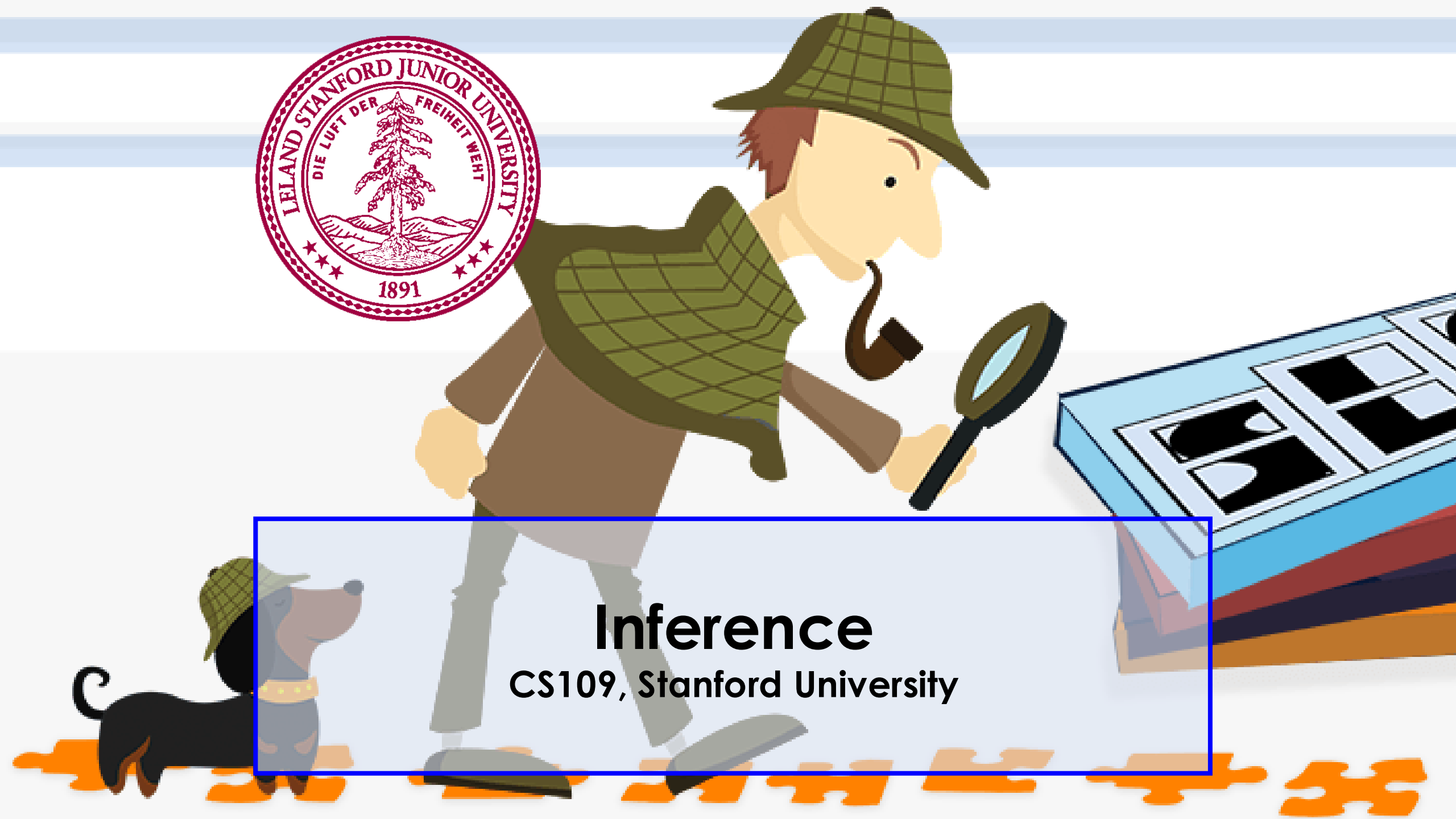


Inference

CS109, Stanford University

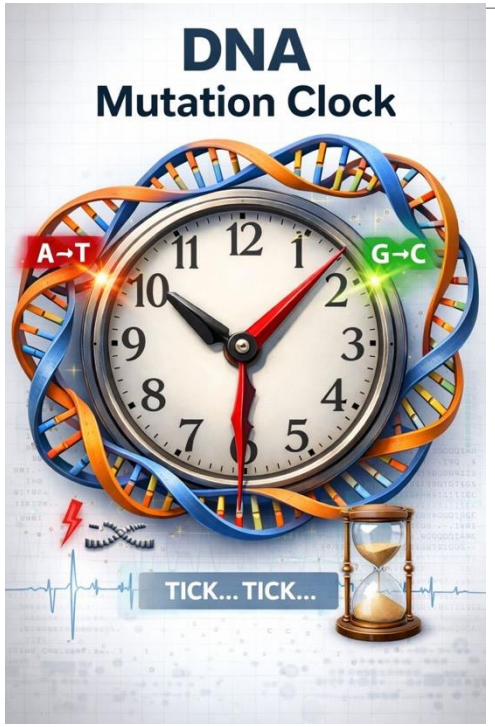
Announcements

(1) PEP is Monday-Wednesday !!! Have you signed up? This is required part of the course.
[Signup Link]

(2) We will release midterm study materials early next week. Midterm exam is Tuesday, Feb 10th 7-9pm. Midterm covers content through Weds (2/4) lecture.

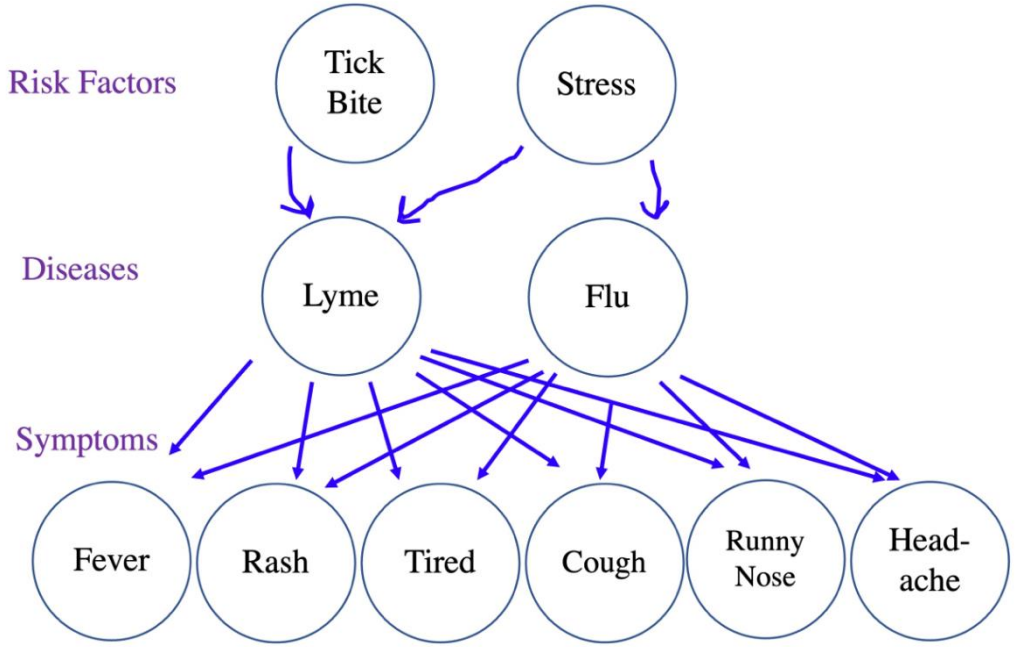
(3) Pset 4 has been released. Get started ASAP. This one is longer than others and is one of the best ways to study for the midterm.

PSET #4



Chess.com Puzzles

Chess.com is a website for playing chess. They are trying to estimate how well a player can solve chess puzzles (puzzle ability) as a random variable, A, which can take on integer values in the range 0 to 100 inclusive. Higher abilities mean the player is better at chess puzzles. Note that ability is **discrete**.



Today: Stanford Eye Test

Learning Goals

1. Combine Bayes Theorem and Random Variables

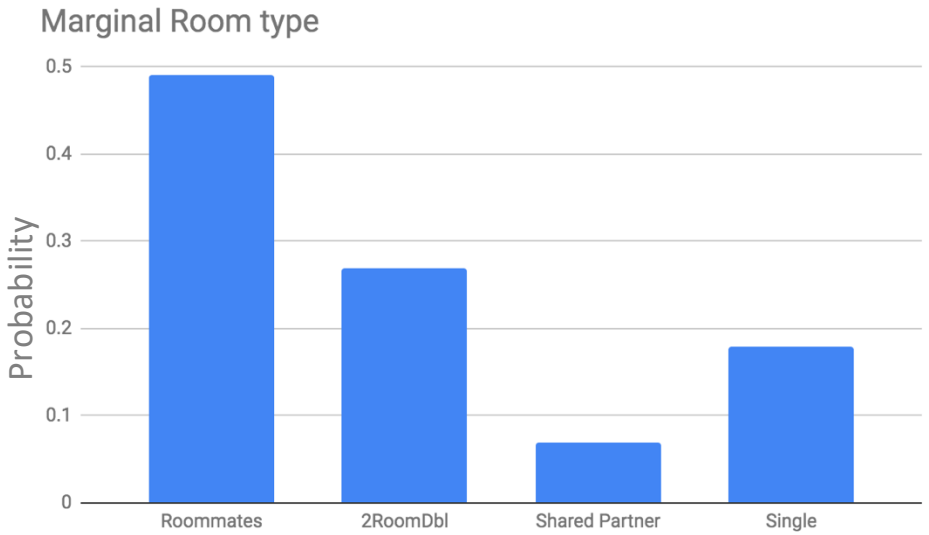
Review

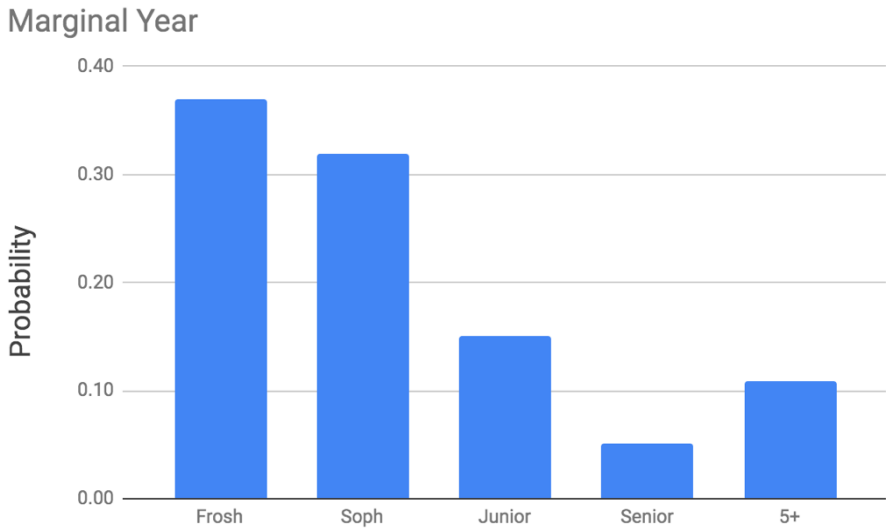
Probabilistic Models

Joint

	Roommates	2RoomDbl	Shared Partner	Single	
Frosh	0.30	0.07	0.00	0.00	0.37
Soph	0.12	0.18	0.00	0.03	0.32
Junior	0.04	0.01	0.00	0.10	0.15
Senior	0.01	0.02	0.02	0.01	0.05
5+	0.02	0.00	0.05	0.04	0.11
	0.49	0.27	0.07	0.18	1.00

Marginals





Today: Inference

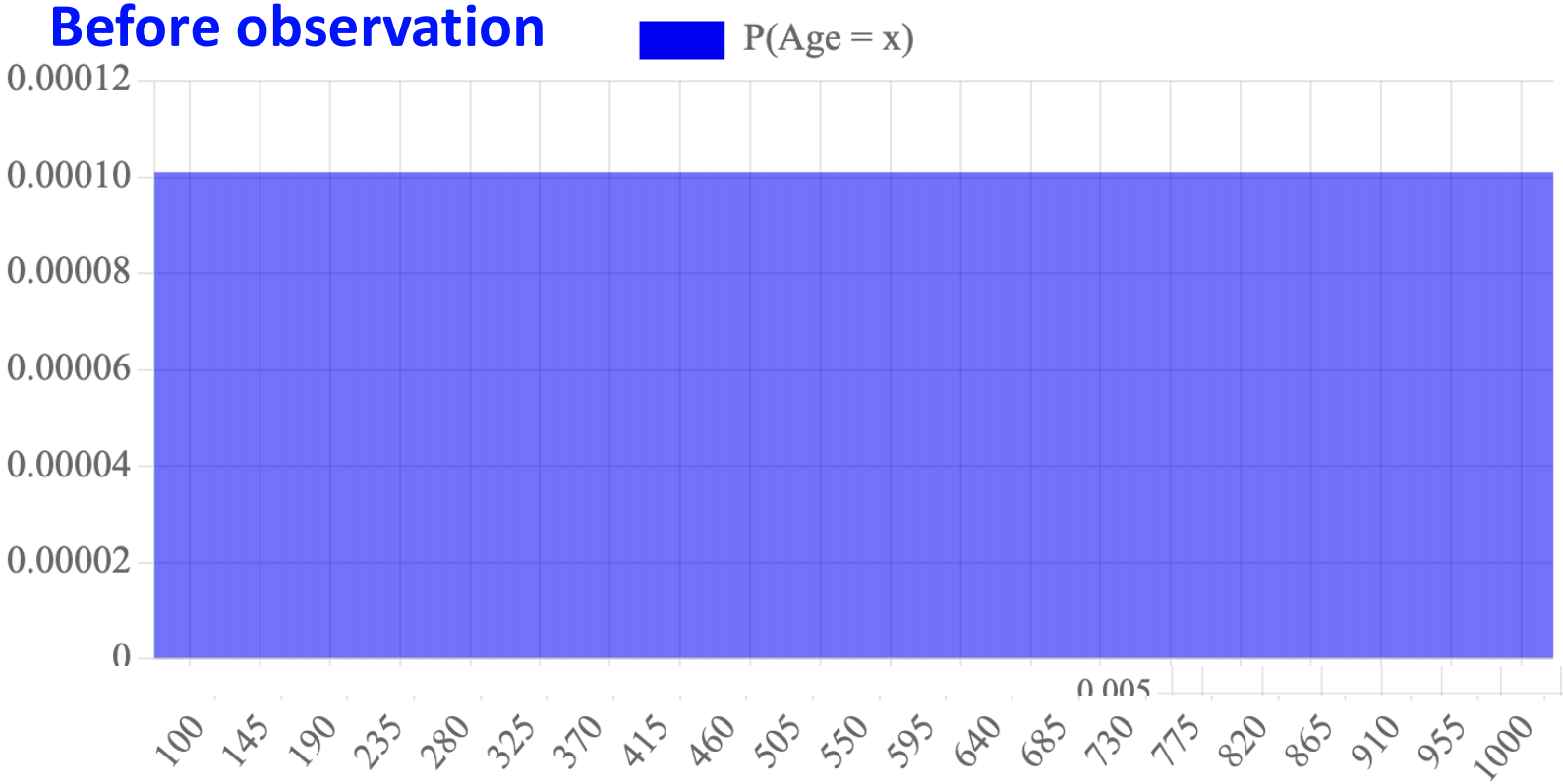
Inference *noun*

Updating one's belief about a random variable (or multiple) based on conditional knowledge regarding another random variable (or multiple) in a probabilistic model.

