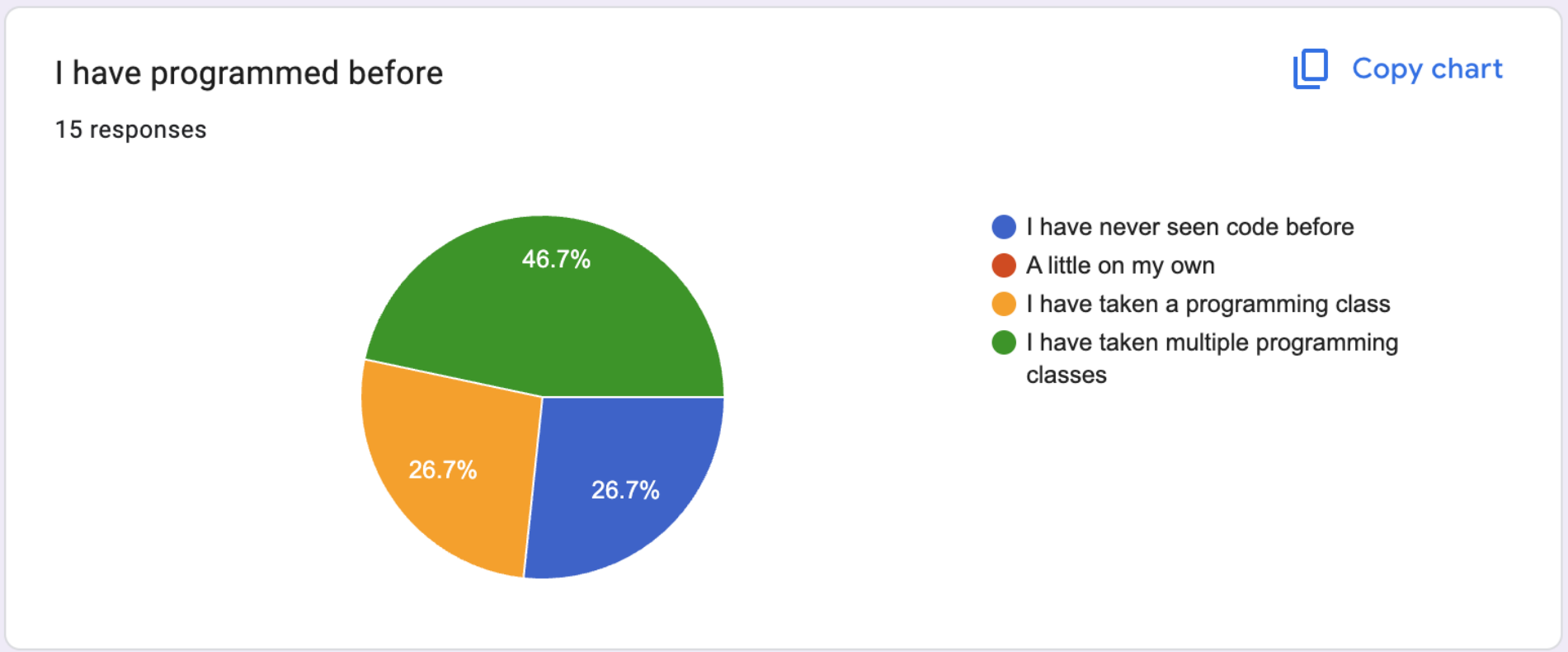
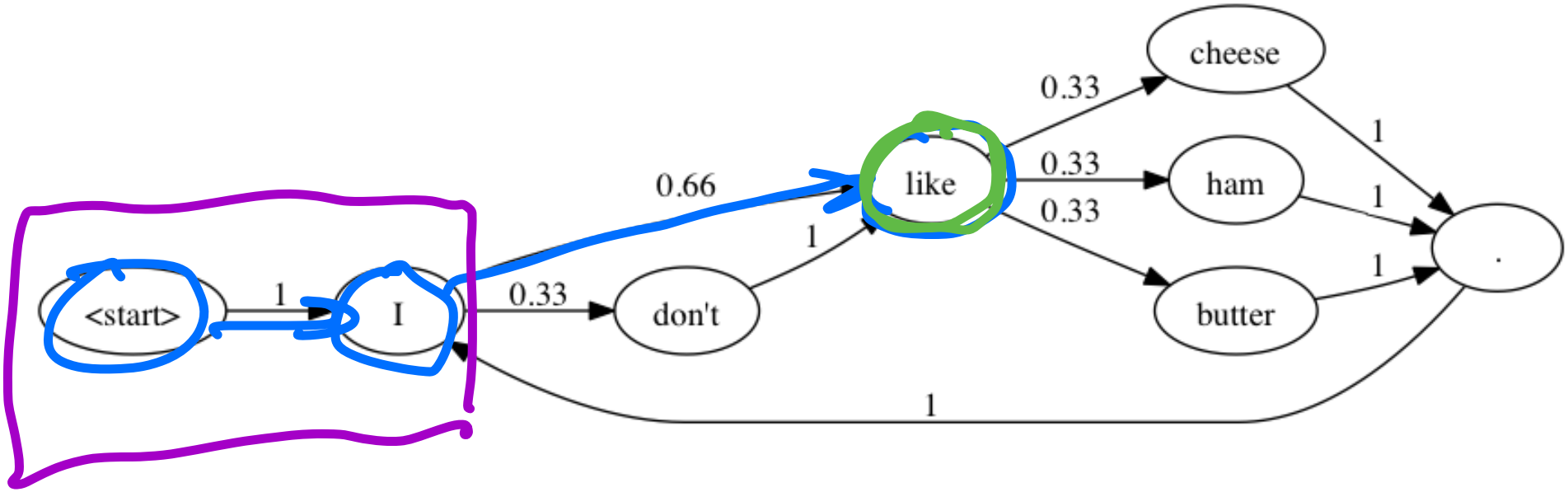
CSE 10124
Text as Data

Intro Survey Results



- **For those of you new to programming, be humble enough to ask questions!**
- For those of you that are coding prodigies, think and ask about the algorithms!

