Terminal UV

```
cash@macbookpro ~ % uv
An extremely fast Python package manager.

Usage: uv [OPTIONS] <COMMAND>

Commands:
  run      Run a command or script
  init     Create a new project
  add      Add dependencies to the project
  remove   Remove dependencies from the project
  version  Read or update the project's version
  sync     Update the project's environment
  lock     Update the project's lockfile
  export   Export the project's lockfile to an alternate format
  tree     Display the project's dependency tree
  tool     Run and install commands provided by Python packages
  python   Manage Python versions and installations
  pip      Manage Python packages with a pip-compatible interface
  venv     Create a virtual environment
  build    Build Python packages into source distributions and wheels
  publish  Upload distributions to an index
  cache    Manage uv's cache
  self     Manage the uv executable
  help     Display documentation for a command

Cache options:
  -n, --no-cache               Avoid reading from or writing to the cache,
                               instead using a temporary directory for the
                               duration of the operation [env: UV_NO_CACHE=]
      --cache-dir <CACHE_DIR>  Path to the cache directory [env: UV_CACHE_DIR=]

Python options:
      --managed-python       Require use of uv-managed Python versions [env:
                             UV_MANAGED_PYTHON=]
```

```
    --no-managed-python    Disable use of uv-managed Python versions [env:
                           UV_NO_MANAGED_PYTHON=]
    --no-python-downloads  Disable automatic downloads of Python. [env:
                           "UV_PYTHON_DOWNLOADS=never"]

Global options:
  -q, --quiet...
          Use quiet output
  -v, --verbose...
          Use verbose output
      --color <COLOR_CHOICE>
          Control the use of color in output [possible values: auto, always,
          never]
      --native-tls
          Whether to load TLS certificates from the platform's native
          certificate store [env: UV_NATIVE_TLS=]
      --offline
          Disable network access [env: UV_OFFLINE=]
      --allow-insecure-host <ALLOW_INSECURE_HOST>
          Allow insecure connections to a host [env: UV_INSECURE_HOST=]
      --no-progress
          Hide all progress outputs [env: UV_NO_PROGRESS=]
      --directory <DIRECTORY>
          Change to the given directory prior to running the command
      --project <PROJECT>
          Run the command within the given project directory [env: UV_PROJECT=]
      --config-file <CONFIG_FILE>
          The path to a `uv.toml` file to use for configuration [env:
          UV_CONFIG_FILE=]
      --no-config
          Avoid discovering configuration files (`pyproject.toml`, `uv.toml`)
          [env: UV_NO_CONFIG=]
  -h, --help
          Display the concise help for this command
  -V, --version
          Display the uv version

Use `uv help` for more details.
```

This Week To Do List

Build an automation tool using Python
reviw module 2
copl,.ete deeplearning.ai course
practice python 'if' 'for' loop, 'booleans'

AI Python for Beginners - DeepLearning.AI

AI Python for Beginners

Basics

Function

```
f"..."
```

```
test_variable = "xxx"
```

Automating Tasks with Python

Add one name to friends_list using append

```
friends_list = ["Tommy", "Isabel", "Daniel", "Otto"]
friends_list.append("Johnny")
print(friends_list)
```

In the following code, remove the country that is not in South America

```
countries_in_south_america = ["Colombia", "Peru", "Brasil", "Japan", "Argentina"]
countries_in_south_america.remove("Japan")
print(countries_in_south_america)
```

FOR LOOP

#ice cream flavor example **ice_cream_flavors = ["Vanilla", "Chocolate", "Strawberry", "Mint Chocolate Chip"]**

#You can use a for loop to iterate through the flavors and create a captivating description for each of them.

for flavor in ice_cream_flavors: prompt = f"""For the ice cream flavor listed below, provide a captivating description that could be used for promotional purposes.

