🎯Project Goal

Build a **real-time fraud detection system** that flags high-risk transactions with **high recall at low false positives**, deploy it as a **FastAPI** service, and monitor performance drift.

Deliverables (What you'll ship)

1. **EDA & Modeling Notebook** (`fraud_detection.ipynb`) with insights and model comparisons.
2. **Trained model artifacts** (`model.pkl`, `encoder.pkl`, `threshold.json`).
3. **Realtime scoring API** (FastAPI: `serve.py`) with `/predict` endpoint.
4. **Batch scoring job** (`batch_score.py`) for daily backfills.
5. **Model Card** (`MODEL_CARD.md`) describing data, metrics, risks, ethics.
6. **Monitoring dashboard stub** (script pushing metrics to csv/SQLite + Grafana-ready).
7. **README** with run commands + decisions.

Datasets (pick one to start)

- **Credit Card Fraud (European card transactions, highly imbalanced)**: 284,807 rows, 492 frauds (~0.172%). *Great for core classification + imbalance handling.*
- **IEEE-CIS Fraud Detection (online payments; wider feature variety)**: bigger, messier; great stretch goal.

  Start with the first dataset for a 1-week MVP, then extend ideas to IEEE-CIS.

🎿Success Metrics

- **Primary**: Recall@Precision≥0.90 (catch ≥X% fraud while keeping precise alerts)
- **Secondary**: PR-AUC (preferred for imbalance), ROC-AUC, Latency p95 < 50ms per request, **Cost savings** proxy via custom cost matrix
- **Business metric**: Estimated **chargeback avoided** = #TruePositives × avg_loss − #FalsePositives × ops_cost

---

🕹️One-Week Plan (MVP)

Day 1: Data loading, quick EDA, leakage checks, baseline splits
Day 2: Feature engineering (amount z-scores by user/merchant/hour), imbalance strategies
Day 3: Train baselines (LogReg, RandomForest, XGBoost), PR-AUC comparison
Day 4: Threshold tuning by cost; calibration; error analysis
Day 5: Package model; build FastAPI `/predict` + schema validation

Day 6: Batch scorer + monitoring hooks (drift, precision@k)
Day 7: Write MODEL_CARD, README, final metrics; demo script

🎍 Data Schema (typical)

```
transaction_id: str
user_id: str
merchant_id: str
amount: float
currency: str
country: str
city: str
device_id: str
channel: enum [POS, WEB, APP]
merchant_category: str
timestamp: datetime
label: int  # 1=fraud, 0=legit
```

🧩Feature Engineering (core ideas)

- **Behavioral**: amount_vs_user_mean (z-score), txn_hour, day_of_week, txn_velocity (n txns per 1h/24h), last_amount_diff, first_time_device_for_user
- **Geospatial**: distance_from_last_location, country_mismatch_user_profile
- **Merchant**: merchant_risk_score (historical fraud rate), mcc_encoding
- **Device/Network**: device_first_seen_age, ip_risk_bucket
- **Encoding**: WOE/Target encoding for high-cardinality IDs (with **K-fold** to avoid leakage)
- **Scaling**: Standardize continuous features; keep pipeline with `ColumnTransformer`

Imbalance Handling

- Class weights (LogReg/XGB scale_pos_weight)
- Threshold tuning using **expected cost**
- Oversampling (SMOTE) or undersampling on **train only**
- **Focal loss** (if using LightGBM/XGBoost custom obj)

🧨Evaluation Protocol

- Stratified train/valid/test split by **time** (simulate future)
- Metrics: **PR-AUC**, ROC-AUC, Recall@Precision≥0.90, Confusion Matrix
- **Calibration**: Reliability curve / Brier score

- **Error Analysis**: slice by merchant, amount bucket, hour; SHAP for top model

---

Models to Compare (MVP first)

1. Logistic Regression (with class_weight)
2. XGBoost / LightGBM (handles nonlinearity + imbalance)
3. IsolationForest / One-Class SVM (unsupervised baseline)
4. Autoencoder (optional stretch) for reconstruction error
5. Graph approach (stretch): link accounts-devices-merchants to detect rings

---

Project Structure

```
fraud-detection/
├── data/                    # raw & processed (gitignored)
├── notebooks/
│   └── fraud_detection.ipynb
├── src/
│   ├── features.py          # feature builders
│   ├── train.py             # training entrypoint
│   ├── evaluate.py          # metrics & reports
│   ├── infer.py             # local inference helper
│   ├── serve.py             # FastAPI app
│   └── monitor.py           # drift & performance logging
├── models/
│   ├── model.pkl
│   ├── preproc.pkl
│   └── threshold.json
├── MODEL_CARD.md
├── requirements.txt
└── README.md
```

---

requirements.txt (minimal)

```
pandas
numpy
scikit-learn
xgboost
lightgbm
fastapi
uvicorn
```

```
pyyaml
pydantic
shap
joblib
```

🧨Notebook Outline (`notebooks/fraud_detection.ipynb`)

