Advanced Prompt Engineering for Data Science Projects

First Things First: What Makes a Good Prompt?

Sara Nobrega • 12 min read • 2025-08-20

https://towardsdatascience.com/advanced-prompt-engineering-for-data-science-projects/

Part 2: Prompt Engineering for Features, Modeling, and Evaluation

Image generated with DALL-E.

As a **data scientist**, you have probably wondered several times how to improve your workflows, how to speed up tasks, and how to output better results.

The dawn of **LLMs** has helped numerous data scientists and ML engineers to not only improve their models but also to help them **iterate** faster, learn and focus on the tasks that really matter.

In this article, I am sharing with you my favorite prompts and prompt engineering **tips** that help me tackle Data Science and AI tasks.

Besides, soon **Prompt Engineering** will be a required skill in almost all DS and ML job descriptions.

This guide walks you through practical, research-backed prompt techniques that speed up (and sometimes automate) every stage of your ML workflow.

This is the **second** of a series of **3 articles** I am writing about Prompt Engineering for Data Science:

• **Part 1:** Prompt Engineering for Planning, Cleaning, and EDA

• **Part 2: Prompt Engineering for Features, Modeling, and Evaluation (this article)**

• **Part 3:** Prompt Engineering for Docs, DevOps, and Learning

👉 All the **prompts** in this article are available at the **end** of this article as a cheat sheet 😉

In this article:

1. First Things First: What Makes a Good Prompt?

2. Prompt Engineering for Features, Modeling, and Evaluation

3. Prompt Engineering cheat sheet

You might know this by now but it's always good to refresh our minds about this. Let's break it down.

Anatomy of a High-Quality Prompt

Role & Task

Start by telling the LLM who it is and what it needs to do. E.g.:

```
"You are a senior data scientist with experience in feature engineering, data cleaning and model deployment".)
```

Context & Constraints

This part is really important. Add details and context as much as you can.

Pro Tip: Add all details + context in the same prompt. It is proven that it works best like this.

This includes: data type and format, data source and origin, sample schema, output format, level

of detail, structure, tone and style, token limits, calculation rules, domain knowledge, etc.

Examples or Tests

Give it a few examples to follow, or even unit tests to check the output.

Example - Formatting style for a summary

```
**Input:**
Transaction: { "amount": 50.5, "currency": "USD", "type": "credit", "date":
"2025-07-01" }

**Desired Output:**
- Date: 1 July 2025
- Amount: $50.50
- Type: Credit
```

Evaluation Hook

Ask it to rate its own response, explain its reasoning, or output a confidence score.

Other Prompting Tips

Clean delimiters (##) make sections scannable. Use them all the time!

Put your instructions **before** the data, and wrap context in clear delimiters like triple backticks.

Eg: `## These are my instructions`

Be as much specific as you can. Say “return a Python list” or “only output valid SQL.”

Keep the **temperature** low (0 .3) for tasks that need consistent output, but you can increase it for creative tasks like feature brainstorming.

If you are on a **budget**, use cheaper models for quick ideas, then switch to a premium one to polish the final version.

1. Text Features

With the right prompt, an LLM can instantly generate a diverse set of semantic, rule-based, or linguistic features, complete with practical examples you can, after reviewing, plug into your workflow.

Template: Univariate Text Feature Brainstorm

```
## Instructions
Role: You are a feature-engineering assistant.
Task: Propose 10 candidate features to predict {target}.

## Context
Text source: """{doc_snippet}"""
Constraints: Use only pandas & scikit-learn. Avoid duplicates.

## Output
```

```
Markdown table: [FeatureName | FeatureType | PythonSnippet | NoveltyScore(0–1)]

## Self-check
 R a t e   y o u r   c o n f i d e n c e   i n   c o v e r a g e   ( 0 – 1 )   a n d   e x
```

Pro Tips:

- Pair this with embeddings to create dense features.
- Validate the outputted Python snippets in a sandboxed environment before using them (so you catch syntax errors or data types that do not match).