Last Class

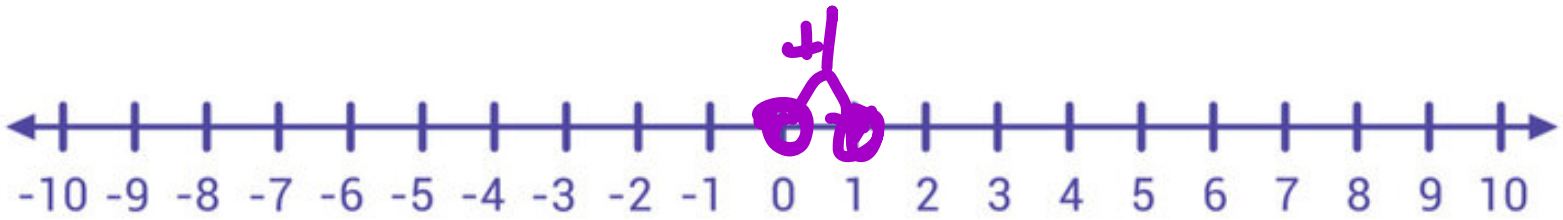


Markov Babbler

<start> | I like

Introduction to Markov Processes

A systematic method for **generating a sequence of random variables** where the current value is probabilistically dependent on the value of the prior variable



Specifically, **selecting the next variable is dependent upon** the previous state

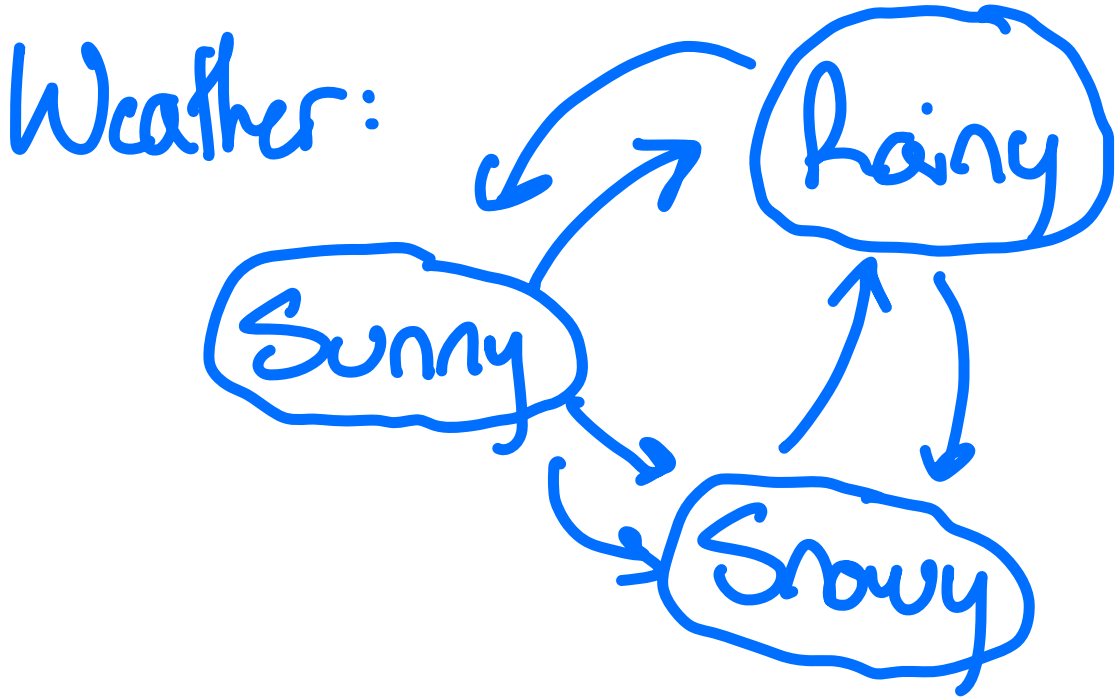
Markovian Terms

The **changes of state** of the system are called transition and the probabilities associated are called transition probabilities

The **process is characterized by**:

1. a state space
2. a transition matrix
3. a initial state

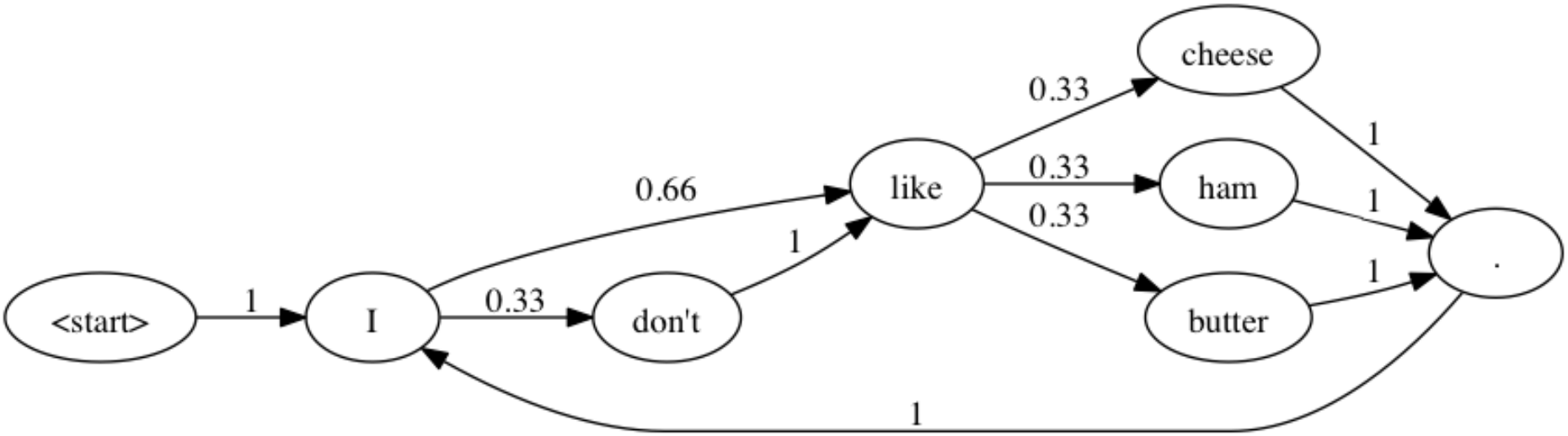
Markov Chains



Many things can be modelled as Markov Chains

Markov Babbler

“<SOS> And yet I got ganked while questing. <EOS>”



Creating our Graph

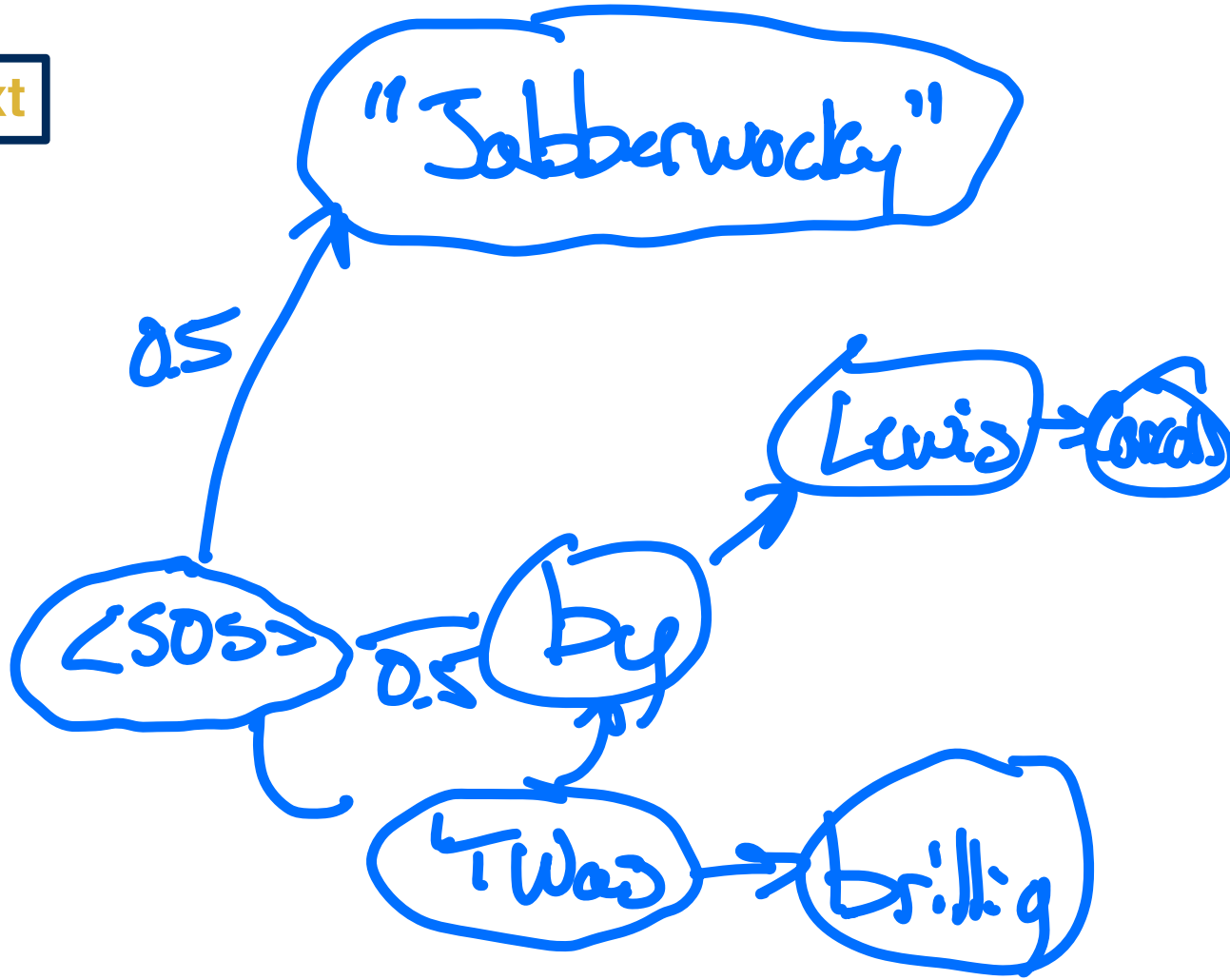
jabberwocky.txt

"Jabberwocky"
by Lewis Carroll

'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.

"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"

He took his vorpal sword in hand:
Long time the manxome foe he sought—
So rested he by the Tumtum tree



<SOS> 'Twas by . .

