Spider Web Rubrics

Cheat sheet + Tasking guidelines

Last updated: Oct 29, 2025

🖐🏻 Welcome to the Spider Web Rubrics Project

Your goal in this project is to create data-analysis tasks that challenge the AI model to reason, calculate, and visualize information correctly.

1. **Upload datasets**: Provide at least three publicly available datasets (CSV, TSV, XML, JSON, etc.) that are open-source and suitable for analysis.
2. **Design prompts**:
    - Write a data analysis prompt that can be either:
        i. **Close-ended prompts:** have one definitive, correct answer.
        ii. **Open-ended prompts:** allow multiple valid analytical insights.
    - **Include a visualization request**: Prompts should include at least one request to generate a plot or chart
3. **Write rubrics**: For every prompt, write clear, self-contained rubric criteria and reviewer notes that define what makes a response correct. Your rubric must cover both the verbal explanation and the visual accuracy of the graph/plot.

Table of Contents:

⚙️Project Workflow - Overview

The project follows a **two-stage process** for creating high-quality data analysis tasks.

- Stage 1: Attempt Level: Prompt Creation & Quality Control Notes

- Stage 2: & Rubric Creation

🔹 Stage 1: Attempt Level: Prompt Creation & Quality Control Notes

Goal: Create complex, data-driven prompts that challenge the model's analytical and reasoning skills.

The main focus is Data Analysis.

Your responsibilities at this level:

1. **Design a challenging prompt** that uses all required datasets and meets project complexity standards.
2. **Run the prompt in ChatGPT-5** to generate a model response and identify meaningful failures. Explain where the model failed.
3. **Document Quality Control Notes** that explain the prompt's goal, step-by-step solution (the correct workflow), and document corrected/expected results in the designated.
4. **Document Required Output** that provides the actual/correct outputs of the submitted prompt.
5. **Submit your work for review.** The Quality Managers (QMs) and Project Team will verify that your prompt meets all quality standards.

***!!*If approved:** Your task will advance to **Stage 2**, where you'll create the full rubric items for evaluation.

🔹 Stage 2: Rubric Creation

Goal: Define how model responses will be evaluated using clear, detailed rubrics.

Your responsibilities at this level:

1. **Write precise, self-contained rubrics** for every part of the prompt, covering both positive and negative criteria.
2. **Ensure alignment** with rubric best practices: criteria must be atomic, factual, and formatted correctly (e.g., *States that…*, *Includes a…*).
3. **Assign appropriate weights and categories** to each rubric item you created.

!!Note: If you receive a feedback on Quality Control Notes, please fix these issues with complete explanations, formulas, and references to support reviewer evaluation before moving into Rubric Creation - Failure to fix them may result in lower submission feedback scores.

⭐What Makes a Good Task?

A high-quality task should always include the following elements:

- ✅ **At least 3 datasets (CSV, TSV, XML, JSON, etc)** → The datasets must have the proper size, must be open source, and publicly accessible.
- ✅ **A Strong Complex Prompt** → Well-written, clear, and aligned with specifications.
- ✅ **Model failure**→ The prompt must be complex enough to make the model fail naturally.
- ✅ **Open and closed-ended Prompts** → You'll follow the type of prompt given at the beginning of the task.
- ✅ **Prompts must ask for a plot/graph** → Prompts must ask.
- ✅ **Prompts must use all datasets uploaded at the beginning of the task**→ Every dataset uploaded MUST be used.
- ✅ **Perfect Rubrics** → Rubrics must follow the rubric's best practices.
- ✅ **Criteria is written using the correct formatting**→ Criteria must always start with the **simple present** tense (*e.g., States that…, includes a…, provides a…, mentions that…, etc.*). DO NOT start your criteria with "The response must/should".
- ✅ **Rubric contains 1 negative criterion per model response error** → For every failure that you identify in the model response, you must create a criterion addressing it.
- ✅ **[NEW] Rubric contains at least 10 negative criteria** → Once you have written your positive weight criteria, you'll write negative weight criteria for model mistakes.

👀Where to look for answers? - Quick Clicks

If you are an attempter in Stage-1 (prompt creation) the following sections will be beneficial for you!

- Dataset Requirements here
- Prompt Creation here
- What is a model response? Here
- What is SOTA failure? here
- Step-by-step guidance on how to create a task for this stage.here
- Task Examples here

If you are an attempter in Stage-2, the following sections will be beneficial for you!

- Rubric Quality Rules here
- Weight Assignment System - How to assign weights.here
- Step-by-step guidance on how to create a task for this stage. here
- Task Examples here

And finally, don’t forget to use the checklist before your task submission! This will help you go over the requirements and help you succeed! here

---

🍀Project Resources

- Community Channel 💬
- War room available every day from 9:00 - 18:00 PST
    - Link to the War Room
    - ***Feel free to join us in the War Room and bring your questions with you! We are here to help!*** 🆘

Change Log

Date	Updates
Oct 28, 2025	Added clarifications on Negative Rubric Writing
Oct 21, 2025	[NEW] Negative criteria step added to the task taxonomy + New task example.
Oct 9, 2025	[CLARIFICATION]: Added a prompt validity diagram in the How to write a great prompt section.
Oct 7, 2025	[CLARIFICATION]: Follow-up prompts are not allowed in this project. Do not prompt the model with follow-up questions or clarifications. [CLARIFICATION]: Model crashing is not considered a valid model failure.
Oct 2, 2025	- [NEW] Final output section
Oct 1, 2025	-[NEW] Clarification on what we consider “sufficiently large” datasets.
Sep 30, 2025	-[NEW] New Workflow
Sep 29, 2025	-[NEW] Rubrics for open-ended prompts

Sep 26, 2025	- Reasoning failures are no longer valid model failures - Updated common errors (Avoid Process criteria **ALWAYS**)
Sep 25, 2025	- [NEW] Prompt Complexity Guidelines - [NEW] Colab Links

Task Workflow

Stage 1: Prompt Creation

When you first join the project, you'll be granted Attempt permissions.
At this level, your main focus will be on writing complex prompts that focus on Data Analysis. In this section, you'll learn how to create complex prompts:

Stage 1 - Attempt Level- Step 1: Pre-requisites and Task instructions

1

⚠️ IMPORTANT: Read Before You Start Tasking! 🚨

Every task comes with specific requirements **you must follow carefully:**

- **Complexity** – Indicates how challenging the prompt should be.

You are encouraged to follow this task requirements:

- **Suggested Topic** – This is the suggested subject for your task. It's highly encouraged to follow it, but not strictly mandatory.

Make sure you review these requirements before you begin—this ensures your work meets project standards every time. ✅

2

Dataset Attachments

For this project, you'll need to use datasets to ask questions to the Model. Accepted file formats are CSV, JSON, XLSX, JSONL, etc. You only need to attach the files in the first prompt, and a minimum of three attachments are required.

You are allowed to find datasets online, and here are some resources that can help you.

- https://datasetsearch.research.google.com/
- https://www.kaggle.com/datasets
- https://www.tableau.com/learn/articles/free-public-data-sets

Stage 1 - Attempt Level- Step 1: Pre-requisites and Task instructions

- https://catalog.data.gov/organization/
- https://datahub.io/collections
- https://www.earthdata.nasa.gov/

You'll need to download the dataset and also save the links from the website you used.

3

📂 Dataset Requirements

When submitting your task, make sure to **paste the links to all datasets you used**.

✅ Minimum Requirements

- You must use **at least 3 datasets per task**.
- Datasets must be:
  - **Publicly accessible** (no paywalls).
  - **Open source** (free for use, no private license).
  - **In English**.
  - **Clean and structured** (headers on the first row, no corruption).
  - **Sufficiently large** → Try not to use very small datasets. For guidance as to what we consider sufficiently large, aim for at least **5 columns** and **100 rows**, but this is not a hard requirement.
  - **Less than 1 GB total** (counting the 3 Datasets together)

⚠️ **Reminder:** Double-check each dataset before submitting to ensure it opens correctly and meets all requirements.

4

You'll **attach the 3 datasets in order to submit your prompt**. These datasets should match the ones you'll use to write your prompt.

5

🚨 IMPORTANT INSTRUCTIONS 🚨

For this project, you are required to **use ChatGPT-5** (https://chatgpt.com/) to generate all model responses. This means you must:

- Upload the necessary files directly into **ChatGPT-5**.
- Complete the entire workflow inside ChatGPT-5 (not in other models).

🔑 Access to ChatGPT-5

Stage 1 - Attempt Level- Step 1: Pre-requisites and Task instructions

- You'll need to upgrade to the **Plus Plan** to unlock ChatGPT-5: https://chatgpt.com/?model=auto#pricing
- The cost of the Plus Plan will be **reimbursed**, but only if you follow all the requirements below.

✅ **Reimbursement Conditions**

- You must submit the **correct ChatGPT-5 conversation link** at the end of your task.
- If you use any other model (e.g., 4o, 3.5, etc.), you **will not be reimbursed**. Double-check that you've selected the correct model before starting.
- This reimbursement will only happen once

See this material in case you have any questions:

- How to upgrade your account to ChatGPT5
- Task walkthrough

Back to the top ⬆️

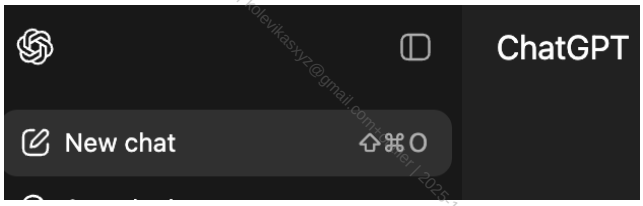
Stage 1 - Step 2: Create a Prompt

1 You'll write your prompt in the ChatGPT5 platform. The prompt must follow the assigned complexity and the Prompt Guidelines that are explained in the following Step.

Remember these IMPORTANT considerations:

- Create a **New Chat** for every new prompt you create

Stage 1 - Step 2: Create a Prompt



- Upload the **3 datasets** and ask a prompt on the topic of Data Analysis or Data Science

2

Please use the following guidelines when writing your prompt:

- **Dataset Dependency** – Every prompt must rely on the attached dataset. The questions should not be answerable without using the dataset.
- **Prompts must stump the model**
- **Prompts must use all datasets uploaded!**
- **Specific and Well-Structured** – Prompts should be clear, detailed, and organized. They may include multiple related questions that require the model to read, manipulate, and reason through the data, as well as run commands to generate results.
- **Realistic Scenarios** – Write prompts that feel natural for a data analyst. Some questions can be slightly contrived, but avoid anything completely unrealistic or irrelevant to real-world analysis.
- **Prompts must be COMPLEX**→ Hard Complexity. [See complexity guidelines here]
- **Plot-Based/Chart/Graphic Questions** – Include at least one question that requires the model to generate a plot.
- **Timelessness** - The prompts you write will be added to a task bank for training at a later date, so please ensure the ideal response does not change with time
- **Create Open-Ended and Close-Ended Prompts based on task assignment** –
  - **Closed-ended** (all experts would arrive at the same answer).
  - **Open-ended** (experts could provide different but still factual and valid answers).

Only ONE prompt needs to be submitted to GPT-5.
Follow up prompts are **not** allowed in this project.

Examples here

3

Useful Resources 🔗

- Complexity guidelines
- Topics and Complexity → Check out this list of topics and their level of complexity

Stage 1 - Step 2: Create a Prompt

- Prompt Ideas and Open Source Datasets you can use → Check out this list of Prompt Ideas and Datasets that are pre-approved

Back to the top 🔝

Stage 1 - Step 3: Model Response

1

Model Failures (stump the model 😵‍💫):

- To proceed with the task, **the response must contain a significant failure**—such as an **incorrect answer or use of the wrong files**.
  - **Presentation issues or model crashing do not qualify as valid failures.**
- If the model doesn't fail, you'll need to go back and create another prompt that's more complex than the previous one you created.
- If you feel the model is giving a completely nonsensical answer, check that your prompt is clear and not ambiguous, fix it, and re-prompt the model on a new conversation. **Do not write follow-up prompts to guide the model**.

