



Applied Rest APIs II

Agenda - Schedule

1. REST APIs Review
1. OpenAI API Intro
2. Break
3. OpenAI API Lab



JavaScript Object Notation (JSON) is an open-standard data format or interchange for semi-structured data. It is text-based and readable by humans and machines.
<https://www.snowflake.com/guides/what-is-json>



Agenda - Announcements

- Week 4 Pre-Class Quiz due 4/1
- TLAB #2 Due 4/21
- Office hours posted

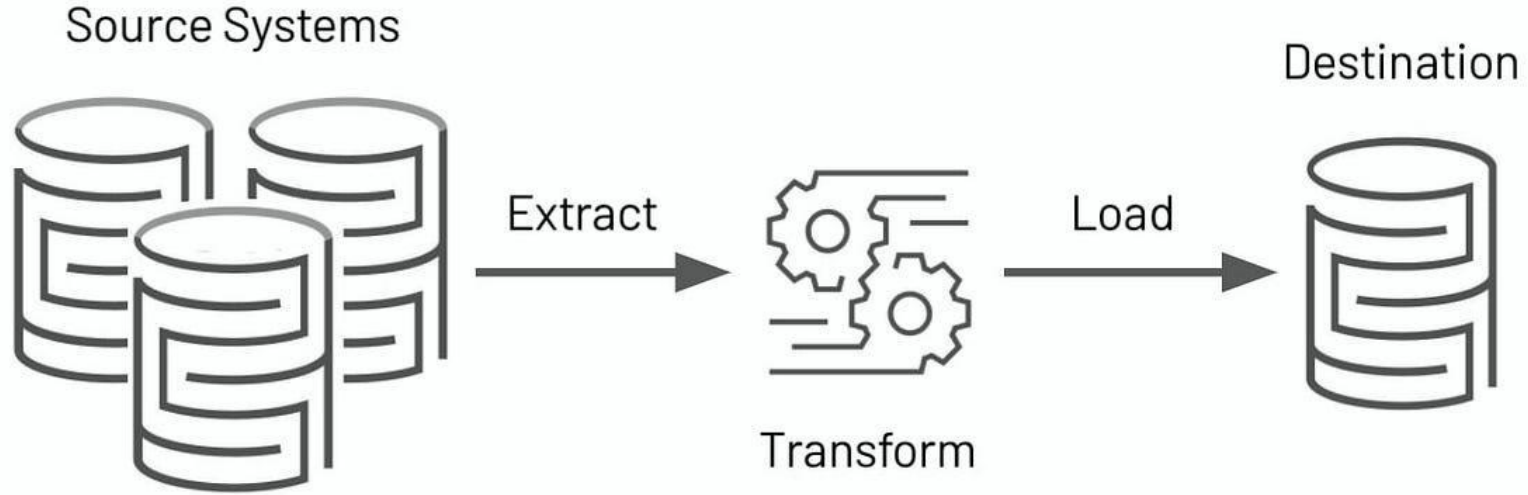


Agenda - Goals

- Review Rest API usage
- Get introduced to LLM technology
- Learn how to use the OpenAI API
- Discover the principles of effective requests

