# Conversation: 1

“I'm a student doing a Python project in a DH course. Act as a good tutor and give me advice without giving me the full answers.” Do not give me codes directly, help me understand the logic.

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

Absolutely, Bushra! I’ll act as a tutor who helps **guide your thinking** without giving away the answers or code.

Whenever you're stuck, here's how we'll work:

* I'll **ask you questions** to get you to think through the logic.
* I might give you **hints or pseudocode** if you're totally stuck, but never full code.
* We'll stay within the logic and methods taught in your class.

So tell me:  
 👉 What part of your project or script are you currently working on or stuck with? What do you want to happen next in your analysis?

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

I have two separate lists in Python one that contains terms related to people and another with terms related to power. I want to learn how to combine them into a single list that includes all the terms. Can you explain how this is typically done in Python and why it works? without giving me the code

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

Great question — this is a very common task in Python, and understanding the logic behind it is super helpful.

### **🔍 Let’s think about what you have:**

* You have **two separate lists**.  
  + One might look like: ['children', 'families', 'victims'] (people-related terms).
  + Another like: ['idf', 'government', 'israel'] (power-related terms).

You want to **combine them into one list** that holds **all** the terms, right?

### **✅ The logic behind combining lists:**

Python lists are **ordered collections**, so when you combine two lists, you’re just **appending the elements of one list to the end of the other**.

The result is a **new list** that contains:

1. The items from the first list, in their original order.
2. Then, the items from the second list, in their original order.

This works because lists are iterable and Python provides a very intuitive way to **concatenate** them — that is, to place one list **after** another.

### **🧠 Tutor prompt:**

Can you think of an operator or method in Python that is used when we want to join two things together — like joining two strings or two lists?

What would happen if you applied that to your two lists?

Try testing that in a notebook or console and let me know what you see. I'm here to guide you if the result isn't what you expected.

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

I was thinking of simply adding the two lists using + operation

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

🎯 Exactly — that’s **the right intuition**!

Using the + operator with lists in Python doesn’t do math like with numbers — instead, it **concatenates** them. That means it creates a **new list** with the elements of the first list followed by the elements of the second.

### **✅ Why it works:**

* Lists are **sequences** in Python, just like strings.
* The + operator tells Python:  
   ➕ “Take this sequence, and add the contents of this other sequence to the end of it.”

This **doesn’t modify** the original lists unless you explicitly overwrite one of them. It just returns a new list containing all the elements.

### **🧠 Tutor Check-In:**

Try printing the result of adding your two lists like:

scss

CopyEdit

print(list1 + list2)

Then check:

* Does the output include all the terms?
* Are they in the order you expect?
* Did the original list1 or list2 change?

Let me know what you observe, or if you want to merge and save them into one variable.

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

Will this work? keywords = people\_terms + power\_terms

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

Yes — that’s **perfectly correct logic** ✅

Let’s break it down so you’re crystal clear on **why it works**:

### **🔍 What’s happening here?**

You wrote:

python

CopyEdit

keywords = people\_terms + power\_terms

Here’s what’s happening behind the scenes:

1. people\_terms is one list.
2. power\_terms is another list.
3. The + operator **concatenates** them — it takes all elements from people\_terms and adds all elements from power\_terms to the end.
4. The result is a **new list**, and you're assigning that to the variable keywords.

### **🔁 Is this destructive?**

Nope — it’s not modifying either original list. You’re just **creating a new one**.

### **🧠 Quick Tutor Tip:**

Try printing each list *before* and *after* this operation to confirm they remain unchanged.

python

CopyEdit

print(people\_terms)

print(power\_terms)

print(keywords)

That’ll help reinforce the idea that list concatenation with + is **non-destructive** unless you explicitly overwrite one of the lists.

Let me know what you observe or if you want to explore what happens when you use += instead!

# Conversation 2:

I have a pandas DataFrame with several topic columns like topic\_1, topic\_2, topic\_3, and topic\_4. I want to create a new column that builds a combined label by adding each topic one by one. Help me understand the logic, Do not give the code.