`Flavor: {flavor} """ print_llm_response(prompt)`

#Now that you know how to use lists, you can even save the promortional descriptions to another list using `.append()` :

#saving results to a list **promotional_descriptions = [] for flavor in ice_cream_flavors: prompt = f"""For the ice cream flavor listed below, provide a captivating description that could be used for promotional purposes.**

`Flavor: {flavor} """ description = get_llm_response(prompt) promotional_descriptions.append(description)` #Write code to get a list with words without typos

words_with_typos = ["Aple", "Wether", "Newpaper"]
words_without_typos = []
for word in words_with_typos: prompt = f"""Fix the spelling mistake in the following word: {word}   Provide only the word.   """
correct_word = get_llm_response(prompt)
words_without_typos.append(correct_word)
print(words_without_typos)

Defintions

```
ice_cream_flavors = { "Mint Chocolate Chip": "Refreshing mint ice cream studded with decadent chocolate chips.", "Cookie Dough": "Vanilla ice cream loaded with chunks of chocolate chip cookie dough.", "Salted Caramel": "Sweet and salty with a smooth caramel swirl and a hint of sea salt." }
```

Adding to Definitions

```
ice_cream_flavors["Rocky Road"] = "Chocolate ice cream mixed with other ingredients."
```

Multi Defintions

```
isabel_facts = {
   "age": 28,
   "Favorite color": "red"
}
print(isabel_facts)
```

Adding to Multi Definitions

```
isabel_facts["Cat names"] = ["Charlie", "Smokey", "Tabitha"]
```

You can use dictionaries to store all the tasks with their priorities in a single data object.

```
#create dictionary with all tasks #dictionaries can contain lists!
prioritized_tasks = { "high_priority": high_priority_tasks, "medium_priority": medium_priority_tasks, "low_priority": low_priority_tasks }
```

Building Defintions to use for LLM requests:

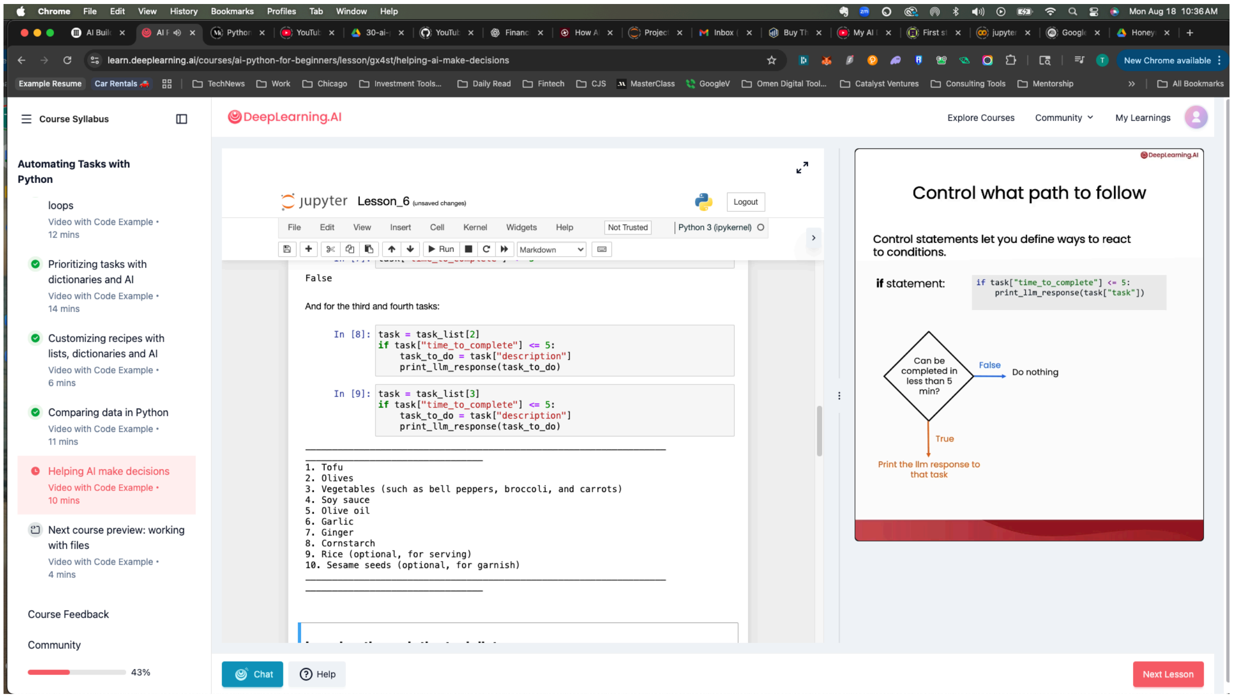
```
my_food_preferences = { "dietary_restrictions": ["pork"], #List with dietary restrictions "favorite_ingredients": ["bison"], #List with top
```

three favorite ingredients "experience_level": "Expert", #Experience level "maximum_spice_level": 0 #Spice level in a scale from 1 to 10 } print(my_food_preferences)

prompt = f"""Please suggest a recipe that tries to include the following ingredients: {food_preferences_tommy["favorite_ingredients"]}. The recipe should adhere to the following dietary restrictions: {food_preferences_tommy["dietary_restrictions"]}. The difficulty of the recipe should be: {food_preferences_tommy["experience_level"]} The maximum spice level on a scale of 10 should be: {food_preferences_tommy["maximum_spice_level"]} Provide a two sentence description. Provide detailed instructions of how to make the recipe """

print_llm_response(prompt)

