EJERCICIOS PRÁCTICOS - GUÍA MLOps

Prácticas por Módulo con Metodología Explorar → Reproducir → Extender

Guía MLOps v3.0 | DuqueOM | Noviembre 2025

EJERCICIOS PRÁCTICOS

Guía Completa de Ejercicios MLOps

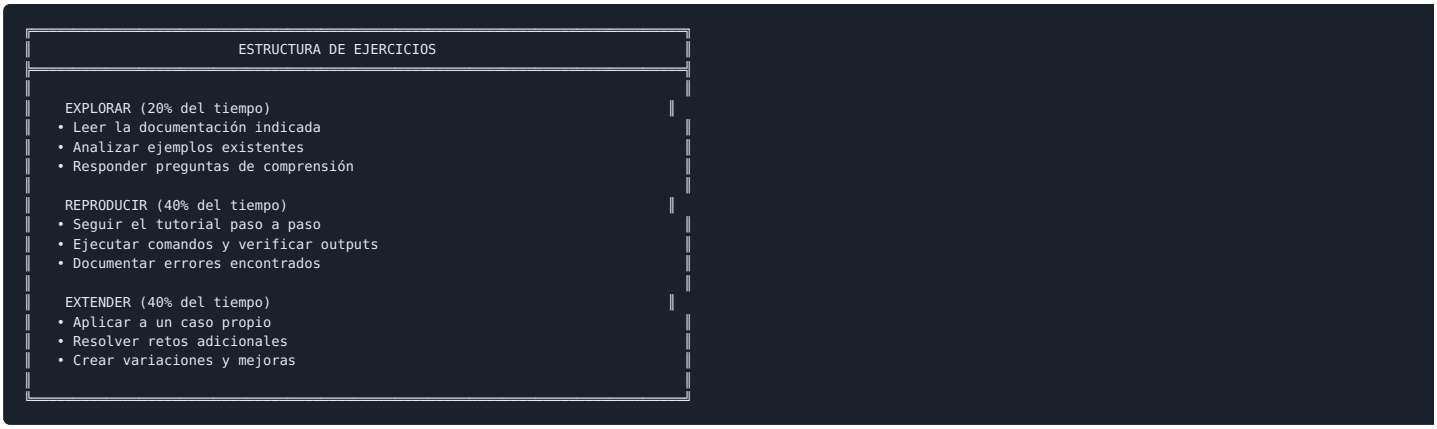
Metodología: Explorar → Reproducir → Extender

Total Ejercicios	Dificultad Progresiva	Tiempo Estimado
45+ ejercicios	Básico → Avanzado	40-60 horas

Tabla de Contenidos

Instrucciones Generales

Metodología de Cada Ejercicio



Cómo Usar Este Documento

1. **Lee el ejercicio completo** antes de comenzar
2. **Prepara tu entorno** según los prerrequisitos
3. **Sigue la metodología** Explorar → Reproducir → Extender
4. **Verifica con el script** de validación cuando esté disponible
5. **Consulta las soluciones** en EJERCICIOS_SOLUCIONES.md solo después de intentarlo

Niveles de Dificultad

Símbolo	Nivel	Tiempo Estimado
	Básico	30-60 min
	Intermedio	1-2 horas
	Avanzado	2-4 horas
	Reto Extra	Variable

Módulo 01: Fundamentos MLOps

Ejercicio 1.1 - Identificar Nivel de Madurez MLOps

Objetivo: Evaluar el nivel de madurez MLOps de un proyecto existente.