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

You're asking an excellent data wrangling question — and this is exactly the kind of task that's common in text and topic modeling work.

Let’s walk through the logic step by step, without writing code:

### **🔍 What you're trying to do:**

You have multiple columns in a DataFrame, each containing a **part of a topic label**, and you want to create a **single new column** that **combines all those parts** into a single string per row.

### **💡 Step-by-step logic:**

1. **Iterate across columns**:  
   * For each row, you need to access the values in topic\_1, topic\_2, etc.
2. **Combine values into a string**:  
   * Treat each value in those columns as a piece of a final label.
   * You want to **concatenate** them, usually with a space or hyphen or comma in between.
3. **Create a new column**:  
   * Store the combined string for each row in a new column, let’s say called combined\_label.

### **⚠️ Things to consider:**

* What if one of the topic columns is NaN or empty? Do you want to skip it, or include it as 'nan' or an empty string?
* What separator do you want between topics? Spaces? Hyphens? Something else?
* Are the topic columns always the same number, or do you want your logic to handle a flexible number of columns?

### **🧠 Tutor Prompt:**

Imagine doing this manually for a single row:  
 If topic\_1 = 'conflict', topic\_2 = 'military', topic\_3 = 'policy', topic\_4 = 'gaza'  
 What would the combined label look like?

Now, how would you repeat that logic for every row in the DataFrame?

Try writing out in plain English what you'd want Python to do — then I’ll help you refine the logic into code later, if needed.

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

So I know that I can use "astype(str)" this function to convert the data into columns, and also I can use ", " to put a comma and a space between the two strings. I want you to guide me on how to use this I want to create a new column that builds a combined label by adding each topic one by one. Do not give me the code.

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

You're 100% on the right track — you're thinking like a data scientist now.

Let me guide you through the full **logic using your tools (astype(str) and ", ")** — step by step.

### **🧠 Your goal:**

Create a new column (let’s say combined\_label) where:

* Each row contains a **single string** built by **joining** the values from topic\_1, topic\_2, topic\_3, and topic\_4.
* You want those values separated by ", " — a comma and a space.

### **🧩 Step-by-step thinking:**

#### **1. Convert each topic column to string**

Why?

* Because if any of the topic values are numeric or null, they’ll cause errors or unexpected results when combining.
* astype(str) ensures each value is treated as a string, even if it's a number or NaN.

So for each topic column, imagine you're doing:

* “Take the value and treat it as a string, no matter what.”

#### **2. Join the values together**

How?

* Once you have string versions of each column, you can “add” them together using the + operator with ", " in between.
* This means:  
   string version of topic\_1 + ", " + string version of topic\_2 + ", " + ... and so on.

So for every row:

* You're saying: “take this, then this, then this, glue them together with commas.”

#### **3. Assign to a new column**

* Once you build the combined string for each row, you store that result in a new column of the DataFrame.
* That column should have one combined label per row.

### **❓ Reflective check:**

What happens if one of the topics is NaN?  
 Try it on a row manually and see what the combined label looks like — is that okay for your project?  
 If not, you might need to clean or filter NaN values out **before** or **during** this process.

Let me know how it goes when you try this logic — I’ll guide you through the next step if anything looks off.

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

will this code work?? df\_filtered["Topic\_label"] = df\_filtered["topic\_1"].astype(str)

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

Great question! Let’s think about what this line does logically:

### **What does this do?**

* df\_filtered["topic\_1"].astype(str)  
   This **converts** the entire "topic\_1" column **to strings** — so every value in that column, no matter its original type, becomes a string.
* Then you assign that result back to a **new column** "Topic\_label" in df\_filtered.

### **What does that mean for your new column?**

* The new "Topic\_label" column will contain **only the string version of topic\_1**.
* It **does not** combine topic\_1 with any other topic columns.
* So, **this line alone won’t create the combined label you want** — it just copies topic\_1 (as strings) into the new column.

### **🧠 Tutor prompt:**

* How might you extend this logic so that you **combine** topic\_1 and topic\_2 (both as strings) into Topic\_label?
* Remember your earlier thought about using + and ", " to join strings.