TLDR: conditional probability with random variables.

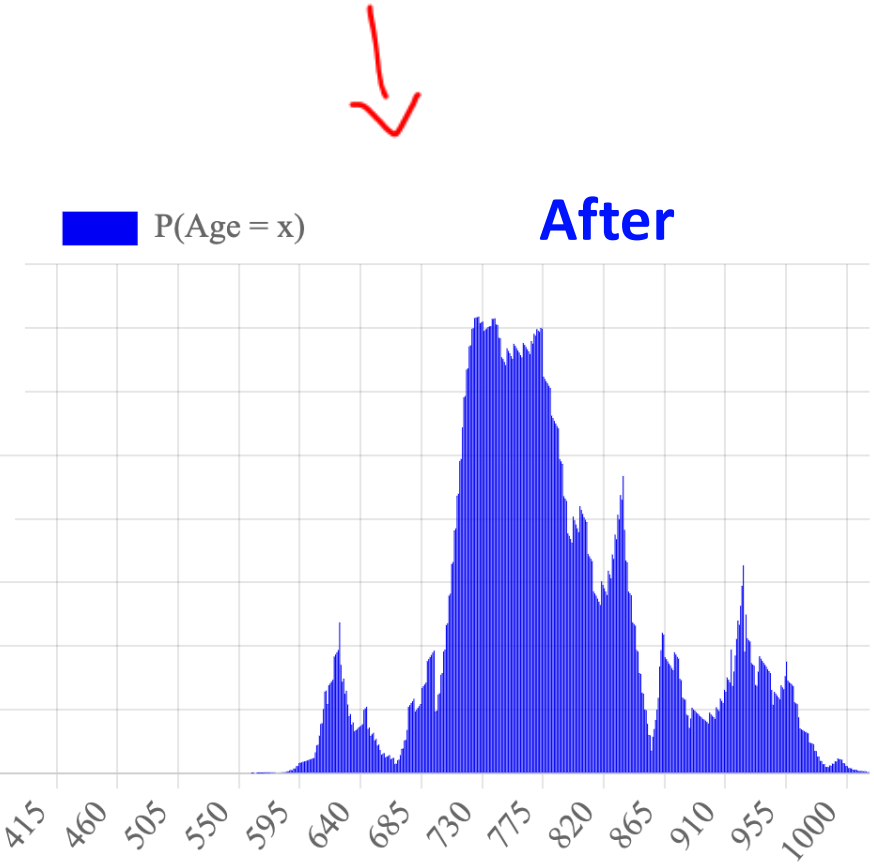
Update Belief PMF

Before observation



Observation

Remaining C14: 900



Bayesian Carbon Dating: Inference Overview

Let A be how many years old the sample is (A = 100 means the sample is 100 years old)
Let M be the observed amount of C14 left in the sample

$$P(A = i|M = 900) = \frac{P(M = 900|A = i)P(A = i)}{P(M = 900)}$$

$$= P(M = 900|A = i) \cdot P(A = i) \cdot K$$

Such that

$$K = \frac{1}{\sum_i P(M = 900|A = i)P(A = i)}$$

Probability of Having 900 Remain

$$P(M = 900|A = i)$$

There were originally 1000 C14 molecules.
Each molecule remains independently with equal probability p_i
What is the probability that 900 remain?

$$M \sim \text{Bin}(n = 1000, p = p_i)$$

$$P(M = 900|A = i) = \binom{1000}{900}(p_i)^{900} \cdot (1 - p_i)^{100}$$

Each molecules' time to live is exponential with λ = 1/8267

Let T be the time to decay for any one molecule

$$T \sim \text{Exp}(\lambda = 1/8267) \quad p_i = P(T > i) = 1 - P(T < i) = e^{-\frac{i}{8267}}$$

Inference as Code

$$P(A = i|M = 900) = P(M = 900|A = i) \cdot P(A = i) \cdot K$$

```
def update_belief(m = 900):
  """
  Returns a dictionary A, where A[i] contains the
  corresponding probability, P(A = i| M = 900).
  m is the number of C14 molecules remaining and i
  is age in years. i is in the range 100 to 10000
  """
  pr_A = {}
  n_years = 9901
  for i in range(100,10000+1):
    prior = 1 / n_years # P(A = i)
    likelihood = calc_likelihood(m, i) # P(M=m | A=i)
    pr_A[i] = prior * likelihood
  # implicitly computes the normalization constant
  normalize(pr_A)
  return pr_A
```

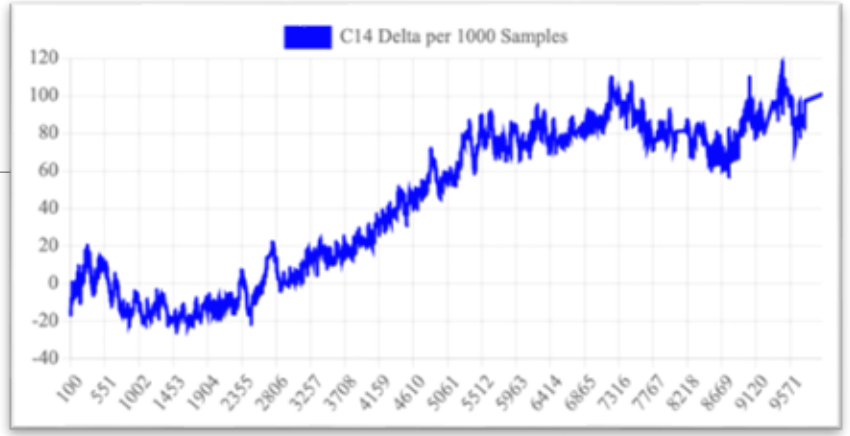
Probability of Having 900 Remain

$$P(M = 900 | A = i)$$

```
def calc_likelihood(m = 900, age):
  """
  Computes P(M = m | A = age), the probability of
  having m molecules left given the sample is age
  years old. Uses the exponential decay of C14
  """
  n_original = 1000
  p_remain = math.exp(-age/C14_MEAN_LIFE)
  return stats.binom.pmf(m, n_original, p_remain)
```

Probability of Having 900 Remain

$$P(M = 900 | A = i)$$




```
def calc_likelihood(m = 900, age):
  """
  Computes P(M = m | A = age), the probability of
  having m molecules left given the sample is age
  years old. Uses the exponential decay of C14
  """
  n_original = 1000 + delta_start(age)
  p_remain = math.exp(-age/C14_MEAN_LIFE)
  return stats.binom.pmf(m, n_original, p_remain)
```

Bayes with variables

Can mix discrete and continuous

Why is inference hard?

Normalization term is trippy at first

Likelihood term can be complex

Bayes + Continuous Random Variables

Let X be a **continuous** random variable

Let N be a **discrete** random variable

$$P(N = n|X = x) = \frac{P(X = x|N = n)P(N = n)}{P(X = x)}$$

$$P(N = n|X = x) = \frac{f(X = x|N = n) \cdot \epsilon \cdot P(N = n)}{f(X = x) \cdot \epsilon}$$

$$P(N = n|X = x) = \frac{f(X = x|N = n) \cdot P(N = n)}{f(X = x)}$$

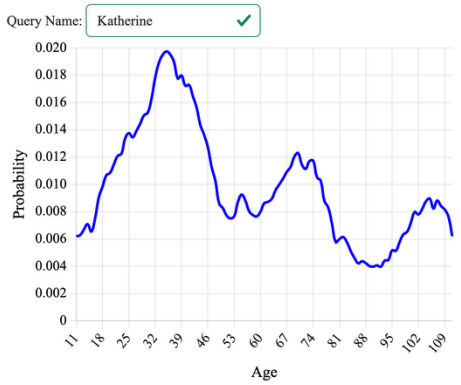
End Review

Today: Compare and Contrast Many Examples

Age from C14

Updated Delivery Prob

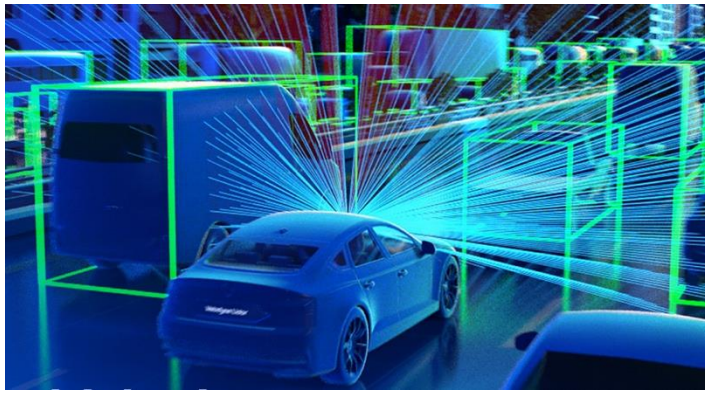
Age from Name



Hidden Chambers

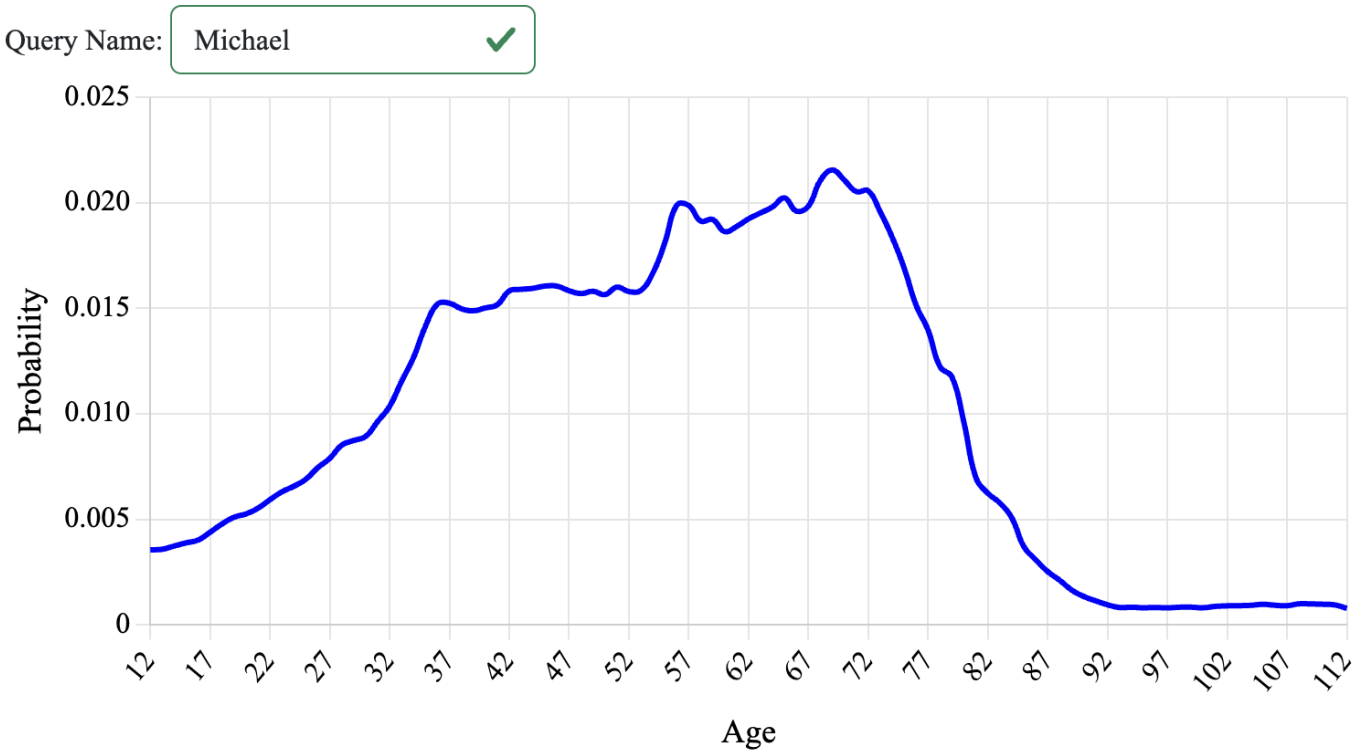
Stanford Eye Test

Updating Lidar Belief



Name 2 Age

Because of shifting patterns in name popularity, a person's name is a hint as to their age. The United States publishes a data which contains counts of how many US residents were born with a given name in a given year, based off Social Security applications. We can use inference to compute the reverse probability distribution: an updated belief in a person's age, given their name. As a reminder, if I know the year someone was born, I can calculate their age within one year.