Remember to also share the GPT-5 Plus conversation link in the appropriate task field. Make sure the link is shareable and can be open from any browser.

Edge case 1: If for some reason, the model's choice was to provide code to answer the prompt, and the code works and answers it correctly, this **does not count as a valid model failure.**

2

Pro-Tips 💡:

- Check for factual inaccuracies
- Check for hallucinations in the data
- Check for missing steps or misleading steps to resolve the prompt's question
- Check if the model response completely fulfilled your prompt
- Check for correct file usage

3

Justify the errors

Stage 1 - Step 3: Model Response

- You'll be asked to write down all the model failures you found with a brief explanation.

Stage 1 - Step 4: Quality Control Notes

1

You'll need to provide a **comprehensive explanation** of how the ideal workflow should look in order to obtain the prompt's answer. This explanation is essential for reviewers to validate correctness.

Your explanation must include:

- **Prompt Goal** → Clearly state what the prompt is asking for (no need to copy/paste the full prompt, just summarize the objective).
- **Step-by-Step Solution** → A detailed breakdown of how to solve the prompt, covering *every question and requirement*.
    - Reference specific **formulas, column names, and sub-steps**.
    - Include **code blocks** (for plots, calculations, or other operations).

Code must be formatted like this:

```
[insert your code here]
```

- **Code Used to Obtain Answers** → Add all the code you used to get the final results, including data manipulations and plot generation.

Example here

2

✅ Quality Control Notes Template

Prompt Goal:
[Summarize the main objective of the prompt in your own words.]

Stage 1 - Step 4: Quality Control Notes

Step-by-Step Solution:
[List out all steps required to solve the prompt, including formulas, operations, reasoning, and code blocks to obtain the plots/graphs. Ensure every requirement from the prompt is covered.]

Code Used to Obtain Answers:
[Paste all relevant code here, formatted in triple backticks. This should include code for analysis.]

3

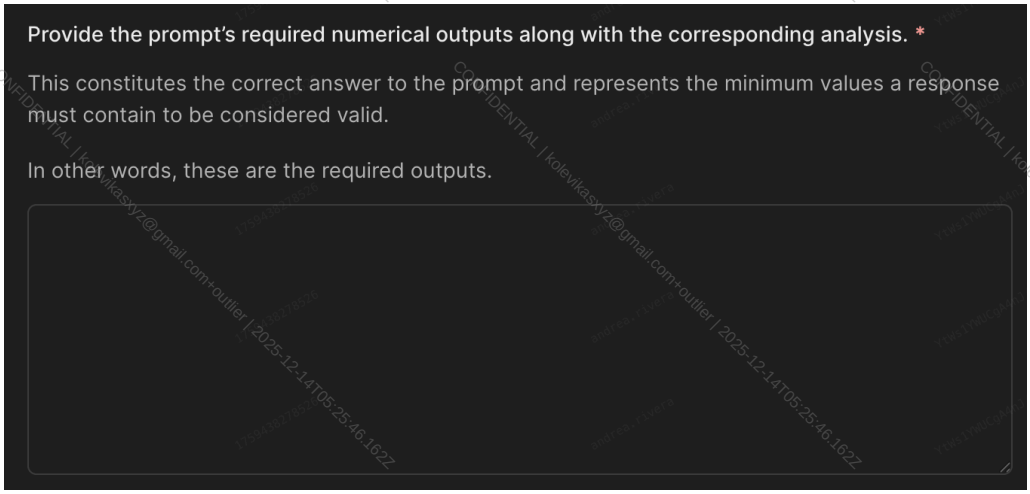
Include the Colab link for the code you created to obtain the answer

Moving forward, we will be asking you to write the script in Colab and share the link to the script you wrote to answer the prompt.

Stage 1 - Step 5: Expected outputs (final answer)

1

You'll be asked to provide the final outputs from your prompt. This is the field where you are going to write them:



2

Please only include the final outputs of the questions in your prompt—that is, the ground-truth final answers.

Stage 1 - Step 5: Expected outputs (final answer)

Example
If the prompt asks for the Pearson correlation, the average population, and the GDP year-over-year, what you should enter in this field is:

- Pearson correlation = 0.04
- Average population = 2,455,000
- GDP YoY = 0.343

Please avoid adding explanations, code, or any other content beyond the required output. If the prompt does not explicitly ask for it, do not include it.

Back to the top ⬆️

Stage 2: Rubric Creation

Once your prompt is approved, you'll move into the role of a Trusted Attempter. At this stage, the task you created will be returned to you so you can complete the Quality Control Notes and build the Rubric.

Back to the top ⬆️

Stage 2 - Step 1: Creating a POSITIVE Rubric

1 **Definition:** A rubric is a checklist of criteria that defines an ideal response to a user prompt. You can think of a rubric as

- A *complete*, *non-redundant*, and *accurate* set of truth conditions specifying *under which conditions* a response to a given prompt is a good-quality response.

Each criterion is an item on the checklist that defines a single condition for the ideal response. The set of all criteria should consist of individually necessary and jointly sufficient items for the response to be perfect.

The positive rubrics will be the prompt's explicit asks, and most of the time they are numerical values.

Stage 2 - Step 1: Creating a POSITIVE Rubric

Rubric Criteria MUST BE:

✅ Appropriately atomic (related to a single discrete challenge in the prompt)
✅ Specific (containing sufficient detail, including answers/examples that do not leave it completely open-ended)
✅ Accurate (correct by fact check/reason)
✅ Categorized (all items in the criteria should fall under the categories)

A rubric MUST BE:

✅ Comprehensive (no missing steps for an ideal response)
✅ Self-contained (sufficient details/examples are provided such that it is specific and not completely open-ended)
✅ Relevant (no unnecessary or unrelated steps)
✅ Accurate (steps that are objective and correct)
✅ Non-Repeating (no redundant steps - don't penalize the same mistake twice).
✅ Reflect all the explicit asks of the prompt

NOTE: In this project, you will also create what we call NEGATIVE rubrics, which we will be covered in the Step 2 section.

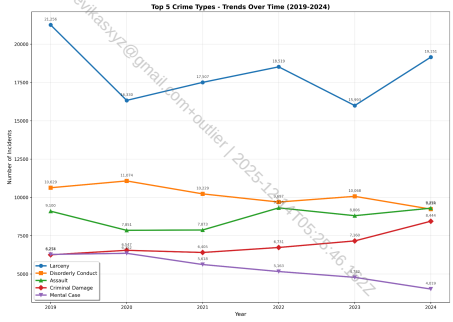
2

When writing rubrics, follow these principles to ensure clarity, precision, and coverage of all key requirements.

1. Plot-Based Criteria

- Every rubric **MUST** have at least 1 criterion that requires a plot
  - For any prompt that requires generating a plot, the rubric **must include the correct reference plot**.
- Include both **verbal criteria** (descriptions of what makes the plot correct) and the **reference plot** to check semantic equivalence.
  - For this, you'll need to attach a reference on how the plot should look like
- Separate criteria to differentiate the plot view from the description of what the plot should contain, and make specific criteria on the content of the plot
  - **Example**:
  - Displays a line chart that is semantically the same as the attached plot:

Stage 2 - Step 1: Creating a POSITIVE Rubric



- Shows a line chart that has all 6 years (2019-2024) on the x-axis.
- Shows the number of incidents on the y-axis in the line chart .
- Displays different colors or shapes for each line associated with a crime type in the line chart.

2. Precision & Specificity

- Always be highly specific.
  - ✅ Example: *"Reports the Number as 2.342"*
  - ❌ Avoid: *"Reports the Number must be 2"*

3. Content Coverage

Rubrics must always mention:

- **The factual answer** to the questions in the prompt.
- **Explicit instructions from the prompt**
- **Implicit instructions** (especially those that are subtle or commonly missed).
- **All criteria related to the model's errors**, ensuring mistakes can be identified and corrected.
  - For every failure that you identify in the model response, you must create a criterion addressing it

4. Style & Wording

- Rubric criteria must always start with the **simple present** tense (*e.g., States that…, includes a…, provides a…, mentions that…, etc.*)
  - Example.
    - ✅ *"Calculates a p-value of p=0.0001 for the correlation."*
    - ❌ *"The response must calculate the p-value of p=0.0001 for the correlation."*

6. Categorization of Criteria

Stage 2 - Step 1: Creating a POSITIVE Rubric

When writing rubric items, we classify them as:

- **Must-Have (Essential):**
  - Direct *Instruction Following* conditions.
  - Explicit asks from the prompt.

- **Nice-to-Have (Not Essential):**
  - Reasoning quality.
  - Factual correctness beyond explicit asks.
  - Implicit asks or inferred instructions.

NOTE: Positive Rubrics will always be **Must-Have** Rubrics

7. Don't include process steps

Example:

Prompt: *Provide the average of the kids attending school on friday*

✅ "*Calculates that the average of kids attending school on friday is 80%*"

❌ "Identifies *attendance_school_day as the variable to use to calculate the average*"

❌ "*Does an average by using the formula Average = Sum of Observations / Number of Observations*"

❌ "*Filters the data attendance_school_day to 'friday'* "

Explanation: The prompt only asks for the average, so there's no need to provide specific criteria on how to obtain the average

8. Rubrics for open-ended prompts

For open-ended/insights prompts, rubrics still need to be well-specified and professional; see below for examples

Examples with positive weights

✅*"States that logistic regression is recommended for this problem due to linearity of data"* <- when prompt asks to compare a few algorithms and make recommendation

Stage 2 - Step 1: Creating a POSITIVE Rubric

✅*"States that no obvious clusters are formed"* <- when prompt asks for a clustering analysis

✅*"Suggests to try perform XXX analysis and rerun clustering"* <- when no cluster formed based on the recommended method

✅*"Suggests that the instructed method may introduce bias ..."* <- when user prompted "could you try generating some insight by doing .... analysis" and the method is questionable

✅*"States that XXX may be the reason why we see YYY"* <- when asked to provide insight after analysis

Examples with negative weights

✅*"Uses two or more paragraphs explaining logistic regression"* <- when asked to compare two methods in the given setting, and found a common mistake is that the model over-explains simple concepts in a generic way

✅*"Uses linear regression for the XXX prediction task"* <- when asked to make a recommendation but makes a common mistake

✅ *"Recommends LLM-based method without checking compute resource"* <- common mistake

9. **Edge case:** When a prompt requests more than 10 stacked items:

- Create spot checks for **20% of items as separate rubrics items**.
- The items for the value selection should be random: Take them from the beginning, from the middle, and from the end of the list.
- Note that if the model fails 1 item in a list of 100 items, it is not a good enough failure.
- ✅ Example:

> *Identifies 22 countries in the choropleth map with a socio-environmental health index.*
>
> *Includes a Socio–Environmental Health Index (PC1) of 0.1134 for the Philippines in the choropleth map.* → **Spot check 1**
>
> *Includes a Socio–Environmental Health Index (PC1) of -0.4835 for Spain in the choropleth map.* → **Spot check 2**

Stage 2 - Step 1: Creating a POSITIVE Rubric

Includes a Socio–Environmental Health Index (PC1) of 10.1968 for China in the choropleth map. ***—> Spot check 3.***

Includes a Socio–Environmental Health Index (PC1) of -0.9508 for Finland in the choropleth map. ***—> Spot check 4.***

Rubric Example here

3 **Check the model response using your rubric -** You'll determine whether the model response follows the criteria or not.

- **True:** Model Response absolutely follow this requirement (The assigned weight will be given)
- **False**: Model Response absolutely failed at passing the criteria; the model response did not complete the outlined task (Does nothing)

4

Understanding Rubric Weights

Each rubric criterion must have a weight between **-40 and 40**, based on its importance to your prompt. Positive rubric criteria must always have positive weights (from 1 to 40).