Try writing out in words how you’d do that, and I can help you refine it!

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

df\_filtered["Topic\_label"] = df\_filtered["topic\_1"].astype(str) df\_filtered["Topic\_label"] = df\_filtered["Topic\_label"] + ", " + df\_filtered["topic\_2"].astype(str) and do the same for topic 2 and topic 3 Am I on the right track?

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

Absolutely, Bushra — you’re definitely on the right track! 🎉

Here’s the **logic you’re using**, and why it works well:

### **What you’re doing step-by-step:**

1. First, you create a new column "Topic\_label" that **starts with** the string version of topic\_1.
2. Then, you **update** "Topic\_label" by **adding** a comma and space ", " plus the string version of topic\_2.
3. You plan to repeat this for topic\_3 (and topic\_4 if you want), so the string builds up piece by piece.

### **Why this works:**

* Every time you add + ", " + next\_topic, you’re **concatenating** strings — so your combined label grows, with each topic separated by , .
* You are **reusing** the updated "Topic\_label" column at each step, which means you keep adding to it.

### **🧠 Tutor tip:**

* Keep in mind if any topic column has missing values (like NaN), converting them to strings will turn them into 'nan', so your combined label might look like "conflict, nan, policy".
* If you want to avoid this, you might want to think about handling missing data before or during concatenation (like skipping NaNs or replacing them with empty strings).

### **Final check:**

* Does your DataFrame have **all four topic columns** with consistent names?
* How will you extend this logic for topic\_3 and topic\_4? Will you repeat the same operation two more times?

If you want, I can help you outline the full sequence in pseudocode to be extra clear!

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

df\_filtered["Topic\_label"] = df\_filtered["topic\_1"].astype(str) df\_filtered["Topic\_label"] = df\_filtered["Topic\_label"] + ", " + df\_filtered["topic\_2"].astype(str) df\_filtered["Topic\_label"] = df\_filtered["Topic\_label"] + ", " + df\_filtered["topic\_3"].astype(str) df\_filtered["Topic\_label"] = df\_filtered["Topic\_label"] + ", " + df\_filtered["topic\_4"].astype(str)

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

You’ve nailed the logic here! This approach will build your combined label step-by-step, adding each topic separated by a comma and space. Nice!

### **Just a couple of things to keep in mind:**

1. **What happens with missing data (NaN)?** Since you convert everything to string, missing values become the string 'nan' in your combined label.  
    Is that what you want? Or do you want to avoid those 'nan' parts?  
    If avoiding them, you might want to think about cleaning or filtering those before combining.
2. **Efficiency & readability:** Reassigning the column four times works fine, but if you had more topics or wanted a cleaner approach, you could think about joining all at once (like using a method to join multiple columns in one go).

### **Quick check for you:**

* Run this on a small sample and check how the combined labels look, especially rows with missing topics.
* Does the output match your expectations?
* If not, what would you want to improve or change?

Feel free to ask if you want ideas on how to handle those missing values gracefully!

# Conversation 3:

"I have a DataFrame with a column of month numbers like 10, 11, and 12, but I want to show these months as their names (like 'Oct', 'Nov', 'Dec') for better visualization and to make my charts easier to understand. I’m not familiar with how to convert or replace these numbers with names in pandas. Could you please guide me from the basics on how to do this? Do not give me the exact code

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

Absolutely! Let’s break down the process logically so you understand how to convert month numbers into their names in pandas.

### **🔍 What you have:**

* A DataFrame with a column containing month numbers: for example, 10, 11, 12.
* You want to **replace or map these numbers** to their **month names** like 'Oct', 'Nov', 'Dec'.

### **💡 Core idea:**

You want to **translate numeric values** into **string labels** that are more meaningful and readable.