Explorar

1. Lee la sección 1.3 del Módulo 01 sobre niveles de madurez
2. Revisa el checklist de cada nivel (0-3)

Reproducir

1. Descarga o clona un proyecto ML público (ej: un repositorio de Kaggle)
2. Completa la siguiente tabla de evaluación:

```
| Criterio                    | Nivel 0 | Nivel 1 | Nivel 2 | Nivel 3 |
|:----------------------------|:-------:|:-------:|:-------:|:-------:|
| Control de versiones (Git)  |   □     |   □     |   □     |   □     |
| Versionado de datos         |   □     |   □     |   □     |   □     |
| Experimentos trackeados     |   □     |   □     |   □     |   □     |
| Tests automatizados         |   □     |   □     |   □     |   □     |
| CI/CD configurado           |   □     |   □     |   □     |   □     |
| Containerizado              |   □     |   □     |   □     |   □     |
| Monitoreo en producción     |   □     |   □     |   □     |   □     |
```

3. Determina el nivel actual y qué falta para subir al siguiente

Extender

1. **Reto 1:** Evalúa 3 proyectos diferentes y compara sus niveles
2. **Reto 2:** Crea un plan de acción para llevar un proyecto de Nivel 0 a Nivel 2
3. **Reto 3:** Diseña un script que automatice parte de esta evaluación

Criterios de Aceptación: - [ ] Tabla completada con justificaciones - [ ] Nivel identificado correctamente - [ ] Plan de mejora documentado

---

Ejercicio 1.2 - Diseñar Ciclo de Vida MLOps

Objetivo: Diseñar el ciclo de vida completo para un caso de uso específico.

Explorar

1. Lee la sección sobre el ciclo de vida MLOps (Data → Dev → Deploy → Monitor)
2. Analiza el diagrama del portafolio de referencia

Reproducir

Dado el siguiente caso de uso: > "Predictor de abandono de clientes (churn) para un banco con 100K clientes activos"

1. Dibuja el diagrama del ciclo de vida específico
2. Lista las herramientas que usarías en cada fase
3. Identifica los checkpoints de calidad

Extender

1. **Reto 1:** Adapta el ciclo para un caso de NLP (análisis de sentimiento)
2. **Reto 2:** Diseña el ciclo para un modelo de visión por computador
3. **Reto 3:** Añade consideraciones de escalabilidad (1M+ predicciones/día)

Entregable: Documento markdown con diagramas ASCII y justificaciones

---

Ejercicio 1.3 - Análisis de Trade-offs Tecnológicos

Objetivo: Evaluar y justificar decisiones tecnológicas.

Reproducir

Completa la tabla comparativa para cada categoría:

Tracking de Experimentos:

Criterio	MLflow	W&B	Neptune	Comet
Open Source				
Self-hosted				
UI/UX				
Integración sklearn				
Costo				
Recomendación				

Versionado de Datos:

Criterio	DVC	Git LFS	Delta Lake	LakeFS
Curva aprendizaje				
Pipelines ML				
Big Data				
Integración Git				
Recomendación				

Extender

Escribe un ADR (Architecture Decision Record) justificando tus elecciones.

Módulo 02: Diseño del Proyecto

Ejercicio 2.1 - Completar ML Canvas

Objetivo: Diseñar un proyecto ML usando el framework ML Canvas.

Reproducir

Para el proyecto "Predictor de Precios de Casas":

```yaml
# ml_canvas.yaml
proyecto: "House Price Predictor"

1_prediccion:
  que_predecir: ""  # Completar
  tipo_problema: "" # clasificación/regresión/clustering

2_datos:
  fuente: ""
  volumen: ""
  calidad: ""

3_features:
  principales:
    - ""
  derivadas:
    - ""

4_modelo:
  baseline: ""
  candidatos:
    - ""

5_evaluacion:
  metrica_principal: ""
  metricas_secundarias:
    - ""

6_integracion:
  como_se_usara: ""
  latencia_requerida: ""

7_feedback:
  como_mejorar: ""
```

Extender

1. **Reto 1:** Crea un ML Canvas para un sistema de recomendación
2. **Reto 2:** Crea un ML Canvas para detección de fraude
3. **Reto 3:** Presenta tu ML Canvas en formato visual (diagrama)

Ejercicio 2.2 - Definir Métricas de Éxito

Objetivo: Seleccionar y justificar métricas apropiadas.