Number of people with this name born this year

$$P(B = b, N = n) \approx \frac{\text{count}(b, n)}{k}$$

Birth year

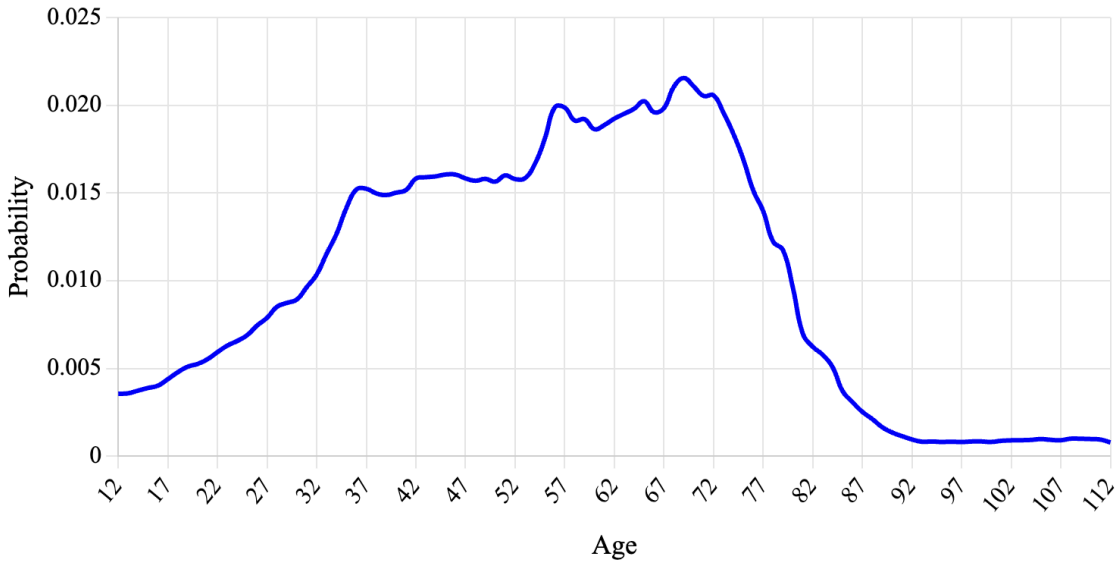
Name

Size of Dataset

Name 2 Age

Because of shifting patterns in name popularity, a person's name is a hint as to their age. The United States publishes a data which contains counts of how many US residents were born with a given name in a given year, based off Social Security applications. We can use inference to compute the reverse probability distribution: an updated belief in a person's age, given their name. As a reminder, if I know the year someone was born, I can calculate their age within one year.

Query Name: Michael ✓



$$P(B = b, N = n) \approx \frac{\text{count}(b, n)}{k}$$

Birth year

Name

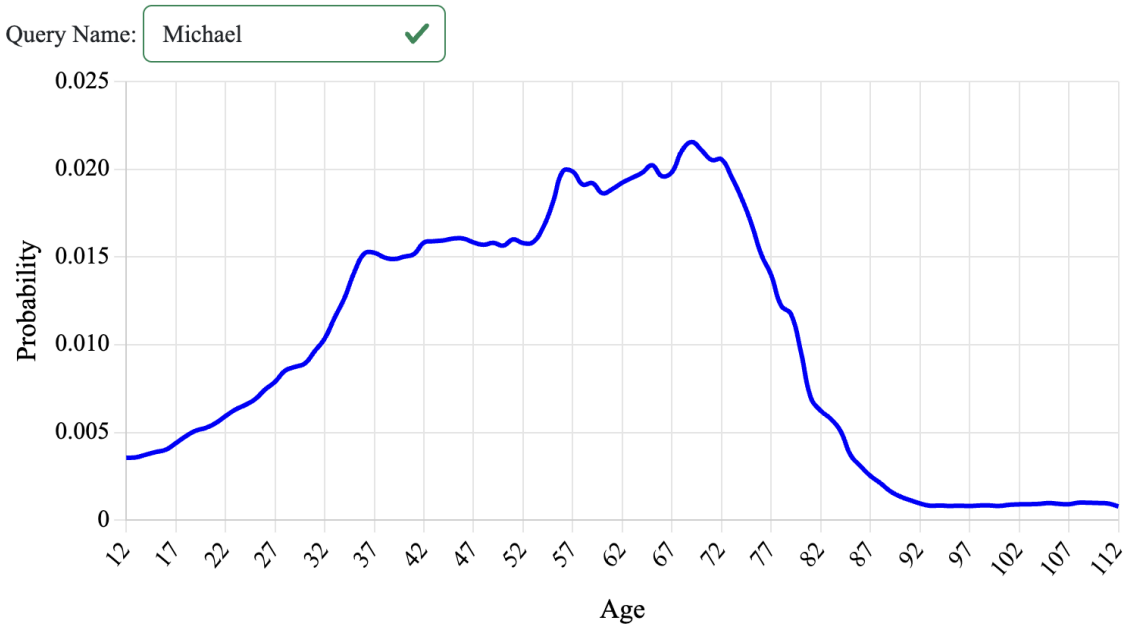
Size of Dataset

$$P(B = 1964 | N = \text{Michael})$$

How can you use the joint to compute this?

Name 2 Age

Because of shifting patterns in name popularity, a person's name is a hint as to their age. The United States publishes a data which contains counts of how many US residents were born with a given name in a given year, based off Social Security applications. We can use inference to compute the reverse probability distribution: an updated belief in a person's age, given their name. As a reminder, if I know the year someone was born, I can calculate their age within one year.



$$P(B = b, N = n) \approx \frac{\text{count}(b, n)}{k}$$

Birth year

Name

Size of Dataset

$$P(B = 1964 | N = \text{Michael})$$

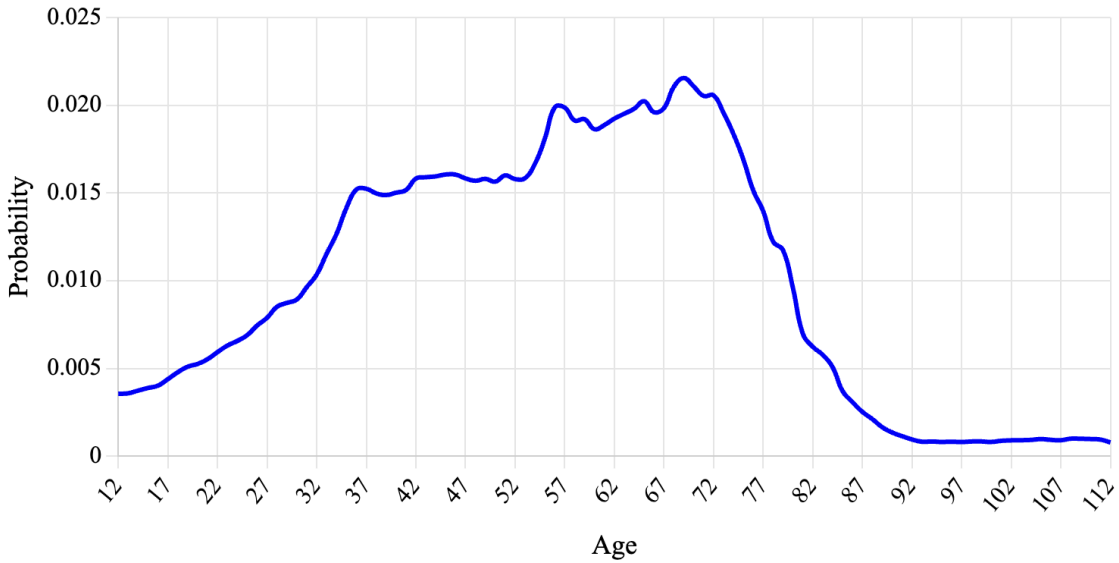
How can you use the joint to compute this?

$$\frac{P(B = 1964, N = \text{Michael})}{\sum_b P(B = b, N = \text{Michael})}$$

Name 2 Age

Because of shifting patterns in name popularity, a person's name is a hint as to their age. The United States publishes a data which contains counts of how many US residents were born with a given name in a given year, based off Social Security applications. We can use inference to compute the reverse probability distribution: an updated belief in a person's age, given their name. As a reminder, if I know the year someone was born, I can calculate their age within one year.

Query Name: Michael ✔



$$P(B = b, N = n) \approx \frac{\text{count}(b, n)}{k}$$

Birth year

Name

Size of Dataset

$$P(B = b | N = \text{Michael})$$

Describe how you would compute the entire PMF using code.

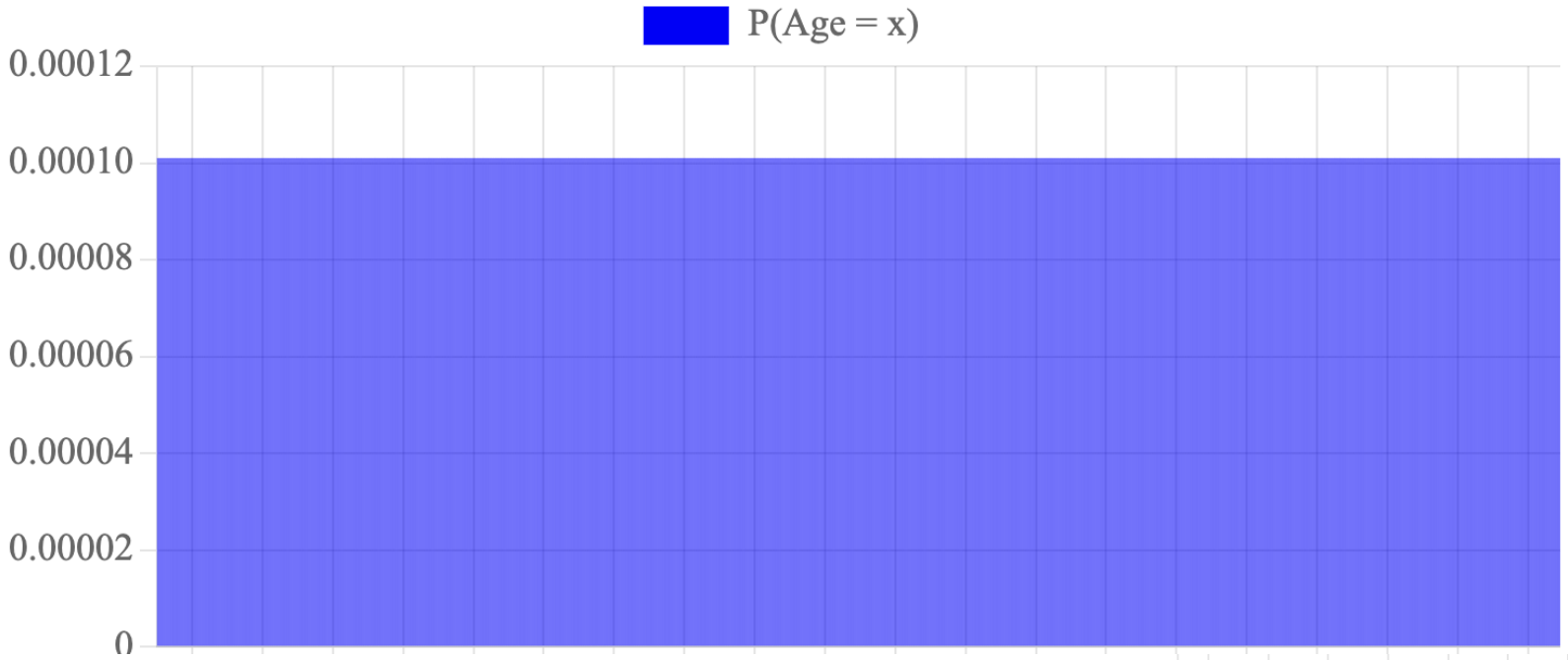
Name 2 Age

$$P(B = b | N = \text{Michael})$$

Describe how you would compute the entire PMF using code.

```
def update_belief_name_to_age(name = 'Michael'):
    # pr_age[i] is P(Age = i| name).
    # prob_name_and_age is just a counting from the US
    # Social Security database.
    pr_age = {}
    for i in range(10,110):
        pr_age[i] = calc_prob_name_and_age(name, i)
    # implicitly computes the normalization constant
    normalize(pr_age)
    return pr_age
```

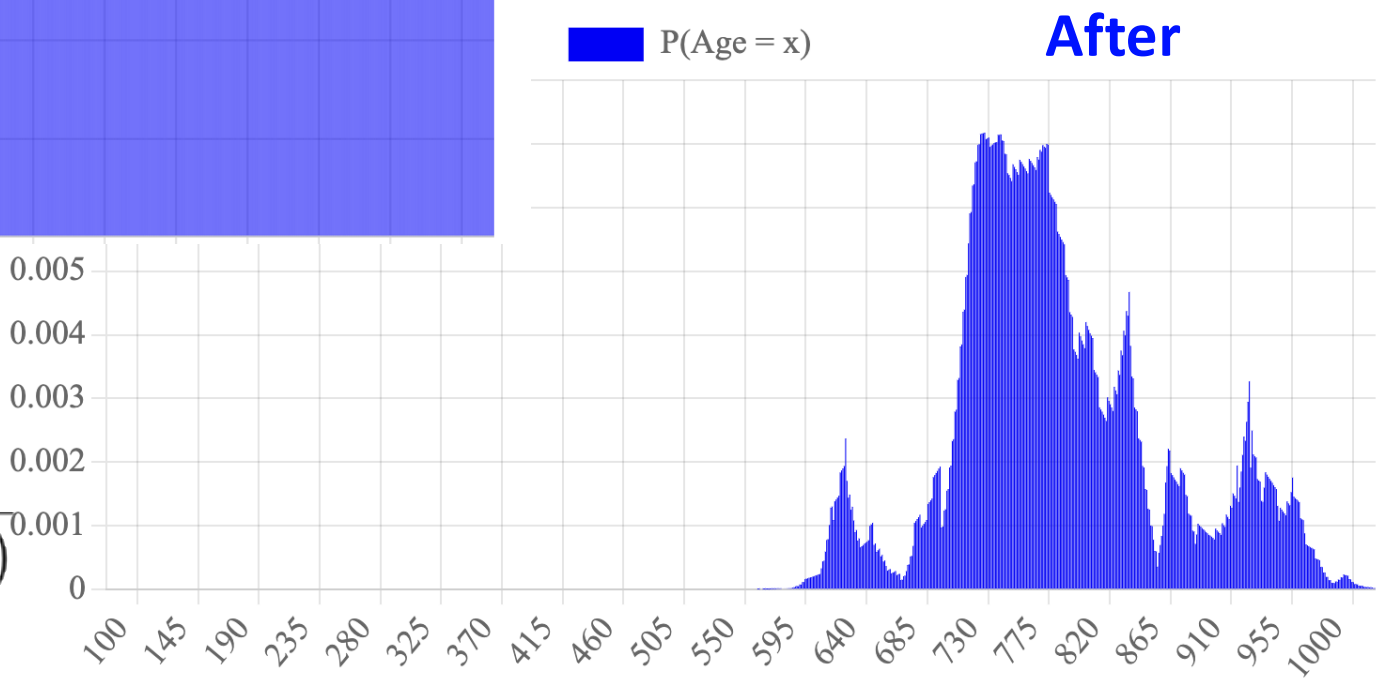
Compare and Contrast Code: (1) Bayesian Carbon Dating