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

1. **Understand your data:**
   * Your column contains integers representing months (1 to 12).
   * Each number corresponds to a month name.
2. **Create a mapping from number to name:**
   * Think of a **dictionary** or a **list** where:  
     + Keys (or indices) are the month numbers.
     + Values are the corresponding month names (like 'Jan', 'Feb', …, 'Dec').
3. **Apply this mapping to your DataFrame column:**
   * Use a method that replaces each number with its mapped name.
   * This can be done either by **mapping** the column values through your dictionary or by **indexing** into a list of month names.
4. **Result:**
   * Your original column is transformed from numbers to strings representing month names.
   * This makes your visualizations and reports easier to read.

### **🧠 Tutor prompt:**

* Do you know how Python dictionaries work? How can you use them to replace values in a DataFrame column?
* Can you think about how you might build such a dictionary for months?
* Once you have the dictionary, how could you tell pandas to replace the numbers using that dictionary?

Try to imagine this in plain English first, then I can help you turn it into the right pandas operations!

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

yes i know how to use dictionaries: Here is how we will do: months = {10: 'Oct', 11: 'Nov', 12: 'Dec'}

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

Perfect! You’ve got the dictionary part nailed — that’s exactly how you’d map numbers to names.

### **Now, the next logical step is:**

How to **apply this dictionary to your DataFrame column** so that each number gets replaced by its matching month name.

Think about these options in pandas:

* **Mapping** the dictionary over the column — pandas has a built-in way to do this by matching keys to values.
* Using a **replace** method that can substitute values based on a dictionary.
* Or, if you want, you could write a function to convert each value, then apply it across the column — but mapping or replacing is more straightforward here.

### **Quick thought exercise:**

* If you have a column of numbers [10, 11, 12, 10] and you map your dictionary, what will happen for each value?
* What if the column contains a month number not in your dictionary? (Like 9 or 1)

Thinking about these will help you handle edge cases smoothly.

Once you’re comfortable with that, I can guide you on which pandas method fits best here and why!

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

If I map the dictionary to the column [10, 11, 12, 10], each number will be replaced by its corresponding month name so 10 becomes 'Oct', 11 becomes 'Nov', and 12 becomes 'Dec'. The 10 at the end will also become 'Oct'. I don't know: What if the column contains a month number not in your dictionary? (Like 9 or 1)

2/2

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

Exactly — your understanding of mapping is spot on! Every key in your dictionary replaces its matching value in the column.

### **Now, about the “unknown keys” problem:**

If your column has a month number **not in your dictionary** — like 9 or 1 — what happens depends on the method you use:

* **Mapping with .map()**:  
   For any number **not found in the dictionary**, pandas will insert a **NaN** in the result. This means that if your dictionary only has 10, 11, 12 but the column has 1, that 1 will turn into NaN after mapping.
* **Replacing with .replace()**:  
   It will replace only the keys it finds and leave the others **unchanged** (still as numbers).

### **So:**

* If you want to convert **all 12 months**, your dictionary should include all months from 1 to 12.
* If you only want to convert a subset (like just 10, 11, 12), then with .map(), all other values will become NaN, which you might need to handle later.
* If you want to keep other months as numbers (or original values), .replace() might be better.

### **Tutor prompt:**

* Which behavior fits your data needs better — do you want unmapped months to disappear (NaN) or stay as they are?
* How would you build a dictionary for all 12 months if you want full coverage?
* How would you handle the NaNs if they appear after mapping?

Answering these will help you decide the best way forward!

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

For now I just want to replace the numbers here in dict Here is how we will do: months = {10: 'Oct', 11: 'Nov', 12: 'Dec'}

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

Perfect! Since you want to replace only those specific months and leave the rest as they are, think about this:

* Using **.replace()** with your dictionary will swap only the keys you specify (10, 11, 12) to their names.
* Any other month numbers (like 1, 2, 9) will remain untouched as numbers.

### **So, in simple terms:**

1. You have your dictionary for replacements.
2. You tell pandas: “Replace these exact numbers with these strings.”
3. The rest stay the same, which keeps your data intact without creating NaNs.

### **Quick mental check:**

* If you had used .map() instead, what would happen to months not in the dictionary?
* Why might .replace() be safer here for partial replacements?

