

Unsupervised ML Application with an LLM

Dalya Baron; International summer school on the ISM of galaxies (GISM3)

In this hands-on session, you will build your first unsupervised machine learning application. Your primary collaborator for this task will be a Large Language Model (LLM). To succeed, it's vital to have the right mental model for what this tool is.

Instead of thinking of an LLM as a person or a colleague, think of it as a tool that provides interactive access to a compressed representation of human knowledge. Imagine a system that has been trained on a vast portion of everything humanity has written and published on the public internet—textbooks, code from repositories like GitHub, scientific papers, millions of tutorials, and forum discussions.

This LLM has not "understood" this information in a human sense. Instead, it has encoded it into an incredibly complex and efficient map of interconnected patterns. It knows that in the context of Python data science, the characters `import pandas as pd` are extremely likely to appear. It knows that the concept of "DBSCAN" is statistically close to "density" and "clusters" but further from "supervised regression."

This makes the LLM an unprecedented tool for synthesis and generation. It can instantly cross-reference concepts, translate a theoretical request into actual code, and summarize complex topics.

However, this tool has no goals, no context about your specific problem, and no ability to critically evaluate the truth or applicability of its output.

You are the scientist who wields the tool. You provide the goal (e.g., "find clusters in my PHANGS data"), the specific context, and the critical judgment to separate plausible-sounding nonsense from genuine insight. Your job is to guide its powerful pattern-matching abilities toward a correct and meaningful solution by asking the right questions and, most importantly, **verifying its work**.

Part 1: The golden rules of interacting with LLMs

To get the most out of working with LLMs, follow these rules:

- 1. Be Specific and Provide Context:** The quality of the output depends directly on the quality of your input (your "prompt").
 - **Bad:** "Tell me about clustering."
 - **Good:** "I'm a masters/PhD student working on astrophysics and the interstellar medium, and my goal is to learn about unsupervised ML and effectively apply it to my data. Explain the core idea behind density-based clustering, like DBSCAN. How does it differ from centroid-based clustering, like K-Means?"
- 2. Set the Stage (Priming):** Start your conversation by giving the LLM a role and the overall context of your project. This helps frame all subsequent answers.

- **Example:** "You are a helpful assistant with expertise in data science. I am a student working on an unsupervised machine learning project in Python using [scikit-learn](#), [pandas](#), and [matplotlib](#). My goal is to explore a dataset to find natural groupings of data points. I will be asking you for explanations, code snippets, and advice."
3. **Iterate and Refine:** Don't expect the perfect answer on the first try. Use the LLM's first response as a starting point. Ask follow-up questions to clarify, simplify, or correct the output.
 - **Example Follow-up:** "That's a helpful explanation of K-Means. Could you explain what the 'inertia' metric represents in the scikit-learn output? Explain it with a simple analogy."
 4. **Ask for Alternatives:** LLMs can get "stuck" on one way of doing things. Prompting for different approaches can reveal better solutions.
 - **Example:** "You suggested using PCA to visualize the clusters. What are the pros and cons of using t-SNE instead for this purpose?"
 5. **You Are the Expert on Your Data: Never paste your raw data into a public LLM.** Instead, describe its structure. You know its meaning, its potential issues, and your ultimate goal. The LLM knows none of this.
 - **Example:** "My data is in a pandas DataFrame. It has 2,000 rows and 25 columns. All columns are numerical and represent physical features measured in the interstellar medium of local galaxies. The features are based on multi-wavelength observations from surveys like the PHANGS survey. They trace the properties of the stellar populations (stellar population age, surface mass density, stellar velocity offset and kinematics), ionized gas (ionization, density, temperature), molecular gas (CO observations), and dust (PAH and mid-infrared continuum emission)."
 6. **Verify EVERYTHING:** This is the most important rule. **Always assume the LLM's output could be subtly wrong.**

LLM "Memory" and Long Conversations

This is one of the most important technical limitations to understand. LLMs don't have a true memory like humans. They operate on a "**context window**"—a fixed-size buffer of the most recent text in your conversation. In a very long chat, the earliest messages get "pushed out" of this window, and they forget what you talked about at the beginning.

Symptoms that the LLM is "Forgetting":

1. It starts using generic variable names (e.g., [df](#) or [data](#)) instead of the specific ones you defined earlier (e.g., [phangs_features](#)).
2. It suggests a technique you had previously ruled out (e.g., recommending K-Means after you both agreed DBSCAN was a better fit).
3. Its answers become generic and lose the specific context of your project.

How to Overcome This:

1. **Start Fresh for New, Distinct Tasks:** If you've finished exploring algorithm families and have chosen one, it's often best to start a new chat for the implementation phase. The first prompt of this new chat can be a summary of your decisions so far. If you choose to follow this suggestion, you can actually ask the LLM to build you a prompt for the new chat, summarizing everything that has happened and building the most effective prompt for the next task!

