Calculating Ï€ on Pi Day

Author: Eric Silberstein

Claps: 430

Date: Mar 14

Happy Pi Day! I posted to our data science channel asking for smart, dumb, serious, funny, creative, or boring ways to estimate Ï€. Hereâ€™s what came back:

Vinicius Aurichio

```
(6*sum(1/n**2 for n in range(1, 1001)))**(0.5)
```

```
=> 3.1406380562059946
```

Computing the sum of the inverse of the square of all natural numbers is known as the Basel problem. The exact result is piÂ²/6 and can be obtained in a variety of ways (this is my favorite). We can approximate pi by computing a partial sum and solving for pi from it.

Eric Silberstein

```
import numpy as np
delta = 1e-9
4 * delta * np.sum((1 - np.arange(0, 1, delta) ** 2) ** 0.5)
```

```
=> 3.1415926555897618
```

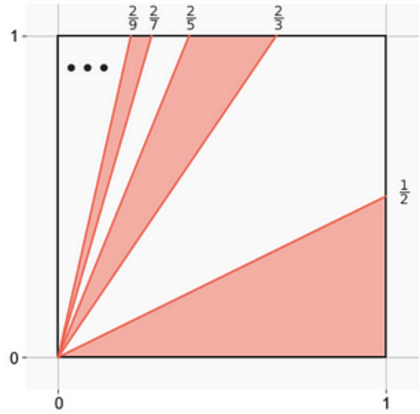
Center a circle of radius one at the origin. Calculate the area of the quarter of the circle in the top right quadrant by summing the area of lots of rectangles of width delta and height calculated using the pythagorean theorem. That area should be pi/4.

Tom Dinitz

```
import numpy as np
n = 100_000_000
5 - 4 * np.mean(np.round(np.random.rand(n)/np.random.rand(n)) % 2 == 0)
```

```
=> 3.14152348
```

Take a random point in the positive 1x1 square, and round the ratio of its coordinates. The probability that you get an even number is $\frac{5-\pi}{4}$. To compute this, note that we must have either $0 \leq \frac{y}{x} \leq \frac{1}{2}$, or $2n - \frac{1}{2} \leq \frac{y}{x} \leq 2n + \frac{1}{2}$, for some n >= 1. In other words, our point has to be between the x-axis and $y = \frac{1}{2}x$, or between the lines $y = (2n - \frac{1}{2})x$ and $y = (2n + \frac{1}{2})x$:



This latter category corresponds to triangles with height 1 whose bases stretch between $\frac{2}{4n+1}$ and $\frac{2}{4n-1}$. The area of these triangles can be summed using the Leibniz formula for pi/4:

$$\frac{1}{2}\left(\left(\frac{2}{3} - \frac{2}{5}\right) + \left(\frac{2}{7} - \frac{2}{9}\right) + \ldots\right) = \left(\frac{1}{3} - \frac{1}{5}\right) + \left(\frac{1}{7} - \frac{1}{9}\right) + \ldots = 1 - \frac{\pi}{4}$$

Thus, adding in the triangle between the x-axis and $y = \frac{1}{2}x$, we get the total area (i.e. the probability we were after): $\frac{1}{4} + (1 - \frac{\pi}{4}) = \frac{5-\pi}{4}$.

Sofiane Hadji

```
M = 1_000_000
2 * np.product([4 * (k**2) / (4 * (k**2)-1) for k in range(1,M)])

=> 3.1415918681913633
```

This approximation comes from the Wallis integrals

$I_n = \int_0^{\pi/2} sin^n x\, dx$, a family of integrals which follows the following recurrence relation:

- $I_0 = \pi/2$
- $I_1 = 1$
- $I_n = \frac{n-1}{n} I_{n-2}$ for $n \geq 2$ (proof by integration by parts)

We can then compute the value of the integral for odd and even values of n:

$I_{2n} = \pi \prod_{p=1}^{n} \frac{2p-1}{2p}$ and $I_{2n+1} = 2 \prod_{p=1}^{n} \frac{2p}{2p+1}$.

