**Exercise 2A:**

**Solution 1:**

I am working on python coding, I know how to make regex for different kind of spellings of a single name and I have defined a flexible\_regex under which I want to develop some variations to cath the alternative spellings and transliteration. Don't show me the code, teach me how to do it.

###### **ChatGPT said:**

Great! Since you already know how to make regex and have a flexible\_regex variable defined, let’s walk through the **concepts** and **techniques** to design regex that catches variations in names due to alternative spellings, transliterations, or common typing quirks.

### **1. Understand Common Variations**

Ask: *How do people usually spell this name differently?*

For example, for the name **“Muhammad”**, possible variations are:

* Muhammad, Mohammad, Mohamad, Muhammed, Mohmmad, etc.

You’ll need to account for:

* **Vowel variations** (u ↔ o, a ↔ e)
* **Double letters** becoming single (mm → m)
* **Dropped vowels or letters**
* **Phonetic similarities** (e.g., "f" ↔ "ph")

### **2. Use Character Classes and Optional Groups**

Regex allows:

* [uo] → matches either "u" or "o"
* (mm)? → matches "mm" or nothing (optional group)
* a+ → matches one or more "a" characters
* (d|t) → matches "d" or "t", useful for dialects

### **3. Break the Name Into Parts**

Break complex names into meaningful **syllables or units** and write regex for each part, allowing variation within them.

Example: For "Muhammad":

* M[uo]h?a?m{1,2}[ae]d

Explanation:

* M[uo] → M followed by "u" or "o"
* h?a? → optional "h" or "a"
* m{1,2} → one or two "m"s
* [ae]d → ends with "ad" or "ed"

### **4. Use Non-Capturing Groups If Needed**

To keep things clean and prevent unintended group capturing:

* (?:...) is used instead of (...) when you don’t need to capture the group.

### **5. Apply Case-Insensitive Flag**

Most name variations are **case-insensitive**, so use the re.IGNORECASE flag (or (?i) inline).

### **6. Test Variations**

Have a list of known variations and test your regex using a loop or in an interactive tool like regex101 or your script.

Would you like a step-by-step guide for building regex for a specific name you have in mind?

4o

##### **You said:**

Is this right if I make something like this: name = re.sub(r"Kh", r"(Kh|K|Ḫ)")

###### **ChatGPT said:**

You're very close — but what you're trying to do is **not quite how re.sub() works**.

Let’s clarify:

### **🔁 What re.sub() Does:**

re.sub(pattern, replacement, string) replaces parts of a **given string** that match a pattern.

But in your example:

python

Copy code

name = re.sub(r"Kh", r"(Kh|K|Ḫ)")

You're using re.sub() as if it’s building a regex pattern — but it's actually **substituting** text in a string.

### **✅ What You Likely Meant To Do:**

If you're trying to **define a regex pattern** that can match "Kh", "K", or "Ḫ", you should construct the pattern **as a string**, like this:

python

Copy code

flexible\_regex = r"(Kh|K|Ḫ)"

Now you can use this regex in functions like re.search(), re.match(), or re.findall() to match any of those spellings.

### **🔁 If You Do Want to Replace in Text:**

If you're applying it **on a string**, then re.sub() is correct:

python

Copy code

text = "Khan"

new\_text = re.sub(r"Kh", r"(Kh|K|Ḫ)", text)

# But this makes the actual string: "(Kh|K|Ḫ)an", not a regex

This isn't helpful unless you're preparing **dynamic regex patterns** from existing strings.

### **Summary:**

* ✅ Use re.sub() to modify text strings.
* ✅ Use r"(Kh|K|Ḫ)" as a **regex pattern** if you're building flexible\_regex.
* ❌ Don’t use re.sub() to build regex patterns — just write the regex string directly.

Would you like help building a full flexible\_regex pattern for a sample name with multiple variations?

4o

##### **You said:**

Okay I Have a "name' variable here so what if I do this: name = re.sub(r"Kh", r"(Kh|K|Ḫ)", name) and I have to add the iggnore case also, help me with that as well

###### **ChatGPT said:**

Great — now that you're working with an actual name variable and want to **replace certain letter patterns with regex groups**, here's the correct way to use re.sub() **with ignore case**.

### **✅ Your Goal:**

You want to **substitute all instances of "Kh"** (case-insensitively) in name with the regex group (Kh|K|Ḫ) — likely to later use this updated string as a regex pattern.

### **✅ Correct Syntax:**

python

Copy code

import re

name = re.sub(r"Kh", r"(Kh|K|Ḫ)", name, flags=re.IGNORECASE)

### **💡 How It Works:**

* r"Kh": the pattern to match (it will match "Kh", "kh", "kH", or "KH" because of the flag)
* r"(Kh|K|Ḫ)": the replacement string — note this inserts the **regex group into the string**
* flags=re.IGNORECASE: makes the match case-insensitive

### **⚠️ Important Note:**

After this substitution, name becomes a **string containing a regex pattern** (e.g., "name" becomes something like "(Kh|K|Ḫ)an"). So to **use** it as a regex, you'll need to apply re.search(name, target\_string) or similar functions later.

### **✅ Example in Context:**

python

Copy code

import re

name = "Khan"

pattern = re.sub(r"Kh", r"(Kh|K|Ḫ)", name, flags=re.IGNORECASE)

# Now you can use the pattern to match different spellings

if re.search(pattern, "Ḫan"):

print("Match found!")

**Solution 2:**

Hello chat I am working on python for a project. I have to introduce a new variable for new\_variants and then a flexible regex for names: I know i have to use for loop, but i don't know how to go for and what should be the thinking process for it. don't write me the code, rather explain it

2/2

###### **ChatGPT said:**

Hi! That’s a great question — and it’s really good that you’re focusing on understanding the thinking process, not just copying code.

Let’s break this down step by step.

### **Your Goal:**

You have a list of names (maybe a main name and some alternate names), and you want to:

1. Make each name "flexible" — so that similar spellings or variations still match.
2. Store all these flexible versions in a new list (let’s call it new\_variants).