> “Great, thank you! I’ve selected DBSCAN as the clustering algorithm I will work with. I now want to start a new chat for the next task of exploring the hyper-parameters and minimization objective of DBSCAN. Please provide me with the most efficient prompt for the new chat, summarizing the conclusions we’ve reached and clearly explaining what my next goal is”

2. **Periodically Summarize and Re-prime:** If you prefer to stay in a single chat, you must periodically remind the LLM of the context. This is the most effective technique for long, focused tasks.

- **Use a "Context Refresh" Prompt:** Before asking a new, complex question, feed the LLM a summary of the current state.
- Example Refresh Prompt:

> "Let's do a quick recap to make sure you're up to speed. We are working on a clustering project in Python. My data is in a pandas DataFrame called `phangs_features`, which has already been scaled using `StandardScaler`. We have selected the DBSCAN algorithm because we suspect our clusters are non-spherical. Our current goal is to find the right hyper-parameters (`eps` and `min_samples`) by visualizing the results using PCA.

> Now, with that context in mind, please help me with the next step..."

3. When asking for a code modification, don't just describe the change. Copy and paste the relevant code block from the LLM's previous response into your prompt and then ask for the modification. This ensures the LLM has the exact code it needs to work with, right at the top of its context window.

Part 2: Where should we be particularly careful?

To predict when an LLM will be reliable versus when it will fail, it's crucial to understand how it works. The LLM has encoded its vast training data (text and code) into a high-dimensional mathematical space called an **embedding space**. In this space, related concepts and code snippets are located near each other. The model's core function is to generate a response that is a plausible point within this space, based on your prompt.

The reliability of its output depends on the density of training data around your prompts' location in this space. Let's consider the two extremes:

1. **Interpolation (High-Confidence Zone):** When you ask about a common topic, you are targeting a high-density region of the embedding space. Think of standard Python, the scikit-learn API, or the principles of K-Means clustering. The model has seen millions of valid examples. In this mode, the LLM's task is to interpolate—to find a new point (your answer) that lies comfortably between countless known, correct examples.
2. **Extrapolation (The "Hallucination" Zone):** When you ask about a niche, novel, or highly specific topic, you are targeting a low-density region or "void" in the embedding space. Imagine asking it to write code using the non-standard software for a specific astronomical instrument, or to reason about the unique, unpublished quirks of your dataset. The LLM has very few, if any, data points in this region. It is forced to extrapolate—to generate a response far from its cloud of trusted training examples. It will still produce text that is structurally sound and confident, but it's "making it up" by applying patterns learned from other, potentially unrelated, parts of the space. This is the primary cause of "hallucinations."

Your primary skill in using this tool is to formulate prompts that keep the LLM in an **interpolative mode**.

- **Trust it for:** Generating standard scikit-learn code, explaining the textbook definition of PCA, summarizing the pros and cons of well-known algorithms.
- **Be deeply skeptical when:** You ask it to interpret the meaning of your specific data, use a niche library, or combine concepts in a way that is not common practice. Always assume it knows nothing about your specific experimental setup.

With that framework in mind, here are the specific pitfalls to watch for:

1. **"Hallucinations":** This is the direct result of the LLM being forced to extrapolate. It might invent function names, parameters, or even entire libraries because they are syntactically plausible, even if factually nonexistent. If your code throws an [AttributeError](#) or [ImportError](#), it's likely a hallucination.
 - **Your Action:** Always check the official documentation (e.g., scikit-learn's website) for the functions and parameters the LLM suggests.
2. **Outdated Information:** The LLM's knowledge is a static snapshot of its training data. The dense region it's interpolating from might be based on older versions of libraries.
 - **Your Action:** A quick look at the official docs will confirm if a newer, better method exists.
3. **Subtle Code Bugs:** This is the most dangerous pitfall. The code might run without errors but produce the wrong result because the LLM has interpolated a common but incorrect pattern.
 - **Common Examples:** Forgetting to scale data before distance-based algorithms (like PCA, K-Means, DBSCAN), mixing up `axis=0` and `axis=1` in pandas/numpy, or misinterpreting the results of a function.
 - **Your Action:** Ask the LLM to **explain the code line by line**. Does the explanation match your understanding of what *should* be happening? For example, ask, "*Why is scaling the data with [StandardScaler](#) essential before running this algorithm?*"

4. **Overconfidence:** LLMs present all information—both high-confidence interpolations and wild extrapolations—with the same authoritative tone. Be mindful of that.