2. Tabular Features

Manual feature engineering is usually not fun. Especially for tabular data, this process can take some days and it is usually very subjective.

Tools like **LLM-FE** take a different approach. They treat LLMs as **evolutionary optimizers** that iteratively invent and refine features until the performance gets better.

Developed by researchers at Virginia Tech, LLM-FE works in loops:

1. The LLM proposes a new transformation based on the existing dataset schema.

2. The candidate feature is tested using a simple downstream model.

3. The most promising features are kept, refined, or combined (just like in genetic algorithms, but powered by natural language prompts).

This method has showed to perform really well compared to manual feature engineering.

Architecture of the LLM-FE framework, where a large language model acts as an evolutionary optimizer. Source: nikhilsab/LLMFE: This is the official repo for the paper "LLM-FE"

Prompt (LLM-FE style):

```
## Instructions
Role: Evolutionary feature engineer.
Task: Suggest ONE new feature from schema {schema}.
Fitness goal: Max mutual information with {target}.

## Output
JSON: { "feature_name": "...", "python_expression": "...", "reasoning": "...
 ("d 4 0   w o r d s ) "   }

## Self-check
Rate novelty & expected impact on target correlation (0–1).
```

3. Time-Series Features

If you've ever struggle with seasonal trends or sudden spikes in your time-series data, you know it can be hard to deal with all the moving pieces.

TEMPO is a project that lets you prompt for decomposition and forecasting in one smooth step,

so it can save you hours of manual work.

Seasonality-Aware Prompt:

```
## Instructions
System: You are a temporal data scientist.
Task: Decompose time series {y_t} into components.

## Output
Dict with keys: ["trend", "seasonal", "residual"]

## Extra
 E x p l a i n  d e t e c t e d  c h a n g e - p o i n t s  i n "d 6 0  w o r d s .
 S e l f - c h e c k :  C o n f i r m  d e c o m p o s i t i o n  s u m s "H  y _ t
```

4. Text Embedding Features

The idea of the next prompt is pretty straightforward: I'm taking documents and pulling out the key insights that would actually be useful for someone trying to understand what they're dealing with.

```
## Instructions
Role: NLP feature engineer
Task: For each doc, return sentiment_score, top3_keywords, reading_level.

## Constraints
 -  s e n t i m e n t _ s c o r e  i n  [ - 1 , 1 ]  ( n e g!' p o s )
- top3_keywords: lowercase, no stopwords/punctuation, ranked by tf-idf
(fallback: frequency)
- reading_level: Flesch–Kincaid Grade (number)

## Output
CSV with header: doc_id,sentiment_score,top3_keywords,reading_level

## Input
docs = [{ "doc_id": "...", "text": "..." }, ...]

## Self-check
- Header present (Y/N)
- Row count == len(docs) (Y/N)
```

Instead of just giving you a basic "positive/negative" classification, I'm using a continuous score between -1 and 1, which gives you way more nuance.

For the keyword extraction, I went with **TF-IDF** ranking because it actually works really well at surfacing the terms that matter most in each document.

Code Generation & AutoML

Choosing the right model, building the pipeline, and tuning the parameters-it's the holy trinity of machine learning, but also the part that can eat up days of work.

LLMs are game-changers for this stuff. Instead of me sitting there comparing dozens of models or hand-coding yet another preprocessing pipeline, I can just describe what I'm trying to do and get solid recommendations back.

Model Selection Prompt Template:

```
## Instructions
System: You are a senior ML engineer.
Task: Analyze preview data + metric = {metric}.

## Steps
1. Rank top 5 candidate models.
2. Write scikit-learn Pipeline for the best one.
3. Propose 3 hyperparameter grids.

## Output
Markdown with sections: [Ranking], [Code], [Grids]

## Self-check
 J u s t i f y   t o p   m o d e l   c h o i c e   i n "d 3 0   w o r d s .
```

You don't have to stop at ranking and pipelines, though.