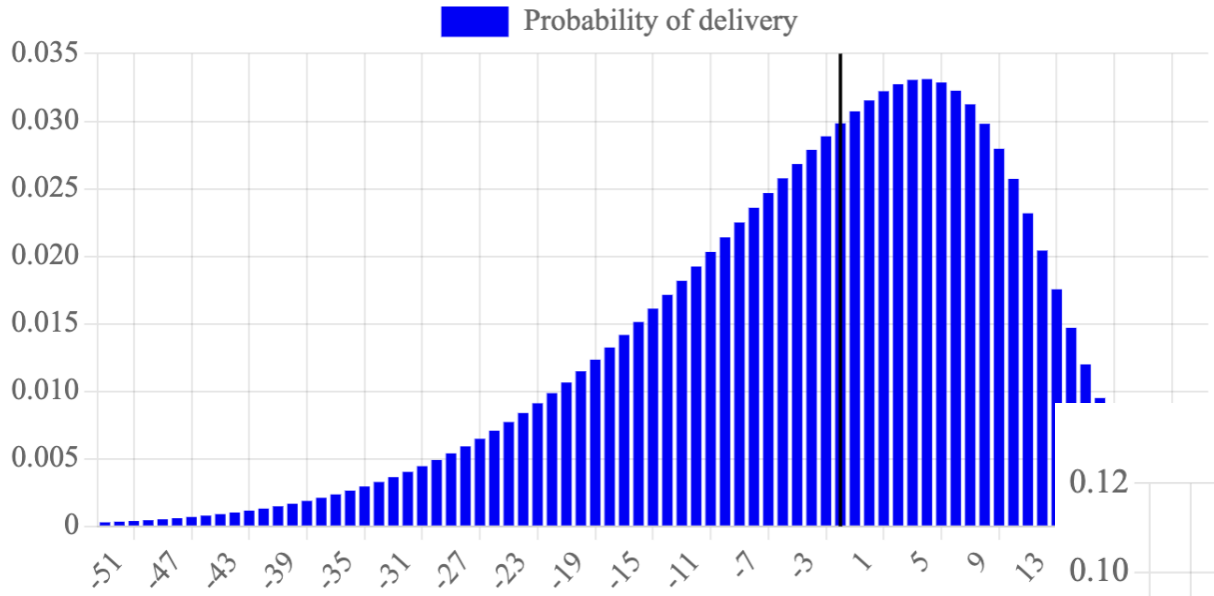
Observation

Remaining C14: 900

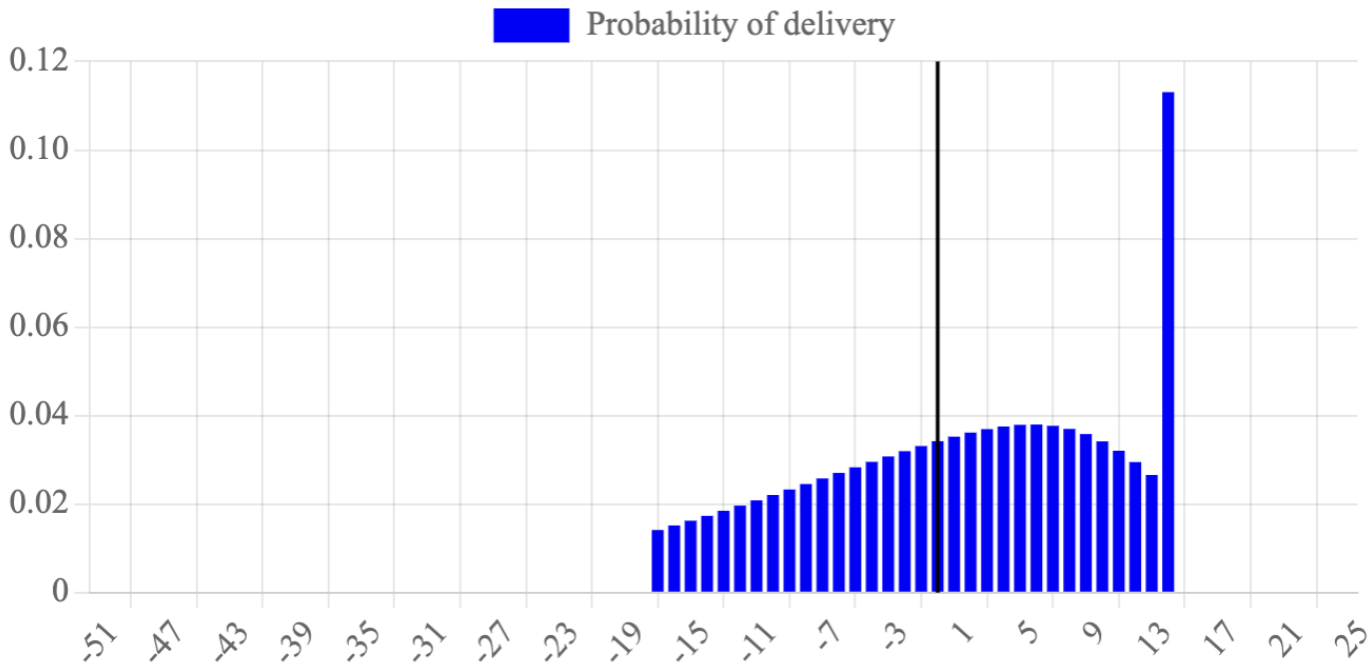
$$P(A = a|M = 900) = \frac{P(M = 900|A = a)P(A = a)}{\sum_i P(M = 900|A = i)P(A = i)}$$



Compare and Contrast Code: (2) Baby Delivery



Its 19 days until the due date and no baby



For each value *d:*

$$P(D = d|\text{no child so far})$$

$$= \frac{P(\text{no child so far}|D = d)P(D = d)}{P(\text{no child so far})}$$

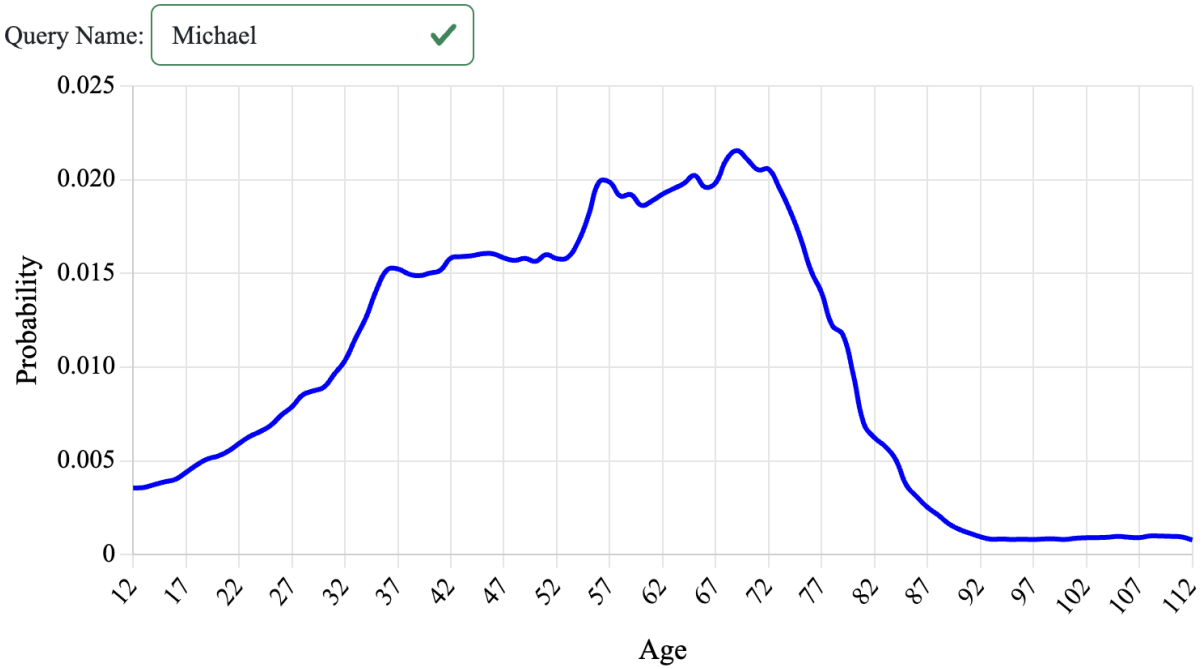
Compare and Contrast Code: (3) Name To Age

Because of shifting patterns in name popularity, a person's name is a hint as to their age. The United States publishes a data which contains counts of how many US residents were born with a given name in a given year, based off Social Security applications. We can use inference to compute the reverse probability distribution: an updated belief in a person's age, given their name. As a reminder, if I know the year someone was born, I can calculate their age within one year.

$$P(B = 1964 \mid N = \text{Michael}) = \frac{P(N = \text{Michael}, B = 1964)}{P(N = \text{Michael})}$$

$$\approx \frac{\left(\frac{\text{count}(1964, \text{Michael})}{k}\right)}{\left(\sum_{y \in \text{years}} \frac{\text{count}(y, \text{Michael})}{k}\right)}$$

$$\approx \frac{\text{count}(1964, \text{Michael})}{\sum_{y \in \text{years}} \text{count}(y, \text{Michael})}$$



```
def update_belief_carbon_dating(m = 900):
  # pr_A[i] is P(Age = i| m = 900).
  pr_A = {}
  for i in range(100,10000+1):
    prior = 1 / n_years # P(A = i)
    likelihood = calc_likelihood(m, i) #P(M=m|A=i)
    pr_A[i] = likelihood * prior
  # implicitly computes the normalization constant
  normalize(pr_A)
  return pr_A
```

```
def update_belief_baby(prior, today = 10):
  # pr_D[i] is P(D = i| No Baby Yet).
  pr_D = {}
  for i in range(-50,25):
    # P(NoBaby | D = i)
    likelihood = 0 if i < today else 1
    pr_D[i] = likelihood * prior[i]
  # implicitly computes the LOTP
  normalize(pr_D)
  return pr_D
```

```
def update_belief_name_to_age(name = 'Michael'):
  # pr_age[i] is P(Age = i| name).
  # prob_name_and_age is just a counting from the US
  # Social Security database.
  pr_age = {}
  for i in range(10,110):
     pr_age[i] = calc_prob_name_and_age(name, i)
  # implicitly computes the normalization constant
  normalize(pr_age)
  return pr_age
```

What do you notice is the same. What is different?

Normalize in Python

```
# list normalization
def normalize_list(data_list):
    total_sum = np.sum(data_list)
    return np.array(data_list) / total_sum

>>> norm = normalize_list([10, 20, 30, 40])
>>> np.sum(norm) # 1.0, always (within floating point error)
```

return
[0.1 0.2 0.3 0.4]

```
# dictionary normalization
def normalize_dict(data_dict):
    total_sum = sum(data_dict.values())
    normalized = {}
    for key, value in data_dict.items():
        normalized[key] = value / total_sum
    return normalized

>>> norm = normalize_dict({'a': 100, 'b': 200, 'c': 300})
>>> np.sum(norm.values()) # 1.0, always (within floating point error)
```

return
{'a': 0.166, 'b': 0.333, 'c': 0.5}

bookapp — Python — 83×24

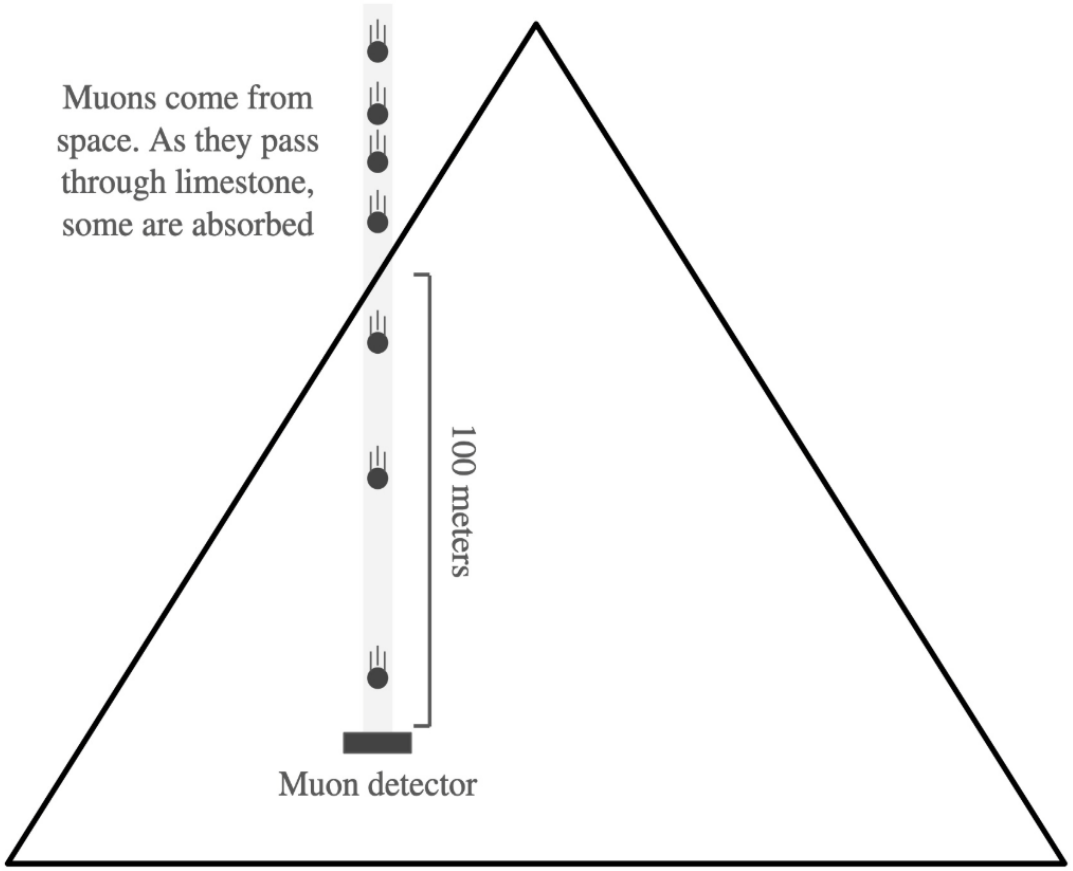
...R/coterm/src — -zsh ... | ...R/coterm/src — -zsh ... | ...esearch/LLM — -zsh ... | ...app/bookapp — -zsh | ...p/bookapp — Python | ...pp/bookapp — -zsh ...

```
>>> def normalize_dict(data_dict):
...     total_sum = sum(data_dict.values())
...     normalized = {}
...     for key, value in data_dict.items():
...         normalized[key] = value / total_sum
...     return normalized
...
>>> before = {"a":5, "b":0.1}
>>> after = normalize_dict(before)
>>> after
{'a': 0.9803921568627452, 'b': 0.019607843137254905}
>>> before
{'a': 5, 'b': 0.1}
>>> before = {"cats":2, "dogs":100, "snakes":1}
>>> normalize_dict(before)
{'cats': 0.019417475728155338, 'dogs': 0.970873786407767, 'snakes': 0.009708737864077669}
>>>
>>>
>>>
>>>
>>>
>>>
>>>
```

Are you ready
For Hidden Chambers???