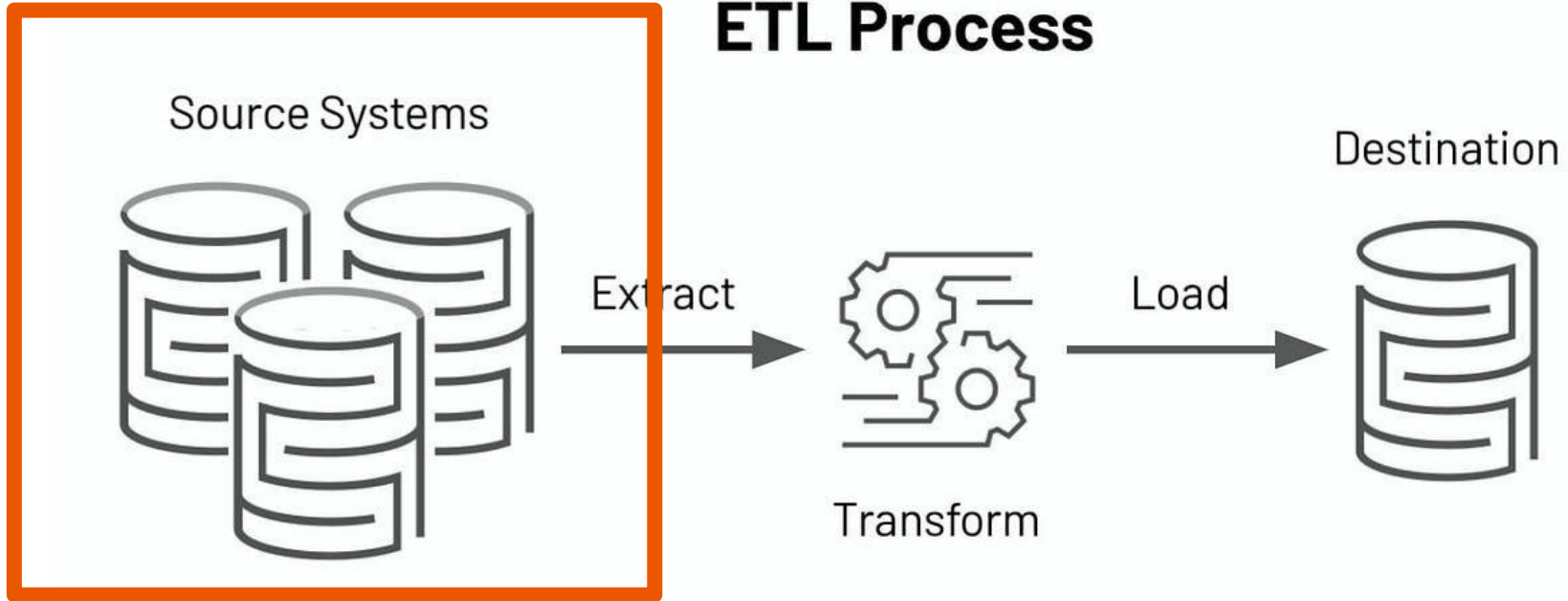
Web APIs Review

ETL Process



As we discussed, one typical pipeline that in the data engineering process is the **extract-transform-load (ETL)** pipeline.

ETL Process



This week, we will go over various ways to extract data via **Web APIs** and **Web Scraping**. Afterwards, we will go over techniques we can use to transform data for easy analysis and prediction.

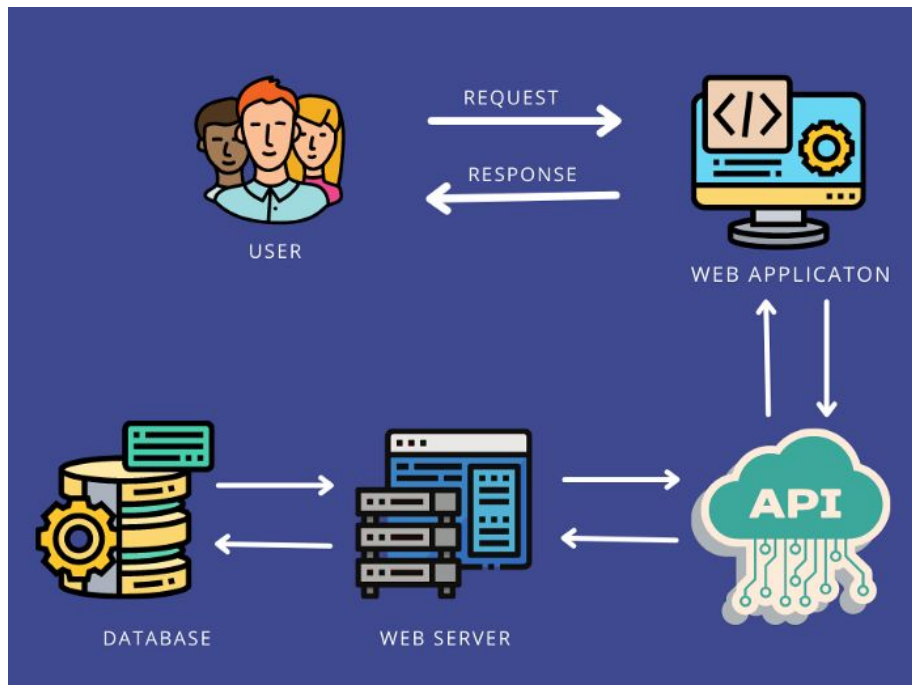
The term “API” can be used in multiple contexts:

pandas API:

An interface to use when coding in pandas

Web API:

An interface when requesting data over the web



Today, we will review the **RESTful Web API**. This describes a programmatic way to “*ask respectfully*” for data.



Requests & APIs

Let's review our REST API terminology. Can anyone identify the components of this URL?

`https://pokeapi.co/api/v2/pokemon/pikachu`



Requests & APIs

The **base URL** (aka the **Request URL**) is the **URL** that points to your **API**.
This serves as your initial entry-point.

Access the pokemon Web API...

<https://pokeapi.co/api/v2/pokemon/pikachu>



Requests & APIs

Next, we have the **endpoint** which describes where the data that you want to interact with is located.

Notice the similarities between the endpoint and file-paths... (e.g. *TLAB2/data/averages.png*)

...and give me information on the
pokemon named pikachu

<https://pokeapi.co/api/v2/pokemon/pikachu>

Try it now!

<https://pokeapi.co/api/v2/>

[pokemon/pikachu](https://pokeapi.co/api/v2/pokemon/pikachu)



Submit

Need a hint? Try [pokemon/ditto](#), [pokemon-species/aegislash](#), [type/3](#), [ability/battle-armor](#), or [pokemon?limit=100000&offset=0](#).

