# Clase 229 — Capstone 1: problema tabular end-to-end (EDA, modelo, API, dashboard)

Parte: 8 — Capstones · Fuente: integrador de Partes 0-7 + Huyen, C. Designing Machine Learning Systems (O'Reilly, 2022) caps. 4-7 + Géron, A. Hands-On ML, 3ª ed. caps. 2-3.

Duración estimada: 180 min.

# Clase 229 — Capstone 1: problema tabular end-to-end (EDA, modelo, API, dashboard)

> *Parte: 8 — Capstones · Fuente: integrador de Partes 0-7 + Huyen, C. Designing Machine Learning Systems (O'Reilly, 2022) caps. 4-7 + Géron, A. Hands-On ML, 3ª ed. caps. 2-3. Duración estimada: 180 min.*

## Objetivo

Integrar en un único proyecto entregable todo lo aprendido en Partes 0-7: cargar un dataset tabular real, hacer EDA, construir un pipeline ColumnTransformer reproducible, entrenar y tunear un modelo gradient-boosting con MLflow tracking, serializar el modelo, exponerlo vía FastAPI con Pydantic v2, construir un dashboard Streamlit con SHAP, y dejar todo bajo CI con GitHub Actions. La entrega debe poder reproducirse desde un clon limpio con uv sync y un docker compose up.

## Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Diseñar un pipeline ML end-to-end siguiendo el flujo de Huyen (Designing ML Systems): data → features → modelo → tracking → serving → monitoring.
- Tunear un gradient-boosting con Optuna (≥50 trials) y loguear cada run en MLflow con parámetros, métricas y artefactos.
- Exponer el modelo en FastAPI con schemas Pydantic v2 validados y latencia P95 < 200 ms.
- Construir un dashboard Streamlit con explicaciones SHAP por predicción + drift monitor (Evidently).
- Publicar una Model Card (Mitchell et al. 2019) con uso intencionado, métricas por subgrupo y limitaciones conocidas.

## Fases del capstone

| # | Fase | Duración | Entregable |
|---|---|---|---|
| 1 | EDA | 25 min | notebooks/01_eda.ipynb + reporte ydata-pro |
| 2 | Feature engineering | 25 min | src/features.py con ColumnTransformer repr |
| 3 | Modelo + tuning | 40 min | src/train.py con Optuna 50 trials + MLflow |
| 4 | Tracking + Model Card | 20 min | mlruns/ + MODEL_CARD.md |
| 5 | API FastAPI | 35 min | src/api/main.py con /predict, /health, Ope |
| 6 | Dashboard + CI | 35 min | app.py Streamlit con SHAP + .github/workfl |

## Definiciones y características

- Stack 2026 recomendado: uv (lock + venv) + Polars o pandas 2.x + scikit-learn 1.5+ + xgboost/lightgbm + optuna 3.x + mlflow 2.x + fastapi 0.115+ + pydantic 2.x + streamlit 1.x + shap + evidently 0.4+ + docker compose.
- Métricas clasificación: ROC-AUC (ranking), F1 al threshold elegido, log-loss (calibración), Brier score,

PR-AUC si hay desbalance >10:1.
- Métricas regresión: RMSE, MAE, R², MAPE (cuidado con y≈0).
- Threshold selection: el modelo devuelve probabilidad; el threshold se elige por F1, Youden's J, o coste-beneficio. No usar 0.5 por default.
- MLflow run: una ejecución de entrenamiento con params, metrics, tags y artifacts (modelo + plots). Optuna integra vía MLflowCallback.
- FastAPI schema: BaseModel Pydantic v2 con Field(..., ge=0, le=100) para validar rangos; OpenAPI auto-generado en /docs.
- Streamlit dashboard: app reactiva en Python puro; cachear cargas pesadas con @st.cache_resource.
- Model Card (Mitchell 2019): documento corto con uso intencionado, datos de entrenamiento, métricas por subgrupo, consideraciones éticas, limitaciones.
- CI smoke test: GitHub Actions corre pytest + levanta la API en background + hace curl /health → debe responder 200.

## Dataset / recursos

- Dataset sugerido (elegí uno): UCI Adult (clasificación binaria, 48K filas, mix num+cat), Telco Customer Churn (Kaggle, 7K filas, churn binario), o Ames House Prices (Kaggle, 1.5K filas, regresión).
- Librerías: pandas o polars, scikit-learn, xgboost/lightgbm, optuna, mlflow, fastapi, uvicorn, pydantic, streamlit, shap, evidently, joblib, pytest, httpx.
- Infra local: Docker + compose.yml con 3 servicios (mlflow, api, streamlit).

## Ejercicios

1. EDA: cargar dataset, generar ydata-profiling HTML, identificar missing, outliers, balance de clases. Documentar 5 hallazgos en notebooks/01_eda.ipynb.
2. Pipeline FE: armar ColumnTransformer con OneHotEncoder(handle_unknown='ignore') para cat, StandardScaler para num, SimpleImputer para missing. Persistir el preprocessor.
3. Modelo + Optuna: entrenar baseline (LogisticRegression) y challenger (XGBClassifier/LGBMClassifier). Tunear con Optuna 50 trials maximizando ROC-AUC sobre validación. Loguear cada trial en MLflow.
4. API: en src/api/main.py, definir PredictRequest(BaseModel) con todas las features tipadas y PredictResponse(BaseModel) con probability y prediction. Cargar modelo en lifespan. Endpoint /predict + /health.
5. Dashboard: app.py Streamlit con (a) form input → llama a la API y muestra predicción, (b) plot SHAP waterfall de la última predicción, (c) Evidently report comparando producción vs training data.