Hidden Pyramid Chambers with Poisson + Bayes

Basics of Muography

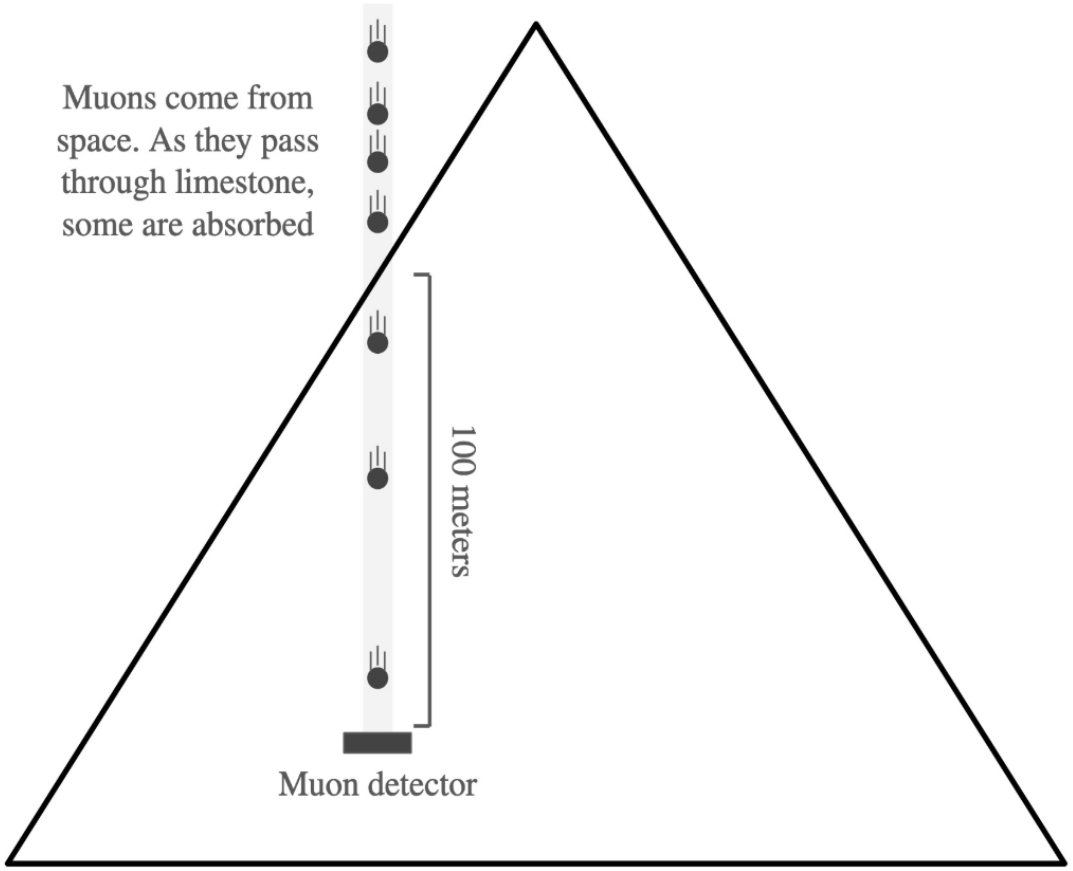


Beer Lambert Law

$$\lambda_x = 100 \cdot e^{-x/40}$$

Rate of muons depends on x, amount of limestone

a. (6 points) Imagine the entire 100 meter path is limestone. In that case, the rate of muons arriving per month on the detection plate is $100 \cdot e^{-100/40} = 8.2$. What is the probability that in one month you would observe 12 muons?

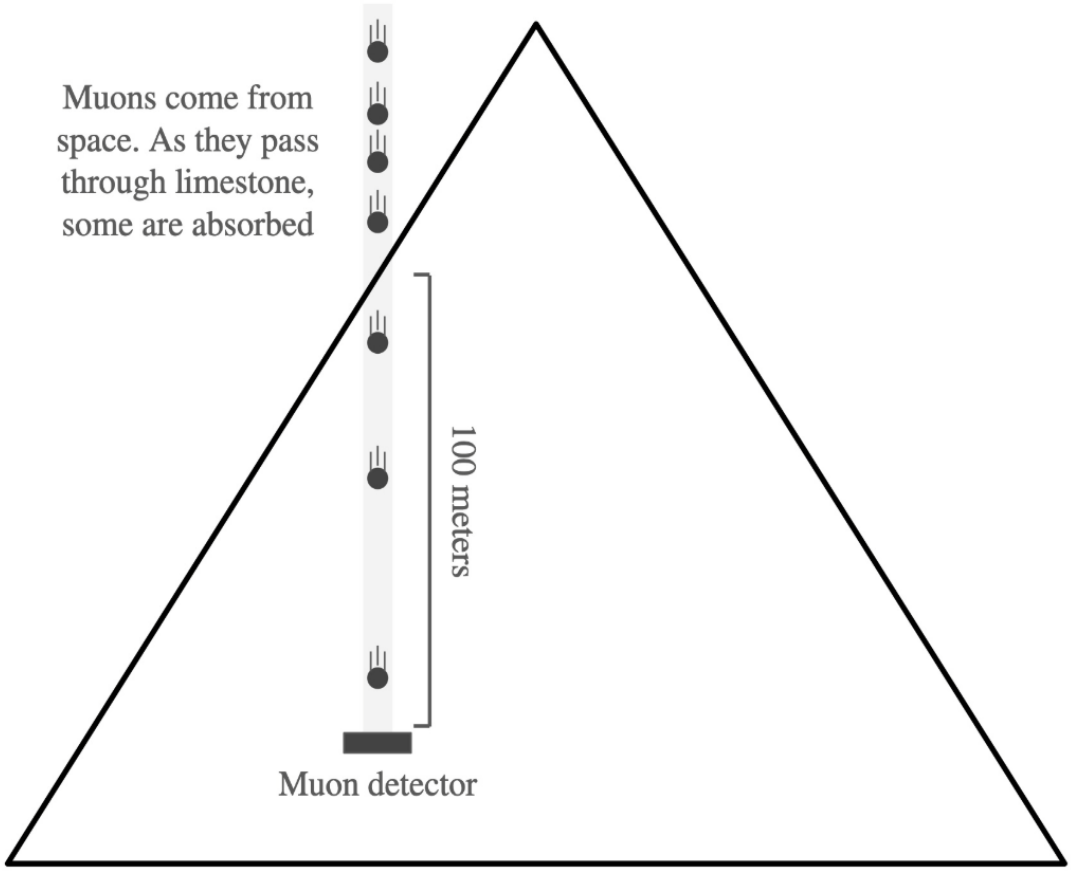


Beer Lambert Law

$$\lambda_x = 100 \cdot e^{-x/40}$$

Rate of muons depends on x, amount of limestone

b. (14 points) Let X be your belief in the meters of limestone above the detection plate. Your prior belief is that any number of meters from 0 to 100 is equally likely: $X \sim \text{Uni}(0, 100)$. After one month, your detection plate has been hit by 12 muons. What is your updated belief in X?
Recall: You may leave your answer with integrals or sums. You don't need to simplify for full credit.

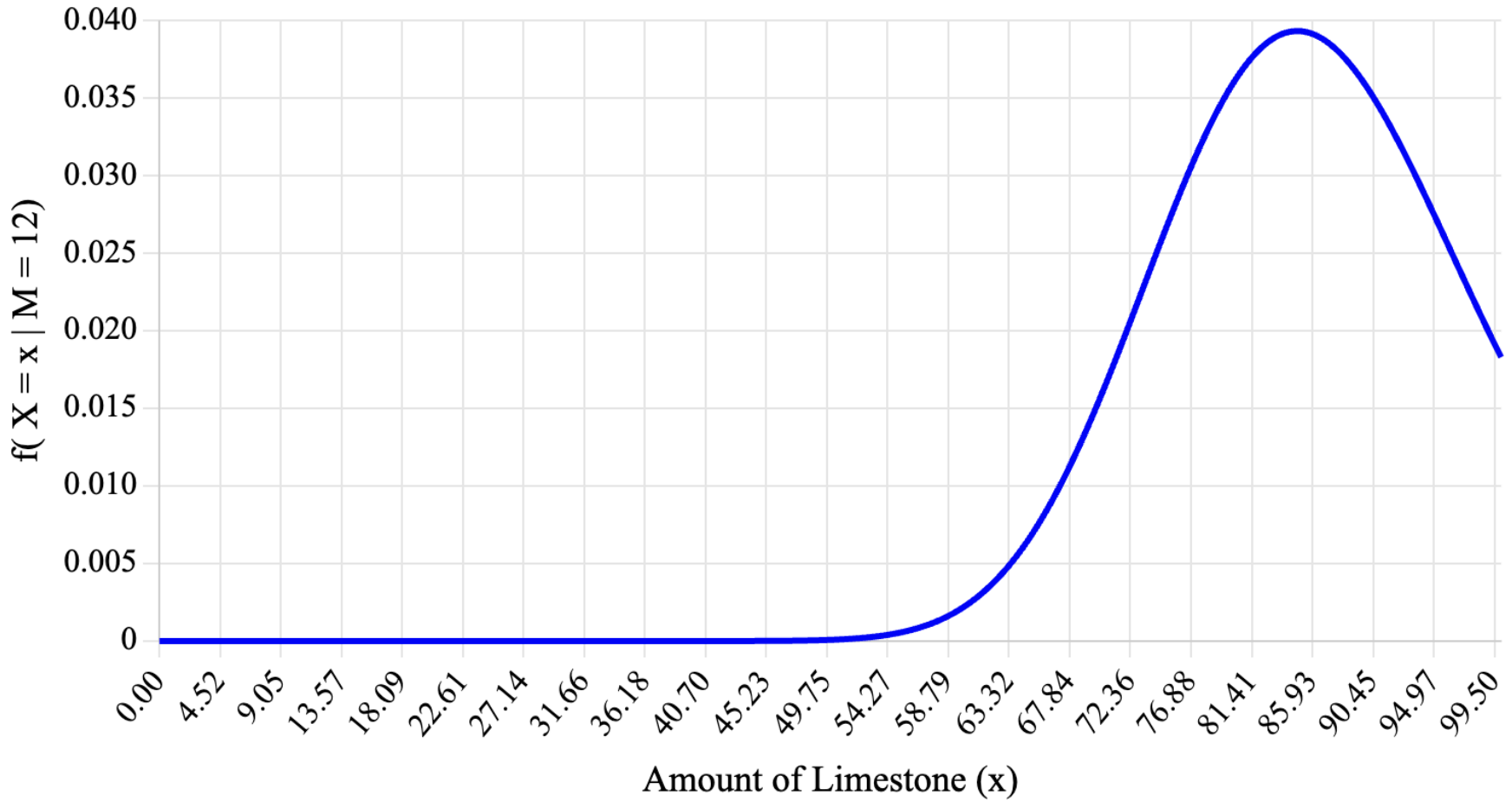


$$\lambda_x = 100 \cdot e^{-x/40}$$

Rate of muons depends on x, amount of limestone

Here is what that PDF equation looks like:

Number of muons (m): 12



Are you ready
For Stanford Acuity Test???

A Better Eye Test

https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(21)02149-8/fulltext

https://www.science.org/content/article/eye-robot-artificial-intelligence-dramatically-improves-accuracy-classic-eye-exam

