SOLUCIONES DE EJERCICIOS - GUÍA MLOps

Soluciones Detalladas Paso a Paso

Guía MLOps v3.0 | DuqueOM | Noviembre 2025

SOLUCIONES DE EJERCICIOS

Soluciones Detalladas con Explicaciones

⚠ *Consulta solo después de intentar resolver los ejercicios*

Tabla de Contenidos

Módulo 01: Fundamentos

Solución 1.1 - Identificar Nivel de Madurez MLOps

Tabla de Evaluación Completada (Ejemplo para proyecto típico de Kaggle):

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	✓			

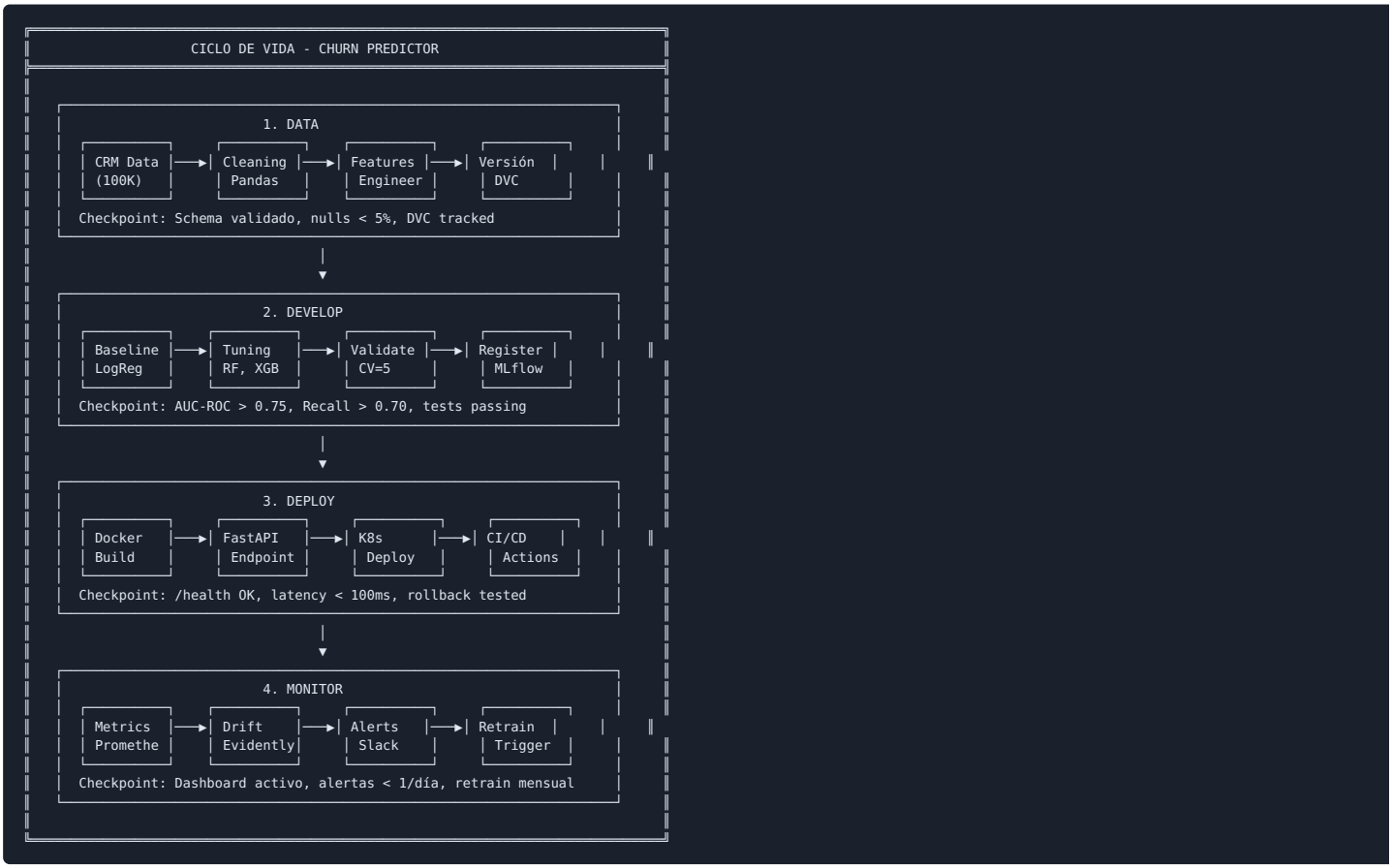
Resultado: Nivel 0 - Manual, no reproducible

