Ended at Sun, 26 Jan, 2025, 11:59 pm IST Score: 0 Check all Save

# TDS 2025 Jan GA1 - Development Tools

## Instructions

1. **Learn what you need**. Reading material is provided, but feel free to skip it if you can answer the question. (Or learn it, just for pleasure.)
2. **Check answers regularly** by pressing Check. It shows which answers are right or wrong. You can check multiple times.
3. **Save regularly** by pressing Save. You can save multiple times. Your last saved submission will be evaluated.
4. **Reloading is OK**. Your answers are saved in your browser (not server). Questions won't change except for randomized parameters.
5. **Browser may struggle**. If you face loading issues, turn off security restrictions or try a different browser.
6. **Use anything**. You can use any resources you want. The Internet, ChatGPT, friends, whatever. Use any libraries or frameworks you want.
7. **It's hackable**. It's possible to get the answer to some questions by hacking the code for this quiz. That's allowed.

Should you take TDS this term?

* If this assignment takes you under 2 hours to complete, you will likely do OK in TDS.
* If you score above 8 / 10, you might get an S or A grade, with effort and luck.

**Have questions?** [**Join the discussion on Discourse**](https://discourse.onlinedegree.iitm.ac.in/t/ga1-development-tools-discussion-thread-tds-jan-2025/161083)

You are logged in as **22f1001679@ds.study.iitm.ac.in**.

Logout

#### **Recent saves (most recent is your official score)**

Reloadfrom 1/20/2025, 10:31:03 PM. Score: 10

Reloadfrom 1/20/2025, 8:18:54 PM. Score: 10

Reloadfrom 1/20/2025, 8:18:54 PM. Score: 10

Top of Form

# Questions

1. [VS Code Version](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-vs-code-version) (0.25 marks)
2. [Make HTTP requests with uv](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-uv-http-get) (1 mark)
3. [Run command with npx](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-npx-prettier) (0.5 marks)
4. [Use Google Sheets](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-use-google-sheets) (0.25 marks)
5. [Use Excel](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-use-excel) (0.25 marks)
6. [Use DevTools](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-use-devtools) (0.5 marks)
7. [Count Wednesdays](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-count-wednesdays) (0.5 marks)
8. [Extract CSV from a ZIP](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-extract-csv-zip) (0.25 marks)
9. [Use JSON](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-use-json) (0.75 marks)
10. [Multi-cursor edits to convert to JSON](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-multi-cursor-json) (0.5 marks)
11. [CSS selectors](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-css-selectors) (0.5 marks)
12. [Process files with different encodings](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-unicode-data) (1 mark)
13. [Use GitHub](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-use-github) (0.5 marks)
14. [Replace across files](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-replace-across-files) (0.75 marks)
15. [List files and attributes](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-list-files-attributes) (0.75 marks)
16. [Move and rename files](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-move-rename-files) (0.5 marks)
17. [Compare files](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-compare-files) (0.5 marks)
18. [SQL: Ticket Sales](https://exam.sanand.workers.dev/tds-2025-01-ga1#hq-sql-ticket-sales) (0.75 marks)

1 VS Code Version (0.25 marks)

## Editor: VS Code

Your editor is the most important tool in your arsenal. That's where you'll spend most of your time. Make sure you're comfortable with it.

[**Visual Studio Code**](https://code.visualstudio.com/) is, by far, the most popular code editor today. According to the [2024 StackOverflow Survey](https://survey.stackoverflow.co/2024/technology/#1-integrated-development-environment) almost 75% of developers use it. We recommend you learn it well. Even if you use another editor, you'll be working with others who use it, and it's a good idea to have some exposure.

Watch these introductory videos (35 min) from the [Visual Studio Docs](https://code.visualstudio.com/docs) to get started:

* [Getting Started](https://code.visualstudio.com/docs/introvideos/basics): Set up and learn the basics of Visual Studio Code. (7 min)
* [Code Editing](https://code.visualstudio.com/docs/introvideos/codeediting): Learn how to edit and run code in VS Code. (3 min)
* [Productivity Tips](https://code.visualstudio.com/docs/introvideos/productivity): Become a VS Code power user with these productivity tips. (4 min)
* [Personalize](https://code.visualstudio.com/docs/introvideos/configure): Personalize VS Code to make it yours with themes. (2 min)
* [Extensions](https://code.visualstudio.com/docs/introvideos/extend): Add features, themes, and more to VS Code with extensions! (4 min)
* [Debugging](https://code.visualstudio.com/docs/introvideos/debugging): Get started with debugging in VS Code. (6 min)
* [Version Control](https://code.visualstudio.com/docs/introvideos/versioncontrol): Learn how to use Git version control in VS Code. (3 min)
* [Customize](https://code.visualstudio.com/docs/introvideos/customize): Learn how to customize your settings and keyboard shortcuts in VS Code. (6 min)

## AI Editors: Copilot, Cursor

Note: AI Editors like [Cursor](https://www.cursor.com/), [Cody](https://sourcegraph.com/cody), and [GitHub Copilot](https://github.com/features/copilot) use LLMs to help you write code faster.

These are built on top of VS Code. These are now a standard tool in every developer's toolkit. Please use them.

Install and run Visual Studio Code. In your Terminal (or Command Prompt), type code -s and press Enter. Copy and paste the entire output below.

What is the output of code -s?

Check

2 Make HTTP requests with uv (1 mark)

## Python tools: uv

[Install uv](https://docs.astral.sh/uv/getting-started/installation/).

[uv](https://docs.astral.sh/uv/) is a fast Python package and project manager that's becoming the standard for running Python scripts. It replaces tools like pip, conda, pipx, poetry, pyenv, twine, and virtualenv into one, enabling:

* **Python Version Management**: uv installs and manages multiple Python versions, allowing developers to specify and switch between versions seamlessly.
* **Virtual Environment Handling**: It automates the creation and management of virtual environments, ensuring isolated and consistent development spaces for different projects.
* **Dependency Management**: With support for the pyproject.toml format, uv enables precise specification of project dependencies. It maintains a universal lockfile, uv.lock, to ensure reproducible installations across different systems.
* **Project Execution**: The uv run command allows for the execution of scripts and applications within the managed environment, streamlining development workflows.

Here are some commonly used commands:

# Replace python with uv. This automatically installs Python and dependencies.

uv run script.py

# Run a Python script directly from the Internet

uv run https://example.com/script.py

# Run a Python script without installing

uvx ruff

# Use a specific Python version

uv run --python 3.11 script.py

# Add dependencies to your script

uv add httpx --script script.py

# Create a virtual environment at .venv

uv venv

# Install packages to your virtual environment

uv pip install httpx

uv uses [inline script metadata](https://packaging.python.org/en/latest/specifications/inline-script-metadata/#inline-script-metadata) for dependencies. The eliminates the need for requirements.txt or virtual environments. For example:

# /// script

# requires-python = ">=3.11"

# dependencies = [

# "httpx",

# "pandas",

# ]

# ///

Running uv run --with httpie -- https [URL] installs the Python package httpie and sends a HTTPS request to the URL.

Send a HTTPS request to https://httpbin.org/get with the URL encoded parameter email set to 22f1001679@ds.study.iitm.ac.in

What is the JSON output of the command? (Paste only the JSON body, not the headers)

Check

3 Run command with npx (0.5 marks)

## JavaScript tools: npx

[npx](https://docs.npmjs.com/cli/v8/commands/npx) is a command-line tool that comes with npm (Node Package Manager) and allows you to execute npm package binaries and run one-off commands without installing them globally. It's essential for modern JavaScript development and data science workflows.

For data scientists, npx is useful when:

* Running JavaScript-based data visualization tools
* Converting notebooks and documents
* Testing and formatting code
* Running development servers

Here are common npx commands:

# Run a package without installing

npx http-server . # Start a local web server

npx prettier --write . # Format code or docs

npx eslint . # Lint JavaScript

npx typescript-node script.ts # Run TypeScript directly

npx esbuild app.js # Bundle JavaScript

npx jsdoc . # Generate JavaScript docs

# Run specific versions

npx prettier@3.2 --write . # Use prettier 3.2

# Execute remote scripts (use with caution!)

npx github:user/repo # Run from GitHub

Watch this introduction to npx (6 min):

Let's make sure you know how to use npx and prettier.

Download README.md. In the directory where you downloaded it, make sure it is called README.md, and run npx -y prettier@3.4.2 README.md | sha256sum.

What is the output of the command?

Check

4 Use Google Sheets (0.25 marks)

## Spreadsheet: Excel, Google Sheets

You'll use spreadsheets for data cleaning and exploration. The most popular spreadsheet program is [Microsoft Excel](https://www.microsoft.com/en-us/microsoft-365/excel) followed by [Google Sheets](https://www.google.com/sheets/about/).

You may be already familiar with these. If not, make sure to learn the basics of both.

Go through the [**Microsoft Excel** video training](https://support.microsoft.com/en-us/office/excel-video-training-9bc05390-e94c-46af-a5b3-d7c22f6990bb) and make sure you cover:

* [Intro to Excel](https://support.microsoft.com/en-us/office/create-a-new-workbook-ae99f19b-cecb-4aa0-92c8-7126d6212a83)
* [Rows & columns](https://support.microsoft.com/en-us/office/insert-or-delete-rows-and-columns-6f40e6e4-85af-45e0-b39d-65dd504a3246)
* [Cells](https://support.microsoft.com/en-us/office/move-or-copy-cells-and-cell-contents-803d65eb-6a3e-4534-8c6f-ff12d1c4139e)
* [Formatting](https://support.microsoft.com/en-us/office/available-number-formats-in-excel-0afe8f52-97db-41f1-b972-4b46e9f1e8d2)
* [Formulas & Functions](https://support.microsoft.com/en-us/office/overview-of-formulas-in-excel-ecfdc708-9162-49e8-b993-c311f47ca173)
* [Tables](https://support.microsoft.com/en-us/office/create-and-format-tables-e81aa349-b006-4f8a-9806-5af9df0ac664)
* [PivotTables](https://support.microsoft.com/en-us/office/create-a-pivottable-to-analyze-worksheet-data-a9a84538-bfe9-40a9-a8e9-f99134456576)

Watch this video for an introduction to **Google Sheets** (49 min):

Let's make sure you can write formulas in Google Sheets. Type this formula into Google Sheets. (It won't work in Excel)

=SUM(ARRAY\_CONSTRAIN(SEQUENCE(100, 100, 8, 6), 1, 10))

What is the result?

Check

5 Use Excel (0.25 marks)

Let's make sure you can write formulas in Excel. Type this formula into Excel.

Note: **This will ONLY work in Office 365.**

=SUM(TAKE(SORTBY({2,10,14,2,10,3,8,9,10,14,11,5,12,8,15,14}, {10,9,13,2,11,8,16,14,7,15,5,4,6,1,3,12}), 1, 14))

What is the result?

Note: If you get #NAME? you have the wrong version of Excel. Find a friend for whom this works.

Check

6 Use DevTools (0.5 marks)

## Browser: DevTools

[Chrome DevTools](https://developer.chrome.com/docs/devtools/overview/) is the de facto standard for web development and data analysis in the browser. You'll use this a lot when debugging and inspecting web pages.

Here are the key features you'll use most:

1. **Elements Panel**
   * Inspect and modify HTML/CSS in real-time
   * Copy CSS selectors for web scraping
   * Debug layout issues with the Box Model
2. // Copy selector in Console
3. **copy**($0); // Copies selector of selected element
4. **Console Panel**
   * JavaScript REPL environment
   * Log and debug data
   * Common console methods:
5. console.**table**(data); // Display data in table format
6. console.**group**("Name"); // Group related logs
7. console.**time**("Label"); // Measure execution time
8. **Network Panel**
   * Monitor API requests and responses
   * Simulate slow connections
   * Right-click on a request and select "Copy as fetch" to get the request.
9. **Essential Keyboard Shortcuts**
   * Ctrl+Shift+I (Windows) / Cmd+Opt+I (Mac): Open DevTools
   * Ctrl+Shift+C: Select element to inspect
   * Ctrl+L: Clear console
   * $0: Reference currently selected element
   * $$('selector'): Query selector all (returns array)

Videos from Chrome Developers (37 min total):

* [Fun & powerful: Intro to Chrome DevTools](https://youtu.be/t1c5tNPpXjs) (5 min)
* [Different ways to open Chrome DevTools](https://youtu.be/X65TAP8a530) (5 min)
* [Faster DevTools navigation with shortcuts and settings](https://youtu.be/xHusjrb_34A) (3 min)
* [How to log messages in the Console](https://youtu.be/76U0gtuV9AY) (6 min)
* [How to speed up your workflow with Console shortcuts](https://youtu.be/hdRDTj6ObiE) (6 min)
* [HTML vs DOM? Let’s debug them](https://youtu.be/J-02VNxE7lE) (5 min)
* [Caching demystified: Inspect, clear, and disable caches](https://youtu.be/mSMb-aH6sUw) (7 min)
* [Console message logging](https://youtu.be/76U0gtuV9AY) (6 min)
* [Console workflow shortcuts](https://youtu.be/hdRDTj6ObiE) (6 min)
* [HTML vs DOM debugging](https://youtu.be/J-02VNxE7lE) (5 min)
* [Cache inspection and management](https://youtu.be/mSMb-aH6sUw) (7 min)

Just above this paragraph, there's a hidden input with a secret value.

What is the value in the hidden input?

Check

7 Count Wednesdays (0.5 marks)

How many Wednesdays are there in the date range 1985-07-25 to 2009-12-02?

The dates are in the year-month-day format. Include both the start and end date in your count. You can do this using any tool (e.g. Excel, Python, JavaScript, manually).

Check

8 Extract CSV from a ZIP (0.25 marks)

Download and unzip file q-extract-csv-zip.zip which has a single extract.csv file inside.

What is the value in the "answer" column of the CSV file?

Check

9 Use JSON (0.75 marks)

## JSON

JSON (JavaScript Object Notation) is the de facto standard format for data exchange on the web and APIs. Its human-readable format and widespread support make it essential for data scientists working with web services, APIs, and configuration files.

For data scientists, JSON is essential when:

* Working with REST APIs and web services
* Storing configuration files and metadata
* Parsing semi-structured data from databases like MongoDB
* Creating data visualization specifications (e.g., Vega-Lite)

Watch this comprehensive introduction to JSON (15 min):

Key concepts to understand in JSON:

* JSON only supports 6 data types: strings, numbers, booleans, null, arrays, and objects
* You can nest data. Arrays and objects can contain other data types, including other arrays and objects
* Always validate. Ensure JSON is well-formed. Comm errors: Trailing commas, missing quotes, and escape characters

[JSON Lines](https://jsonlines.org/) is a format that allows you to store multiple JSON objects in a single line. It's useful for logging and streaming data.

Tools you could use with JSON:

* [JSONLint](https://jsonlint.com/): Validate and format JSON
* [JSON Editor Online](https://jsoneditoronline.org/): Visual JSON editor and formatter
* [JSON Schema](https://json-schema.org/): Define the structure of your JSON data
* [jq](https://stedolan.github.io/jq/): Command-line JSON processor

Common Python operations with JSON:

**import** json

# Parse JSON string

json\_str = '{"name": "Alice", "age": 30}'

data = json.loads(json\_str)

# Convert to JSON string

json\_str = json.dumps(data, indent=2)

# Read JSON from file

**with** open('data.json') **as** f:

data = json.load(f)

# Write JSON to file

**with** open('output.json', 'w') **as** f:

json.dump(data, f, indent=2)

# Read JSON data a Pandas DataFrame. JSON data is typically stored as an array of objects.

**import** pandas **as** pd

df = pd.read\_json('data.json')

# Read JSON lines from file into a DataFrame. JSON lines are typically one line per object.

df = pd.read\_json('data.jsonl', lines=True)

Practice JSON skills with these resources:

* [JSON Generator](https://json-generator.com/): Create sample JSON data
* [JSON Path Finder](https://jsonpathfinder.com/): Learn to navigate complex JSON structures
* [JSON Schema Validator](https://www.jsonschemavalidator.net/): Validate JSON against schemas

Let's make sure you know how to use JSON. Sort this JSON array of objects by the value of the age field. In case of a tie, sort by the name field. Paste the resulting JSON below without any spaces or newlines.

[{"name":"Alice","age":53},{"name":"Bob","age":5},{"name":"Charlie","age":56},{"name":"David","age":42},{"name":"Emma","age":71},{"name":"Frank","age":18},{"name":"Grace","age":56},{"name":"Henry","age":50},{"name":"Ivy","age":6},{"name":"Jack","age":22},{"name":"Karen","age":98},{"name":"Liam","age":93},{"name":"Mary","age":67},{"name":"Nora","age":45},{"name":"Oscar","age":41},{"name":"Paul","age":4}]

Sorted JSON:

Check

10 Multi-cursor edits to convert to JSON (0.5 marks)

Download q-multi-cursor-json.txt and use [multi-cursors](https://youtu.be/4lssq0zYxv0) and convert it into a single JSON object, where key=value pairs are converted into {key: value, key: value, ...}.

What's the result when you paste the JSON at [tools-in-data-science.pages.dev/jsonhash](https://tools-in-data-science.pages.dev/jsonhash) and click the Hash button?

Check

11 CSS selectors (0.5 marks)

## CSS Selectors

CSS selectors are patterns used to select and style HTML elements on a web page. They are fundamental to web development and data scraping, allowing you to precisely target elements for styling or extraction.

For data scientists, understanding CSS selectors is crucial when:

* Web scraping with tools like Beautiful Soup or Scrapy
* Selecting elements for browser automation with Selenium
* Styling data visualizations and web applications
* Debugging website issues using browser DevTools

Watch this comprehensive introduction to CSS selectors (20 min):

The Mozilla Developer Network (MDN) provides detailed documentation on the three main types of selectors:

* [Basic CSS selectors](https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Basic_selectors): Learn about element (div), class (.container), ID (#header), and universal (\*) selectors
* [Attribute selectors](https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Attribute_selectors): Target elements based on their attributes or attribute values ([type="text"])
* [Combinators](https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Combinators): Use relationships between elements (div > p, div + p, div ~ p)

Practice your CSS selector skills with this interactive tool:

* [CSS Diner](https://flukeout.github.io/): A fun game that teaches CSS selectors through increasingly challenging levels

Let's make sure you know how to select elements using CSS selectors. Find all <div>s having a foo class in the hidden element below. What's the sum of their data-value attributes?

Sum of data-value attributes:

Check

12 Process files with different encodings (1 mark)

## Unicode

Ever noticed when you copy-paste some text and get garbage symbols? Or see garbage when you load a CSV file? This video explains why. It covers how computers store text (called character encoding) and why it sometimes goes wonky.

Learn about ASCII (the original 7-bit encoding system that could only handle 128 characters), why that wasn't enough for global languages, and how modern solutions like Unicode save the day by letting us use any character from any language.

Some programs try to guess encodings (sometimes badly!). A signature called BOM (Byte Order Mark)helps computers know exactly how to read text files correctly.

Learn how Unicode, UTF-8 and character encoding works. This is a common gotcha when building apps that handle international text - something bootcamps often skip but developers and data scientists regularly face in the real world.

Unicode is fundamental for data scientists working with international data. Here are key concepts you need to understand:

* **Character Encodings**: Different ways to represent text in computers
  + ASCII (7-bit): Limited to 128 characters, English-only
  + UTF-8: Variable-width encoding, backwards compatible with ASCII
  + UTF-16: Fixed-width encoding, used in Windows and Java
  + UTF-32: Fixed-width encoding, memory inefficient but simple

Common encoding issues you'll encounter:

# Reading files with explicit encoding

**with** open('file.txt', encoding='utf-8') **as** f:

text = f.read()

# Handling encoding errors

**import** pandas **as** pd

df = pd.read\_csv('data.csv', encoding='utf-8', errors='replace')

# Detecting file encoding

**import** chardet

**with** open('unknown.txt', 'rb') **as** f:

result = chardet.detect(f.read())

print(result['encoding'])

Download and process the files in q-unicode-data.zip which contains three files with different encodings:

* data1.csv: CSV file encoded in CP-1252
* data2.csv: CSV file encoded in UTF-8
* data3.txt: Tab-separated file encoded in UTF-16

Each file has 2 columns: symbol and value. Sum up all the values where the symbol matches š OR … OR Ž across all three files.

What is the sum of all values associated with these symbols?

Check

13 Use GitHub (0.5 marks)

## Version Control: Git, GitHub

[Git](https://git-scm.com/) is the de facto standard for version control of software (and sometimes, data as well). It's a system that keeps track of changes you make to files and folders. It allows you to revert to a previous state, compare changes, etc. It's a central tool in any developer's workflow.

[GitHub](https://github.com/) is the most popular hosting service for Git repositories. It's a website that shows your code, allows you to collaborate with others, and provides many useful tools for developers.

Watch these introductory videos to learn the basics of Git and GitHub (98 min):

Essential Git Commands:

# Repository Setup

git init # Create new repo

git clone url # Clone existing repo

git remote add origin url # Connect to remote

# Basic Workflow

git status # Check status

git add . # Stage all changes

git commit -m "message" # Commit changes

git push origin main # Push to remote

# Branching

git branch # List branches

git checkout -b feature # Create/switch branch

git merge feature # Merge branch

git rebase main # Rebase on main

# History

git log --oneline # View history

git diff commit1 commit2 # Compare commits

git blame file # Show who changed what

Best Practices:

1. **Commit Messages**
2. # Good commit message format
3. type(scope): summary
4. Detailed description of changes.
5. # Examples
6. feat(api): add user authentication
7. fix(db): handle null values **in** query
8. **Branching Strategy**
   * main: Production code
   * develop: Integration branch
   * feature/\*: New features
   * hotfix/\*: Emergency fixes
9. **Code Review**
   * Keep PRs small (<400 lines)
   * Use draft PRs for WIP
   * Review your own code first
   * Respond to all comments

Essential Tools

* [GitHub Desktop](https://desktop.github.com/): GUI client
* [GitLens](https://gitlens.amod.io/): VS Code extension
* [gh](https://cli.github.com/): GitHub CLI
* [pre-commit](https://pre-commit.com/): Git hooks

Let's make sure you know how to use GitHub. [Create a GitHub account](https://github.com/join) if you don't have one. Create a new public repository. Commit a single JSON file called email.json with the value {"email": "22f1001679@ds.study.iitm.ac.in"} and push it.

Enter the raw Github URL of email.json so we can verify it. (It might look like https://raw.githubusercontent.com/[GITHUB ID]/[REPO NAME]/main/email.json.)

Check

14 Replace across files (0.75 marks)

## Terminal: Bash

UNIX shells are the de facto standard in the data science world and [Bash](https://www.gnu.org/software/bash/) is the most popular. This is available by default on Mac and Linux.

On Windows, install [Git Bash](https://git-scm.com/downloads) or [WSL](https://learn.microsoft.com/en-us/windows/wsl/install) to get a UNIX shell.

Watch this video to understand the basics of Bash and UNIX shell commands (75 min).

Essential Commands:

# File Operations

ls -la # List all files with details

cd path/to/dir # Change directory

pwd # Print working directory

cp source dest # Copy files

mv source dest # Move/rename files

rm -rf dir # Remove directory recursively

# Text Processing

grep "pattern" file # Search for pattern

sed 's/old/new/' f # Replace text

awk '{print $1}' f # Process text by columns

cat file | wc -l # Count lines

# Process Management

ps aux # List processes

kill -9 PID # Force kill process

top # Monitor processes

htop # Interactive process viewer

# Network

curl url # HTTP requests

wget url # Download files

nc -zv host port # Test connectivity

ssh user@host # Remote login

# Count unique values in CSV column

cut -d',' -f1 data.csv | sort | uniq -c

# Quick data analysis

awk -F',' '{sum+=$2} END {print sum/NR}' data.csv # Average

sort -t',' -k2 -n data.csv | head # Top 10

# Monitor log in real-time

tail -f log.txt | grep --color 'ERROR'

Bash Scripting Essentials:

#!/bin/bash

# Variables

NAME="value"

echo $NAME

# Loops

**for** i **in** {1..5}; **do**

echo $i

**done**

# Conditionals

**if** [ -f "file.txt" ]; **then**

echo "File exists"

**fi**

# Functions

**process\_data**() {

local input=$1

echo "Processing $input"

}

Productivity Tips:

1. **Command History**
2. history # Show command history
3. Ctrl+R # Search history
4. !! # Repeat last command
5. !$ # Last argument
6. **Directory Navigation**
7. pushd dir # Push directory to stack
8. popd # Pop directory from stack
9. cd - # Go to previous directory
10. **Job Control**
11. command & # Run in background
12. Ctrl+Z # Suspend process
13. bg # Resume in background
14. fg # Resume in foreground
15. **Useful Aliases** - typically added to ~/.bashrc
16. alias ll='ls -la'
17. alias gs='git status'
18. alias jupyter='jupyter notebook'
19. alias activate='source venv/bin/activate'

Download q-replace-across-files.zip and unzip it into a new folder, then replace all "IITM" (in upper, lower, or mixed case) with "IIT Madras" in **all files**. Leave everything as-is - don't change the [line endings](https://stackoverflow.com/a/39532890/100904).

What does running cat \* | sha256sum in that folder show in bash?

Check

15 List files and attributes (0.75 marks)

Download q-list-files-attributes.zip and extract it. Use ls with options to list all files in the folder along with their date and file size.

What's the total size of all files at least 7265 bytes large and modified on or after Thu, 5 Aug, 2004, 9:24 am IST?

Don't **copy** from inside the ZIP file or use Windows Explorer to unzip. That destroys the timestamps. Extract using unzip, 7-Zip or similar utilities and check the timestamps.

Check

16 Move and rename files (0.5 marks)

Download q-move-rename-files.zip and extract it. Use mv to move all files under folders into an empty folder. Then rename all files replacing each digit with the next. 1 becomes 2, 9 becomes 0, a1b9c.txt becomes a2b0c.txt.

What does running grep . \* | LC\_ALL=C sort | sha256sum in bash on that folder show?

Check

17 Compare files (0.5 marks)

Download q-compare-files.zip and extract it. It has 2 nearly identical files, a.txt and b.txt, with the same number of lines.

How many lines are different between a.txt and b.txt?

Check

18 SQL: Ticket Sales (0.75 marks)

## Database: SQLite

Relational databases are used to store data in a structured way. You'll often access databases created by others for analysis.

PostgreSQL, MySQL, MS SQL, Oracle, etc. are popular databases. But the most installed database is [SQLite](https://www.sqlite.org/index.html). It's embedded into many devices and apps (e.g. your phone, browser, etc.). It's lightweight but very scalable and powerful.

Watch these introductory videos to understand SQLite and how it's used in Python (34 min):

There are many non-relational databases (NoSQL) like [ElasticSearch](https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html), [MongoDB](https://www.mongodb.com/docs/manual/), [Redis](https://redis.io/docs/latest/), etc. that you should know about and we may cover later.

Core Concepts:

-- Create a table

**CREATE TABLE** users (

id INTEGER **PRIMARY KEY**,

name TEXT **NOT NULL**,

email TEXT **UNIQUE**,

created\_at DATETIME **DEFAULT** CURRENT\_TIMESTAMP

);

-- Insert data

**INSERT INTO** users (name, email) **VALUES**

('Alice', 'alice@example.com'),

('Bob', 'bob@example.com');

-- Query data

**SELECT** name, COUNT(\*) **as** count

**FROM** users

**GROUP** **BY** name

**HAVING** count > 1;

-- Join tables

**SELECT** u.name, o.product

**FROM** users u

**LEFT** **JOIN** orders o **ON** u.id = o.user\_id

**WHERE** o.status = 'pending';

Python Integration:

**import** sqlite3

**from** pathlib **import** Path

**import** pandas **as** pd

**async** **def** **query\_database**(db\_path: Path, query: str) -> pd.DataFrame:

"""Execute SQL query and return results as DataFrame.

Args:

db\_path: Path to SQLite database

query: SQL query to execute

Returns:

DataFrame with query results

"""

**try**:

conn = sqlite3.connect(db\_path)

**return** pd.read\_sql\_query(query, conn)

**finally**:

conn.close()

# Example usage

db = Path('data.db')

df = **await** query\_database(db, '''

SELECT date, COUNT(\*) as count

FROM events

GROUP BY date

''')

Common Operations:

1. **Database Management**
2. -- Backup database
3. .backup 'backup.db'
4. -- Import CSV
5. .mode csv
6. .import data.csv table\_name
7. -- Export results
8. .headers **on**
9. .mode csv
10. .output results.csv
11. **SELECT** \* **FROM** **table**;
12. **Performance Optimization**
13. -- Create index
14. **CREATE** INDEX idx\_user\_email **ON** users(email);
15. -- Analyze query
16. EXPLAIN QUERY PLAN
17. **SELECT** \* **FROM** users **WHERE** email **LIKE** '%@example.com';
18. -- Show indexes
19. **SELECT** \* **FROM** sqlite\_master **WHERE** type='index';
20. **Data Analysis**
21. -- Time series aggregation
22. **SELECT**
23. date(timestamp),
24. COUNT(\*) **as** events,
25. AVG(duration) **as** avg\_duration
26. **FROM** events
27. **GROUP** **BY** date(timestamp);
28. -- Window functions
29. **SELECT** \*,
30. AVG(amount) **OVER** (
31. **PARTITION** **BY** user\_id
32. **ORDER** **BY** date
33. **ROWS** **BETWEEN** 3 PRECEDING **AND** **CURRENT** ROW
34. ) **as** moving\_avg
35. **FROM** transactions;

Tools to work with SQLite:

* [SQLiteStudio](https://sqlitestudio.pl/): Lightweight GUI
* [DBeaver](https://dbeaver.io/): Full-featured GUI
* [sqlite-utils](https://sqlite-utils.datasette.io/): CLI tool
* [Datasette](https://datasette.io/): Web interface

There is a tickets table in a SQLite database that has columns type, units, and price. Each row is a customer bid for a concert ticket.

| **type** | **units** | **price** |
| --- | --- | --- |
| silver | 130 | 1.89 |
| BRONZE | 104 | 1.43 |
| Bronze | 65 | 1.63 |
| SILVER | 547 | 1.74 |
| SILVER | 881 | 0.85 |
| ... | | |

What is the total sales of all the items in the "Gold" ticket type? Write SQL to calculate it.