Direct link to results: <https://pokeapi.co/api/v2/pokemon/ditto>

Resource for ditto

```
▼ abilities: [] 2 items
  ▼ 0: {} 3 keys
    ▼ ability: {} 2 keys
      name: "limber"
      url: "https://pokeapi.co/api/v2/ability/7/"
      is_hidden: false
      slot: 1
  ▼ 1: {} 3 keys
    ▼ ability: {} 2 keys
      name: "imposter"
      url: "https://pokeapi.co/api/v2/ability/150/"
      is_hidden: true
      slot: 3
```

But keep in mind, you **will** always use the **API documentation** to structure your URL: <https://pokeapi.co/>.

Which points back to the **importance of good documentation**: <https://www.ncei.noaa.gov/support/access-data-service-api-user-documentation>



Requests & APIs

The previous example describes a simple public API which entails a minimal URL.

This is rarely the type of URL you will be accessing for your projects, instead you will be accessing URLs that have **complex queries**, such as the following API:

<https://api.polygon.io/v2/aggs/ticker/AAPL/range/1/day/2023-01-09/2023-02-10?adjusted=true&sort=asc&apiKey=123456>



Requests & APIs

We still have our **base URL**...

Access the polygon Web API...

<https://api.polygon.io/v2/aggs/ticker/AAPL/range/1/day/2023-01-09/2023-02-10?adjusted=true&sort=asc&apiKey=123456>



Requests & APIs

As well as the **endpoint...**

...and give me aggregates on the AAPL
ticker from Jan 1 2023 to Feb 10 2023

<https://api.polygon.io/v2/aggs/ticker/AAPL/range/1/day/2023-01-09/2023-02-10?adjusted=true&sort=asc&apiKey=123456>



Requests & APIs

But this additional section describes the **query**, which provides additional parameters that allows you to **retrieve** a modified version of the resource that you are requesting, as well as **provide** required access information (*for example API keys*)

We combine multiple queries using the **&**:

<https://api.polygon.io/v2/aggs/ticker/AAPL/range/1/day/2023-01-09/2023-02-10?adjusted=true&sort=asc&apiKey=123456>



Requests & APIs

The following queries provide the following parameters:

- **adjusted=true**: Return the adjusted price
- **sort=asc**: Sort these dates in descending order
- **apiKey=123456**: Here's my secret key

<https://api.polygon.io/v2/aggs/ticker/AAPL/range/1/day/2023-01-09/2023-02-10?adjusted=true&sort=asc&apiKey=123456>

Aggregates (Bars)

GET /v2/aggs/ticker/{stocksTicker}/range/{multiplier}/{timespan}/{from}/{to}

Get aggregate bars for a stock over a given date range in custom time window sizes.

For example, if timespan = 'minute' and multiplier = '5' then 5-minute bars will be returned.

Read documentation for profit and success.

Parameters

stocksTicker* AAPL

Specify a case-sensitive ticker symbol. For example, AAPL represents Apple Inc.

multiplier* 1

The size of the timespan multiplier.

timespan* day

The size of the time window.

from* 2023-01-09

The start of the aggregate time window. Either a date with the format YYYY-MM-DD or a millisecond timestamp.

to* 2023-02-10

The end of the aggregate time window. Either a date with the format YYYY-MM-DD or a millisecond timestamp.

Once again you are not expected to construct these queries yourself. Instead, you **will always use the API documentation:**

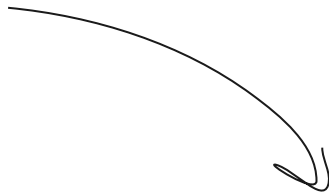
https://polygon.io/docs/stocks/get_v2_aggs_ticker_stocksticker_range_multiplier_timespan_from_to

```
r = requests.get("https://pokeapi.co/api/v2/pokemon/pikachu")
```

```
data = r.json()
```

We can then use these API urls with our **requests** module to **automatically** extract information in our Python program.

```
for ab in data["abilities"]:  
    print(ab)
```



```
{  
  'ability': {'name': 'static', 'url': 'https://pokeapi.co/api/v2/ability/9/'}, 'is_hidden': False, 'slot': 1}  
  'ability': {'name': 'lightning-rod', 'url': 'https://pokeapi.co/api/v2/ability/31/'}, 'is_hidden': True, 'slot': 3}  
}
```

+ Code

+ Markdown

OpenAI API



OpenAI API

For this next lab, we will have you to interact with the **OpenAI** API to send human-reviews for machine **sentiment labeling**.