https://ojs.aaai.org/index.php/AAAI/article/view/5384/5240

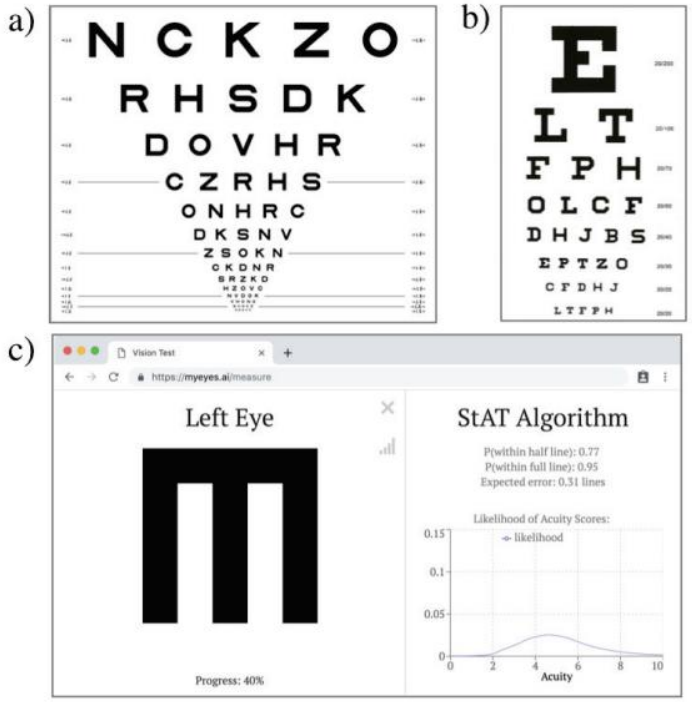
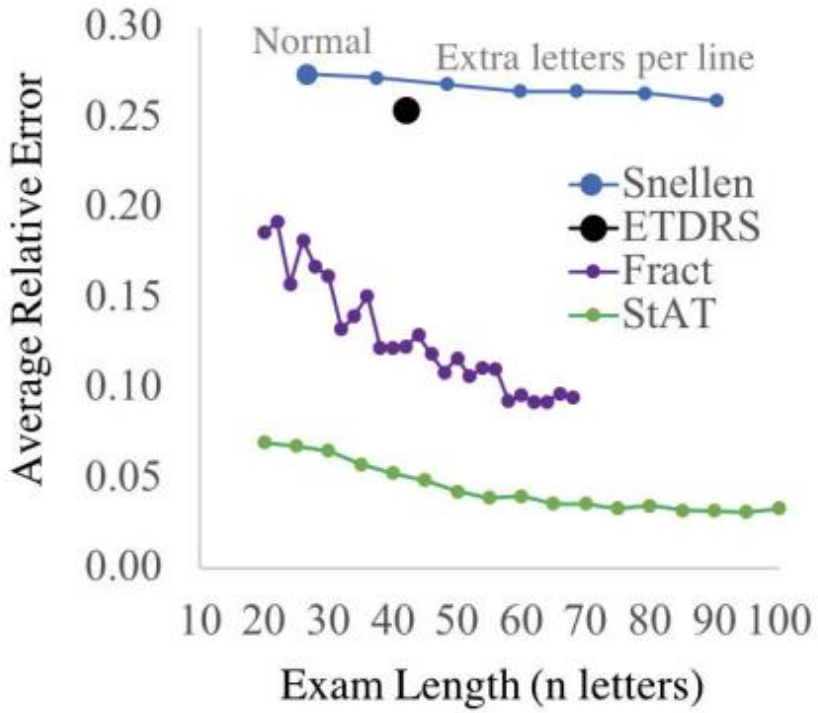
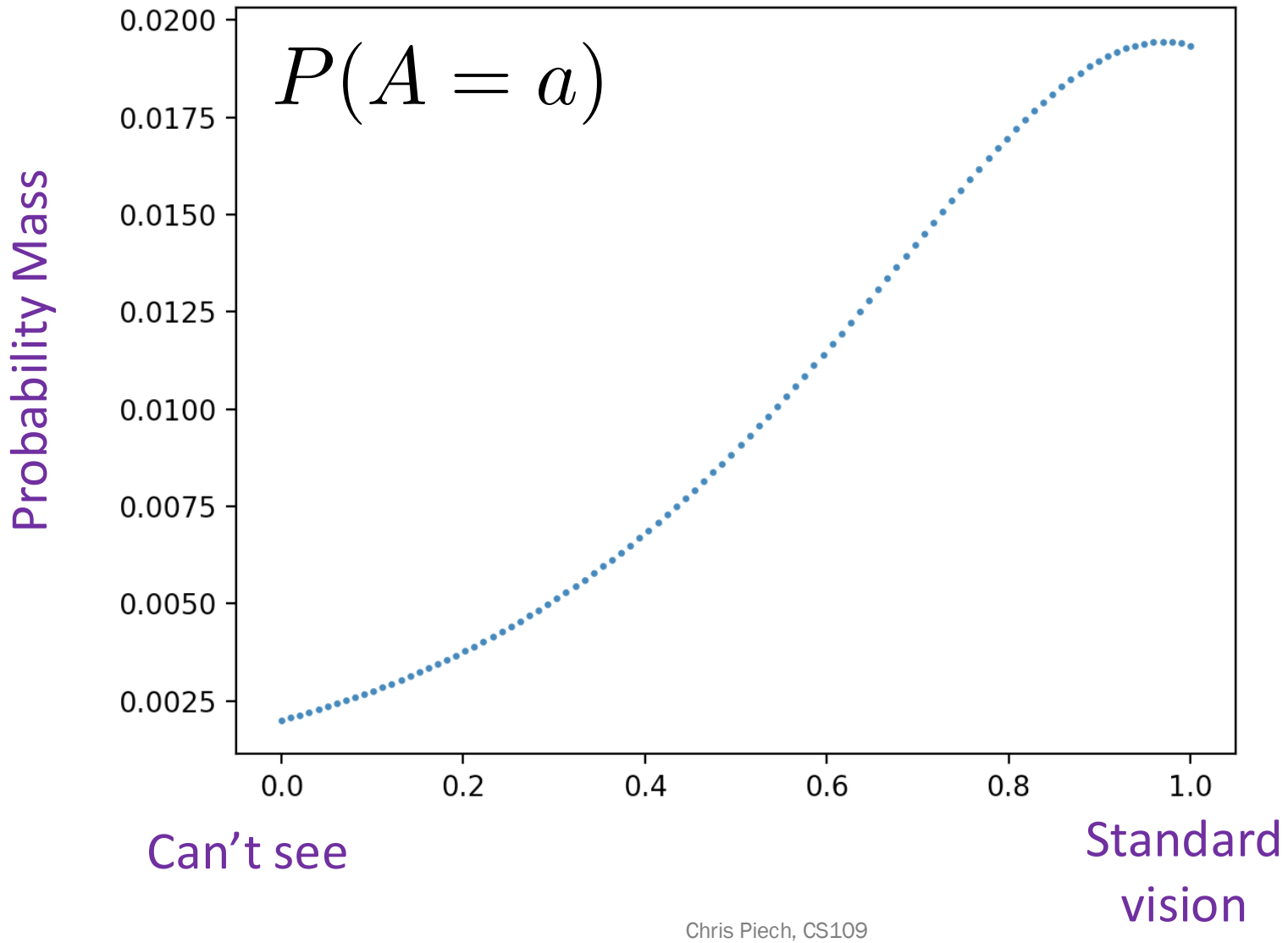


Figure 1: a) ETDRS, b) Snellen and c) StAT eye exams.



Prior Belief in Ability to See (Random Var A)



PMF is Actually Stored as a Dictionary

```
def main():
  belief = get_prior_belief()
```

a	P(A=a)
0.00	0.00198
0.01	0.00205
0.02	0.00211
0.03	0.00218
0.04	0.00225
0.05	0.00233
0.06	0.0024
0.07	0.00248
0.08	0.00256
0.09	0.00264
0.10	0.00273
0.11	0.00281
0.12	0.0029
0.13	0.00299
0.14	0.00309
0.15	0.00319
0.16	0.00329
0.17	0.00339
0.18	0.0035

a	P(A=a)
0.20	0.00372
0.21	0.00384
0.22	0.00396
0.23	0.00408
0.24	0.00421
0.25	0.00434
0.26	0.00447
0.27	0.00461
0.28	0.00475
0.29	0.00489
0.30	0.00504
0.31	0.00519
0.32	0.00535
0.33	0.00551
0.34	0.00567
0.35	0.00584
0.36	0.00601
0.37	0.00619
0.38	0.00637

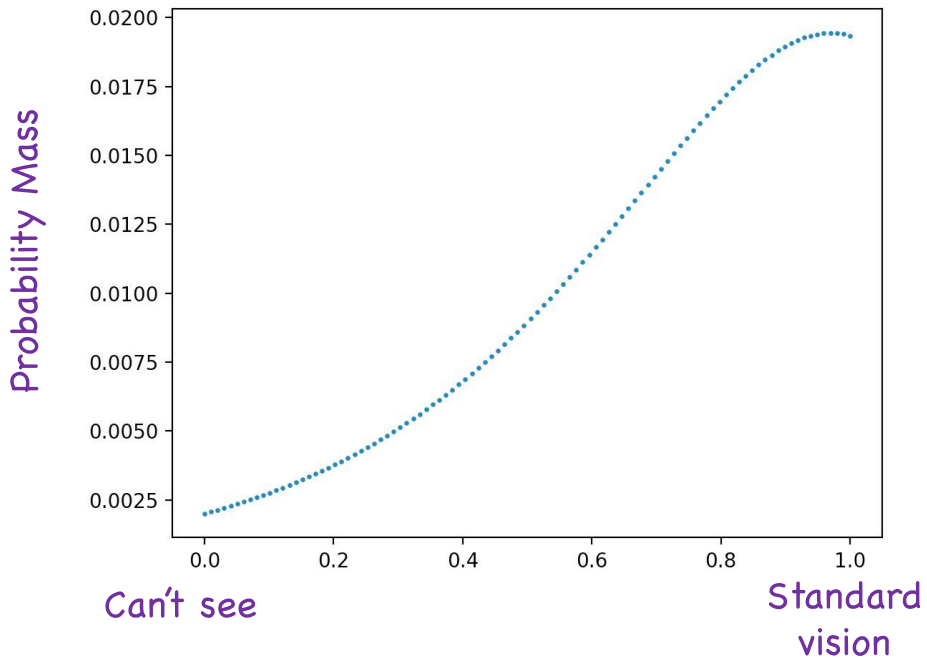
. . .

a	P(A=a)
0.80	0.01684
0.81	0.01708
0.82	0.01731
0.83	0.01753
0.84	0.01774
0.85	0.01795
0.86	0.01814
0.87	0.01832
0.88	0.01848
0.89	0.01864
0.90	0.01877
0.91	0.0189
0.92	0.019
0.93	0.01909
0.94	0.01916
0.95	0.01921
0.96	0.01924
0.97	0.01925
0.98	0.01924

Prior Belief in Ability to See (Random Var A)

```
belief = get_prior_belief()
```

As a graph



As a dictionary

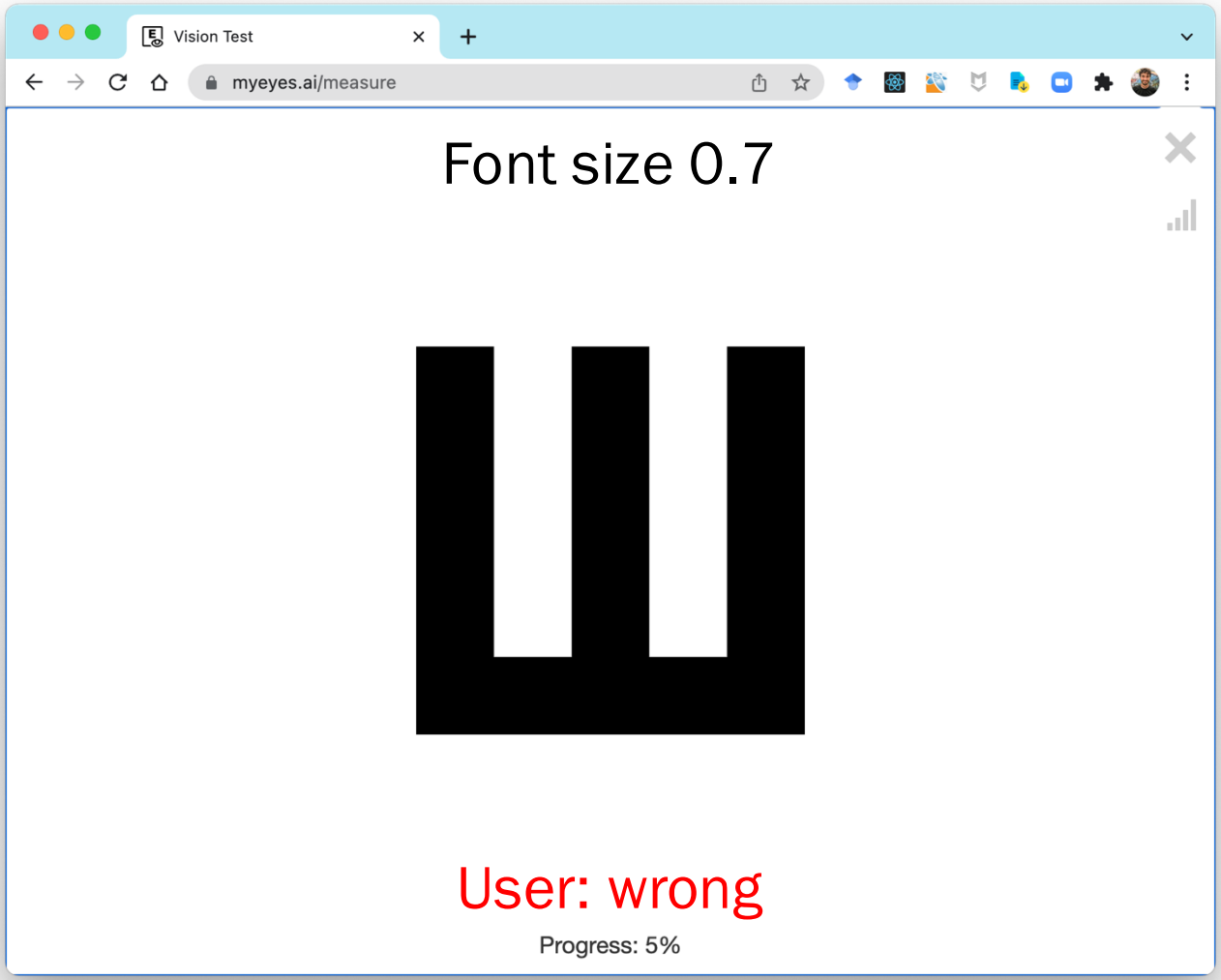
a	P(A=a)
0.00	0.00198
0.01	0.00205
0.02	0.00211
0.03	0.00218
0.04	0.00225
0.05	0.00233
0.06	0.0024
0.07	0.00248
0.08	0.00256
0.09	0.00264
0.10	0.00273
0.11	0.00281
0.12	0.0029
0.13	0.00299
0.14	0.00309
0.15	0.00319
0.16	0.00329
0.17	0.00339
0.18	0.0035
0.19	0.00361

a	P(A=a)
0.20	0.00372
0.21	0.00384
0.22	0.00396
0.23	0.00408
0.24	0.00421
0.25	0.00434
0.26	0.00447
0.27	0.00461
0.28	0.00475
0.29	0.00489
0.30	0.00504
0.31	0.00519
0.32	0.00535
0.33	0.00551
0.34	0.00567
0.35	0.00584
0.36	0.00601
0.37	0.00619
0.38	0.00637
0.39	0.00655

...

a	P(A=a)
0.80	0.01684
0.81	0.01708
0.82	0.01731
0.83	0.01753
0.84	0.01774
0.85	0.01795
0.86	0.01814
0.87	0.01832
0.88	0.01848
0.89	0.01864
0.90	0.01877
0.91	0.0189
0.92	0.019
0.93	0.01909
0.94	0.01916
0.95	0.01921
0.96	0.01924
0.97	0.01925
0.98	0.01924
0.99	0.01921

The Patient is Shown One Letter and They Get it Wrong



Observation $Y = 0$

Number or Dictionary?