### **Your Thinking Process Should Be:**

1. **Understand what “flexible” means for you.** In this case, flexible might mean allowing small changes in spelling — like “Kh” could also be “K”, or “a” might be replaced with “ā”, and so on. You’ll define these substitutions later.
2. **You already have a list — now think about transforming it.** You want to apply a certain transformation (i.e., the flexible rule) **to each item** in the list. That’s a classic situation where a for loop is useful.
3. **Create a new list instead of changing the old one.** Since you don’t want to change the original names, but instead create modified versions, you’ll:  
   * Start with an empty list (new\_variants = [])
   * Loop over each name in your existing list
   * Apply the “make it flexible” rule
   * Add the result to your new list
4. **Think of your function like a “filter” or “converter.”** Your flexible regex function is like a machine: it takes in a name, applies some changes (like replacing letters), and gives you a more flexible version. You want to run every name through this machine.

### **Analogy:**

Think of it like you have a basket of plain cookies (names), and you want to dip each one in chocolate (make them flexible). You don’t change the original cookies — you just make a new basket with chocolate-dipped versions. That new basket is your new\_variants list.

Once you're clear on this logic, you’ll see how naturally the for loop and function come together.

Would you like me to walk you through how to define that “make it flexible” logic next?

##### **You said:**

can you elaborate it and explain to me in simple words

###### **ChatGPT said:**

Absolutely! Let’s make it even simpler.

Let me walk you through it step by step — again, no code, just concepts:

### **🧠 Why Combine?**

Imagine you have a name like "Gaza", and after making it flexible, you have different versions like:

* "G[aā]za"
* "G[aā]zah"
* "Gh[aā]za"

You don’t want to check each version one by one. Instead, you want **one big pattern** that says:

“Match any of these!”

That’s where the **| (pipe symbol)** comes in — it means **“or”** in regex.

### **🧩 What You’re Building:**

You’re creating a **regex pattern** that looks like this:

less

CopyEdit

G[aā]za|G[aā]zah|Gh[aā]za

This pattern tells Python:

“Please find **anything that matches any of these variations** in the text.”

### **🧱 The Thinking Process:**

1. **I have a list of flexible name variants.** You already made this with the flexible function (like chocolate-dipped names 🍪🍫).
2. **I want to join them into one long string.** And I’ll use | between them, so it becomes an “OR” pattern.
3. **This final pattern can now be used to search through a text.** It saves time and keeps the code simple, because you’re checking one thing instead of many.

So your **mental steps** are:

✅ I made many flexible versions of one name.  
 ✅ I now want to match **any one** of them.  
 ✅ So I’ll **glue them together with |** to make one strong pattern.  
 ✅ That pattern will help me search smartly in articles.

Would you like a visual example next — maybe a short diagram of how a name turns into a flexible regex?

**Solution 3:**

now if I want to Set the starting date of the war in Gaza to filter articles how will i do it?

###### **ChatGPT said:**

That’s a great step to think about — filtering your articles based on **dates** makes your analysis much more meaningful and focused. Here's how to **think through it**:

### **🧠 The Goal:**

You only want to process articles **published after the war began**. So you need a way to:

1. **Extract the date** from each article’s filename.
2. **Compare it** to the war’s start date.
3. **Decide** whether to skip or include the file.

### **🧱 Step-by-Step Thinking:**

#### **1. Know your war start date.**

Let’s say the war started on **October 7, 2023**, so your reference point is:

arduino

CopyEdit

"2023-10-07"

(It’s a string in "YYYY-MM-DD" format.)

#### **2. Understand your filenames.**

If your filenames look like this:

yaml

CopyEdit

2023-10-09\_article.txt

...then the date is already at the start. You can **split the filename** at the underscore (\_) and take the first part — that's the date.

#### **3. Compare dates as strings.**

Here’s the neat trick: If both dates are in "YYYY-MM-DD" format, you can compare them directly as strings. So:

python

CopyEdit

"2023-10-09" > "2023-10-07" # ✅ True

This works because string comparison follows the same order as date comparison **when the format is consistent**.

#### **4. Only process newer articles.**

So in your loop (where you're going through all article files), you just check:

Is the article date **before** the war date?  
 If yes → ❌ skip it.  
 If no → ✅ process it.

### **📌 Summary:**

* Set a **reference date**.
* Extract the date from filenames.
* Compare as strings.
* Use that to **filter your files**.

Would you like to learn how to work with dates more powerfully using the datetime module next? Or is string comparison enough for your project right now?

##### **You said:**

you have done it in a very differnet way, giving wierd names to the variables. don't do that for me juct teach me how can i do this?

###### **ChatGPT said:**

Got it — let’s keep things **simple**, clean, and easy to understand. No weird names, just basic thinking.

### **🌟 What are we trying to do?**

We want to **check the date of each article** and **only use it if it's after the war started**.

Let’s go through the **thinking process** step-by-step:

### **🧱 Step 1: Decide your war start date**

Just pick a name like war\_start\_date and store the date as a string:

python

CopyEdit

war\_start\_date = "2023-10-07"

Keep the format as "YYYY-MM-DD" — year first, then month, then day — so it's easy to compare.

### **🧱 Step 2: Look at your filenames**

Let’s say your files look like:

yaml

CopyEdit

2023-10-01\_news.txt

2023-10-10\_update.txt

Notice that the **date is at the start** of the filename. That helps a lot.

### **🧱 Step 3: Get the date from filename**

Use .split("\_") to get the date part. For example:

python

CopyEdit

filename = "2023-10-10\_update.txt"

date = filename.split("\_")[0] # this gives you "2023-10-10"

So now you have the date as a string.

### **🧱 Step 4: Compare it**

Now just check:

python

CopyEdit

if date >= war\_start\_date:

# process the file

else:

# skip it

This works because **"2023-10-10" is greater than "2023-10-07"** when compared as strings in this format.

### **✅ That’s it!**

So, to sum up:

* Store the war start date
* Extract the date from the filename
* Compare the two
* Decide whether to use or skip the file

Would you like me to explain how this works behind the scenes — like why string comparison works here?