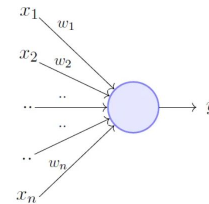
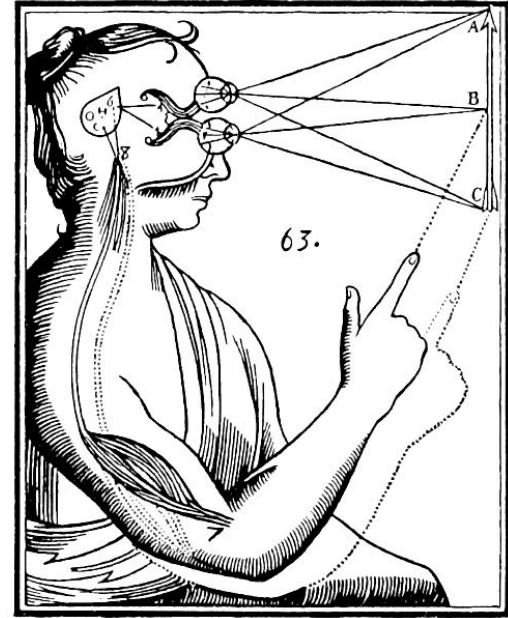
However, before we dive into the syntax, let's go over a **brief overview of the context and expected usage** of **large language model** technology.

There's a lot of mixed messaging regarding LLMs, so we would like to emphasize a restrained look on how you should use LLMs in your workplace.

Neural Networks

In Phase 2 of this fellowship, we will learn about a type of machine learning algorithm called a **perceptron**.

This takes inspiration from the neurons in our brain, but **purely exist as a mathematical function**.



$$\hat{y} = W^T X$$

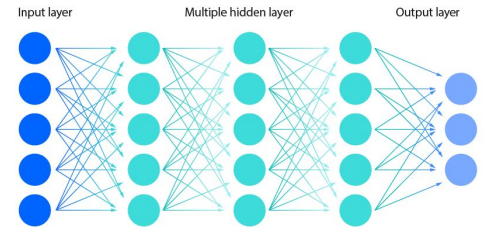
Neural Networks

We can organize these perceptrons in specific **architecture** to have them perform some impressive tasks such as:

- Computer vision
- Numeric prediction
- Classification
- Natural language processing



Deep neural network

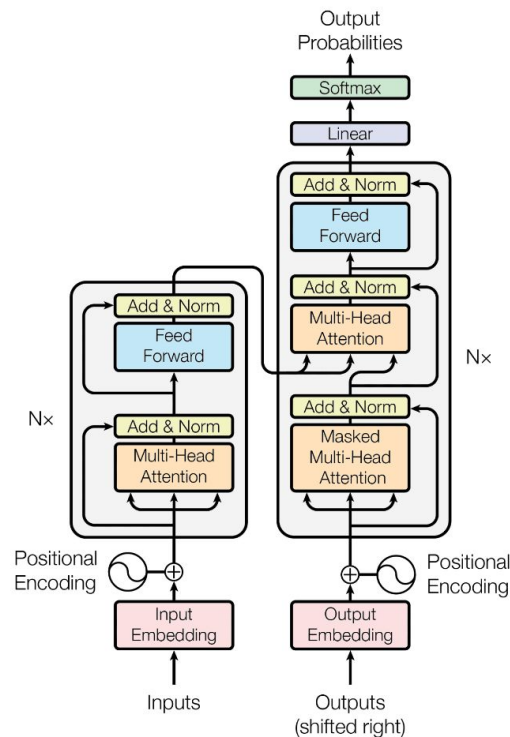


Neural Networks - Transformers

Back in 2017, Google researchers published a paper titled *Attention is All you Need* to introduce a neural network architecture to **translate text between different languages**.

In summary, this proposes a model for probabilistic **text prediction**. That is, the model is selecting the next word (*token*) based on **probabilities**.

This eventually became ChatGPT.



"I hate carrots."

"Can I buy robux."

*"Carrots are
gross."*

*"I don't want to
sleep."*

"I hate carrots."

*"Can I have my
Ipad."*

"I hate carrots."

For example, let's say you are babysitting your niece/nephew/cousin etc. You've spent a few days with them and there are certain phrases you hear them say over and over again.

"I hate carrots."

"Can I buy robux."

"Carrots are gross."

"I don't want to sleep."

"I hate carrots."

"Can I have my Ipad."

"I hate carrots."



"I hate ..."

You hear them begin a sentence again with "I hate..." Based on your historical data, what do you predict is going to be the next word of this sentence.

"I hate carrots."

"Can I buy robux."

"Carrots are gross."

"I don't want to sleep."

"I hate carrots."

"Can I have my Ipad."

"I hate carrots."



"I hate carrots."

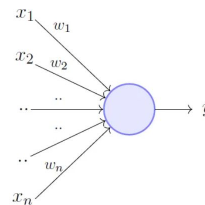
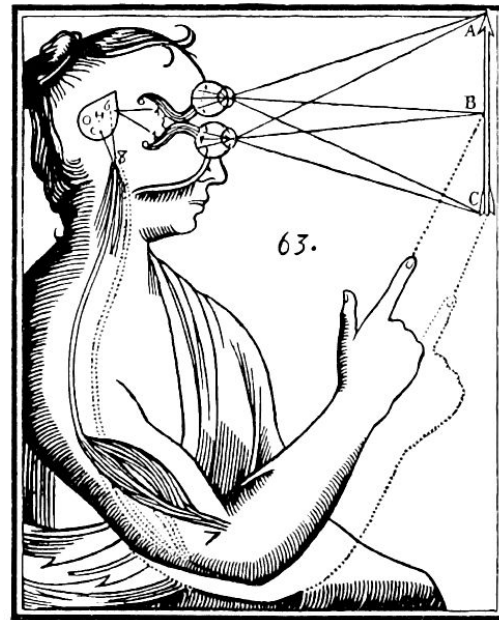
This is a massive over-simplification of ChatGPT, but this is a useful mental model to use going forward when we talk about large language models.