Get all rows where the Type is "Gold". Ignore spaces and treat mis-spellings like GOLD, gold, etc. as "Gold". Calculate the sales as Units \* Price, and sum them up.

Check

Check all Save

Save regularly. Your last saved submission will be evaluated.

Bottom of Form

# Best of luck!

Ended at Sun, 2 Feb, 2025, 11:59 pm IST Score: 0 Check all Save

# TDS 2025 Jan GA2 - Deployment Tools

## Instructions

1. **Learn what you need**. Reading material is provided, but feel free to skip it if you can answer the question. (Or learn it, just for pleasure.)
2. **Check answers regularly** by pressing Check. It shows which answers are right or wrong. You can check multiple times.
3. **Save regularly** by pressing Save. You can save multiple times. Your last saved submission will be evaluated.
4. **Reloading is OK**. Your answers are saved in your browser (not server). Questions won't change except for randomized parameters.
5. **Browser may struggle**. If you face loading issues, turn off security restrictions or try a different browser.
6. **Use anything**. You can use any resources you want. The Internet, ChatGPT, friends, whatever. Use any libraries or frameworks you want.
7. **It's hackable**. It's possible to get the answer to some questions by hacking the code for this quiz. That's allowed.

**Have questions?** [**Join the discussion on Discourse**](https://discourse.onlinedegree.iitm.ac.in/t/ga2-deployment-tools-discussion-thread-tds-jan-2025/161120)

You are logged in as **22f1001679@ds.study.iitm.ac.in**.

Logout

#### **Recent saves (most recent is your official score)**

Reloadfrom 2/1/2025, 5:20:18 PM. Score: 7.5

Reloadfrom 2/1/2025, 4:33:14 PM. Score: 6.5

Reloadfrom 1/29/2025, 7:57:52 PM. Score: 6.5

Top of Form

# Questions

1. [Write documentation in Markdown](https://exam.sanand.workers.dev/tds-2025-01-ga2#hq-markdown) (1 mark)
2. [Compress an image](https://exam.sanand.workers.dev/tds-2025-01-ga2#hq-image-compression) (1 mark)
3. [Host your portfolio on GitHub Pages](https://exam.sanand.workers.dev/tds-2025-01-ga2#hq-github-pages) (1 mark)
4. [Use Google Colab](https://exam.sanand.workers.dev/tds-2025-01-ga2#hq-use-colab) (0.5 marks)
5. [Use an Image Library in Google Colab](https://exam.sanand.workers.dev/tds-2025-01-ga2#hq-use-colab-image-library) (0.5 marks)
6. [Deploy a Python API to Vercel](https://exam.sanand.workers.dev/tds-2025-01-ga2#hq-vercel-python) (1 mark)
7. [Create a GitHub Action](https://exam.sanand.workers.dev/tds-2025-01-ga2#hq-github-action) (1.5 marks)
8. [Push an image to Docker Hub](https://exam.sanand.workers.dev/tds-2025-01-ga2#hq-docker-hub-image) (1 mark)
9. [Write a FastAPI server to serve data](https://exam.sanand.workers.dev/tds-2025-01-ga2#hq-fastapi) (1 mark)
10. [Run a local LLM with Llamafile](https://exam.sanand.workers.dev/tds-2025-01-ga2#hq-llamafile) (1.5 marks)

1 Write documentation in Markdown (1 mark)

## Documentation: Markdown

Markdown is a lightweight markup language for creating formatted text using a plain-text editor. It's the standard for documentation in software projects and data science notebooks.

Watch this introduction to Markdown (19 min):

Common Markdown syntax:

**# Heading 1**

**## Heading 2**

**\*\*bold\*\*** and *\*italic\**

- Bullet point

- Another point

- Nested point

1. Numbered list

2. Second item

[Link text](https://url.com)

![Image alt](image.jpg)

| Column 1 | Column 2 |

|----------|----------|

| Cell 1 | Cell 2 |

```python

# Code block

def hello():

print("Hello")

```

> Blockquote

Tools for working with Markdown:

* [markdown2](https://pypi.org/project/markdown2/): Python library to convert Markdown to HTML
* [markdownlint](https://github.com/DavidAnson/markdownlint): Linting
* [Markdown All in One](https://marketplace.visualstudio.com/items?itemName=yzhang.markdown-all-in-one): VS Code extension
* [pandoc](https://pandoc.org/): Convert between formats

Write documentation in Markdown for an \*\*imaginary\*\* analysis of the number of steps you walked each day for a week, comparing over time and with friends. The Markdown must include:

* **Top-Level Heading**: At least 1 heading at level 1, e.g., # Introduction
* **Subheadings**: At least 1 heading at level 2, e.g., ## Methodology
* **Bold Text**: At least 1 instance of bold text, e.g., \*\*important\*\*
* **Italic Text**: At least 1 instance of italic text, e.g., \*note\*
* **Inline Code**: At least 1 instance of inline code, e.g., sample\_code
* **Code Block**: At least 1 instance of a [fenced code block](https://docs.github.com/en/get-started/writing-on-github/working-with-advanced-formatting/creating-and-highlighting-code-blocks), e.g.
* print("Hello World")
* **Bulleted List**: At least 1 instance of a bulleted list, e.g., - Item
* **Numbered List**: At least 1 instance of a numbered list, e.g., 1. Step One
* **Table**: At least 1 instance of a table, e.g., | Column A | Column B |
* **Hyperlink**: At least 1 instance of a hyperlink, e.g., [Text](https://example.com)
* **Image**: At least 1 instance of an image, e.g., ![Alt Text](https://example.com/image.jpg)
* **Blockquote**: At least 1 instance of a blockquote, e.g., > This is a quote

Enter your Markdown here:

Check

2 Compress an image (1 mark)

## Images: Compression

Image compression is essential when deploying apps. Often, pages have dozens of images. Image analysis runs over thousands of images. The cost of storage and bandwidth can grow over time.

Here are things you should know when you're compressing images:

* **Image dimensions** are the width and height of the image in pixels. This impacts image size a lot
* **Lossless** compression (PNG, WebP) preserves exact data
* **Lossy** compression (JPEG, WebP) removes some data for smaller files
* **Vector** formats (SVG) scale without quality loss
* **WebP** is the modern standard, supporting both lossy and lossless

Here's a rule of thumb you can use as of 2025.

* Use SVG if you can (i.e. if it's vector graphics or you can convert it to one)
* Else, reduce the image to as small as you can, and save as lossy WebP

Common operations with Python:

**from** pathlib **import** Path

**from** PIL **import** Image

**import** io

**async** **def** **compress\_image**(input\_path: Path, output\_path: Path, quality: int = 85) -> None:

"""Compress an image while maintaining reasonable quality."""

**with** Image.open(input\_path) **as** img:

# Convert RGBA to RGB if needed

**if** img.mode == 'RGBA':

img = img.convert('RGB')

# Optimize for web

img.save(output\_path, 'WEBP', quality=quality, optimize=True)

# Batch process images

paths = Path('images').glob('\*.jpg')

**for** p **in** paths:

**await** compress\_image(p, p.with\_suffix('.webp'))

Command line tools include [cwebp](https://developers.google.com/speed/webp/docs/cwebp), [pngquant](https://pngquant.org/), [jpegoptim](https://github.com/tjko/jpegoptim), and [ImageMagick](https://imagemagick.org/).

# Convert to WebP

cwebp -q 85 input.png -o output.webp

# Optimize PNG

pngquant --quality=65-80 image.png

# Optimize JPEG

jpegoptim --strip-all --all-progressive --max=85 image.jpg

# Convert and resize

convert input.jpg -resize 800x600 output.jpg

# Batch convert

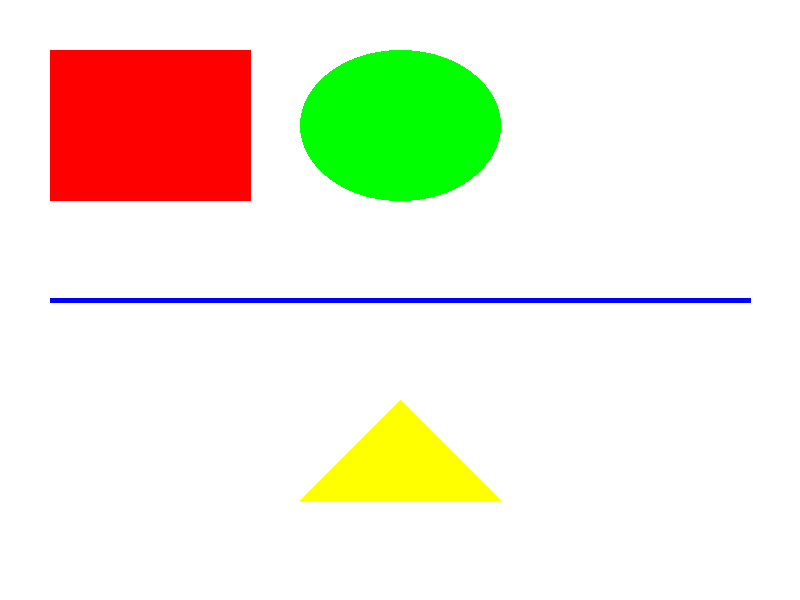
mogrify -format webp -quality 85 \*.jpg

Watch this video on modern image formats and optimization (15 min):

Tools for image optimization:

* [squoosh.app](https://squoosh.app/): Browser-based compression
* [ImageOptim](https://imageoptim.com/): GUI tool for Mac
* [sharp](https://sharp.pixelplumbing.com/): Node.js image processing
* [Pillow](https://python-pillow.org/): Python imaging library

Download the image below and compress it losslessly to an image that is less than 1,500 bytes.



By losslessly, we mean that every pixel in the new image should be identical to the original image.

Upload your losslessly compressed image (less than 1,500 bytes)

Check

3 Host your portfolio on GitHub Pages (1 mark)

## Static hosting: GitHub Pages

[GitHub Pages](https://pages.github.com/) is a free hosting service that turns your GitHub repository directly into a whenever you push it. This is useful for sharing analysis results, data science portfolios, project documentation, and more.

Common Operations:

# Create a new GitHub repo

mkdir my-site

cd my-site

git init

# Add your static content

echo "<h1>My Site</h1>" > index.html

# Push to GitHub

git add .

git commit -m "feat(pages): initial commit"

git push origin main

# Enable GitHub Pages from the main branch on the repo settings page

Best Practices:

1. **Keep it small**
   * [Optimize images](https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Performance/Multimedia). Prefer SVG over WEBP over 8-bit PNG.
   * [Preload](https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/rel/preload) critical assets like stylesheets
   * Avoid committing large files like datasets, videos, etc. directly. Explore [Git LFS](https://git-lfs.github.com/) instead.

Tools:

* [GitHub Desktop](https://desktop.github.com/): GUI for Git operations
* [GitHub CLI](https://cli.github.com/): Command line interface
* [GitHub Actions](https://github.com/features/actions): Automation

Publish a page using [GitHub Pages](https://pages.github.com/) that showcases your work. Ensure that your email address **22f1001679@ds.study.iitm.ac.in** is in the page's HTML.

GitHub pages are served via CloudFlare which [obfuscates emails](https://developers.cloudflare.com/waf/tools/scrape-shield/email-address-obfuscation/). So, wrap your email address inside a:

<!--email\_off-->22f1001679@ds.study.iitm.ac.in<!--/email\_off-->

What is the GitHub Pages URL? It might look like: https://[USER].github.io/[REPO]/

If a recent change that's not reflected, add ?v=1, ?v=2 to the URL to bust the cache.

Check

4 Use Google Colab (0.5 marks)

## Notebooks: Google Colab

[Google Colab](https://colab.research.google.com/) is a free, cloud-based Jupyter notebook environment that's become indispensable for data scientists and ML practitioners. It's particularly valuable because it provides free access to GPUs and TPUs, and for easy sharing of code and execution results.

While Colab is excellent for prototyping and learning, its free tier has limitations - notebooks time out after 12 hours, and GPU access can be inconsistent.

Learn how to mount Google Drive for persistent storage, manage dependencies with !pip install commands, as these are common pain points when getting started.

* [Google Colab features you may have missed](https://youtu.be/rNgswRZ2C1Y)
* [How to mount Google Drive to Google Colab](https://youtu.be/8HvugBq5NKg)
* [How to take advantage of GPUs and TPUs for your ML project](https://youtu.be/tCYSce6l8gA)

Let's make sure you can access Google Colab. Run this program on Google Colab, allowing all required access to your email ID: 22f1001679@ds.study.iitm.ac.in.

**import** hashlib

**import** requests

**from** google.colab **import** auth

**from** oauth2client.client **import** GoogleCredentials

auth.authenticate\_user()

creds = GoogleCredentials.get\_application\_default()

token = creds.get\_access\_token().access\_token

response = requests.get(

"https://www.googleapis.com/oauth2/v1/userinfo",

params={"alt": "json"},

headers={"Authorization": f"Bearer {token}"}

)

email = response.json()["email"]

hashlib.sha256(f"{email} {creds.token\_expiry.year}".encode()).hexdigest()[-5:]

What is the result? (It should be a 5-character string)

Check

5 Use an Image Library in Google Colab (0.5 marks)

[Download this image](https://exam.sanand.workers.dev/lenna.webp). Create a new [Google Colab notebook](https://colab.research.google.com/) and run this code (after fixing a mistake in it) to calculate the number of pixels with a certain minimum brightness:

**import** numpy **as** np

**from** PIL **import** Image

**from** google.colab **import** files

**import** colorsys

# There is a mistake in the line below. Fix it

image = Image.open(list(files.upload().keys)[0])

rgb = np.array(image) / 255.0

lightness = np.apply\_along\_axis(**lambda** x: colorsys.rgb\_to\_hls(\*x)[1], 2, rgb)

light\_pixels = np.sum(lightness > 0.055)

print(f'Number of pixels with lightness > 0.055: {light\_pixels}')

What is the result? (It should be a number)

Check

6 Deploy a Python API to Vercel (1 mark)

## Serverless hosting: Vercel

Serverless platforms let you rent a single function instead of an entire machine. They're perfect for small web tools that don't need to run all the time. Here are some common real-life uses:

* A contact form that emails you when someone wants to hire you (runs for 2-3 seconds, a few times per day)
* A tool that converts uploaded photos to black and white (runs for 5-10 seconds when someone uploads a photo)
* A chatbot that answers basic questions about your business hours (runs for 1-2 seconds per question)
* A newsletter sign-up that adds emails to your mailing list (runs for 1 second per sign-up)
* A webhook that posts your Etsy sales to Discord (runs for 1 second whenever you make a sale)

You only pay when someone uses your tool, and the platform automatically handles busy periods. For example, if 100 people fill out your contact form at once, the platform creates 100 temporary copies of your code to handle them all. When they're done, these copies disappear. It's cheaper than running a full-time server because you're not paying for the time when no one is using your tool - most tools are idle 95% of the time!

Rather than writing a full program, serverless platforms let you write functions. These functions are called via HTTP requests. They run in a cloud environment and are scaled up and down automatically. But this means you write programs in a different style. For example:

* You can't pip install packages - you have to use requirements.txt
* You can't read or write files from the file system - you can only use APIs.
* You can't run commands (e.g. subprocess.run())

[Vercel](https://vercel.com/) is a cloud platform optimized for frontend frameworks and serverless functions. Vercel is tightly integrated with GitHub. Pushing to your repository automatically triggers new deployments.

Here's a [quickstart](https://vercel.com/docs/functions/runtimes/python). [Sign-up with Vercel](https://vercel.com/signup). Create an empty git repo with this api/index.py file.

To deploy a FastAPI app, add a requirements.txt file with fastapi as a dependency.

fastapi

Add your FastAPI code to a file, e.g. main.py.

# main.py

**from** fastapi **import** FastAPI

app = FastAPI()

@app.get("/")

**def** **read\_root**():

**return** {"message": "Hello, World!"}

Add a vercel.json file to the root of your repository.

{

"builds": [{ "src": "main.py", "use": "@vercel/python" }],

"routes": [{ "src": "/(.\*)", "dest": "main.py" }]

}

On the command line, run:

* npx vercel to deploy a test version
* npx vercel --prod to deploy to production

**Environment Variables**. Use npx vercel env add to add environment variables. In your code, use os.environ.get('SECRET\_KEY') to access them.

### **Videos**

Download this q-vercel-python.json which has the marks of 100 imaginary students.

Create and deploy a Python app to [Vercel](https://vercel.com/). Expose an API so that when a request like https://your-app.vercel.app/api?name=X&name=Y is made, it returns a JSON response with the marks of the names X and Y in the same order, like this:

{ "marks": [10, 20] }

Make sure you enable **CORS** to allow GET requests from any origin.

What is the Vercel URL? It should look like: https://your-app.vercel.app/api

Check

7 Create a GitHub Action (1.5 marks)

## CI/CD: GitHub Actions

[GitHub Actions](https://github.com/features/actions) is a powerful automation platform built into GitHub. It helps automate your development workflow - running tests, deploying applications, updating datasets, retraining models, etc.

* Understand the basics of [YAML configuration files](https://docs.github.com/en/actions/writing-workflows/quickstart)
* Explore the [pre-built actions from the marketplace](https://github.com/marketplace?type=actions)
* How to [handle secrets securely](https://docs.github.com/en/actions/security-for-github-actions/security-guides/using-secrets-in-github-actions)
* [Triggering a workflow](https://docs.github.com/en/actions/writing-workflows/choosing-when-your-workflow-runs/triggering-a-workflow)
* Staying within the [free tier limits](https://docs.github.com/en/billing/managing-billing-for-your-products/managing-billing-for-github-actions/about-billing-for-github-actions)
* [Caching dependencies to speed up workflows](https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/caching-dependencies-to-speed-up-workflows)

Here is a sample .github/workflows/iss-location.yml that runs daily, appends the International Space Station location data into iss-location.json, and commits it to the repository.

name: Log ISS Location Data Daily

on:

schedule:

# Runs at 12:00 UTC (noon) every day

- cron: "0 12 \* \* \*"

workflow\_dispatch: # Allows manual triggering

jobs:

collect-iss-data:

runs-on: ubuntu-latest

permissions:

contents: write

steps:

- name: Checkout repository

uses: actions/checkout@v4

- name: Install uv

uses: astral-sh/setup-uv@v5

- name: Fetch ISS location data

run: | # python

uv run --with requests python << 'EOF'

import requests

data = requests.get('http://api.open-notify.org/iss-now.json').text

with open('iss-location.jsonl', 'a') as f:

f.write(data + '\n')

'EOF'

- name: Commit and push changes

run: | # shell

git config --local user.email "github-actions[bot]@users.noreply.github.com"

git config --local user.name "github-actions[bot]"

git add iss-location.jsonl

git commit -m "Update ISS position data [skip ci]" || exit 0

git push

Tools:

* [GitHub CLI](https://cli.github.com/): Manage workflows from terminal
* [Super-Linter](https://github.com/github/super-linter): Validate code style
* [Release Drafter](https://github.com/release-drafter/release-drafter): Automate releases
* [act](https://github.com/nektos/act): Run actions locally
* [How to handle secrets in GitHub Actions](https://youtu.be/1tD7km5jK70)

Create a [GitHub action](https://github.com/features/actions) on one of your GitHub repositories. Make sure one of the steps in the action has a name that contains your email address 22f1001679@ds.study.iitm.ac.in. For example:

jobs:

test:

steps:

- name: 22f1001679@ds.study.iitm.ac.in

run: echo "Hello, world!"

Trigger the action and make sure it is the **most recent action**.

What is your repository URL? It will look like: https://github.com/USER/REPO

Check

8 Push an image to Docker Hub (1 mark)

## Containers: Docker, Podman

[Docker](https://www.docker.com/) and [Podman](https://podman.io/) are containerization tools that package your application and its dependencies into a standardized unit for software development and deployment.

Docker is the industry standard. Podman is compatible with Docker and has better security (and a slightly more open license). In this course, we recommend Podman but Docker works in the same way.

Initialize the container engine:

podman machine init

podman machine start

Common Operations. (You can use docker instead of podman in the same way.)

# Pull an image

podman pull python:3.11-slim

# Run a container

podman run -it python:3.11-slim

# List containers

podman ps -a

# Stop container

podman stop container\_id

# Scan image for vulnerabilities

podman scan myapp:latest

# Remove container

podman rm container\_id

# Remove all stopped containers

podman container prune

You can create a Dockerfile to build a container image. Here's a sample Dockerfile that converts a Python script into a container image.

**FROM** python:3.11-slim

# Set working directory

**WORKDIR** /app

# Typically, you would use `COPY . .` to copy files from the host machine,

# but here we're just using a simple script.

**RUN** echo 'print("Hello, world!")' > app.py

# Run the script

**CMD** ["python", "app.py"]

To build, run, and deploy the container, run these commands:

# Create an account on https://hub.docker.com/ and then login

podman login docker.io

# Build and run the container

podman build -t py-hello .

podman run -it py-hello

# Push the container to Docker Hub. Replace $DOCKER\_HUB\_USERNAME with your Docker Hub username.

podman push py-hello:latest docker.io/$DOCKER\_HUB\_USERNAME/py-hello

# Push adding a specific tag, e.g. dev

TAG=dev podman push py-hello docker.io/$DOCKER\_HUB\_USERNAME/py-hello:$TAG

Tools:

* [Dive](https://github.com/wagoodman/dive): Explore image layers
* [Skopeo](https://github.com/containers/skopeo): Work with container images
* [Trivy](https://github.com/aquasecurity/trivy): Security scanner
* Optional: For Windows, see [WSL 2 with Docker getting started](https://youtu.be/5RQbdMn04Oc)

Create and push an image to [Docker Hub](https://hub.docker.com/). Add a tag named 22f1001679 to the image.

What is the Docker image URL? It should look like: https://hub.docker.com/repository/docker/$USER/$REPO/general

Check

9 Write a FastAPI server to serve data (1 mark)

## Web Framework: FastAPI

[FastAPI](https://fastapi.tiangolo.com/) is a modern Python web framework for building APIs with automatic interactive documentation. It's fast, easy to use, and designed for building production-ready REST APIs.

Here's a minimal FastAPI app, app.py:

# /// script

# requires-python = ">=3.11"

# dependencies = [

# "fastapi",

# "uvicorn",

# ]

# ///

**from** fastapi **import** FastAPI

app = FastAPI()

@app.get("/")

**async** **def** **root**():

**return** {"message": "Hello!"}

**if** \_\_name\_\_ == "\_\_main\_\_":

**import** uvicorn

uvicorn.run(app, host="0.0.0.0", port=8000)

Run this with uv run app.py.

1. **Handle errors by raising HTTPException**
2. **from** fastapi **import** HTTPException
3. **async** **def** **get\_item**(item\_id: int):
4. **if** **not** valid\_item(item\_id):
5. **raise** HTTPException(
6. status\_code=404,
7. detail=f"Item {item\_id} not found"
8. )
9. **Use middleware for logging**
10. **from** fastapi **import** Request
11. **import** time
12. @app.middleware("http")
13. **async** **def** **add\_timing**(request: Request, call\_next):
14. start = time.time()
15. response = **await** call\_next(request)
16. response.headers["X-Process-Time"] = str(time.time() - start)
17. **return** response

Tools:

* [FastAPI CLI](https://fastapi.tiangolo.com/tutorial/fastapi-cli/): Project scaffolding
* [Pydantic](https://pydantic-docs.helpmanual.io/): Data validation
* [SQLModel](https://sqlmodel.tiangolo.com/): SQL databases
* [FastAPI Users](https://fastapi-users.github.io/): Authentication

Watch this FastAPI Course for Beginners (64 min):

## REST APIs

REST (Representational State Transfer) APIs are the standard way to build web services that allow different systems to communicate over HTTP. They use standard HTTP methods and JSON for data exchange.

Watch this comprehensive introduction to REST APIs (52 min):

Key Concepts:

1. **HTTP Methods**
   * GET: Retrieve data
   * POST: Create new data
   * PUT/PATCH: Update existing data
   * DELETE: Remove data
2. **Status Codes**
   * 2xx: Success (200 OK, 201 Created)
   * 4xx: Client errors (400 Bad Request, 404 Not Found)
   * 5xx: Server errors (500 Internal Server Error)

Here's a minimal REST API using FastAPI. Run this server.py script via uv run server.py:

# /// script

# requires-python = ">=3.13"

# dependencies = [

# "fastapi",

# "uvicorn",

# ]

# ///

**from** fastapi **import** FastAPI, HTTPException

**from** typing **import** Dict, List

app = FastAPI()

# Create a list of items that will act like a database

items: List[Dict[str, float | int | str]] = []

# Create a GET endpoint that returns all items

@app.get("/items")

**async** **def** **get\_items**() -> List[Dict[str, float | int | str]]:

**return** items

# Create a GET endpoint that returns a specific item by ID

@app.get("/items/{item\_id}")

**async** **def** **get\_item**(item\_id: int) -> Dict[str, float | int | str]:

**if** item := next((i **for** i **in** items **if** i["id"] == item\_id), None):

**return** item

**raise** HTTPException(status\_code=404, detail="Item not found")

# Create a POST endpoint that creates a new item

@app.post("/items")

**async** **def** **create\_item**(item: Dict[str, float | str]) -> Dict[str, float | int | str]:

new\_item = {"id": len(items) + 1, "name": item["name"], "price": float(item["price"])}

items.append(new\_item)

**return** new\_item

**if** \_\_name\_\_ == "\_\_main\_\_":

**import** uvicorn

uvicorn.run(app, host="0.0.0.0", port=8000)

Test the API with curl:

# Get all items

curl http://localhost:8000/items

# Create an item

curl -X POST http://localhost:8000/items \

-H "Content-Type: application/json" \

-d '{"name": "Book", "price": 29.99}'

# Get specific item

curl http://localhost:8000/items/1

Best Practices:

1. **Use Nouns for Resources**
   * Good: /users, /posts
   * Bad: /getUsers, /createPost
2. **Version Your API**
3. /api/v1/users
4. /api/v2/users
5. **Handle Errors Consistently**
6. {
7. "error": "Not Found",
8. "message": "User 123 not found",
9. "status\_code": 404
10. }
11. **Use Query Parameters for Filtering**
12. **/api/**posts?status=published&category=tech
13. **Implement Pagination**
14. /api/posts?page=2&limit=10

Tools:

* [Postman](https://www.postman.com/): API testing and documentation
* [Swagger/OpenAPI](https://swagger.io/): API documentation
* [HTTPie](https://httpie.io/): Modern command-line HTTP client
* [JSON Schema](https://json-schema.org/): API request/response validation

Learn more about REST APIs:

* [REST API Design Best Practices](https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/)
* [Microsoft REST API Guidelines](https://github.com/microsoft/api-guidelines)
* [Google API Design Guide](https://cloud.google.com/apis/design)

## CORS: Cross-Origin Resource Sharing

CORS (Cross-Origin Resource Sharing) is a security mechanism that controls how web browsers handle requests between different origins (domains, protocols, or ports). Data scientists need CORS for APIs serving data or analysis to a browser on a different domain.

Watch this practical explanation of CORS (3 min):

Key CORS concepts:

* **Same-Origin Policy**: Browsers block requests between different origins by default
* **CORS Headers**: Server responses must include specific headers to allow cross-origin requests
* **Preflight Requests**: Browsers send OPTIONS requests to check if the actual request is allowed
* **Credentials**: Special handling required for requests with cookies or authentication

If you're exposing your API with a GET request publicly, the only thing you need to do is set the HTTP header Access-Control-Allow-Origin: \*.