## Homework verificable

Entregar un repo público en GitHub con:

1. Estructura src/, notebooks/, tests/, MODEL_CARD.md, README.md, pyproject.toml lockeado con uv.
2. MLflow tracking con ≥3 runs registradas (baseline + 2 challengers) y artefactos (modelo, plots de ROC, calibration curve).
3. FastAPI con /predict y /health, schemas Pydantic v2 validados, OpenAPI en /docs, latencia P95 < 200 ms medida con locust o httpx.
4. Dashboard Streamlit con SHAP plot por predicción y screenshot en el README.
5. Model Card completa (uso intencionado, métricas por subgrupo si aplica, limitaciones).

6. CI GitHub Actions corriendo pytest + smoke test contra /health con la API levantada en el runner. Badge verde en el README.

Criterio de aceptación: clonar el repo desde cero → uv sync → docker compose up → API responde a /predict con un request válido → dashboard Streamlit accesible en localhost:8501 → CI verde en GitHub.

## Errores comunes

| Síntoma / mensaje | Causa y cómo arreglar |
|---|---|
| Modelo tunea AUC=0.99 en validación, 0.65 | Data leakage — features computadas con inf |
| ROC-AUC perfecto pero el modelo es inútil | Target leakage — alguna feature es proxy d |
| 422 Unprocessable Entity al llamar /predic | FastAPI sin schema Pydantic o tipos mal de |
| Latencia /predict > 1 segundo | Cargás el modelo en cada request. Fix: car |
| Streamlit re-entrena modelo en cada intera | No cacheaste. Fix: @st.cache_resource para |
| MLflow run sin artefactos, solo métricas | Olvidaste mlflow.sklearn.log_model(model, |

## Preguntas frecuentes

¿XGBoost o LightGBM?

Cualquiera. LightGBM suele ser 2-5× más rápido en entrenamiento sobre datasets >100K filas; XGBoost tiene mejor soporte en producción legacy. Para el capstone: elegí uno y justificalo en el README.

¿Necesito GPU?

No. Datasets tabulares <1M filas entrenan en CPU en minutos con XGBoost/LGBM. Reservá GPU para deep learning (Parte 7).

¿FastAPI o Flask?

FastAPI. Pydantic v2 da validación automática + OpenAPI gratis + async nativo. Flask sigue siendo válido pero pedís el doble de boilerplate.

¿Cómo pruebo la API en CI sin levantar Docker?

pytest + httpx.AsyncClient(app=app) — testea la app FastAPI in-process, sin red. Smoke test real con Docker corre en otro job del workflow.

¿La Model Card es opcional?

No. Es entregable obligatorio. 1-2 páginas Markdown con: propósito, datos, métricas por subgrupo (género/edad/región si aplica), limitaciones conocidas, contacto. Mitchell et al. 2019 tiene el template.

## Referencias

- Huyen, C. Designing Machine Learning Systems (O'Reilly, 2022) — caps. 4 (Training Data), 5 (Feature Engineering), 7 (Model Deployment).
- Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow, 3ª ed. (O'Reilly, 2022) — caps. 2-3, el end-to-end project canónico.
- MLflow Tracking — runs, params, metrics, artifacts.
- FastAPI tutorial — Pydantic v2 + OpenAPI auto.

- Streamlit docs — cache_resource, cache_data, layout.
- Mitchell, M. et al. Model Cards for Model Reporting (FAT* 2019) — el template oficial.

## Siguiente clase

Clase 230 — Capstone 2: NLP o series de tiempo end-to-end

## Apéndice: notebook (primer bloque)

Esqueleto reproducible del capstone: EDA → FE → modelo → tuning → tracking → API stub → dashboard stub → Model Card. Requisitos mínimos: numpy, pandas, scikit-learn, optuna, joblib. mlflow es opcional (capturado con try/except). Seed = 42 en todo el notebook.

```
import numpy as np, pandas as pd
from sklearn.model_selection import train_test_split

SEED = 42
rng = np.random.default_rng(SEED)
n = 10_000

# Dataset sintético tipo churn: 12 features (num + cat + bool)
tenure       = rng.integers(1, 73, n)
monthly_chg  = rng.normal(70, 25, n).clip(15, 200)
total_chg    = monthly_chg * tenure + rng.normal(0, 50, n)
age          = rng.integers(18, 85, n)
n_services   = rng.integers(1, 8, n)
support_call = rng.poisson(1.5, n)
contract     = rng.choice(['month', 'one_year', 'two_year'], n, p=[0.55, 0.25, 0.20])
payment      = rng.choice(['credit', 'debit', 'bank', 'check'], n)
internet     = rng.choice(['fiber', 'dsl', 'none'], n, p=[0.45, 0.40, 0.15])
gender       = rng.choice(['M', 'F'], n)
paperless    = rng.choice([True, False], n)
auto_pay     = rng.choice([True, False], n, p=[0.4, 0.6])

# Inyectar missing (~3% en monthly_chg)
miss_idx = rng.choice(n, size=int(n*0.03), replace=False)
monthly_chg_obs = monthly_chg.copy(); monthly_chg_obs[miss_idx] = np.nan

# Señal del target: contratos mes-a-mes + soporte alto + sin auto-pay → churn
logit = (
   -1.5
   + 1.4 * (contract == 'month')
   - 0.8 * (contract == 'two_year')
# ... (truncado)
```

## Archivos complementarios

- notebook.ipynb