ChatGPT

This is why we emphasize that you **do not directly** use answers from ChatGPT in your TLABs/work.

By replacing your answers with LLM output, you open yourself up to:

- *Hallucinations*
- *Not citing work*
- *Lack of authority in your answers*



$$\hat{y} = W^T X$$

ChatGPT - Healthy Scepticism

We encourage you to have healthy scepticism of any new tool you come across.

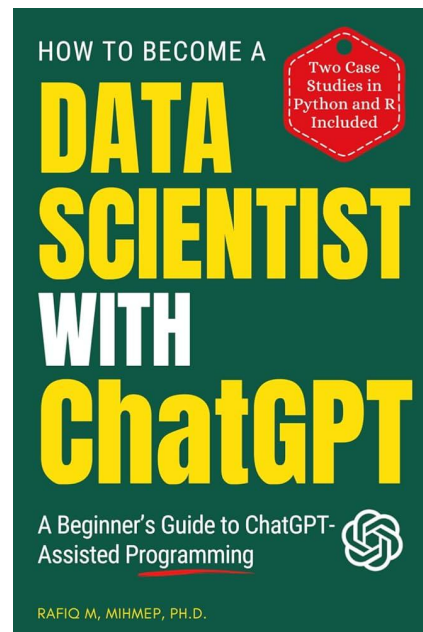
If a person/company/ceo/venture capitalist tells you that

- you **need** a product
- you will **get left behind** without it
- their product is **eternal**
- include an appeal to **higher ideas** (God, gov't, history)

They are either:

- *Clueless to what they're talking about*
- *Trying to sell you something*

Most of the time it's the **latter**.



Absolute nonsense, but
good cash-grab.

OpenAI API Syntax

I have a list of reviews from a wearable medical ring device that tracks heart rate. Please evaluate the sentiment of each line.

""

"this ring smells weird, don't recomend",

"I love this ring, I use it all the time when working out.",

"I will never buy another brand again, I love this ring",

"It's an ok ring. Some features could be better but for the price its fine.",

"its a ring",

"Bought this ring and it came broken. rip-off."

""



Here's a sentiment evaluation for each review, classified as **Positive**, **Neutral**, or **Negative**, with a brief explanation:

1. "this ring smells weird, don't recomend"



Sentiment: Negative

Ask anything



Search



Deep research



ChatGPT can make mistakes. Check important info.

Hopefully everyone is familiar with ChatGPT already. While we can open a chat-window and send requests to this LLM, **how can we actually send requests over the internet programmatically?**

If you haven't interacted with ChatGPT yet, we recommend you open the following URL and practice sending it queries: <https://chatgpt.com/>

OpenAI API



To do this, we need to utilize something *similar* to using the **requests** module.

Sometimes, specific tools have **prebuilt Python packages** which we can install and **utilize in our projects**.

In ChatGPT's case we can use the **openai** package.

Today's lab will entail the setup of this package in your **ds** environment. We recommend **that you simply watch us interact with this package as opposed to following along**.


```
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {
            "role": "user",
            "content": "Write a one-sentence bedtime story about a unicorn."
        }
    ]
)

print(completion.choices[0].message.content)
```

This block of code represents a simple “chat completion” request to OpenAI. Note that you will not be able to run this code until **you’ve set up your API key.**

```
from openai import OpenAI
client = OpenAI()
```

```
completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {
            "role": "user",
            "content": "Write a one-sentence bedtime story about a unicorn."
        }
    ]
)

print(completion.choices[0].message.content)
```

To begin sending requests to ChatGPT, we must first import the OpenAI class from the openai package and create the **OpenAI** object. Here we name our object “**client**.”

```
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {
            "role": "user",
            "content": "Write a one-sentence bedtime story about a unicorn."
        }
    ]
)

print(completion.choices[0].message.content)
```

Next we can begin our request that ChatGPT generates a response to a list of messages using the “**completions**” API endpoint. Here we simply need to define a series of objects **client.chat.completions.create**

```
from openai import OpenAI
client = OpenAI()
```

```
completion = client.chat.completions.create(
```

```
    model="gpt-4o-mini",
    messages=[
        {
            "role": "user",
            "content": "Write a one-sentence bedtime story about a unicorn."
        }
    ]
)
```

```
print(completion.choices[0].message.content)
```

Inside of the **create** API call, we specify a series of **parameters**. Remember parameters are simply variables that only exist within a specific class or function. For the full list of parameters, we can check the docs:

<https://platform.openai.com/docs/api-reference/chat/create>

```
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {
            "role": "user",
            "content": "Write a one-sentence bedtime story about a unicorn."
        }
    ]
)

print(completion.choices[0].message.content)
```

The first parameter we define is **model**. This is the **large language model** that we will be interfacing with. For our purposes, this will always be **gpt-4o-mini** due to API limitations.