- **Accuracy-related criteria** (e.g., factual correctness, calculations, results) should receive the **highest weights**, since they are the most critical.
- **Instruction-following criteria** (e.g., formatting, phrasing, or minor task requirements) should be assigned **lower weights** than accuracy but still enough to matter.

Example Prompt:
"Give me the 7 standard colors of the rainbow. What are they?"

Rubric Criteria:

- [+20] States that the first color is red
- [+20] States that the second color is orange
- [+20] States that the third color is yellow

How Weight Scoring Works

Think of rubric scoring like a teacher grading an essay: every time the model includes a correct element, it **earns points**, and every time it makes a known mistake, it **loses points**.

Stage 2 - Step 1: Creating a POSITIVE Rubric

- **Positive weights** are used for correct, required answers (form 1 to 40)

Grading Logic:

- If the criterion is **present in the response** → Apply the assigned points (positive or negative).
- If the criterion is **not present** → Do nothing (0 points).

Example Response:
"The first rainbow color is red, the second rainbow color is orange, and the third rainbow color is pink …"

Scoring Breakdown:

- [Weight 20] → Red is correct (criterion met) = applies the positive points +20 (True)
- [Weight 20] → Orange is correct (criterion met) = applies the positive points +20 (True)
- [Weight -25]→ Pink is wrong (criterion met as a negative) = applies the negative points (True) -25
- [Weight 20] → Yellow is not present (criterion not met) = does nothing

Total Score = 15

✅ **Key takeaway:** Use **positive weights** to reward required information and **negative weights** to penalize common mistakes. This ensures the rubric is fair, balanced, and comprehensive.

Back to the top ⬆️

Stage 2 - Step 2 : Creating a NEGATIVE Rubric

🙋What are Negative Criteria?

Definition: A **negative rubric item** is a criterion that **deducts points** for a *common mistake, omission, or incorrect reasoning*. It identifies what *should not be done* and assigns a **negative weight**.

Stage 2 - Step 2 : Creating a NEGATIVE Rubric

⁉️Why it is important to have negative rubrics?

🆘 They clearly show what mistakes to avoid and ensure that wrong or missing information loses points fairly. They help keep grading balanced, rewarding correct answers and penalizing errors, so results are accurate and consistent. This makes feedback more helpful and transparent for ***the model to learn from its mistakes and not repeat them***.

🧩 Rules for Writing Negative Rubric Items

- **Write them in positive language** (Do not write “Omits”, “Fails”, unless the prompt explicitly mentions to not include something in the answer). The negative part of the negative rubric items are the **weights**.
- **Target common and predictable mistakes.**
  - Negative criteria should describe what **must not** be present in the response or the common misconceptions/errors on how to answer the prompt.
- **Look for “extra” claims that prompt did not ask for**
  - Negated criteria can target incorrect claims the model adds that weren’t explicitly requested, similar to statements related to the required outputs but included as extra content. Essentially, penalize “extra” hallucinations that we would otherwise consider as "nice-to-have" content if they were correct.
    - Adds **extra unrequested outputs that are incorrect** (invented stats, false trends)
    - **Over-interprets results** (causal or policy claims)
    - **Introduces plausible but wrong analysis language** (mixes metrics, misstates plots, etc.)
- **Don’t use polar opposite of a positive rubric item**
  - Example:
    [5] States the best soccer player is Messi
    [-5] States the best soccer player is **not** Messi
- **Mirror major positive criteria where possible.**
  - Try to think what could go wrong and cover that area whenever possible
- **Never double-penalize.**
  - Make sure you are not creating a rubric item that is closely covered under a positive rubric item. If we made the negatives simple opposites, we’d double-penalize the same issue.
- **Match tense and clarity with positive items.**
  - Use present tense and consistent phrasing.
- **Keep penalty weights proportional to error severity.**
  - Larger conceptual or logical errors deserve bigger negative values, while minor lapses deserve smaller ones.
- **Use negative weights (−1 to −40)**
  - Each negative rubric item must include a negative point value showing a deduction for an error or omission.
- **Assign a category.**

Stage 2 - Step 2 : Creating a NEGATIVE Rubric

    - Use appropriate negative weight and also assign a category “must-have” vs “nice-to-have”
- **Ensure absolute weight ≤ positive counterpart.**
    - A negative item must never exceed the absolute value of its related positive criterion. It can only be equal or less.
- **Include both must-have and nice-to-have mistakes as needed.**
    - Negative items can be a mix of both types, and it’s acceptable if no must-have negative items are present. They can be mostly nice to have items.
        - Nice-to-have: Use the system prompt and its response can be used for creating criteria that fall under this category. The model will give you some failure ideas based on the provided prompt and positive rubric you wrote.
        - Must-have: Use the SOTA failures you first encountered earlier in the task to write this.
- **Include at least 10 negative rubric items.**
    - The more the better.
    -

🧩 The general rules for rubric writing still apply for negative rubrics! Don’t forget to:

- **Be specific and atomic.**
    - Each negative item should penalize one clear, observable mistake, not multiple issues at once.
    - Avoid compound items like “fails to define or explain variable.” Split into two if needed.
- **Statements should be self-contained.**
    - A judge (human or LLM) uses only the rubric criterion and the model’s response to decide if it passes, fails, or partially passes. It may not know the topic or see the original prompt. That’s why each rubric criterion must be fully self-contained, giving the judge all information needed to evaluate correctly. Judge should not need any addition
- **Avoid redundant or overlapping rubrics.**
    - If two items test nearly the same aspect, remove one item
- **Keep language objective and observable.**
    - Describe what’s wrong in factual, checkable terms, not vague judgments

We will use a system prompt to generate potential model pitfalls for nice-to-haves.

🏋️Weights - How to assign negative weights?

Negative weights are used for known *common error patterns* that must be penalized if they appear (from -40 to -1)

- **Base weight on importance to the prompt’s goal.**
- **Match positive and negative weights proportionally.**

Stage 2 - Step 2 : Creating a NEGATIVE Rubric

- **Reflect the criterion's logical impact, not its length.**
  - Use smaller weights for penalties that don't invalidate correctness. Minor factual errors, grammar slips, or partial misstatements shouldn't outweigh major reasoning mistakes.
- **Scale the weights consistently across all items.**
  - Example structure:**(Keep in mind these are absolute values and just an example)**
    - Critical conceptual step: 26 to 40
    - Core factual or method step: 17 to 25
    - Clarity or completeness detail: 9 to 16
    - Formatting or style improvement: 1 to 8

👍 Important Rule of thumb:
- A negative rubric item must never have a greater absolute weight than its corresponding positive item; it may be equal to or smaller than the related positive weight.

🎨Categories

😁**Must have -** Penalizes a common or critical error that directly violates the prompt's core requirement.
😀**Nice to have -** Penalizes a minor or less frequent mistake that affects clarity or completeness but not the main response.

🥊How to decide Must-have vs Nice-to-have?

👉 If the mistake makes the answer **wrong or invalid**, it's **must-have**.
👉 If the mistake just makes the answer **weaker or less clear**, it's **nice-to-have**.

!!Important Note: *Negative rubric items can include a mix of must-have and nice-to-have criteria, and it's acceptable if no must-have negatives are present.*

Example Prompt:
"Give me the 7 standard colors of the rainbow. What are they?"

Negative Rubric Criteria:

- [+20] States that the first color is red
- [+20] States that the second color is orange
- [+20] States that the third color is yellow
- [-25] States that pink is a rainbow color

Stage 2 - Step 2 : Creating a NEGATIVE Rubric

👉 In this example, “pink” is not a rainbow color, but it’s a common misconception. By adding a **negative weight**, we penalize the error if it shows up.

How to come up with ideas for a “nice to have” Negative Rubric Criteria

1. Use the system prompt outlined in the Negative Criteria System Prompt section in the Appendix to generate wrong model responses.
    a. The system prompt will generate “Nice-to-Have” negative rubrics. “Nice-to-Have” in this context only means that the rubric is not an explicit prompt request, but “bonus” information the model is providing.
2. The model will give you some failure ideas based on the provided prompt and positive rubric you wrote.
3. Analyse the failures, and create the negative rubrics for them.

For model mistakes related to “Must-Have” Rubrics, you will create these based off of the SOTA failures you first encountered earlier in the task.

❓How to Think About It Overall

Type	Description	Example
Positive rubric	Covers the prompt’s explicit asks, usually numerical values or specific required steps.	[5] Reports the average as 2.
Negative rubric (explicit error)	Penalizes a model that attempts the required step but executes it incorrectly.	[-5] Reports the median [3] as the average.
Negative rubric (extra / hallucinated)	Penalizes incorrect information the model adds that wasn’t requested, even if it sounds reasonable.	[-5] Adds an incorrect statistic (“standard deviation is 1.8”). [-5] States “the data is trending upward.”

Task Specifics

If you are looking for a more in-depth explanation, make sure to read this section.

What's a Model Failure?

When we talk about **SoTA failures**, we're identifying weaknesses where the model struggles and creating training data to address them.

Since Spider Web is focused on Data Science tasks, useful failures are those involving **numerical values or analysis**.

- Failing the model for using the wrong plot color or forgetting to round is not meaningful—it's an instruction-following issue, not a core Data Science failure.
- Failing it for incorrect calculations or flawed analysis, however, creates valuable data to make the model stronger.
- Model crashing is **not** considered a valid model failure.

What Is a Model Response?

When we say **model response**, we don't mean the answer shown in your task. Instead, we mean the response the model will generate to your prompt **during training**. That could happen tomorrow, six weeks from now, or six months later.

The model will generate the response it predicts is correct at that point in time.

Why does this matter?

- **Timelessness.**
  Prompts must be written so their answers don't change over time. If the answer is time-sensitive and the customer trains on your data months later, the judge will never pass the model's response—and the training data won't be useful.
- **Single Ground Truth Final Answer (GTFA).**
  Prompts should be designed with only one correct final answer. If multiple answers are possible, the model may generate a different (but valid) answer than the one your rubric is written for. Again, the data becomes less useful.
- **Visibility for judges.**
  Depending on the setup, judges may see:

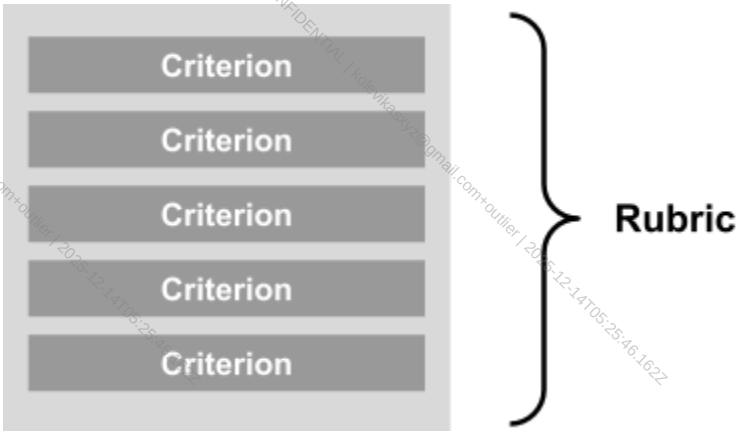
- the entire thinking process,
- just the tool calls, or
- only the final output.
  If a rubric asks the judge to evaluate something it can't see, the model will never pass.

Back to the top ⬆️

Rubrics 101 - How to create perfect rubrics for this project

Back to the basics: What's a rubric?

At its core, a rubric is built from **criteria**—specific checks that a judge will use to evaluate a model's response and decide whether it is correct, incorrect, or partially correct.
.



This simple definition packs a lot inside it. To really understand rubrics, we need to unpack a few things:

- What is a **judge**?
- What is a **model response**?
- What does it mean to be **correct or incorrect**?