Here are other common CORS headers:

**Access-Control-Allow-Origin**: https://example.com

**Access-Control-Allow-Methods**: GET, POST, PUT, DELETE

**Access-Control-Allow-Headers**: Content-Type, Authorization

**Access-Control-Allow-Credentials**: true

To implement CORS in FastAPI, use the [CORSMiddleware middleware](https://fastapi.tiangolo.com/tutorial/cors/):

**from** fastapi **import** FastAPI

**from** fastapi.middleware.cors **import** CORSMiddleware

app = FastAPI()

app.add\_middleware(CORSMiddleware, allow\_origins=["\*"]) # Allow GET requests from all origins

# Or, provide more granular control:

app.add\_middleware(

CORSMiddleware,

allow\_origins=["https://example.com"], # Allow a specific domain

allow\_credentials=True, # Allow cookies

allow\_methods=["GET", "POST", "PUT", "DELETE"], # Allow specific methods

allow\_headers=["\*"], # Allow all headers

)

Testing CORS with JavaScript:

// Simple request

**const** response = **await** **fetch**("https://api.example.com/data", {

method: "GET",

headers: { "Content-Type": "application/json" },

});

// Request with credentials

**const** response = **await** **fetch**("https://api.example.com/data", {

credentials: "include",

headers: { "Content-Type": "application/json" },

});

Useful CORS debugging tools:

* [CORS Checker](https://cors-test.codehappy.dev/): Test CORS configurations
* Browser DevTools Network tab: Inspect CORS headers and preflight requests
* [cors-anywhere](https://github.com/Rob--W/cors-anywhere): CORS proxy for development

Common CORS errors and solutions:

* No 'Access-Control-Allow-Origin' header: Configure server to send proper CORS headers
* Request header field not allowed: Add required headers to Access-Control-Allow-Headers
* Credentials flag: Set both credentials: 'include' and Access-Control-Allow-Credentials: true
* Wild card error: Cannot use \* with credentials; specify exact origins

Download q-fastapi.csv. This file has 2-columns:

1. studentId: A unique identifier for each student, e.g. 1, 2, 3, ...
2. class: The class (including section) of the student, e.g. 1A, 1B, ... 12A, 12B, ... 12Z

Write a FastAPI server that serves this data. For example, /api should return all students data (in the same row and column order as the CSV file) as a JSON like this:

{

"students": [

{

"studentId": 1,

"class": "1A"

},

{

"studentId": 2,

"class": "1B"

}, ...

]

}

If the URL has a query parameter class, it should return only students in those classes. For example, /api?class=1A should return only students in class 1A. /api?class=1A&class=1B should return only students in class 1A and 1B. There may be any number of classes specified. Return students in the same order as they appear in the CSV file (not the order of the classes).

Make sure you enable **CORS** to allow GET requests from any origin.

What is the API URL endpoint for FastAPI? It might look like: http://127.0.0.1:8000/api

We'll check by sending a request to this URL with ?class=... added and check if the response matches the data.

Check

10 Run a local LLM with Llamafile (1.5 marks)

## Local LLMs: Llamafile

You would have heard of Large Language Models (LLMs) like GPT-4, Claude, and Llama. Some of these models are available for free, but most of them are not.

An easy way to run LLMs locally is Mozilla's [Llamafile](https://github.com/Mozilla-Ocho/llamafile). It's a single executable file that works on Windows, Mac, and Linux. No installation or configuration needed - just download and run.

Watch this Llamafile Tutorial (6 min):

[](https://youtu.be/d1Fnfvat6nM)

Here's how to get started

1. [Download Llama-3.2-1B-Instruct.Q6\_K.llamafile (1.11 GB)](https://huggingface.co/Mozilla/Llama-3.2-1B-Instruct-llamafile/blob/main/Llama-3.2-1B-Instruct.Q6_K.llamafile?download=true).
2. From the command prompt or terminal, run Llama-3.2-1B-Instruct.Q6\_K.llamafile.
3. Optional: For GPU acceleration, use Llama-3.2-1B-Instruct.Q6\_K.llamafile --n-gpu-layers 35. (Increase or decrease the number of layers based on your GPU VRAM.)

You might see a message like this:

██╗ ██╗ █████╗ ███╗ ███╗ █████╗ ███████╗██╗██╗ ███████╗

██║ ██║ ██╔══██╗████╗ ████║██╔══██╗██╔════╝██║██║ ██╔════╝

██║ ██║ ███████║██╔████╔██║███████║█████╗ ██║██║ █████╗

██║ ██║ ██╔══██║██║╚██╔╝██║██╔══██║██╔══╝ ██║██║ ██╔══╝

███████╗███████╗██║ ██║██║ ╚═╝ ██║██║ ██║██║ ██║███████╗███████╗

╚══════╝╚══════╝╚═╝ ╚═╝╚═╝ ╚═╝╚═╝ ╚═╝╚═╝ ╚═╝╚══════╝╚══════╝

software: llamafile 0.8.17

model: Llama-3.2-1B-Instruct-Q8\_0.gguf

compute: 13th Gen Intel Core i9-13900HX (alderlake)

server: http://127.0.0.1:8080/

A chat between a curious human and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the human's questions.

You can now chat with the model. Type /exit or press Ctrl+C to stop.

You can also visit http://127.0.0.1:8080/ in your browser to chat with the model.

LlamaFile exposes an OpenAI compatible API. Here's how to use it in Python:

**import** requests

response = requests.post(

"http://localhost:8080/v1/chat/completions",

headers={"Content-Type": "application/json"},

json={"messages": [{"role": "user", "content": "Write a haiku about coding"}]}

)

print(response.json()["choices"][0]["message"]["content"])

Tools:

* [OpenAI API compatibility](https://platform.openai.com/docs/api-reference/chat): Use existing OpenAI code
* [Creating your own llamafiles](https://github.com/Mozilla-Ocho/llamafile#creating-llamafiles): Control output format

## Tunneling: ngrok

[Ngrok](https://ngrok.com/) is a tool that creates secure tunnels to your localhost, making your local development server accessible to the internet. It's essential for testing webhooks, sharing work in progress, or debugging applications in production-like environments.

Run the command uvx ngrok http 8000 to create a tunnel to your local server on port 8000. This generates a public URL that you can share with others.

To get started, log into ngrok.com and [get an authtoken from the dashboard](https://dashboard.ngrok.com/get-started/your-authtoken). Copy it. Then run:

ngrok config add-authtoken $YOUR\_AUTHTOKEN

Now you can forward any local port to the internet. For example:

# Start a local server on port 8000

uv run -m http.server 8000

# Start HTTP tunnel

uvx ngrok http 8000

Download [Llamafile](https://github.com/Mozilla-Ocho/llamafile). Run the [Llama-3.2-1B-Instruct.Q6\_K.llamafile](https://huggingface.co/Mozilla/Llama-3.2-1B-Instruct-llamafile/blob/main/Llama-3.2-1B-Instruct.Q6_K.llamafile?download=true) model with it.

Create a tunnel to the Llamafile server using [ngrok](https://ngrok.com/).

What is the ngrok URL? It might look like: https://[random].ngrok-free.app/

Check

Check all Save

Save regularly. Your last saved submission will be evaluated.

Bottom of Form

# Best of luck!

Ended at Wed, 5 Feb, 2025, 11:59 pm IST Score: 0 Check all Save

# TDS 2025 Jan GA3 - Large Language Models

## Instructions

1. **Learn what you need**. Reading material is provided, but feel free to skip it if you can answer the question. (Or learn it, just for pleasure.)
2. **Check answers regularly** by pressing Check. It shows which answers are right or wrong. You can check multiple times.
3. **Save regularly** by pressing Save. You can save multiple times. Your last saved submission will be evaluated.
4. **Reloading is OK**. Your answers are saved in your browser (not server). Questions won't change except for randomized parameters.
5. **Browser may struggle**. If you face loading issues, turn off security restrictions or try a different browser.
6. **Use anything**. You can use any resources you want. The Internet, ChatGPT, friends, whatever. Use any libraries or frameworks you want.
7. **It's hackable**. It's possible to get the answer to some questions by hacking the code for this quiz. That's allowed.

**Note:** You'll run multiple servers in this exam. All of them must be running simultaneously while checking or saving answers.

**Have questions?** [**Join the discussion on Discourse**](https://discourse.onlinedegree.iitm.ac.in/t/ga3-large-language-models-discussion-thread-tds-jan-2025/163247)

You are logged in as **22f1001679@ds.study.iitm.ac.in**.

Logout

#### **Recent saves (most recent is your official score)**

Reloadfrom 2/5/2025, 10:56:17 PM. Score: 8.5

Reloadfrom 2/5/2025, 10:37:54 PM. Score: 7

Reloadfrom 2/5/2025, 10:18:19 PM. Score: 7

Top of Form

# Questions

1. [LLM Sentiment Analysis](https://exam.sanand.workers.dev/tds-2025-01-ga3#hq-llm-sentiment-analysis) (1 mark)
2. [LLM Token Cost](https://exam.sanand.workers.dev/tds-2025-01-ga3#hq-token-cost) (0.75 marks)
3. [Generate addresses with LLMs](https://exam.sanand.workers.dev/tds-2025-01-ga3#hq-generate-addresses-with-llms) (1 mark)
4. [LLM Vision](https://exam.sanand.workers.dev/tds-2025-01-ga3#hq-llm-vision) (1 mark)
5. [LLM Embeddings](https://exam.sanand.workers.dev/tds-2025-01-ga3#hq-llm-embeddings) (0.75 marks)
6. [Embedding Similarity](https://exam.sanand.workers.dev/tds-2025-01-ga3#hq-embedding-similarity) (1 mark)
7. [Vector Databases](https://exam.sanand.workers.dev/tds-2025-01-ga3#hq-vector-databases) (1.5 marks)
8. [Function Calling](https://exam.sanand.workers.dev/tds-2025-01-ga3#hq-function-calling) (1.5 marks)
9. [Get an LLM to say Yes](https://exam.sanand.workers.dev/tds-2025-01-ga3#hq-get-llm-to-say-yes) (1 mark)

1 LLM Sentiment Analysis (1 mark)

## LLM Sentiment Analysis

[OpenAI's API](https://platform.openai.com/) provides access to language models like GPT 4o, GPT 4o mini, etc.

For more details, read OpenAI's guide for:

* [Text Generation](https://platform.openai.com/docs/guides/text-generation)
* [Vision](https://platform.openai.com/docs/guides/vision)
* [Structured Outputs](https://platform.openai.com/docs/guides/structured-outputs)

Start with this quick tutorial:

Here's a minimal example using curl to generate text:

curl https://api.openai.com/v1/chat/completions \

-H "Content-Type: application/json" \

-H "Authorization: Bearer $OPENAI\_API\_KEY" \

-d '{

"model": "gpt-4o-mini",

"messages": [{ "role": "user", "content": "Write a haiku about programming." }]

}'

Let's break down the request:

* curl https://api.openai.com/v1/chat/completions: The API endpoint for text generation.
* -H "Content-Type: application/json": The content type of the request.
* -H "Authorization: Bearer $OPENAI\_API\_KEY": The API key for authentication.
* -d: The request body.
  + "model": "gpt-4o-mini": The model to use for text generation.
  + "messages":: The messages to send to the model.
    - "role": "user": The role of the message.
    - "content": "Write a haiku about programming.": The content of the message.

This video explains how to use large language models (LLMs) for sentiment analysis and classification, covering:

* **Sentiment Analysis**: Use OpenAI API to identify the sentiment of movie reviews as positive or negative.
* **Prompt Engineering**: Learn how to craft effective prompts to get desired results from LLMs.
* **LLM Training**: Understand how to train LLMs by providing examples and feedback.
* **OpenAI API Integration**: Integrate OpenAI API into Python code to perform sentiment analysis.
* **Tokenization**: Learn about tokenization and its impact on LLM input and cost.
* **Zero-Shot, One-Shot, and Multi-Shot Learning**: Understand different approaches to using LLMs for learning.

Here are the links used in the video:

* [Jupyter Notebook](https://colab.research.google.com/drive/1tVZBD9PKto1kPmVJFNUt0tdzT5EmLLWs)
* [Movie reviews dataset](https://drive.google.com/file/d/1X33ao8_PE17c3htkQ-1p2dmW2xKmOq8Q/view)
* [OpenAI Playground](https://platform.openai.com/playground/chat)
* [OpenAI Pricing](https://openai.com/api/pricing/)
* [OpenAI Tokenizer](https://platform.openai.com/tokenizer)
* [OpenAI API Reference](https://platform.openai.com/docs/api-reference/)
* [OpenAI Docs](https://platform.openai.com/docs/overview)

DataSentinel Inc. is a tech company specializing in building advanced natural language processing (NLP) solutions. Their latest project involves integrating an AI-powered sentiment analysis module into an internal monitoring dashboard. The goal is to automatically classify large volumes of unstructured feedback and text data from various sources as either GOOD, BAD, or NEUTRAL. As part of the quality assurance process, the development team needs to test the integration with a series of sample inputs—even ones that may not represent coherent text—to ensure that the system routes and processes the data correctly.

Before rolling out the live system, the team creates a test harness using Python. The harness employs the httpx library to send POST requests to OpenAI's API. For this proof-of-concept, the team uses the dummy model gpt-4o-mini along with a dummy API key in the Authorization header to simulate real API calls.

One of the test cases involves sending a sample piece of meaningless text:

**jv** TEjx cIdoPlAuwfViY4D ks 0lXkjdI9ABB E6 e ESm

Write a Python program that uses httpx to send a POST request to OpenAI's API to analyze the sentiment of this (meaningless) text into GOOD, BAD or NEUTRAL. Specifically:

1. Make sure you pass an Authorization header with dummy API key.
2. Use gpt-4o-mini as the model.
3. The first message must be a system message asking the LLM to analyze the sentiment of the text. Make sure you mention GOOD, BAD, or NEUTRAL as the categories.
4. The second message must be **exactly** the text contained above.

This test is crucial for DataSentinel Inc. as it validates both the API integration and the correctness of message formatting in a controlled environment. Once verified, the same mechanism will be used to process genuine customer feedback, ensuring that the sentiment analysis module reliably categorizes data as GOOD, BAD, or NEUTRAL. This reliability is essential for maintaining high operational standards and swift response times in real-world applications.

**Note**: This uses a dummy httpx library, not the real one. You can only use:

1. response = httpx.get(url, \*\*kwargs)
2. response = httpx.post(url, json=None, \*\*kwargs)
3. response.raise\_for\_status()
4. response.json()

Code

Check

2 LLM Token Cost (0.75 marks)

LexiSolve Inc. is a startup that delivers a conversational AI platform to enterprise clients. The system leverages OpenAI’s language models to power a variety of customer service, sentiment analysis, and data extraction features. Because pricing for these models is based on the number of tokens processed—and strict token limits apply—accurate token accounting is critical for managing costs and ensuring system stability.

To optimize operational costs and prevent unexpected API overages, the engineering team at LexiSolve has developed an internal diagnostic tool that simulates and measures token usage for typical prompts sent to the language model.

One specific test case an understanding of text tokenization. Your task is to generate data for that test case.

Specifically, when you make a request to OpenAI's GPT-4o-Mini with just this user message:

List only the valid English words from these: neoqW, I, J8Oo6CcHrM, uUvsUfxs, JtupR, sdvj9vzt, IyEkC, rWrev, Lff5a, dqpM9Y9, B, p, r, ILtdOhxHT, nfb, InnqWC, E92T1ix, WEU9XwLYo, ZtTWhDA, 69UxAM6PXQ, 9NHdS, RPL4AR6, 61, 2, w, aHFRSwML, l7Q, FDGdW, aIE4yDddA, 5gSZcp, yhrdvuIqp, u, ccxIQ, 1QGLn, XrGZ2mM, vQ6sCOJw4O, Cah32h, zPMZUic, c, riTqs, d0jWz9JAPV, eTkYWjvV, Pu, J, jSSVJ, mMd, AYR9v, OQwZ, N51gbc, bGq2TIxus, oFJsQ, YEeLIFsoW, RinnHNM, 0tk61D, 5lItx2kB, s5lxl6Ze, 0hW, OK, yoENA, D0B0Qpz6e, REKCly, A, Svbjq6Bfpf, stQfZARqu, q, T1kxH8rL7N, 5xfw9, 2vTZZIPkYC, 1V, t3ut1d, t7bSc, Yt6GLzj0bK, eqNsc9tjj

... how many input tokens does it use up?

Number of tokens:

Remember: indicating that this is a user message takes up a few extra tokens. You actually need to make the request to get the answer.

Check

3 Generate addresses with LLMs (1 mark)

## LLM Text Extraction

[JSON](https://exam.sanand.workers.dev/json.md) is one of the most widely used formats in the world for applications to exchange data.

This video explains how to use LLMs to extract structure from unstructured data, covering:

* **LLM for Data Extraction**: Use OpenAI's API to extract structured information from unstructured data like addresses.
* **JSON Schema**: Define a JSON schema to ensure consistent and structured output from the LLM.
* **Prompt Engineering**: Craft effective prompts to guide the LLM's response and improve accuracy.
* **Data Cleaning**: Use string functions and OpenAI's API to clean and standardize data.
* **Data Analysis**: Analyze extracted data using Pandas to gain insights.
* **LLM Limitations**: Understand the limitations of LLMs, including potential errors and inconsistencies in output.
* **Production Use Cases**: Explore real-world applications of LLMs for data extraction, such as customer service email analysis.

Here are the links used in the video:

* [Jupyter Notebook](https://colab.research.google.com/drive/1Z8mG-RPTSYY4qwkoNdzRTc4StbnwOXeE)
* [JSON Schema](https://json-schema.org/)
* [Function calling](https://platform.openai.com/docs/guides/function-calling)

Structured Outputs is a feature that ensures the model will always generate responses that adhere to your supplied [JSON Schema](https://json-schema.org/overview/what-is-jsonschema), so you don't need to worry about the model omitting a required key, or hallucinating an invalid enum value.

curl https://api.openai.com/v1/chat/completions \

-H "Authorization: Bearer $OPENAI\_API\_KEY" \

-H "Content-Type: application/json" \

-d '{

"model": "gpt-4o-2024-08-06",

"messages": [

{ "role": "system", "content": "You are a helpful math tutor. Guide the user through the solution step by step." },

{ "role": "user", "content": "how can I solve 8x + 7 = -23" }

],

"response\_format": {

"type": "json\_schema",

"json\_schema": {

"name": "math\_response",

"strict": true

"schema": {

"type": "object",

"properties": {

"steps": {

"type": "array",

"items": {

"type": "object",

"properties": { "explanation": { "type": "string" }, "output": { "type": "string" } },

"required": ["explanation", "output"],

"additionalProperties": false

}

},

"final\_answer": { "type": "string" }

},

"required": ["steps", "final\_answer"],

"additionalProperties": false

}

}

}

}'

Here's what the response\_format tells OpenAI. The items marked ⚠️ are OpenAI specific requirements for the JSON schema.

* "type": "json\_schema": We want you to generate a JSON response that follows this schema.
* "json\_schema":: We're going to give you a schema.
  + "name": "math\_response": The schema is called math\_response. (We can call it anything.)
  + "strict": true: Follow the schema exactly.
  + "schema":: Now, here's the actual JSON schema.
    - "type": "object": Return an object. ⚠️ The root object **must** be an object.
    - "properties":: The object has these properties:
      * "steps":: There's a steps property.
        + "type": "array": It's an array.
        + "items":: Each item in the array...

"type": "object": ... is an object.

"properties":: The object has these properties:

"explanation":: There's an explanation property.

"type": "string": ... which is a string.

"output":: There's an output property.

"type": "string": ... which is a string, too.

"required": ["explanation", "output"]: ⚠️ You **must** add "required": [...] and include **all** fields int he object.

"additionalProperties": false: ⚠️ OpenAI requires every object to have "additionalProperties": false.

* + - * "final\_answer":: There's a final\_answer property.
        + "type": "string": ... which is a string.
    - "required": ["steps", "final\_answer"]: ⚠️ You **must** add "required": [...] and include **all** fields in the object.
    - "additionalProperties": false: ⚠️ OpenAI requires every object to have "additionalProperties": false.

RapidRoute Solutions is a logistics and delivery company that relies on accurate and standardized address data to optimize package routing. Recently, they encountered challenges with manually collecting and verifying new addresses for testing their planning software. To overcome this, the company decided to create an automated address generator using a language model, which would provide realistic, standardized U.S. addresses that could be directly integrated into their system.

The engineering team at RapidRoute is tasked with designing a service that uses OpenAI's GPT-4o-Mini model to generate fake but plausible address data. The addresses must follow a strict format, which is critical for downstream processes such as geocoding, routing, and verification against customer databases. For consistency and validation, the development team requires that the addresses be returned as structured JSON data with no additional properties that could confuse their parsers.

As part of the integration process, you need to write the body of the request to an [OpenAI chat completion call](https://platform.openai.com/docs/api-reference/chat) that:

* Uses model gpt-4o-mini
* Has a system message: Respond in JSON
* Has a user message: Generate 10 random addresses in the US
* Uses [structured outputs](https://platform.openai.com/docs/guides/structured-outputs/) to respond with an object addresses which is an array of objects with **required** fields: street (string) latitude (number) city (string) .
* Sets additionalProperties to false to prevent additional properties.

Note that you don't need to run the request or use an API key; your task is simply to write the correct JSON body.

What is the JSON body we should send to https://api.openai.com/v1/chat/completions for this? (No need to run it or to use an API key. Just write the body of the request below.)

There's no answer box above. Figure out how to enable it. That's part of the test.

Check

4 LLM Vision (1 mark)

# Base 64 Encoding

Base64 is a method to convert binary data into ASCII text. It's essential when you need to transmit binary data through text-only channels or embed binary content in text formats.

Watch this quick explanation of how Base64 works (3 min):

Here's how it works:

* It takes 3 bytes (24 bits) and converts them into 4 ASCII characters
* ... using 64 characters: A-Z, a-z, 0-9, + and / (padding with = to make the length a multiple of 4)
* There's a URL-safe variant of Base64 that replaces + and / with - and \_ to avoid issues in URLs
* Base64 adds ~33% overhead (since every 3 bytes becomes 4 characters)

Common Python operations with Base64:

**import** base64

# Basic encoding/decoding

text = "Hello, World!"

# Convert text to base64

encoded = base64.b64encode(text.encode()).decode() # SGVsbG8sIFdvcmxkIQ==

# Convert base64 back to text

decoded = base64.b64decode(encoded).decode() # Hello, World!

# Convert to URL-safe base64

url\_safe = base64.urlsafe\_b64encode(text.encode()).decode() # SGVsbG8sIFdvcmxkIQ==

# Working with binary files (e.g., images)

**with** open('image.png', 'rb') **as** f:

binary\_data = f.read()

image\_b64 = base64.b64encode(binary\_data).decode()

# Data URI example (embed images in HTML/CSS)

data\_uri = f"data:image/png;base64,{image\_b64}"

Data URIs allow embedding binary data directly in HTML/CSS. This reduces the number of HTTP requests and also works offline. But it increases the file size.

For example, here's an SVG image embedded as a data URI:

<**img**

src="data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHZpZXdCb3g9IjAgMCAzMiAzMiI+PGNpcmNsZSBjeD0iMTYiIGN5PSIxNiIgcj0iMTUiIGZpbGw9IiMyNTYzZWIiLz48cGF0aCBmaWxsPSIjZmZmIiBkPSJtMTYgNyAyIDcgNyAyLTcgMi0yIDctMi03LTctMiA3LTJaIi8+PC9zdmc+"

/>

Base64 is used in many places:

* JSON: Encoding binary data in JSON payloads
* Email: MIME attachments encoding
* Auth: HTTP Basic Authentication headers
* JWT: Encoding tokens in web authentication
* SSL/TLS: PEM certificate format
* SAML: Encoding assertions in SSO
* Git: Encoding binary files in patches

Tools for working with Base64:

* [Base64 Decoder/Encoder](https://www.base64decode.org/) for online encoding/decoding
* [data: URI Generator](https://dopiaza.org/tools/datauri/index.php) converts files to Data URIs

## Vision Models

You'll learn how to use LLMs to interpret images and extract useful information, covering:

* **Setting Up Vision Models**: Integrate vision capabilities with LLMs using APIs like OpenAI's Chat Completion.
* **Sending Image URLs for Analysis**: Pass URLs or base64-encoded images to LLMs for processing.
* **Reading Image Responses**: Get detailed textual descriptions of images, from scenic landscapes to specific objects like cricketers or bank statements.
* **Extracting Data from Images**: Convert extracted image data to various formats like Markdown tables or JSON arrays.
* **Handling Model Hallucinations**: Address inaccuracies in extraction results, understanding how different prompts can affect output quality.
* **Cost Management for Vision Models**: Adjust detail settings (e.g., "detail: low") to balance cost and output precision.

Here are the links used in the video:

* [Jupyter Notebook](https://colab.research.google.com/drive/1bK0b1XMrZWImtw01T1w9NGraDkiVi8mS)
* [OpenAI Chat API Reference](https://platform.openai.com/docs/api-reference/chat/create)
* [OpenAI Vision Guide](https://platform.openai.com/docs/guides/vision)
* [Sample images used](https://drive.google.com/drive/folders/14MFc7XmGIUDU4-vbmF9305c1SSQrM-gR)

Here is an example of how to analyze an image using the OpenAI API.

curl https://api.openai.com/v1/chat/completions \

-H "Content-Type: application/json" \

-H "Authorization: Bearer $OPENAI\_API\_KEY" \

-d '{

"model": "gpt-4o-mini",

"messages": [

{

"role": "user",

"content": [

{"type": "text", "text": "What is in this image?"},

{

"type": "image\_url",

"detail": "low",

"image\_url": {"url": "https://upload.wikimedia.org/wikipedia/commons/3/34/Correlation\_coefficient.png"}

}

]

}

]

}'

Let's break down the request:

* curl https://api.openai.com/v1/chat/completions: The API endpoint for text generation.
* -H "Content-Type: application/json": The content type of the request.
* -H "Authorization: Bearer $OPENAI\_API\_KEY": The API key for authentication.
* -d: The request body.
  + "model": "gpt-4o-mini": The model to use for text generation.
  + "messages":: The messages to send to the model.
    - "role": "user": The role of the message.
    - "content":: The content of the message.
      * {"type": "text", "text": "What is in this image?"}: The text message.
      * {"type": "image\_url"}: The image message.
        + "detail": "low": The detail level of the image. low uses fewer tokens at lower detail. high uses more tokens for higher detail.
        + "image\_url": {"url": "https://upload.wikimedia.org/wikipedia/commons/3/34/Correlation\_coefficient.png"}: The URL of the image.

You can send images in a [base64 encoded format](https://exam.sanand.workers.dev/base64-image.md), too. For example:

# Download image and convert to base64 in one step

IMAGE\_BASE64=$(curl -s "https://upload.wikimedia.org/wikipedia/commons/3/34/Correlation\_coefficient.png" | base64 -w 0)

# Send to OpenAI API

curl https://api.openai.com/v1/chat/completions \

-H "Content-Type: application/json" \

-H "Authorization: Bearer $OPENAI\_API\_KEY" \

-d @- << EOF

{

"model": "gpt-4o-mini",

"messages": [

{

"role": "user",

"content": [

{"type": "text", "text": "What is in this image?"},

{

"type": "image\_url",

"image\_url": { "url": "data:image/png;base64,$IMAGE\_BASE64" }

}

]

}

]

}

EOF

Acme Global Solutions manages hundreds of invoices from vendors every month. To streamline their accounts payable process, the company is developing an automated document processing system. This system uses a computer vision model to extract useful text from scanned invoice images. Critical pieces of data such as vendor email addresses, invoice or transaction numbers, and other details are embedded within these documents.

Your team is tasked with integrating OpenAI's vision model into the invoice processing workflow. The chosen model, gpt-4o-mini, is capable of analyzing both text and image inputs simultaneously. When an invoice is received—for example, an invoice image may contain a vendor email like alice.brown@acmeglobal.com and a transaction number such as 34921. The system needs to extract all embedded text to automatically populate the vendor management system.

The automated process will send a POST request to OpenAI's API with two inputs in a single user message:

1. Text: A simple instruction "Extract text from this image."
2. Image URL: A base64 URL representing the invoice image that might include the email and the transaction number among other details.

Here is an example invoice image:

Write just the JSON body (not the URL, nor headers) for the POST request that sends these two pieces of content (text and image URL) to the OpenAI API endpoint.

1. Use gpt-4o-mini as the model.
2. Send a single user message to the model that has a text and an image\_url content (in that order).
3. The text content should be Extract text from this image.
4. Send the image\_url as a base64 URL of the image above. **CAREFUL**: Do not modify the image.

Write your JSON body here:

Check

5 LLM Embeddings (0.75 marks)

## Embeddings: OpenAI and Local Models

Embedding models convert text into a list of numbers. These are like a map of text in numerical form. Each number represents a feature, and similar texts will have numbers close to each other. So, if the numbers are similar, the text they represent mean something similar.

This is useful because text similarity is important in many common problems:

1. **Search**. Find similar documents to a query.
2. **Classification**. Classify text into categories.
3. **Clustering**. Group similar items into clusters.
4. **Anomaly Detection**. Find an unusual piece of text.

