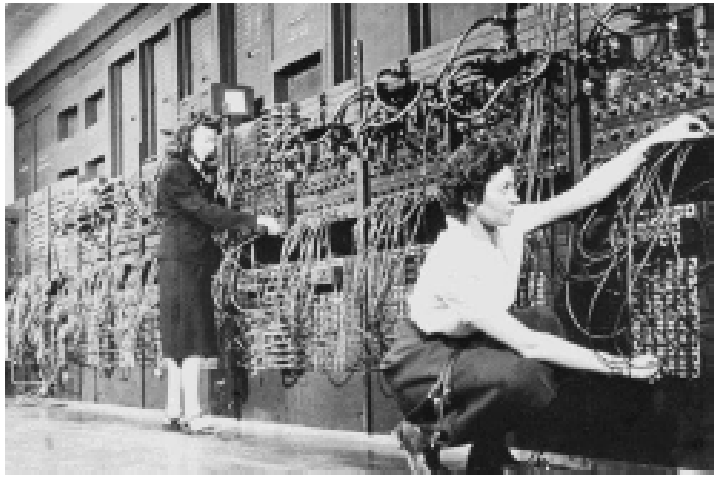
Monte Carlo Simulation

Lesson 8

A Little History

- Ulam, recovering from an illness, was playing a lot of solitaire
- Tried to figure out probability of winning, and failed
- Thought about playing lots of hands and counting number of wins, but decided it would take years
- Asked Von Neumann if he could build a program to simulate many hands on ENIAC



Monte Carlo Simulation

- A method of estimating the value of an unknown quantity using the principles of inferential statistics
- Inferential statistics
  - *Population*: a set of examples
  - *Sample*: a proper subset of a population
  - Key fact: a *random sample* tends to exhibit the same properties as the population from which it is drawn
- Exactly what we did with random walks

An Example

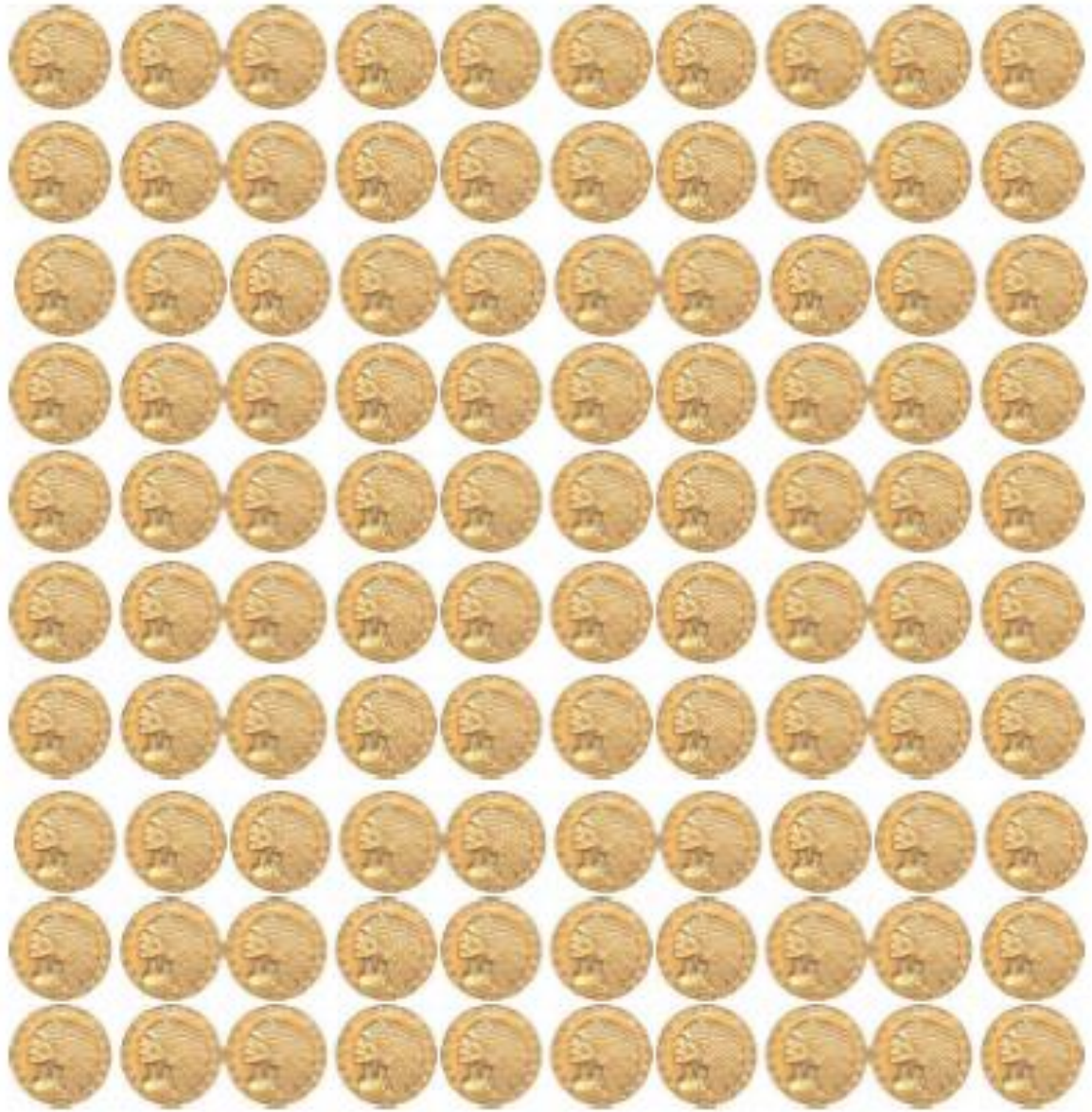
- Given a single coin, estimate fraction of heads you would get if you flipped the coin an infinite number of times
- Consider one flip