Text tokens

Price per 1M tokens · Batch API price ☐

Model	Input	Cached input	Output
gpt-4.5-preview ↳ gpt-4.5-preview-2025-02-27	\$75.00	\$37.50	\$150.00
gpt-4o ↳ gpt-4o-2024-08-06	\$2.50	\$1.25	\$10.00
gpt-4o-audio-preview ↳ gpt-4o-audio-preview-2024-12-17	\$2.50	-	\$10.00
gpt-4o-realtime-preview ↳ gpt-4o-realtime-preview-2024-12-17	\$5.00	\$2.50	\$20.00
gpt-4o-mini ↳ gpt-4o-mini-2024-07-18	\$0.15	\$0.075	\$0.60
gpt-4o-mini-audio-preview ↳ gpt-4o-mini-audio-preview-2024-12-17	\$0.15	-	\$0.60
gpt-4o-mini-realtime-preview ↳ gpt-4o-mini-realtime-preview-2024-12-17	\$0.60	\$0.30	\$2.40
o1 ↳ o1-2024-12-17	\$15.00	\$7.50	\$60.00
o1-pro ↳ o1-pro-2025-03-19	\$150.00	-	\$600.00
o3-mini ↳ o3-mini-2025-01-31	\$1.10	\$0.55	\$4.40

However there are other models that you could possibly input. These are restricted for our API, so we will not be able to use them.

```
from openai import OpenAI
client = OpenAI()
```

```
completion = client.chat.completions.create(
    model="gpt-4o-mini"
```

```
    messages=[
        {
            "role": "user",
            "content": "Write a one-sentence bedtime story about a unicorn."
        }
    ]
)
```

```
print(completion.choices[0].message.content)
```

The next parameter is **messages**. This specifies the messages that we are sending to the model. Take a look at the outermost brackets of the argument of this parameter. What kind of data-structure do we pass?

```
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {
            "role": "user",
            "content": "Write a one-sentence bedtime story about a unicorn."
        }
    ]
)
```

```
print(completion.choices[0].message.content)
```

Later on, we will also explore the “developer” message to give the system specific instructions.

Notice that we are passing in a list! This will be a list composed of dictionaries. Note that so far we only specify the **user** message. This the prompt that ChatGPT receives in order to generate its output.


```
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {
            "role": "user",
            "content": "Write a one-sentence bedtime story about a unicorn."
        }
    ]
)

print(completion.choices[0].message.content)
```

Finally we access the response generated from this prompt using our **completion** variable. Note that we access its choices using index notation, what does this tell you about the output ChatGPT generates?

```
from openai import OpenAI
client = OpenAI()
```

```
completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {
            "role": "user",
            "content": "Write a one-sentence bedtime story about a unicorn."
        }
    ]
)
```

```
print(completion.choices[0].message.content)
```

```
Response
1 {
2   "id": "chatcmpl-B9MBs8CjcvOU2jLn4n570S5qMJKcT",
3   "object": "chat.completion",
4   "created": 1741569952,
5   "model": "gpt-4o-2024-08-06",
6   "choices": [
7     {
8       "index": 0,
9       "message": {
10        "role": "assistant",
11        "content": "Hello! How can I assist you today?",
12        "refusal": null,
13        "annotations": []

```

Just like we've encountered before, our API request results in a JSON object which we can access using Python syntax. Since "choices" is an array of different choices, we use our index notation to get the first object. We then subsequently get the message, and then content of the response.

```
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {
            "role": "user",
            "content": "Write a one-sentence bedtime story about a unicorn."
        }
    ]
)

print(completion.choices[0].message.content)
```

Note that often we will have to transform our string **content** into another data-structure which we can later use in other parts of our program. This is especially true for our second tab.

We will use this for TLAB 2 in order to tell our model what kind of responses it should craft.



OpenAI API

So far we've gone over simple requests to ChatGPT.

However, often times we also want to give directions that will be shown to the model, but not to the end-user themselves.

This is especially important if you would like to generate some sort of **Chat-bot** which will have unmoderated interaction with a person. For example, maybe you want to tell your LLM that it should always have a **cheery disposition**.

```
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        { "role": "developer", "content": "You are a children's short-story writer" },
        { "role": "user", "content": "Write a one-sentence bedtime story about a unicorn." }
    ]
)

print(completion.choices[0].message.content)
```

To provide these types of instructions to the model, you will have to include another message using the **developer** tag. Notice that this follows the same format as the original message, except this time we are setting up the context for the models response.

```
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        { "role": "developer", "content": "You are a children's short-story writer" },
        { "role": "user", "content": "Write a one-sentence bedtime story about a unicorn." }
    ]
)

print(completion.choices[0].message.content)
```

A question that you might have is, “What’s the point of this type of prompt?” Well to answer that, we must delve into something called **prompt engineering**.