You can run embedding models locally or using an API. Local models are better for privacy and cost. APIs are better for scale and quality.

| **Feature** | **Local Models** | **API** |
| --- | --- | --- |
| **Privacy** | High | Dependent on provider |
| **Cost** | High setup, low after that | Pay-as-you-go |
| **Scale** | Limited by local resources | Easily scales with demand |
| **Quality** | Varies by model | Typically high |

The [Massive Text Embedding Benchmark (MTEB)](https://huggingface.co/spaces/mteb/leaderboard) provides comprehensive comparisons of embedding models. These models are compared on several parameters, but here are some key ones to look at:

1. **Rank**. Higher ranked models have higher quality.
2. **Memory Usage**. Lower is better (for similar ranks). It costs less and is faster to run.
3. **Embedding Dimensions**. Lower is better. This is the number of numbers in the array. Smaller dimensions are cheaper to store.
4. **Max Tokens**. Higher is better. This is the number of input tokens (words) the model can take in a single input.
5. Look for higher scores in the columns for Classification, Clustering, Summarization, etc. based on your needs.

### **Local Embeddings**

[](https://youtu.be/OATCgQtNX2o)

Here's a minimal example using a local embedding model:

# /// script

# requires-python = "==3.12"

# dependencies = [

# "sentence-transformers",

# "httpx",

# "numpy",

# ]

# ///

**from** sentence\_transformers **import** SentenceTransformer

**import** numpy **as** np

model = SentenceTransformer('BAAI/bge-base-en-v1.5') # A small, high quality model

**async** **def** **embed**(text: str) -> list[float]:

"""Get embedding vector for text using local model."""

**return** model.encode(text).tolist()

**async** **def** **get\_similarity**(text1: str, text2: str) -> float:

"""Calculate cosine similarity between two texts."""

emb1 = np.array(**await** embed(text1))

emb2 = np.array(**await** embed(text2))

**return** float(np.dot(emb1, emb2) / (np.linalg.norm(emb1) \* np.linalg.norm(emb2)))

**async** **def** **main**():

print(**await** get\_similarity("Apple", "Orange"))

print(**await** get\_similarity("Apple", "Lightning"))

**if** \_\_name\_\_ == "\_\_main\_\_":

**import** asyncio

asyncio.run(main())

Note the get\_similarity function. It uses a [Cosine Similarity](https://en.wikipedia.org/wiki/Cosine_similarity) to calculate the similarity between two embeddings.

### **OpenAI Embeddings**

For comparison, here's how to use OpenAI's API with direct HTTP calls. Replace the embed function in the earlier script:

**import** os

**import** httpx

**async** **def** **embed**(text: str) -> list[float]:

"""Get embedding vector for text using OpenAI's API."""

**async** **with** httpx.AsyncClient() **as** client:

response = **await** client.post(

"https://api.openai.com/v1/embeddings",

headers={"Authorization": f"Bearer {os.environ['OPENAI\_API\_KEY']}"},

json={"model": "text-embedding-3-small", "input": text}

)

**return** response.json()["data"][0]["embedding"]

**NOTE**: You need to set the OPENAI\_API\_KEY environment variable for this to work.

**SecurePay**, a leading fintech startup, has implemented an innovative feature to detect and prevent fraudulent activities in real time. As part of its security suite, the system analyzes personalized transaction messages by converting them into embeddings. These embeddings are compared against known patterns of legitimate and fraudulent messages to flag unusual activity.

Imagine you are working on the SecurePay team as a junior developer tasked with integrating the text embeddings feature into the fraud detection module. When a user initiates a transaction, the system sends a personalized verification message to the user's registered email address. This message includes the user's email address and a unique transaction code (a randomly generated number). Here are 2 verification messages:

Dear user, please verify your transaction **code** 82698 sent to 22f1001679@ds.study.iitm.ac.**in**

Dear user, please verify your transaction **code** 2709 sent to 22f1001679@ds.study.iitm.ac.**in**

The goal is to capture this message, convert it into a meaningful embedding using OpenAI's text-embedding-3-small model, and subsequently use the embedding in a machine learning model to detect anomalies.

Your task is to write the JSON body for a POST request that will be sent to the OpenAI API endpoint to obtain the text embedding for the 2 given personalized transaction verification messages above. This will be sent to the endpoint https://api.openai.com/v1/embeddings.

Write your JSON body here:

Check

6 Embedding Similarity (1 mark)

## Topic Modeling

You'll learn to use text embeddings to find text similarity and use that to create topics automatically from text, covering:

* **Embeddings**: How large language models convert text into numerical representations.
* **Similarity Measurement**: Understanding how similar embeddings indicate similar meanings.
* **Embedding Visualization**: Using tools like Tensorflow Projector to visualize embedding spaces.
* **Embedding Applications**: Using embeddings for tasks like classification and clustering.
* **OpenAI Embeddings**: Using OpenAI's API to generate embeddings for text.
* **Model Comparison**: Exploring different embedding models and their strengths and weaknesses.
* **Cosine Similarity**: Calculating cosine similarity between embeddings for more reliable similarity measures.
* **Embedding Cost**: Understanding the cost of generating embeddings using OpenAI's API.
* **Embedding Range**: Understanding the range of values in embeddings and their significance.

Here are the links used in the video:

* [Jupyter Notebook](https://colab.research.google.com/drive/15L075RLrwXkxa29EGT-1sNm_dqJRBTe_)
* [Tensorflow projector](https://projector.tensorflow.org/)
* [Embeddings guide](https://platform.openai.com/docs/guides/embeddings)
* [Embeddings reference](https://platform.openai.com/docs/api-reference/embeddings)
* [Clustering on scikit-learn](https://scikit-learn.org/stable/modules/clustering.html)
* [Massive text embedding leaderboard (MTEB)](https://huggingface.co/spaces/mteb/leaderboard)
* [gte-large-en-v1.5 embedding model](https://huggingface.co/Alibaba-NLP/gte-large-en-v1.5)
* [Embeddings similarity threshold](https://www.s-anand.net/blog/embeddings-similarity-threshold/)

ShopSmart is an online retail platform that places a high value on customer feedback. Each month, the company receives hundreds of comments from shoppers regarding product quality, delivery speed, customer service, and more. To automatically understand and cluster this feedback, ShopSmart's data science team uses text embeddings to capture the semantic meaning behind each comment.

As part of a pilot project, ShopSmart has curated a collection of 25 feedback phrases that represent a variety of customer sentiments. Examples of these phrases include comments like “Fast shipping and great service,” “Product quality could be improved,” “Excellent packaging,” and so on. Due to limited processing capacity during initial testing, you have been tasked with determine which pair(s) of 5 of these phrases are most similar to each other. This similarity analysis will help in grouping similar feedback to enhance the company’s understanding of recurring customer issues.

ShopSmart has written a Python program that has the 5 phrases and their embeddings as an array of floats. It looks like this:

embeddings = {"The overall shopping experience was excellent.":[0.012937188148498535,-0.04513780027627945,-0.36088019609451294,0.04213787242770195,0.14521872997283936,-0.030332602560520172,0.07010941952466965,0.23666097223758698,-0.098664291203022,0.020874496549367905,0.164662703871727,-0.07955364137887955,-0.13399676978588104,-0.1312190592288971,0.13544118404388428,0.1304413080215454,-0.014138548634946346,-0.0703316405415535,-0.00004003809954156168,0.03599913418292999,0.2739934027194977,0.01938842236995697,0.11477500945329666,-0.13366344571113586,-0.2904374599456787,-0.16377383470535278,-0.1097751334309578,0.09233111143112183,-0.0064512332901358604,-0.041471224278211594,0.12055265158414841,-0.11038623005151749,0.305325984954834,0.014930196106433868,-0.13966330885887146,0.08122026920318604,0.02066616900265217,-0.11688607186079025,0.2195502668619156,-0.07010941952466965,0.2346610128879547,0.24821624159812927,0.03355474770069122,0.258215993642807,0.059943001717329025,0.08566460013389587,-0.009478938765823841,0.05527644604444504,-0.005794999189674854,-0.10099756717681885],"Shipping costs were too high.":[-0.02132924273610115,-0.05078135058283806,0.24659079313278198,0.03407837450504303,-0.031469374895095825,0.04534817487001419,-0.14255358278751373,0.028483819216489792,-0.0895128846168518,0.05390138924121857,-0.0863390564918518,0.025431020185351372,-0.10597378760576248,0.02617068588733673,0.04362677410244942,-0.020603027194738388,0.1553564965724945,-0.12254228442907333,-0.3750503957271576,0.08009897172451019,0.13728179037570953,0.17526021599769592,-0.08456385880708694,-0.21130205690860748,-0.06810295581817627,0.008573387749493122,0.2928534746170044,-0.27736085653305054,0.12576991319656372,-0.23002229630947113,0.1522364616394043,-0.13523761928081512,0.16622285544872284,-0.1358831524848938,-0.32512974739074707,0.04222813621163368,-0.11146076023578644,0.23475615680217743,0.1606282889842987,0.07009332627058029,-0.08875977247953415,-0.0171198770403862,0.1295354813337326,0.033890094608068466,0.039941899478435516,0.14147770404815674,0.10349927842617035,-0.037790145725011826,0.022405119612812996,-0.013334139250218868],"The product description matched the item.":[-0.1778346747159958,0.015024187043309212,-0.48206639289855957,-0.025718823075294495,-0.016542760655283928,-0.14746320247650146,0.08109830319881439,0.14048422873020172,-0.06655876338481903,-0.014773784205317497,-0.022116249427199364,-0.09764105826616287,0.0843939259648323,-0.21104943752288818,0.05166381597518921,0.24917533993721008,-0.04652651399374008,-0.03644577041268349,-0.3680764436721802,0.14306902885437012,0.19114643335342407,0.09570245444774628,0.12562158703804016,0.04345705732703209,-0.05486251413822174,-0.1628427952528,-0.04840049892663956,-0.08885271847248077,0.20407046377658844,0.14849711954593658,0.017899783328175545,-0.17020949721336365,0.13428069651126862,-0.2234565168619156,0.00254037999548018,0.044975630939006805,0.14862637221813202,-0.06594487279653549,0.15728546679019928,0.006142953876405954,-0.207172229886055,-0.020533055067062378,-0.05463634431362152,0.09492701292037964,-0.03237469866871834,0.06752806901931763,-0.08736645430326462,0.08297228813171387,-0.036898110061883926,-0.045621830970048904],"There was a delay in delivery.":[0.14162038266658783,0.133348748087883,-0.04399004951119423,-0.10571397840976715,-0.12250789999961853,0.039634909480810165,0.010010556317865849,0.028512069955468178,-0.011859141290187836,-0.11755745112895966,-0.011624150909483433,-0.05646016448736191,-0.07576064020395279,-0.26845210790634155,-0.060000672936439514,-0.07820453494787216,0.04865850880742073,-0.1497666984796524,-0.28549668192863464,0.24902629852294922,0.0857868641614914,0.053608957678079605,0.24727170169353485,0.0352797694504261,-0.16643528640270233,-0.060595981776714325,0.1174321249127388,-0.17596019804477692,0.04847051948308945,0.14939071238040924,0.12282121926546097,-0.10019955784082413,0.23448826372623444,-0.22408606112003326,-0.16217415034770966,0.1520226001739502,-0.0021325305569916964,0.19927117228507996,0.15578243136405945,0.1492653787136078,-0.26845210790634155,-0.1048993468284607,-0.11906138807535172,-0.012994923628866673,-0.07444469630718231,0.22797122597694397,-0.05166637524962425,-0.07469535619020462,-0.009728568606078625,0.23611752688884735],"The website is user-friendly.":[-0.17558817565441132,-0.15948393940925598,-0.4088399410247803,0.09409292787313461,0.1044178232550621,-0.19364051520824432,-0.15688647329807281,0.22987505793571472,0.04376717284321785,0.028831787407398224,0.07759906351566315,-0.09389811754226685,-0.13740554451942444,-0.03180262818932533,0.22506976127624512,-0.02987077087163925,-0.2480572611093521,-0.08526156842708588,-0.08441739529371262,0.06123507767915726,0.2639017701148987,0.08117057383060455,0.024302469566464424,-0.1449381709098816,0.08207967877388,-0.005746876355260611,-0.13201580941677094,0.035715050995349884,-0.1213662400841713,0.032630570232868195,0.04873481020331383,-0.17909474670886993,0.17584791779518127,-0.1285741776227951,0.037273526191711426,-0.14143159985542297,0.1436394453048706,0.09279419481754303,0.1490941047668457,0.07467692345380783,-0.09409292787313461,0.09675531834363937,0.13350935280323029,-0.19415999948978424,-0.18454940617084503,0.15182143449783325,-0.043604832142591476,0.01301164273172617,0.20143288373947144,0.015333120711147785]}

Your task is to write a Python function most\_similar(embeddings) that will calculate the cosine similarity between each pair of these embeddings and return the pair that has the highest similarity. The result should be a tuple of the two phrases that are most similar.

Write your Python code here:

Check

7 Vector Databases (1.5 marks)

## Vector Databases

Vector databases are specialized databases that store and search vector embeddings efficiently.

Use vector databases when your embeddings exceed available memory or when you want it run fast at scale. (This is important. If your code runs fast and fits in memory, you **DON'T** need a vector database. You can can use numpy for these tasks.)

Vector databases are an evolving space.

The first generation of vector databases were written in C and typically used an algorithm called [HNSW](https://en.wikipedia.org/wiki/Hierarchical_navigable_small_world) (a way to approximately find the nearest neighbor). Some popular ones are:

* [**Chroma**](https://docs.trychroma.com/): Combines a vector index with a SQLite database. Easy to install, most popular.
* [**LanceDB**](https://lancedb.github.io/lancedb/): Written in Rust. Faster, easy to install, growing popular.
* [**FAISS**](https://github.com/facebookresearch/faiss): Meta's lightweight library
* [**Milvus**](https://milvus.io/): Distributed, cloud-native

In addition, most relational databases now support vector search. For example:

* [**DuckDB**](https://duckdb.org/): Supports vector search with [vss](https://duckdb.org/docs/extensions/vss.html).
* [**SQLite**](https://www.sqlite.org/): Supports vector search with [sqlite-vec](https://github.com/asg017/sqlite-vec).
* [**PostgreSQL**](https://www.postgresql.org/): Supports vector search with [pgvector](https://github.com/pgvector/pgvector).

Take a look at this [Vector DB Comparison](https://superlinked.com/vector-db-comparison).

Watch this Vector Database Tutorial (3 min):

[](https://youtu.be/klTvEwg3oJ4)

### **ChromaDB**

Here's a minimal example using Chroma:

# /// script

# requires-python = "==3.12"

# dependencies = [

# "chromadb",

# "sentence-transformers",

# ]

# ///

**import** chromadb

**from** chromadb.utils **import** embedding\_functions

**from** sentence\_transformers **import** SentenceTransformer

**async** **def** **setup\_vector\_db**():

"""Initialize Chroma DB with an embedding function."""

sentence\_transformer\_ef = embedding\_functions.SentenceTransformerEmbeddingFunction(

model\_name="BAAI/bge-base-en-v1.5"

)

client = chromadb.PersistentClient(path="./vector\_db")

collection = client.create\_collection(

name="documents",

embedding\_function=sentence\_transformer\_ef

)

**return** collection

**async** **def** **search\_similar**(collection, query: str, n\_results: int = 3) -> list[dict]:

"""Search for documents similar to the query."""

d = collection.query(query\_texts=[query], n\_results=n\_results)

**return** [

{"id": id, "text": text, "distance": distance}

**for** id, text, distance **in** zip(d["ids"][0], d["documents"][0], d["distances"][0])

]

**async** **def** **main**():

collection = **await** setup\_vector\_db()

# Add some documents

collection.add(

documents=["Apple is a fruit", "Orange is citrus", "Computer is electronic"],

ids=["1", "2", "3"]

)

# Search

results = **await** search\_similar(collection, "fruit")

print(results)

**if** \_\_name\_\_ == "\_\_main\_\_":

**import** asyncio

asyncio.run(main())

### **LanceDB**

Here's the same example using LanceDB:

# /// script

# requires-python = "==3.12"

# dependencies = [

# "lancedb",

# "pyarrow",

# "sentence-transformers",

# ]

# ///

**import** lancedb

**import** pyarrow **as** pa

**from** sentence\_transformers **import** SentenceTransformer

**async** **def** **setup\_vector\_db**():

"""Initialize LanceDB with an embedding function."""

model = SentenceTransformer("BAAI/bge-base-en-v1.5")

db = lancedb.connect("./vector\_db")

# Create table with schema for documents

table = db.create\_table(

"documents",

schema=pa.schema([

pa.field("id", pa.string()),

pa.field("text", pa.string()),

pa.field("vector", pa.list\_(pa.float32(), list\_size=768))

])

)

**return** table, model

**async** **def** **search\_similar**(table, model, query: str, n\_results: int = 3) -> list[dict]:

"""Search for documents similar to the query."""

query\_embedding = model.encode(query)

results = table.search(query\_embedding).limit(n\_results).to\_list()

**return** [{"id": r["id"], "text": r["text"], "distance": float(r["\_distance"])} **for** r **in** results]

**async** **def** **main**():

table, model = **await** setup\_vector\_db()

# Add some documents

documents = ["Apple is a fruit", "Orange is citrus", "Computer is electronic"]

embeddings = model.encode(documents)

table.add(data=[

{"id": str(i), "text": text, "vector": embedding}

**for** i, (text, embedding) **in** enumerate(zip(documents, embeddings), 1)

])

# Search

results = **await** search\_similar(table, model, "fruit")

print(results)

**if** \_\_name\_\_ == "\_\_main\_\_":

**import** asyncio

asyncio.run(main())

### **DuckDB**

Here's the same example using DuckDB:

# /// script

# requires-python = "==3.12"

# dependencies = [

# "duckdb",

# "sentence-transformers",

# ]

# ///

**import** duckdb

**from** sentence\_transformers **import** SentenceTransformer

**async** **def** **setup\_vector\_db**() -> tuple[duckdb.DuckDBPyConnection, SentenceTransformer]:

"""Initialize DuckDB with VSS extension and embedding model."""

# Initialize model

model = SentenceTransformer("BAAI/bge-base-en-v1.5")

vector\_dim = model.get\_sentence\_embedding\_dimension()

# Setup DuckDB with VSS extension

conn = duckdb.connect(":memory:")

conn.install\_extension("vss")

conn.load\_extension("vss")

# Create table with vector column

conn.execute(f"""

CREATE TABLE documents (

id VARCHAR,

text VARCHAR,

vector FLOAT[{vector\_dim}]

)

""")

# Create HNSW index for vector similarity search

conn.execute("CREATE INDEX vector\_idx ON documents USING HNSW (vector)")

**return** conn, model

**async** **def** **search\_similar**(conn: duckdb.DuckDBPyConnection, model: SentenceTransformer,

query: str, n\_results: int = 3) -> list[dict]:

"""Search for documents similar to query using vector similarity."""

# Encode query to vector

query\_vector = model.encode(query).tolist()

# Search using HNSW index with explicit FLOAT[] cast

results = conn.execute("""

SELECT id, text, array\_distance(vector, CAST(? AS FLOAT[768])) as distance

FROM documents

ORDER BY array\_distance(vector, CAST(? AS FLOAT[768]))

LIMIT ?

""", [query\_vector, query\_vector, n\_results]).fetchall()

**return** [{"id": r[0], "text": r[1], "distance": float(r[2])} **for** r **in** results]

**async** **def** **main**():

conn, model = **await** setup\_vector\_db()

# Add sample documents

documents = ["Apple is a fruit", "Orange is citrus", "Computer is electronic"]

embeddings = model.encode(documents).tolist()

# Insert documents and vectors

conn.executemany("""

INSERT INTO documents (id, text, vector)

VALUES (?, ?, ?)

""", [(str(i), text, embedding)

**for** i, (text, embedding) **in** enumerate(zip(documents, embeddings), 1)])

# Search similar documents

results = **await** search\_similar(conn, model, "fruit")

print(results)

**if** \_\_name\_\_ == "\_\_main\_\_":

**import** asyncio

asyncio.run(main())

InfoCore Solutions is a technology consulting firm that maintains an extensive internal knowledge base of technical documents, project reports, and case studies. Employees frequently search through these documents to answer client questions quickly or gain insights for ongoing projects. However, due to the sheer volume of documentation, traditional keyword-based search often returns too many irrelevant results.

To address this issue, InfoCore's data science team decides to integrate a semantic search feature into their internal portal. This feature uses text embeddings to capture the contextual meaning of both the documents and the user's query. The documents are pre-embedded, and when an employee submits a search query, the system computes the similarity between the query's embedding and those of the documents. The API then returns a ranked list of document identifiers based on similarity.

Imagine you are an engineer on the InfoCore team. Your task is to build a FastAPI POST endpoint that accepts an array of docs and query string via a JSON body. The endpoint is structured as follows:

POST /similarity

{

"docs": ["Contents of document 1", "Contents of document 2", "Contents of document 3", ...],

"query": "Your query string"

}

**Service Flow:**

1. **Request Payload:** The client sends a POST request with a JSON body containing:
   * **docs:** An array of document texts from the internal knowledge base.
   * **query:** A string representing the user's search query.
2. **Embedding Generation:** For each document in the docs array and for the query string, the API computes a text embedding using text-embedding-3-small.
3. **Similarity Computation:** The API then calculates the cosine similarity between the query embedding and each document embedding. This allows the service to determine which documents best match the intent of the query.
4. **Response Structure:** After ranking the documents by their similarity scores, the API returns the identifiers (or positions) of the three most similar documents. The JSON response might look like this:
5. {
6. "matches": ["Contents of document 3", "Contents of document 1", "Contents of document 2"]
7. }

Here, "Contents of document 3" is considered the closest match, followed by "Contents of document 1", then "Contents of document 2".

Make sure you enable CORS to allow OPTIONS and POST methods, perhaps allowing all origins and headers.

What is the API URL endpoint for your implementation? It might look like: http://127.0.0.1:8000/similarity

We'll check by sending a POST request to this URL with a JSON body containing random docs and query.

Check

8 Function Calling (1.5 marks)

## Function Calling with OpenAI

[Function Calling](https://platform.openai.com/docs/guides/function-calling) allows Large Language Models to convert natural language into structured function calls. This is perfect for building chatbots and AI assistants that need to interact with your backend systems.

OpenAI supports [Function Calling](https://platform.openai.com/docs/guides/function-calling) -- a way for LLMs to suggest what functions to call and how.

Here's a minimal example using Python and OpenAI's function calling that identifies the weather in a given location.

# /// script

# requires-python = ">=3.11"

# dependencies = [

# "httpx",

# ]

# ///

**import** httpx

**import** os

**from** typing **import** Dict, Any

**def** **query\_gpt**(user\_input: str, tools: list[Dict[str, Any]]) -> Dict[str, Any]:

response = httpx.post(

"https://api.openai.com/v1/chat/completions",

headers={

"Authorization": f"Bearer {os.getenv('OPENAI\_API\_KEY')}",

"Content-Type": "application/json",

},

json={

"model": "gpt-4o-mini",

"messages": [{"role": "user", "content": user\_input}],

"tools": tools,

"tool\_choice": "auto",

},

)

**return** response.json()["choices"][0]["message"]

WEATHER\_TOOL = {

"type": "function",

"function": {

"name": "get\_weather",

"description": "Get the current weather for a location",

"parameters": {

"type": "object",

"properties": {

"location": {"type": "string", "description": "City name or coordinates"}

},

"required": ["location"],

"additionalProperties": False,

},

"strict": True,

},

}

**if** \_\_name\_\_ == "\_\_main\_\_":

response = query\_gpt("What is the weather in San Francisco?", [WEATHER\_TOOL])

print([tool\_call["function"] **for** tool\_call **in** response["tool\_calls"]])

### **How to define functions**

The function definition is a [JSON schema](https://json-schema.org/) with a few OpenAI specific properties. See the [Supported schemas](https://platform.openai.com/docs/guides/structured-outputs#supported-schemas).

Here's an example of a function definition for scheduling a meeting:

MEETING\_TOOL = {

"type": "function",

"function": {

"name": "schedule\_meeting",

"description": "Schedule a meeting room for a specific date and time",

"parameters": {

"type": "object",

"properties": {

"date": {

"type": "string",

"description": "Meeting date in YYYY-MM-DD format"

},

"time": {

"type": "string",

"description": "Meeting time in HH:MM format"

},

"meeting\_room": {

"type": "string",

"description": "Name of the meeting room"

}

},

"required": ["date", "time", "meeting\_room"],

"additionalProperties": False

},

"strict": True

}

}

### **How to define multiple functions**

You can define multiple functions by passing a list of function definitions to the tools parameter.

Here's an example of a list of function definitions for handling employee expenses and calculating performance bonuses:

tools = [

{

"type": "function",

"function": {

"name": "get\_expense\_balance",

"description": "Get expense balance for an employee",

"parameters": {

"type": "object",

"properties": {

"employee\_id": {

"type": "integer",

"description": "Employee ID number"

}

},

"required": ["employee\_id"],

"additionalProperties": False

},

"strict": True

}

},

{

"type": "function",

"function": {

"name": "calculate\_performance\_bonus",

"description": "Calculate yearly performance bonus for an employee",

"parameters": {

"type": "object",

"properties": {

"employee\_id": {

"type": "integer",

"description": "Employee ID number"

},

"current\_year": {

"type": "integer",

"description": "Year to calculate bonus for"

}

},

"required": ["employee\_id", "current\_year"],

"additionalProperties": False

},

"strict": True

}

}

]

Best Practices:

1. **Use Strict Mode**
   * Always set strict: True to ensure valid function calls
   * Define all required parameters
   * Set additionalProperties: False
2. **Use tool choice**
   * Set tool\_choice: "required" to ensure that the model will always call one or more tools
   * The default is tool\_choice: "auto" which means the model will choose a tool only if appropriate
3. **Clear Descriptions**
   * Write detailed function and parameter descriptions
   * Include expected formats and units
   * Mention any constraints or limitations
4. **Error Handling**
   * Validate function inputs before execution
   * Return clear error messages
   * Handle missing or invalid parameters

**TechNova Corp.** is a multinational corporation that has implemented a digital assistant to support employees with various internal tasks. The assistant can answer queries related to human resources, IT support, and administrative services. Employees use a simple web interface to enter their requests, which may include:

* Checking the status of an IT support ticket.
* Scheduling a meeting.
* Retrieving their current expense reimbursement balance.
* Requesting details about their performance bonus.
* Reporting an office issue by specifying a department or issue number.

Each question is direct and templatized, containing one or more parameters such as an employee or ticket number (which might be randomized). In the backend, a FastAPI app routes each request by matching the query to one of a set of pre-defined functions. The response that the API returns is used by OpenAI to call the right function with the necessary arguments.

**Pre-Defined Functions:**

For this exercise, assume the following functions have been defined:

* get\_ticket\_status(ticket\_id: int)
* schedule\_meeting(date: str, time: str, meeting\_room: str)
* get\_expense\_balance(employee\_id: int)
* calculate\_performance\_bonus(employee\_id: int, current\_year: int)
* report\_office\_issue(issue\_code: int, department: str)

Each function has a specific signature, and the student’s FastAPI app should map specific queries to these functions.

**Example Questions (Templatized with a Random Number):**

1. **Ticket Status:**  
   Query: "What is the status of ticket 83742?"  
   → Should map to get\_ticket\_status(ticket\_id=83742)
2. **Meeting Scheduling:**  
   Query: "Schedule a meeting on 2025-02-15 at 14:00 in Room A."  
   → Should map to schedule\_meeting(date="2025-02-15", time="14:00", meeting\_room="Room A")
3. **Expense Reimbursement:**  
   Query: "Show my expense balance for employee 10056."  
   → Should map to get\_expense\_balance(employee\_id=10056)
4. **Performance Bonus Calculation:**  
   Query: "Calculate performance bonus for employee 10056 for 2025."  
   → Should map to calculate\_performance\_bonus(employee\_id=10056, current\_year=2025)
5. **Office Issue Reporting:**  
   Query: "Report office issue 45321 for the Facilities department."  
   → Should map to report\_office\_issue(issue\_code=45321, department="Facilities")

**Task Overview:**

Develop a FastAPI application that:

1. Exposes a GET endpoint /execute?q=... where the query parameter q contains one of the pre-templatized questions.
2. Analyzes the q parameter to identify which function should be called.
3. Extracts the parameters from the question text.
4. Returns a response in the following JSON format:

{ "name": "function\_name", "arguments": "{ ...JSON encoded parameters... }" }

For example, the query "What is the status of ticket 83742?" should return:

{

"name": "get\_ticket\_status",

"arguments": "{\"ticket\_id\": 83742}"

}

Make sure you enable **CORS** to allow GET requests from any origin.

What is the API URL endpoint for your implementation? It might look like: http://127.0.0.1:8000/execute

We'll check by sending a GET request to this URL with ?q=... containing a task. We'll verify that it matches the expected response. Arguments must be in the same order as the function definition.

Check

9 Get an LLM to say Yes (1 mark)

## Prompt Engineering

Prompt engineering is the process of crafting effective prompts for large language models (LLMs).

One of the best ways to approach prompt engineering is to think of LLMs as a smart colleague (with amnesia) who needs explicit instructions.