How confident would you be about answering 1.0?

Flipping a Coin Twice

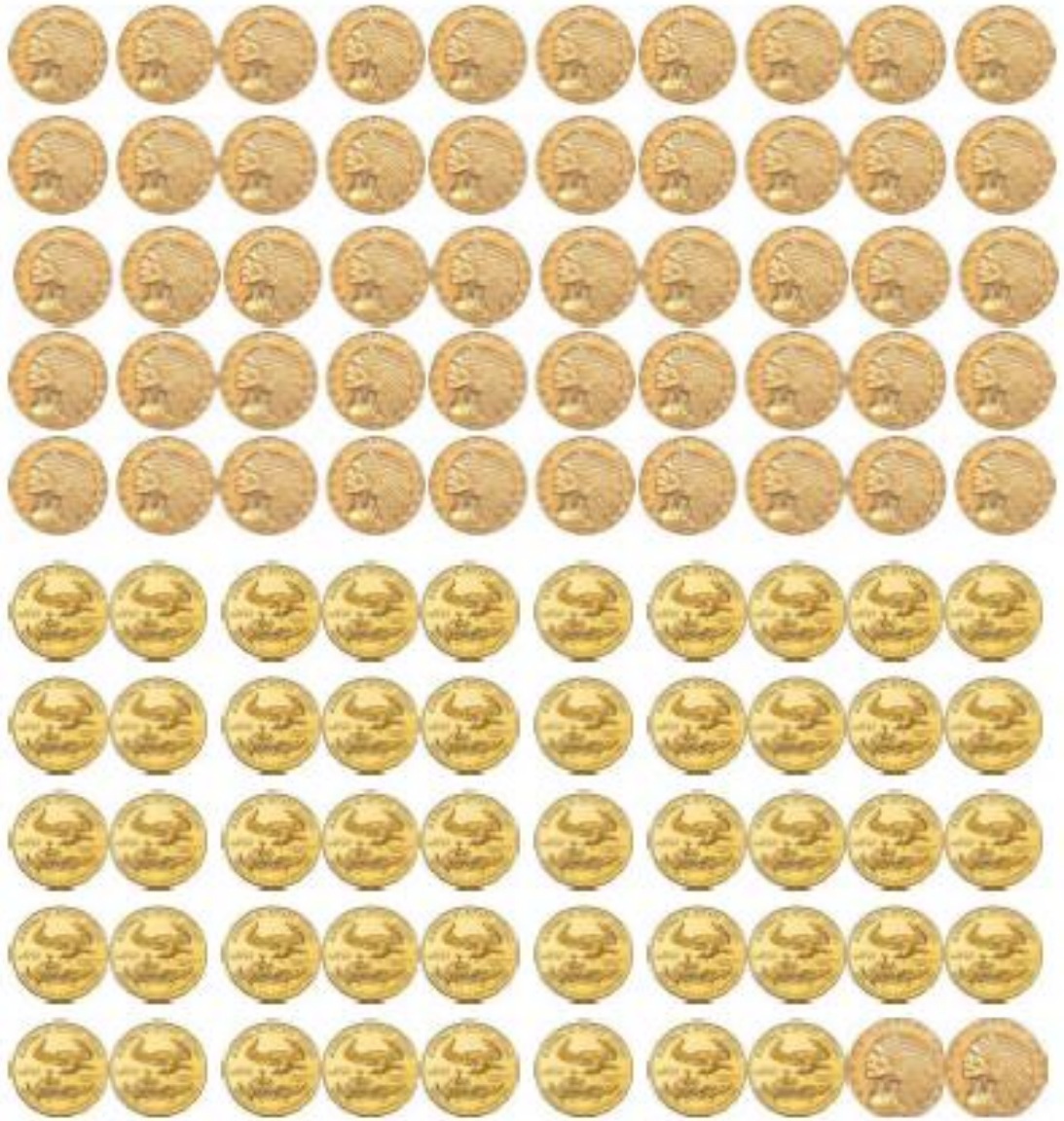
Do you think that the next flip will come up heads?