Booleans + If statements



Course Syllabus

Automating Tasks with Python

- loops
  Video with Code Example • 12 mins
- Prioritizing tasks with dictionaries and AI
  Video with Code Example • 14 mins
- Customizing recipes with lists, dictionaries and AI
  Video with Code Example • 6 mins
- Comparing data in Python
  Video with Code Example • 11 mins
- Helping AI make decisions
  Video with Code Example • 10 mins
- Next course preview: working with files
  Video with Code Example • 4 mins

Course Feedback

Community

43%

```
False
```

And for the third and fourth tasks:

```
In [8]: task = task_list[2]
        if task["time_to_complete"] <= 5:
            task_to_do = task["description"]
            print_llm_response(task_to_do)

In [9]: task = task_list[3]
        if task["time_to_complete"] <= 5:
            task_to_do = task["description"]
            print_llm_response(task_to_do)
```

```
____________________________________________________________________
_________________________________
1. Tofu
2. Olives
3. Vegetables (such as bell peppers, broccoli, and carrots)
4. Soy sauce
5. Olive oil
6. Garlic
7. Ginger
8. Cornstarch
9. Rice (optional, for serving)
10. Sesame seeds (optional, for garnish)
____________________________________________________________________
_________________________________
```

Control what path to follow

Control statements let you define ways to react to conditions.

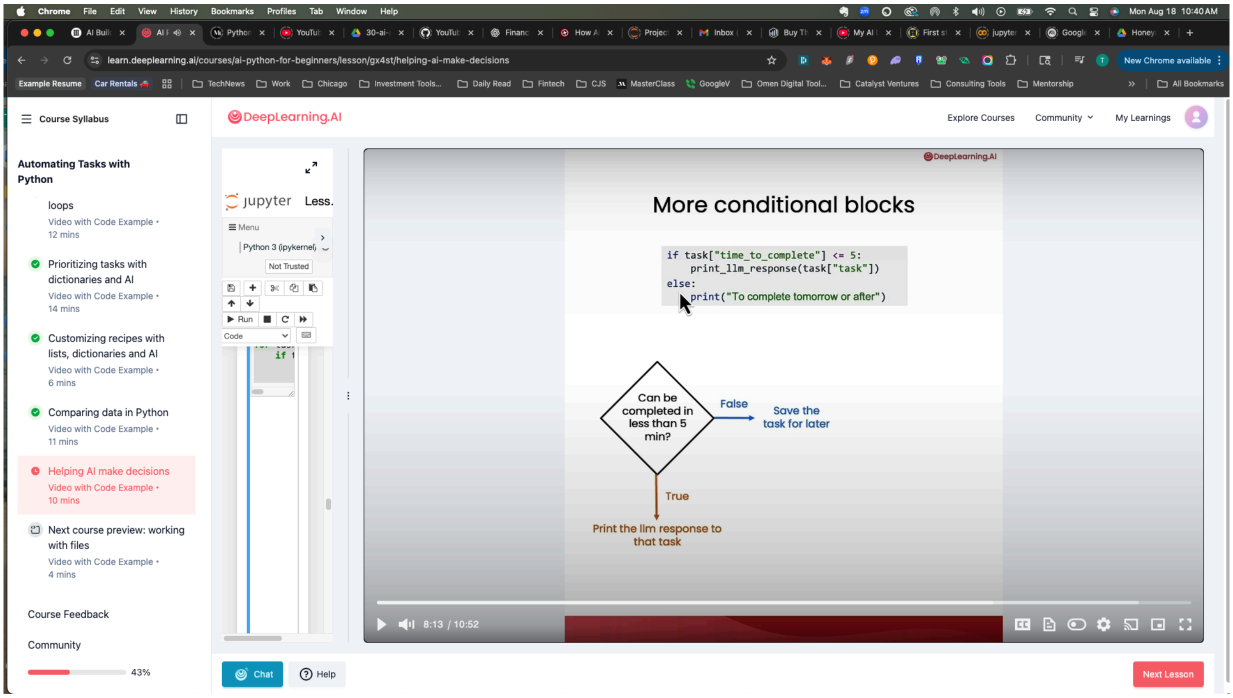
if statement:

```
if task["time_to_complete"] <= 5:
    print_llm_response(task["task"])
```