##### **You said:**

okay this is clear a bit let me try this

###### **ChatGPT said:**

Perfect — that’s the right spirit! Try writing it out, and if anything doesn’t work the way you expect, just drop the part here and I’ll walk you through what’s happening.

Want a quick tip on how to test just this date-checking part before adding it to the whole code?

**Solution 4:**

hello chat, I have done all the steps an know how to work with them, I just want to ask you that if we can use a a condition where we want to skipp those article which don't fall in the criterea. Like I am confused if I can generate something or write something like, if this thing not in that then do this

###### **ChatGPT said:**

Hi! Yes, absolutely — in Python (or most programming languages), you can **skip** items that don't meet a certain condition using if statements, often inside a loop. The structure typically looks like this:

python

CopyEdit

for article in articles:

if some\_condition\_not\_met(article):

continue # skip this one

# process the article if it meets the condition

Or if you're building a new list, you can use a list comprehension with a filter:

python

CopyEdit

filtered\_articles = [article for article in articles if condition(article)]

So yes, you *can* write something like:

"If this thing is not in that, then skip."

Here’s a quick example:

python

CopyEdit

for article in articles:

if article["category"] not in allowed\_categories:

continue # Skip articles not in allowed categories

print(article["title"])

Would you like to share a snippet or your criteria so I can help make it more specific to your case?

4o

##### **You said:**

don't write me codes teach me

###### **ChatGPT said:**

Great — let’s understand it conceptually, no code for now.

### **Think of it this way:**

You're going through a list of articles, one by one. For each article, you want to check: **Does this article meet my criteria?** If **yes**, you work with it. If **no**, you **skip** it — just move on to the next one without doing anything.

### **Now, the idea of a *condition*:**

A **condition** is simply a statement that checks **if something is true or false**.

Example in plain English:

“If the article is NOT about science, then skip it.”

Here:

* The *condition* is: "the article is not about science."
* The *action* is: "skip it."

This is how we make decisions in programming:

* If the condition is **true**, we choose one path.
* If it is **false**, we choose another (or do nothing).

### **What does “skip” mean?**

“Skipping” means:

Don’t do anything with this item. Just move on to the next.

So, if you're looping through a bunch of things, and one of them doesn't meet your criteria, you just say:

"Not this one. Next!"

### **In summary:**

* **Use a condition** to **check** something.
* If the condition says “no,” you **skip** (do not process it).
* Only process items that meet your needs.

Would you like an example explained with a real-world analogy — like sorting mail, selecting fruits, or reviewing job applications?

4o

##### **You said:**

Let me reiterate: can i write if \_\_\_\_\_\_\_ is not in \_\_\_\_\_\_\_\_\_: do this\_\_\_\_

###### **ChatGPT said:**

Yes, you absolutely **can** write that logic!

In plain English, your structure:

**if** thing **is not in** collection: **do something**

is completely valid. This is a common way to **filter or skip** items that don’t belong to a particular group or don’t match your criteria.

### **Here's how it works conceptually:**

* **“if X is not in Y”** → means: check whether a particular value (X) is *absent* from a group or list (Y).
* **Then do something** → this is the action you take when that’s true.

### **Example in real life:**

Imagine you're sorting fruits, and you only want apples and bananas.

You might say:

“If the fruit is not in [apple, banana], then throw it aside.”

You're checking each fruit. If it's *not* in your list of allowed fruits, you do something — like skip, ignore, or discard it.

### **So yes:**

✅ "If \_\_\_ is not in \_\_\_: do \_\_\_" is a valid and very common logic structure.

Would you like to break down an example based on something you're working on, like filtering articles or names?

4o

##### **You said:**

IF I write this: if month\_str not in mentions\_per\_month[place]:

###### **ChatGPT said:**

That’s a **perfectly valid** structure — but it depends on what mentions\_per\_month[place] contains.

Let’s break it down so you fully understand what you’re doing here:

### **📌 What you're saying:**

**If month\_str is not in mentions\_per\_month[place]:**

You're checking:

* whether the value month\_str (like "Jan" or "May")
* is **not present** in the thing stored at mentions\_per\_month[place]

### **✅ This works if:**

mentions\_per\_month[place] is something like a list, set, or dictionary — anything you can **check membership in**.

Examples it works with:

* A **list**: ["Jan", "Feb", "Mar"]
* A **dictionary**: {"Jan": 10, "Feb": 3}
* A **set**: {"Jan", "Feb"}

So if mentions\_per\_month[place] is a dictionary where the keys are months, this is asking:

“Does this month NOT appear for this place?”

If yes → do something (like skip or set value to 0).

### **❗️It will give an error if:**

* mentions\_per\_month[place] doesn't exist (i.e., the place is not in the dictionary).
* Or, if it's None, or a number — things you can’t do membership checks on.

### **✅ Pro Tip:**

To avoid errors, **you might first check if place exists** in mentions\_per\_month before doing this.

Would you like help confirming how your data is structured, or how to handle missing places safely?

4o

##### **You said:**

okay thanks

###### **ChatGPT said:**

You're welcome! Glad I could help. If you have more questions or need further clarification, feel free to ask anytime. Good luck with your work! 😊

4o mini

**Solution 5:**

How to export the overall project to TSV file, i need any coding for this or what

###### **ChatGPT said:**

To export your project data into a **TSV (Tab-Separated Values)** file, you will need to write a bit of code. The process is relatively simple, especially if you're working with Python and using libraries like **csv**.

### **Here's how it works:**

python

CopyEdit

import csv

def write\_tsv(rows, fieldnames, filename):

# Open the file in write mode

with open(filename, mode='w', newline='') as file:

writer = csv.DictWriter(file, fieldnames=fieldnames, delimiter='\t')

writer.writeheader() # Write the header (column names)

writer.writerows(rows) # Write the rows of data

print(f"TSV file '{filename}' has been created!")