Plan de mejora a Nivel 2:

```
## Plan de Mejora: Nivel 0 → Nivel 2

### Fase 1: Fundamentos (Semana 1-2)
1. Inicializar Git y crear estructura de repositorio
2. Añadir .gitignore y requirements.txt
3. Convertir notebooks a scripts modulares

### Fase 2: Versionado (Semana 3)
1. Inicializar DVC
2. Trackear datasets con DVC
3. Configurar remote storage

### Fase 3: Automatización (Semana 4-5)
1. Crear pipeline sklearn
2. Configurar MLflow para tracking
3. Escribir tests unitarios básicos
4. Configurar GitHub Actions CI

### Fase 4: Deployment (Semana 6)
1. Crear Dockerfile
2. Exponer modelo via FastAPI
3. Documentar con README profesional
```

Solución 1.2 - Diseñar Ciclo de Vida MLOps

Diagrama para Predictor de Churn:



Herramientas por Fase:

Fase	Herramientas	Justificación
Data	Pandas, DVC, Great Expectations	Estándar industria, integra con Git
Develop	sklearn, MLflow, pytest	Reproducible, trackeable
Deploy	Docker, FastAPI, GitHub Actions, K8s	Escalable, automatizado
Monitor	Prometheus, Grafana, Evidently	Open source, completo

Solución 1.3 - Análisis de Trade-offs

Tracking de Experimentos:

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

Justificación MLflow: - Open source = sin vendor lock-in - Self-hosted = control de datos - Model Registry incluido - Estándar de facto en la industria

ADR Ejemplo:

```
# ADR-001: Selección de Experiment Tracking

## Estado
Aceptado

## Contexto
Necesitamos una herramienta para trackear experimentos ML que sea:
- Fácil de aprender para estudiantes
- Gratuita y sin límites de uso
- Integrable con nuestro stack (sklearn, Python)
- Con posibilidad de self-hosting

## Decisión
Usaremos **MLflow** como herramienta de experiment tracking.

## Consecuencias

### Positivas
- Sin costos de licencia
- Control total de los datos
- Amplia documentación y comunidad
- Model Registry integrado

### Negativas
- UI menos pulida que W&B
- Requiere setup inicial de servidor
- Menos features de colaboración

## Alternativas Consideradas
- W&B: Descartada por ser SaaS con límites en tier gratuito
- Neptune: Similar a W&B
- Sin tracking: Descartado por no ser reproducible
```

Módulo 02: Diseño

Solución 2.1 - ML Canvas Completado

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

1_prediccion:
  que_predecir: "Precio de venta de una casa en USD"
  tipo_problema: "regresión"
  granularidad: "Por propiedad individual"

2_datos:
  fuente: "Kaggle House Prices Dataset / MLS listings"
  volumen: "~1500 propiedades training, actualización mensual"
  calidad: "Medio - requiere limpieza de outliers y nulls"
  privacidad: "Datos públicos, sin PII"

3_features:
  principales:
    - "OverallQual: Calidad general (1-10)"
    - "GrLivArea: Área habitable (sqft)"
    - "TotalBsmtSF: Área sótano (sqft)"
    - "GarageCars: Capacidad garage"
  derivadas:
    - "TotalSF = GrLivArea + TotalBsmtSF"
    - "Age = YrSold - YearBuilt"
    - "Remodeled = YearRemodAdd > YearBuilt"
    - "PricePerSqFt (solo para análisis, no feature)"

4_modelo:
  baseline: "LinearRegression con features top-5"
  candidatos:
    - "Ridge/Lasso para regularización"
    - "RandomForest para no-linealidad"
    - "XGBoost para mejor performance"
    - "Stacking ensemble final"

5_evaluacion:
  metrica_principal: "RMSE (Root Mean Squared Error)"
  metricas_secundarias:
    - "MAE (Mean Absolute Error)"
    - "R² (Coeficiente de determinación)"
    - "MAPE (Mean Absolute Percentage Error)"
  threshold: "RMSE < $25,000 para producción"

