Plexe FDE: Take Home Assignment

Take-Home Assignment: ML for a Marketplace

Calendar time: 5 days from receipt **Expected effort:** 4–6 hours **Debrief:** After submission, we'll schedule a 45-minute call to walk through your work. Come prepared to explain your decisions, demo the API, and discuss what you'd do differently with more time.

Context

At Plexe, we help businesses go from a vague idea to a production-ready ML model. As a Forward Deployed Engineer, you'll sit with customers who know they have a problem but often can't articulate what model they need. Your job is to listen, explore their data, figure out what's worth building, and deliver something that actually moves a business metric.

This assignment simulates exactly that.

The Customer

You're working with the operations team at a mid-size online marketplace, a platform connecting independent sellers to buyers across a large, geographically diverse country. They've given you access to their data and one conversation:

"We're growing fast but our margins are getting squeezed. Sellers are complaining, buyers leave bad reviews, and we don't really know where to focus. We've heard ML can help but we don't know what to build first. Can you look at our data and tell us what would actually make a difference?"

That's it. They haven't told you what to predict. **You need to figure that out.**

The Dataset

Use the **Brazilian E-Commerce Public Dataset by Olist**, available on Kaggle:

👉 **https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce**

This is real, anonymized data from a Brazilian marketplace. It contains ~100K orders across 9 linked tables:

- Orders, order items, payments
- Customer and seller information
- Product details and categories
- Customer reviews (1–5 star ratings + free text)
- Geolocation data (zip code → lat/lng)

What We Want You To Do

1. Identify 1–2 ML problems worth solving

Explore the data and decide what model(s) would create the most value for this marketplace.

We don't want you to build everything. We want you to make a compelling case for *why* the problem(s) you chose matter most, and *why you ruled out* the alternatives. Show us the data exploration that led to your decision. This is the most important part of the exercise.

2. Build and train the model

- Use proper train/validation/test methodology. Explain your splitting strategy.
- If you tried multiple approaches (different algorithms, feature strategies), show us what you learned from comparing them. But one well-built model with strong evaluation beats two rushed ones.
- Track your experiments.

3. Evaluate it properly

This is where we'll spend the most time in the review. We want to see:

- **Metric selection:** Justify which metrics matter for your specific problem and why.
- **Error analysis:** Look at the cases the model gets wrong. Are there patterns? What segments does the model struggle with?
- **Practical limitations:** Be honest about when to trust and when not to trust this model.

4. Deploy it

- Serve the model behind a **REST API** (FastAPI, Flask — your choice).
- The API should accept a realistic input payload and return predictions plus a brief explanation of top contributing features (e.g., SHAP values).
- Include a sample `curl` command or test script.

5. AI Tool Usage Log (Mandatory)

Submit the **full chat/prompt history** from whatever AI tools you used (Cursor, Claude, ChatGPT, Copilot, etc.). Export or screenshot the conversations.

We **expect** you to use AI. We're not penalizing it but will evaluate:

- How you **decompose a vague problem** into concrete prompts
- Whether you **validate and correct** AI output or blindly paste it
- Where you **override** the AI's suggestions with your own judgment
- Your sense of **when AI helps vs. when it doesn't**

If you genuinely didn't use AI tools, state that and explain your workflow.

Submission

A **GitHub repository** containing:

```
None
├── README.md                    # Setup instructions
├── notebooks/                   # EDA, experimentation
├── src/                         # Clean production-style code
│   ├── train.py
│   ├── evaluate.py
│   ├── serve.py
│   └── ...
├── Dockerfile
├── ai_chat_logs/                # Your AI tool conversation
exports
└── experiments/                 # Tracked experiment results
```

A Few Notes

- **Don't over-engineer.** Sharp thinking on one well-chosen problem beats a mediocre attempt at five things. If you're running over 6 hours, stop and ship what you have.
- **AutoML is fine** (including Plexe), but you still need to demonstrate you understand what's happening. If you use AutoML for training, invest more in evaluation.
- **Production-level intent, not perfection.** Clean structure, error handling, a working Dockerfile. Not a fully CI/CD'd microservice.

We're looking for how you think, not just what you build. Good luck.