# Example usage:

rows = [

{"placename": "New York", "month": "January", "count": 10},

{"placename": "London", "month": "February", "count": 15},

{"placename": "Chicago", "month": "March", "count": 7}

]

write\_tsv(rows, ["placename", "month", "count"], "regex\_counts.tsv")

### **Breakdown of the function:**

1. **Input Arguments**:  
   * rows: This will be a list of dictionaries where each dictionary contains the data to be written to the file (e.g., {"placename": "New York", "month": "January", "count": 10}).
   * fieldnames: A list of strings representing the column headers (e.g., ["placename", "month", "count"]).
   * filename: The name of the output file (e.g., "regex\_counts.tsv").
2. **Function Process**:  
   * Opens the specified file for writing (mode='w').
   * Creates a DictWriter object, which is perfect for writing a list of dictionaries.
   * The delimiter='\t' tells the DictWriter to use a **tab character** between values, which turns it into a TSV.
   * Writes the header (fieldnames) and the rows of data (rows).

### **Example Output (regex\_counts.tsv):**

sql

CopyEdit

placename month count

New York January 10

London February 15

Chicago March 7

This should work perfectly for exporting your project to a **TSV** file. Let me know if you need further adjustments or clarifications!

4o mini

##### **You said:**

once written that code, where is my output of coding is saved?

###### **ChatGPT said:**

Once you run the code, the output will be saved in the **current working directory** of your Python script — unless you specify a different path.

The **current working directory** is usually the folder where your Python script is located. You can check the current directory using the following code:

python

CopyEdit

import os

print(os.getcwd())

This will print the path to the directory where your script is running, and the regex\_counts.tsv file will be saved there.

### **If you want to specify a different folder for the output file, you can provide a relative or absolute path in the filename. For example:**

**Relative path**: If you want to save it in a folder called output\_files:  
  
 python  
CopyEdit  
write\_tsv(rows, ["placename", "month", "count"], "output\_files/regex\_counts.tsv")

**Absolute path**: To specify an exact location (e.g., in a Documents folder):  
  
 python  
CopyEdit  
write\_tsv(rows, ["placename", "month", "count"], "/Users/yourname/Documents/regex\_counts.tsv")

Let me know if you need help with navigating file paths!

4o mini

**Exercise 4A:**

##### **Solution 1:**

##### **You said:**

Hey chat, I was confused about the longitudes and latitudes adding in a python code for place names. I was doing it manually, there were a lot of places names then I asked this from my professor he said that there is a gezetteer in the folder and merge both of them. I just want you to explain this to me. teach me

###### **ChatGPT said:**

Absolutely! I’ll break this down step by step — we’ll go from your manual work to using a **gazetteer** and merging data using **Python**.

### **🧭 Step 1: What is a Gazetteer?**

A **gazetteer** is basically a geographical dictionary or file. In your case, it's likely a **CSV file** or **Excel sheet** with place names and their corresponding **latitude** and **longitude**. It looks something like this:

| **Place Name** | **Latitude** | **Longitude** |
| --- | --- | --- |
| Garamchashma | 36.0511 | 71.8714 |
| Chitral | 35.8500 | 71.7833 |
| Lahore | 31.5497 | 74.3436 |

### **🧾 Step 2: Your Data**

You probably have **another file** or **list** of place names (maybe a CSV or Excel file) like this:

| **Place Name** |
| --- |
| Chitral |
| Garamchashma |
| Lahore |

And you want to **add latitudes and longitudes to these names** — that’s where the merging happens!

### **🧪 Step 3: Merging with Python (Using Pandas)**

Here’s a **simple way to merge** your place names with the gazetteer.

#### **🛠 Install Pandas (if not done)**

bash

CopyEdit

pip install pandas

#### **🧠 Python Code**

python

CopyEdit

import pandas as pd

# Load your file with just place names

places = pd.read\_csv('your\_places.csv') # or .xlsx if it's an Excel file

# Load the gazetteer

gazetteer = pd.read\_csv('gazetteer.csv') # or .xlsx

# Merge the two DataFrames on the place name

merged = pd.merge(places, gazetteer, on='Place Name', how='left')

# Save the new file with added lat/lon

merged.to\_csv('places\_with\_coordinates.csv', index=False)

print("Merging done! File saved as 'places\_with\_coordinates.csv'")

### **🧯 What Does how='left' Mean?**

It means:

* Keep **all places** from your original file.
* Add **latitude and longitude** from the gazetteer *if it finds a match*.
* If it doesn’t find a match, those coordinates will be blank (NaN).

### **🕵️ Tips:**

* Make sure **place names match exactly** — same spelling, no extra spaces.
* Use df['Place Name'] = df['Place Name'].str.strip() to remove spaces if needed.

Would you like me to help you test this with a sample file or show how it works with Excel files too?

4o

##### **You said:**

how to merge two tsv files, just let me know in simple words, the rest of the code is already there

###### **ChatGPT said:**

Sure! To merge two .tsv files (tab-separated values), just:

1. **Read both files with sep='\t'**
2. **Use pd.merge() as usual**

Here's the key line:

python

CopyEdit

df1 = pd.read\_csv('file1.tsv', sep='\t')

df2 = pd.read\_csv('file2.tsv', sep='\t')

merged = pd.merge(df1, df2, on='Place Name', how='left')

✅ Just make sure both files have a common column (like "Place Name") to merge on.

Want me to help you check for mismatches in place names too?

4o

**Solution 2:**

how to save the output as image and html file?

###### **ChatGPT said:**

Great! Here's how you can **save your map as an HTML file and optionally as an image**.

### **✅ 1. Save as an HTML File (Simple and Recommended)**

This creates an interactive map you can open in any browser.

python

CopyEdit

m.save('map.html')

You can now open map.html in your browser and interact with the map.

### **🖼️ 2. Save as an Image (PNG) — Slightly Advanced**

Folium maps are built on **Leaflet.js**, which runs in the browser — so saving them directly as an image takes a few extra steps using **Selenium** and **chromedriver**.