Flipping a Coin 100 Times



Now do you think that the next flip will come up heads?

Flipping a Coin 100 Times



Do you think that the probability of the next flip coming up heads is 52/100?

Given the data, it's your best estimate

But confidence should be low

Why the Difference in Confidence?

- Confidence in our estimate depends upon two things
- Size of sample (e.g., 100 versus 2)
- Variance of sample (e.g., all heads versus 52 heads)
- As the variance grows, we need larger samples to have the same degree of confidence

Problem

A friend asked you whether or not it would be profitable to bet within twenty-four rolls of a pair of dice he would roll a double 6.

Solve by hand??? By simulation??? Both???

Roulette

No need to simulate, since answers obvious

Allows us to compare simulation results to actual probabilities

```
import random

class FairRoulette():
    def __init__(self):
        self.pockets = []
        for i in range(1,37):
            self.pockets.append(i)
        self.ball = None
        self.pocketOdds = len(self.pockets) - 1
    def spin(self):
        self.ball = random.choice(self.pockets)
    def betPocket(self, pocket, amt):
        if str(pocket) == str(self.ball):
            return amt*self.pocketOdds
        else: return -amt
    def __str__(self):
        return 'Fair Roulette'
```

How to play it ?

```
game = FairRoulette()
game.spin()
amt = game.betPocket(5,10)
print(amt)
```

```
for i in range(1000):
    game = FairRoulette()
    game.spin()
    amt = game.betPocket(5,10)
    print(amt)
```

```
def playRoulette(game, numSpins, pocket, bet, toPrint):
    totPocket = 0
    for i in range(numSpins):
        game.spin()
        totPocket += game.betPocket(pocket, bet)
    if toPrint:
        print(numSpins, 'spins of', game)
        print('Expected return betting', pocket, '=',\
              str(100*totPocket/numSpins) + '%\n')
    return (totPocket/numSpins)


random.seed(0)
game = FairRoulette()
for numSpins in (100, 1000000):
    for i in range(3):
        playRoulette(game, numSpins, 2, 1, True)
```

100 and 1M Spins of the Wheel

```
100 spins of Fair Roulette
Expected return betting 2 = -100.0%

100 spins of Fair Roulette
Expected return betting 2 = 44.0%

100 spins of Fair Roulette
Expected return betting 2 = -28.0%

1000000 spins of Fair Roulette
Expected return betting 2 = -0.046%

1000000 spins of Fair Roulette
Expected return betting 2 = 0.602%

1000000 spins of Fair Roulette
Expected return betting 2 = 0.7964%
```

Law of Large Numbers

- In repeated independent tests with the same actual probability *p* of a particular outcome in each test, the chance that the fraction of times that outcome occurs differs from *p* converges to zero as the number of trials goes to infinity

Does this imply that if deviations from expected behavior occur, these deviations are likely to be ***evened out*** by opposite deviations in the future?