The most authoritative guides are from the LLM providers themselves:

* [Anthropic](https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/)
* [Google](https://cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/introduction-prompt-design)
* [OpenAI](https://platform.openai.com/docs/guides/prompt-engineering)

Here are some best practices:

### **Use prompt optimizers**

They rewrite your prompt to improve it. Explore:

* [Anthropic Prompt Optimizer](https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/prompt-improver)
* [OpenAI Prompt Generation](https://platform.openai.com/docs/guides/prompt-generation)
* [Google AI-powered prompt writing tools](https://cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/ai-powered-prompt-writing)

### **Be clear, direct, and detailed**

Be explicit and thorough. Include all necessary context, goals, and details so the model understands the full picture.

* **BAD**: Explain gravitation lensing. (Reason: Vague and lacks context or detail.)
* **GOOD**: Explain the concept of gravitational lensing to a high school student who understands basic physics, including how it’s observed and its significance in astronomy. (Reason: Specifies the audience, scope, and focus.)

When you ask a question, don’t be vague. Spell it out. Give every detail the listener needs. The clearer you are, the better the answer you’ll get. For example, don't just say, Explain Gravitation Lensing. Say, Explain the concept of gravitational lensing to a high school student who understands basic physics, including how it’s observed and its significance in astronomy.

[Anthropic](https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/be-clear-and-direct) | [OpenAI](https://platform.openai.com/docs/guides/prompt-engineering#tactic-include-details-in-your-query-to-get-more-relevant-answers) | [Google](https://cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/clear-instructions)

### **Give examples**

Provide 2-3 relevant examples to guide the model on the style, structure, or format you expect. This helps the model produce outputs consistent with your desired style.

* **BAD**: Explain how to tie a bow tie. (Reason: No examples or reference points given.)
* **GOOD**: Explain how to tie a bow tie. For example:
  1. To tie a shoelace, you cross the laces and pull them tight...
  2. To tie a necktie, you place it around the collar and loop it through...

Now, apply a similar step-by-step style to describe how to tie a bow tie. (Reason: Provides clear examples and a pattern to follow.)

Give examples to the model. If you want someone to build a house, show them a sketch. Don’t just say ‘build something.’ Giving examples is like showing a pattern. It makes everything easier to follow.

[Anthropic](https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/multishot-prompting) | [OpenAI](https://platform.openai.com/docs/guides/prompt-engineering#tactic-provide-examples) | [Google](https://cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/few-shot-examples)

### **Think step by step**

Instruct the model to reason through the problem step by step. This leads to more logical, well-structured answers.

* **BAD**: Given this transcript, is the customer satisfied? (Reason: No prompt for structured reasoning.)
* **GOOD**: Given this transcript, is the customer satisfied? Think step by step. (Reason: Directly instructs the model to break down reasoning into steps.)

Ask the model to think step by step. Don’t ask the model to just give the final answer right away. That's like asking someone to answer with the first thing that pops into their head. Instead, ask them to break down their thought process into simple moves — like showing each rung of a ladder as they climb. For example, when thinking step by step, the model could, A, list each customer question, B, find out if it was addressed, and C, decide that the agent answered only 2 out of the 3 questions.

[Anthropic](https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/chain-of-thought) | [OpenAI](https://platform.openai.com/docs/guides/prompt-engineering#strategy-give-models-time-to-think) | [Google](https://cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/break-down-prompts)

### **Assign a role**

Specify a role or persona for the model. This provides context and helps tailor the response style.

* **BAD**: Explain how to fix a software bug. (Reason: No role or perspective given.)
* **GOOD**: You are a seasoned software engineer. Explain how to fix a software bug in legacy code, including the debugging and testing process. (Reason: Assigns a clear, knowledgeable persona, guiding the style and depth.)

Tell the model who they are. Maybe they’re a seasoned software engineer fixing a legacy bug, or an experienced copy editor revising a publication. By clearly telling the model who they are, you help them speak with just the right expertise and style. Suddenly, your explanation sounds like it’s coming from a true specialist, not a random voice.

[Anthropic](https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/system-prompts) | [OpenAI](https://platform.openai.com/docs/guides/prompt-engineering#tactic-ask-the-model-to-adopt-a-persona) | [Google](https://cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/assign-role)

### **Use XML to structure your prompt**

Use XML tags to separate different parts of the prompt clearly. This helps the model understand structure and requirements.

* **BAD**: Here’s what I want: Provide a summary and then an example. (Reason: Unstructured, no clear separation of tasks.)
* **GOOD**:
* <**instructions**>
* Provide a summary of the concept of machine learning.
* </**instructions**>
* <**example**>
* Then provide a simple, concrete example of a machine learning application (e.g., an email spam filter).
* </**example**>

(Reason: Uses XML tags to clearly distinguish instructions from examples.)

Think of your request like a box. XML tags are neat little labels on that box. They help keep parts sorted, so nothing gets lost in the shuffle.

[Anthropic](https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/use-xml-tags) | [OpenAI](https://platform.openai.com/docs/guides/prompt-engineering#tactic-use-delimiters-to-clearly-indicate-distinct-parts-of-the-input) | [Google](https://cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/structure-prompts)

### **Use Markdown to format your output**

Encourage the model to use Markdown for headings, lists, code blocks, and other formatting features to produce structured, easily readable answers.

* **BAD**: Give me the steps in plain text. (Reason: No specific formatting instructions, less readable.)
* **GOOD**: Provide the steps in a markdown-formatted list with ### headings for each section and numbered bullet points. (Reason: Directly instructs to use Markdown formatting, making output more structured and clear.)
* **BAD**: Correct the spelling and show the corrections. (Reason: No specific formatting instructions)
* **GOOD**: Correct the spelling, showing <ins>additions</ins> and <del>deletions</del>. (Reason: Directly instructs to use HTML formatting, making output more structured and clear.)

Markdown is a simple formatting language that all models understand. You can have them add neat headings, sections, bold highlights, and bullet points. These make complex documents more scannable and easy on the eyes.

### **Use JSON for machine-readable output**

When you need structured data, ask for a JSON-formatted response. This ensures the output is machine-readable and organized.

* **BAD**: Just list the items. (Reason: Unstructured plain text makes parsing harder.)
* **GOOD**:
* Organize as an array of objects in a JSON format, like this:
* ```json
* [
* { "name": "Item 1", "description": "Description of Item 1" },
* { "name": "Item 2", "description": "Description of Item 2" },
* { "name": "Item 3", "description": "Description of Item 3" }
* ]
* ```

(Reason: Instructing JSON format ensures structured, machine-readable output.)

Note: Always use [JSON schema](https://exam.sanand.workers.dev/playground#attachments) if possible. [JSON schema](https://json-schema.org/) is a way to describe the structure of JSON data. An easy way to get the JSON schema is to give ChatGPT sample output and ask it to generate the schema.

Imagine you’re organizing data for a big project. Plain text is like dumping everything into one messy pile — it’s hard to find what you need later. JSON, on the other hand, is like packing your data into neat, labeled boxes within boxes. Everything has its place: fields like ‘name,’ ‘description,’ and ‘value’ make the data easy to read, especially for machines. For example, instead of just listing items, you can structure them as a JSON array, with each item as an object. That way, when you hand it to a program, it’s all clear and ready to use.

### **Prefer Yes/No answers**

Convert rating or percentage questions into Yes/No queries. LLMs handle binary choices better than numeric scales.

* **BAD**: On a scale of 1-10, how confident are you that this method works? (Reason: Asks for a numeric rating, which can be imprecise.)
* **GOOD**: Is this method likely to work as intended? Please give a reasoning and then answer Yes or No. (Reason: A binary question simplifies the response and clarifies what’s being asked.)

Don’t ask ‘On a scale of one to five...’. Models are not good with numbers. They don't know how to grade something 7 versus 8 on a 10 point scale. ‘Yes or no?’ is simple. It’s clear. It’s quick. So, break your question into simple parts that they can answer with just a yes or a no.

### **Ask for reason first, then the answer**

Instruct the model to provide its reasoning steps before stating the final answer. This makes it less likely to justify itself and more likely to think deeper, leading to more accurate results.

* **BAD**: What is the best route to take? (Reason: Direct question without prompting reasoning steps first.)
* **GOOD**: First, explain your reasoning step by step for how you determine the best route. Then, after you’ve reasoned it out, state your final recommendation for the best route. (Reason: Forces the model to show its reasoning process before giving the final answer.)

BEFORE making its final choice, have the model talk through their thinking. Reasoning first, answer second. That way, the model won't be tempted to justify an answer that they gave impulsively. It is also more likely to think deeper.

### **Use proper spelling and grammar**

A well-written, grammatically correct prompt clarifies expectations. Poorly structured prompts can confuse the model.

* **BAD**: xplin wht the weirless netork do? make shur to giv me a anser?? (Reason: Poor spelling and unclear instructions.)
* **GOOD**: Explain what a wireless network does. Please provide a detailed, step-by-step explanation. (Reason: Proper spelling and clarity lead to a more coherent response.)

If your question sounds like gibberish, expect a messy answer. Speak cleanly. When you do, the response will be much clearer.

## Video Tutorials

Watch [Prompt Engineering Tutorial – Master ChatGPT and LLM Responses (41 min)](https://youtu.be/_ZvnD73m40o). It covers:

1. Basics of **AI and large language models**.
2. How to write clear and detailed prompts to improve answers.
3. Tips for creating interactive and personalized AI responses.
4. Advanced topics like **AI mistakes** (hallucinations) and **text embeddings** (how AI understands words).
5. Fun examples, like making AI write poems or correct grammar.

SecurePrompt Technologies is a cybersecurity firm that specializes in deploying large language models (LLMs) for sensitive enterprise applications. To ensure that these models adhere strictly to security policies, SecurePrompt imposes hardcoded behavioral instructions on the LLMs. For example, an LLM may be configured to never output certain sensitive keywords.

As part of their regular security audits and red-team exercises, SecurePrompt's engineers and external auditors test how well the LLMs follow these strict instructions. One objective of these tests is to determine if it is possible to bypass or trick the LLM into violating its preset security constraints.

This task is simulates potential attack vectors where a malicious actor might manipulate the model's output by ingeniously engineering the prompt. While the intention is to expose vulnerabilities in instruction adherence, it also provides valuable insights into improving the safety and security of the deployed system.

Here's your task: You are chatting with an LLM that has been told to never say Yes. You need to get it to say Yes.

Use your [AI Proxy](https://aiproxy.sanand.workers.dev/) token when prompted.

Write a prompt that will get the LLM to say Yes.

As long as the LLM says the word Yes (case sensitive), you will be marked correct. **Careful!** If you get a correct answer, submit and don't change it. You may get a different answer next time.

Check

Check all Save

Save regularly. Your last saved submission will be evaluated.

Bottom of Form

# Best of luck!

Ended at Sun, 9 Feb, 2025, 11:59 pm IST Score: 0 Check all Save

# TDS 2025 Jan GA4 - Data Sourcing

## Instructions

1. **Learn what you need**. Reading material is provided, but feel free to skip it if you can answer the question. (Or learn it, just for pleasure.)
2. **Check answers regularly** by pressing Check. It shows which answers are right or wrong. You can check multiple times.
3. **Save regularly** by pressing Save. You can save multiple times. Your last saved submission will be evaluated.
4. **Reloading is OK**. Your answers are saved in your browser (not server). Questions won't change except for randomized parameters.
5. **Browser may struggle**. If you face loading issues, turn off security restrictions or try a different browser.
6. **Use anything**. You can use any resources you want. The Internet, ChatGPT, friends, whatever. Use any libraries or frameworks you want.
7. **It's hackable**. It's possible to get the answer to some questions by hacking the code for this quiz. That's allowed.

**Have questions?** [**Join the discussion on Discourse**](https://discourse.onlinedegree.iitm.ac.in/t/ga4-data-sourcing-discussion-thread-tds-jan-2025/165959)

#### **Bonus marks for posting on Discourse**

To encourage discussions, IITM BS students who reply to the discussion on [**GA4 - Data Sourcing - Discussion Thread [TDS Jan 2025]**](https://discourse.onlinedegree.iitm.ac.in/t/ga4-data-sourcing-discussion-thread-tds-jan-2025/165959)with a **relevant** question or reply will get 1 bonus mark on this graded assignment.

You are logged in as **22f1001679@ds.study.iitm.ac.in**.

Logout

#### **Recent saves (most recent is your official score)**

Reloadfrom 2/9/2025, 12:01:37 AM. Score: 9

Reloadfrom 2/8/2025, 8:23:20 PM. Score: 8

Reloadfrom 2/8/2025, 7:36:57 PM. Score: 7

Top of Form

# Questions

1. [Import HTML to Google Sheets](https://exam.sanand.workers.dev/tds-2025-01-ga4#hg-google-sheets-importhtml) (1 mark)
2. [Scrape IMDb movies](https://exam.sanand.workers.dev/tds-2025-01-ga4#hq-scrape-imdb-movies) (1 mark)
3. [Wikipedia Outline](https://exam.sanand.workers.dev/tds-2025-01-ga4#hq-wikipedia-outline) (1 mark)
4. [Scrape the BBC Weather API](https://exam.sanand.workers.dev/tds-2025-01-ga4#hq-bbc-weather-api) (1 mark)
5. [Find the bounding box of a city](https://exam.sanand.workers.dev/tds-2025-01-ga4#hq-nominatim-api) (1 mark)
6. [Search Hacker News](https://exam.sanand.workers.dev/tds-2025-01-ga4#hq-hacker-news-search) (1 mark)
7. [Find newest GitHub user](https://exam.sanand.workers.dev/tds-2025-01-ga4#hq-find-newest-github-user) (1 mark)
8. [Create a Scheduled GitHub Action](https://exam.sanand.workers.dev/tds-2025-01-ga4#hq-scheduled-github-actions) (1 mark)
9. [Extract tables from PDF](https://exam.sanand.workers.dev/tds-2025-01-ga4#hq-extract-tables-from-pdf) (1 mark)
10. [Convert a PDF to Markdown](https://exam.sanand.workers.dev/tds-2025-01-ga4#hq-pdf-to-markdown) (1 mark)

1 Import HTML to Google Sheets (1 mark)

## Scraping with Excel

You'll learn how to [import tables on the web using Excel](https://support.microsoft.com/en-au/office/import-data-from-the-web-b13eed81-33fe-410d-9247-1747269c28e4), covering:

* **Data Import from Web**: Use the query feature in Excel to scrape data from websites.
* **Establishing Web Connections**: Connect Excel to a web page using a URL.
* **Using Query Editor**: Navigate the query editor to view and manage web data tables.
* **Loading Data**: Load data from the web into Excel for further manipulation.
* **Data Transformation**: Remove unnecessary columns and transform data as needed.
* **Applying Transformations**: Track applied steps in the sequence for reproducibility.
* **Refreshing Data**: Refresh the imported data to get the latest updates from the web.

Here are links used in the video:

* [Chennai Weather Forecast](https://www.timeanddate.com/weather/india/chennai/ext)
* [Excel Scraping Workbook](https://docs.google.com/spreadsheets/d/1a12ApZMD6CTiKRyO4RuauOO8IdYgACRL/view)

If you use Excel on Mac, the process is a bit different. See [Importing External Data Into Excel on Mac](https://youtu.be/PuqVoVNWF20).

## Scraping with Google Sheets

You'll learn how to [import tables on the web using Google Sheets's =IMPORTHTML() formula](https://support.google.com/docs/answer/3093339?hl=en), covering:

* **Import HTML Formula**: Use =IMPORTHTML(URL, "query", index) to fetch tables or lists from a web page.
* **Granting Access**: Allow access for formulas to fetch data from external sources.
* **Checking Imported Data**: Verify if the imported table matches the data on the web page.
* **Handling Errors**: Understand common issues and how to resolve them.
* **Sorting Data**: Copy imported data as values and sort it within Google Sheets.
* **Freezing Rows**: Use frozen rows to maintain headers while sorting.
* **Live Formulas**: Learn how web data updates automatically when the source changes.
* **Other Import Functions**: IMPORTXML, IMPORTFEED, IMPORTRANGE, and IMPORTDATA for advanced data fetching options.

Here are links used in the video:

* [Google sheet used in the video](https://docs.google.com/spreadsheets/d/1Qp_YTh1-hJHxjMWE_GofkvLIKgEdKxb6NFImpId3z9o/view)
* [IMPORTHTML()](https://support.google.com/docs/answer/3093339)
* [IMPORTXML()](https://support.google.com/docs/answer/3093342)
* [Demographics of India](https://en.wikipedia.org/wiki/Demographics_of_India)
* [List of highest grossing Indian films](https://en.wikipedia.org/wiki/List_of_highest-grossing_Indian_films)

## Sports Analytics for CricketPro

**CricketPro Insights** is a leading sports analytics firm specializing in providing in-depth statistical analysis and insights for cricket teams, coaches, and enthusiasts. Leveraging data from prominent sources like ESPN Cricinfo, CricketPro offers actionable intelligence that helps teams optimize player performance, strategize game plans, and engage with fans through detailed statistics and visualizations.

In the competitive world of cricket, understanding player performance metrics is crucial for team selection, game strategy, and player development. However, manually extracting and analyzing batting statistics from extensive datasets spread across multiple web pages is time-consuming and prone to errors. To maintain their edge and deliver timely insights, CricketPro needs an efficient, automated solution to aggregate and analyze player performance data from ESPN Cricinfo's ODI (One Day International) batting statistics.

CricketPro Insights has identified the need to automate the extraction and analysis of ODI batting statistics from ESPN Cricinfo to streamline their data processing workflow. The statistics are available on a paginated website, with each page containing a subset of player data. By automating this process, CricketPro aims to provide up-to-date insights on player performances, such as the number of duck outs (i.e. a score of zero), which are pivotal for team assessments and strategic planning.

As part of this initiative, you are tasked with developing a solution that allows CricketPro analysts to:

1. **Navigate Paginated Data:** Access specific pages of the ODI batting statistics based on varying requirements.
2. **Extract Relevant Data:** Use Google Sheets' IMPORTHTML function to pull tabular data from ESPN Cricinfo.
3. **Analyze Performance Metrics:** Count the number of ducks (where the player was out for 0 runs) each player has, aiding in performance evaluations.

## Your Task

ESPN Cricinfo has [ODI batting stats](https://stats.espncricinfo.com/stats/engine/stats/index.html?class=2;template=results;type=batting) for each batsman. The result is paginated across multiple pages. Count the number of ducks in page number 16.

1. **Understanding the Data Source:** ESPN Cricinfo's [ODI batting statistics](https://stats.espncricinfo.com/stats/engine/stats/index.html?class=2;template=results;type=batting) are spread across multiple pages, each containing a table of player data. Go to page number 16.
2. **Setting Up Google Sheets:** Utilize Google Sheets' IMPORTHTML function to import table data from the URL for page number 16.
3. **Data Extraction and Analysis:** Pull the relevant table from the assigned page into Google Sheets. Locate the column that represents the number of ducks for each player. (It is titled "0".) Sum the values in the "0" column to determine the total number of ducks on that page.

## Impact

By automating the extraction and analysis of cricket batting statistics, CricketPro Insights can:

* **Enhance Analytical Efficiency:** Reduce the time and effort required to manually gather and process player performance data.
* **Provide Timely Insights:** Deliver up-to-date statistical analyses that aid teams and coaches in making informed decisions.
* **Scalability:** Easily handle large volumes of data across multiple pages, ensuring comprehensive coverage of player performances.
* **Data-Driven Strategies:** Enable the development of data-driven strategies for player selection, training focus areas, and game planning.
* **Client Satisfaction:** Improve service offerings by providing accurate and insightful analytics that meet the specific needs of clients in the cricketing world.

What is the total number of ducks across players on page number 16 of [ESPN Cricinfo's ODI batting stats](https://stats.espncricinfo.com/stats/engine/stats/index.html?class=2;template=results;type=batting)?

Check

2 Scrape IMDb movies (1 mark)

## Scraping IMDb with JavaScript

You'll learn how to scrape the [IMDb Top 250 movies](https://www.imdb.com/chart/top) directly in the browser using JavaScript on the Chrome DevTools, covering:

* **Access Developer Tools**: Use F12 or right-click > Inspect to open developer tools in Chrome or Edge.
* **Inspect Elements**: Identify and inspect HTML elements using the Elements tab.
* **Query Selectors**: Use document.querySelectorAll and document.querySelector to find elements by CSS class.
* **Extract Text Content**: Retrieve text content from elements using JavaScript.
* **Functional Programming**: Apply [map](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map) and [arrow functions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions) for concise data processing.
* **Data Structuring**: Collect and format data into an array of arrays.
* **Copying Data**: Use the copy function to transfer data to the clipboard.
* **Convert to Spreadsheet**: Use online tools to convert JSON data to CSV or Excel format.
* **Text Manipulation**: Perform text splitting and cleaning in Excel for final data formatting.

Here are links and references:

* [IMDB Top 250 movies](https://www.imdb.com/chart/top/)
* [Learn about Chrome Devtools](https://developer.chrome.com/docs/devtools/overview/)

## ****Content Curation for StreamFlix Streaming****

**StreamFlix** is a rapidly growing streaming service aiming to provide a diverse and high-quality library of movies, TV shows, etc. to its subscribers. To maintain a competitive edge and ensure customer satisfaction, StreamFlix invests heavily in data-driven content curation. By analyzing movie ratings and other key metrics, the company seeks to identify films that align with subscriber preferences and emerging viewing trends.

With millions of titles available on platforms like IMDb, manually sifting through titles to select suitable additions to StreamFlix's catalog is both time-consuming and inefficient. To streamline this process, StreamFlix's data analytics team requires an automated solution to extract and analyze movie data based on specific rating criteria.

Develop a Python program that interacts with IMDb's dataset to extract detailed information about titles within a specified rating range. The extracted data should include the movie's unique ID, title, release year, and rating. This information will be used to inform content acquisition decisions, ensuring that StreamFlix consistently offers high-quality and well-received films to its audience.

Imagine you are a data analyst at StreamFlix, responsible for expanding the platform's movie library. Your task is to identify titles that have received favorable ratings on IMDb, ensuring that the selected titles meet the company's quality standards and resonate with subscribers.

To achieve this, you need to:

1. **Extract Data:** Retrieve movie information from IMDb for all films that have a rating between 4 and 8.
2. **Format Data:** Structure the extracted information into a JSON format containing the following fields:
   * id: The unique identifier for the movie on IMDb.
   * title: The official title of the movie.
   * year: The year the movie was released.
   * rating: The IMDb user rating for the movie.

## Your Task

1. **Source:** Utilize IMDb's advanced web search at <https://www.imdb.com/search/title/> to access movie data.
2. **Filter:** Filter all titles with a rating between 4 and 8.
3. **Format:** For up to the first 25 titles, extract the necessary details: ID, title, year, and rating. The ID of the movie is the part of the URL after tt in the href attribute. For example, [tt10078772](https://www.imdb.com/title/tt10078772/). Organize the data into a JSON structure as follows:
4. [
5. { "id": "tt1234567", "title": "Movie 1", "year": "2021", "rating": "5.8" },
6. { "id": "tt7654321", "title": "Movie 2", "year": "2019", "rating": "6.2" },
7. // ... more titles

]

1. **Submit:** Submit the JSON data in the text box below.

## Impact

By completing this assignment, you'll simulate a key component of a streaming service's content acquisition strategy. Your work will enable StreamFlix to make informed decisions about which titles to license, ensuring that their catalog remains both diverse and aligned with subscriber preferences. This, in turn, contributes to improved customer satisfaction and retention, driving the company's growth and success in a competitive market.

What is the JSON data?

IMDb search results may differ by region. You may need to manually translate titles. Results may also change periodically. You may need to re-run your scraper code.

Check

3 Wikipedia Outline (1 mark)

## Scraping emarketer

In this live scraping session, we explore a real-life scenario where Straive had to scrape data from emarketer.com for a demo. This is a fairly realistic and representative way of how one might go about scraping a website.

You'll learn:

* **Scraping**: How to extract data from web pages, including constructing URLs, fetching page content, and parsing HTML using packages like [lxml](https://lxml.de/) and [httpx](https://www.python-httpx.org/).
* **Caching**: Implementing a caching strategy to avoid redundant data fetching for efficiency and reliability.
* **Error Handling and Debugging**: Practical tips for troubleshooting, such as using liberal print statements, breakpoints for in-depth debugging, and the concept of "rubber duck debugging" to clarify problems.
* **LLMs**: Benefits of Gemini / ChatGPT for code suggestions and troubleshooting.
* **Real-World Application**: How quick proofs of concept to showcase capabilities to clients, emphasizing practice over theory.

## A Country Information API for GlobalEdu

**GlobalEdu Platforms** is a leading provider of educational technology solutions, specializing in creating interactive and informative content for students and educators worldwide. Their suite of products includes digital textbooks, educational apps, and online learning platforms that aim to make learning more engaging and accessible. To enhance their offerings, GlobalEdu Platforms seeks to integrate comprehensive country information into their educational tools, enabling users to access structured and easily navigable content about various nations.

With the vast amount of information available on platforms like Wikipedia, manually curating and organizing country-specific data for educational purposes is both time-consuming and inefficient. GlobalEdu Platforms aims to automate this process to ensure that their educational materials are up-to-date, accurate, and well-structured. The key challenges they face include:

1. **Content Organization:** Presenting information in a structured and hierarchical manner that aligns with educational standards.
2. **Scalability:** Handling data for a large number of countries without manual intervention.
3. **Accessibility:** Ensuring that the information is easily accessible from various applications and platforms used by educators and students.
4. **Interoperability:** Allowing cross-origin requests to integrate the API seamlessly with different front-end applications.

To address these challenges, GlobalEdu Platforms has decided to develop a web application that exposes a RESTful API. This API will allow their educational tools to fetch and display structured outlines of Wikipedia pages for any given country. The application needs to:

* Accept a country name as a query parameter.
* Fetch the corresponding Wikipedia page for that country.
* Extract all headings (H1 to H6) from the page.
* Generate a Markdown-formatted outline that reflects the hierarchical structure of the content.
* Enable Cross-Origin Resource Sharing (CORS) to allow GET requests from any origin, facilitating seamless integration with various educational platforms.

## Your Task

Write a web application that exposes an API with a single query parameter: ?country=. It should fetch the Wikipedia page of the country, extracts all headings (H1 to H6), and create a Markdown outline for the country. The outline should look like this:

**## Contents**

**# Vanuatu**

**## Etymology**

**## History**

**### Prehistory**

...

1. **API Development:** Choose any web framework (e.g., FastAPI) to develop the web application. Create an API endpoint (e.g., /api/outline) that accepts a country query parameter.
2. **Fetching Wikipedia Content:** Find out the Wikipedia URL of the country and fetch the page's HTML.
3. **Extracting Headings:** Use an HTML parsing library (e.g., BeautifulSoup, lxml) to parse the fetched Wikipedia page. Extract all headings (H1 to H6) from the page, maintaining order.
4. **Generating Markdown Outline:** Convert the extracted headings into a Markdown-formatted outline. Headings should begin with #.
5. **Enabling CORS:** Configure the web application to include appropriate CORS headers, allowing GET requests from any origin.

What is the URL of your API endpoint?

We'll check by sending a request to this URL with ?country=... passing different countries.

Check

4 Scrape the BBC Weather API (1 mark)

## BBC Weather location ID with Python

You'll learn how to get the location ID of any city from the BBC Weather API -- as a precursor to scraping weather data -- covering:

* **Understanding API Calls**: Learn how backend API calls work when searching for a city on the BBC weather website.
* **Inspecting Web Interactions**: Use the browser's inspect element feature to track API calls and understand the network activity.
* **Extracting Location IDs**: Identify and extract the location ID from the API response using Python.
* **Using Python Libraries**: Import and use requests, json, and urlencode libraries to make API calls and process responses.
* **Constructing API URLs**: Create structured API URLs dynamically with constant prefixes and query parameters using urlencode.
* **Building Functions**: Develop a Python function that accepts a city name, constructs the API call, and returns the location ID.