Creating our Graph

shakespeare.txt

From fairest creatures we desire increase,

That thereby beauty's rose might never die,

But as the riper should by time decease,

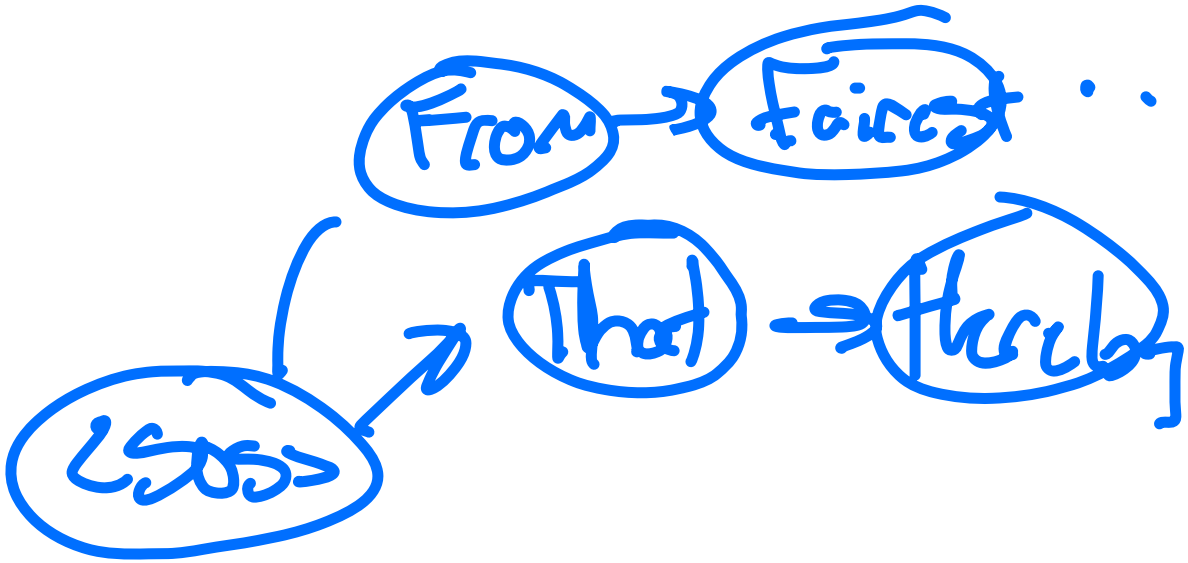
His tender heir might bear his memory:

But thou contracted to thine own bright eyes,

Feed'st thy light's flame with self-substantial fuel,

Making a famine where abundance lies,

Thy self thy foe, to thy sweet self too cruel:



Differences

The word "Brillig" never appears in Shakespeare so we'd never see our Model say it! It can only say words it has seen before.