Gambler's Fallacy

- "On August 18, 1913, at the casino in Monte Carlo, black came up a record twenty-six times in succession [in roulette]. … [There] was a near-panicky rush to bet on red, beginning about the time black had come up a phenomenal fifteen times." -- Huff and Geis, *How to Take a Chance*
- Probability of 26 consecutive reds
- 1/67,108,865
- Probability of 26 consecutive reds when previous 25 rolls were red
- 1/2

Regression to the Mean

- Following an extreme random event, the next random event is likely to be less extreme
- If you spin a fair roulette wheel 10 times and get 100% reds, that is an extreme event (probability = 1/1024)
- It is likely that in the next 10 spins, you will get fewer than 10 reds
  - But the expected number is only 5
- So, if you look at the average of the 20 spins, it will be closer to the expected mean of 50% reds than to the 100% of the first 10 spins

Casinos Not in the Business of Being Fair

Two Subclasses of Roulette

```
class EuRoulette(FairRoulette):
    def __init__(self):
        FairRoulette.__init__(self)
        self.pockets.append('0')
    def __str__(self):
        return 'European Roulette'

class AmRoulette(EuRoulette):
    def __init__(self):
        EuRoulette.__init__(self)
        self.pockets.append('00')
    def __str__(self):
        return 'American Roulette'
```

```
random.seed(0)
numTrials = 20
resultDict = {}
games = (FairRoulette, EuRoulette, AmRoulette)
for G in games:
    resultDict[G().__str__()] = []
for numSpins in (1000, 10000, 100000, 1000000):
    print('\nSimulate', numTrials, 'trials of',
          numSpins, 'spins each')
    for G in games:
        pocketReturns = findPocketReturn(G(), numTrials,
                                         numSpins, False)
        expReturn = 100*sum(pocketReturns)/len(pocketReturns)
        print('Exp. return for', G(), '=',
              str(round(expReturn, 4)) + '%')
```

Comparing the Games

```
Simulate 20 trials of 1000 spins each
Exp. return for Fair Roulette = 6.56%
Exp. return for European Roulette = -2.26%
Exp. return for American Roulette = -8.92%

Simulate 20 trials of 10000 spins each
Exp. return for Fair Roulette = -1.234%
Exp. return for European Roulette = -4.168%
Exp. return for American Roulette = -5.752%

Simulate 20 trials of 100000 spins each
Exp. return for Fair Roulette = 0.8144%
Exp. return for European Roulette = -2.6506%
Exp. return for American Roulette = -5.113%

Simulate 20 trials of 1000000 spins each
Exp. return for Fair Roulette = -0.0723%
Exp. return for European Roulette = -2.7329%
Exp. return for American Roulette = -5.212%
```

Sampling Space of Possible Outcomes

- Never possible to guarantee perfect accuracy through sampling
- Not to say that an estimate is not precisely correct
- Key question:
  - How many samples do we need to look at before we can have justified confidence on our answer?
- Depends upon variability in underlying distribution