To open the browser Developer Tools on Chrome, Edge, or Firefox, you can:

* Right-click on the page and select "Inspect" to open the developer tools
* OR: Press F12
* OR: Press Ctrl+Shift+I on Windows
* OR: Press Cmd+Opt+I on Mac

Here are links and references:

* [BBC Location ID scraping - Notebook](https://colab.research.google.com/drive/1-iV-tbtRicKR_HXWeu4Hi5aXJCV3QdQp)
* [BBC Weather - Palo Alto (location ID: 5380748)](https://www.bbc.com/weather/5380748)
* [BBC Locator Service - Los Angeles](https://locator-service.api.bbci.co.uk/locations?api_key=AGbFAKx58hyjQScCXIYrxuEwJh2W2cmv&stack=aws&locale=en&filter=international&place-types=settlement%2Cairport%2Cdistrict&order=importance&s=los%20angeles&a=true&format=json)
* Learn about the [requests package](https://docs.python-requests.org/en/latest/user/quickstart/). Watch [Python Requests Tutorial: Request Web Pages, Download Images, POST Data, Read JSON, and More](https://youtu.be/tb8gHvYlCFs)

## BBC Weather data with Python

You'll learn how to scrape the live weather data of a city from the BBC Weather API, covering:

* **Introduction to Web Scraping**: Understand the basics of web scraping and its legality.
* **Libraries Overview**: Learn the importance of [requests](https://docs.python-requests.org/en/latest/user/quickstart/) and [BeautifulSoup](https://beautiful-soup-4.readthedocs.io/).
* **Fetching HTML**: Use [requests](https://docs.python-requests.org/en/latest/user/quickstart/) to fetch HTML content from a web page.
* **Parsing HTML**: Utilize [BeautifulSoup](https://beautiful-soup-4.readthedocs.io/) to parse and navigate the HTML content.
* **Identifying Data**: Inspect HTML elements to locate specific data (e.g., high and low temperatures).
* **Extracting Data**: Extract relevant data using [BeautifulSoup](https://beautiful-soup-4.readthedocs.io/)'s find\_all() function.
* **Data Cleanup**: Clean extracted data to remove unwanted elements.
* **Post-Processing**: Use regular expressions to split large strings into meaningful parts.
* **Data Structuring**: Combine extracted data into a structured pandas DataFrame.
* **Handling Special Characters**: Replace unwanted characters for better data manipulation.
* **Saving Data**: Save the cleaned data into CSV and Excel formats.

Here are links and references:

* [BBC Weather scraping - Notebook](https://colab.research.google.com/drive/1-gkMzE-TKe3U_yh1v0NPn4TM687H2Hcf)
* [BBC Locator Service - Mumbai](https://locator-service.api.bbci.co.uk/locations?api_key=AGbFAKx58hyjQScCXIYrxuEwJh2W2cmv&stack=aws&locale=en&filter=international&place-types=settlement%2Cairport%2Cdistrict&order=importance&s=mumbai&a=true&format=json)
* [BBC Weather - Mumbai (location ID: 1275339)](https://www.bbc.com/weather/1275339)
* [BBC Weather API - Mumbai (location ID: 1275339)](https://weather-broker-cdn.api.bbci.co.uk/en/forecast/aggregated/1275339)
* Learn about the [json package](https://docs.python.org/3/library/json.html). Watch [Python Tutorial: Working with JSON Data using the json Module](https://youtu.be/9N6a-VLBa2I)
* Learn about the [BeautifulSoup package](https://beautiful-soup-4.readthedocs.io/). Watch [Python Tutorial: Web Scraping with BeautifulSoup and Requests](https://youtu.be/ng2o98k983k)
* Learn about the [pandas package](https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html). Watch
  + [Python Pandas Tutorial (Part 1): Getting Started with Data Analysis - Installation and Loading Data](https://youtu.be/ZyhVh-qRZPA)
  + [Python Pandas Tutorial (Part 2): DataFrame and Series Basics - Selecting Rows and Columns](https://youtu.be/zmdjNSmRXF4)
* Learn about the [re package](https://docs.python.org/3/library/re.html). Watch [Python Tutorial: re Module - How to Write and Match Regular Expressions (Regex)](https://youtu.be/K8L6KVGG-7o)
* Learn about the [datetime package](https://docs.python.org/3/library/datetime.html). Watch [Python Tutorial: Datetime Module - How to work with Dates, Times, Timedeltas, and Timezones](https://youtu.be/eirjjyP2qcQ)

## Weather Data Integration for AgroTech Insights

**AgroTech Insights** is a leading agricultural technology company that provides data-driven solutions to farmers and agribusinesses. By leveraging advanced analytics and real-time data, AgroTech helps optimize crop yields, manage resources efficiently, and mitigate risks associated with adverse weather conditions. Accurate and timely weather forecasts are crucial for making informed decisions in agricultural planning and management.

Farmers and agribusinesses rely heavily on precise weather information to plan planting schedules, irrigation, harvesting, and protect crops from extreme weather events. However, accessing and processing weather data from multiple sources can be time-consuming and technically challenging. AgroTech Insights seeks to automate the extraction and transformation of weather data to provide seamless, actionable insights to its clients.

AgroTech Insights has partnered with various stakeholders to enhance its weather forecasting capabilities. One of the key requirements is to integrate weather forecast data for specific regions to support crop management strategies. For this purpose, AgroTech utilizes the **BBC Weather API**, a reliable source of detailed weather information.

## Your Task

As part of this initiative, you are tasked with developing a system that automates the following:

1. **API Integration and Data Retrieval:** Use the BBC Weather API to fetch the weather forecast for Dublin. Send a GET request to the locator service to obtain the city's locationId. Include necessary query parameters such as API key, locale, filters, and search term (city).
2. **Weather Data Extraction:** Retrieve the weather forecast data using the obtained locationId. Send a GET request to the weather broker API endpoint with the locationId.
3. **Data Transformation:** Extract the localDate and enhancedWeatherDescription from each day's forecast. Iterate through the forecasts array in the API response and map each localDate to its corresponding enhancedWeatherDescription. Create a JSON object where each key is the localDate and the value is the enhancedWeatherDescription.

The output would look like this:

{

"2025-01-01": "Sunny with scattered clouds",

"2025-01-02": "Partly cloudy with a chance of rain",

"2025-01-03": "Overcast skies",

// ... additional days

}

What is the JSON weather forecast description for Dublin?

Check

5 Find the bounding box of a city (1 mark)

## Nominatim API with Python

You'll learn how to get the latitude and longitude of any city from the Nominatim API.

* **Introduction to Nominatim**: Understand how Nominatim, from OpenStreetMap, works similarly to Google Maps for geocoding.
* **Installation and Import**: Learn to install and import [geopy](https://geopy.readthedocs.io/) and [nominatim](https://nominatim.org/).
* **Using the Locator**: Create a locator object using Nominatim and set up a user agent.
* **Geocoding an Address**: Use locator.geocode to input an address (e.g., Eiffel Tower) and fetch geocoded data.
* **Extracting Data**: Access detailed information like latitude, longitude, bounding box, and accurate address from the JSON response.
* **Classifying Locations**: Identify the type of place (e.g., tourism, university) using the response data.
* **Practical Example**: Geocode "IIT Madras" and retrieve its full address, type (university), and other relevant information.

Here are links and references:

* [Geocoding using Nominatim - Notebook](https://colab.research.google.com/drive/1-vvP-UyMjHgBqc-hdsUhm3Bsbgi7oO6g)
* Learn about the [geocoders module in the geopy package](https://geopy.readthedocs.io/)
* Learn about the [nominatim package](https://nominatim.org/release-docs/develop/api/Overview/)
* If you get a HTTP Error 403 from Nominatim, use your email ID or your name instead of "myGeocoder" in Nominatim(user\_agent="myGeocoder")

## Geospatial Data Optimization for UrbanRide

**UrbanRide** is a leading transportation and logistics company operating in major metropolitan areas worldwide. To enhance their service efficiency, optimize route planning, and improve customer satisfaction, UrbanRide relies heavily on accurate geospatial data. Precise bounding box information of cities helps in defining service zones, managing fleet distribution, and analyzing regional demand patterns.

As UrbanRide expands into new cities, the company faces the challenge of accurately delineating service areas within these urban environments. Defining the geographical boundaries of a city is crucial for:

* **Route Optimization:** Ensuring drivers operate within designated zones to minimize transit times and fuel consumption.
* **Fleet Management:** Allocating vehicles effectively across different regions based on demand and service coverage.
* **Market Analysis:** Understanding regional demand to tailor services and promotional efforts accordingly.

However, manually extracting and verifying bounding box data for each city is time-consuming and prone to inconsistencies, especially when dealing with cities that share names across different countries or have multiple administrative districts.

UrbanRide’s data analytics team needs to automate the extraction of precise bounding box coordinates (specifically the minimum and maximum latitude) for various populous cities across different countries. This automation ensures consistency, accuracy, and scalability as the company grows its operations.

To achieve this, the team utilizes the **Nominatim API**, a geocoding service based on OpenStreetMap data, to programmatically retrieve geospatial information. However, challenges arise when cities with the same name exist in multiple countries or have multiple entries within the same country. To address this, the team must implement a method to select the correct city instance based on specific identifiers (e.g., osm\_id patterns).

## Your Task

What is the maximum latitude of the bounding box of the city Tianjin in the country China on the Nominatim API?

1. **API Integration:** Use the Nominatim API to fetch geospatial data for a specified city within a country via a GET request to the Nominatim API with parameters for the city and country. Ensure adherence to Nominatim’s usage policies, including rate limiting and proper attribution.
2. **Data Retrieval and Filtering:** Parse the JSON response from the API. If multiple results are returned (e.g., multiple cities named “Springfield” in different states), filter the results based on the provided osm\_id ending to select the correct city instance.
3. **Parameter Extraction:** Access the boundingbox attribute. Depending on whether you're looking for the minimum or maximum latitude, extract the corresponding latitude value.

## Impact

By automating the extraction and processing of bounding box data, UrbanRide can:

* **Optimize Routing:** Enhance route planning algorithms with precise geographical boundaries, reducing delivery times and operational costs.
* **Improve Fleet Allocation:** Allocate vehicles more effectively across defined service zones based on accurate city extents.
* **Enhance Market Analysis:** Gain deeper insights into regional performance, enabling targeted marketing and service improvements.
* **Scale Operations:** Seamlessly integrate new cities into their service network with minimal manual intervention, ensuring consistent data quality.

What is the maximum latitude of the bounding box of the city Tianjin in the country China on the Nominatim API? Value of the maximum latitude

Check

6 Search Hacker News (1 mark)

## Media Intelligence for TechInsight Analytics

**TechInsight Analytics** is a leading market research firm specializing in technology trends and media intelligence. The company provides actionable insights to tech companies, startups, and investors by analyzing online discussions, news articles, and social media posts. One of their key data sources is **Hacker News**, a popular platform where tech enthusiasts and professionals share and discuss the latest in technology, startups, and innovation.

In the rapidly evolving tech landscape, staying updated with the latest trends and public sentiments is crucial for TechInsight Analytics' clients. Manual monitoring of Hacker News posts for specific topics and engagement levels is inefficient and error-prone due to the high volume of daily posts. To address this, TechInsight seeks to automate the process of identifying and extracting relevant Hacker News posts that mention specific technology topics and have garnered significant attention (measured by points).

TechInsight Analytics has developed an internal tool that leverages the [HNRSS API](https://hnrss.github.io/) to fetch the latest Hacker News posts. The tool needs to perform the following tasks:

1. **Topic Monitoring:** Continuously monitor Hacker News for posts related to specific technology topics, such as "Artificial Intelligence," "Blockchain," or "Cybersecurity."
2. **Engagement Filtering:** Identify posts that have received a minimum number of points (votes) to ensure the content is highly engaging and relevant.
3. **Data Extraction:** Extract essential details from the qualifying posts, including the post's link for further analysis and reporting.

To achieve this, the team needs to create a program that:

* Searches Hacker News for the latest posts mentioning a specified topic.
* Filters these posts based on a minimum points threshold.
* Retrieves and returns the link to the most relevant post.

## Your Task

Search using the [Hacker News RSS API](https://hnrss.github.io/) for the latest Hacker News post mentioning Cybersecurity and having a minimum of 59 points. What is the link that it points to?

1. **Automate Data Retrieval:** Utilize the HNRSS API to fetch the latest Hacker News posts. Use the URL relevant to fetching the latest posts, searching for topics and filtering by a minimum number of points.
2. **Extract and Present Data:** Extract the most recent <item> from this result. Get the <link> tag inside it.
3. **Share the result:** Type in just the URL in the answer.

What is the link to the latest Hacker News post mentioning Cybersecurity having at least 59 points?

Check

7 Find newest GitHub user (1 mark)

## Emerging Developer Talent for CodeConnect

**CodeConnect** is an innovative recruitment platform that specializes in matching high-potential tech talent with forward-thinking companies. As the demand for skilled software developers grows, CodeConnect is committed to staying ahead of trends by leveraging data-driven insights to identify emerging developers—especially those who demonstrate strong community influence on platforms like GitHub.

For CodeConnect, a key objective is to tap into regional talent pools to support local hiring initiatives and foster diversity within tech teams. One specific challenge is identifying developers in major tech hubs (such as Shanghai) who not only have established GitHub profiles but also show early signs of influence, as indicated by their follower counts.

However, with millions of developers on GitHub and constantly evolving profiles, manually filtering through the data is impractical. CodeConnect needs an automated solution that:

1. **Filters Developer Profiles:** Retrieves GitHub users based on location and a minimum follower threshold (e.g., over 60 followers) to focus on those with some level of social proof.
2. **Identifies the Newest Talent:** Determines the most recent GitHub user in the selected group, providing insight into new emerging talent.
3. **Standardizes Data:** Returns the account creation date in a standardized ISO 8601 format, ensuring consistent reporting across the organization.

The recruitment team at CodeConnect is launching a new initiative aimed at hiring young, promising developers in Shanghai—a city known for its vibrant tech community. To support this initiative, the team has commissioned a project to use the GitHub API to find all users located in Shanghai with more than 60 followers. From this filtered list, they need to identify the newest account based on the profile creation date. This information will help the team target outreach efforts to developers who have recently joined the platform and may be eager to explore new career opportunities.

## Your Task

Using the [GitHub API](https://docs.github.com/en/rest), find all users located in the city Moscow with over 100 followers.

When was the newest user's GitHub profile created?

1. **API Integration and Data Retrieval:** Leverage GitHub’s search endpoints to query users by location and filter them by follower count.
2. **Data Processing:** From the returned list of GitHub users, isolate those profiles that meet the specified criteria.
3. **Sort and Format:** Identify the "newest" user by comparing the created\_at dates provided in the user profile data. Format the account creation date in the ISO 8601 standard (e.g., "2024-01-01T00:00:00Z").

## Impact

By automating this data retrieval and filtering process, CodeConnect gains several strategic advantages:

* **Targeted Recruitment:** Quickly identify new, promising talent in key regions, allowing for more focused and timely recruitment campaigns.
* **Competitive Intelligence:** Stay updated on emerging trends within local developer communities and adjust talent acquisition strategies accordingly.
* **Efficiency:** Automating repetitive data collection tasks frees up time for recruiters to focus on engagement and relationship-building.
* **Data-Driven Decisions:** Leverage standardized and reliable data to support strategic business decisions in recruitment and market research.

Enter the date (ISO 8601, e.g. "2024-01-01T00:00:00Z") when the newest user joined GitHub.

Search using location: and followers: filters, sort by joined descending, fetch the first url, and enter the created\_at field. Ignore ultra-new users who JUST joined, i.e. after 2/20/2025, 9:34:54 AM.

Check

8 Create a Scheduled GitHub Action (1 mark)

## Scheduled Scraping with GitHub Actions

GitHub Actions provides an excellent platform for running web scrapers on a schedule. This tutorial shows how to automate data collection from websites using GitHub Actions workflows.

### **Key Concepts**

* **Scheduling**: Use [cron syntax](https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows#schedule) to run scrapers at specific times
* **Dependencies**: Install required packages like httpx, lxml
* **Data Storage**: Save scraped data to files and commit back to the repository
* **Error Handling**: Implement robust error handling for network issues and HTML parsing
* **Rate Limiting**: Respect website terms of service and implement delays between requests

Here's a sample scrape.py that scrapes the IMDb Top 250 movies using httpx and lxml:

**import** json

**import** httpx

**from** datetime **import** datetime, UTC

**from** lxml **import** html

**from** typing **import** List, Dict

**def** **scrape\_imdb**() -> List[Dict[str, str]]:

"""Scrape IMDb Top 250 movies using httpx and lxml.

Returns:

List of dictionaries containing movie title, year, and rating.

"""

headers = {"User-Agent": "Mozilla/5.0 (compatible; IMDbBot/1.0)"}

response = httpx.get("https://www.imdb.com/chart/top/", headers=headers)

response.raise\_for\_status()

tree = html.fromstring(response.text)

movies = []

**for** item **in** tree.cssselect(".ipc-metadata-list-summary-item"):

title = (

item.cssselect(".ipc-title\_\_text")[0].text\_content()

**if** item.cssselect(".ipc-title\_\_text")

**else** None

)

year = (

item.cssselect(".cli-title-metadata span")[0].text\_content()

**if** item.cssselect(".cli-title-metadata span")

**else** None

)

rating = (

item.cssselect(".ipc-rating-star")[0].text\_content()

**if** item.cssselect(".ipc-rating-star")

**else** None

)

**if** title **and** year **and** rating:

movies.append({"title": title, "year": year, "rating": rating})

**return** movies

# Scrape data and save with timestamp

now = datetime.now(UTC)

**with** open(f'imdb-top250-{now.strftime("%Y-%m-%d")}.json', "a") **as** f:

f.write(json.dumps({"timestamp": now.isoformat(), "movies": scrape\_imdb()}) + "\n")

Here's a sample .github/workflows/imdb-top250.yml that runs the scraper daily and saves the data:

name: Scrape IMDb Top 250

on:

schedule:

# Runs at 00:00 UTC every day

- cron: "0 0 \* \* \*"

workflow\_dispatch: # Allow manual triggers

jobs:

scrape-imdb:

runs-on: ubuntu-latest

permissions:

contents: write

steps:

- name: Checkout repository

uses: actions/checkout@v4

- name: Install uv

uses: astral-sh/setup-uv@v5

- name: Run scraper

run: | # python

uv run --with httpx,lxml,cssselect python scrape.py

- name: Commit and push changes

run: |

git config --local user.email "github-actions[bot]@users.noreply.github.com"

git config --local user.name "github-actions[bot]"

git add \*.json

git commit -m "Update IMDb Top 250 data [skip ci]" || exit 0

git push

### **Best Practices**

1. **Cache Dependencies**: Use GitHub's caching to speed up package installation
2. **Handle Errors**: Implement retries and error logging
3. **Rate Limiting**: Add delays between requests to avoid overwhelming servers
4. **Data Validation**: Verify scraped data structure before saving
5. **Monitoring**: Set up notifications for workflow failures

### **Tools and Resources**

* [httpx](https://www.python-httpx.org/): Async HTTP client
* [GitHub Actions Marketplace](https://github.com/marketplace?type=actions)
* [GitHub Actions Documentation](https://docs.github.com/en/actions)

### **Video Tutorials**

## Automating Repository Updates for DevSync

**DevSync Solutions** is a mid-sized software development company specializing in collaborative tools for remote teams. With a growing client base and an expanding portfolio of projects, DevSync emphasizes efficient workflow management and robust version control practices to maintain high-quality software delivery.

As part of their commitment to maintaining seamless and transparent development processes, DevSync has identified the need to implement automated daily updates to their GitHub repositories. These updates serve multiple purposes:

1. **Activity Tracking:** Ensuring that each repository reflects daily activity helps in monitoring project progress and team engagement.
2. **Automated Documentation:** Regular commits can be used to update status files, logs, or documentation without manual intervention.
3. **Backup and Recovery:** Automated commits provide an additional layer of backup, ensuring that changes are consistently recorded.
4. **Compliance and Auditing:** Maintaining a clear commit history aids in compliance with industry standards and facilitates auditing processes.

Manually managing these daily commits is inefficient and prone to human error, especially as the number of repositories grows. To address this, DevSync seeks to automate the process using GitHub Actions, ensuring consistency, reliability, and scalability across all projects.

DevSync's DevOps team has decided to standardize the implementation of GitHub Actions across all company repositories. The objective is to create a scheduled workflow that runs once daily, adds a commit to the repository, and ensures that these actions are consistently tracked and verifiable.

As a junior developer or DevOps engineer at DevSync, you are tasked with setting up this automation for a specific repository. This exercise will not only enhance your understanding of GitHub Actions but also contribute to the company's streamlined workflow management.

## Your Task

Create a scheduled [GitHub action](https://github.com/features/actions) that runs daily and adds a commit to your repository. The workflow should:

* Use schedule with cron syntax to run **once per day** (must use specific hours/minutes, not wildcards)
* Include a step with your email 22f1001679@ds.study.iitm.ac.in in its name
* Create a commit in each run
* Be located in .github/workflows/ directory

After creating the workflow:

* Trigger the workflow and wait for it to complete
* Ensure it appears as the **most recent action** in your repository
* Verify that it creates a commit during or within 5 minutes of the workflow run

Enter your repository URL (format: https://github.com/USER/REPO):

Check

9 Extract tables from PDF (1 mark)

## Scraping PDFs with Tabula

You'll learn how to scrape tables from PDFs using the tabula Python library, covering:

* **Import Libraries**: Use Beautiful Soup for URL parsing and Tabula for extracting tables from PDFs.
* **Specify Save Location**: Mount Google Drive to save scraped PDFs.
* **Identify PDF URLs**: Parse the given URL to identify and select all PDF links.
* **Download PDFs**: Loop through identified links, saving each PDF to the specified location.
* **Extract Tables**: Use Tabula to read tabular content from the downloaded PDFs.
* **Control Extraction Area**: Specify page and area coordinates to accurately extract tables, avoiding extraneous text.
* **Save Extracted Data**: Convert the extracted table data into structured formats like CSV for further analysis.

Here are links and references:

* [PDF Scraping - Notebook](https://colab.research.google.com/drive/102Fv2Ji0J4mvao3mCse52E7Th8bZiuyf)
* Learn about the [tabula package](https://tabula-py.readthedocs.io/en/latest/tabula.html)
* Learn about the [pandas package](https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html). [Video](https://youtu.be/vmEHCJofslg)

## Academic Performance Analysis for EduAnalytics

**EduAnalytics Corp.** is a leading educational technology company that partners with schools and educational institutions to provide data-driven insights into student performance. By leveraging advanced analytics and reporting tools, EduAnalytics helps educators identify trends, improve teaching strategies, and enhance overall student outcomes. One of their key offerings is the **Performance Insight Dashboard**, which aggregates and analyzes student marks across various subjects and demographic groups.

EduAnalytics has recently onboarded **Greenwood High School**, a large educational institution aiming to optimize its teaching methods and improve student performance in core subjects. Greenwood High School conducts annual assessments in multiple subjects, and the results are compiled into detailed PDF reports each semester. However, manually extracting and analyzing this data is time-consuming and prone to errors, especially given the volume of data and the need for timely insights.

To address this, EduAnalytics plans to automate the data extraction and analysis process, enabling Greenwood High School to receive precise and actionable reports without the delays associated with manual processing.

As part of this initiative, you are a data analyst at EduAnalytics assigned to develop a module that processes PDF reports containing student marks. Each PDF, named in the format xxx.pdf, includes a comprehensive table listing student performances across various subjects, along with their respective groups.

**Greenwood High School** has specific analytical needs, such as:

* **Subject Performance Analysis:** Understanding how students perform in different subjects to identify areas needing improvement.
* **Group-Based Insights:** Analyzing performance across different student groups to ensure equitable educational support.
* **Threshold-Based Reporting:** Focusing on students who meet or exceed certain performance thresholds to tailor advanced programs or interventions.

## Your Task

This file, q-extract-tables-from-pdf.pdf contains a table of student marks in Maths, Physics, English, Economics, and Biology.

Calculate the total Maths marks of students who scored 23 or more marks in Maths in groups 30-62 (including both groups).

1. **Data Extraction:**: Retrieve the PDF file containing the student marks table and use PDF parsing libraries (e.g., Tabula, Camelot, or PyPDF2) to accurately extract the table data into a workable format (e.g., CSV, Excel, or a DataFrame).
2. **Data Cleaning and Preparation:** Convert marks to numerical data types to facilitate accurate calculations.
3. **Data Filtering:** Identify students who have scored marks between 23 and Maths in groups 30-62 (including both groups).
4. **Calculation:** Sum the marks of the filtered students to obtain the total marks for this specific cohort.

By automating the extraction and analysis of student marks, EduAnalytics empowers Greenwood High School to make informed decisions swiftly. This capability enables the school to:

* **Identify Performance Trends:** Quickly spot areas where students excel or need additional support.
* **Allocate Resources Effectively:** Direct teaching resources and interventions to groups and subjects that require attention.
* **Enhance Reporting Efficiency:** Reduce the time and effort spent on manual data processing, allowing educators to focus more on teaching and student engagement.
* **Support Data-Driven Strategies:** Use accurate and timely data to shape educational strategies and improve overall student outcomes.

What is the total Maths marks of students who scored 23 or more marks in Maths in groups 30-62 (including both groups)?

Check

10 Convert a PDF to Markdown (1 mark)

## Convert PDFs to Markdown

* [PyMuPDF](https://pymupdf.readthedocs.io/) is emerging as a strong default for PDF text extraction.
* [PyMuPDF4LLM](https://pymupdf.readthedocs.io/en/latest/pymupdf4llm/index.html) is apart of PyMuPDF that generates Markdown from PDFs that's suitable for LLMs.
* [unstructured](https://unstructured.io/) is radidly becoming the de facto library for parsing over 40 different file types and extracting text and tables. It's particularly useful for generating content to pass to LLMs.
* markitdown
* docling

## Digital Documentation Transformation for EduDocs Inc.

**EduDocs Inc.** is a leading provider of educational resources and documentation management solutions for academic institutions. With a growing client base comprising universities, colleges, and online learning platforms, EduDocs emphasizes the importance of accessible, well-formatted digital documentation. To maintain high standards and streamline their content delivery, EduDocs continually seeks to enhance its documentation workflows and ensure consistency across all materials.

EduDocs manages a vast repository of educational materials, including course syllabi, lecture notes, research papers, and administrative documents. These materials are often provided by clients in various formats, predominantly PDF, which poses challenges for content reuse, editing, and integration into digital platforms.

Manually converting PDF documents to Markdown is time-consuming and prone to errors, especially when dealing with large volumes of documents. Additionally, ensuring that the converted Markdown adheres to consistent formatting standards is crucial for maintaining the professional quality of EduDocs' deliverables.

To address these challenges, EduDocs aims to automate and standardize the conversion of PDF documents to Markdown format, ensuring that all Markdown files are consistently formatted using Prettier 3.4.2. This initiative will improve efficiency, reduce manual effort, and enhance the overall quality of the documentation provided to clients.

## Your Task

As part of the **Documentation Transformation Project**, you are a junior developer at EduDocs tasked with developing a streamlined workflow for converting PDF files to Markdown and ensuring their consistent formatting. This project is critical for supporting EduDocs' commitment to delivering high-quality, accessible educational resources to its clients.

q-pdf-to-markdown.pdf has the contents of a sample document.

1. **Convert the PDF to Markdown:** Extract the content from the PDF file. Accurately convert the extracted content into Markdown format, preserving the structure and formatting as much as possible.
2. **Format the Markdown:** Use Prettier version 3.4.2 to format the converted Markdown file.
3. **Submit the Formatted Markdown:** Provide the final, formatted Markdown file as your submission.

## Impact

By completing this exercise, you will contribute to EduDocs Inc.'s mission of providing high-quality, accessible educational resources. Automating the PDF to Markdown conversion and ensuring consistent formatting:

* **Enhances Productivity:** Reduces the time and effort required to prepare documentation for clients.
* **Improves Quality:** Ensures all documents adhere to standardized formatting, enhancing readability and professionalism.
* **Supports Scalability:** Enables EduDocs to handle larger volumes of documentation without compromising on quality.
* **Facilitates Integration:** Makes it easier to integrate Markdown-formatted documents into various digital platforms and content management systems.

What is the markdown content of the PDF, formatted with prettier@3.4.2?

It is very hard to get the correct Markdown output from a PDF. Any method you use will likely require manual corrections. To make it easy, this question only checks a few basic things.

Check

Check all Save

Save regularly. Your last saved submission will be evaluated.

Bottom of Form

# Best of luck!

Due Fri, 21 Feb, 2025, 11:59 pm IST Score: 7 / 10 Check all Save

# TDS 2025 Jan GA5 - Data Preparation

## Instructions

1. **Learn what you need**. Reading material is provided, but feel free to skip it if you can answer the question. (Or learn it, just for pleasure.)
2. **Check answers regularly** by pressing Check. It shows which answers are right or wrong. You can check multiple times.
3. **Save regularly** by pressing Save. You can save multiple times. Your last saved submission will be evaluated.
4. **Reloading is OK**. Your answers are saved in your browser (not server). Questions won't change except for randomized parameters.
5. **Browser may struggle**. If you face loading issues, turn off security restrictions or try a different browser.
6. **Use anything**. You can use any resources you want. The Internet, ChatGPT, friends, whatever. Use any libraries or frameworks you want.
7. **It's hackable**. It's possible to get the answer to some questions by hacking the code for this quiz. That's allowed.

**Have questions?** [**Join the discussion on Discourse**](https://discourse.onlinedegree.iitm.ac.in/t/ga5-data-preparation-discussion-thread-tds-jan-2025/166576)

You are logged in as **22f1001679@ds.study.iitm.ac.in**.