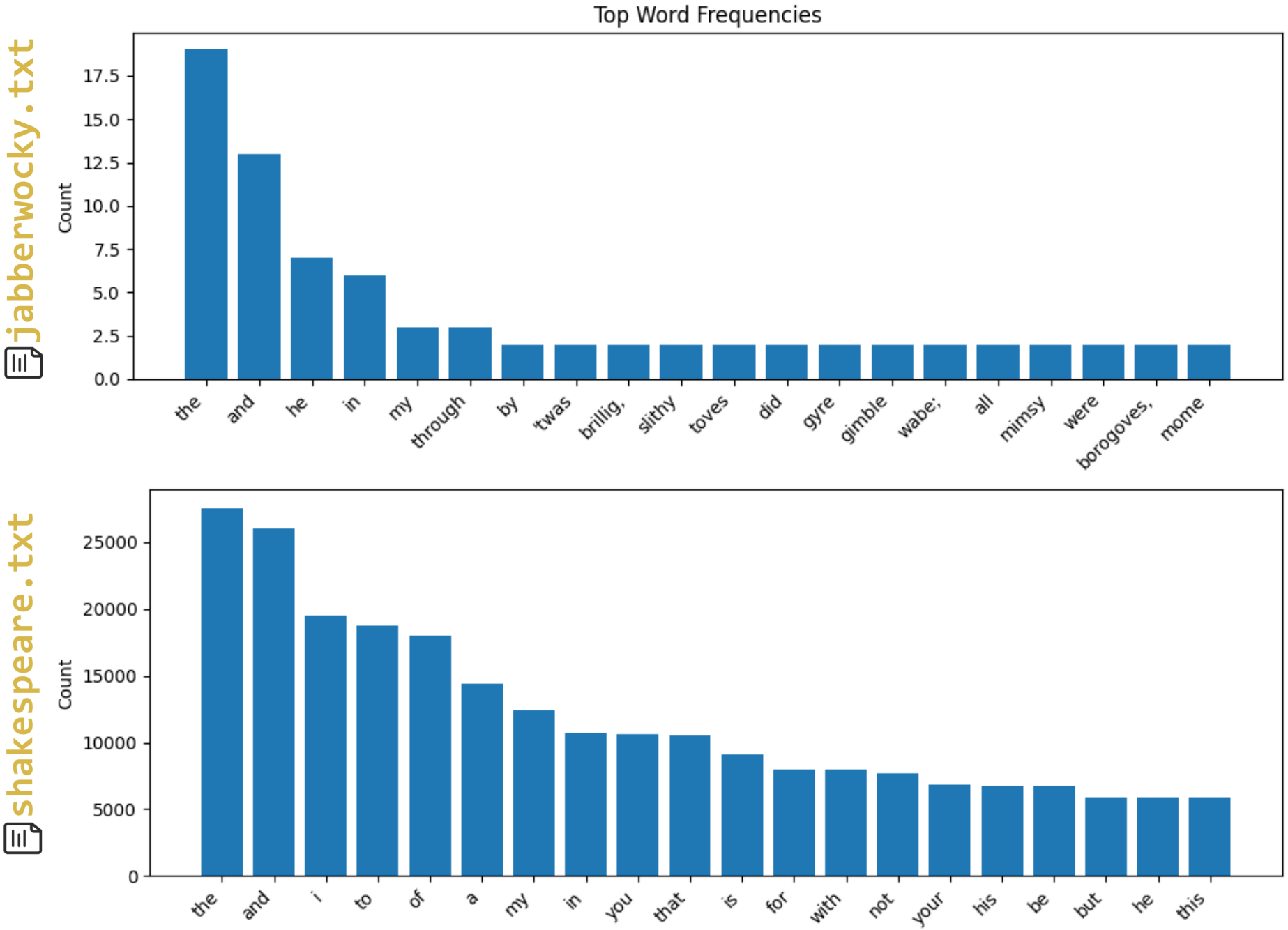
Statistical Distributions

A **statistical distribution** describes the likelihood of diffent outcomes

shakespeare "brillig?" → 0%

One way you can think about what **AI is** doing is it's **trying to learn distributions**