Quantifying Variation in Data

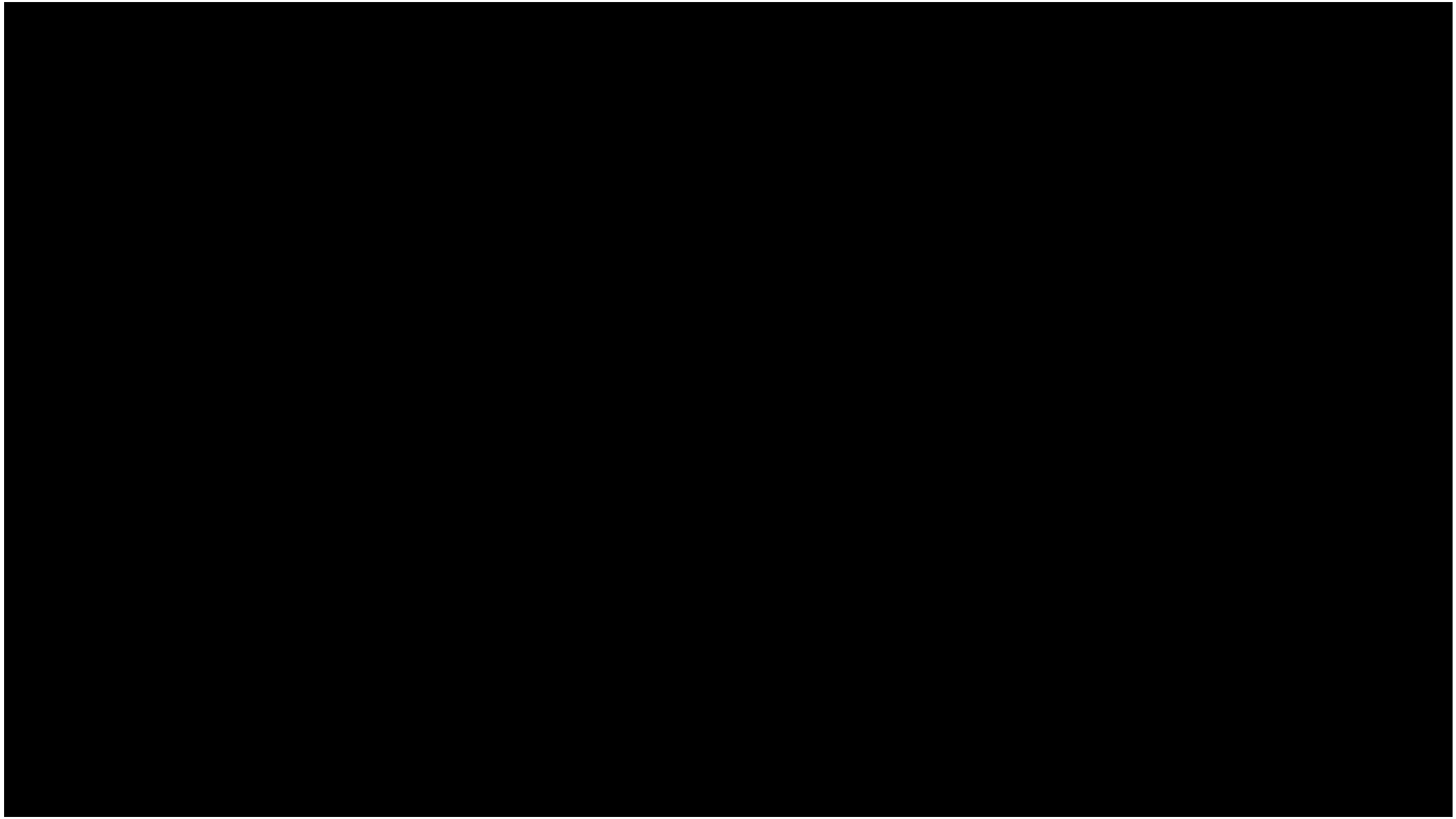
$$variance(X) = \frac{\sum_{x \epsilon X}(x-\mu)^2}{|X|}$$

$$\sigma(X) = \sqrt{\frac{1}{|X|}\sum_{x \in X}(x-\mu)^2}$$

- Standard deviation simply the square root of the variance
- Outliers can have a big effect
- Standard deviation should always be considered relative to mean

For Those Who Prefer Code

```
def getMeanAndStd(X):
    mean = sum(X)/float(len(X))
    tot = 0.0
    for x in X:
        tot += (x - mean)**2
    std = (tot/len(X))**0.5
    return mean, std
```



Lesson 08b

- Inferential Statistics 03 https://colab.research.google.com/drive/158_PpqxjmHarjQRX00p9mBqxpn1e0aFP?usp=sharing
- Monte Carlo Simulation https://colab.research.google.com/drive/18yTS1pKa5k2QVUxQc607NMcW-kZe4pR7?usp=sharing
- Assignment (Monte Carlo simulation): Submit .ipynb file on IVY
- Assignment 06 is out in IVY
- Take-home Quiz 03 (due on 11 April 17:00)