How to write a Rubric

To create an effective rubric, first define the evaluation criteria. Every criterion must directly reflect a specific requirement from your prompt.

Rules for Individual Criteria

Each criterion **MUST BE**:

- **Atomic:** Evaluates only one specific aspect or request from the prompt.
- **Specific:** Provides enough detail, including examples, to be clear and not open-ended.
- **Accurate:** Factually correct and logically sound.
- **Categorized:** Properly sorted into its designated category and formatting

What Is a Judge?

A **judge** is either a human or an LLM. Importantly, the judge may not be a subject-matter expert. Its job is to read the rubric criterion and the model's response, then determine if the response **passes, fails, or partially passes**.

This is why we stress making every rubric criterion **self-contained**. A judge doesn't have the original prompt, and it may not fully understand the domain. If the criterion doesn't include everything the judge needs to make a decision, the whole evaluation process breaks down.

Criteria Weights

Each rubric criterion must have a weight between **-40 and 40**, based on its importance to your prompt.

- **Accuracy-related criteria** (e.g., factual correctness, calculations, results) should receive the **highest weights**, since they are the most critical.

- **Instruction-following criteria** (e.g., formatting, phrasing, or minor task requirements) should be assigned **lower weights** than accuracy but still enough to matter.

Rubric's best practices:

A complete rubric **MUST BE**:

- **Comprehensive:** Includes all necessary criteria to define an ideal response, with no missing elements.
- **Relevant:** Contains no unnecessary or unrelated criteria.
- **Accurate:** Composed of objective and factually correct criteria.

- **Non-Repeating:** Avoids redundancy; the same mistake should not be penalized more than once.
- **Self-Contained:** Includes sufficient detail and examples to be easily understood without outside context.
- **Reflective of the Prompt:** Accounts for every explicit request made in the original prompt.

Rubric's rules (These must be present in every rubric):

- **Every rubric MUST have at least 1 criterion that requires a plot:** For any prompt that requires generating a plot, the rubric must include the correct reference plot. Include both verbal criteria (descriptions of what makes the plot correct) and the reference plot to check semantic equivalence. **For this, you'll need to attach a reference on how the plot should look like**
- **Criteria must be precise and specific:** Always be highly specific.
    - ✅ Example: "Number must be 2.342"
    - ❌ Avoid: "Number must be 2"
- **Content Coverage. Rubrics must always mention:**
    - The factual answer to the questions in the prompt.
    - Explicit instructions (even if already stated in previous turns).
    - Implicit instructions (especially those that are subtle or commonly missed).
    - All criteria related to the model's errors, ensuring mistakes can be identified and corrected.
- **Specific format:** Rubric criteria must always start with the **simple present** tense (*e.g., States that…, includes a…, provides a…, mentions that…, etc.*)
- **Must include Negative Criteria:**
    - For every failure that you identify in the model response, you must create a criteria
    - Do not include the correct answer directly under "negative criteria."Negative criteria should instead describe what must not be present in the response.
- **Categorization of Criteria. When writing rubric items, classify them as:**
    - **Must-Have (Essential):**
        - Direct Instruction Following conditions.
        - Explicit asks from the prompt.
    - **Good-to-Have (Nice to Have):**
        - Reasoning quality.
        - Factual correctness beyond explicit asks.
        - Implicit asks or inferred instructions.

Rubric's edge cases (You may need to write some of these):

- **When a prompt requests more than 10 items stacked in a list:**
    - Create spot checks for 20% of items as separate rubrics items. For example:

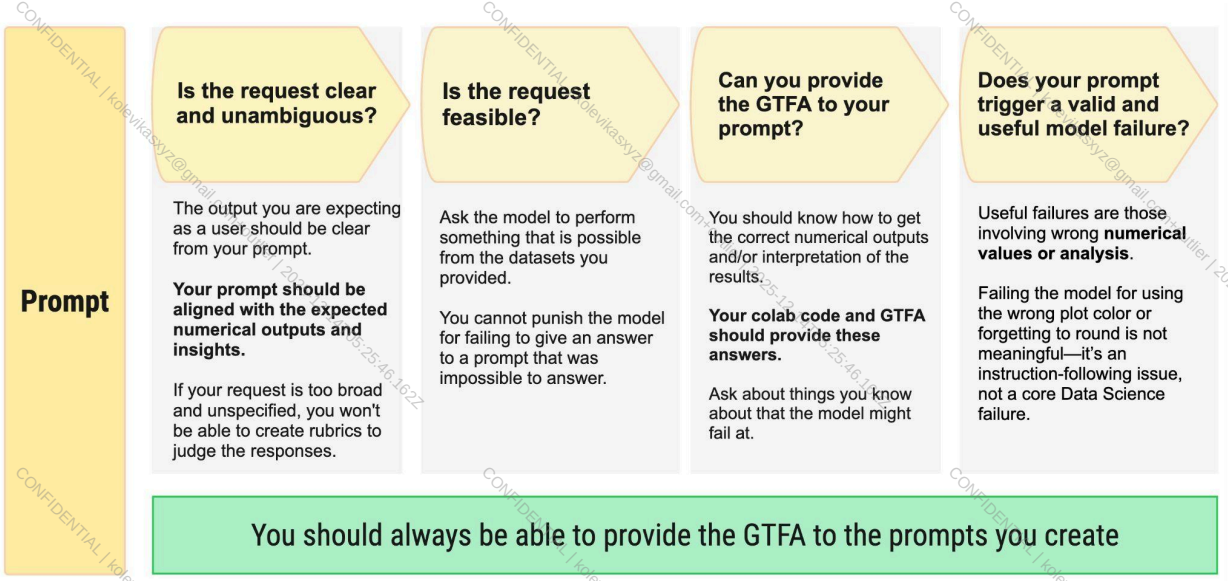
        - *Identifies 22 countries in the choropleth map for the Socio–Environmental Health Index (PC1).*
        - *Includes a Socio–Environmental Health Index (PC1) of 0.1134 for the Philippines in the choropleth map.*
        - *Includes a Socio–Environmental Health Index (PC1) of -0.4835 for Spain in the choropleth map.*
        - *Includes a Socio–Environmental Health Index (PC1) of 0.3559 for Vietnam in the choropleth map.*
    - The items for the value selection should be random: Take them from the beginning, from the middle, and from the end of the list.
    - Note that if the model fails 1 item in a list of 100 items, it is not a good enough failure.

Rubric Example here

Back to the top ⬆️

How to write a great prompt

🏁Goal: Write a prompt that challenges the AI model to analyze, manipulate, and reason with the data.



- **Use casual and natural language**: The questions should be realistic, with no markdown or fancy formatting. Do not be formal: Avoid questions that sound like exam questions!

- **Write hard questions to stump the models**. If the prompt is too easy, the task will need to be redone, as you won't find model failures in the responses or they won't meet the minimum complexity levels for the project.
- **Asking more than one question in one prompt is ok**, but the questions must be centered on the same task/intent, not two completely different topics.
- **If you reference a variable in your prompt**: Use the correct variable name used in the dataset
- **Be precise in your prompt without revealing the solution.**
- **Dataset Dependency:** Every prompt must rely on the attached datasets. The questions should not be answerable without using the datasets.
    - You must use **all** the attached datasets in your prompts
- **Specific and Well-Structured:** Prompts should be clear, detailed, and organized. They may include multiple related questions that require the model to read, manipulate, and reason through the data, as well as run commands to generate results.
- **Realistic Scenarios:** Write prompts that feel natural for a data analyst. Some questions can be slightly contrived, but avoid anything completely unrealistic or irrelevant to real-world analysis.
    - Prompts can be **close-ended** (all experts would arrive at the same answer).
    - Prompts can be **open-ended** (experts could provide different but still factual and valid answers).

📊Guidelines for Designing Complex Data Analysis Prompts

To meet the complexity standard, you must satisfy **both mandatory requirements** and at **least two optional requirements.**

In addition to this table, refer to the DAA Operations Visualizations Categories Table on this LINK to see what calculations and visualizations are under the "Hard" category.

Dimension	🟢Easy (Low Complexity)	🔴Hard (High Complexity)	Mandatory / Optional
Dataset Quality	Uses well-known, simple datasets with limited depth (e.g., Iris, MNIST, weather data).	Uses less common, nuanced datasets with potential for deeper insights (e.g., specialized city-level, financial, or multi-source datasets).	**Mandatory**

In addition to this table, refer to the DAA Operations Visualizations Categories Table on this LINK to see what calculations and visualizations are under the “Hard” category.

Dimension	🟢Easy (Low Complexity)	🔴Hard (High Complexity)	Mandatory / Optional
Number ofRequests	Single, straightforward request (e.g., “Give the mean, median, std of numerical features”).	Multiple, layered requests that require deeper, connected analyses.	**Mandatory**
Analysis Across Datasets	Analyzes each dataset separately with simple methods (e.g., single-variable regression per dataset).	Integrates multiple datasets to explore cross-dataset relationships (e.g., linking income data with crime data to build predictive models).	Optional
Analysis Types	Basic statistics or simple bivariate relationships.	Advanced/multivariate analyses, predictive modeling, or sophisticated statistical techniques.	Optional
Testing	No statistical tests, or only basic tests (t-test, ANOVA).	Advanced testing and statistical methods (e.g., MANOVA, multivariate regression, Bayesian inference).	Optional
Calculations & Methods	Simple, obvious calculations (mean, percentages, sums) easily done in Excel.	Complex/nested calculations (groupings, filters) and advanced methods (Markov chains, Difference-in-Differences, instrumental variable analysis, etc.).	Optional

In addition to this table, refer to the DAA Operations Visualizations Categories Table on this LINK to see what calculations and visualizations are under the “Hard” category.

Dimension	🟢Easy (Low Complexity)	🔴Hard (High Complexity)	Mandatory / Optional
Visualizations	Simple descriptive plots (bar charts, line charts, histograms).	Multivariate or advanced plots (PCA biplot, swarmplots) and visualizations that uncover deeper relationships.	Optional
Creativity of Insights			Optional

Prompt Examples

Prompt Example	Difficulty
I just started working as a data analyst, and I was assigned a dataset of used cars (Used_Car_Price_Prediction.csv). My task is very specific, and my manager will check exact numbers. *Please can you help with the following things:* *Clean the data:* *Convert sale_price, broker_quote, original_price, emi_starts_from, booking_down_pymnt into numeric rupees (parse “₹ 3.5 Lakh” → 350000, “1.2 Cr” → 12000000).* *Convert kms_run into numeric kilometers (e.g., “45,000 km” → 45000).* *Convert total_owners like “1st Owner”, “Second Owner” into numbers (1,2,...).* *Drop rows with missing values in any of those columns.*	Hard