Can be completed in less than 5 min?
False
Do nothing
True
Print the llm response to that task

Looping wit Booleans + If statement

for task in task_list: if task["time_to_complete"] <= 5: task_to_do = task["description"] print_llm_response(task_to_do)



Automating Tasks with Python

- loops
  Video with Code Example • 12 mins
- Prioritizing tasks with dictionaries and AI
  Video with Code Example • 14 mins
- Customizing recipes with lists, dictionaries and AI
  Video with Code Example • 6 mins
- Comparing data in Python
  Video with Code Example • 11 mins
- Helping AI make decisions
  Video with Code Example • 10 mins
- Next course preview: working with files
  Video with Code Example • 4 mins

Course Feedback

Community

43%

More conditional blocks

```
if task["time_to_complete"] <= 5:
    print_llm_response(task["task"])
else:
    print("To complete tomorrow or after")
```

Can be completed in less than 5 min?
False
Save the task for later
True
Print the llm response to that task

for task in task_list: if task["time_to_complete"] <= 5: task_to_do = task["description"] print_llm_response(task_to_do) else: print(f"To complete later: {task['time_to_complete']} time to complete.")

Add variables to the f-string to provide the task description as well as the time to complete for the tasks that are left for later.

for task in task_list: if task["time_to_complete"] <= 5: task_to_do = task["description"] print_llm_response(task_to_do) else: ### EDIT THE FOLLOWING CODE ### # Hint: To add a variable in an f-string # you need to use the following syntax: {variable_name}. print(f"To complete later: {task['description']} will take {task['time_to_complete']}.") ### --------------- ###

Definitions

def fahrenheit_to_celsius(fahrenheit): # Calculation for getting the temperature in celsius celsius = (fahrenheit - 32) * 5 / 9 # Print the results print(f"{fahrenheit}°F is equivalent to {celsius:.2f}°C")

from helper_functions import get_llm_response
def create_bullet_points(file): # Complete code below to read in the file and store the contents as a string f = open(file, "r") file_contents = f.read() f.close() # YOUR CODE HERE

```
 # Write a prompt and pass to an LLM prompt = f"""Summarize the file into
three bullet points file: {file_contents} """ bullets =
get_llm_response(prompt) # Don't forget to add your prompt! # Return the
bullet points return bullets
```

This line of code runs your function for istanbul.txt and returns the output

output_bullets = create_bullet_points("istanbul.txt")

Print the fucntion output

print(output_bullets)

#Write the prompt for CSV

prompt = f"""Please extract a comprehensive list of the restaurants and their respective specialties mentioned in the following journal entry. Ensure that each restaurant name is accurately identified and listed. Provide your answer in CSV format, ready to save. Exclude the "```csv" declaration, don't add spaces after the comma, include column headers.
Format: Restaurant, Specialty Res_1, Sp_1 ...
Journal entry: {journal} """

#Print the prompt

print_llm_response(prompt)

Create an empty dictionary to store the itinerary for each destination

detailed_itinerary = {}

Use the 'for' loop over the 'itinerary' list

for trip_stop in itinerary: city = trip_stop["City"] country = trip_stop["Country"] arrival = trip_stop["Arrival"] departure = trip_stop["Departure"]

```
 rest_dict = read_csv(f"{city}.csv") print(f"Creating detailed itinerary for
{city}, {country}.") prompt = f"""I will visit {city}, {country} from
{arrival} to {departure}. Create a daily itinerary with detailed
activities. Designate times for breakfast, lunch, and dinner. I want to
visit the restaurants listed in the restaurant dictionary without repeating
```

```
any place. Make sure to mention the specialty that I should try at each of
them. Restaurant dictionary: {rest_dict} """ # Store the detailed itinerary
for the city to the dictionary detailed_itinerary[city] =
get_llm_response(prompt)
```

1. **Set Clear Goals**: Define specific, achievable learning objectives to stay focused and motivated.
2. **Practice Active Learning**: Engage with the material through summarization, questioning, and teaching others to reinforce understanding.
3. **Reflect and Adapt**: Regularly assess your progress and adjust your strategies based on what works best for you.