belief

$$P(A = a|Y = 0) = \frac{P(Y = 0|A = a)P(A = a)}{P(Y = 0)}$$

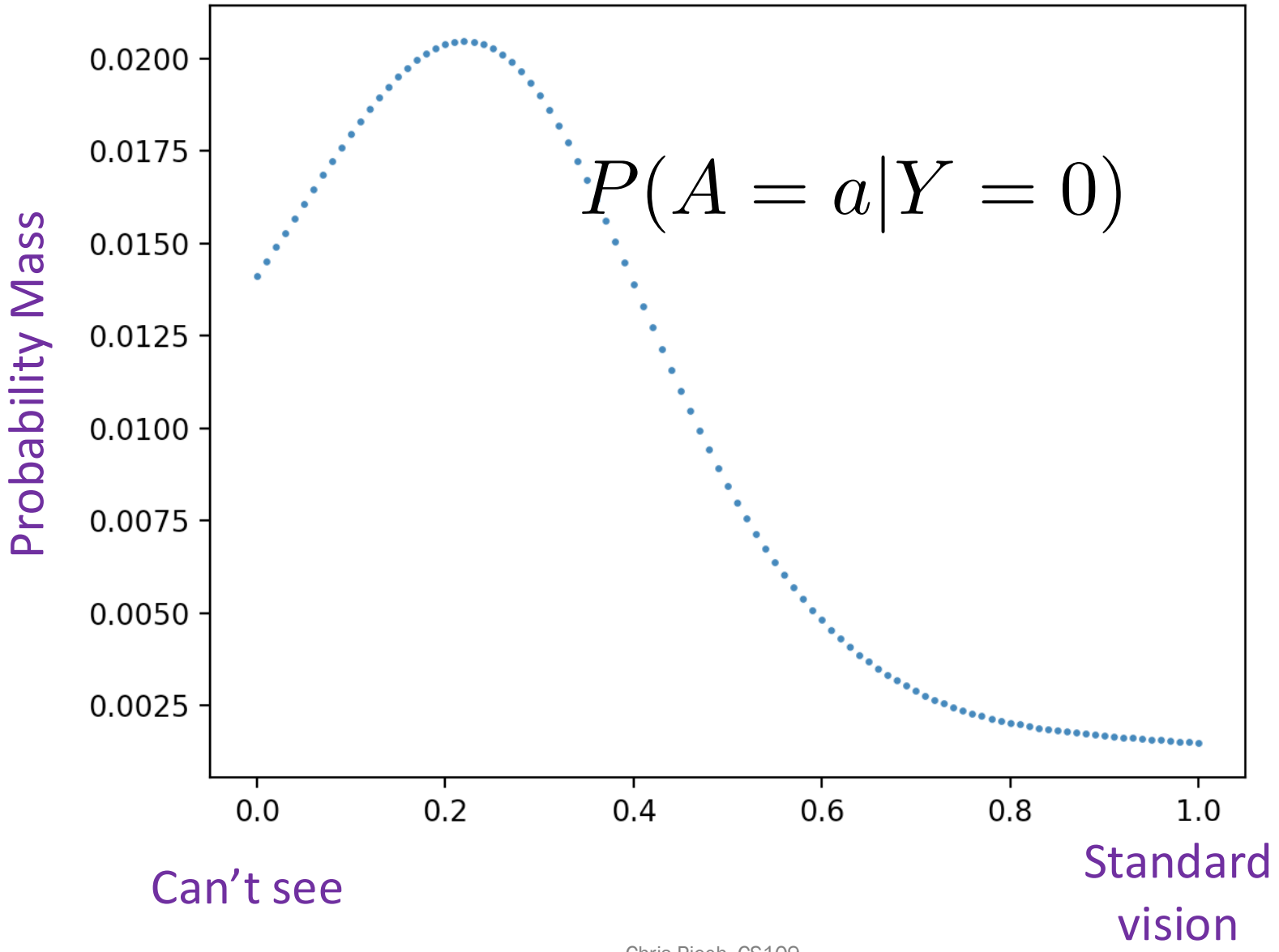
belief[a] = 0.001

Today: I am going to simplify the units of vision

Normally doctors measure ability to see in logarithmic units. To make today's demo easier to understand

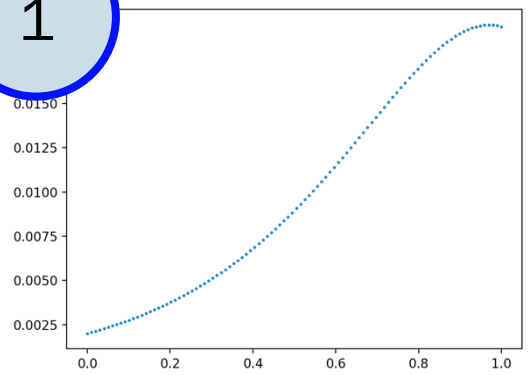
I have translated both onto a [0, 1] scale.
Where 0 means can't see and 1.0 is standard vision

Posterior Belief in Ability to See (Random Var A)



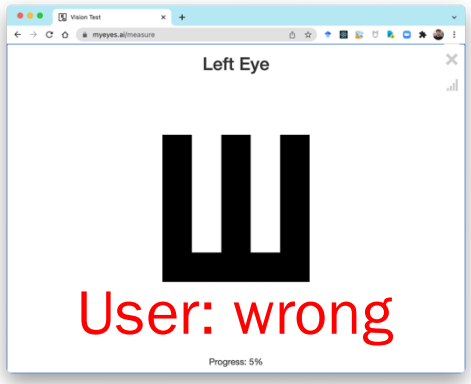
Bayes with Random Variables

1



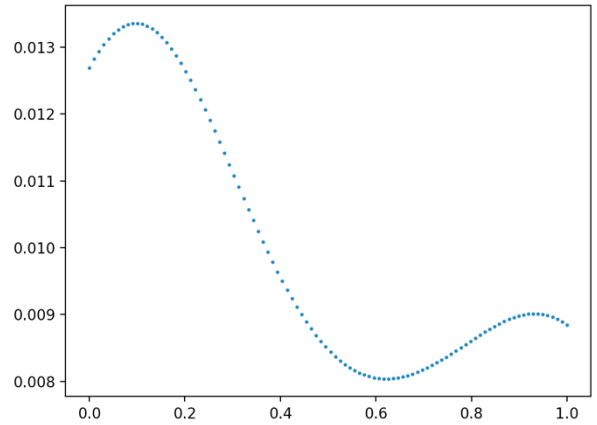
$$P(A = a)$$

2



Observation $Y = 0$
(At font size s_1)

3

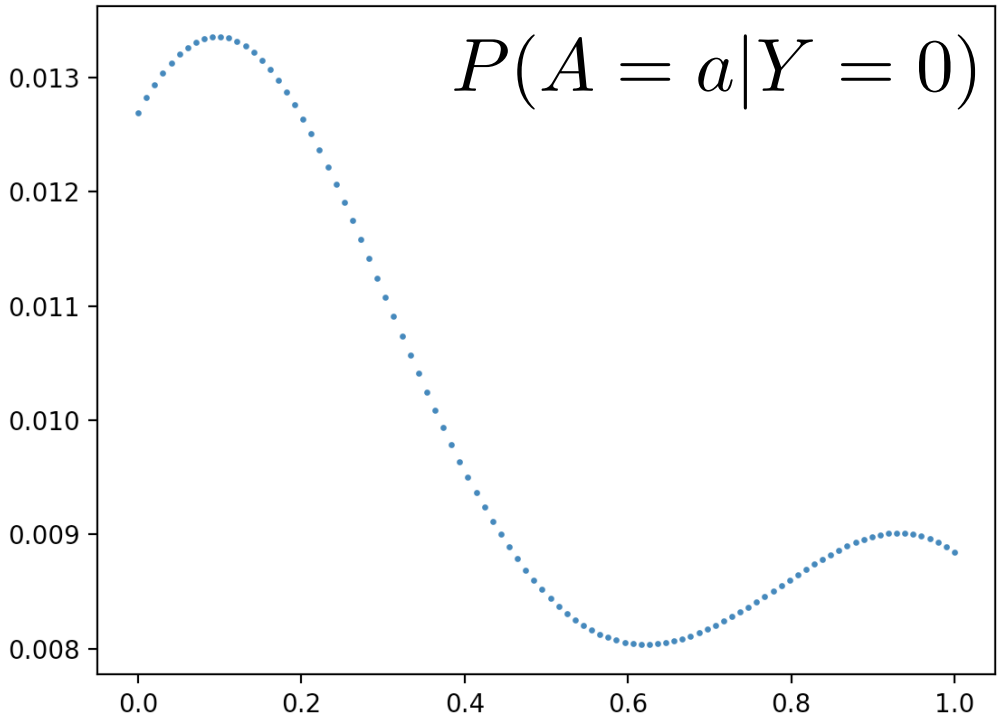


$$P(A = a|Y = 0)$$

$$P(A = a|Y = 0) = \frac{P(Y = 0|A = a)P(A = a)}{P(Y = 0)}$$

Inference on a non-bernoulli random variable

In plain English: run bayes for each value of a




```
# RV bayes as code
def update(belief, obs):
    for a in support:
      prior_a = belief[a]
      likelihood = calc_likelihood(a, obs)
      belief[a] = prior_a * likelihood
    normalize(belief)
```

likelihood

$$P(A = a|Y = 0) = \frac{P(Y = 0|A = a)P(A = a)}{P(Y = 0)}$$

Normalize???

```
# RV bayes as code
def update(belief, obs):
    for a in support:
      prior_a = belief[a]
      likelihood = calc_likelihood(a, obs)
      belief[a] = prior_a * likelihood
    normalize(belief)
```

In plain English: this is the numerator, summed over all values of A

$$P(A=a|Y=0) = \frac{P(Y=0|A=a)P(A=a)}{P(Y=0)}$$

$$= \frac{P(Y=0|A=a)P(A=a)}{\sum_{x \in A} P(Y=0, A=x)}$$

$$= \frac{P(Y=0|A=a)P(A=a)}{\sum_{x \in A} P(Y=0|A=x)P(A=x)}$$

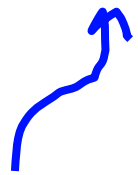
Inference

In general Bayes theorem with a random variable is like the cellphone problem: multiple possible assignments to keep track of

Still true when some variables are continuous

Random Variables

Not all beliefs can be represented as a **function**. **Dictionary** / table is a great way to represent a random variable belief.



This is formally called non-parametric

Representing Continuous Variables

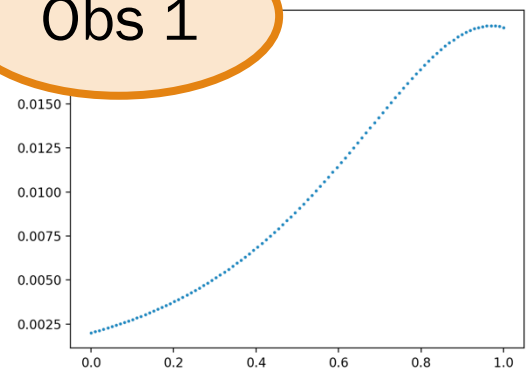
Dictionary can also be used to represent a discretization of a **continuous random var**

I do it all the time! Yay compute!

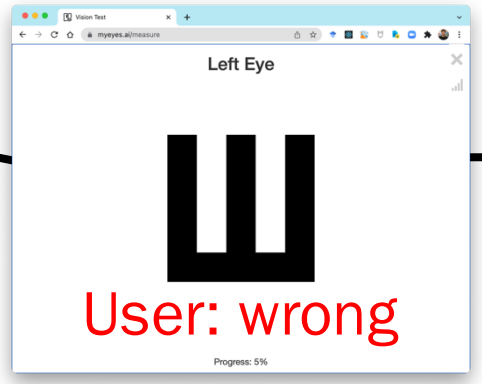
Multiple observations??

Multiple Observations

Obs 1

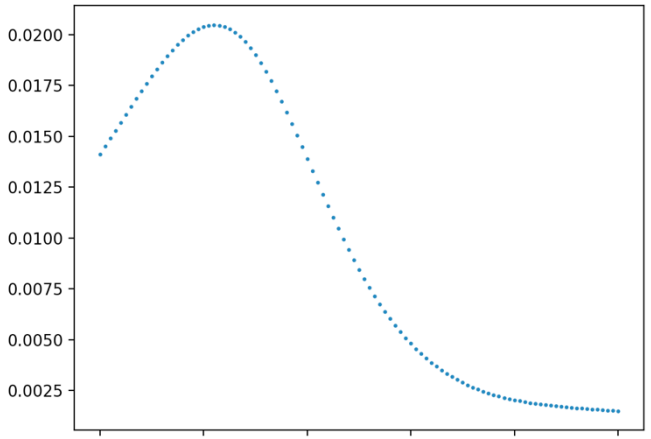


$P(A = a)$



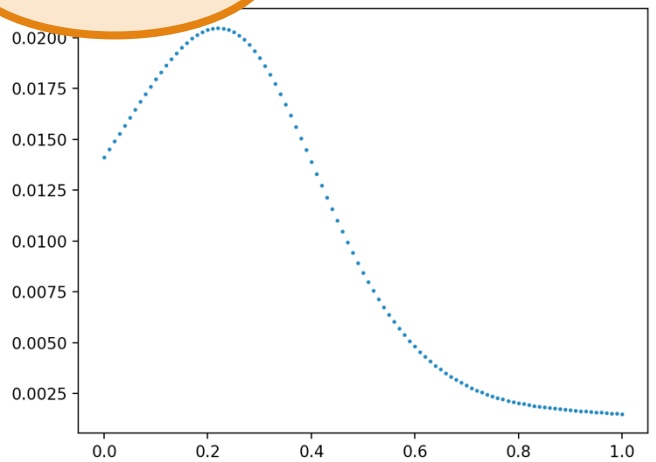
Observation $Y = 0$

(At font size 0.7)

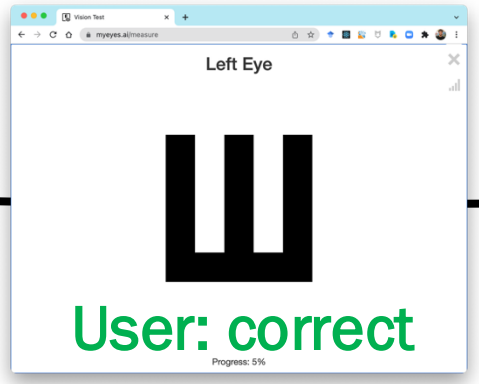


$P(A = a|Y = 0)$

Obs 2

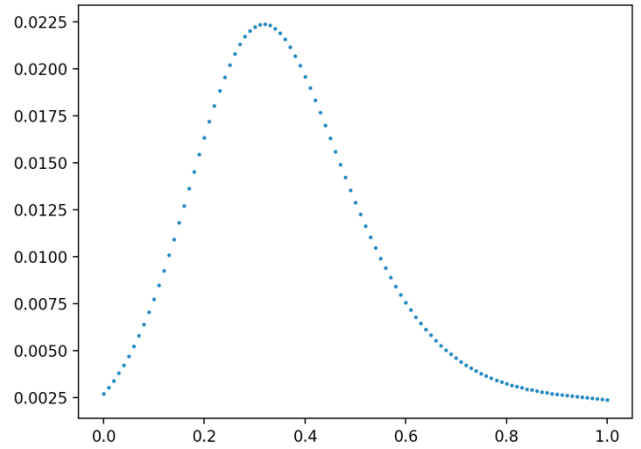


$P(A = a)$



Observation $Y = 0$

(At font size 0.8)