Reproducir

Para cada escenario, define: - Métrica técnica principal - Métricas secundarias - Métrica de negocio

Escenario	Métrica Técnica	Secundarias	Negocio
Detección de spam			
Predicción de ventas			
Diagnóstico médico			
Sistema de recomendación			
Predicción de churn			

Extender

Implementa una función Python que calcule todas las métricas para un clasificador:

```python
def evaluate_classifier(y_true, y_pred, y_prob):
    """
    Calcula métricas completas para clasificación binaria.

    Returns:
        dict con accuracy, precision, recall, f1, auc, etc.
    """
    # Tu implementación aquí
    pass
```

Módulo 03: Estructura del Repositorio

Ejercicio 3.1 - Crear Estructura Profesional

Objetivo: Inicializar un proyecto con estructura profesional.

Reproducir

Ejecuta los siguientes comandos para crear la estructura:

```bash
# Crear proyecto
mkdir mi_proyecto_ml && cd mi_proyecto_ml

# Crear estructura
mkdir -p {data/{raw,processed,external},src/{data,features,models,visualization},tests,notebooks,configs,docs}

# Crear archivos base
touch README.md requirements.txt setup.py .gitignore
touch src/__init__.py src/data/__init__.py src/models/__init__.py
touch tests/__init__.py tests/test_data.py tests/test_model.py

# Verificar estructura
tree -L 2
```

Output Esperado:

```
mi_proyecto_ml/
├── README.md
├── requirements.txt
├── setup.py
├── .gitignore
├── configs/
├── data/
│   ├── external/
│   ├── processed/
│   └── raw/
├── docs/
├── notebooks/
├── src/
│   ├── __init__.py
│   ├── data/
│   ├── features/
│   ├── models/
│   └── visualization/
└── tests/
    ├── __init__.py
    ├── test_data.py
    └── test_model.py
```

Extender

1. **Reto 1:** Añade un Makefile con comandos útiles
2. **Reto 2:** Crea un script `create_project.sh` que automatice esto
3. **Reto 3:** Añade pre-commit hooks para linting

Script de Validación:

```bash
#!/bin/bash
# validate_structure.sh
REQUIRED_DIRS=("data/raw" "data/processed" "src" "tests" "notebooks" "configs")
REQUIRED_FILES=("README.md" "requirements.txt" ".gitignore")

echo "Validando estructura..."
for dir in "${REQUIRED_DIRS[@]}"; do
    if [ -d "$dir" ]; then
        echo "✓ $dir"
    else
        echo "✗ $dir - FALTA"
    fi
done

for file in "${REQUIRED_FILES[@]}"; do
    if [ -f "$file" ]; then
        echo "✓ $file"
    else
        echo "✗ $file - FALTA"
    fi
done
```

Módulo 04: Git y GitHub

Ejercicio 4.1 - Flujo de Trabajo Git Básico

Objetivo: Dominar el flujo básico de Git.

Reproducir

```bash
# 1. Inicializar repositorio
git init
git config user.name "Tu Nombre"
git config user.email "tu@email.com"

# 2. Primer commit
echo "# Mi Proyecto ML" > README.md
git add README.md
git commit -m "feat: initial commit with README"

# 3. Crear branch de desarrollo
git checkout -b develop

# 4. Añadir feature
git checkout -b feature/add-requirements
echo "pandas==2.0.0" > requirements.txt
git add requirements.txt
git commit -m "feat: add initial requirements"

# 5. Merge a develop
git checkout develop
git merge feature/add-requirements

# 6. Ver historial
git log --oneline --graph --all
```

Extender

1. **Reto 1:** Simula un conflicto de merge y resuélvelo
2. **Reto 2:** Usa `git rebase` en lugar de merge
3. **Reto 3:** Configura un hook pre-commit que verifique el formato de commits

---

Ejercicio 4.2 - Pull Request Profesional

Objetivo: Crear un Pull Request con descripción profesional.

