Senior Data Engineer Task: Credit Risk Prediction

Hi! We're really happy you're here. This task is your chance to show how you think, build, and ship. We love code that's clean, easy to follow, and gets the job done.

Take your time, but don't overthink it. The whole exercise should take around 3-4 hours.

Scenario

You've just joined the data team at Atto, a fast-growing FinTech company that builds tools and products enabling businesses to provide lending products to consumers. Our platform makes it easier for lenders to make faster, fairer, and more accurate credit decisions.

One of our key clients, a major consumer lender, has been struggling with rising default rates - hitting 12% last quarter. They've been relying on basic rules and spreadsheets to assess which customers to approve. They've asked us to help them move toward a data-driven credit decisioning system - one that can process transaction history and predict the likelihood of a customer defaulting within 90 days.

Your manager has handed you a prototype dataset and a basic model that the data science team trained last week. Your objective is threefold:

1. Clean and prepare the raw transaction data so it's model-ready
2. Build a simple API that can serve predictions to the risk platform
3. Document your approach and think through how this would work in production

The team is eager to test this in a live environment soon, so they need something simple, reliable, and production-ready.

Overview

As described in the scenario above, this task has three parts:

1. Data Engineering: Turn some raw CSVs into something you could actually model.
2. API Development: Take an existing model and expose it behind a simple FastAPI endpoint.
3. Documentation: Tell us about your approach; what was tricky, what decisions you made, and how you'd scale or deploy it in the real world.

Part 1: Data Engineering

You'll find a couple of CSVs in the `data/` folder:

- `transactions.csv` - sample customer transactions
- `labels.csv` - a small table of whether each customer defaulted within 90 days

Sample data format:

```
# transactions.csv
customer_id,transaction_id,amount,type,description,timestamp
CUST001,TXN001,-45.50,debit,TESCO STORE 123,2024-01-15 10:23:00
CUST001,TXN002,2500.00,credit,SALARY PAYMENT,2024-01-20 09:00:00
```

```
# labels.csv
customer_id,defaulted
CUST001,0
CUST002,1
```

Requirements

1. Load and explore the data
   - Load both files
   - Document any data quality issues you find (nulls, duplicates, outliers)
   - Feel free to make reasonable assumptions, but note them in your code
2. Feature engineering
   - Aggregate transactions so there's one row per customer
   - Create these features as a minimum:
     - Number of transactions
     - Total debit and total credit
     - Average transaction amount
   - Add 1-3 features of your own design
3. Text processing on the `description` column
   - Clean the text: lowercase, remove punctuation and numbers
   - Extract meaningful signals by counting or flagging common merchant categories
   - Examples: "rent", "tesco", "netflix", "salary", "transfer", etc.
4. Create the training dataset
   - Merge everything with the labels to form a training-ready dataset
   - Format: one row per customer, all features + target column
   - Save your result as `artifacts/training_set.csv`
5. Exploratory Data Analysis (optional)
   - Include 2-3 simple visualizations showing feature distributions
   - This helps us understand your exploratory thinking

Expected Output

Your `training_set.csv` should look something like this:

```
customer_id,num_transactions,total_debit,total_credit,avg_amount,has_rent,has_salary,...,defaulted
CUST001,15,-1250.50,3500.00,150.03,1,1,...,0
```

Part 2: API Development

We've already trained a simple model for you. It's waiting at `artifacts/model.joblib`. Your job is to build a small FastAPI service that can load that model and return predictions.

Requirements

Create a `POST /predict` endpoint that accepts a JSON payload with the same features from Part 1 and returns predictions.

Input schema example

```
{
  "customer_id": "CUST001",
  "num_transactions": 15,
  "total_debit": -1250.50,
  "total_credit": 3500.00,
  "avg_amount": 150.03,
  "has_rent": 1,
  "has_salary": 1
}
```

Expected Output

```
{
  "customer_id": "CUST001",
  "probability": 0.81,
  "prediction": 1
}
```

Part 3: Documentation

Drop a short file called `README.md` in the root of the repo.

A note on AI tools: We know it's 2025 and everyone uses AI. You're welcome to use Copilot, Claude, Gemini, or similar tools as part of your workflow - just be transparent about where and how you used them.

Answer these questions (and feel free to add anything else that helps explain your approach):

1. What part of the exercise did you find most challenging, and why?
2. What tradeoffs did you make? (e.g., speed vs. accuracy, simplicity vs. completeness)
3. Assume this needs to run in production with these constraints:
   - Cloud provider: Azure
   - Budget: £500/month
   - Latency requirement: <100ms per prediction
   - Expected traffic: 1000 predictions/hour initially

   What would you improve or change first?
4. How would you deploy the FastAPI service and make the model artifact available?
5. If transaction volume jumped from thousands to millions per day, how would you rethink Part 1?
6. What metrics would you track in production and why? What could go wrong with this model in production?
7. If you used AI tools such as ChatGPT, Claude, Copilot, or any other tools:
   - Where and how did you use them? (e.g., boilerplate code, debugging, syntax help)

- How did they help or hinder your process?

Submission

Once you're done:

- Push your code to a Git repo (Github / GitLab) and send us the link (or share a zip)
- Include your `README.md` file in the root
- Make sure it runs from a fresh clone (`pip install -r requirements.txt`)
- Include any setup instructions in a brief section at the top of your `README.md`

Good luck!