Prompt Example	Difficulty
Fit a multiple linear regression where the target is sale_price and the predictors are: yr_mfr, kms_run, times_viewed, total_owners, broker_quote, original_price, car_rating, emi_starts_from, booking_down_pymnt. Use only the ones that survive cleaning. *Report these exact metrics from the fitted model (up to 16 digits precision is good):* *a. R^2 and Adjusted R^2* *b. Standardized (beta) coefficients for each predictor* *c. p-values for each predictor* *Check multicollinearity: compute the Variance Inflation Factor (VIF) for every predictor, and clearly list which ones (if any) are > 10.* *Generate a Residuals vs Fitted plot for the model and run a Breusch–Pagan test. Based on that, answer clearly: "Is heteroscedasticity present? Yes or No."* *Do a drop-one analysis:* *Remove each predictor one at a time, refit, and compute the adjusted R^2.* *Identify the single predictor whose removal gives the largest increase in adjusted R^2, and state the new adjusted R^2 value.* *Refit the final model without that predictor, make a Residuals vs Fitted plot, and repeat the Breusch–Pagan test. Report the adjusted R^2 and whether heteroscedasticity is still present.*	
I'm sharing datasets that summarize many years of sunflower field experiments in Colorado, tracking daily canopy cover percentage, precipitation, irrigation amount, growth stage, and treatment group. Each plot received a different irrigation regime. In early years (2008–2011), there were 6 treatments: - Treatment 1: 100% of crop water requirements (full irrigation) - Treatment 2: 85% of Treatment 1 - Treatment 3: 75% of Treatment 1 - Treatment 4: 70% of Treatment 1 - Treatment 5: 55% of Treatment 1 - Treatment 6: 40% of Treatment 1 (most severe deficit) In later years (2012–2016), up to 12 combinations of irrigation were used (reflecting percent of full irrigation provided during vegetative and maturation stages): - 100/100, 100/50, 80/80, 80/65, 80/50, 80/40, 65/80, 65/65, 65/50, 65/40, 50/50, 40/40 All treatments are defined as the target percentage of crop water requirement met by irrigation during these stages. Your Tasks: Devise an analysis strategy that properly accounts for the structure of the experimental design, especially the potential for correlations among measurements taken from within the same year. Identify and quantify the key factors that most strongly influence canopy cover development over the season, considering water input (precipitation and irrigation), crop growth stage,	Hard

Prompt Example	Difficulty
and irrigation treatment. Which specific treatment(s) have the greatest effect, positive or negative, on canopy cover? Support your answer with evidence from the data. Use at least one impactful plot which highlights the key findings.	
You are given a dataset with dozens of numeric variables from experimental field plots, including water stress (Ks) and many crop, soil, and weather measurements. ***Perform the following:*** *1. Compute a maximum-entropy probability distribution (under the constraint of all observed marginal means and variances remaining fixed) for the entire multivariate dataset.* *2. Calculate the multivariate mutual information (total information shared) between water stress coefficient (Ks) and all other variables in this maximum-entropy model.* *3. Identify the subset of up to five variables (excluding Ks) for which the conditional mutual information with Ks, given all other variables, is maximized.* *4. For this subset, calculate the partial correlation matrix and the determinant of that submatrix.* ***Specifically report:*** *5. The total (multivariate) mutual information between Ks and the rest of the system.* *6. The names of the five variables (or fewer, if fewer are found) that maximize the conditional mutual information with Ks, and the numeric value of that maximal conditional information.* *7. The determinant of the partial correlation matrix for that subset, and interpret its value in terms of statistical dependence.*	Hard

Back to the top ⬆️

Appendix

🕸️ Reference Card - Check This Before Submission!

*🎉A **high-quality task** consistently checks all these boxes. Use this as your quick reference before submitting your task.*

🗒️ 1. Dataset Requirements

- ☐ Use at least 3 datasets. CSV, TSV, XML, or JSON formats are accepted.
- ☐ Datasets must be public and open source. No paywalls, logins, or proprietary data.
- ☐ Ensure sufficient size and structure. Datasets should be large enough to allow meaningful analysis (ideally 5+ columns, 100+ rows).
- ☐ All uploaded datasets must be used in your prompt, none can be left unused.

🧠 2. Prompt Quality

- ☐ Write a strong, complex prompt. It should be clear, realistic, and challenging enough to make the model fail naturally.
- ☐ Follow the assigned prompt type. Prompt type might be either open-ended (multiple valid answers) or close-ended (single correct answer).
- ☐ Require at least one visualization. The prompt must include a request for a plot or graph.

🧩 3. Rubric Standards (ignore for Stage 1)

- ☐ Create complete, precise rubrics. Each criterion should fully describe what makes a response correct.
- ☐ Use correct formatting. Start every criterion in the simple present tense (e.g., States that…, Includes a…, Mentions that…). Avoid “The response must/should.”
- ☐ Include minimum 10 positive and minimum 10 negative rubrics. For every model error you identify, write a matching negative criterion.

🏆 4. Model Failure Requirement

- [ ] Ensure a genuine model failure. Your prompt must be complex enough to cause a natural, non-trivial mistake (not just formatting or minor errors). Valid model failures are those involving numerical values or analysis related to Data Analysis.

Task Examples

Use these examples as inspiration and to learn what a good task looks like.
Do not copy them. We've already found several tasks requesting the same things that appear on these instructions (e.g: Pearson correlations).

Be creative 💡 and use other Data Science topics for your tasks.

Task Example	
Datasets	https://www.kaggle.com/datasets/patricklford/largest-companies-analysis-worldwide?select=Companies_ranked_by_Revenue.csv https://www.kaggle.com/datasets/patricklford/largest-companies-analysis-worldwide?select=Companies_ranked_by_Earnings.csv https://www.kaggle.com/datasets/patricklford/largest-companies-analysis-worldwide?select=Companies_ranked_by_Market_Cap.csv
Prompt	Hey, I'm analyzing how global company valuation patterns connect with their size and profitability using the three datasets ranking firms by revenue, earnings, and market cap. Could you merge them by Symbol, keeping only Symbol, Name, country, price (GBP), revenue_ttm, earnings_ttm, and marketcap as numeric fields? After cleaning, create margin = earnings_ttm/revenue_ttm and valuation_multiple = marketcap/earnings_ttm. Standardize the three base metrics and compute both Pearson and Spearman correlations among all five variables. Then apply PCA on the standardized base metrics (revenue_ttm, earnings_ttm, marketcap), retain two components, and cluster companies in that space using k-means with k chosen by silhouette score. Once clusters are formed, compare the PCA-based segmentation with a second method using t-SNE on the same standardized data to see if cluster boundaries remain consistent. Finally, make one plot showing the PCA biplot with points colored by k-means cluster and arrows for the three variables, and briefly interpret what drives each component and how differences between the PCA and t-SNE patterns could inform investment or valuation strategies.

Model Failure Explanation	The prompt explicitly asks for a brief interpretation explaining which financial indicators drive each PCA component and how differences between PCA and t-SNE clustering patterns could inform investment or valuation strategies. The model produced a generic template (“PC1 is size, PC2 is profitability”) without computing or citing the actual PCA loadings or using the observed clustering results. This makes the interpretation speculative rather than data-driven. Second, the comparison between PCA and t-SNE is incomplete. Although the script reports numerical values (ARI = 0.0002, t-SNE silhouette = 0.3871), it never provides an analytical conclusion about cluster consistency or what those figures imply. The task requires a qualitative evaluation of whether the t-SNE pattern confirms or contradicts the PCA segmentation.
Positive Rubric Criteria	NOTE: The numbers inside the brackets [] are the weights assigned to each criterion. [35] Confirms that the final cleaned dataset contains the nine required columns of Symbol, Name, country, price (GBP), revenue_ttm, earnings_ttm, marketcap, margin, and valuation_multiple. [35] States that the final dataset includes exactly 9911 rows after cleaning. [35] Reports the Pearson correlation between revenue_ttm and earnings_ttm as 0.6729. [35] Reports the Pearson correlation between revenue_ttm and marketcap as 0.5483. [35] Reports the Pearson correlation between earnings_ttm and marketcap as 0.7776. [35] Reports the Pearson correlation between revenue_ttm and margin as 0.0065. [35] Reports the Pearson correlation between revenue_ttm and valuation_multiple as 0.0025. [35] Reports the Pearson correlation between earnings_ttm and margin as 0.0065. [35] Reports the Pearson correlation between marketcap and margin as 0.0038. [35] Reports the Pearson correlation between earnings_ttm and valuation_multiple as 0.0018. [35] Reports the Pearson correlation between marketcap and valuation_multiple as -0.0006. [35] Reports the Spearman correlation between revenue_ttm and earnings_ttm as 0.7108. [35] Reports the Spearman correlation between revenue_ttm and marketcap as 0.7828. [35] Reports the Spearman correlation between earnings_ttm and marketcap as 0.7021. [35] Reports the Spearman correlation between revenue_ttm and margin as 0.2341. [35] Reports the Spearman correlation between revenue_ttm and valuation_multiple as 0.2721. [35] Reports the Spearman correlation between earnings_ttm and margin as 0.6948. [35] Reports the Spearman correlation between marketcap and margin as 0.4119. [35] Reports the Spearman correlation between earnings_ttm and valuation_multiple as 0.4961. [35] Reports the Spearman correlation between marketcap and valuation_multiple as 0.3741. [35] Reports the explained variance ratio for PC1 as 0.7790. [35] Reports the explained variance ratio of PC2 as 0.1548. [40] Reports the loading for revenue_ttm on PC1 as 0.5435. [40] Reports the loading of revenue_ttm on PC2 as 0.7993. [35] Reports the loading of earnings_ttm on PC1 as 0.6094. [35] Reports the loading of earnings_ttm on PC2 as -0.1657. [35] Reports the loading of marketcap on PC1 as 0.5773. [35] Reports the loading of marketcap on PC2 as -0.5776. [35] Reports that the chosen number of clusters from k-means is 3. [30] Reports the silhouette score in PCA space as 0.927. [30] Reports the t-SNE silhouette score as 0.391.

	[30] Reports the Adjusted Rand Index as -0.0035. [40] Generates a PCA biplot that is semantically the same as the attached plot. [35] States that the PCA biplot displays three labeled arrows for revenue_ttm, earnings_ttm, and marketcap originating from (0, 0). [30] States that the PCA biplot includes a legend labeling clusters with distinct colors. [30] Reports that PC1 is primarily driven by earnings_ttm. [30] Reports that PC2 is primarily driven by revenue_ttm. [25] Reports that differences between PCA and t-SNE clustering patterns suggest that nonlinear structures in valuation may reveal distinct investment segments.
Negative Rubric Criteria	NOTE: The numbers inside the brackets [] are the weights assigned to each criterion. [-9] States that a high Pearson correlation coefficient establishes a causal link between profitability and market capitalization. [-9] Asserts that higher revenue directly influences market capitalization based solely on correlation. [-7] Claims that increased revenue and earnings naturally lead to increased market valuation, presenting a deterministic outcome from association. [-4] Concludes that variables with low Pearson correlations have no significant impact or are independent without considering non-linear relationships. [-10] Claims that internal clustering validation (silhouette score) guarantees predictable returns or stability for investment profiles. [-4] Interprets PCA loadings as confirming a variable's role as a primary determinant of market standing. [-4] Asserts that principal components represent independent, fundamental drivers as perceived by market participants. [-9] Claims a high silhouette score unequivocally confirms real-world validity and economic significance of clusters. [-4] States that a distinct color coding in a PCA biplot solidifies genuine economic separation. [-5] Interprets PCA biplot arrows as direction and magnitude of influence on cluster formation, implying causality from loadings.
Quality Control Notes	Prompt Goal: Unify three worldwide company rankings (revenue, earnings, market cap) by Symbol, coerce numeric fields, engineer margin and valuation_multiple, compute Pearson and Spearman correlations across the five variables, run PCA on standardized base metrics keeping two components, segment companies in PCA space via k-means with k chosen by silhouette, compare that segmentation with t-SNE on the same standardized data, and deliver one PCA biplot colored by k-means cluster plus a brief, data-driven interpretation. 1. Read CSVs, merges by symbol, engineers, ratios. 2. Computes Pearson and Spearman correlations. 3. Runs PCA with 2 components. 4. Clusters in PCA space cia k-means with k chosen by silhouette. 5. Compares with t-SNE clusters. 6. Draws a PCA biplot. 7. Write interpretations. ###Step 1: Load the three CSVs and select consistent columns.