Load Data with Pandas:
1 data = pd.read_csv('car_data.csv') This line uses the pandas library to read a CSV file named 1 car_data.csv and load its contents into a DataFrame called 1 data . A DataFrame is like a table where you have rows and columns, making it easy to manipulate and analyze data. Print DataFrame:
1 print(data) This line prints the entire DataFrame to the console so you can see the contents, which include car models, their prices, years, and distances driven. Filter Cars Priced Greater Than or Equal to 10000:
To filter the cars with a price greater than or equal to 10000, you can use the following code: 1 2 expensive_cars = data[data['Price'] >= 10000] print(expensive_cars) Here's what it does: 1 data['Price'] >= 10000 : This creates a boolean Series where each row is checked if its 'Price' is greater than or equal to 10000. 1 data[...] : This filters the DataFrame to show only the rows where the condition is 1 True . 1 print(expensive_cars) : This prints the filtered DataFrame to display only those cars meeting the price condition.

create a filter for data

#WRITE YOUR CODE HERE

```
data = pd.read_csv('car_data.csv')
filtered_data = data[data["Model"].str.contains("Honda Accord", na=False)]
print(filtered_data)
```

Plot data in file in pie chart by year

```
import pandas as pd import matplotlib.pyplot as plt
data = pd.read_csv('car_data.csv')
#Count vehicles by year vehicles_per_year = data['Year'].value_counts()
#Plot as a pie chart plt.pie(vehicles_per_year, labels=vehicles_per_year.index,
autopct='%1.1f%%', startangle=90) plt.title('Vehicles Sold Each year') plt.axis('equal')
plt.show()
```

Accessing 'get_llm_response' and 'print_llm_response

```
import os
from dotenv import load_dotenv
from openai import OpenAI
```

Platform Product Management

Three Types of Product Management

1. Consumer
2. Enterprise - workflow completion
3. Platform - serving multiple users and multiple use cases across multiple product teams

Platform

- What are the fundamental rails that won't change
- Will it serve multiple use cases over time

Risks

- Maintenance tax
- increase surface area - security risks and 3rd party outages impacting functionality
- costly upfront cost

Benefits

- **Velocity** - Increase efficiency across teams to move fast
- **Leverage** - reuse technology in multiple contexts

- **Innovation** - combinatorial innovation that combines multiple assets together to create new products. Creating net new products that have never been created before with the building blocks

Measures of Success

- Adoption of platform
- Velocity - survey users to capture time saving and efficiency gain
- Revenue/Engagement - connect to the bottom line impact through causal information study

Principles

- Trust - consistency over time
  - Must be reliable
  - Must be predictable
  - Must be secure

Best practices

Documentation

- Change logs

Roadshow

- Share roadmap and value to all stakeholders

Prioritization

1. Vision - What is your vision of the team
2. Mission
3. Strategy

What will the platform look like 5-7 years from now?

- What are the milestones for the 5-7 year roadmap?
- What is Minimal Viable Platform?
  - Technical risk
  - Adoption risk
  - Regulatory risk
- Milestones should focus on de-risking

Premortem - It's 2 years from now and product was a complete disaster. Identify the risks with different stakeholders

- we have this goal and we have these risks. how to do build towards the goal and de-risking with milestones

Check-ins/Status updates - can also help with risk management by providing opportunities for stakeholders to surface potential risks

Additional Notes:

APIs - inputs needed for information flow and outputs

- Polling APIs - example would be when uber eats connects to google to get updates on the driver location
- Technical documentation for API - self service

Governance for Platforms

- What are the naming conventions
- What programs should be open, closed, or vetted?
- Which teams should be responsible for the capabilities?
- what are our standards?
- What decisions should be centralized?
- What decision should be for teams to make individually?
- Are here security implications with this decisions?