Reproducir

1. Crea un fork del repositorio de ejemplo (o usa el tuyo)
2. Crea una rama `feature/improve-readme`
3. Realiza cambios y haz commit
4. Crea un PR con esta plantilla:

```bash
##  Descripción
[Breve descripción del cambio]

##  Tipo de Cambio
- [ ] Bug fix
- [ ] Nueva feature
- [ ] Mejora de documentación
- [ ] Refactoring

##  Checklist
- [ ] El código sigue las guías de estilo
- [ ] He añadido tests
- [ ] La documentación está actualizada
- [ ] Todos los tests pasan

##  Screenshots (si aplica)
[Capturas de pantalla]

##  Issues Relacionados
Closes #XX
```

Módulo 05: DVC

Ejercicio 5.1 - Inicializar DVC

Objetivo: Configurar DVC en un proyecto existente.

Reproducir

```bash
# 1. Instalar DVC
pip install dvc dvc-s3

# 2. Inicializar
cd mi_proyecto_ml
dvc init

# 3. Configurar remote local (para práctica)
mkdir -p /tmp/dvc-storage
dvc remote add -d myremote /tmp/dvc-storage

# 4. Trackear un dataset
# Primero descarga un dataset de ejemplo
curl -o data/raw/iris.csv https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv

# Añadir a DVC
dvc add data/raw/iris.csv

# 5. Commit cambios
git add data/raw/iris.csv.dvc data/raw/.gitignore
git commit -m "data: add iris dataset with DVC"

# 6. Push datos
dvc push

# 7. Verificar
dvc status
```

Extender

1. **Reto 1:** Configura un remote en S3 o GCS
2. **Reto 2:** Crea un pipeline DVC con `dvc.yaml`
3. **Reto 3:** Implementa versionado de datasets (v1, v2) con tags

---

Ejercicio 5.2 - Pipeline DVC

Objetivo: Crear un pipeline reproducible con DVC.

Reproducir

Crea los siguientes archivos:

dvc.yaml:

```yaml
stages:
  prepare:
    cmd: python src/data/prepare.py
    deps:
      - src/data/prepare.py
      - data/raw/iris.csv
    outs:
      - data/processed/iris_clean.csv

  train:
    cmd: python src/models/train.py
    deps:
      - src/models/train.py
      - data/processed/iris_clean.csv
    outs:
      - models/model.pkl
    metrics:
      - metrics/scores.json:
          cache: false

  evaluate:
    cmd: python src/models/evaluate.py
    deps:
      - src/models/evaluate.py
      - models/model.pkl
      - data/processed/iris_clean.csv
    metrics:
      - metrics/evaluation.json:
          cache: false
```

```bash
# Ejecutar pipeline
dvc repro

# Ver métricas
dvc metrics show

# Comparar con versión anterior
dvc metrics diff
```

Módulo 06: Pipeline ML

Ejercicio 6.1 - Pipeline sklearn Básico

Objetivo: Crear un pipeline de preprocessing + modelo.

Reproducir

```python
# src/models/pipeline.py
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
import joblib

def create_pipeline():
    """Crea pipeline de preprocessing + modelo."""

    # Definir transformadores
    numeric_features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', StandardScaler(), numeric_features),
        ])

    # Pipeline completo
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))
    ])

    return pipeline

def train(data_path: str, model_path: str):
    """Entrena y guarda el modelo."""
    # Cargar datos
    df = pd.read_csv(data_path)
    X = df.drop('species', axis=1)
    y = df['species']

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

    # Entrenar
    pipeline = create_pipeline()
    pipeline.fit(X_train, y_train)

    # Evaluar
    score = pipeline.score(X_test, y_test)
    print(f"Test accuracy: {score:.4f}")

    # Guardar
    joblib.dump(pipeline, model_path)
    print(f"Model saved to {model_path}")

    return pipeline, score

if __name__ == "__main__":
    train("data/processed/iris_clean.csv", "models/model.pkl")
```

Extender

1. **Reto 1:** Añade validación cruzada al pipeline
2. **Reto 2:** Implementa búsqueda de hiperparámetros con GridSearchCV
3. **Reto 3:** Crea un pipeline para datos con features categóricas y numéricas

---

Ejercicio 6.2 - Evitar Data Leakage

Objetivo: Identificar y corregir data leakage.