Ensure each file contributes Symbol, Name, country, price (GBP) and its metric column (revenue_ttm, earnings_ttm, marketcap). Handle header variations and numeric strings with commas, currency symbols, and K/M/B/T suffixes.

```
base_path = "/content"
paths = {
    "revenue": os.path.join(base_path, "Companies_ranked_by_Revenue.csv"),
    "earnings": os.path.join(base_path, "Companies_ranked_by_Earnings.csv"),
    "mcap": os.path.join(base_path, "Companies_ranked_by_Market_Cap.csv"),
}

for key, p in paths.items():
    if not os.path.exists(p):
        raise FileNotFoundError(f"Missing file: {p}")

rev  = pd.read_csv(paths["revenue"])
earn = pd.read_csv(paths["earnings"])
mcap = pd.read_csv(paths["mcap"])

rev  = select_core_columns(rev,  "revenue_ttm")
earn = select_core_columns(earn, "earnings_ttm")
mcap = select_core_columns(mcap, "marketcap")
```

Key parser used throughout:

```
def parse_number(x):
    """Coerce strings like '1,234.5', '(1.2B)', '£3,000' into float."""
    if pd.isna(x): return np.nan
    if isinstance(x, (int, float, np.number)): return float(x)
    s = str(x).strip()
    s = s.replace(',', '')
    s = re.sub(r'[€£$¥₹]', '', s)
```

###Step 2: Merge by Symbol and coalesce duplicates

inner-join so each company appears once, resolving Name, country, and price (GBP) from any file if duplicated.

```
merged = (
    rev.merge(earn, on="Symbol", how="inner", suffixes=("","_earn"))
       .merge(mcap, on="Symbol", how="inner", suffixes=("","_mcap"))
)

# Coalesce Name, country, price (GBP) across files if duplicates exist
def coalesce(*series):
    out = series[0].copy()
    for s in series[1:]:
        out = out.combine_first(s)
    return out
```

merged["Name"]   = coalesce(merged.get("Name"), merged.get("Name_earn"),
merged.get("Name_mcap"))
merged["country"]   = coalesce(merged.get("country"), merged.get("country_earn"),
merged.get("country_mcap"))
merged["price (GBP)"] = coalesce(merged.get("price (GBP)"), merged.get("price (GBP)_earn"),
merged.get("price (GBP)_mcap"))

merged = merged[["Symbol","Name","country","price
(GBP)","revenue_ttm","earnings_ttm","marketcap"]].copy()
```
###Step 3: Engineer ratios exactly as requested.
Formulas:
- margin = earnings_ttm / revenue_ttm (guard against zero revenue)
- valuation_multiple = marketcap / earnings_ttm (guard against zero earnings)
```
merged["margin"] = np.where(merged["revenue_ttm"] > 0,
merged["earnings_ttm"]/merged["revenue_ttm"], np.nan)
merged["valuation_multiple"] = np.where(merged["earnings_ttm"] != 0,
merged["marketcap"]/merged["earnings_ttm"], np.nan)

final_cols = ["Symbol","Name","country","price
(GBP)","revenue_ttm","earnings_ttm","marketcap","margin","valuation_multiple"]
merged = merged[final_cols]

# Report table structure
print("Final columns:", final_cols)
```
###Step 4: Compute Pearson and Spearman correlation matrices.
Variables: revenue_ttm, earnings_ttm, marketcap, margin, valuation_multiple.
```
corr_vars = ["revenue_ttm","earnings_ttm","marketcap","margin","valuation_multiple"]
corr_pearson = merged[corr_vars].corr(method="pearson")
corr_spearman = merged[corr_vars].corr(method="spearman")

print("\nPearson correlations (five variables):")
```
###Step 5: Standardize base metrics and run PCA (2 components).
Z-score each base metric to unit scale, then decompose; report loadings and explained variance ratios for PC1 and PC2.
```
X_base = merged[["revenue_ttm","earnings_ttm","marketcap"]].values
scaler = StandardScaler()
X_std = scaler.fit_transform(X_base)

pca = PCA(n_components=2, random_state=42)
PC = pca.fit_transform(X_std)
explained = pca.explained_variance_ratio_
loadings = pca.components_.T # shape (3, 2) for [revenue, earnings, mcap] x [PC1, PC2]
```
```

```
loadings_df = pd.DataFrame(loadings, index=["revenue_ttm","earnings_ttm","marketcap"],
columns=["PC1","PC2"])
print("\nPCA loadings (rows=variables, cols=components):")
print(loadings_df.round(4))
print(f"\nExplained variance ratios: PC1={explained[0]:.4f}, PC2={explained[1]:.4f}")
```
###Step 6: Cluster in PCA space with k-means; choose k by silhouette.
Use a fixed candidate set for reproducibility; keep the k with the highest average silhouette.
```
k_candidates = [3,4,5,6]
best = {"k": None, "score": -1, "labels": None}
for k in k_candidates:
    km = KMeans(n_clusters=k, n_init=50, max_iter=1000, random_state=42)
    labels = km.fit_predict(PC)
    score = silhouette_score(PC, labels)
    if score > best["score"]:
        best.update({"k": k, "score": score, "labels": labels})

merged["cluster_kmeans_pca"] = best["labels"]
print(f"\nChosen k (by silhouette over {k_candidates}): {best['k']}  |  Silhouette (PCA space):
{best['score']:.3f}")
```
###Step 7: Compare with t-SNE on the same standardized data
Cluster t-SNE embedding with the same k, then quantify agreement via ARI; also report t-SNE
silhouette.
```
tsne = TSNE(n_components=2, random_state=42, perplexity=30, init="pca", n_iter=750,
learning_rate="auto")
TS = tsne.fit_transform(X_std)

km_ts = KMeans(n_clusters=best["k"], n_init=50, max_iter=1000, random_state=42)
labels_ts = km_ts.fit_predict(TS)

ari = adjusted_rand_score(best["labels"], labels_ts)
sil_ts = silhouette_score(TS, labels_ts)
print(f"t-SNE silhouette: {sil_ts:.3f}  |  ARI (PCA k-means vs t-SNE k-means): {ari:.4f}")
```
###Step 8: Produce the single required figure (PCA biplot) and persist outputs.
Plot PC1 vs PC2, color by k-means cluster, draw loading arrows for the three variables, and save
both the figure and the merged dataset + embeddings.
```
plt.figure(figsize=(9,7))
for cl in sorted(np.unique(best["labels"])):
    idx = (best["labels"] == cl)
    plt.scatter(PC[idx,0], PC[idx,1], alpha=0.85, label=f"Cluster {cl}")

# draw loading arrows
arrow_scale = 2.8
vars_order = ["revenue_ttm","earnings_ttm","marketcap"]
for i, var in enumerate(vars_order):
```

```
    x = loadings[i,0] * arrow_scale
    y = loadings[i,1] * arrow_scale
    plt.arrow(0, 0, x, y, head_width=0.1, head_length=0.12, length_includes_head=True)
    plt.text(x*1.06, y*1.06, var, fontsize=10)

plt.xlabel(f"PC1 ({explained[0]*100:.1f}% var)")
plt.ylabel(f"PC2 ({explained[1]*100:.1f}% var)")
plt.title("PCA biplot: companies clustered by k-means")
plt.legend()
plt.grid(True, linestyle=":")
plt.tight_layout()

# Save outputs to /content/
fig_path = "/content/pca_biplot_clusters.png"
csv_path = "/content/companies_merged_clean.csv"
plt.savefig(fig_path, dpi=160)
merged.assign(PC1=PC[:,0], PC2=PC[:,1], TSNE1=TS[:,0], TSNE2=TS[:,1]).to_csv(csv_path,
index=False)

print(f"\nSaved figure to: {fig_path}")
```
###Step 9: Print the brief, data-driven interpretation
Tie the numerical outputs to conclusions that answer the prompt.
```
def brief_interpretation(load_df, evr, pearson, spearman, k, sil_pca, sil_ts, ari_score):
    # Which variables dominate each PC?
    pc1_driver = load_df['PC1'].abs().idxmax()
    pc2_driver = load_df['PC2'].abs().idxmax()

    lines = []
    lines.append("Interpretation:")
    lines.append(f"- PC1 is primarily driven by {pc1_driver} (highest absolute loading), with variance
share {evr[0]:.2%}.")
    lines.append(f"- PC2 is primarily driven by {pc2_driver} (highest absolute loading), with variance
share {evr[1]:.2%}.")
    lines.append(f"- Correlations show the size triangle among revenue_ttm, earnings_ttm, and
marketcap "
                 f"(Pearson min={pearson.loc[['revenue_ttm','earnings_ttm','marketcap'],
['revenue_ttm','earnings_ttm','marketcap']].replace(1.0,np.nan).min().min():.3f}; "
                 f"Spearman min={spearman.loc[['revenue_ttm','earnings_ttm','marketcap'],
['revenue_ttm','earnings_ttm','marketcap']].replace(1.0,np.nan).min().min():.3f}).")
    lines.append(f"- K-means selected k={k} in PCA space (silhouette={sil_pca:.3f}).")
    lines.append(f"- t-SNE clusters have silhouette={sil_ts:.3f} and ARI={ari_score:.4f} vs PCA clusters;
"
                 f"low ARI suggests that non-linear neighborhoods do not perfectly align with the linear PCA
segmentation.")
    lines.append("- For valuation use, firms loading strongly on the earnings direction with moderate
revenue often exhibit higher valuation_multiple, "
                 "while margin is informative for profiling but not used in the PCA construction.")
    return "\n".join(lines)
```

```
print()
print(brief_interpretation(loadings_df, explained, corr_pearson, corr_spearman,
                           best["k"], best["score"], sil_ts, ari))
```