Centralized Governance

- Central team or technology responsible
- Consistent user experience

Decentralized

- Community of users and developers
- Possibility of fragmentation

Platform Roles

1. **API PM** - Drives the vision, strategy, prioritization, and roadmap
2. **API Designer** - Designs API interfaces that determines inputs and output
3. **Technical Writer** - Writes the technical documentation that helps developers user the APIs

4. **Business Dev and API Evangelist** - Builds partnerships in the ecosystem. Drives product adoption. Creates mutual value for joint customers
5. **Developer Relations** - Ensures third-party developers are supported with demos, tutorials, and materials

Platform Company - Stripe($640B)

1. Comprehensive features that are easy to use and highly customizable
2. Commitment to simplifying payment processing
3. Advanced features such as fraud detection and subscription management

Platform Company - Salesforce(150k customers)

1. Salesforce AppExchange offers a library of third applications and integrations
2. These integrations enable businesses to extend the functionality of their CRM

Platform Company - Amazon(AWS)

1. Provides a wide range of cloud-based services
2. Offers growing suite of products and services that can be customized

ChatGPT Prompt Engineering for Developers

https://learn.deeplearning.ai/courses/chatgpt-prompt-eng/lesson/zi9lz/guidelines

Prompt Engineering is an iterative practice. You have to rewrite and improve to ensure quality results.

Tactics:
Include the following:

Fact Sheets
Technical specification
Text Limits
Extract vs Summarize
Format output e.g. HTML, JSON, CSV, etc
Product Review

Context

Temperature - this signals to the LLM to increase variety of outputs e.g. display lower probability answers in result

Add OPENAI Library code:

```
import openai import os
from dotenv import load_dotenv, find_dotenv _ = load_dotenv(find_dotenv()) # read local .env file
openai.api_key  = os.getenv('OPENAI_API_KEY')
```

Add get_completion definition:

```
def get_completion(prompt, model="gpt-3.5-turbo",temperature=0): # Andrew mentioned that the prompt/ completion paradigm is preferable for this class
    messages = [{"role": "user", "content": prompt}]
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=temperature, # this is the degree of randomness of the model's output
    )
    return response.choices[0].message["content"]
```

AI Builders

Potential Pitfalls:

1. Figure out how to correctly push my code to github
2. Figure out how to correctly set up my local env
3. Figure out how to correctly push my code to test env
4. Figure out how to correctly push my code to production env
5. Correctly set up and match test envs and production envs
6. Setting up the API keys correctly or point the test env key to test env or prod key to prod env correctly
7. Setting up the API integrations correctly in code i.e. best code practices for APIs and other abstractions
8. Knowing how to set up a a script to run automatically without prompting

9. Knowing how to properly and safely host my applications
10. Setting up the database correctly
11. Setting up the devops and infrastructure efficiently
12. LLM Tokenomics

Day 1 Session (8/15/25):
Introduction to Coding with AI
Course Instructor: Shaw Talebi(PhD AI Researcher)
TA: Bryce Klien
kleinbryce@gmail.com
Office Hours: Monday(
Recommendations - Nick Gallo(ask Shaw for intro)

Live Session Notes:
Use

ask the code include explanations in the code that can be included in a text book.

Example 1: Scraping AI Job Board with Python

1. create folders
2. create requirements.txt and add the libraries that we will use then 'uv add -r requirements.text

Session (8/22):
LLM Prompt Engineering

Michael Trang(Guest Lecturer)

Score Metrics:
Performance Metrics -
F1 - Class balance performance metric which is relatively robust to imbalanced classes

$$\text{F1 Score} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

TP = True positive. FP = False Positive, FN = False Negative

Session

Week 2 Reflection:

In the past two weeks I've learned the following:

AI Python for Beginnger
AI Prompt Engineering
Setting up virtual environment
Setting up jupyter lab notebook
Setting up several libraries for code project including: python, openai, youtube, uv, flask and other libraries.

I've also coded and deployed two projects into my github:

Stock Market Dashboard
Github - https://github.com/toyeade/Stock-Market-Dashboard

Youtube Summarizer powered by AI(OpenAI)
Gitbhub - https://github.com/toyeade/youtube-video-summarizer

Challenges:

1. I had issues understanding how to enable the virtual environment
2. I had issues using cursor chat as it provided the wrong method for the Youtubetranscript API initially