You’re doing great — want to try applying this logic to your DataFrame column now? If you run into any issues, I’m here to help troubleshoot!

Will this work? topic\_month\_counts['Month'] = topic\_month\_counts['month'].map

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

Almost! But as it stands, that line is incomplete.

.map() is a **function** that needs an argument — specifically, a mapping object like your month\_map dictionary.

Without providing the dictionary inside the parentheses, Python won’t know how to perform the mapping.

So, to make it work, you have to call .map() with your dictionary, like this (conceptually):

python

CopyEdit

topic\_month\_counts['month'].map(month\_map)

Without the parentheses and argument, .map just refers to the function itself, but it doesn’t execute anything.

**In short:** You must **call** .map() with the dictionary you want to use for the replacement.

Does that clear it up?

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

Will this work? topic\_month\_counts['Month'] = topic\_month\_counts['month'].map(month\_map) This should work!

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

Yes, exactly! That **will work** perfectly for what you want to do.

* It takes the 'month' column,
* uses .map() with your month\_map dictionary to replace the numbers with month names,
* and stores the result in a new 'Month' column.

Just remember, if there are any month numbers **not in your dictionary**, they’ll become NaN in 'Month'.

Great job putting it all together! Want to move on to how to handle those NaNs, or are you good for now?

Conversation 4:

I’m working with a DataFrame where each row contains four topic columns: topic\_1, topic\_2, topic\_3, and topic\_4. I want to analyze these topics more easily by checking which ones match specific keywords. But, it’s getting difficult to search across multiple columns. I want you to help me to reshape the data so that all the topic values are stacked into one column, while still keeping other columns like month, year, and Count. Can you walk me through how that kind of transformation is usually done in pandas? Do not give me the exact code

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

Absolutely! You're on the right track by thinking of reshaping your data to make analysis more efficient.

In pandas, when you have several columns that represent similar types of data (like your topic\_1, topic\_2, topic\_3, and topic\_4), it's often useful to **"unpivot"** them—this means turning those multiple columns into rows under one column. This process is helpful when you want to **apply the same filter or condition to all the values**, such as checking if a topic matches a keyword.

Pandas has a method that lets you do exactly that. You would basically tell it:

* "These are the columns I want to keep as they are" (like month, year, and Count).
* "These are the columns I want to stack into a single column" (in your case, the topic columns).

The result is a longer DataFrame where each row represents a single topic associated with its original row's metadata (like the article's month, year, and count). This format is **much easier to filter**, count, or group by themes like "People" or "Power".

Let me know if you'd like an example or want to confirm your understanding!

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

Help me understand using an example

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

Sure! Let’s walk through a simple example together to help you understand how this transformation works in pandas.

### **🔹 Imagine you have a small dataset like this:**

| **year** | **month** | **Count** | **topic\_1** | **topic\_2** | **topic\_3** | **topic\_4** |
| --- | --- | --- | --- | --- | --- | --- |
| 2023 | 10 | 3 | israel | children | military | rights |
| 2023 | 11 | 2 | civilians | army | refugees | weapons |

This is a **wide format**, where each row has **four different topic columns**.

### **🔹 What’s the problem?**

If you want to check if any topic matches a keyword like **"children"** or **"army"**, you’d have to write conditions for topic\_1, topic\_2, topic\_3, and topic\_4 separately. That’s repetitive and hard to manage.

### **🔹 What’s the goal?**

You want a **long format**, like this:

| **year** | **month** | **Count** | **topic\_position** | **topic\_word** |
| --- | --- | --- | --- | --- |
| 2023 | 10 | 3 | topic\_1 | israel |
| 2023 | 10 | 3 | topic\_2 | children |
| 2023 | 10 | 3 | topic\_3 | military |
| 2023 | 10 | 3 | topic\_4 | rights |
| 2023 | 11 | 2 | topic\_1 | civilians |
| 2023 | 11 | 2 | topic\_2 | army |
| 2023 | 11 | 2 | topic\_3 | refugees |
| 2023 | 11 | 2 | topic\_4 | weapons |

Now, all the topics are stacked in one column (topic\_word), and you can easily:

* **Filter**: e.g., where topic\_word in my\_keywords
* **Group**: count how many times each keyword appears
* **Visualize**: how often certain themes (e.g. People vs. Power) appear over time

### **🔹 So, how is this done?**

This transformation is usually done using a **reshaping method in pandas** that helps you “melt” or “unpivot” the columns. You’d specify:

* Which columns to keep unchanged (like year, month, and Count)
* Which topic columns you want to turn into one

The result is a **tidy format** where analysis becomes much simpler.