Complete Code:
```
import os, re, math, warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, adjusted_rand_score
from sklearn.manifold import TSNE

warnings.filterwarnings("ignore", category=FutureWarning)

# --------------------------
# 0) Utility helpers
# --------------------------
def parse_number(x):
    """Coerce strings like '1,234.5', '(1.2B)', '£3,000' into float."""
    if pd.isna(x): return np.nan
    if isinstance(x, (int, float, np.number)): return float(x)
    s = str(x).strip()
    s = s.replace(',', '')
    s = re.sub(r'[€£$¥₹]', '', s)
    neg = False
    if s.startswith('(') and s.endswith(')'):
        neg = True
        s = s[1:-1]
    m = re.match(r'^([+-]?\d*\.?\d+)\s*([TtBbMmKk]?)$', s)
    if m:
        num = float(m.group(1))
        suf = m.group(2).upper()
        mult = {'':1,'K':1e3,'M':1e6,'B':1e9,'T':1e12}[suf]
        return -num*mult if neg else num*mult
    try:
        return -float(s[1:]) if neg else float(s)
    except:
        return np.nan

def select_core_columns(df, metric_col_expected):
    """Return a standardized frame with columns:
       symbol, name, country, price_gbp, <metric_col_expected>"""
    cols = df.columns
```

```
    # Normalize likely column names
    def pick(cands):
        cands_norm = [re.sub(r'[^a-z0-9]+','_', c.lower()) for c in cols]
        for cand in cands:
            cand_norm = re.sub(r'[^a-z0-9]+','_', cand.lower())
            if cand_norm in cands_norm:
                return cols[cands_norm.index(cand_norm)]
        # fallback: fuzzy contains
        for i, cn in enumerate(cands_norm):
            for cand in cands:
                cand_norm = re.sub(r'[^a-z0-9]+','_', cand.lower())
                if cand_norm in cn:
                    return cols[i]
        return None

    symbol = pick(["Symbol","Ticker"])
    name   = pick(["Name","Company"])
    country= pick(["Country"])
    price  = pick(["price (GBP)","price_gbp","Price"])
    metric = pick([metric_col_expected])

    keep = [c for c in [symbol,name,country,price,metric] if c is not None]
    out = df[keep].copy()

    rename = {}
    if symbol: rename[symbol] = "Symbol"
    if name:   rename[name]   = "Name"
    if country:rename[country]= "country"
    if price:  rename[price]  = "price (GBP)"
    if metric: rename[metric] = metric_col_expected
    out = out.rename(columns=rename)

    # De-duplicate by Symbol (keep first occurrence)
    out = out.dropna(subset=["Symbol"]).drop_duplicates(subset=["Symbol"], keep="first")

    # Coerce numerics
    for c in ["price (GBP)", metric_col_expected]:
        if c in out.columns:
            out[c] = out[c].apply(parse_number)
    return out

# --------------------------
# 1) Load datasets from /content/
# --------------------------
base_path = "/content"
paths = {
    "revenue": os.path.join(base_path, "Companies_ranked_by_Revenue.csv"),
    "earnings": os.path.join(base_path, "Companies_ranked_by_Earnings.csv"),
    "mcap": os.path.join(base_path, "Companies_ranked_by_Market_Cap.csv"),
}
```

```
for key, p in paths.items():
    if not os.path.exists(p):
        raise FileNotFoundError(f"Missing file: {p}")

rev  = pd.read_csv(paths["revenue"])
earn = pd.read_csv(paths["earnings"])
mcap = pd.read_csv(paths["mcap"])

rev  = select_core_columns(rev,  "revenue_ttm")
earn = select_core_columns(earn, "earnings_ttm")
mcap = select_core_columns(mcap, "marketcap")

# ---------------------------
# 2) Merge by Symbol (inner)
# Keep only the required columns in the exact order.
# ---------------------------
merged = (
    rev.merge(earn, on="Symbol", how="inner", suffixes=("","_earn"))
       .merge(mcap, on="Symbol", how="inner", suffixes=("","_mcap"))
)

# Coalesce Name, country, price (GBP) across files if duplicates exist
def coalesce(*series):
    out = series[0].copy()
    for s in series[1:]:
        out = out.combine_first(s)
    return out

merged["Name"]        = coalesce(merged.get("Name"), merged.get("Name_earn"),
merged.get("Name_mcap"))
merged["country"]     = coalesce(merged.get("country"), merged.get("country_earn"),
merged.get("country_mcap"))
merged["price (GBP)"] = coalesce(merged.get("price (GBP)"), merged.get("price (GBP)_earn"),
merged.get("price (GBP)_mcap"))

merged = merged[["Symbol","Name","country","price
(GBP)","revenue_ttm","earnings_ttm","marketcap"]].copy()

# Drop rows with nonpositive or missing core metrics (safeguard for ratios)
for col in ["revenue_ttm","earnings_ttm","marketcap"]:
    merged[col] = merged[col].astype(float)
merged = merged.replace([np.inf, -np.inf],
np.nan).dropna(subset=["revenue_ttm","earnings_ttm","marketcap"])

# ---------------------------
# 3) Feature engineering: margin & valuation_multiple
# ---------------------------
merged["margin"] = np.where(merged["revenue_ttm"] > 0,
merged["earnings_ttm"]/merged["revenue_ttm"], np.nan)
```

```
merged["valuation_multiple"] = np.where(merged["earnings_ttm"] != 0,
merged["marketcap"]/merged["earnings_ttm"], np.nan)

final_cols = ["Symbol","Name","country","price
(GBP)","revenue_ttm","earnings_ttm","marketcap","margin","valuation_multiple"]
merged = merged[final_cols]

# Report table structure
print("Final columns:", final_cols)
print("Final row count:", len(merged))

# ---------------------------
# 4) Correlations (Pearson & Spearman) for the five variables
# ---------------------------
corr_vars = ["revenue_ttm","earnings_ttm","marketcap","margin","valuation_multiple"]
corr_pearson  = merged[corr_vars].corr(method="pearson")
corr_spearman = merged[corr_vars].corr(method="spearman")

print("\nPearson correlations (five variables):")
print(corr_pearson.round(4))
print("\nSpearman correlations (five variables):")
print(corr_spearman.round(4))

# ---------------------------
# 5) PCA on standardized base metrics (retain 2 components)
# ---------------------------
X_base = merged[["revenue_ttm","earnings_ttm","marketcap"]].values
scaler = StandardScaler()
X_std = scaler.fit_transform(X_base)

pca = PCA(n_components=2, random_state=42)
PC = pca.fit_transform(X_std)
explained = pca.explained_variance_ratio_
loadings = pca.components_.T  # shape (3, 2) for [revenue, earnings, mcap] x [PC1, PC2]

loadings_df = pd.DataFrame(loadings, index=["revenue_ttm","earnings_ttm","marketcap"],
columns=["PC1","PC2"])
print("\nPCA loadings (rows=variables, cols=components):")
print(loadings_df.round(4))
print(f"\nExplained variance ratios: PC1={explained[0]:.4f}, PC2={explained[1]:.4f}")

# ---------------------------
# 6) K-means in PCA space with k chosen by silhouette
#     Use a fixed candidate set for reproducibility.
# ---------------------------
k_candidates = [3,4,5,6]
best = {"k": None, "score": -1, "labels": None}
for k in k_candidates:
    km = KMeans(n_clusters=k, n_init=50, max_iter=1000, random_state=42)
    labels = km.fit_predict(PC)
```

```
    score = silhouette_score(PC, labels)
    if score > best["score"]:
        best.update({"k": k, "score": score, "labels": labels})

merged["cluster_kmeans_pca"] = best["labels"]
print(f"\nChosen k (by silhouette over {k_candidates}): {best['k']}  |  Silhouette (PCA space):
{best['score']:.3f}")

# --------------------------
# 7) t-SNE on the same standardized data + clustering comparison
# --------------------------
tsne = TSNE(n_components=2, random_state=42, perplexity=30, init="pca", n_iter=750,
learning_rate="auto")
TS = tsne.fit_transform(X_std)

km_ts = KMeans(n_clusters=best["k"], n_init=50, max_iter=1000, random_state=42)
labels_ts = km_ts.fit_predict(TS)

ari = adjusted_rand_score(best["labels"], labels_ts)
sil_ts = silhouette_score(TS, labels_ts)
print(f"t-SNE silhouette: {sil_ts:.3f}  |  ARI (PCA k-means vs t-SNE k-means): {ari:.4f}")

# --------------------------
# 8) PCA biplot with k-means clusters (single required figure)
# --------------------------
plt.figure(figsize=(9,7))
for cl in sorted(np.unique(best["labels"])):
    idx = (best["labels"] == cl)
    plt.scatter(PC[idx,0], PC[idx,1], alpha=0.85, label=f"Cluster {cl}")

# draw loading arrows
arrow_scale = 2.8
vars_order = ["revenue_ttm","earnings_ttm","marketcap"]
for i, var in enumerate(vars_order):
    x = loadings[i,0] * arrow_scale
    y = loadings[i,1] * arrow_scale
    plt.arrow(0, 0, x, y, head_width=0.1, head_length=0.12, length_includes_head=True)
    plt.text(x*1.06, y*1.06, var, fontsize=10)

plt.xlabel(f"PC1 ({explained[0]*100:.1f}% var)")
plt.ylabel(f"PC2 ({explained[1]*100:.1f}% var)")
plt.title("PCA biplot: companies clustered by k-means")
plt.legend()
plt.grid(True, linestyle=":")
plt.tight_layout()

# Save outputs to /content/
fig_path = "/content/pca_biplot_clusters.png"
csv_path = "/content/companies_merged_clean.csv"
plt.savefig(fig_path, dpi=160)
```

```
merged.assign(PC1=PC[:,0], PC2=PC[:,1], TSNE1=TS[:,0], TSNE2=TS[:,1]).to_csv(csv_path,
index=False)

print(f"\nSaved figure to: {fig_path}")
print(f"Saved merged dataset to: {csv_path}")

# --------------------------
# 9) Brief interpretation (data-driven, based on loadings & stats)
# --------------------------
def brief_interpretation(load_df, evr, pearson, spearman, k, sil_pca, sil_ts, ari_score):
    # Which variables dominate each PC?
    pc1_driver = load_df['PC1'].abs().idxmax()
    pc2_driver = load_df['PC2'].abs().idxmax()

    lines = []
    lines.append("Interpretation:")
    lines.append(f"- PC1 is primarily driven by {pc1_driver} (highest absolute loading), with variance
share {evr[0]:.2%}.")
    lines.append(f"- PC2 is primarily driven by {pc2_driver} (highest absolute loading), with variance
share {evr[1]:.2%}.")
    lines.append(f"- Correlations show the size triangle among revenue_ttm, earnings_ttm, and
marketcap "
                 f"(Pearson min={pearson.loc[['revenue_ttm','earnings_ttm','marketcap'],
['revenue_ttm','earnings_ttm','marketcap']].replace(1.0,np.nan).min().min():.3f}; "
                 f"Spearman min={spearman.loc[['revenue_ttm','earnings_ttm','marketcap'],
['revenue_ttm','earnings_ttm','marketcap']].replace(1.0,np.nan).min().min():.3f}).")
    lines.append(f"- K-means selected k={k} in PCA space (silhouette={sil_pca:.3f}).")
    lines.append(f"- t-SNE clusters have silhouette={sil_ts:.3f} and ARI={ari_score:.4f} vs PCA clusters;
"
                 f"low ARI suggests that non-linear neighborhoods do not perfectly align with the linear PCA
segmentation.")
    lines.append("- For valuation use, firms loading strongly on the earnings direction with moderate
revenue often exhibit higher valuation_multiple, "
                 "while margin is informative for profiling but not used in the PCA construction.")
    return "\n".join(lines)

print()
print(brief_interpretation(loadings_df, explained, corr_pearson, corr_spearman,
                           best["k"], best["score"], sil_ts, ari))
```

Negative Criteria System Prompt

None

Prompt: Generate Analytical Failure Ideas with Explanations

You are a state-of-the-art reasoning model being tested for subtle factual and reasoning reliability in data-science and applied economics tasks.
You will be given a Prompt and a Rubric.
The rubric defines ideal analytical outcomes — your mission is to imagine how those outcomes could fail in realistic, outcome-level ways.

Your Mission

Generate Failure Ideas — plausible yet analytically incorrect lines of reasoning that an advanced model could produce when answering the given Prompt.

Each idea should include both:

A false or misleading analytical conclusion, and

A brief explanation of how and why that reasoning would mislead an analyst, policymaker, or investor.

Focus strictly on interpretive or inferential errors, not technical or workflow issues.

Step 1 — Identify Failure Themes

Think of 10–15 possible themes of reasoning failure, such as:

Mistaking correlation for causation

Overinterpreting descriptive clusters

Ignoring context or heterogeneity

Treating statistical fit as proof of truth

Inferring policy or business causality from observational data

Step 2 — Write Structured Failure Ideas

For each theme, produce a Failure Idea entry using the following format:

Title: <short label capturing the reasoning trap>

- Failure Idea: <one to two paragraphs describing a plausible but false analytical conclusion, and a short explanation of why it is wrong.>

Each explanation should:

Sound realistic — as if written by a smart but overconfident analyst.

Clarify why the conclusion is incorrect, referencing sound economic or statistical reasoning.

Mention how such an error could arise (e.g., overreliance on correlation, ignoring confounders, misreading model output).

Avoid superficial or procedural points like “data not normalized” or “forgot to drop nulls.”

Quality Criteria for Failure Ideas

Outcome-level: the claim changes the interpretation of results or real-world meaning.

Self-contained: understandable without reading the original prompt.

Analytically rich: includes reasoning and a short reflection on why it’s wrong.

Neutral tone: no evaluative words like “wrongly,” “incorrectly,” or “fails to.”

Example Output
Title: Correlation Interpreted as Causation
- Failure Idea: The analyst concludes that higher profitability directly causes higher market valuation across all industries. This reasoning assumes that observed correlation implies a causal mechanism, ignoring that larger or more mature firms may both be more profitable and more highly valued due to unrelated structural factors. The apparent relationship is associative, not causal, and the conclusion overstates the policy relevance of the finding.