6_integracion:
  como_se_usara: "API REST para tasaciones instantáneas"
  latencia_requerida: "< 200ms p99"
  volumen_esperado: "~1000 requests/día"
  usuarios: "Agentes inmobiliarios, compradores"

7_feedback:
  como_mejorar: "Comparar predicción vs precio real de venta"
  frecuencia_retrain: "Trimestral o si MAPE > 15%"
  drift_monitoring: "Distribución de features principales"
```

Módulo 03: Estructura

Solución 3.1 - Script de Automatización

```bash
#!/bin/bash
# create_ml_project.sh
# Uso: ./create_ml_project.sh nombre_proyecto

PROJECT_NAME=${1:-"ml_project"}

echo " Creando proyecto: $PROJECT_NAME"

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

cd "$PROJECT_NAME"

# Crear archivos Python
touch src/__init__.py
touch src/data/__init__.py src/data/prepare.py src/data/validate.py
touch src/features/__init__.py src/features/build_features.py
touch src/models/__init__.py src/models/train.py src/models/predict.py
touch tests/__init__.py tests/test_data.py tests/test_model.py

# README.md
cat > README.md << 'EOF'
# Project Name

## Description
Brief description of the project.

## Installation
```bash
pip install -r requirements.txt
```

Usage

```bash
python src/models/train.py
```

Project Structure

```
├── data/
├── src/
├── tests/
├── notebooks/
└── models/
```

EOF
```

requirements.txt

cat > requirements.txt << 'EOF' pandas>=2.0.0 numpy>=1.24.0 scikit-learn>=1.3.0 mlflow>=2.8.0 pytest>=7.4.0 black>=23.0.0 flake8>=6.1.0 EOF

.gitignore

cat > .gitignore << 'EOF' # Python **pycache**/ *.py[cod] .env venv/

Data

data/raw/ *data/processed/* !data/raw/.gitkeep !data/processed/.gitkeep

Models

models/*.pkl models/*.joblib

IDE

.vscode/ .idea/

MLflow

mlruns/

DVC

/dvc.lock EOF

Makefile

cat > Makefile << 'EOF' .PHONY: install test lint train clean

install: pip install -r requirements.txt

test: pytest tests/ -v –cov=src

lint: flake8 src/ tests/ black src/ tests/ –check

format: black src/ tests/

train: python src/models/train.py

clean: find . -type d -name **pycache** -exec rm -rf {} + find . -type f -name "*.pyc" -delete EOF

.gitkeep en carpetas vacías

touch data/raw/.gitkeep data/processed/.gitkeep data/external/.gitkeep

echo " Proyecto $PROJECT_NAME creado exitosamente!" echo "" echo "Siguientes pasos:" echo " cd $PROJECT_NAME" echo " git init" echo " python -m ven venv" echo " source venv/bin/activate" echo " make install"

````
---

## Módulo 05: DVC

### Solución 5.2 - Pipeline DVC Completo

**src/data/prepare.py:**
```python
"""Preparación de datos."""
import pandas as pd
from pathlib import Path

def prepare_data(input_path: str, output_path: str) -> None:
    """Limpia y prepara el dataset."""
    print(f" Leyendo datos de {input_path}")
    df = pd.read_csv(input_path)

    # Limpieza
    df = df.dropna()
    df.columns = df.columns.str.lower().str.replace(' ', '_')

    # Guardar
    Path(output_path).parent.mkdir(parents=True, exist_ok=True)
    df.to_csv(output_path, index=False)
    print(f" Datos guardados en {output_path}")
    print(f"   Filas: {len(df)}, Columnas: {len(df.columns)}")

if __name__ == "__main__":
    prepare_data("data/raw/iris.csv", "data/processed/iris_clean.csv")
````

src/models/train.py:

```python
"""Entrenamiento del modelo."""
import pandas as pd
import joblib
import json
from pathlib import Path
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score

def train_model(data_path: str, model_path: str, metrics_path: str) -> None:
    """Entrena y guarda el modelo con métricas."""
    print(f" Cargando datos de {data_path}")
    df = pd.read_csv(data_path)

    X = df.drop('species', axis=1)
    y = df['species']

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

    print(" Entrenando modelo...")
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    # Evaluar
    y_pred = model.predict(X_test)
    metrics = {
        "accuracy": accuracy_score(y_test, y_pred),
        "f1_weighted": f1_score(y_test, y_pred, average='weighted')
    }

    # Guardar modelo
    Path(model_path).parent.mkdir(parents=True, exist_ok=True)
    joblib.dump(model, model_path)
    print(f" Modelo guardado en {model_path}")

    # Guardar métricas
    Path(metrics_path).parent.mkdir(parents=True, exist_ok=True)
    with open(metrics_path, 'w') as f:
        json.dump(metrics, f, indent=2)
    print(f" Métricas: {metrics}")

if __name__ == "__main__":
    train_model(
        "data/processed/iris_clean.csv",
        "models/model.pkl",
        "metrics/scores.json"
    )
```