jabberwocky.txt vs. shakespeare.txt

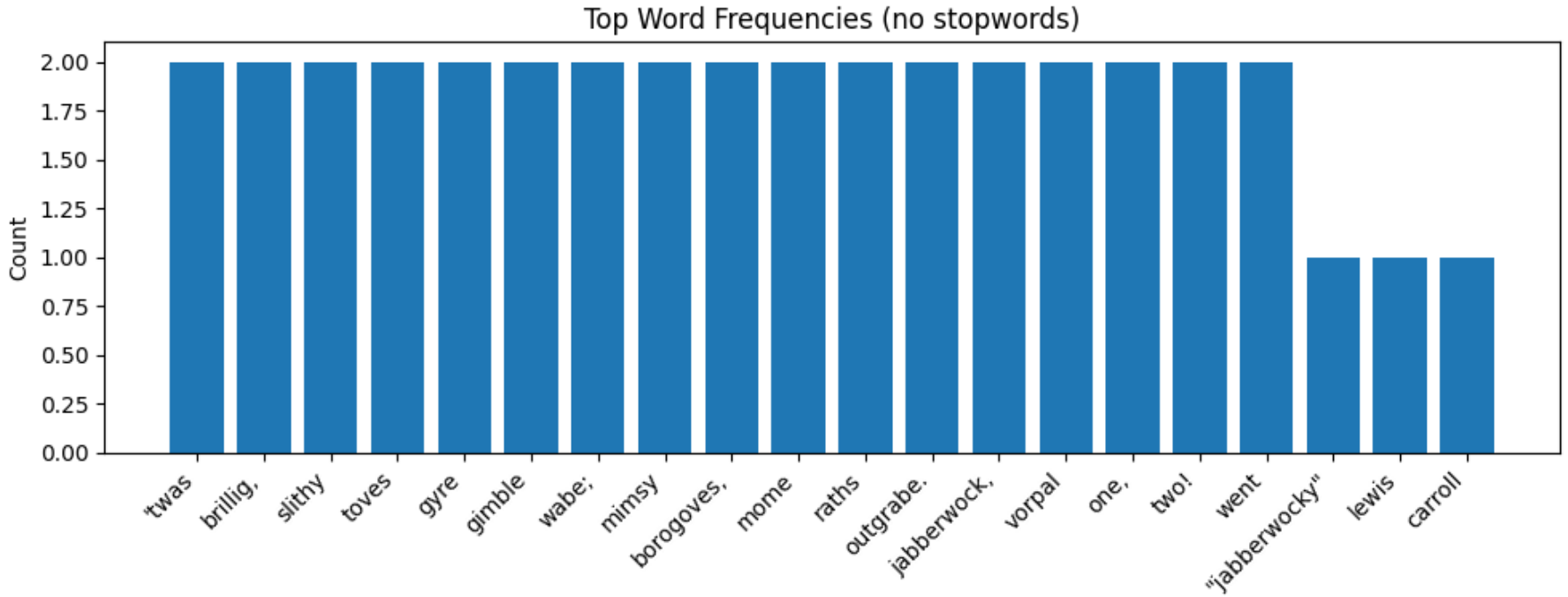


Stop Words

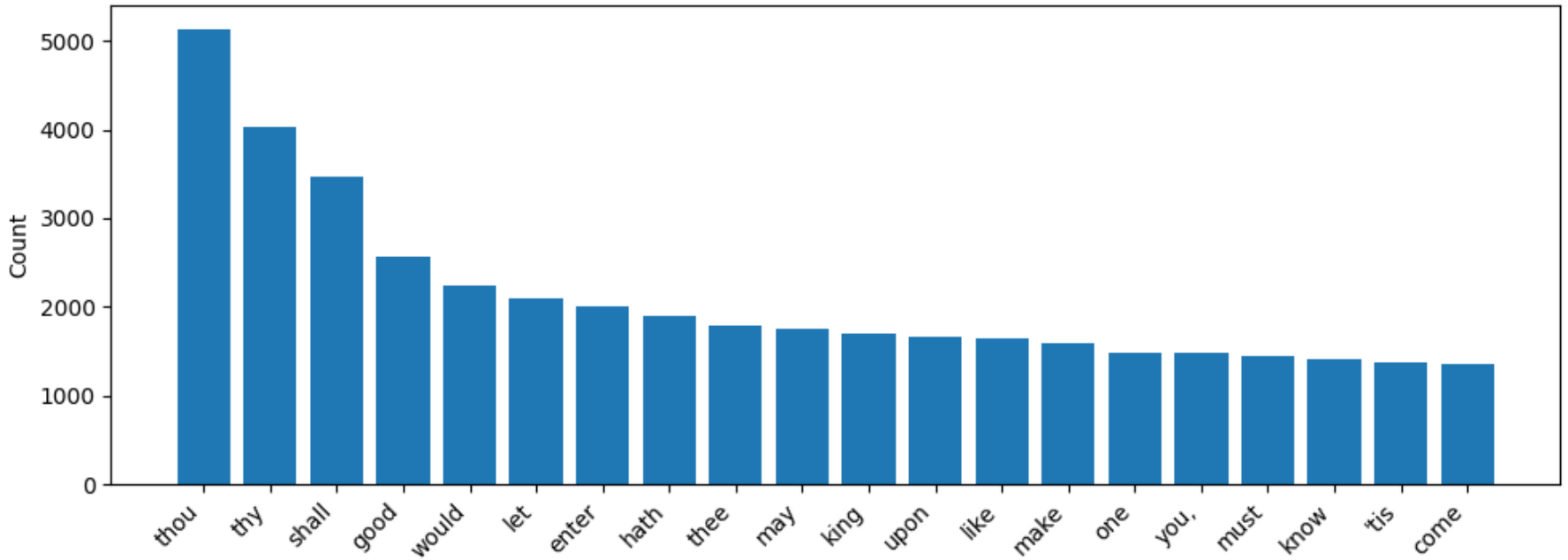
Stop words are **common, high-frequency words that often carry little specific meaning** and are **filtered out** in Natural Language Processing (NLP) systems

jabberwocky.txt vs. shakespeare.txt

jabberwocky.txt



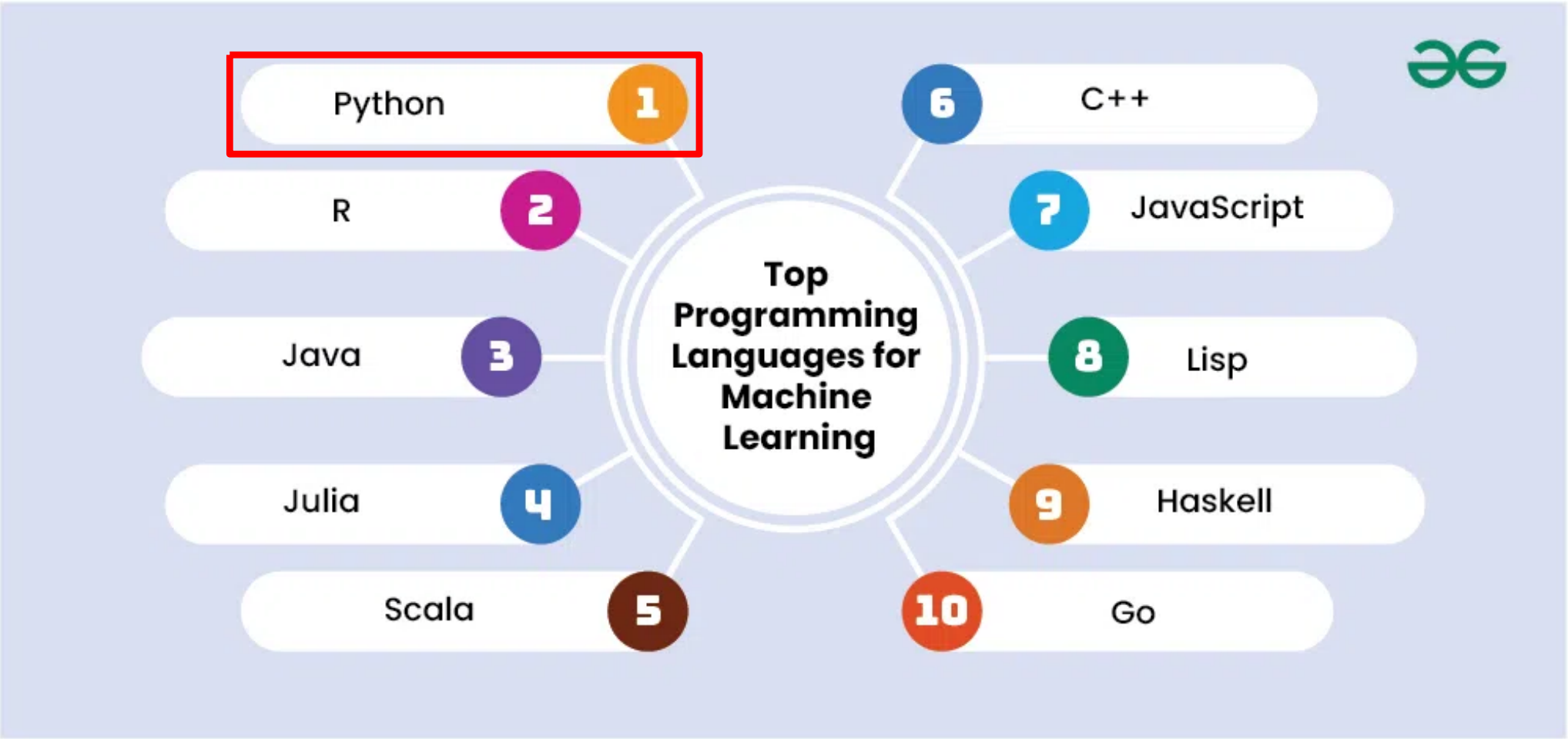
shakespeare.txt



Sampling Distributions and Generative AI

"**Generative artificial intelligence** is a subset of artificial intelligence that uses generative models to produce text, images, videos, or other forms of data. **These models learn the underlying patterns and structures of their training data** and use them to produce new data" — Wikipedia

