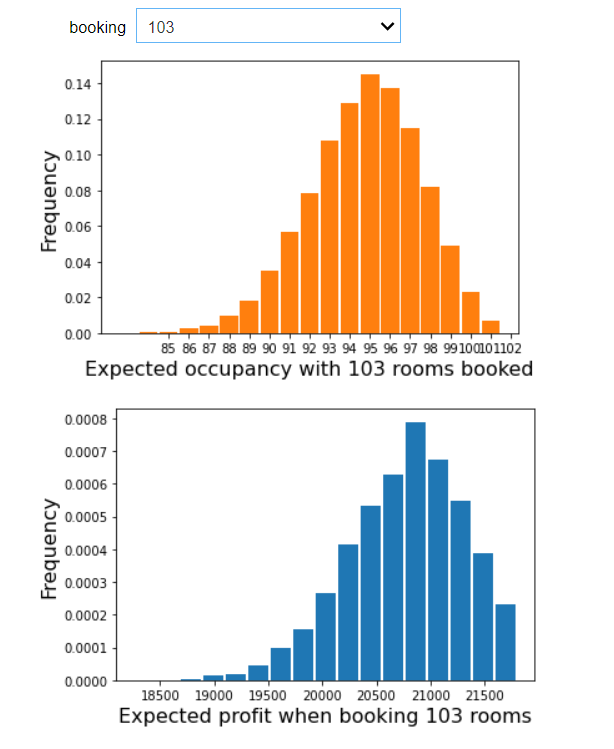
Now what is our actual “expected profit”? The answer is not complicated: you can simply take the average of all the entire profit distribution, so you’ll sum up all 10,000 elements in your array, and divide them by 10,000!



The expected profit when booking 104 rooms on any given night is around $21,035 USD!

Let’s plot the resulting “profit distribution”





## Using Python to define Optimal numbers of rooms booked

JNB