(I_n) is a decreasing sequence (check $I_n - I_{n+1}$), so we can write $I_{2n+1} \leq I_{2n} \leq I_{2n-1}$ and $1 \leq \frac{I_{2n}}{I_{2n+1}} \leq \frac{I_{2n-1}}{I_{2n+1}} = \frac{2n+1}{2n}$

Therefore $\lim_{n \to +\infty} \frac{I_{2n}}{I_{2n+1}} = 1$. Rewriting this quotient, we end up with

$\lim_{n \to +\infty} \prod_{p=1}^{n} \frac{4p^2}{4p^2-1} = \frac{\pi}{2}$

Eric Silberstein

```
import numpy as np
n = 100_000_000
4 * np.sum((np.random.rand(n) ** 2 + np.random.rand(n) ** 2) ** 0.5 < 1) /

=> 3.14188724
```

Throw darts at the 1x1 square in the positive-X, positive-Y quadrant sitting at the origin. Use the pythagorean theorem to see how many land within the circle of radius one centered at the origin. Since the area of that fourth of the circle is pi/4, and the area of the square is 1, the ratio of darts in the circle to all darts should be pi/4.

ChatGPT (courtesy of Conor Dowling)

C Print the first 1000 digits of pi using python

Here is a python script that calculates and prints the first 1000 digits of pi using the Bailey–Borwein–Plouffe formula:

Copy code

```
import decimal

# Set the precision to 1000 digits
decimal.getcontext().prec = 1000

# The Bailey-Borwein-Plouffe formula for pi
# See
https://en.wikipedia.org/wiki/Bailey%E2%80%93Borwein%E2%80%93Plouffe_formula
def bbp_formula(k):
    decimal.getcontext().prec += 2
    pi = 0
    for k in range(0, 1000):
        pi += (1 / decimal.Decimal(16)**k) * (
            decimal.Decimal(4) / (8 * k + 1)
            - decimal.Decimal(2) / (8 * k + 4)
            - decimal.Decimal(1) / (8 * k + 5)
            - decimal.Decimal(1) / (8 * k + 6)
        )
    return +pi

# Calculate and print the first 1000 digits of pi
pi = bbp_formula(1000)
print(pi)
```

This script uses the `decimal` module to set the precision of the calculation to 1000 digits and then applies the Bailey–Borwein–Plouffe formula to calculate the value of pi to that precision. The result is then printed to the console.

It prints 1002 digits and the last 2 are wrong but in fairness I only asked for 1000.