Reproducir

El siguiente código tiene data leakage. Identifica el problema y corrígelo:

```python
# ✗ CÓDIGO CON LEAKAGE - CORRÍGELO
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Cargar datos
df = pd.read_csv("data.csv")
X = df.drop('target', axis=1)
y = df['target']

# Normalizar ANTES del split (LEAKAGE!)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2
)

# Entrenar
model = RandomForestClassifier()
model.fit(X_train, y_train)
print(f"Score: {model.score(X_test, y_test)}")
```

Preguntas: 1. ¿Dónde está el data leakage? 2. ¿Por qué es un problema? 3. ¿Cómo lo corriges?

Módulo 07: MLflow

Ejercicio 7.1 - Tracking Básico

Objetivo: Registrar un experimento en MLflow.

Reproducir

```python
# src/experiments/train_with_mlflow.py
import mlflow
import mlflow.sklearn
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score

# Configurar tracking
mlflow.set_tracking_uri("mlruns")
mlflow.set_experiment("iris-classification")

# Cargar datos
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, test_size=0.2, random_state=42
)

# Experimento
with mlflow.start_run(run_name="rf-baseline"):
    # Parámetros
    n_estimators = 100
    max_depth = 5

    mlflow.log_param("n_estimators", n_estimators)
    mlflow.log_param("max_depth", max_depth)

    # Entrenar
    model = RandomForestClassifier(
        n_estimators=n_estimators,
        max_depth=max_depth,
        random_state=42
    )
    model.fit(X_train, y_train)

    # Evaluar
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')

    mlflow.log_metric("accuracy", accuracy)
    mlflow.log_metric("f1_score", f1)

    # Guardar modelo
    mlflow.sklearn.log_model(model, "model")

    print(f"Run ID: {mlflow.active_run().info.run_id}")
    print(f"Accuracy: {accuracy:.4f}")

# Ver UI
# mlflow ui --port 5000
```

Extender

1. **Reto 1:** Compara 3 modelos diferentes en el mismo experimento
2. **Reto 2:** Usa autologging: `mlflow.sklearn.autolog()`
3. **Reto 3:** Registra un modelo en el Model Registry

Módulo 08: Testing

Ejercicio 8.1 - Tests Unitarios Básicos

Objetivo: Escribir tests para funciones de data processing.

Reproducir

```python
# tests/test_data.py
import pytest
import pandas as pd
import numpy as np
from src.data.prepare import clean_data, validate_schema

class TestDataCleaning:
    """Tests para funciones de limpieza de datos."""

    def test_clean_data_removes_nulls(self):
        """Verifica que clean_data elimina filas con nulls."""
        df = pd.DataFrame({
            'a': [1, 2, np.nan],
            'b': [4, 5, 6]
        })
        result = clean_data(df)
        assert result.isnull().sum().sum() == 0

    def test_clean_data_preserves_valid_rows(self):
        """Verifica que no se eliminan filas válidas."""
        df = pd.DataFrame({
            'a': [1, 2, 3],
            'b': [4, 5, 6]
        })
        result = clean_data(df)
        assert len(result) == 3

    def test_validate_schema_correct(self):
        """Verifica schema válido."""
        df = pd.DataFrame({
            'feature1': [1.0, 2.0],
            'feature2': [3.0, 4.0],
            'target': [0, 1]
        })
        expected_schema = ['feature1', 'feature2', 'target']
        assert validate_schema(df, expected_schema) == True

    def test_validate_schema_missing_column(self):
        """Verifica que detecta columnas faltantes."""
        df = pd.DataFrame({
            'feature1': [1.0, 2.0]
        })
        expected_schema = ['feature1', 'feature2', 'target']
        with pytest.raises(ValueError):
            validate_schema(df, expected_schema)
```

```python
# Ejecutar tests
pytest tests/test_data.py -v

# Con coverage
pytest tests/ --cov=src --cov-report=html
```

Ejercicio 8.2 - Tests de Modelo

Objetivo: Escribir tests para el modelo ML.