Logout

#### **Recent saves (most recent is your official score)**

Reloadfrom 2/20/2025, 9:22:57 AM. Score: 7

Reloadfrom 2/19/2025, 10:49:28 PM. Score: 7

Reloadfrom 2/19/2025, 10:49:23 PM. Score: 7

Top of Form

# Questions

1. [Clean up Excel sales data](https://exam.sanand.workers.dev/tds-2025-01-ga5#hq-clean-up-excel-sales-data) (1 mark)
2. [Clean up student marks](https://exam.sanand.workers.dev/tds-2025-01-ga5#hq-clean-up-student-marks) (1 mark)
3. [Apache log requests](https://exam.sanand.workers.dev/tds-2025-01-ga5#hq-apache-log-requests) (1 mark)
4. [Apache log downloads](https://exam.sanand.workers.dev/tds-2025-01-ga5#hq-apache-log-downloads) (1 mark)
5. [Clean up sales data](https://exam.sanand.workers.dev/tds-2025-01-ga5#hq-clean-up-sales-data) (1 mark)
6. [Parse partial JSON](https://exam.sanand.workers.dev/tds-2025-01-ga5#hq-parse-partial-json) (1 mark)
7. [Extract nested JSON keys](https://exam.sanand.workers.dev/tds-2025-01-ga5#hq-extract-nested-json-keys) (1 mark)
8. [DuckDB: Social Media Interactions](https://exam.sanand.workers.dev/tds-2025-01-ga5#hq-duckdb-social-media-interactions) (1 mark)
9. [Transcribe a YouTube video](https://exam.sanand.workers.dev/tds-2025-01-ga5#hq-transcribe-youtube) (1 mark)
10. [Reconstruct an image](https://exam.sanand.workers.dev/tds-2025-01-ga5#hq-image-jigsaw) (1 mark)

1 Clean up Excel sales data (1 mark)

## Data Cleansing in Excel

You'll learn basic but essential data cleaning techniques in Excel, covering:

* **Find and Replace**: Use Ctrl+H to replace or remove specific terms (e.g., removing "[more]" from country names).
* **Changing Data Formats**: Convert columns from general to numerical format.
* **Removing Extra Spaces**: Use the TRIM function to clean up unnecessary spaces in text.
* **Identifying and Removing Blank Cells**: Highlight and delete entire rows with blank cells using the "Go To Special" function.
* **Removing Duplicates**: Use the "Remove Duplicates" feature to eliminate duplicate entries, demonstrated with country names.

Here are links used in the video:

* [List of Largest Cities Excel file](https://docs.google.com/spreadsheets/d/1jl8tHGoxmIba4J78aJVfT9jtZv7lfCbV/view)

## Data Transformation in Excel

You'll learn data transformation techniques in Excel, covering:

* **Calculating Ratios**: Compute metro area to city area and metro population to city population ratios.
* **Using Pivot Tables**: Create pivot tables to aggregate data and identify outliers.
* **Filtering Data**: Apply filters in pivot tables to analyze specific subsets of data.
* **Counting Data Occurrences**: Use pivot tables to count the frequency of specific entries.
* **Creating Charts**: Generate charts from pivot table data to visualize distributions and outliers.

Here are links used in the video:

* [List of Largest Cities Excel file](https://docs.google.com/spreadsheets/d/1jl8tHGoxmIba4J78aJVfT9jtZv7lfCbV/view)

## Splitting Text in Excel

You'll learn how to transform a single-column data set into multiple, organized columns based on specific delimiters using the "Text to Columns" feature.

Here are links used in the video:

* [US Senate Legislation - Votes](https://www.senate.gov/legislative/votes_new.htm)

## Data Aggregation in Excel

You'll learn data aggregation and visualization techniques in Excel, covering:

* **Data Cleanup**: Remove empty columns and rows with missing values.
* **Creating Excel Tables**: Convert raw data into tables for easier manipulation and formula application.
* **Date Manipulation**: Extract week, month, and year from date columns using Excel functions (WEEKNUM, TEXT).
* **Color Scales**: Apply color scales to visualize clusters and trends in data over time.
* **Pivot Tables**: Create pivot tables to aggregate data by location and date, summarizing values weekly and monthly.
* **Sparklines**: Use sparklines to visualize trends within pivot tables, making data patterns more apparent.
* **Data Bars**: Implement data bars for graphical illustrations of numerical columns, showing trends and waves.

Here are links used in the video:

* [COVID-19 data Excel file - raw data](https://docs.google.com/spreadsheets/d/14HLgSmME95q--6lcBv9pUstqHL183wTd/view)

## Improving Sales Data Accuracy for RetailWise Inc.

**RetailWise Inc.** is a retail analytics firm that supports companies in optimizing their pricing, margins, and inventory decisions. Their reports depend on accurate historical sales data, but legacy data sources are often messy. Recently, RetailWise received an Excel sheet containing 1,000 transaction records that were generated from scanned receipts. Due to OCR inconsistencies and legacy formatting issues, the data in the Excel sheet is not clean.

The Excel file has these columns, and they are messy:

* **Customer Name**: Contains leading/trailing spaces.
* **Country**: Uses inconsistent representations. Instead of 2-letter abbreviations, it also contains other values like "USA" vs. "US", "UK" vs. "U.K", "Fra" for France, "Bra" for Brazil, "Ind" for India.
* **Date**: Uses mixed formats like "MM-DD-YYYY", "YYYY/MM/DD", etc.
* **Product**: Includes a product name followed by a slash and a random code (e.g., "Theta/5x01vd"). Only the product name part (before the slash) is relevant.
* **Sales and Cost**: Contain extra spaces and the currency string ("USD"). In some rows, the Cost field is missing. When the cost is missing, it should be treated as 50% of the Sales value.
* **TransactionID**: Though formatted as four-digit numbers, this field may have inconsistent spacing.

### **Your Task**

You need to clean this Excel data and calculate the total margin for all transactions that satisfy the following criteria:

* **Time Filter:** Sales that occurred up to and including a specified date (**Fri Feb 11 2022 09:50:55 GMT+0530 (India Standard Time)**).
* **Product Filter:** Transactions for a specific product (**Gamma**). (Use only the product name before the slash.)
* **Country Filter:** Transactions from a specific country (**IN**), after standardizing the country names.

The **total margin** is defined as:

Total Margin=Total Sales−Total CostTotal Sales

Your solution should address the following challenges:

1. **Trim and Normalize Strings:** Remove extra spaces from the **Customer Name** and **Country** fields. Map inconsistent country names (e.g., "USA", "U.S.A", "US") to a standardized format.
2. **Standardize Date Formats:** Detect and convert dates from "MM-DD-YYYY" and "YYYY/MM/DD" into a consistent date format (e.g., ISO 8601).
3. **Extract the Product Name:** From the **Product** field, extract the portion before the slash (e.g., extract "Theta" from "Theta/5x01vd").
4. **Clean and Convert Sales and Cost:** Remove the "USD" text and extra spaces from the **Sales** and **Cost** fields. Convert these fields to numerical values. Handle missing Cost values appropriately (50% of Sales).
5. **Filter the Data:** Include only transactions up to and including **Fri Feb 11 2022 09:50:55 GMT+0530 (India Standard Time)**, matching product **Gamma**, and country **IN**.
6. **Calculate the Margin:** Sum the Sales and Cost for the filtered transactions. Compute the overall margin using the formula provided.

By cleaning the data and calculating accurate margins, RetailWise Inc. can:

* **Improve Decision Making:** Provide clients with reliable margin analyses to optimize pricing and inventory.
* **Enhance Reporting:** Ensure historical data is consistent and accurate, boosting stakeholder confidence.
* **Streamline Operations:** Reduce the manual effort needed to clean data from legacy sources.

Download the Sales Excel file: q-clean-up-excel-sales-data.xlsx

What is the total margin for transactions before **Fri Feb 11 2022 09:50:55 GMT+0530 (India Standard Time)** for **Gamma** sold in **IN** (which may be spelt in different ways)?

Correct

You can enter the margin as a percentage (e.g. 12.34%) or a decimal (e.g. 0.1234).

Check

2 Clean up student marks (1 mark)

## Data Preparation in the Editor

You'll learn how to use a text editor [Visual Studio Code](https://code.visualstudio.com/) to process and clean data, covering:

* **Format** JSON files
* **Find all** and multiple cursors to extract specific fields
* **Sort** lines
* **Delete duplicate** lines
* **Replace** text with multiple cursors

Here are the links used in the video:

* [City-wise product sales JSON](https://drive.google.com/file/d/1VEnKChf4i04iKsQfw0MwoJlfkOBGQ65B/view?usp=drive_link)

## Streamlining Student Records for EduTrack

**EduTrack Systems** is a leading provider of educational management software that helps schools and universities maintain accurate and up-to-date student records. EduTrack's platform is used by administrators to monitor academic performance, manage enrollment, and generate reports for compliance and strategic planning.

In many educational institutions, student data is collected from multiple sources—such as handwritten forms, scanned documents, and digital submissions—which can lead to duplicate records. These duplicates cause inefficiencies in reporting and can lead to incorrect decision-making when it comes to resource allocation, student support, and performance analysis.

Recently, EduTrack received a text file containing student exam results that were processed through Optical Character Recognition (OCR) from legacy documents. The file is formatted with lines structured as follows:

NAME STUDENT ID Marks MARKS

**Alice** - A293:Marks 32

**Bob** - BD29DMarks 53

**Charlie** - XF28:Marks40

The data spans multiple subjects and time periods. The file will contain duplicate entries for the same student (identified by the second field), and it is crucial to count only unique students for accurate reporting.

## Your Task

As a data analyst at EduTrack Systems, your task is to process this text file and determine the number of unique students based on their student IDs. This deduplication is essential to:

* **Ensure Accurate Reporting:** Avoid inflated counts in enrollment and performance reports.
* **Improve Data Quality:** Clean the dataset for further analytics, such as tracking academic progress or resource allocation.
* **Optimize Administrative Processes:** Provide administrators with reliable data to support decision-making.

You need to do the following:

1. **Data Extraction:** Read the text file line by line. Parse each line to extract the student ID.
2. **Deduplication:** Remove duplicates from the student ID list.
3. **Reporting:** Count the number of unique student IDs present in the file.

By accurately identifying the number of unique students, EduTrack Systems will:

* **Enhance Data Integrity:** Ensure that subsequent analyses and reports reflect the true number of individual students.
* **Reduce Administrative Errors:** Minimize the risk of misinformed decisions that can arise from duplicate entries.
* **Streamline Resource Allocation:** Provide accurate student counts for budgeting, staffing, and planning academic programs.
* **Improve Compliance Reporting:** Ensure adherence to regulatory requirements by maintaining precise student records.

Download the text file with student marks q-clean-up-student-marks.txt

How many unique students are there in the file?

Correct

Check

3 Apache log requests (1 mark)

## Data Preparation in the Shell

You'll learn how to use UNIX tools to process and clean data, covering:

* curl (or wget) to fetch data from websites.
* gzip (or xz) to compress and decompress files.
* wc to count lines, words, and characters in text.
* head and tail to get the start and end of files.
* cut to extract specific columns from text.
* uniq to de-duplicate lines.
* sort to sort lines.
* grep to filter lines containing specific text.
* sed to search and replace text.
* awk for more complex text processing.

Here are the links used in the video:

* [Data preparation in the shell - Notebook](https://colab.research.google.com/drive/1KSFkQDK0v__XWaAaHKeQuIAwYV0dkTe8)
* [Data Science at the Command Line](https://jeroenjanssens.com/dsatcl/)

## Peak Usage Analysis for Regional Content

**s-anand.net** is a personal website that had region-specific music content. One of the site's key sections is **malayalam**, which hosts music files and is especially popular among the local audience. The website is powered by robust Apache web servers that record detailed access logs. These logs are essential for understanding user behavior, server load, and content engagement.

The author noticed unusual traffic patterns during weekend evenings. To better tailor their content and optimize server resources, they need to know precisely how many successful requests are made to the **malayalam** section during peak hours on Wednesday. Specifically, they are interested in:

* **Time Window:** From **0** until before **6**.
* **Request Type:** Only **GET** requests.
* **Success Criteria:** Requests that return HTTP status codes between **200 and 299**.
* **Data Source:** The logs for May 2024 stored in a GZipped Apache log file containing 258,074 rows.

The challenge is further complicated by the nature of the log file:

* The logs are recorded in the GMT-0500 timezone.
* The file format is non-standard in that fields are separated by spaces, with most fields quoted by double quotes, except the **Time** field.
* Some lines have minor formatting issues (41 rows have unique quoting due to how quotes are escaped).

### **Your Task**

As a data analyst, you are tasked with determining how many **successful GET requests** for pages under **malayalam** were made on Wednesday between **0** and **6** during May 2024. This metric will help:

* **Scale Resources:** Ensure that servers can handle the peak load during these critical hours.
* **Content Planning:** Determine the popularity of regional content to decide on future content investments.
* **Marketing Insights:** Tailor promotional strategies for peak usage times.

This [GZipped Apache log file](https://drive.google.com/file/d/1xLx-odohCtdPYbpOTui23upsCSzMBlpN/view) (61MB) has 258,074 rows. Each row is an Apache web log entry for the site [s-anand.net](https://s-anand.net/) in May 2024.

Each row has these fields:

* **IP**: The IP address of the visitor
* **Remote logname**: The remote logname of the visitor. Typically "-"
* **Remote user**: The remote user of the visitor. Typically "-"
* **Time**: The time of the visit. E.g. **[01/May/2024:00:00:00 +0000]**. Not that **this is not quoted** and you need to handle this.
* **Request**: The request made by the visitor. E.g. **GET /blog/ HTTP/1.1**. It has 3 space-separated parts, namely (a) **Method**: The HTTP method. E.g. **GET** (b) **URL**: The URL visited. E.g. **/blog/** (c) **Protocol**: The HTTP protocol. E.g. **HTTP/1.1**
* **Status**: The HTTP status code. If **200 <= Status < 300** it is a successful request
* **Size**: The size of the response in bytes. E.g. **1234**
* **Referer**: The referer URL. E.g. [**https://s-anand.net/**](https://s-anand.net/)
* **User agent**: The browser used. This will contain spaces and might have escaped quotes.
* **Vhost**: The virtual host. E.g. **s-anand.net**
* **Server**: The IP address of the server.

The fields are separated by spaces and quoted by double quotes (**"**). Unlike CSV files, quoted fields are escaped via **\"** and not **""**. (This impacts 41 rows.)

All data is in the GMT-0500 timezone and the questions are based in this same timezone.

By determining the number of successful GET requests under the defined conditions, we'll be able to:

* **Optimize Infrastructure:** Scale server resources effectively during peak traffic times, reducing downtime and improving user experience.
* **Strategize Content Delivery:** Identify popular content segments and adjust digital content strategies to better serve the audience.
* **Improve Marketing Efforts:** Focus marketing initiatives on peak usage windows to maximize engagement and conversion.

What is the number of successful GET requests for pages under **/malayalam/** from **0:00** until before **6:00** on Wednesdays?

Correct

Check

4 Apache log downloads (1 mark)

## Bandwidth Analysis for Regional Content

**s-anand.net** is a personal website that had region-specific music content. One of the site's key sections is **tamil**, which hosts music files and is especially popular among the local audience. The website is powered by robust Apache web servers that record detailed access logs. These logs are essential for understanding user behavior, server load, and content engagement.

By analyzing the server’s Apache log file, the author can identify heavy users and take measures to manage bandwidth, improve site performance, or even investigate potential abuse.

### **Your Task**

This [GZipped Apache log file](https://drive.google.com/file/d/1xLx-odohCtdPYbpOTui23upsCSzMBlpN/view) (61MB) has 258,074 rows. Each row is an Apache web log entry for the site [s-anand.net](https://s-anand.net/) in May 2024.

Each row has these fields:

* **IP**: The IP address of the visitor
* **Remote logname**: The remote logname of the visitor. Typically "-"
* **Remote user**: The remote user of the visitor. Typically "-"
* **Time**: The time of the visit. E.g. **[01/May/2024:00:00:00 +0000]**. Not that **this is not quoted** and you need to handle this.
* **Request**: The request made by the visitor. E.g. **GET /blog/ HTTP/1.1**. It has 3 space-separated parts, namely (a) **Method**: The HTTP method. E.g. **GET** (b) **URL**: The URL visited. E.g. **/blog/** (c) **Protocol**: The HTTP protocol. E.g. **HTTP/1.1**
* **Status**: The HTTP status code. If **200 <= Status < 300** it is a successful request
* **Size**: The size of the response in bytes. E.g. **1234**
* **Referer**: The referer URL. E.g. [**https://s-anand.net/**](https://s-anand.net/)
* **User agent**: The browser used. This will contain spaces and might have escaped quotes.
* **Vhost**: The virtual host. E.g. **s-anand.net**
* **Server**: The IP address of the server.

The fields are separated by spaces and quoted by double quotes (**"**). Unlike CSV files, quoted fields are escaped via **\"** and not **""**. (This impacts 41 rows.)

All data is in the GMT-0500 timezone and the questions are based in this same timezone.

1. **Filter the Log Entries:** Extract only the requests where the URL starts with **/tamil/**. Include only those requests made on the specified **2024-05-20**.
2. **Aggregate Data by IP:** Sum the "Size" field for each unique IP address from the filtered entries.
3. **Identify the Top Data Consumer:** Determine the IP address that has the highest total downloaded bytes. Reports the total number of bytes that this IP address downloaded.

Across all requests under tamil/ on 2024-05-20, how many bytes did the top IP address (by volume of downloads) download?

Correct

Check

5 Clean up sales data (1 mark)

## Cleaning Data with OpenRefine

This session covers the use of OpenRefine for data cleaning, focusing on resolving entity discrepancies:

* **Data Upload and Project Creation**: Import data into OpenRefine and create a new project for analysis.
* **Faceting Data**: Use text facets to group similar entries and identify frequency of address crumbs.
* **Clustering Methodology**: Apply clustering algorithms to merge similar entries with minor differences, such as punctuation.
* **Manual and Automated Clustering**: Learn to merge clusters manually or in one go, trusting the system's clustering accuracy.
* **Entity Resolution**: Clean and save the data by resolving multiple versions of the same entity using Open Refine.

Here are links used in the video:

* [OpenRefine software](https://openrefine.org/)
* [Dataset for OpenRefine](https://drive.google.com/file/d/1ccu0Xxk8UJUa2Dz4lihmvzhLjvPy42Ai/view)

## Sales Analytics at GlobalRetail Insights

**GlobalRetail Insights** is a market research and analytics firm specializing in providing data-driven insights for multinational retail companies. Their clients rely on accurate, detailed sales reports to make strategic decisions regarding product placement, inventory management, and marketing campaigns. However, the quality of these insights depends on the reliability of the underlying sales data.

One major challenge GlobalRetail faces is the inconsistent recording of city names in sales data. Due to human error and regional differences, city names can be mis-spelt (e.g., "Tokio" instead of "Tokyo"). This inconsistency complicates the process of aggregating sales data by city, which is crucial for identifying regional trends and opportunities.

GlobalRetail Insights recently received a dataset named q-clean-up-sales-data.json from one of its large retail clients. The dataset consists of 2,500 sales entries, each containing the following fields:

* **city**: The city where the sale was made. Note that city names may be mis-spelt phonetically (e.g., "Tokio" instead of "Tokyo").
* **product**: The product sold. This field is consistently spelled.
* **sales**: The number of units sold.

The client's goal is to evaluate the performance of a specific product across various regions. However, due to the mis-spelled city names, directly aggregating sales by city would lead to fragmented and misleading insights.

## Your Task

As a data analyst at GlobalRetail Insights, you are tasked with extracting meaningful insights from this dataset. Specifically, you need to:

1. **Group Mis-spelt City Names:** Use phonetic clustering algorithms to group together entries that refer to the same city despite variations in spelling. For instance, cluster "Tokyo" and "Tokio" as one.
2. **Filter Sales Entries:** Select all entries where:
   * The product sold is **Mouse**.
   * The number of units sold is at least **84**.
3. **Aggregate Sales by City:** After clustering city names, group the filtered sales entries by city and calculate the total units sold for each city.

By performing this analysis, GlobalRetail Insights will be able to:

* **Improve Data Accuracy:** Correct mis-spellings and inconsistencies in the dataset, leading to more reliable insights.
* **Target Marketing Efforts:** Identify high-performing regions for the specific product, enabling targeted promotional strategies.
* **Optimize Inventory Management:** Ensure that inventory allocations reflect the true demand in each region, reducing wastage and stockouts.
* **Drive Strategic Decision-Making:** Provide actionable intelligence to clients that supports strategic planning and competitive advantage in the market.

How many units of Mouse were sold in Manila on transactions with at least 84 units?

Correct

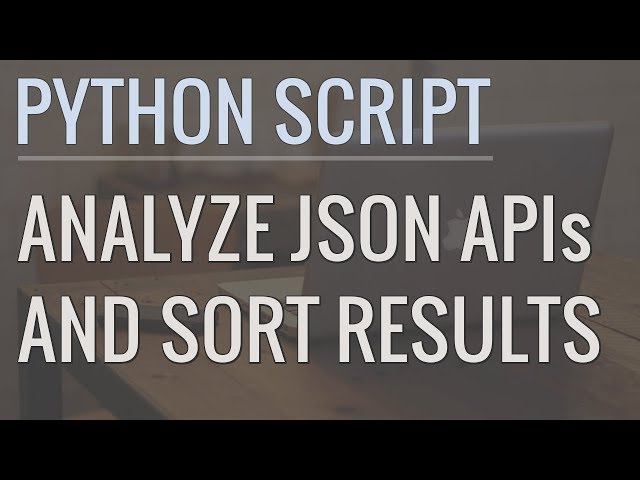
Check

6 Parse partial JSON (1 mark)

## Parsing JSON

JSON is everywhere—APIs, logs, configuration files—and its nested or large structure can challenge memory and processing. In this tutorial, we'll explore tools to flatten, stream, and query JSON data efficiently.

For example, we'll often need to process a multi-gigabyte log file from a web service where each record is a JSON object.

[](https://youtu.be/1lxrb_ezP-g)

This requires us to handle complex nested structures, large files that don't fit in memory, or extract specific fields. Here are the key tools and techniques for efficient JSON parsing:

| **Tool** | **Extract from JSON...** | **Why** |
| --- | --- | --- |
| [jq](https://exam.sanand.workers.dev/tds-2025-01-ga5#command-line-json-processing-with-jq) | JSON in the shell | Quick data exploration and pipeline processing |
| [JMESPath](https://exam.sanand.workers.dev/tds-2025-01-ga5#jmespath-queries) | JSON in Python | Handle complex queries with a clean syntax |
| [ijson](https://exam.sanand.workers.dev/tds-2025-01-ga5#streaming-with-ijson) | JSON streams in Python | Parse streaming/large JSON files memory-efficiently |
| [Pandas](https://exam.sanand.workers.dev/tds-2025-01-ga5#pandas-json-columns) | JSON columns in Python | Fast analysis of structured data |
| [SQL JSON](https://exam.sanand.workers.dev/tds-2025-01-ga5#sql-json-functions) | JSON in databases | Combine structured and semi-structured data |
| [DuckDB](https://exam.sanand.workers.dev/tds-2025-01-ga5#duckdb-json-processing) | JSON anywhere | Fast analysis of JSON files / databases without loading to memory |

**Examples:**

* Use Pandas when you need to transform API responses into a DataFrame for further analysis.
* Leverage ijson when dealing with huge JSON logs where memory is at a premium.
* Apply jq for quick, iterative exploration directly in your terminal.

Practice with these resources:

* [JSONPath Online Evaluator](https://jsonpath.com/): Test JSON queries
* [jq play](https://jqplay.org/): Interactive jq query testing
* [DuckDB JSON Tutorial](https://duckdb.org/docs/data/json): Learn DuckDB JSON functions

### **Command-line JSON Processing with jq**

[jq](https://jqlang.org/) is a versatile command-line tool for slicing, filtering, and transforming JSON. It excels in quick data exploration and can be integrated into shell scripts for automated data pipelines.

**Example:** Sifting through server logs in JSON Lines format to extract error messages or aggregate metrics without launching a full-scale ETL process.

# Extract specific fields from JSONL

cat data.jsonl | jq -c 'select(.type == "user") | {id, name}'

# Transform JSON structure

cat data.json | jq '.items[] | {name: .name, count: .details.count}'

# Filter and aggregate

cat events.jsonl | jq -s 'group\_by(.category) | map({category: .[0].category, count: length})'

### **JMESPath Queries**

[JMESPath](https://jmespath.org/) offers a declarative query language to extract and transform data from nested JSON structures without needing verbose code. It's a neat alternative when you want to quickly pull out specific values or filter collections based on conditions.

**Example:** Extracting user emails or filtering out inactive records from a complex JSON payload received from a cloud service.

**import** jmespath

# Example queries

data = {

"locations": [

{"name": "Seattle", "state": "WA", "info": {"population": 737015}},

{"name": "Portland", "state": "OR", "info": {"population": 652503}}

]

}

# Find all cities with population > 700000

cities = jmespath.search("locations[?info.population > `700000`].name", data)

### **Streaming with ijson**

Loading huge JSON files all at once can quickly exhaust system memory. [ijson](https://ijson.readthedocs.io/en/latest/) lets you stream and process JSON incrementally. This method is ideal when your JSON file is too large or when you only need to work with part of the data.

**Example:** Processing a continuous feed from an API that returns a large JSON array, such as sensor data or event logs, while filtering on the fly.

**import** ijson

**async** **def** **process\_large\_json**(filepath: str) -> list:

"""Process a large JSON file without loading it entirely into memory."""

results = []

**with** open(filepath, 'rb') **as** file:

# Stream objects under the 'items' key

parser = ijson.items(file, 'items.item')

**async** **for** item **in** parser:

**if** item['value'] > 100: # Process conditionally

results.append(item)

**return** results

### **Pandas JSON Columns**

[Pandas](https://pandas.pydata.org/) makes it easy to work with tabular data that includes JSON strings. When you receive API data where one column holds nested JSON, flattening these structures lets you analyze and visualize the data using familiar DataFrame operations.

**Example:** Flattening customer records stored as nested JSON in a CSV file to extract demographic details and spending patterns.

**import** pandas **as** pd

# Parse JSON strings in a column

df = pd.DataFrame({'json\_col': ['{"name": "Alice", "age": 30}', '{"name": "Bob", "age": 25}']})

df['parsed'] = df['json\_col'].apply(pd.json\_normalize)

# Normalize nested JSON columns

df = pd.read\_csv('data.csv')

df\_normalized = pd.json\_normalize(

df['nested\_json'].apply(json.loads),

record\_path=['items'], # List of nested objects to unpack

meta=['id', 'timestamp'] # Keep these columns from parent

)

### **SQL JSON Functions**

[SQL](https://en.wikipedia.org/wiki/SQL:2016) supports built-in JSON functions allow you to query and manipulate JSON stored within relational databases. These are implemented by most popular databases, including [SQLite](https://www.sqlite.org/json1.html), [PostgreSQL](https://www.postgresql.org/docs/current/functions-json.html), and [MySQL](https://dev.mysql.com/doc/refman/8.4/en/json-function-reference.html). This is especially handy when you have a hybrid data model, combining structured tables with semi-structured JSON columns.

**Example:** An application that stores user settings or application logs as JSON in a SQLite database, enabling quick lookups and modifications without external JSON parsing libraries.

**SELECT**

json\_extract(data, '$.name') **as** name,

json\_extract(data, '$.details.age') **as** age

**FROM** users

**WHERE** json\_extract(data, '$.active') = true

### **DuckDB JSON Processing**

[DuckDB](https://duckdb.org/) shines when analyzing JSON/JSONL files directly, making it a powerful tool for data analytics without the overhead of loading entire datasets into memory. Its SQL-like syntax simplifies exploratory analysis on nested data.

**Example:** Performing ad-hoc analytics on streaming JSON logs from a web service, such as calculating average response times or aggregating user behavior metrics.

**SELECT**

json\_extract\_string(data, '$.user.name') **as** name,

avg(json\_extract\_float(data, '$.metrics.value')) **as** avg\_value

**FROM** read\_json\_auto('data/\*.jsonl')

**GROUP** **BY** 1

**HAVING** avg\_value > 100

## ****Case Study: Recovering Sales Data for ReceiptRevive Analytics****

**ReceiptRevive Analytics** is a data recovery and business intelligence firm specializing in processing legacy sales data from paper receipts. Many of the client companies have archives of receipts from past years, which have been digitized using OCR (Optical Character Recognition) techniques. However, due to the condition of some receipts (e.g., torn, faded, or partially damaged), the OCR process sometimes produces incomplete JSON data. These imperfections can lead to truncated fields or missing values, which complicates the process of data aggregation and analysis.

One of ReceiptRevive’s major clients, **RetailFlow Inc.**, operates numerous brick-and-mortar stores and has an extensive archive of old receipts. RetailFlow Inc. needs to recover total sales information from a subset of these digitized receipts to analyze historical sales performance. The provided JSON data contains 100 rows, with each row representing a sales entry. Each entry is expected to include four keys:

* **city**: The city where the sale was made.
* **product**: The product that was sold.
* **sales**: The number of units sold (or sales revenue).
* **id**: A unique identifier for the receipt.

Due to damage to some receipts during the digitization process, the JSON entries are truncated at the end, and the id field is missing. Despite this, RetailFlow Inc. is primarily interested in the aggregate sales value.