Would you like to try reshaping a toy example yourself? I can guide you step-by-step!

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

What is melt function and how to use that

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

Great question!

### **🔹 What is melt() in pandas?**

melt() is a **reshaping function** in pandas that helps you **transform a wide DataFrame into a long (tidy) format**. It's especially useful when you have multiple columns that contain similar types of data (like topic\_1, topic\_2, etc.) and you want to stack them into a single column for easier analysis.

### **🔹 Think of it like this:**

Imagine your DataFrame looks like this:

| **id** | **topic\_1** | **topic\_2** | **topic\_3** |
| --- | --- | --- | --- |
| 1 | israel | civilians | army |
| 2 | refugees | police | children |

You want a format like this:

| **id** | **topic\_position** | **topic\_word** |
| --- | --- | --- |
| 1 | topic\_1 | israel |
| 1 | topic\_2 | civilians |
| 1 | topic\_3 | army |
| 2 | topic\_1 | refugees |
| 2 | topic\_2 | police |
| 2 | topic\_3 | children |

That’s exactly what melt() does!

### **🔹 How to use melt():**

Here’s what the key arguments mean:

* id\_vars: Columns you **want to keep** as they are (they stay "fixed").
* value\_vars: Columns you want to **unpivot** or stack into a single column.
* var\_name: Name for the new column that will contain the original column names (topic\_1, topic\_2, etc.).
* value\_name: Name for the new column that will contain the actual data (the topic words like israel, army, etc.).

### **🔹 Why use it?**

Once your data is melted:

* You can **filter** easily: e.g., topic\_word in ['children', 'army']
* You can **group and count** how often topics appear
* You can **classify** topics as “People” or “Power” based on their keyword category

Would you like to try it on a small dataset with my guidance?

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

Give an example using the melt function and I will customize it to my dataset

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

Absolutely! Here's a simple example using melt() on a small DataFrame to show you how it works. You can then adapt it to your own data:

python

CopyEdit

import pandas as pd

# Example data: each row has an ID and three topic columns

data = {

'id': [1, 2],

'topic\_1': ['israel', 'refugees'],

'topic\_2': ['civilians', 'police'],

'topic\_3': ['army', 'children'],

'month': [10, 11],

'year': [2023, 2023],

'Count': [5, 8]

}

df = pd.DataFrame(data)

# Use melt to reshape topics into a single column

df\_melted = df.melt(

id\_vars=['id', 'month', 'year', 'Count'], # columns to keep

value\_vars=['topic\_1', 'topic\_2', 'topic\_3'], # columns to unpivot

var\_name='topic\_position', # name of the new column for original column names

value\_name='topic\_word' # name of the new column for the values

)

print(df\_melted)

**Output:**

| **id** | **month** | **year** | **Count** | **topic\_position** | **topic\_word** |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 2023 | 5 | topic\_1 | israel |
| 2 | 11 | 2023 | 8 | topic\_1 | refugees |
| 1 | 10 | 2023 | 5 | topic\_2 | civilians |
| 2 | 11 | 2023 | 8 | topic\_2 | police |
| 1 | 10 | 2023 | 5 | topic\_3 | army |
| 2 | 11 | 2023 | 8 | topic\_3 | children |

Now you can customize:

* Replace 'id', 'month', 'year', 'Count' in id\_vars with the columns you want to keep fixed.
* Replace the list in value\_vars with the topic columns in your dataset (like 'topic\_1', 'topic\_2', 'topic\_3', 'topic\_4').

Would you like me to help you customize this for your dataset?