Reproducir

```python
# tests/test_model.py
import pytest
import numpy as np
from sklearn.datasets import make_classification
from src.models.pipeline import create_pipeline

class TestModel:
    """Tests para el modelo ML."""

    @pytest.fixture
    def sample_data(self):
        """Genera datos de ejemplo."""
        X, y = make_classification(
            n_samples=100,
            n_features=4,
            n_classes=2,
            random_state=42
        )
        return X, y

    def test_pipeline_fits(self, sample_data):
        """Verifica que el pipeline entrena sin errores."""
        X, y = sample_data
        pipeline = create_pipeline()
        pipeline.fit(X, y)
        assert hasattr(pipeline, 'predict')

    def test_predictions_valid_shape(self, sample_data):
        """Verifica shape de predicciones."""
        X, y = sample_data
        pipeline = create_pipeline()
        pipeline.fit(X, y)
        predictions = pipeline.predict(X)
        assert predictions.shape == y.shape

    def test_predictions_valid_values(self, sample_data):
        """Verifica que predicciones son clases válidas."""
        X, y = sample_data
        pipeline = create_pipeline()
        pipeline.fit(X, y)
        predictions = pipeline.predict(X)
        assert set(predictions).issubset(set(y))

    def test_model_accuracy_above_baseline(self, sample_data):
        """Verifica que accuracy supera baseline random."""
        X, y = sample_data
        pipeline = create_pipeline()
        pipeline.fit(X, y)
        accuracy = pipeline.score(X, y)
        # Baseline para clasificación binaria = 0.5
        assert accuracy > 0.5
```

Módulo 09: CI/CD

Ejercicio 9.1 - GitHub Actions Básico

Objetivo: Configurar CI básico con GitHub Actions.

Reproducir

Crea el archivo `.github/workflows/ci.yml`:

```yaml
name: CI Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.10'

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
          pip install pytest pytest-cov flake8

      - name: Lint with flake8
        run: |
          flake8 src/ tests/ --max-line-length=100

      - name: Run tests
        run: |
          pytest tests/ -v --cov=src --cov-report=xml

      - name: Upload coverage
        uses: codecov/codecov-action@v3
        with:
          file: ./coverage.xml
```

Extender

1. **Reto 1:** Añade job para construir Docker image
2. **Reto 2:** Añade matriz de versiones de Python
3. **Reto 3:** Configura deploy automático a staging

Módulo 10: Docker

Ejercicio 10.1 - Dockerfile para ML

Objetivo: Crear un Dockerfile optimizado para modelos ML.

Reproducir

```dockerfile
# Dockerfile
FROM python:3.10-slim

WORKDIR /app

# Instalar dependencias del sistema
RUN apt-get update && apt-get install -y \
    gcc \
    && rm -rf /var/lib/apt/lists/*

# Copiar requirements primero (para cache)
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copiar código
COPY src/ ./src/
COPY models/ ./models/

# Variables de entorno
ENV PYTHONPATH=/app
ENV MODEL_PATH=/app/models/model.pkl

# Puerto
EXPOSE 8000

# Comando por defecto
CMD ["python", "-m", "src.api.main"]
```

```bash
# Build
docker build -t ml-model:v1 .

# Run
docker run -p 8000:8000 ml-model:v1

# Verificar tamaño
docker images ml-model:v1
```

Extender

1. **Reto 1:** Reduce el tamaño de la imagen a < 500MB
2. **Reto 2:** Implementa multi-stage build
3. **Reto 3:** Añade healthcheck

Módulo 11: FastAPI

Ejercicio 11.1 - API REST Básica

Objetivo: Crear una API para servir predicciones.