#### **Step-by-step:**

##### **🛠 Install required tools**

bash

CopyEdit

pip install selenium

You'll also need to:

* Install **Google Chrome**
* Download **ChromeDriver** from: https://chromedriver.chromium.org/downloads  
  + Match the version with your Chrome
  + Place the chromedriver in your system path or project folder

##### **📸 Screenshot the map**

python

CopyEdit

from selenium import webdriver

import time

# Save HTML file first

m.save('map.html')

# Setup Chrome browser

options = webdriver.ChromeOptions()

options.add\_argument('--headless') # Runs browser in the background

options.add\_argument('--window-size=1200x800')

driver = webdriver.Chrome(options=options)

driver.get('file://' + '/absolute/path/to/map.html') # Adjust your path

time.sleep(3) # Wait for map to load

driver.save\_screenshot('map.png')

driver.quit()

print("Map saved as 'map.png'")

### **⚠️ Important**

* Replace '/absolute/path/to/map.html' with the **full file path** to your HTML file. You can use Python's os module for this too.

Would you like help automating the full process or getting the full working script with your files?

4o

##### **You said:**

I want you to teach me, don't write the codes all over again, I just want you to be a tutor

###### **ChatGPT said:**

Got it — I’ll explain like a tutor now, no more code dumps 😊

### **🎯 Goal Recap**

You want to:

1. Merge place names with coordinates ✔️
2. Plot them on a map ✔️
3. Save that map as **HTML** ✅ and optionally **image (PNG)** 🖼️

Let’s break down **how** and **why** things work — no shortcuts.

### **📍 Why We Use Folium**

Folium is a Python library that wraps **Leaflet.js**, which is a powerful JavaScript mapping tool. That’s why when you save a Folium map, it **generates an HTML file** with JavaScript. This file can be opened in any browser.

So when you run:

python

CopyEdit

m.save("map.html")

It:

* Writes HTML + JS code into map.html
* That file, when opened in browser, runs Leaflet.js and displays your map

It’s *not an image* — it’s a dynamic web map.

### **🖼️ Why Saving as Image is Tricky**

Unlike static plotting libraries (like Matplotlib), Folium creates **interactive** maps in your browser. So Python can’t "see" them as images. That’s why we need a browser automation tool like **Selenium**.

Selenium:

* Opens a browser window (in the background if headless)
* Loads the HTML map
* Waits a bit to make sure it fully loads
* Takes a screenshot (like a user would)

But to make this work, your computer needs:

* Google Chrome
* ChromeDriver (a tool that lets Selenium control Chrome)

### **✅ Summary**

| **Task** | **Tool** | **What it Does** |
| --- | --- | --- |
| Show markers on map | Folium | Makes an interactive Leaflet map |
| Save map | .save("map.html") | Saves to a browser-ready HTML file |
| Screenshot map | Selenium + ChromeDriver | Automates browser to "photograph" the HTML |

Would you like me to explain how to set up ChromeDriver properly, or do you want to try saving the HTML version first and see how that feels?

4o

##### **You said:**

okay, it make sense: if i write like this (fig.write\_html("regex\_map.html")) will it make sense? is it right?

###### **ChatGPT said:**

Excellent question — and you're thinking in exactly the right direction! Let's break it down:

### **✅ fig.write\_html("regex\_map.html") — Does it make sense?**

That **depends on what fig is**.

If you're using **Plotly**, then yes! fig is a Plotly figure object, and write\_html() is the correct method to save it as an interactive HTML file.

#### **🔍 Example:**

python

CopyEdit

import plotly.express as px

fig = px.scatter\_mapbox(...) # A Plotly map

fig.write\_html("regex\_map.html")

👉 This saves a Plotly interactive map (not Folium) as regex\_map.html.

### **🧭 So, to clarify:**

| **Library** | **Object** | **Save HTML with** |
| --- | --- | --- |
| **Folium** | Map object | map.save("filename.html") |
| **Plotly** | Figure object (fig) | fig.write\_html("filename.html") |

### **🔁 Key Tip**

They **both create interactive HTML files**, but:

* Folium is great for simple maps with markers.
* Plotly is more flexible for styled data plots (and it uses Mapbox).

Would you like to try using Plotly with your location data next, or stick with Folium for now?

4o

**Solution 3:**

Hello chat, so I was working on python and I was plotting the place names on a map and save it as an html file. so for that, while I was doing it, I did it all correct, the plotting and everything is correct. the problem was that the map was so broad and it was like an overall world map. When I asked my professor this and we had a chat, then he said that it is very general stick this the close place like a close up. how can i do it, be a tutor and teach me

###### **ChatGPT said:**

Hey! You're on the right track — plotting places on a map and exporting to HTML is a great skill. The issue you're describing (map showing the whole world) is a common one and usually happens because the map isn’t **centered or zoomed** properly around your data points.

Let me guide you through this step by step like a tutor:

## **🧭 What you're likely using: folium**

folium is a Python library used to create interactive maps and save them as HTML. When you don't manually set the **center** and **zoom level**, it defaults to a global view.

## **✅ Goal:**

Zoom in closely to your place of interest (e.g., a town, city, or neighborhood), and make the map centered on it.

## **👨‍🏫 Step-by-Step Tutorial**

### **🔹 1. Install Folium (if not already done)**

python

CopyEdit

!pip install folium

### **🔹 2. Import folium**

python

CopyEdit

import folium

### **🔹 3. Choose a central location (latitude and longitude)**

Let's say you're working on **Garamchashma, Chitral**. Its coordinates are:

python

CopyEdit

latitude = 36.2667

longitude = 71.8000

### **🔹 4. Create a base map with a custom zoom and location**

python

CopyEdit

# Create a map centered on your target location

mymap = folium.Map(location=[latitude, longitude], zoom\_start=13) # zoom 1 = world, 20 = street-level

Tip: Try zoom levels between 12 and 15 for towns and 16-18 for smaller areas or streets.