Programming in Python



Python Programming

A Crash Course

Variables

Variables allow us to **store data** using a convenient name.

```
meow = 'cat'
print(meow)
print('kitty' + meow)

num = 67
print(num)
num = 5

num_2 = 3
print(num + num_2)

boolean = True
print(boolean & True)
```

```
cat
kittycat
67
8
True
```

Types

Variables have a **type**:

- **int** (a whole number)
- **str** (a “string”)
- **bool** (true or false)

and many others!

```
print(type(meow))
print(type(num))
print(type(boolean))
```

```
<class 'str'>
<class 'int'>
<class 'bool'>
```

```
meow = 'cat'
print(meow)
print('kitty' + meow)

num = 67
print(num)
num = 5

num_2 = 3
print(num + num_2)

boolean = True
print(boolean & True)
```

```
cat
kittycat
67
8
True
```

Conditionals

Often times we want to **check the value of a variable, and do different things** depending on what it is. The most common **conditional** is the **if statement**

```
i_like_cats = True

if i_like_cats:
  print('meow')

if i_like_cats:
  print('meow')
else:
  print('woof')
```

```
meow
meow
```

Data Structures

In addition to basic things like **str** and **int**, we can store more complex things called **data structures** that may allow us to store multiple values in a **variable**

```
animals = ['cat', 'dog', 'fish']
print(type(animals))
print(animals)
```

```
<class 'list'>
['cat', 'dog', 'fish']
```

The simplest data structures is called a **list**

Lists

lists can hold just about anything you would want, even other **lists**!

```
animals = ['cat', 'dog', 'fish']
print(type(animals))
print(animals)
print(animals[1])
```

NOTE: the first item in a **list** is **index 0**, NOT **1**!

```
<class 'list'>
['cat', 'dog', 'fish']
dog
```

We can access specific items using **square brackets** and an "**index**"

Dictionaries

```
animal_count = {
    'cat': 5,
    'dog': 3,
    'fish': 1
}

print(type(animal_count))
print(animal_count['cat'])
```

```
<class 'dict'>
5
```

```
animal_noises = {
    'cat': 'meow',
    'dog': 'bark',
    'fish': 'blub'
}

print(type(animal_noises))
print(animal_noises['fish'])
```

```
<class 'dict'>
blub
```

Dictionaries store groups of two things, called: **"key - value pairs"**. To access items in a dictionary we use **square brackets** again, but with a **value** in the brackets instead of an index

Loops

When we have a data structure, we often want to **loop** over it. The most common loop is the **for loop**

```
for animal in animals:
  print(animal)

for animal, noise in animal_noises.items():
  print(animal, noise)
```

```
cat
dog
fish
cat meow
dog woof
fish blub
```

Functions

Often times we will want to **run the same set of instructions on different variables**, we can define a **function** to achieve this

```
def get_lines_list(source_file):
    lines = []
    with open(source_file) as f:
        for line in f:
            lines.append(line.strip().split())

    return lines
```

Comments

```
# This is a single line comment
# print('meow')

"""
This is a multiline comment
print('meow')
"""


print('meow')
```

```
meow
```

Oftentimes while programming you will want to leave **explanatory text in the code**, these are called **comments**. In python we do this with either a **pound symbol: #** if the comment is a single line, or **triple quotes: “””** if the comment is multiline

Creating our Graph: Code

```
def build_graph_word(lines, graph=None):
  if not graph:
    graph = defaultdict(Counter)

  for line in lines:
      if line:
        graph['<SOS>'][line[0]] += 1

        for idx in range(0, len(line) - 1):
          curr_token = line[idx]
          next_token = line[idx + 1]

          graph[curr_token][next_token] += 1

        graph[line[-1]]['<EOS>'] += 1

  return graph
```