1. Load data, parse dates, basic sanity checks (duplicates, missing, label ratio)
2. **Leakage scan**: columns too predictive? drop/guard
3. EDA: histograms of amount/time, fraud by hour/merchant
4. Train/valid/test split by time
5. Pipeline: `ColumnTransformer` (scaler + target encoding) → model
6. Compare models via **PR-AUC**; plot precision-recall curve
7. Threshold tuning via cost matrix
8. Calibration (Platt/Isotonic)
9. SHAP for top model; slice analysis
10. Save artifacts (model + preprocessors + threshold)

Cost-Aware Threshold Tuning (snippet)

```python
import numpy as np
# Costs (example):
C_TP = +150  # saved chargeback
C_FP = -10   # ops/manual review cost
C_FN = -300  # missed fraud cost
C_TN = 0

# Choose threshold maximizing expected net value
probs = clf.predict_proba(X_val)[:,1]
thresholds = np.linspace(0.01, 0.99, 99)
best_t, best_val = None, -1e9
for t in thresholds:
    preds = (probs >= t).astype(int)
    TP = ((preds==1) & (y_val==1)).sum()
    FP = ((preds==1) & (y_val==0)).sum()
    FN = ((preds==0) & (y_val==1)).sum()
    TN = ((preds==0) & (y_val==0)).sum()
    value = TP*C_TP + FP*C_FP + FN*C_FN + TN*C_TN
    if value > best_val:
        best_t, best_val = t, value
print({'best_threshold': best_t, 'expected_value': best_val})
```

Training Entry (`src/train.py` skeleton)

```
# src/train.py
import joblib, json
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import average_precision_score, roc_auc_score
from xgboost import XGBClassifier

NUM_COLS = ["amount"]
CAT_COLS = ["country","channel","merchant_category"]
TARGET = "label"

# TODO: replace with actual loaders
df = pd.read_csv("data/transactions.csv")
X = df.drop(columns=[TARGET])
y = df[TARGET]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

pre = ColumnTransformer([
    ("scaler", StandardScaler(), NUM_COLS),
    ("onehot",
__import__("sklearn").preprocessing.OneHotEncoder(handle_unknown="ignore"),
CAT_COLS),
])

model = XGBClassifier(
    n_estimators=500,
    max_depth=6,
    learning_rate=0.05,
    subsample=0.8,
    colsample_bytree=0.8,
    eval_metric="logloss",
    scale_pos_weight=10  # tweak via pos/neg ratio
)

pipe = Pipeline([("pre", pre), ("clf", model)])
pipe.fit(X_train, y_train)

probs = pipe.predict_proba(X_test)[:,1]
```

```
print({
  "pr_auc": float(average_precision_score(y_test, probs)),
  "roc_auc": float(roc_auc_score(y_test, probs))
})

joblib.dump(pipe, "models/model.pkl")
json.dump({"threshold": 0.5}, open("models/threshold.json","w"))
```

FastAPI Service (`src/serve.py`)

```
# src/serve.py
from fastapi import FastAPI
from pydantic import BaseModel
import joblib, json
import numpy as np

app = FastAPI(title="Fraud Detection API")
pipe = joblib.load("models/model.pkl")
th = json.load(open("models/threshold.json"))['threshold']

class Txn(BaseModel):
    amount: float
    country: str
    channel: str
    merchant_category: str

@app.post("/predict")
def predict(txn: Txn):
    X = [[txn.amount, txn.country, txn.channel, txn.merchant_category]]
    proba = float(pipe.predict_proba(X)[:,1][0])
    is_fraud = proba >= th
    return {"fraud_probability": proba, "flag": bool(is_fraud)}

# Run: uvicorn src.serve:app --reload
```

Batch Scoring (`src/batch_score.py`)

```
import pandas as pd, joblib, json
from pathlib import Path

pipe = joblib.load("models/model.pkl")
```

```python
th = json.load(open("models/threshold.json"))['threshold']

inp = pd.read_csv("data/new_transactions.csv")
probs = pipe.predict_proba(inp)[:,1]
inp["fraud_probability"] = probs
inp["flag"] = (probs >= th).astype(int)
Path("outputs").mkdir(exist_ok=True)
inp.to_csv("outputs/scored.csv", index=False)
```

📷Monitoring Hooks (`src/monitor.py`)

- Log counts by day: total txns, %flagged, base_rate shift
- Track **PSI** / KL divergence for key features vs training
- Track **precision/recall** weekly using human-labeled feedback
- Alert on drift or precision < target

🪁MODEL_CARD.md (template)

- **Model purpose**: real-time fraud flagging for manual review/blocking
- **Intended users**: risk ops, payment gateway
- **Data**: source, time window, preprocessing
- **Performance**: PR-AUC, Recall@P≥0.9, calibration
- **Ethical/risk**: false positives impact, localization bias, appeal process
- **Monitoring**: drift checks, retrain cadence, rollback plan

---

Guardrails & Best Practices

- Prevent **data leakage** (no post-event info)
- Strict **train/validation by time**
- **PII safety**: hash IDs, minimize exposure; access control
- **Explainability** for ops: reason codes via SHAP top features
- **Human-in-the-loop**: review queue for medium scores

🖌️Next Steps

1. Download dataset & place as `data/transactions.csv`
2. Run `python src/train.py` → produce `models/`
3. Start API: `uvicorn src.serve:app --reload`
4. Score batch: `python src/batch_score.py`
5. Iterate on features + threshold using cost model