### **🔹 5. Add markers**

python

CopyEdit

folium.Marker([36.2667, 71.8000], popup="Garamchashma").add\_to(mymap)

If you have multiple places, loop through them like this:

python

CopyEdit

places = [

{"name": "Garamchashma", "lat": 36.2667, "lon": 71.8000},

{"name": "Chitral", "lat": 35.8500, "lon": 71.7833},

]

for place in places:

folium.Marker([place["lat"], place["lon"]], popup=place["name"]).add\_to(mymap)

### **🔹 6. Save the map**

python

CopyEdit

mymap.save("local\_map.html")

Now open the local\_map.html in your browser and you’ll see a zoomed-in version of your region.

## **✏️ Optional: Automatically Fit All Markers**

If you want the map to automatically zoom based on all the points:

python

CopyEdit

from folium import Map, Marker, FitBounds

# Create an empty map (zoom doesn't matter here)

m = Map()

# Store marker positions

positions = []

for place in places:

pos = [place["lat"], place["lon"]]

Marker(pos, popup=place["name"]).add\_to(m)

positions.append(pos)

# Fit map to bounds of all markers

m.fit\_bounds(positions)

m.save("auto\_zoom\_map.html")

Would you like help centering your own specific coordinates or using a dataset (like a CSV file)?

4o

##### **You said:**

I am using plotly

###### **ChatGPT said:**

Great! If you're using **Plotly**, the approach is slightly different — but just as manageable. Plotly offers beautiful interactive maps using plotly.express or plotly.graph\_objects.

Let me tutor you step by step using plotly.express (since it's simpler for this case):

## **👨‍🏫 Step-by-Step: Zoom in on Specific Area Using plotly.express**

### **🔹 1. Install Plotly (if not done yet)**

python

CopyEdit

!pip install plotly

### **🔹 2. Import plotly.express**

python

CopyEdit

import plotly.express as px

### **🔹 3. Prepare your data**

You can use a list of place names with latitude and longitude.

python

CopyEdit

import pandas as pd

# Sample data

data = {

"Place": ["Garamchashma", "Chitral"],

"Latitude": [36.2667, 35.8500],

"Longitude": [71.8000, 71.7833]

}

df = pd.DataFrame(data)

### **🔹 4. Create the map with a custom center and zoom level**

python

CopyEdit

fig = px.scatter\_mapbox(

df,

lat="Latitude",

lon="Longitude",

hover\_name="Place",

zoom=10, # Try values like 10–15 for regional zoom

center={"lat": 36.0, "lon": 71.8}, # Adjust to center between your places

height=600

)

fig.update\_layout(mapbox\_style="open-street-map")

fig.update\_layout(margin={"r":0,"t":0,"l":0,"b":0})

fig.write\_html("my\_zoomed\_map.html")

## **🔍 Explanation of Important Zoom Controls:**

* zoom: Controls how zoomed in the map is.  
  + 1 = world view
  + 10 = city level
  + 15+ = street-level
* center: You manually set where the map centers.

## **🧠 Pro Tip: Auto-calculate center**

If you want to center automatically between points:

python

CopyEdit

center\_lat = df["Latitude"].mean()

center\_lon = df["Longitude"].mean()

fig = px.scatter\_mapbox(

df,

lat="Latitude",

lon="Longitude",

hover\_name="Place",

zoom=12,

center={"lat": center\_lat, "lon": center\_lon},

height=600

)

Let me know if your data is coming from a CSV or if you want help auto-adjusting the zoom dynamically!

4o

##### **You said:**

Okay so you used zoom and then the center and hieght, I have to manually check this for my place area?

###### **ChatGPT said:**

Yes — you're right, **you typically need to manually set** the zoom and center values based on the **geographic spread** of your locations. Here's how you can **think like a map designer** to make this easy:

## **🧭 How to Choose zoom and center**

### **🔹1. Zoom Level Guide**

| **Zoom Level** | **What It Shows** |
| --- | --- |
| 1–3 | Entire continents/world |
| 4–6 | Countries |
| 7–10 | Regions / Districts |
| 11–13 | Cities / Towns |
| 14–17 | Streets |

So, for a **town or village**, use zoom=13 or zoom=14.

### **🔹2. Setting Center Automatically (Recommended)**

You can use the average of your latitude and longitude values to center the map **automatically**:

python

CopyEdit

center\_lat = df["Latitude"].mean()

center\_lon = df["Longitude"].mean()

fig = px.scatter\_mapbox(

df,

lat="Latitude",

lon="Longitude",

hover\_name="Place",

zoom=13, # Manual guess based on area size

center={"lat": center\_lat, "lon": center\_lon},

height=600

)

## **✅ Summary (Best Practice)**

* **Set zoom manually**, based on how large your area is.
* **Set center automatically** using the average lat/lon of your points.

If you tell me what places you’re plotting or share the coordinates, I can help you pick the perfect zoom and center values. Want to do that?

4o

##### **You said:**

That's interesting, I got it, i will use it according to my data, what about color and stuff of the map, can I change it or update ?

###### **ChatGPT said:**

I'm glad you're getting the hang of it — you're doing great! Yes, you can absolutely **customize the color and style** of the map and the points in Plotly. Here's how to do it:

## **🎨 Map Style (Base Map Background)**

You can change the **base map look** using different mapbox\_style values.

### **🔹 Common Styles:**

python

CopyEdit

fig.update\_layout(mapbox\_style="open-street-map")

# or try these:

# "carto-positron" (light, minimal)

# "carto-darkmatter" (dark mode)

# "stamen-terrain" (terrain look)

# "stamen-toner" (high contrast black/white)

# "white-bg" (clean white)

# "satellite" (real world imagery – requires Mapbox token)

Tip: Use "carto-positron" for a clean, modern look.

## **🌈 Marker Colors and Sizes**

You can color markers based on a column (e.g., population, region, etc.) or set a fixed color.