$P(A = a|Y = 1)$

Multiple Observations

Single Observation:

$$P(A = a|R_1) = P(R_1|A = a) \cdot P(A = a) \cdot K$$

Multiple Observations:

$$P(A = a|R_1, R_2) = P(R_1, R_2|A = a) \cdot P(A = a) \cdot K_2$$
$$= P(R_2|A = a) \cdot P(R_1|A = a) \cdot P(A = a) \cdot K_2$$

Multiple Observations

Single Observation:

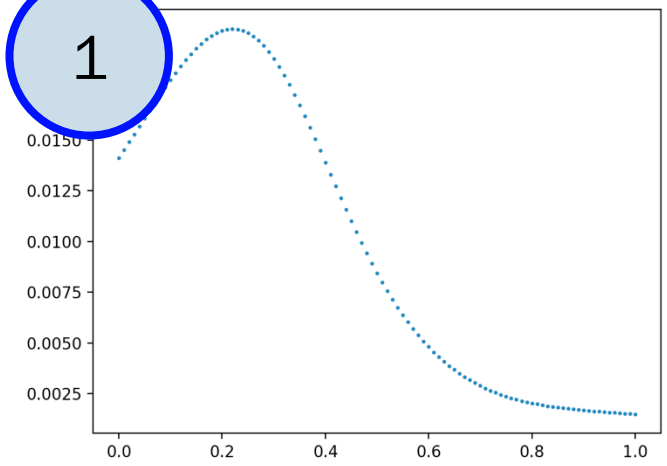
$$P(A = a|R_1) = P(R_1|A = a) \cdot P(A = a) \cdot K$$

Multiple Observations:

$$P(A = a|R_1, R_2) = P(R_1, R_2|A = a) \cdot P(A = a) \cdot K_2$$
$$= P(R_2|A = a) \cdot P(R_1|A = a) \cdot P(A = a) \cdot K_2$$

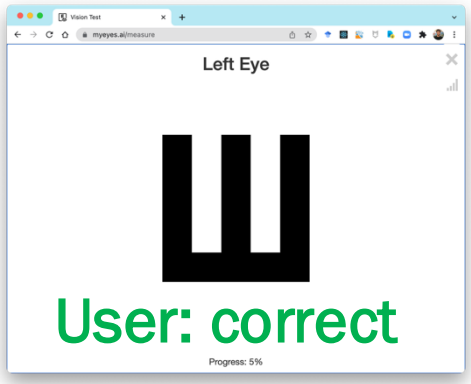
Posterior becomes new prior

1



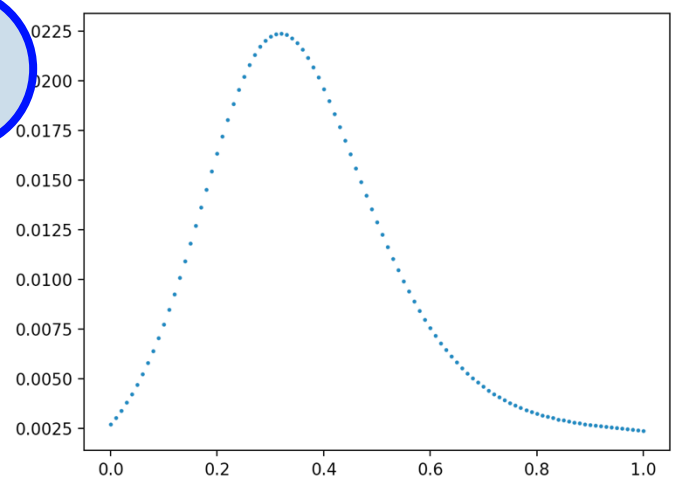
$P(A = a)$

2



Observation $Y = 1$
(At font size *0.8*)

3



$P(A = a|Y = 1)$

$$P(A = a|Y = 1) = \frac{P(Y = 1|A = a)P(A = a)}{P(Y = 1)}$$

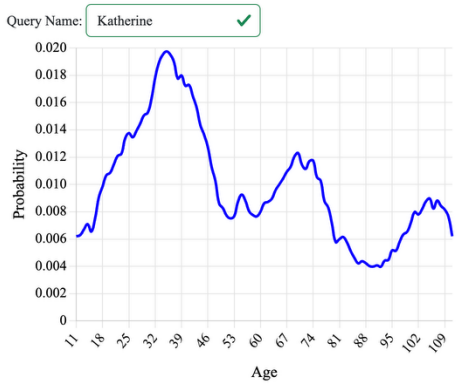
Beyond Inference:
How do you select the next size to show?

Today: Five New Real + Exciting Problems

Age from C14

Updated Delivery Prob

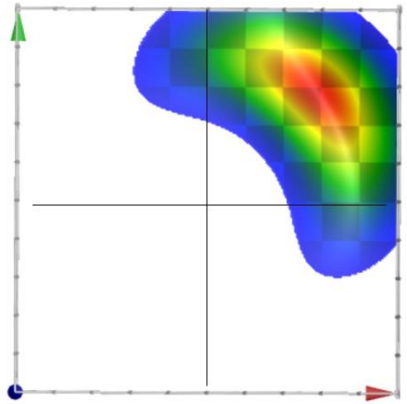
Age from Name

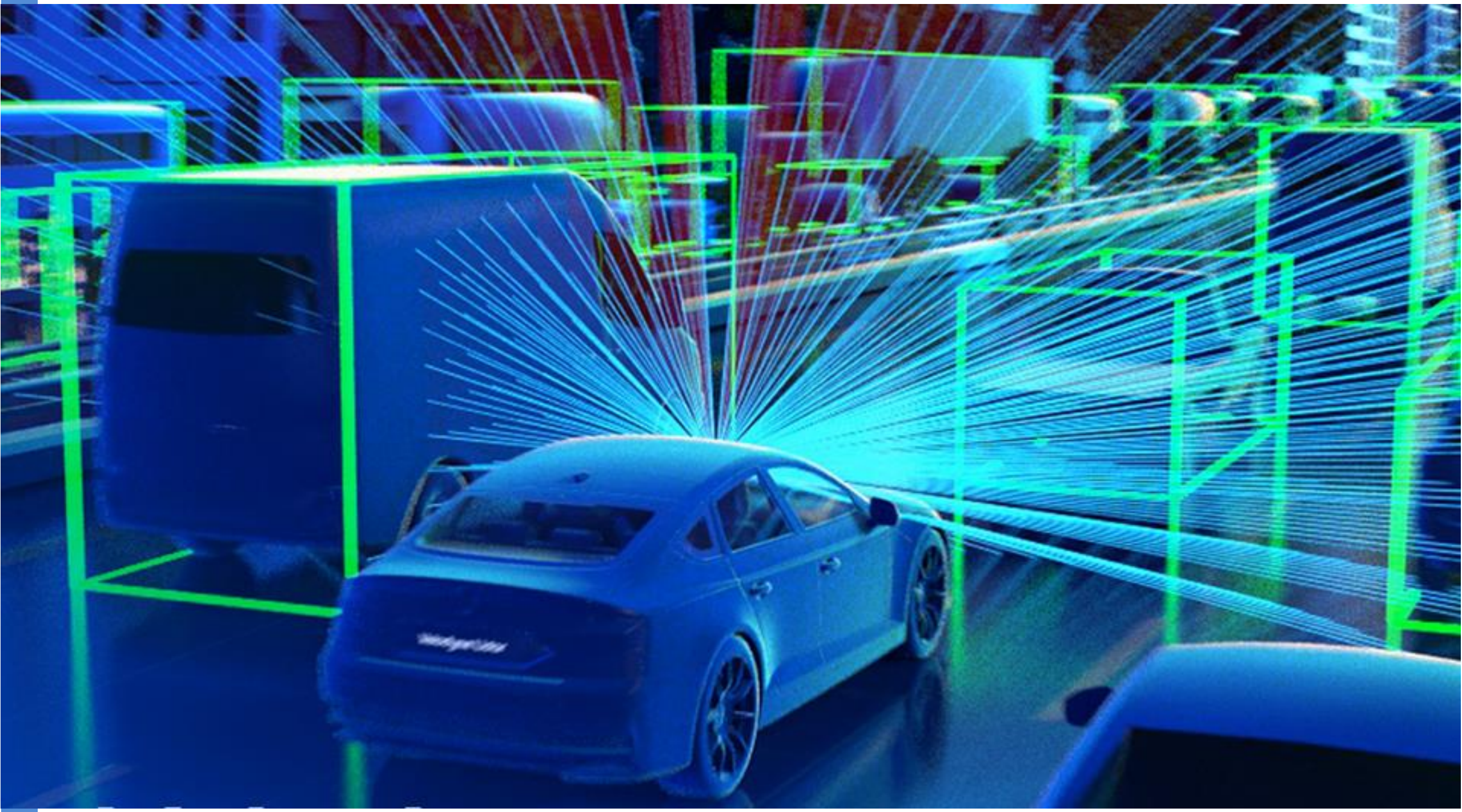


Hidden Chambers

Stanford Eye Test

Cellphone Tracking Cont.



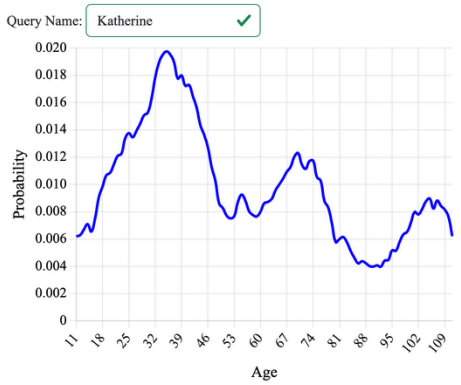


Today: Compare and Contrast Many Examples

Age from C14

Updated Delivery Prob

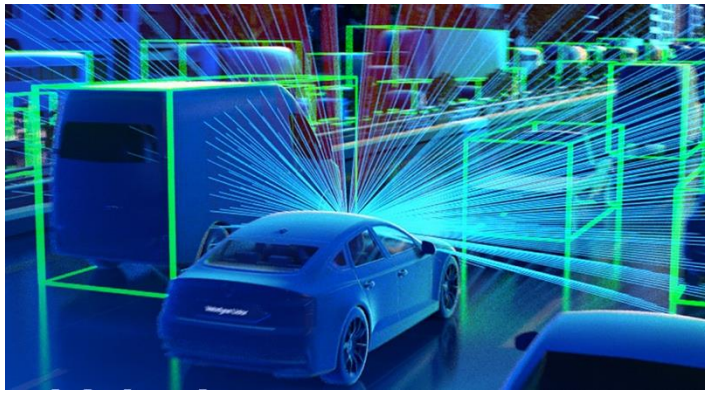
Age from Name



Hidden Chambers

Stanford Eye Test

Updating Lidar Belief



Practice!

psetapp.stanford.edu

PS4

Chess.com Puzzles

Chess.com is a website for playing chess. They are trying to estimate how well a player can solve chess puzzles (puzzle ability) as a random variable, A, which can take on integer values in the range 0 to 100 inclusive. Higher abilities mean the player is better at chess puzzles. Note that ability is **discrete**.

Write a function `update_belief` which takes in a prior belief in a player's puzzle ability and an observation of them solving a puzzle. Your function should infer the posterior belief in the player's ability, based on the observation using Bayes' Theorem (with random variables).

Representation of Belief

Both the `prior` and the `posterior` you return should be probability mass functions for ability. These probability mass functions are represented as a dictionary where the keys are all the values that ability can take on [0, 100]. The value corresponding to key `i` represents $P(A = i)$. The posterior that you return should be a dictionary with the exact same keys, where the value corresponding to key `i` represents $P(A = i|Y = y)$. Y is a Bernoulli random variable which is 1 if the player answered the puzzle correct.

Previous Question | Next Question

Answer Editor | Solution

Python:

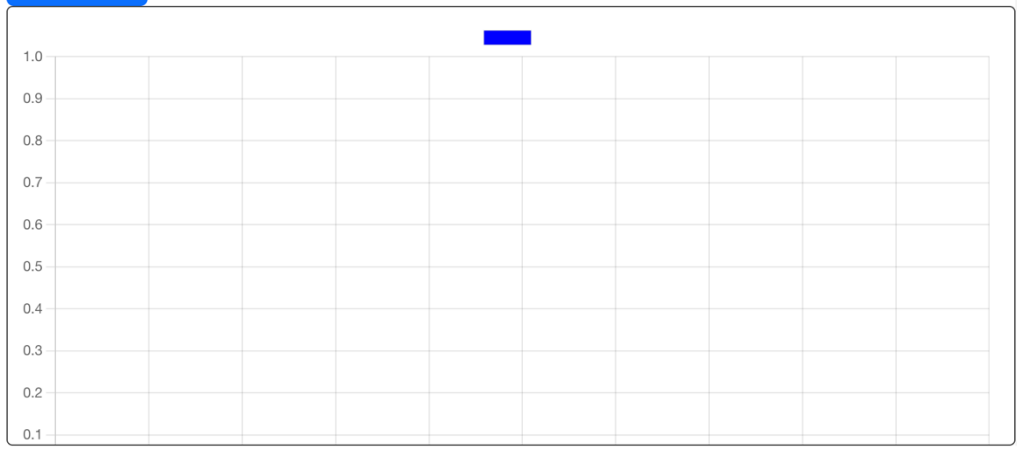
```python
import math

def update_belief(prior, observation):
    # TODO: your code here
    return prior

################################################
# Helper Functions!
################################################

def p_correct_given_ability(ability, difficulty):
    """
    This uses item response theory to model the chance that a
    patient with a given ability will correctly solve a chess
    puzzle
    """
    p_guess = 0.05
    p_slip = 0.08
    scaling = 0.25
```

Run



See you Monday!