Tristan Mills

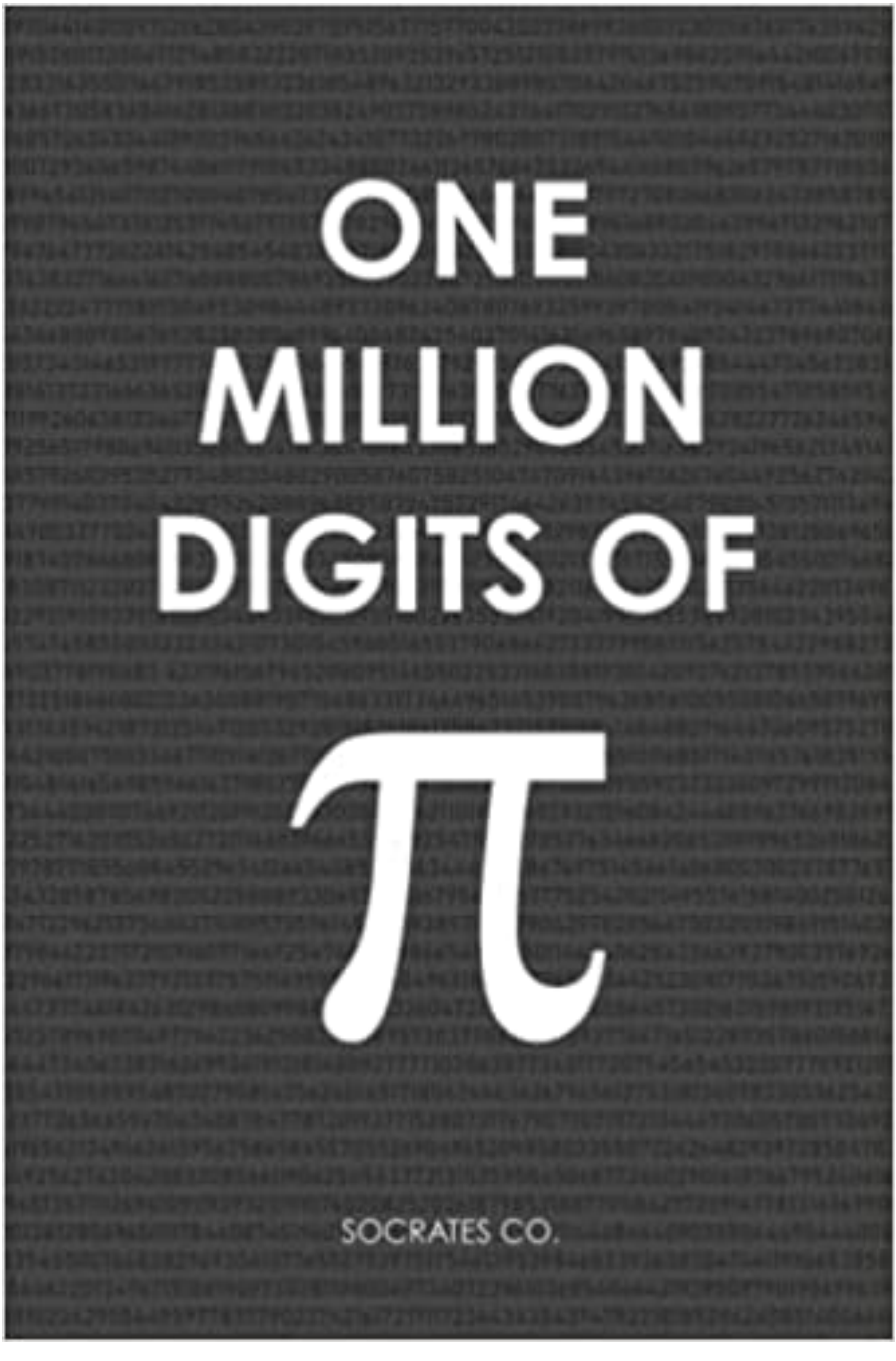
```
m1 = 1.0
m2 = 10_000_000_000
m_sum = (m1 + m2)
mu = (m1 - m2)/m_sum

V1, v2 = 0., 1.
```

```
collisions = 0

while not (v1 <= 0 and v2 < 0 and abs(v1) <= abs(v2)):
   collisions += 1
   if collisions % 2:
       v1, v2 = mu * v1 + 2*m2/m_sum*v2, -mu * v2 + 2*m1/m_sum*v1
   else:
       v1 = -v1

print(collisions)

=> 314159
```

The above script simulates two blocks sliding on an infinite frictionless plane. The first mass is at rest, and the second is moving towards mass and is ten-billion times as massive. They experience perfect elastic collisions between themselves. On the far side of block one is a wall against which block one will also bounce against elastically. (Momentum and kinetic energy are conserved.) By counting the number of collisions between the blocks, but also block one against the wall as block 2â€™s momentum is slowly turned around, we â€œcountâ€� pi without its decimal places. (You can increase the mass difference between the masses, but it needs to be 100^n where n is a positive integer. Also, some major shortcuts have been taken to shorten the script.)

Christina Dedrick

Lookup table. Store the value of Ï€ in some type of memory. Ask for it when needed. Consider buying this book if you find yourself without a computer. A printout of the first 10 digits kept in your wallet will also get you pretty far.



In the same spirit, you can ask someone.

Achu Balasubramanian

```
def calculate_pi(n, i=1):
    return n if i == n else (2 * i - 1) + (i ** 2 / calculate_pi(n, i +1))

4 / calculate_pi(24)

=> 3.141592653589793
```

(I chose n=24 since it was the smallest n where 4/calculate_pi(n) == math.pi returned True). There are some other fun continued fractions (I chose to implement the right-most one):

$$\pi = 3 + \cfrac{1^2}{6 + \cfrac{3^2}{6 + \cfrac{5^2}{6 + \cfrac{7^2}{6 + \cfrac{9^2}{6 + \ddots}}}}} = \cfrac{4}{1 + \cfrac{1^2}{2 + \cfrac{3^2}{2 + \cfrac{5^2}{2 + \cfrac{7^2}{2 + \ddots}}}}} = \cfrac{4}{1 + \cfrac{1^2}{3 + \cfrac{2^2}{5 + \cfrac{3^2}{7 + \cfrac{4^2}{9 + \ddots}}}}}$$

Side note: Ramanujan's Pi approximation is probably my favorite because of just how crazy it is:

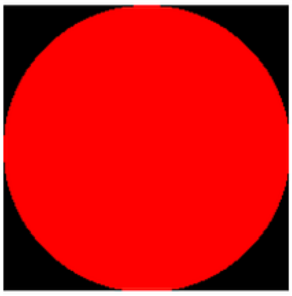
$$\frac{1}{\pi} = \frac{2\sqrt{2}}{99^2} \sum_{k=0}^{\infty} \frac{(4k)!}{k!^4} \frac{26390k + 1103}{396^{4k}}$$

Charlie Natoli

In [2]:
```
import numpy as np
import time
import pandas as pd
from PIL import Image, ImageDraw
```

In [4]:
```
# create an image of a circle
n = 200
im = Image.new(mode="RGB", size=(n, n))
draw = ImageDraw.Draw(im)
draw.pieslice(xy=((0,0),(n,n)), fill='red', start=0,end=360)
im
```

Out[4]:



In [6]:
```
# count up how many pixels there are of each color
pixel_color_counts = pd.array(im.getdata()).value_counts()
pixel_color_counts
```

Out[6]:
```
(0, 0, 0)        8349
(255, 0, 0)     31651
dtype: int64
```

In [7]:
```
# the ratio of a circle's area to a square bounding it is pi / 4.
#    We can compare the number of pixels in the circle's color (circle area)
#    to the number of pixels overall (bounding square area) to appoximate pi
circle_body_color = max(pixel_color_counts)
total_pixels = sum(pixel_color_counts)
4 * circle_body_color / total_pixels
```

Out[7]:
```
3.1651
```

```python
# does it seem to converge on pi if we use larger and larger images? Yes.
# does it scale well? No! Very slow and O(N^2).

def approx_pi_from_image(n):

    im = Image.new(mode="RGB", size=(n, n))
    draw = ImageDraw.Draw(im)
    draw.pieslice(xy=((0,0),(n,n)), fill='red', start=0,end=360)

    pixel_color_counts = pd.array(im.getdata()).value_counts()

    circle_body_color = max(pixel_color_counts)
    total_pixels = sum(pixel_color_counts)

    return 4 * circle_body_color / total_pixels

for n in [100, 1000, 10000, 20000]:
    start = time.time()
    pi = approx_pi_from_image(n)
    time_elapsed = time.time() - start
    print(f'n: {n}, pi; {pi}, seconds elapsed: {time_elapsed:.3f}')
```

```
n: 100, pi; 3.19, seconds elapsed: 0.011
n: 1000, pi; 3.146924, seconds elapsed: 1.046
n: 10000, pi; 3.14215148, seconds elapsed: 101.743
n: 20000, pi; 3.14187167, seconds elapsed: 400.441
```

Nick Hartmann

```python
from getch import getch
from mpmath import mp, pi
mp.dps = 1000

starting_prompt = "Guess the digits of pi!  What comes next after 3.14"
starting_digit = 4

current_prompt = starting_prompt
current_digit = starting_digit
high_score = 0
while True:
    print(current_prompt)
    target_digit = str(pi)[current_digit]
    guessed_digit = getch()
    if guessed_digit == target_digit:
        current_prompt += guessed_digit
        current_digit += 1
    else:
        if current_digit > high_score:
            high_score = current_digit
        print(f"""
              Incorrect!  The next digit was {target_digit}.
              You guessed {current_digit-2} digits after the decimal point
              Your high score is {high_score-2}.

              Press 'y' to play again or any other key to exit
```

```
        """)
    if getch().lower() == "y":
        current_prompt = starting_prompt
        current_digit = starting_digit
    else:
        break
```

The â€œguess and checkâ€� method. Run the code. Youâ€™ll be asked to enter the digits of pi one by one. If you make a mistake, youâ€™ll be sent back to the beginning, but python will tell you what the correct digit was, and you can try again. Your high score will be tracked.