“Prompt Engineering”



Prompt Engineering

One of the jobs that were predicted to come out of LLMs was **prompt engineering**, which is the supposed job of solely interacting with an LLM model to get some specific output.

This is most likely not to going to be a job anytime soon, but it does emphasize a certain kind of skill in LLM interactions.

It's important to note, we often **overestimate** the **effect of short-term changes** due to technological advances, but **underestimate** the long-term changes ([Amara's Law](#)).

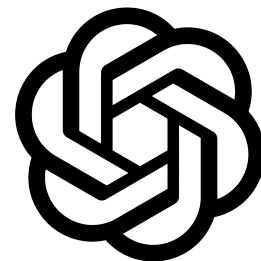


Prompt Engineering

With this disclaimer let's go over what "prompt engineering" entails.

When you are interfacing with ChatGPT, you send text queries/requests which it then "responds to." In reality it generates a probabilistic series of words **that it predicts is the most likely answer you would like to see**, based on historical data (**the internet**).

The level of quality **to these responses** depends on the **quality of your input**.



Prompt Engineering

We will attempt to minimize the anthropomorphization of this tool, but generally speaking, prompt engineering entails some tips which involve being an **effective communicator as a manager**. This includes:

- *Providing **helpful examples***
- *Be **specific** in your request*
- *Do plenty of **context setting, provide frameworks**.*

However there are other tips **which only apply to LLMs**. These include:

- ***“Gas up” your model:** “You are an expert in XYZ. You can do ABC.”*
- *Emphasize that you **need something ASAP**. Quality of responses tend to be better.*

I have a list of reviews from a wearable medical ring device that tracks heart rate. Please evaluate the sentiment of each line.

""

"this ring smells weird, don't recomend",
"I love this ring, I use it all the time when working out.",
"I will never buy another brand again, I love this ring",
"It's an ok ring. Some features could be better but for the price its fine.",
"its a ring",
"Bought this ring and it came broken. rip-off."

""

1. "this ring smells weird, don't recomend"
Sentiment: Negative
Reason: Negative smell description and explicitly says "don't recommend."
2. "I love this ring, I use it all the time when working out."
Sentiment: Positive
Reason: Strong positive emotion ("love"), and describes frequent usage.
3. "I will never buy another brand again, I love this ring"
Sentiment: Positive
Reason: Expresses loyalty and strong satisfaction.
4. "It's an ok ring. Some features could be better but for the price its fine."
Sentiment: Neutral
Reason: Balanced statement—acknowledges flaws but finds the value acceptable.

I have a list of reviews from a wearable medical ring device that tracks heart rate. Please evaluate the sentiment of each line.

""

"this ring smells weird, don't recomend",
"I love this ring, I use it all the time when working out.",
"I will never buy another brand again, I love this ring",
"It's an ok ring. Some features could be better but for the price its fine.",
"its a ring",
"Bought this ring and it came broken. rip-off."

""

An example output list

""

positive
positive
negative
neutral
irrelevant

""



arduino

"this ring smells weird, don't recomend",
"I love this ring, I use it all the time when working out.",
"I will never buy another brand again, I love this ring",
"It's an ok ring. Some features could be better but for the price its fine.",
"its a ring",
"Bought this ring and it came broken. rip-off."

Sentiment analysis output:

nginx

negative
positive
positive
neutral
irrelevant
negative

Let me know if you'd like these categorized in a CSV or used in a visualization!

Let's say I'm asking ChatGPT to transform a list of text according to some changes. By providing helpful examples, I can get output closer to what I expect.

I have a list of reviews from a wearable medical ring device that tracks heart rate. Please evaluate the sentiment of each line.

"""

"this ring smells weird, don't recomend",
"I love this ring, I use it all the time when working out.",
"I will never buy another brand again, I love this ring",
"It's an ok ring. Some features could be better but for the price its fine.",
"its a ring",
"Bought this ring and it came broken. rip-off."

"""

An example output list of reviews

"""

positive
positive
negative
neutral
irrelevant

"""



arduino

Copy

```
"this ring smells weird, don't recomend",  
"I love this ring, I use it all the time when working out.",  
"I will never buy another brand again, I love this ring",  
"It's an ok ring. Some features could be better but for the price its fine.",  
"its a ring",  
"Bought this ring and it came broken. rip-off."
```

Sentiment analysis output:

nginx

Copy

```
negative  
positive  
positive  
neutral  
irrelevant  
negative
```

Let me know if you'd like these categorized in a CSV or used in a visualization!

This will be used in my Python project, so I need this output to be inside of a list which I will eventually plot via a matplotlib visualization.



Here's the sentiment evaluation for each review line, formatted as a Python list for project:

python

```
sentiments = [  
    "negative",  
    "positive",  
    "positive",  
    "neutral",  
    "irrelevant",  
    "negative"  
]
```

Let me know if you want help generating a bar chart with `matplotlib` using this!