Reproducir

```
# src/api/main.py
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import joblib
import numpy as np

app = FastAPI(
    title="ML Model API",
    description="API para predicciones de Iris",
    version="1.0.0"
)

# Cargar modelo
model = joblib.load("models/model.pkl")

class PredictionInput(BaseModel):
    sepal_length: float
    sepal_width: float
    petal_length: float
    petal_width: float

class PredictionOutput(BaseModel):
    prediction: str
    probability: float

@app.get("/health")
def health_check():
    return {"status": "healthy"}

@app.post("/predict", response_model=PredictionOutput)
def predict(input_data: PredictionInput):
    try:
        features = np.array([[
            input_data.sepal_length,
            input_data.sepal_width,
            input_data.petal_length,
            input_data.petal_width
        ]])

        prediction = model.predict(features)[0]
        probability = model.predict_proba(features).max()

        return PredictionOutput(
            prediction=str(prediction),
            probability=float(probability)
        )
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

```
# Ejecutar
uvicorn src.api.main:app --reload

# Probar
curl -X POST "http://localhost:8000/predict" \
  -H "Content-Type: application/json" \
  -d '{"sepal_length": 5.1, "sepal_width": 3.5, "petal_length": 1.4, "petal_width": 0.2}'
```

Módulo 14: Monitoreo

Ejercicio 14.1 - Logging Estructurado

Objetivo: Implementar logging profesional.

Reproducir

```python
# src/utils/logger.py
import logging
import json
from datetime import datetime

class JSONFormatter(logging.Formatter):
    def format(self, record):
        log_record = {
            "timestamp": datetime.utcnow().isoformat(),
            "level": record.levelname,
            "message": record.getMessage(),
            "module": record.module,
            "function": record.funcName,
        }
        if hasattr(record, 'extra'):
            log_record.update(record.extra)
        return json.dumps(log_record)

def get_logger(name: str) -> logging.Logger:
    logger = logging.getLogger(name)
    logger.setLevel(logging.INFO)

    handler = logging.StreamHandler()
    handler.setFormatter(JSONFormatter())
    logger.addHandler(handler)

    return logger

# Uso
logger = get_logger("ml-api")
logger.info("Prediction made", extra={"model": "rf", "latency_ms": 45})
```

Módulos 15-17: Documentación y Demo

Ejercicio 15.1 - README Profesional

Objetivo: Crear un README siguiendo best practices.

Reproducir

Usa la plantilla en templates/README_TEMPLATE.md y completa todas las secciones para tu proyecto.

Checklist del README: - [ ] Título y descripción clara - [ ] Badges (CI, coverage, license) - [ ] Tabla de contenidos - [ ] Instalación paso a paso - [ ] Uso co ejemplos - [ ] Estructura del proyecto - [ ] Contribución - [ ] Licencia - [ ] Contacto

---

Proyecto Integrador Final

Descripción

Desarrolla un portafolio MLOps completo desde cero que incluya:

1. **Proyecto 1:** Clasificación (ej: Churn Prediction)
2. **Proyecto 2:** Regresión (ej: Price Prediction)
3. **Proyecto 3:** NLP o CV (ej: Sentiment Analysis)

Requisitos Mínimos

Componente	Requisito
Repositorio	Estructura profesional
Versionado	Git + DVC configurados
Pipeline	sklearn pipeline reproducible
Tracking	MLflow con al menos 5 experimentos
Testing	> 70% coverage
CI/CD	GitHub Actions funcional
Containerización	Dockerfile optimizado
API	FastAPI con /health y /predict
Documentación	README + Model Card
Demo	Video 3-5 minutos

Entrega

1. URL del repositorio GitHub público
2. Link al video demo (YouTube/Loom)
3. Link a la documentación desplegada (opcional)

Evaluación

Ver RUBRICA_EVALUACION.md para criterios detallados.

Soluciones disponibles en: EJERCICIOS_SOLUCIONES.md

Índice | Rúbrica | Syllabus