Wayne Coburn

Watch this:

Michael Lawson

```
import numpy as np
from math import pi
from sklearn.linear_model import LinearRegression

MAX_RADIUS = 100
MIN_RADIUS = 0.1
NUM_CIRCLES_TO_MEASURE = 200

# set random number seed for replicability
np.random.seed(seed=314)

# calculate the true values of radius-squared and area
radii = np.random.uniform(low=MIN_RADIUS, high=MAX_RADIUS, size=NUM_CIRCLE
radii_squared = np.square(radii)
areas = pi * radii_squared

# add Gaussian noise to areas (simulating measurement area)
areas_with_measurement_error = areas + np.random.normal(loc=0, scale=15)

# fit linear regression to find value of pi
radii_squared_preprocessed_array = radii_squared.reshape(-1, 1)
linear_model = LinearRegression()
linear_model.fit(X=radii_squared_preprocessed_array, y=areas_with_measurem
pi_derived = linear_model.coef_
print("The value of pi, calculated via linear regression assuming Gaussian

=> The value of pi, calculated via linear regression assuming Gaussian err
```

Motivation: Sometimes we have to deal with measurement error in real data. Suppose you have a bunch of circles of different radii, each of which you know, but you donâ€™t know each circleâ€™s area. Youâ€™re able to measure these circlesâ€™ area, but your area-measuring tool has measurement error. Based on previous times youâ€™ve used the tool, youâ€™re reasonably confident this measurement error is mean-zero and near Gaussian. In this situation, linear regression is a great approach to estimate pi â€” it can help you estimate the coefficient between radius-squared and area, after accounting for noise!

Michael Lawson

```
from math import pi

def point_estimator_of_real_number(num):
   return 3

estimated_pi = point_estimator_of_real_number(pi)

print("The value of pi estimated by the Lawsonian 3-Estimator is", estimat

=> The value of pi estimated by the Lawsonian 3-Estimator is 3
```

This method of calculating pi hearkens back to one of the most important lessons I learned in inferential statistics. Early in that class, our professor, Michael Kosorok, asked us to define an estimator. He let us throw out definitions for a few minutes, then walked to the board and wrote out a simple function: â€œT(X) = 3â€� â€” the function T of the data X always takes the value 3. This, he explained, is an estimator â€” itâ€™s a function of the data that returns a well-defined value. Itâ€™s just that most of the time, itâ€™s a very bad estimator. Inferential statistics exists precisely because, unless you understand the properties that make a statistical estimator well-behaved, it might just be the Kosorok 3-Estimator under the hood for all you know.

Iâ€™ve applied that idea to this particular estimation problem. And hey, what do you know â€” the Lawsonian 3-Estimator is actually pretty good at calculating pi!

Just donâ€™t ask it to calculate 2*pi.

David Lustig

One of my favorite ways to approximate pi comes from Buffonâ€™s needle problem. As an abridged history we can imagine that Buffon is frantically dropping needles (or matches or a baguette or something one dimensional) of length (*l)* on a floor with parallel lines a set width apart (t) and he wants to estimate the probability of the needle crossing a line. He runs out of patience and instead decides to publish a paper posing this question and a solution using integral geometry instead. The proof is of medium length and uses some sensible math so we take his word that the solution to this problem is:

p =2/Ï€ * l/t

Unlike Buffon we have computers that never get bored of dropping matches, so we can simulate as follows:

A) Import packages and make a â€œfloor:â€�

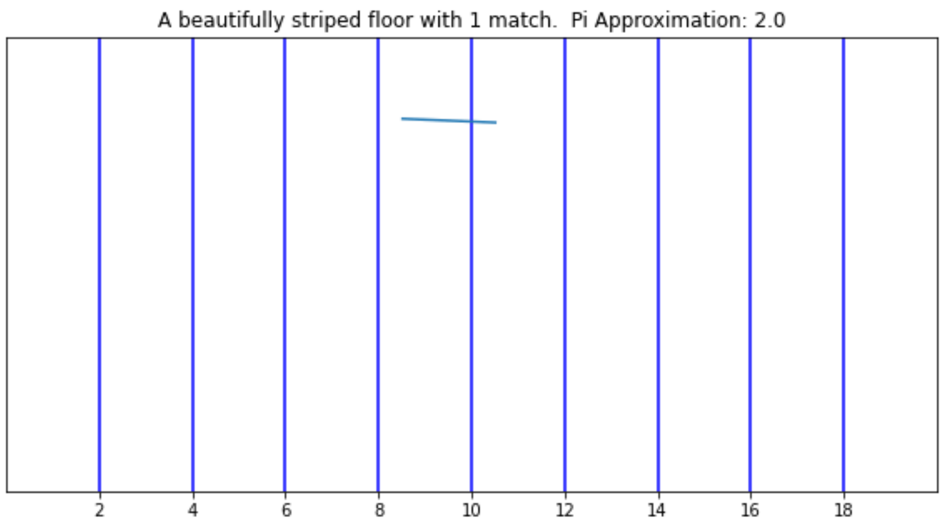
```
import cmath
import math
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from IPython.display import display

fig = plt.figure()
ax = plt.subplot()
ax.set_ylim([0, 20])
ax.set_xlim([0, 20])
for i in range(2, 20, 2):
    plt.axvline(x=i, color='blue')
ax.set_xticks(range(2, 20, 2))
ax.set_yticks([])
plt.title('A beautifully striped floor')
```