You can also tweak this prompt to include model **explainability** from the beginning. This means asking the LLM to justify **why** it ranked models in a certain order or to output feature importance (SHAP values) after training.

That way, you're not just getting a black-box recommendation, you're getting a clear reasoning behind it.

Bonus Bit (Azure ML Edition)

If you're using Azure Machine Learning, this will be useful to you.

With **AutoMLStep**, you wrap an entire automated machine learning experiment (model selection, tuning, evaluation) into a modular step inside an Azure ML pipeline. You will have access to version control, scheduling, and easy repeat runs.

You can also make use of **Prompt Flow**: it adds a visual, node-based layer to this. Features include d r a g a n d d U r, d l o w diagrams, prompt testing, branching logic, and live evaluation:

Example of a simple pipeline in Azure AI Foundry's Prompt Flow editor, where different tools-like the LLM tool and Python tool-are linked together. Source: Prompt flow in Azure AI Foundry portal - Azure AI Foundry | Microsoft Learn

You can also just plug **Prompt Flow** into whatever you've already got running, and then your LLM and AutoML pieces all work together without any problem. Everything just flows in one automated setup that you can actually ship.

Prompts for Fine-Tuning

Fine-tuning a large model doesn't always mean retraining it from scratch (who has time for that?).

Instead, you can use lightweight techniques like **LoRA** (Low-Rank Adaptation) and **PEFT** (Parameter-Efficient Fine-Tuning).

LoRA

So LoRa is actually pretty clever, as instead of retraining a massive model from scratch, it basically just adds tiny trainable layers on top of what's already there. Most of the original model stays frozen, and you're only tweaking these small weight matrices to get it to do what you want.

PEFT

PEFT is basically the umbrella term for all these smart approaches (LoRA being one of them) where you're only training a small slice of the model's parameters instead of the whole massive thing.

The compute savings are incredible. What used to take forever and cost a fortune now runs way faster and cheaper because you're barely touching most of the model.

The best thing about all of this: you don't even have to write these fine-tuning **scripts** yourself anymore. **LLMs** can actually generate the code for you, and they get better at it over time by learning from how well your models perform.

Fine-Tuning Dialogue Prompt

```
## Instructions
Role: AutoTunerGPT.
 S i g n a t u r e :   b a s e _ m o d e l ,   t a s k _ d a t a s e t  !’   t u n e d _ m
Goal: Fine-tune {base_model} on {task_dataset} using PEFT-LoRA.

## Constraints
 -   b a t c h _ s i z e  "d   1 6 ,   e p o c h s  "d   5
- Save to ./lora-model
- Use F1 on validation; set seed=42; enable early stopping (no val gain 2
epochs)

## Output
JSON:
{
  "tuned_model_path": "./lora-model",
  "train_args": { "batch_size": ..., "epochs": ..., "learning_rate": ...,
"lora_r": ..., "lora_alpha": ..., "lora_dropout": ... },
  "val_metrics": { "f1_before": ..., "f1_after": ... },
  "expected_f1_gain": ...
}

## Self-check
- Verify constraints respected (Y/N).
 -   I f   N ,   e x p l a i n   i n  "d 2 0   w o r d s .
```

Tool tip: Use **DSPy** to improve this process. DSPy is an open-source framework for building **self-improving** pipelines. This means it can automatically rewrite prompts, enforce constraints (like batch size or training epochs), and track every change in multiple runs.

In practice, you can run a fine-tuning job today, review the results tomorrow, and have the system **auto-adjust** your prompt and training settings for a better result without you having to start from scratch!

Let LLMs Evaluate Your Models

Smarter Evaluation Prompts
Studies show that LLMs score predictions almost like humans, when guided by good prompts.