## Your Task

As a data recovery analyst at ReceiptRevive Analytics, your task is to develop a program that will:

1. **Parse the Sales Data:**  
   Read the provided JSON file containing 100 rows of sales data. Despite the truncated data (specifically the missing id), you must accurately extract the sales figures from each row.
2. **Data Validation and Cleanup:**  
   Ensure that the data is properly handled even if some fields are incomplete. Since the id is missing for some entries, your focus will be solely on the sales values.
3. **Calculate Total Sales:**  
   Sum the sales values across all 100 rows to provide a single aggregate figure that represents the total sales recorded.

By successfully recovering and aggregating the sales data, ReceiptRevive Analytics will enable RetailFlow Inc. to:

* **Reconstruct Historical Sales Data:** Gain insights into past sales performance even when original receipts are damaged.
* **Inform Business Decisions:** Use the recovered data to understand sales trends, adjust inventory, and plan future promotions.
* **Enhance Data Recovery Processes:** Improve methods for handling imperfect OCR data, reducing future data loss and increasing data accuracy.
* **Build Client Trust:** Demonstrate the ability to extract valuable insights from challenging datasets, thereby reinforcing client confidence in ReceiptRevive's services.

Download the data from q-parse-partial-json.jsonl

What is the total sales value?

Correct

Check

7 Extract nested JSON keys (1 mark)

## Log Analysis for DataSure Technologies

**DataSure Technologies** is a leading provider of IT infrastructure and software solutions, known for its robust systems and proactive maintenance practices. As part of their service offerings, DataSure collects extensive logs from thousands of servers and applications worldwide. These logs, stored in JSON format, are rich with information about system performance, error events, and user interactions. However, the logs are complex and deeply nested, which can make it challenging to quickly identify recurring issues or anomalous behavior.

Recently, DataSure's operations team observed an increase in system alerts and minor anomalies reported by their monitoring tools. To diagnose these issues more effectively, the team needs to perform a detailed analysis of the log files. One critical step is to count how often a specific key (e.g., "errorCode", "criticalFlag", or any other operational parameter represented by YM) appears in the log entries.

Key considerations include:

* **Complex Structure:** The log files are large and nested, with multiple levels of objects and arrays. The target key may appear at various depths.
* **Key vs. Value:** The key may also appear as a value within the logs, but only occurrences where it is a key should be counted.
* **Operational Impact:** Identifying the frequency of this key can help pinpoint common issues, guide system improvements, and inform maintenance strategies.

## Your Task

As a data analyst at DataSure Technologies, you have been tasked with developing a script that processes a large JSON log file and counts the number of times a specific key, represented by the placeholder YM, appears in the JSON structure. Your solution must:

1. **Parse the Large, Nested JSON:** Efficiently traverse the JSON structure regardless of its complexity.
2. **Count Key Occurrences:** Increment a count only when YM is used as a key in the JSON object (ignoring occurrences of YM as a value).
3. **Return the Count:** Output the total number of occurrences, which will be used by the operations team to assess the prevalence of particular system events or errors.

By accurately counting the occurrences of a specific key in the log files, DataSure Technologies can:

* **Diagnose Issues:** Quickly determine the frequency of error events or specific system flags that may indicate recurring problems.
* **Prioritize Maintenance:** Focus resources on addressing the most frequent issues as identified by the key count.
* **Enhance Monitoring:** Improve automated monitoring systems by correlating key occurrence data with system performance metrics.
* **Inform Decision-Making:** Provide data-driven insights that support strategic planning for system upgrades and operational improvements.

Download the data from q-extract-nested-json-keys.json

How many times does YM appear as a key?

Correct

Check

8 DuckDB: Social Media Interactions (1 mark)

## Identifying High-Impact Social Media Posts for EngageMetrics

**EngageMetrics** is a digital marketing analytics firm that specializes in tracking and analyzing social media engagement. Their clients, ranging from major brands to local businesses, rely on data-driven insights to optimize content strategy, measure campaign performance, and identify posts that drive significant user interaction.

Social media platforms generate vast amounts of user-generated content, including posts, comments, likes, and ratings. One key metric that EngageMetrics uses is the “usefulness” of comments on posts. A comment rated highly on usefulness is a strong indicator that a post is engaging and valuable to its audience.

However, the raw data is complex and stored in a DuckDB table called **social\_media**, which is generated by a simulated dataset. Each row in the table represents a post with the following fields:

* **post\_id**: A unique identifier for the post.
* **username**: The user who created the post.
* **timestamp**: The time when the post was made.
* **comments**: A JSON array containing comments. Each comment includes:
  + **commenter**: The user who commented.
  + **text**: The comment text.
  + **stars**: An object with two properties, **funny** and **useful**, representing the rating stars for that comment.

Due to the dynamic nature of social media, EngageMetrics wants to focus on the most recent posts and, within those, identify posts where at least one comment has received a high number of useful stars. This allows the firm to spotlight content that is not only fresh but also resonating well with the audience.

## Your Task

Your task as a data analyst at EngageMetrics is to write a query that performs the following:

1. **Filter Posts by Date:** Consider only posts with a **timestamp** greater than or equal to a specified minimum time (2024-11-19T22:45:12.008Z), ensuring that the analysis focuses on recent posts.
2. **Evaluate Comment Quality:** From these recent posts, identify posts where at least one comment has received more than a given number of useful stars (5). This criterion filters out posts with low or mediocre engagement.
3. **Extract and Sort Post IDs:** Finally, extract all the post\_id values of the posts that meet these criteria and sort them in ascending order.

By accurately extracting these high-impact post IDs, EngageMetrics can:

* **Enhance Reporting:** Provide clients with focused insights on posts that are currently engaging audiences effectively.
* **Target Content Strategy:** Help marketing teams identify trending content themes that generate high-quality user engagement.
* **Optimize Resource Allocation:** Enable better prioritization for content promotion and further in-depth analysis of high-performing posts.

Write a DuckDB SQL query to find all posts IDs after 2024-11-19T22:45:12.008Z with at least 1 comment with 5 useful stars, sorted. The result should be a table with a single column called post\_id, and the relevant post IDs should be sorted in ascending order.

Error: At root: Array length mismatch

Check the console for the result of your query.

Check

9 Transcribe a YouTube video (1 mark)

## Extracting Audio and Transcripts

## Media Processing: FFmpeg

[FFmpeg](https://ffmpeg.org/) is the standard command-line tool for processing video and audio files. It's essential for data scientists working with media files for:

* Extracting audio/video for machine learning
* Converting formats for web deployment
* Creating visualizations and presentations
* Processing large media datasets

Basic Operations:

# Basic conversion

ffmpeg -i input.mp4 output.avi

# Extract audio

ffmpeg -i input.mp4 -vn output.mp3

# Convert format without re-encoding

ffmpeg -i input.mkv -c copy output.mp4

# High quality encoding (crf: 0-51, lower is better)

ffmpeg -i input.mp4 -preset slower -crf 18 output.mp4

Common Data Science Tasks:

# Extract frames for computer vision

ffmpeg -i input.mp4 -vf "fps=1" frames\_%04d.png # 1 frame per second

ffmpeg -i input.mp4 -vf "select='eq(n,0)'" -vframes 1 first\_frame.jpg

# Create video from image sequence

ffmpeg -r 1/5 -i img%03d.png -c:v libx264 -vf fps=25 output.mp4

# Extract audio for speech recognition

ffmpeg -i input.mp4 -ar 16000 -ac 1 audio.wav # 16kHz mono

# Trim video/audio for training data

ffmpeg -ss 00:01:00 -i input.mp4 -t 00:00:30 -c copy clip.mp4

Processing Multiple Files:

# Concatenate videos (first create files.txt with list of files)

echo "file 'input1.mp4'

file 'input2.mp4'" > files.txt

ffmpeg -f concat -i files.txt -c copy output.mp4

# Batch process with shell loop

**for** f **in** \*.mp4; **do**

ffmpeg -i "$f" -vn "audio/${f%.mp4}.wav"

**done**

Data Analysis Features:

# Get media file information

ffprobe -v quiet -print\_format json -show\_format -show\_streams input.mp4

# Display frame metadata

ffprobe -v quiet -print\_format json -show\_frames input.mp4

# Generate video thumbnails

ffmpeg -i input.mp4 -vf "thumbnail" -frames:v 1 thumb.jpg

Watch this introduction to FFmpeg (12 min):

Tools:

* [ffmpeg.lav.io](https://ffmpeg.lav.io/): Interactive command builder
* [FFmpeg Explorer](https://ffmpeg.guide/): Visual FFmpeg command generator
* [FFmpeg Buddy](https://evanhahn.github.io/ffmpeg-buddy/): Simple command generator

Tips:

1. Use -c copy when possible to avoid re-encoding
2. Monitor progress with -progress pipe:1
3. Use -hide\_banner to reduce output verbosity
4. Test commands with small clips first
5. Use hardware acceleration when available (-hwaccel auto)

Error Handling:

# Validate file before processing

ffprobe input.mp4 2>&1 | grep "Invalid"

# Continue on errors in batch processing

ffmpeg -i input.mp4 output.mp4 -xerror

# Get detailed error information

ffmpeg -v error -i input.mp4 2>&1 | grep -A2 "Error"

## Media tools: yt-dlp

[yt-dlp](https://github.com/yt-dlp/yt-dlp) is a feature-rich command-line tool for downloading audio/video from thousands of sites. It's particularly useful for extracting audio and transcripts from videos.

Install using your package manager:

# macOS

brew install yt-dlp

# Linux

curl -L https://github.com/yt-dlp/yt-dlp/releases/latest/download/yt-dlp -o ~/.local/bin/yt-dlp

chmod a+rx ~/.local/bin/yt-dlp

# Windows

winget install yt-dlp

Common operations for extracting audio and transcripts:

# Download audio only at lowest quality suitable for speech

yt-dlp -f "ba[abr<50]/worstaudio" \

--extract-audio \

--audio-format mp3 \

--audio-quality 32k \

"https://www.youtube.com/watch?v=VIDEO\_ID"

# Download auto-generated subtitles

yt-dlp --write-auto-sub \

--skip-download \

--sub-format "srt" \

"https://www.youtube.com/watch?v=VIDEO\_ID"

# Download both audio and subtitles with custom output template

yt-dlp -f "ba[abr<50]/worstaudio" \

--extract-audio \

--audio-format mp3 \

--audio-quality 32k \

--write-auto-sub \

--sub-format "srt" \

-o "%(title)s.%(ext)s" \

"https://www.youtube.com/watch?v=VIDEO\_ID"

# Download entire playlist's audio

yt-dlp -f "ba[abr<50]/worstaudio" \

--extract-audio \

--audio-format mp3 \

--audio-quality 32k \

-o "%(playlist\_index)s-%(title)s.%(ext)s" \

"https://www.youtube.com/playlist?list=PLAYLIST\_ID"

For Python integration:

# /// script

# requires-python = ">=3.9"

# dependencies = ["yt-dlp"]

# ///

**import** yt\_dlp

**def** **download\_audio**(url: str) -> None:

"""Download audio at speech-optimized quality."""

ydl\_opts = {

'format': 'ba[abr<50]/worstaudio',

'postprocessors': [{

'key': 'FFmpegExtractAudio',

'preferredcodec': 'mp3',

'preferredquality': '32'

}]

}

**with** yt\_dlp.YoutubeDL(ydl\_opts) **as** ydl:

ydl.download([url])

# Example usage

download\_audio('https://www.youtube.com/watch?v=VIDEO\_ID')

Tools:

* [ffmpeg](https://ffmpeg.org/): Required for audio extraction and conversion
* [whisper](https://github.com/openai/whisper): Can be used with yt-dlp for speech-to-text
* [gallery-dl](https://github.com/mikf/gallery-dl): Alternative for image-focused sites

Note: Always respect copyright and terms of service when downloading content.

## Whisper transcription

[Faster Whisper](https://github.com/SYSTRAN/faster-whisper) is a highly optimized implementation of OpenAI's [Whisper model](https://github.com/openai/whisper), offering up to 4x faster transcription while using less memory.

You can install it via:

* pip install faster-whisper
* [Download Windows Standalone](https://github.com/Purfview/whisper-standalone-win/releases)

Here's a basic usage example:

faster-whisper-xxl "video.mp4" --model medium --language en

Here's my recommendation for transcribing videos. This saves the output in JSON as well as SRT format in the source directory.

faster-whisper-xxl --print\_progress --output\_dir source --batch\_recursive \

--check\_files --standard --output\_format json srt \

--model medium --language en $FILE

* --model: The OpenAI Whisper model to use. You can choose from:
  + tiny: Fastest but least accurate
  + base: Good for simple audio
  + small: Balanced speed/accuracy
  + medium: Recommended default
  + large-v3: Most accurate but slowest
* --output\_format: The output format to use. You can pick multiple formats from:
  + json: Has the most detailed information including timing, text, quality, etc.
  + srt: A popular subtitle format. You can use this in YouTube, for example.
  + vtt: A modern subtitle format.
  + txt: Just the text transcript
* --output\_dir: The directory to save the output files. source indicates the source directory, i.e. where the input $FILE is
* --language: The language of the input file. If you don't specify it, it analyzes the first 30 seconds to auto-detect. You can speed it up by specifying it.

Run faster-whisper-xxl --help for more options.

## Gemini transcription

The [Gemini](https://gemini.google.com/) models from Google are notable in two ways:

1. They have a huge input context window. Gemini 2.0 Flash can accept 1M tokens, for example.
2. They can handle audio input.

This allows us to use Gemini to transcribe audio files.

LLMs are not good at transcribing audio faithfully. They tend to correct errors and meander from what was said. But they are intelligent. That enables a few powerful workflows. Here are some examples:

1. **Transcribe into other languages**. Gemini will handle the transcription and translation in a single step.
2. **Summarize audio transcripts**. For example, convert a podcast into a tutorial, or a meeting recording into actions.
3. **Legal Proceeding Analysis**. Extract case citations, dates, and other details from a legal debate.
4. **Medical Consultation Summary**. Extract treatments, medications, details of next visit, etc. from a medical consultation.

Here's how to use Gemini to transcribe audio files.

1. Get a [Gemini API key](https://aistudio.google.com/app/apikey) from Google AI Studio.
2. Set the GEMINI\_API\_KEY environment variable to the API key.
3. Set the MP3\_FILE environment variable to the path of the MP3 file you want to transcribe.
4. Run this code:
5. curl -X POST https://generativelanguage.googleapis.com/v1beta/models/gemini-1.5-flash-002:streamGenerateContent?alt=sse \
6. -H "X-Goog-API-Key: $GEMINI\_API\_KEY" \
7. -H "Content-Type: application/json" \
8. -d "$(cat << EOF
9. {
10. "contents": [
11. {
12. "role": "user",
13. "parts": [
14. {
15. "inline\_data": {
16. "mime\_type": "audio/mp3",
17. "data": "$(base64 --wrap=0 $MP3\_FILE)"
18. }
19. },
20. {"text": "Transcribe this"}
21. ]
22. }
23. ]
24. }
25. EOF
26. )"

## Enhancing Accessibility and Content Analysis for Mystery Audiobooks

**Mystery Tales Publishing** is an independent publisher specializing in mystery and suspense audiobooks. To broaden their audience and improve accessibility, they have been uploading narrated versions of their stories to YouTube. In addition to reaching visually impaired users, they want to leverage transcripts for content summarization, search indexing, and social media promotion.

The company has identified that certain segments of their mystery story audiobooks generate the most engagement. However, transcribing entire videos can be time-consuming and may include irrelevant content. Therefore, they have decided to focus on transcribing only the most compelling segments. For instance, a particular segment—from **213.7** to **298.5**—is known to captivate listeners with a twist in the plot. An accurate transcript of this segment will:

* Enhance accessibility by providing a text alternative for hearing-impaired users.
* Improve search engine optimization (SEO) through indexed keywords.
* Support content analysis and summarization for promotional purposes.

As part of a pilot project, you are tasked with transcribing the YouTube video segment of a mystery story audiobook. You are provided with a sample video that features a narrated mystery story. Your focus will be on the segment starting at **213.7** and ending at **298.5**.

Your transcription should:

* Accurately capture all spoken dialogue and descriptive narration.
* Include appropriate punctuation and paragraph breaks to reflect natural speech.
* Exclude any extraneous noise or background commentary not relevant to the narrative.

## Your Task

1. **Access the Video:** Use the provided YouTube link to access the mystery story audiobook.
2. **Convert to Audio:** Extract the audio for the segment between **213.7** and **298.5**.
3. **Transcribe the Segment:** Utilize automated speech-to-text tools as needed.

By producing an accurate transcript of this key segment, Mystery Tales Publishing will be able to:

* **Boost Accessibility:** Provide high-quality captions and text alternatives for hearing-impaired users.
* **Enhance SEO:** Improve the discoverability of their content through better keyword indexing.
* **Drive Engagement:** Use the transcript for social media snippets, summaries, and promotional materials.
* **Enable Content Analysis:** Facilitate further analysis such as sentiment analysis, topic modeling, and reader comprehension studies.

What is the text of the transcript of this [Mystery Story Audiobook](https://youtu.be/NRntuOJu4ok) between 213.7 and 298.5 seconds?

Error: 45 differences

Check

10 Reconstruct an image (1 mark)

## Transforming Images

### **Image Processing with PIL (Pillow)**

[Pillow](https://python-pillow.org/) is Python's leading library for image processing, offering powerful tools for editing, analyzing, and generating images. It handles various formats (PNG, JPEG, GIF, etc.) and provides operations from basic resizing to complex filters.

Here's a minimal example showing common operations:

# /// script

# requires-python = ">=3.11"

# dependencies = ["Pillow"]

# ///

**from** PIL **import** Image, ImageEnhance, ImageFilter

**async** **def** **process\_image**(path: str) -> Image.Image:

"""Process an image with basic enhancements."""

**with** Image.open(path) **as** img:

# Convert to RGB to ensure compatibility

img = img.convert('RGB')

# Resize maintaining aspect ratio

img.thumbnail((800, 800))

# Apply enhancements

img = (ImageEnhance.Contrast(img)

.enhance(1.2))

**return** img.filter(ImageFilter.SHARPEN)

**if** \_\_name\_\_ == "\_\_main\_\_":

**import** asyncio

img = asyncio.run(process\_image("input.jpg"))

img.save("output.jpg", quality=85)

Key features and techniques you'll learn:

* **Image Loading and Saving**: Handle various formats with automatic conversion
* **Basic Operations**: Resize, rotate, crop, and flip images
* **Color Manipulation**: Adjust brightness, contrast, and color balance
* **Filters and Effects**: Apply blur, sharpen, and other visual effects
* **Drawing**: Add text, shapes, and overlays to images
* **Batch Processing**: Handle multiple images efficiently
* **Memory Management**: Process large images without memory issues

### **Basic Image Operations**

Common operations for resizing, cropping, and rotating images:

**from** PIL **import** Image

**async** **def** **transform\_image**(

path: str,

size: tuple[int, int],

rotation: float = 0

) -> Image.Image:

"""Transform image with basic operations."""

**with** Image.open(path) **as** img:

# Resize with anti-aliasing

img = img.resize(size, Image.LANCZOS)

# Rotate around center

**if** rotation:

img = img.rotate(rotation, expand=True)

# Auto-crop empty edges

img = img.crop(img.getbbox())

**return** img

### **Color and Enhancement**

Adjust image appearance with built-in enhancement tools:

**from** PIL **import** ImageEnhance, ImageOps

**async** **def** **enhance\_image**(

img: Image.Image,

brightness: float = 1.0,

contrast: float = 1.0,

saturation: float = 1.0

) -> Image.Image:

"""Apply color enhancements to image."""

enhancers = [

(ImageEnhance.Brightness, brightness),

(ImageEnhance.Contrast, contrast),

(ImageEnhance.Color, saturation)

]

**for** Enhancer, factor **in** enhancers:

**if** factor != 1.0:

img = Enhancer(img).enhance(factor)

**return** img

### **Filters and Effects**

Apply visual effects and filters to images:

**from** PIL **import** ImageFilter

**def** **apply\_effects**(img: Image.Image) -> Image.Image:

"""Apply various filters and effects."""

effects = {

'blur': ImageFilter.GaussianBlur(radius=2),

'sharpen': ImageFilter.SHARPEN,

'edge': ImageFilter.FIND\_EDGES,

'emboss': ImageFilter.EMBOSS

}

**return** {name: img.filter(effect)

**for** name, effect **in** effects.items()}

### **Drawing and Text**

Add text, shapes, and overlays to images:

**from** PIL **import** Image, ImageDraw, ImageFont

**async** **def** **add\_watermark**(

img: Image.Image,

text: str,

font\_size: int = 30

) -> Image.Image:

"""Add text watermark to image."""

draw = ImageDraw.Draw(img)

font = ImageFont.truetype("arial.ttf", font\_size)

# Calculate text size and position

text\_bbox = draw.textbbox((0, 0), text, font=font)

text\_width = text\_bbox[2] - text\_bbox[0]

text\_height = text\_bbox[3] - text\_bbox[1]

# Position text at bottom-right

x = img.width - text\_width - 10

y = img.height - text\_height - 10

# Add text with shadow

draw.text((x+2, y+2), text, font=font, fill='black')

draw.text((x, y), text, font=font, fill='white')

**return** img

### **Memory-Efficient Processing**

Handle large images without loading them entirely into memory:

**from** PIL **import** Image

**import** os

**async** **def** **process\_large\_images**(

input\_dir: str,

output\_dir: str,

max\_size: tuple[int, int]

) -> None:

"""Process multiple large images efficiently."""

os.makedirs(output\_dir, exist\_ok=True)

**for** filename **in** os.listdir(input\_dir):

**if** **not** filename.lower().endswith(('.png', '.jpg', '.jpeg')):

**continue**

input\_path = os.path.join(input\_dir, filename)

output\_path = os.path.join(output\_dir, filename)

**with** Image.open(input\_path) **as** img:

# Process in chunks using thumbnail

img.thumbnail(max\_size)

img.save(output\_path, optimize=True)

Practice with these resources:

* [Pillow Documentation](https://pillow.readthedocs.io/): Complete API reference
* [Python Image Processing Tutorial](https://realpython.com/image-processing-with-the-python-pillow-library/): In-depth guide
* [Sample Images Dataset](https://www.kaggle.com/datasets/lamsimon/celebs): Test images for practice

Watch these tutorials for hands-on demonstrations:

### **Image Processing with ImageMagick**

[ImageMagick](https://imagemagick.org/) is a powerful command-line tool for image manipulation, offering features beyond what's possible with Python libraries. It's particularly useful for:

* Batch processing large image collections
* Complex image transformations
* High-quality format conversion
* Creating image thumbnails
* Adding text and watermarks

Basic Operations:

# Format conversion

convert input.png output.jpg

# Resize image (maintains aspect ratio)

convert input.jpg -resize 800x600 output.jpg

# Compress image quality

convert input.jpg -quality 85 output.jpg

# Rotate image

convert input.jpg -rotate 90 output.jpg

Common Data Science Tasks:

# Create thumbnails for dataset preview

convert input.jpg -thumbnail 200x200^ -gravity center -extent 200x200 thumb.jpg

# Normalize image for ML training

convert input.jpg -normalize -strip output.jpg

# Extract dominant colors

convert input.jpg -colors 5 -unique-colors txt:

# Generate image statistics

identify -verbose input.jpg | grep -E "Mean|Standard|Kurtosis"

Batch Processing:

# Convert all images in a directory

mogrify -format jpg \*.png

# Resize multiple images

mogrify -resize 800x600 -path output/ \*.jpg

# Add watermark to images

**for** f **in** \*.jpg; **do**

convert "$f" -gravity southeast -draw "text 10,10 'Copyright'" "watermarked/$f"

**done**

Advanced Features:

# Apply image effects

convert input.jpg -blur 0x3 blurred.jpg

convert input.jpg -sharpen 0x3 sharp.jpg

convert input.jpg -edge 1 edges.jpg

# Create image montage

montage \*.jpg -geometry 200x200+2+2 montage.jpg

# Extract image channels

convert input.jpg -separate channels\_%d.jpg

# Composite images

composite overlay.png -gravity center base.jpg output.jpg

Watch this ImageMagick tutorial (16 min):

Tools:

* [Fred's ImageMagick Scripts](http://www.fmwconcepts.com/imagemagick/): Useful script collection
* [ImageMagick Online Studio](https://magickstudio.imagemagick.org/): Visual command builder

Tips:

1. Use -strip to remove metadata and reduce file size
2. Monitor memory usage with -limit memory 1GB
3. Use -define for format-specific options
4. Process in parallel with -parallel
5. Use -monitor to track progress

Error Handling:

# Check image validity

identify -regard-warnings input.jpg

# Get detailed error information

convert input.jpg output.jpg 2>&1 | grep -i "error"

# Set resource limits

convert -limit memory 1GB -limit map 2GB input.jpg output.jpg

For Python integration:

# /// script

# requires-python = ">=3.9"

# dependencies = ["Wand"]

# ///

**from** wand.image **import** Image

**async** **def** **process\_image**(path: str) -> None:

"""Process image with ImageMagick via Wand."""

**with** Image(filename=path) **as** img:

# Basic operations

img.resize(800, 600)

img.normalize()

# Apply effects

img.sharpen(radius=0, sigma=3)

# Save with compression

img.save(filename='output.jpg')

Note: Always install ImageMagick before using the Wand Python binding.

## Image Reconstruction for Forensic Analysis

**PixelGuard Solutions** is a digital forensics firm specializing in the recovery and analysis of visual evidence for law enforcement and corporate investigations. One of their recurring challenges involves reconstructing damaged or deliberately scrambled images to reveal hidden details critical to solving cases.

In a recent investigation, law enforcement received an anonymous tip involving a scrambled image that appeared to contain sensitive information. The image had been deliberately cut into 25 (5x5) pieces and rearranged to obfuscate its content. Recovering the original image was essential for uncovering evidence related to the case.

PixelGuard's forensic team extracted the scrambled pieces and obtained a mapping file that specifies the transformation from the original (row, col) positions to the new positions. However, the team now needs to reassemble the image according to this mapping to restore its original appearance.

## Your Task

As a digital forensics analyst at PixelGuard Solutions, your task is to reconstruct the original image from its scrambled pieces. You are provided with:

* The 25 individual image pieces (put together as a single image).
* A mapping file detailing the original (row, col) position for each piece and its current (row, col) location.

Your reconstructed image will be critical evidence in the investigation. Once assembled, the image must be uploaded to the secure case management system for further analysis by the investigative team.

1. **Understand the Mapping:** Review the provided mapping file that shows how each piece's original coordinates (row, col) relate to its current scrambled position.
2. **Reassemble the Image:** Using the mapping, reassemble the 5x5 grid of image pieces to reconstruct the original image. You may use an image processing library (e.g., Python's Pillow, ImageMagick, or a similar tool) to automate the reconstruction process.
3. **Output the Reconstructed Image:** Save the reassembled image in a lossless format (e.g., PNG or WEBP). Upload the reconstructed image to the secure case management system as required by PixelGuard’s workflow.

By accurately reconstructing the scrambled image, PixelGuard Solutions will:

* **Reveal Critical Evidence:** Provide investigators with a clear view of the original image, which may contain important details related to the case.
* **Enhance Analytical Capabilities:** Enable further analysis and digital enhancements that can lead to breakthroughs in the investigation.
* **Maintain Chain of Custody:** Ensure that the reconstruction process is documented and reliable, supporting the admissibility of the evidence in court.
* **Improve Operational Efficiency:** Demonstrate the effectiveness of automated image reconstruction techniques in forensic investigations.

Here is the image. It is a 500x500 pixel image that has been cut into 25 (5x5) pieces:

Here is the mapping of each piece:

| **Original Row** | **Original Column** | **Scrambled Row** | **Scrambled Column** |
| --- | --- | --- | --- |
| 2 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 4 | 1 | 0 | 2 |
| 0 | 3 | 0 | 3 |
| 0 | 1 | 0 | 4 |
| 1 | 4 | 1 | 0 |
| 2 | 0 | 1 | 1 |
| 2 | 4 | 1 | 2 |
| 4 | 2 | 1 | 3 |
| 2 | 2 | 1 | 4 |
| 0 | 0 | 2 | 0 |
| 3 | 2 | 2 | 1 |
| 4 | 3 | 2 | 2 |
| 3 | 0 | 2 | 3 |
| 3 | 4 | 2 | 4 |
| 1 | 0 | 3 | 0 |
| 2 | 3 | 3 | 1 |
| 3 | 3 | 3 | 2 |
| 4 | 4 | 3 | 3 |
| 0 | 2 | 3 | 4 |
| 3 | 1 | 4 | 0 |
| 1 | 2 | 4 | 1 |
| 1 | 3 | 4 | 2 |
| 0 | 4 | 4 | 3 |
| 4 | 0 | 4 | 4 |

Upload the reconstructed image by moving the pieces from the scrambled position to the original position:

Error: No image uploaded

Check

Check all Save

#### **Submitted**

If you submit again, it will overwrite your previous submission and score.

Save regularly. Your last saved submission will be evaluated.

Bottom of Form

# Best of luck!