B) Drop matches at random and check if they cross a line:

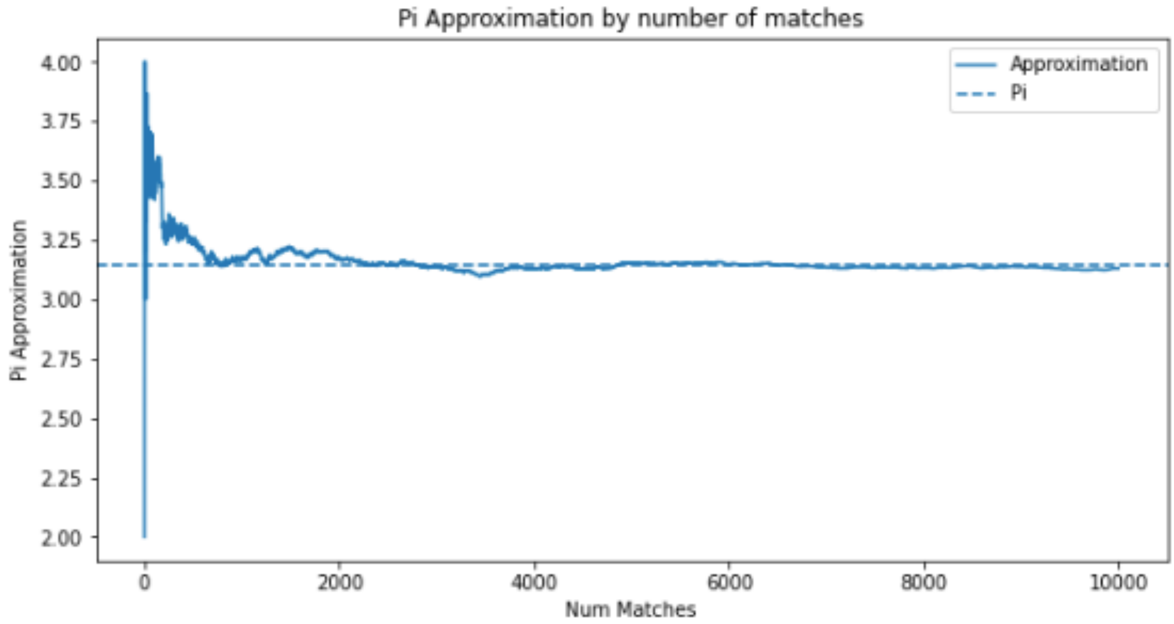
```
def drop_match(x_bounds, y_bounds, match_length, ax):
    x_start = np.random.uniform(x_bounds[0], x_bounds[1])
    y_start = np.random.uniform(y_bounds[0], y_bounds[1])
    pt = cmath.rect(match_length, math.radians(np.random.uniform(0, 360)))
    x_end = pt.real + x_start
    y_end = pt.imag + y_start
    ax.plot([x_start, x_end], [y_start, y_end])
    return [x_start, x_end]

def check_match(xcoords, lines):
    inside = 0
    for line in lines:
        if min(xcoords) < line and max(xcoords) > line:
            inside = 1
```

```
    return inside

fig = plt.figure(figsize=(10,5))
fig.set_facecolor('white')
ax = plt.subplot()
ax.set_ylim([0, 20])
ax.set_xlim([0, 20])
for i in range(2, 20, 2):
    plt.axvline(x=i, color='blue')
ax.set_xticks(range(2, 20, 2))
ax.set_yticks([])
match_counter = 0
match_cross_line_counter = 0
pi_approximations = []
for i in range(10000):
    coords = drop_match([2, 18], [2,18], 2, ax)
    match_counter += 1
    match_cross_line_counter += check_match(coords, range(2, 20, 2))
    pi_approx = 2/(match_cross_line_counter/match_counter)
    pi_approximations.append(pi_approx)
plt.title(f'A beautifully striped floor with {match_counter} matches.  Pi
display(fig)
```