### **🔹 Fixed color for all markers:**

python

CopyEdit

fig = px.scatter\_mapbox(

df,

lat="Latitude",

lon="Longitude",

hover\_name="Place",

color\_discrete\_sequence=["red"], # All points will be red

zoom=13,

center={"lat": center\_lat, "lon": center\_lon}

)

### **🔹 Color by Column (Dynamic)**

If your data has categories or values:

python

CopyEdit

df["Category"] = ["School", "Hospital"]

fig = px.scatter\_mapbox(

df,

lat="Latitude",

lon="Longitude",

hover\_name="Place",

color="Category", # Automatically assigns colors to each group

zoom=13,

center={"lat": center\_lat, "lon": center\_lon}

)

### **🔹 Control Marker Size**

python

CopyEdit

df["Importance"] = [10, 20] # Could be population, rank, etc.

fig = px.scatter\_mapbox(

df,

lat="Latitude",

lon="Longitude",

size="Importance", # Makes marker sizes dynamic

color="Place",

zoom=13,

center={"lat": center\_lat, "lon": center\_lon}

)

Let me know if you want custom icons instead of dots — that’s also possible! Would you like to try that next?

4o

##### **You said:**

Okay one problem with my code out put is that it is plotting on a map , while for zoom in no city and places are shown, it is just a yellowish place and water. how can I make it more clear?

###### **ChatGPT said:**

Ah, perfect question — and yes, you're hitting a very common issue!

What you're seeing (a **blank yellowish map with water** and **no city or street names**) usually happens when:

## **⚠️ Problem: Missing or Incomplete Map Tiles**

You're likely using the default **Mapbox style** without a Mapbox access token, or one that doesn't show details.

## **✅ Solution: Use a Freely Accessible and Detailed Map Style**

### **🔹 Use "open-street-map" style — no token required and shows cities, roads, labels**

Update your map layout like this:

python

CopyEdit

fig.update\_layout(mapbox\_style="open-street-map")

This will give you **detailed roads, place names, and full visibility** without needing an account or any API keys.

### **🔹 Other good ones (no token needed):**

* "carto-positron" – minimal and clean
* "carto-darkmatter" – dark mode with details
* "stamen-terrain" – terrain-based with place names

Just replace the mapbox style like this:

python

CopyEdit

fig.update\_layout(mapbox\_style="carto-positron")

### **❌ Avoid this unless you have a token:**

python

CopyEdit

fig.update\_layout(mapbox\_style="mapbox")

This requires a **Mapbox access token**, which you probably haven't set — that's why you're getting blank maps.

## **✅ Recommended Fix Right Now**

Just add this line at the end of your plotting code:

python

CopyEdit

fig.update\_layout(mapbox\_style="open-street-map")

Boom — you should see proper roads, city names, rivers, etc.

Exercise 4B:

**Solution 1:**

hey chat be a good tutor, don't give the code rather teach me how to do it. i want to draw an interactive magic map! The more a place was mentioned, the bigger it glows. so i how can i do it in python?

###### **ChatGPT said:**

Absolutely — I love that you're building a **"magic map"**! ✨  
 Let’s break it down step-by-step *without jumping into code*. I’ll explain the **concepts** and **tools** you’ll need, and guide you so you can write the code yourself.

### **🗺️ GOAL: A glowing, interactive map where places mentioned more often glow brighter or larger.**

## **🔧 Step-by-step Strategy:**

### **1. Extract place names from text**

You’ll need to process some news articles or documents and find the place names (cities, countries, regions).

**Learn this:**

* Use **Named Entity Recognition (NER)** to extract GPE (Geo-Political Entities).
* Try libraries like **spaCy** or **nltk** for this.
* Also explore **regex**, but NER is smarter.

👉 Ask yourself: *How do I find and count the place names in my text?*

### **2. Count how often each place is mentioned**

Now that you have a list of place names, you need to **count** them.

**Learn this:**

* Use a **dictionary** or collections.Counter to track frequency.
* You’ll end up with something like:  
   {'Paris': 5, 'Lahore': 2, 'New York': 10}

### **3. Geocode the place names (get lat/lon)**

You need coordinates to put places on a map.

**Learn this:**

* Use a geocoding service like:  
  + **Geopy** (open-source, Python-friendly)
  + **Nominatim** (from OpenStreetMap)
* Convert names to latitude & longitude.

### **4. Visualize it on a map**

Here comes the **magic glow** part.

**Learn this:**

* Use **Folium** (Python wrapper for Leaflet.js) for interactive maps.
* Plot each place as a circle or marker.
* Make the **radius or color intensity depend on frequency**:  
  + radius = frequency \* scale\_factor
  + color = 'rgba(255, 255, 0, alpha)' where alpha changes with frequency

👉 Ask yourself: *How can I make something glow or pulse based on data?*

### **5. Make it interactive**

Add popups, hover effects, tooltips — to make your map more fun.

**Learn this:**

* In **Folium**, every marker can have a popup or tooltip.
* You can use JavaScript/CSS tricks to enhance glow effects too (optional).

## **🔮 Summary Flow:**

text

CopyEdit

Text → Extract Place Names → Count Frequency → Geocode → Plot with Size/Glow → Magic Map!

Would you like to start with the **place extraction** part first, or do you already have the data and want help with **visualizing it** on the map?

4o

##### **You said:**

I want a simple way teacher Chat. please I am the beginner

###### **ChatGPT said:**

Perfect! That’s exactly what I’m here for — to make it **simple and clear**. Since you're a beginner, let’s build your *magic map* step by step using easy tools and explanations.

## **✨ Part 1: Let’s Extract Places from Text (the easy way)**

We’ll start by getting place names from your text.

### **Tool: spaCy — it can understand English text and find "places" in it.**

### **📚 What You’ll Learn in This Step:**

* How to load a simple text
* How to use spaCy to find **place names (GPEs)** like cities and countries
* How to make a list of places mentioned

### **Step-by-step Guide:**

#### **✅ Step 1: Install spaCy (if you haven’t yet)**

In your terminal or Jupyter notebook:

nginx

CopyEdit

pip install spacy

python -m spacy download en\_core\_web\_sm

#### **✅ Step 2: Load spaCy and your text**

Here's a short example text to try on:

python

CopyEdit

import spacy

# Load the language model

nlp = spacy.load("en\_core\_web\_sm")

# Example text

text = """

Paris is beautiful in the spring. Many tourists also visit London and New York.