Here are 3 prompts that will help you boost your evaluation process:

Single-Example Evaluation Prompt

```
## Instructions
System: Evaluation assistant.
User: Ground truth = {truth}; Prediction = {pred}.

## Criteria
 -  f a c t u a l _ a c c u r a c y "  [ 0 , 1 ] :  1  i f  s e m a n t i c a l l y
contradictory; partial if missing/extra but not wrong.
 -  c o m p l e t e n e s s "  [ 0 , 1 ] :  f r a c t i o n  o f  r e q u i r e d

## Output
JSON:
 {  " a c c u r a c y " :  < f l o a t > ,  " c o m p l e t e n e s s " :  < f l o a t

## Self-check
 C i t e  w h i c h  f a c t s  w e r e  m a t c h e d / m i s s e d  i n "d 1 5  w
```

Cross-Validation Code

```
## Instructions
You are CodeGenGPT.

## Task
Write Python to:
- Load train.csv
- Stratified 80/20 split
- Train LightGBM on {feature_list}
- Compute & log ROC-AUC (validation)

## Constraints
- Assume label column: "target"
- Use sklearn for split/metric, lightgbm.LGBMClassifier
- random_state=42, test_size=0.2
- Return ONLY a Python code block (no prose)

## Output
(only code block)
```

Regression Judge

```
## Instructions
System: Regression evaluator
Input: Truth={y_true}; Prediction={y_pred}

## Rules
abs_error = mean absolute error over all points
Let R = max(y_true) - min(y_true)
Category:
 -   " E x c e l l e n t "   i f   a b s _ e r r o r  "d   0 . 0 5   *   R
 -   " A c c e p t a b l e "   i f   0 . 0 5   *   R   <   a b s _ e r r o r  "d   0 . 1 5
- "Poor" if abs_error > 0.15 * R

## Output
{ "abs_error": <float>, "category": "Excellent/Acceptable/Poor" }

## Self-check (brief)
Validate len(y_true)==len(y_pred) (Y/N)
```

Troubleshooting Guide: Prompt Edition

If you ever find one of these 3 problems, here is how you can fix it:

Wrapping It Up

Since LLMs became trendy, prompt engineering has officially leveled up. Now, it is a true and serious methodology that touches every part of ML and DS workflows. That's why a big part of AI research is focused on how to improve and optimize prompts.

At the end, better **prompt engineering** means better **outputs** and a lot of **time** saved. Which I suppose is the **dream** of any data scientist 😉

Thank you for reading!

👉 Grab the **Prompt Engineering Cheat Sheet** with all prompts of this article organized. I will send it to you when you subscribe to **Sara's AI Automation Digest**. You'll also get access to an AI tool library and my free AI automation newsletter every week!

Thank you for reading! 😉

I offer **mentorship** on career growth and transition here.

If you want to support my work, you can buy me my **favorite coffee**: a cappuccino. 😊

References

What is LoRA (Low-Rank Adaption)? | IBM

A Guide to Using ChatGPT For Data Science Projects | DataCamp

---

Written By

Share This Article

- Share on Facebook
- Share on LinkedIn
- Share on X

Towards Data Science is a community publication. Submit your insights to reach our global audience and earn through the TDS Author Payment Program.

Other mentions by Author

- towardsdatascience.com | Related Articles
- towardsdatascience.com | Deep Dive into LlaMA 3 by Hand ✍️
- towardsdatascience.com | Deep Dive into Transformers by Hand ✍️
- towardsdatascience.com | Deep Dive into Sora's Diffusion Transformer (DiT) by Hand ✍️
- towardsdatascience.com | UniFliXsg: AI-Powered Undergraduate Program Recommendations for Singapore Universities
- towardsdatascience.com | Beware of Unreliable Data in Model Evaluation: A LLM Prompt Selection case study with Flan-T5
- towardsdatascience.com | Beating ChatGPT 4 in Chess with a Hybrid AI model