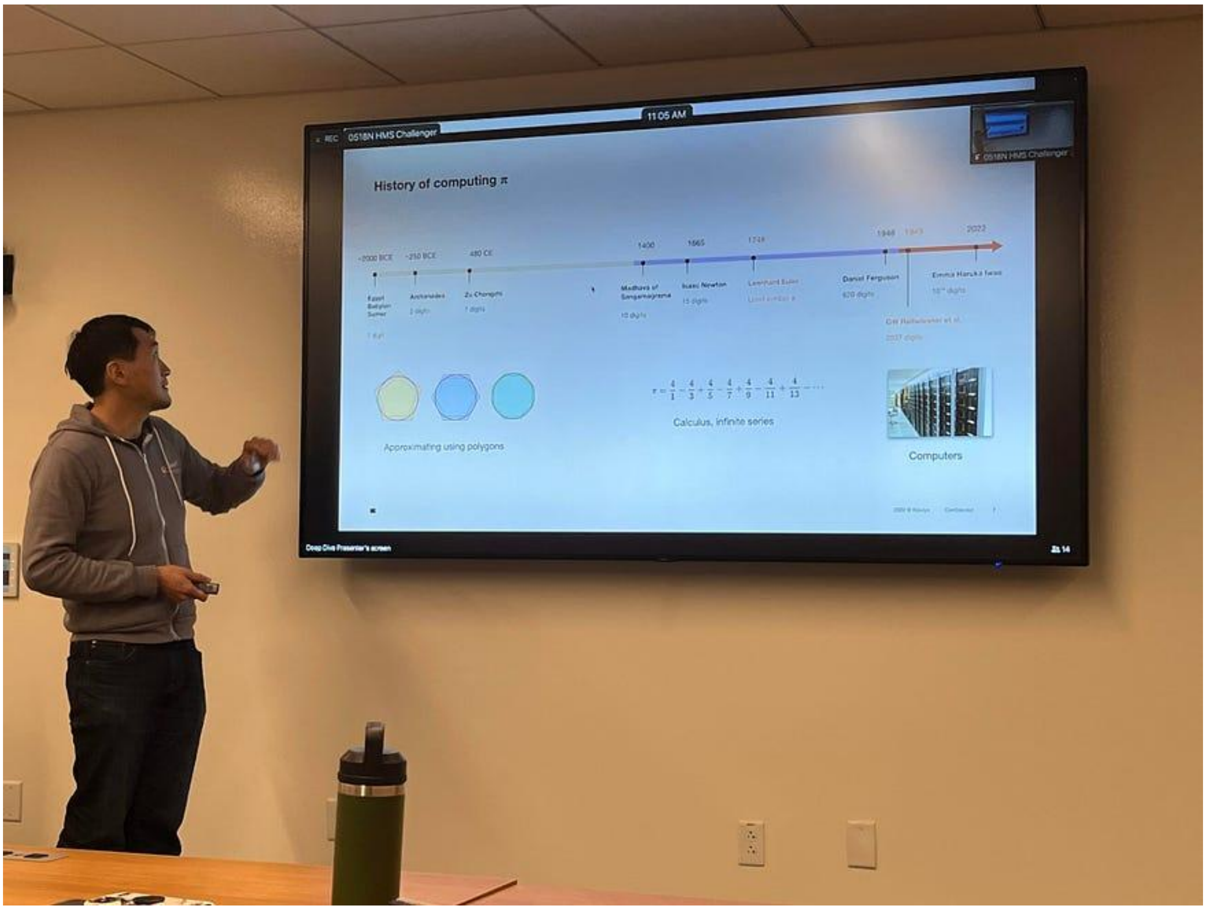


C) Look at how cool it is that our approximation approaches pi:

```
fig = plt.figure(figsize=(10,5))
sns.lineplot(x=range(len(pi_approximations)), y=pi_approximations)
plt.xlabel('Num Matches')
plt.ylabel('Pi Approximation')
plt.axhline(np.pi, linestyle = '--')
plt.title('Pi Approximation by number of matches')
plt.legend(['Approximation', 'Pi'])
```



After 10,000 match drops we have Ï€ â‰ˆ 3.1308703819661865



Dave Xiao talking about Pi and Data Science on Pi Day