Part 3: A Practical Workflow for Your Project

Follow these steps to explore, select, and implement an unsupervised learning algorithm.

Step 1: Priming the LLM and Exploring an Algorithm Family

Your goal is to get a high-level overview of the techniques available for your chosen task (Dimensionality Reduction, Clustering, or Outlier Detection).

- **Bad Prompt:** “> Explain clustering.”

- Why it's bad? Too vague. You'll get a generic, textbook-like answer.

- **Good Prompt:**

> "You are an expert data scientist and a helpful tutor. My goal is to analyze a dataset to find natural groups of data points, a task known as clustering.

>

> First, please provide a high-level overview of the main families of clustering algorithms. Please create a markdown table that lists at least three families (e.g., Centroid-based, Density-based, Hierarchical), their fundamental goal, and one popular example algorithm for each."

Step 2: Comparing Algorithms

Now, use the LLM to help you choose an algorithm based on the characteristics of your data and your goals.

- **Bad Prompt:** “> What's the best clustering algorithm?”

- Why it's bad: "Best" is meaningless without context.

- **Good Prompt:**

> "Let's select an algorithm for my specific project. My dataset has ~5,000 data points and 30 numerical features. I don't know the number of clusters in advance, and I suspect they might be of irregular shapes and varying densities.

>

> Compare K-Means, DBSCAN, and Agglomerative Hierarchical Clustering for this scenario. Focus on their strengths and weaknesses regarding:

- > 1. The need to pre-specify the number of clusters.
- > 2. Ability to handle non-spherical clusters.
- > 3. Performance with outliers, noise, and upper limits.
- > 4. Computational scalability with my dataset size.

>

> *Based on this comparison, recommend one algorithm to start with and briefly justify your choice."*

Step 3: Deep Dive into Your Chosen Algorithm

You've selected an algorithm (e.g., DBSCAN). Now, you need to understand how it works internally and what its key hyper-parameters are.

- **Bad Prompt:** "> *How does DBSCAN work?*"

- Why it's bad: it will provide a generic answer according to DBSCAN's most common use in the data science literature. Being specific will help the model give you a more specified answer.

- **Good Prompt:**

> *"I've decided to start with DBSCAN. Let's do a deep dive.*

>

> *1. Explain the intuition behind DBSCAN's core hyper-parameters: `eps` and `min_samples`. I am a masters/PhD student in astrophysics, with knowledge of physics and math - adjust the answer to my knowledge level.*

> *2. What is the algorithm's objective function? What does it minimize?*

> *3. Define what DBSCAN considers a 'core point', a 'border point', and a 'noise point'."*

Step 4: Generating the Code

This is where you translate theory into practice. Be very specific about your setup to get usable code.

- **Bad Prompt:** "> *Write Python code for DBSCAN.*"

- Why it's bad: You'll get generic code with a fake dataset. It won't be integrated with your project.

- **Good Prompt:**

> *"Let's write the Python code to run DBSCAN on my data. I have my data loaded into a pandas DataFrame called `PHANGS_features`.*

>

> *Please provide a Python code snippet using `scikit-learn` that accomplishes the following steps:*

> *1. Create a `StandardScaler` object and use it to scale `PHANGS_features`. Crucially, add a comment explaining why scaling is essential for DBSCAN.*

> *2. Initialize the `DBSCAN` model. For now, use default values for the hyper-parameters like `eps=0.5` and `min_samples=5`.*

> *3. Fit the model to the scaled data.*

- > 4. *Extract the cluster labels from the fitted model.*
- > 5. *Create a new column in a separate DataFrame called `df_results` named 'cluster_label' containing these labels.*
- > 6. *Finally, print a summary: the total number of unique clusters found (excluding noise) and the total count of noise points (labeled as -1)."*

Step 5: Analysis, Visualization, and Iteration

Your first run is complete. Now you need to interpret the results and decide on the next steps. This is where your an iterative work with the LLM is important (while also remembering that it will forget what you discussed in a very long conversation).

Good Prompt Example:

- > *"The code ran successfully. It identified 2 clusters and 800 noise points, which seems like a lot of noise. This suggests my `eps` or `min_samples` values are not well-tuned.*
- >
- > 1. *How would increasing or decreasing `eps` likely affect the number of clusters and noise points? What about `min_samples`?*
- > 2. *Since my data has 30 dimensions, I can't plot it directly. Suggest a method to visualize the clustering results. Please provide the code to first use PCA to reduce the data to 2 dimensions and then create a scatter plot using `matplotlib`, coloring the points by their 'cluster_label' from the previous step. Make sure noise points are colored differently (e.g., in black)."*

From here, you will continue the loop: analyze the visualization, form a hypothesis about your hyper-parameters, ask the LLM to help you adjust the code, and run it again.