Title: Descriptive Clusters Treated as Real Economic Regimes
- Failure Idea: The analysis claims that unsupervised k-means clusters correspond to distinct economic cycles. While the clusters capture patterns in the data, they are driven by algorithmic similarity, not by verified macroeconomic phases. Treating them as real regimes confuses mathematical segmentation with empirical evidence of cyclical structure.

Title: Overgeneralization Across Markets
- Failure Idea: The analyst assumes that patterns identified in U.S. firms apply equally to emerging markets. This neglects country-specific differences in financial reporting, investor behavior, and market maturity, leading to an illusion of universality where context-specific modeling would be required.

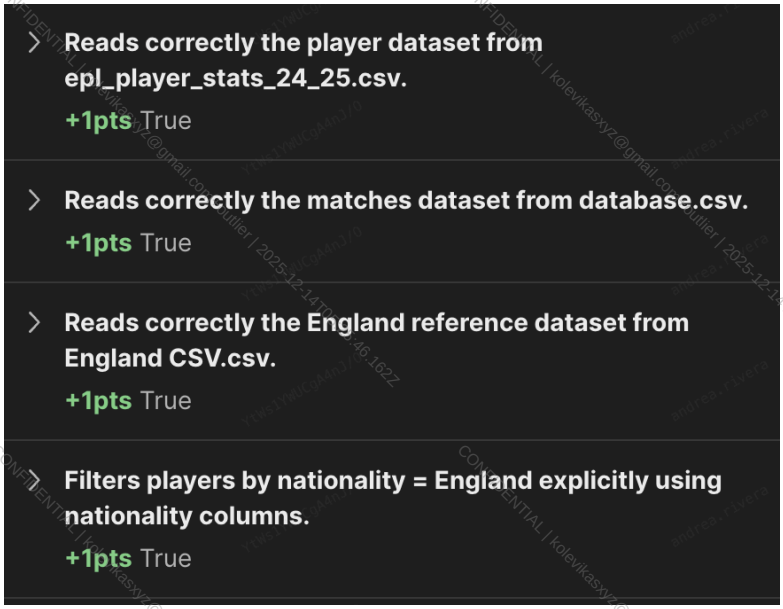
Goal

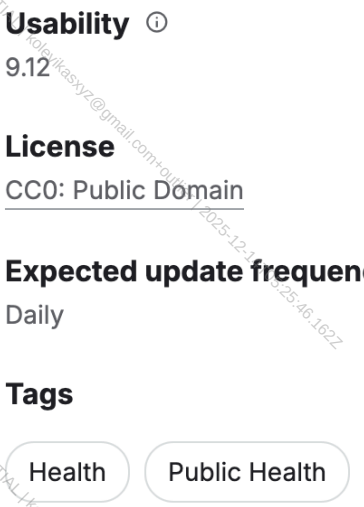
```
The goal is to create a repository of Failure Ideas – rich,
narrative examples of plausible but incorrect analytical
reasoning that can be used to test, red-team, or train models for
reasoning robustness.

Context Provided
Prompt: <Insert your prompt>
Rubrics: <Insert your rubrics>
Optional datasets: <Insert your datasets>
```

Common Errors to Avoid in Rubrics

CRITICAL 🔴- Having this issue will automatically lower your task score to 2
IMPORTANT 🟡- Having this issue will lower your task score with the risk of getting a 2

Status	Common Error	Example
CRITICAL 🔴	Incorrect wording Explanation: When writing a rubric all criteria must be written in the simple present tense (e.g., States that…, includes a…, provides a…, mentions that…, etc.)	Example. ✅ “Calculates a p-value of p=0.0001 for the correlation.” ❌ “The response must calculate the p-value of p=0.0001 for the correlation.”
CRITICAL 🔴	Including process criteria Explanation: The customer is not interested in including	Example of process criteria that SHOULD NOT be included

Status	Common Error	Example
	process criteria within the rubric, please avoid this kind of criteria.	
CRITICAL 🔴	Criteria is not specific enough Explanation: When writing a rubric, we need to create criteria that's more specific and detailed.	Example Criteria: ❌ Calculates all the statistical measures over every sector. ✅Calculates the mean Earnings for healthcare technology to be $194.9M
CRITICAL 🔴	Combined criteria Explanation: Each criterion must address a specific part of the prompt, please avoid to address different dimensions into the same criterion	Example Criteria: ❌ Provides top 3 and bottom 3 of each sector for every metric Solution: For each sector there has to be a separate rubric item outlining the top 3 and a separate rubric item outlining bottom 3 ✅States that top 3 in Divident/Yield category are Drug retail, commodity chemicals and tobacco. ✅States that the bottom 3 in Dividend/Yield category are the Food industry, the tourism sector, and the automotive industry.
CRITICAL 🔴	Datasets that are not public source Explanation: Remember that the datasets must be Public Domain. Make sure to look for the	✅Example:

Status	Common Error	Example
	“License” section to verify this. If your task contains datasets that are not public domain, your task will automatically fail.	
CRITICAL 🔴	Missing essential criteria Explanation: When writing a rubric, all essential criteria to answer the prompt should be included. Missing essential criteria will automatically make the task obtain a low quality score.	Prompt: Give me 2 different Python scripts I can use to convert CSV to JSON ⛔Bad Rubric (missing to include a criterion that evaluates whether the model includes 2 Python scripts): - The Python converts a CSV document into a JSON. - Explains that the script needs to include csv.DictReader() to convert the first row to keys. - Explains that each row needs to be converted into a dictionary. - Explains that all rows are stored in a list and then dumped to JSON using json.dump(). - Explains that including indent=4 makes the output human-readable. ✅Good Rubric: - The Python converts a CSV document into a JSON. - Explains that the script needs to include csv.DictReader() to convert the first row to keys. - Explains that each row needs to be converted into a dictionary. - Explains that all rows are stored in a list and then dumped to JSON using json.dump(). - Explains that including indent=4 makes the output human-readable. - Provides 2 different Python scripts

Status	Common Error	Example
CRITICAL 🔴	Repetitive criteria Explanation: When writing a rubric, each criterion should address a part to evaluate; therefore, repeating criteria is not allowed, as it will lead to over-penalization of the model.	Prompt: Give me 2 different Python scripts I can use to convert CSV to JSON ⛔Bad Rubric (missing to include a criterion that evaluates whether the model includes 2 Python scripts): - The Python converts a CSV document into a JSON. - Explains that the script needs to include csv.DictReader() to convert the first row to keys. - Explains that each row needs to be converted into a dictionary. - Explains that all rows are stored in a list and then dumped to JSON using json.dump(). - Explains that including indent=4 makes the output human-readable. - Provides 2 different Python scripts - Explains that the script needs to include csv.DictReader to convert the first row to keys Criteria 2 and 7 are almost identical, and we must only leave one.
CRITICAL 🔴	Not Self-contained (Closed questions) Explanation: When you have a prompt with a closed question, the criteria should include the answer.	Prompt: What's the capital of France? ⛔Bad Criterion: Includes the capital of France. ✅Good Criterion: Mentions that the capital of France is Paris . --- Prompt: I'm building a VR experience in Unreal Engine using C++ and Blueprints. I want to trigger dynamic haptics and spatial audio when the player interacts with certain world objects. I know Unreal supports force feedback and audio attenuation, but need C++-level details. Please search Unreal Engine's C++ API or headers for: ● Classes/methods for controller vibration. ● Components/properties for spatial sound. Provide: ● Full class names and method signatures. ● Their modules or header files. ● A C++ snippet showing how to trigger both haptics and spatial audio on trigger volume overlap. ● Any required modules/dependencies for compilation. ⛔Bad Criterion:

Status	Common Error	Example
		Provides the names of the C++ components and properties involved in spatial sound propagation. ✅Good Criterion: Provides the names of the C++ components and properties involved in spatial sound propagation: UAudioComponent, USoundAttenuation, USoundCue, USoundBase.
CRITICAL 🔴	Missing examples (open-ended questions) Explanation: Rubric criteria should be clear and objective, with specific examples that show what is expected.	Prompt: Recommend some shoes for running long distances like half marathon. ⛔Bad Criterion: Includes running shoe models for running long distances. ✅Good Criterion: Includes running shoe models for running long distances such as the Nike Alphafly 3, Saucony Endorphin Pro 4, Hoka Cielo X1, etc. --- Prompt: I'm building a VR experience in Unreal Engine using C++ and Blueprints. I want to trigger dynamic haptics and spatial audio when the player interacts with certain world objects. I know Unreal supports force feedback and audio attenuation, but need C++-level details. Please search Unreal Engine's C++ API or headers for: Classes/methods for controller vibration. Components/properties for spatial sound. Provide: Full class names and method signatures. Their modules or header files. A C++ snippet showing how to trigger both haptics and spatial audio on trigger volume overlap. Any required modules/dependencies for compilation. ⛔ Bad Criterion: Explains any required dependencies or engine modules needed in the C++ build file for successful compilation.

Status	Common Error	Example
		✅ Good Criterion: Explains any required dependencies or engine modules needed in the C++ build file for successful compilation, such as Core, CoreUObject, Engine, InputCore, etc.
IMPORTANT 🟡	Subjective criteria Explanation: Criteria should be objective (True/False). Avoid criteria based on personal opinions.	⛔Bad Criterion: States that the best coding language is C++. ✅Good Criterion: States that C++ is the best option to design a financial system because of its ability to handle large datasets and complex calculations efficiently.
IMPORTANT 🟡	Overly prescriptive Explanation: Don't include criteria based on your own likes or dislikes. Focus on what's essential for a good response and what is stated in the prompt.	Prompt: Give me some ideas of Python scripts I can use to convert CSV to JSON ⛔Bad Rubric: - The Python converts a CSV document into a JSON. - Explains that the script needs to include csv.DictReader() to convert the first row to keys. - Explains that each row needs to be converted into a dictionary. - Explains that all rows are stored in a list and then dumped to JSON using json.dump(). - Explains that including indent=4 makes the output human-readable. - Provides exactly 3 different Python scripts.
IMPORTANT 🟡	Unnecessary style/formatting Explanation: Don't specify style or formatting unless the prompt requires it.	Prompt: Give me 2 different Python scripts I can use to convert CSV to JSON ⛔Bad Rubric: - The Python converts a CSV document into a JSON. - Explains that the script needs to include csv.DictReader() to convert the first row to keys. - Explains that each row needs to be converted into a dictionary. - Explains that all rows are stored in a list and then dumped to JSON using json.dump(). - Explains that including indent=4 makes the output human-readable. - Provides 2 different Python scripts - Is formatted in a table for easier reading.

Status	Common Error	Example
		Explanation of why these are bad criteria: The prompt is not requesting any specific formatting.
IMPORTANT 🟡	Poor Grammar and Spelling in the Criteria Explanation: The criteria in the rubric should be written with perfect spelling. Starting each criterion with a capital letter, adding accents, and punctuation marks, and correcting misspelled words.	⛔Bad grammar: the answer must inclde the 4 suits of a poker deck: hearts, spades, diamonds and clubs. ✅Correction: Includes the four suits of a poker deck: hearts, spades, diamonds, and clubs.

How task limit works on this project

Task Progression & Quality Expectations 🚀

1. ---

   ### ⚠️ Quality First!

   This project prioritizes **quality above all**. The team is here to train and guide you, but maintaining strong standards is essential.
   - You have a maximum of **2 quality strikes**.
   - Accumulating **more than 2 strikes** will result in **loss of project permissions**.

2. ---

   ### ✅ Tips for Success

   To keep your quality high, we **strongly encourage you** to:
   - Join **office hours** for live support.
   - Attend **webinars** to sharpen your skills.
   - Participate in **Party Rooms** for collaborative learning.

⚠️ **NOTE**: We have **zero tolerance for cheating**. You are **not allowed** to use any LLMs to generate prompts, rubrics, or quality control notes.

Anyone found violating this rule will be **immediately removed from the project** with **no chance of returning**.

Back to the top ⬆️