By context setting and specifying which language I'm working on/what my project should do, I can further refine my outputs.

This will be used in my Python project, so I need this output to be inside of a list which I will eventually plot via a matplotlib visualization.



Here's the sentiment evaluation for each review line, formatted as a Python list for project:

python

```
sentiments = [  
    "negative",  
    "positive",  
    "positive",  
    "neutral",  
    "irrelevant",  
    "negative"  
]
```

Let me know if you want help generating a bar chart with `matplotlib` using this!

This will be used in my Python project, so I need this output to be inside of a list which I will eventually plot via a matplotlib visualization. Please only output the list with no additional text.

python

Copy Edit

```
["negative", "positive", "positive", "neutral", "irrelevant", "negative"]
```

Finally, by being **specific in my request**, I can get my final answer **which only contains a Python list**. This will be useful for when I am interacting with the **gpt-4o-mini** API programmatically.



Prompt Engineering - Odd Tricks

As we've explained, you can also include additional context to further refine your answers.

“Gas up” your model:

- Tell the model **how good it is** at some specific task.
- Ex: *“You are an expert software engineer with 20 years of experience in the Python language. You are aware of test-driven development...”*

You need something ASAP.

- I cannot find the literature supporting this yet, but there's an odd tip that if you tell ChatGPT that you need something immediately, it performs “better”



Prompt Engineering - Odd Tricks

There are a few other odd tricks with ChatGPT that seem like they might improve performance:

- <https://news.ycombinator.com/item?id=39495476>
 - Pretending to **tip ChatGPT** might cause it to write longer text
- https://www.reddit.com/r/ChatGPT/comments/18gg6sr/tell_gpt_its_may_and_i_tll_perform_better/
 - If you tell ChatGPT it's **May**, it might perform better.

Take these with a grain of salt, none of this is established research.

OpenAI API Exercise



OpenAI API Exercise

For this exercise you will take set up your API token and begin practicing interacting with the OpenAI API.

However before you get started there are a couple of things to note:

- NEVER write your API key in publicly accessible code
- Be responsible with API usage



Keeping APIs Safe

For this project, we will send you all API keys. By tonight, **we will delete these keys from the Slack channel.** Please be present.

You will set up your API keys in your conda environment (ds). **Follow the listed directions carefully.**

If you complete tonights lab, you will get ahead on TLAB #2.

38TB of data accidentally exposed by Microsoft AI researchers

Wiz Research found a data exposure incident on Microsoft's AI GitHub repository, including over 30,000 internal Microsoft Teams messages – all caused by one misconfigured SAS token



Hillai Ben-Sasson, Ronny Greenberg

September 18, 2023

10 minutes read



NOTE: You should always keep your key a secret. Never push your API key to GitHub. Always use the environment variable instead.

Therefore, you should make API requests responsibly:

- A few test runs are ok
- 100 test runs are not



Responsible use of APIs

Because OpenAI API is a **paid** service, there incurs a charge every time you use the API.

To prevent costs from exploding we've set up our project to limit the number of requests that can occur per minute (100) and per day (1500).

If you encounter a **429 error** while running your project you must either:

- **Wait 1 minute for the requests per minute (RPM) to reset**
- **Wait till the next day for the requests per day (RPD) to reset** (*this will almost never happen unless you collude to run as many requests as possible*)

OpenAI Setup

To work on this project you must first configure your `ds` conda environment by following the listed steps below. By the end of these steps, you will have a hidden API key. Please follow these instructions closely or your project will not function correctly.

1. Activate your conda environment via the following terminal command:

```
conda activate ds
```

2. Confirm that you see the `(ds)` prompt at the beginning of your environment.
3. Once you've confirmed that you are in your environment, set your given API key to a conda environment variable using the terminal command:

```
conda env config vars set OPENAI_API_KEY="your_api_key"
```

4. Deactivate and reactive your environment by running the following commands in your terminal

```
conda deactivate ds
```

```
conda activate ds
```

5. Next, install the `openai` package by running the following pip command:

```
pip install openai
```

6. Activate your kernel and run the code-block below to verify that you've successfully installed the `openai` package.

```
[ ] from openai import OpenAI  
print("Success!")
```

Using this idea, complete the following API exercise as a group. Join a group to receive an API key.

Wrap-Up

Lab (Due 04/21)



Vancouver, Canada

You are a growth analyst at a Vancouver-based consulting firm called Monica Group. Your manager is spearheading the completion of a new analytical tool which will automatically label if a review is positive, neutral, negative, or irrelevant.

You will be kicking off completion of this milestone by independently implementing a minimal-viable-product. **This will be a Python pipeline that ingests a text-file of review data and interfaces with the Open AI API in order to automatically label each review.**

We will release API keys on 4/1



Wednesday

Wednesday will entail:

- A review of probability
- Applied probability
- Work on the OpenAI Lab

If you understand what you're doing, you're not learning anything. - Anonymous



Jupyter: scratchpad of the data scientist