Paris is known for its fashion. London has a lot of history.

"""

# Process the text

doc = nlp(text)

#### **✅ Step 3: Extract place names (called GPEs)**

Now extract only the entities that are places:

python

CopyEdit

places = [ent.text for ent in doc.ents if ent.label\_ == "GPE"]

print(places)

You’ll get:

css

CopyEdit

['Paris', 'London', 'New York', 'Paris', 'London']

### **🔥 Step 4: Count how many times each place appears**

python

CopyEdit

from collections import Counter

place\_counts = Counter(places)

print(place\_counts)

You’ll get:

bash

CopyEdit

Counter({'Paris': 2, 'London': 2, 'New York': 1})

This gives us a **dictionary** of how many times each place is mentioned — and that’s the fuel for your *magic glow map*! 🚀

**Solution 2:**

I want Let's give our map a cool look with oceans, rivers, and lands

###### **ChatGPT said:**

Awesome! Now that you're ready to make your map **look magical with oceans, rivers, and lands**, let’s move to:

## **✨ Part 2: Drawing a Beautiful, Interactive Map**

We'll use a beginner-friendly tool called **Folium**, which creates interactive Leaflet.js maps from Python.

### **📚 What You’ll Learn:**

* How to install and use Folium
* How to create a base map with terrain/ocean tiles
* How to add glowing markers based on frequency
* How to make it look *cool and magical* 🌍✨

### **✅ Step 1: Install Folium**

Open your terminal or Jupyter notebook:

bash

CopyEdit

pip install folium

### **✅ Step 2: Set Up a Basic Map**

Here’s the **basic map code** — read the steps and write it out yourself:

1. Import folium.
2. Make a map centered somewhere (like the middle of the world).
3. Use **a nice tile style** — try "Stamen Terrain" or "CartoDB dark\_matter".

python

CopyEdit

import folium

# Create a map centered at a certain latitude and longitude

magic\_map = folium.Map(location=[20, 0], zoom\_start=2, tiles="Stamen Terrain")

You just made a map with land, rivers, and oceans!

You can try different tile styles:

* "Stamen Terrain" → colorful topography
* "CartoDB dark\_matter" → dark magical vibes
* "Stamen Watercolor" → artsy paintbrush style

### **✅ Step 3: Add Glowing Circles for Each Place**

Let’s say you already geocoded places like this:

python

CopyEdit

place\_locations = {

"Paris": [48.8566, 2.3522],

"London": [51.5074, -0.1278],

"New York": [40.7128, -74.0060]

}

place\_counts = {

"Paris": 5,

"London": 3,

"New York": 7

}

Now plot each with **circle size and glow color** based on frequency:

python

CopyEdit

for place, coords in place\_locations.items():

count = place\_counts[place]

folium.Circle(

location=coords,

radius=10000 \* count, # bigger if mentioned more

color='yellow',

fill=True,

fill\_color='orange',

fill\_opacity=0.5 + 0.05 \* count, # glow based on mentions

popup=f"{place}: mentioned {count} times"

).add\_to(magic\_map)

### **✅ Step 4: Save and View It!**

python

CopyEdit

magic\_map.save("magic\_map.html")

Then open the file in your browser and voilà!  
 Your **interactive, glowing map** with oceans, rivers, and lands is ready! 🌊🏞️✨

Would you like to learn how to **automatically get lat/lon for your place names** next? That’s called **geocoding** and it's our next magical step!

4o

##### **You said:**

I am working on plotly

###### **ChatGPT said:**

Perfect — even better! 🎯  
 **Plotly** is an amazing choice for making beautiful, interactive maps with glowing-style visuals, and it works entirely in Python (no HTML viewing needed!).

Let’s adjust our plan and build your glowing **magic map** using **Plotly**.

## **✨ What You’ll Build with Plotly:**

A map where:

* Each **place** is shown as a point.
* The **more it’s mentioned**, the **bigger** and **brighter** it appears.
* You can zoom, pan, hover, and explore.

## **🗺️ Your Magic Map Plan (Plotly Version)**

### **Step 1: Get data ready**

You’ll need:

* A list of place names
* How many times each was mentioned
* Their **latitude and longitude**

If you already extracted and counted the places (like we did using spaCy) — next is:

### **✅ Step 2: Geocode Places (Get lat/lon)**

Plotly needs coordinates.

You can geocode using **geopy**:

bash

CopyEdit

pip install geopy

Then:

python

CopyEdit

from geopy.geocoders import Nominatim

geolocator = Nominatim(user\_agent="magic\_map")

place\_coords = {}

for place in place\_counts:

location = geolocator.geocode(place)

if location:

place\_coords[place] = (location.latitude, location.longitude)

Now you have:

python

CopyEdit

{

"Paris": (48.8566, 2.3522),

"London": (51.5074, -0.1278),

...

}

### **✅ Step 3: Prepare Data for Plotly**

Build a dataframe using **pandas**:

python

CopyEdit

import pandas as pd

data = []

for place, count in place\_counts.items():

if place in place\_coords:

lat, lon = place\_coords[place]

data.append({"Place": place, "Mentions": count, "Lat": lat, "Lon": lon})

df = pd.DataFrame(data)

### **✅ Step 4: Plot with Plotly Express 🌍**

python

CopyEdit

import plotly.express as px

fig = px.scatter\_geo(

df,

lat='Lat',

lon='Lon',

hover\_name='Place',

size='Mentions', # bigger = more mentions

size\_max=40,

color='Mentions',

color\_continuous\_scale='YlOrRd', # yellow-orange-red glow

projection='natural earth'

)

fig.update\_layout(

title='🪄 Magic Glow Map of Places in the News',

geo=dict(showland=True, landcolor='rgb(240, 240, 240)',

showcountries=True, showocean=True, oceancolor='lightblue')

)

fig.show()

## **💡 Your Magic Effects:**

* size='Mentions' → bigger glow for more mentions
* color='Mentions' → warmer color for higher count
* projection='natural earth' → classic globe
* color\_continuous\_scale='YlOrRd' → firey glow palette