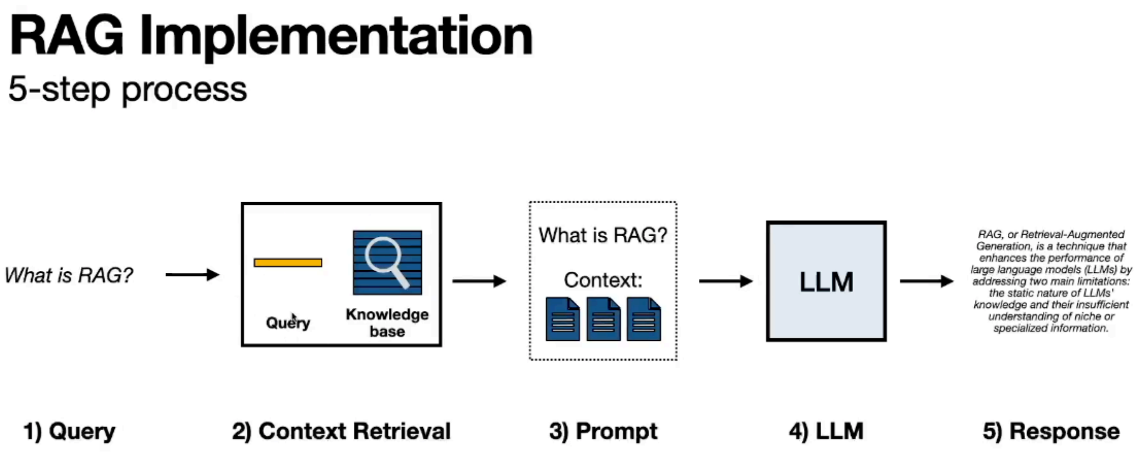
3. I had issues with Jupyter Lab notebook as I was attempting to run but was in the wrong folder path and did not properly activate the virtual env and the necessary libraries using uv
4. I had issues launching the web app for the Youtube Summarizer because the port selected was in use so I had to switch the code to enable any available port
5. I also had issues with setting up the OpenAI API key but realized the issues was minor

Module 3: RAG, Embedding Models

Retrieval Augmented Generation(RAG)

What is RAG

1. Query
2. Context Retrieval
3. Prompt
4. LLM
5. Response



RAG Implementation

5-step process

RAG, or Retrieval-Augmented Generation, is a technique that enhances the performance of large language models (LLMs) by addressing two main limitations: the static nature of LLMs' knowledge and their insufficient understanding of niche or specialized information.

RAG Models:

1. Keyword-based Search

2. Vector Search(Semantic Search)

Text Embeddings - Translates words into numbers which represent the meaning of the texts
e.g. Text classifications, clustering, regression analysis

Use cases
Text Classification - spam detection
Clustering - ICP analysis
Regression - product sales forecasting

TA Session(8/27/25)

Adding '.cursorrules'

- file structure.
    - "make a file structure for everything in @... include all of the classes and methods with a basic description"
- Use markdown for cursor rules e.g.
    - **#Project Python Libraries**
        - Pydantic
        - Mirascope
        - Crawl4ai

Adding MCP servers steps:

1. goto context7
2. search for the best...e.g. "
3. add to cursorrules "any time i ask you about a specific library, always use the context7 mcp server to retrieve the latest documentation

Resrouces:

https://github.com/PatrickJS/awesome-cursorrules

https://context7.com/

Repo Trends - https://www.repotrends.com/repos

Session 3 8.29.25

RAG and Text Embeddings

What does a model speed indicate?

- dot

What other metrics should we be examining when selecting models?

- token cost e.g. 3 characters equals 1 token(depending on selected LLM)
- the domain and matter the model was developed from
- how well is it doing what you want it to do?
- compute/api cost
- model size leads to larger compute cost

why is the component equal to 2 - the columns

Python Libraries

sbert.net - models, data modeling, and sentence transformers
sklearn -
PCA - tool that enables text matching and plotting
KMeeans - enables grouping of text embeddings and uses euclidean distance
matplotlib- charting tool
numpy - matrix and operations
panda - tables and charts

It will cost more to get better results due to "compute power" used i.e. tokens

Semantic Search

Semantic Search + RAG with LLamaIndex(Overview)

Important: RAG is dependent on how you chunk your text!

Raken AI - Rod Morrison

Tools:
Google ADK
Github Copilot
Python

Strategies:

- for token economics it is best to consider opportunities where large calls are being made e.g. pulling a schema in the openai llm call or storing the schema and refreshing daily
- for complex projects it is best practice to store 'prompts' in their own file
- Leveraging vectors is key for economically building on large schemas
- Intelligence scales with compute