<SOS> and <EOS>

<SOS> — **S**tart **o**f **S**equence

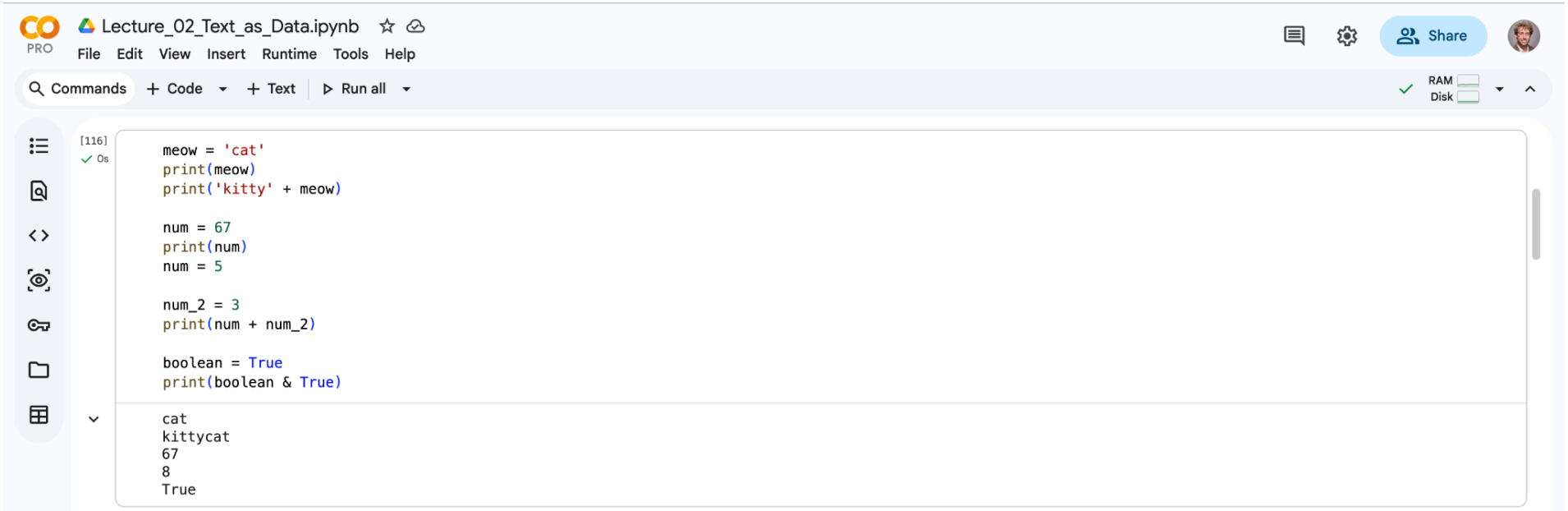
<EOS> — **E**nd **o**f **S**equence

These are special tokens often seen when working with text, another one is:

<UNK> — **Unk**nown

<UNK> is used to represent words that we haven't seen in our vocab before

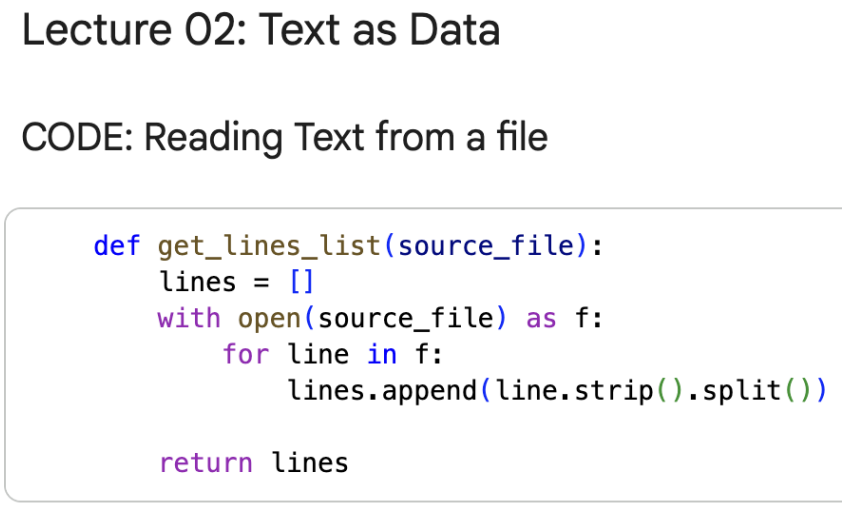
Google Colab



Google Colab lets you run **python** code in your online and integrates with your **Google Drive**! This is primarily what we'll be using in this class.

.ipynb Files

We're going to use a type of file called a **notebook**



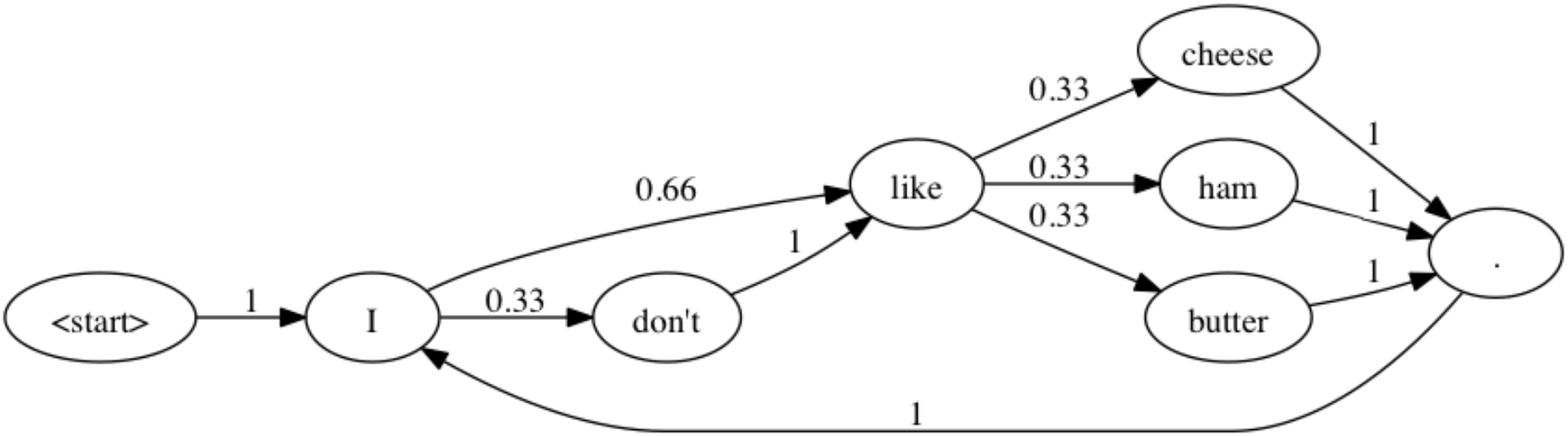
These let you put text and code alongside each other and can display graphs from your code right away. Most data scientists and AI researchers use notebooks every day.

Our First Notebook

Let's open our notebook for today’s lecture and get started!

Notebook

Generating Text from our Graph



We can generate new sentences from our model by **providing it with a "prompt" to get started** and then letting it **probabilistically travel through the graph**.

Character-Level Graph

jabberwocky.txt

"Jabberwocky"

by Lewis Carroll

'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.

"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"

He took his vorpal sword in hand:
Long time the manxome foe he sought—