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
```

Ejecución:
```

```
# Ejecutar pipeline completo
dvc repro

# Salida esperada:
# Running stage 'prepare':
#  Leyendo datos de data/raw/iris.csv
#  Datos guardados en data/processed/iris_clean.csv
# Running stage 'train':
#  Cargando datos de data/processed/iris_clean.csv
#  Entrenando modelo...
#  Modelo guardado en models/model.pkl
#  Métricas: {'accuracy': 1.0, 'f1_weighted': 1.0}

# Ver métricas
dvc metrics show
# metrics/scores.json:
#     accuracy: 1.0
#     f1_weighted: 1.0
```

Módulo 06: Pipeline ML

Solución 6.2 - Corregir Data Leakage

Problema identificado: El `StandardScaler` se ajusta (`fit_transform`) sobre TODO el dataset antes del split. Esto significa que las estadísticas (media, std) de scaler incluyen información del test set, causando data leakage.

Código Corregido:

```
#  CÓDIGO SIN LEAKAGE
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline

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

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

# Opción 1: Fit solo en train, transform en ambos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)  # fit + transform
X_test_scaled = scaler.transform(X_test)        # solo transform

model = RandomForestClassifier()
model.fit(X_train_scaled, y_train)
print(f"Score: {model.score(X_test_scaled, y_test)}")

# Opción 2 (MEJOR): Usar Pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', RandomForestClassifier())
])

pipeline.fit(X_train, y_train)  # Pipeline maneja el fit/transform correctamente
print(f"Score: {pipeline.score(X_test, y_test)}")
```

Por qué es un problema: 1. El modelo “ve” información del test durante entrenamiento 2. Las métricas de evaluación son optimistas (sobreestimadas) 3. E modelo no generaliza igual en producción

---

Módulo 08: Testing

Solución 8.1 - Implementación Completa

src/data/prepare.py:

```python
"""Funciones de preparación de datos."""
import pandas as pd
from typing import List

def clean_data(df: pd.DataFrame) -> pd.DataFrame:
    """
    Limpia un DataFrame eliminando filas con valores nulos.

    Args:
        df: DataFrame a limpiar

    Returns:
        DataFrame sin valores nulos
    """
    return df.dropna()

def validate_schema(df: pd.DataFrame, expected_columns: List[str]) -> bool:
    """
    Valida que el DataFrame tenga las columnas esperadas.

    Args:
        df: DataFrame a validar
        expected_columns: Lista de columnas requeridas

    Returns:
        True si el schema es válido

    Raises:
        ValueError: Si faltan columnas
    """
    missing = set(expected_columns) - set(df.columns)
    if missing:
        raise ValueError(f"Columnas faltantes: {missing}")
    return True
```

tests/test_data.py completo:

```python
"""Tests para módulo de datos."""
import pytest
import pandas as pd
import numpy as np
from src.data.prepare import clean_data, validate_schema

class TestCleanData:
    """Tests para la función clean_data."""

    def test_removes_rows_with_nulls(self):
        """Verifica eliminación de filas con nulls."""
        df = pd.DataFrame({
            'a': [1, 2, np.nan, 4],
            'b': [5, np.nan, 7, 8]
        })
        result = clean_data(df)

        assert len(result) == 2
        assert result.isnull().sum().sum() == 0

    def test_preserves_complete_rows(self):
        """Verifica que filas completas se mantienen."""
        df = pd.DataFrame({
            'a': [1, 2, 3],
            'b': [4, 5, 6]
        })
        result = clean_data(df)

        assert len(result) == 3
        pd.testing.assert_frame_equal(result, df)

    def test_empty_dataframe(self):
        """Verifica comportamiento con DataFrame vacío."""
        df = pd.DataFrame()
        result = clean_data(df)

        assert len(result) == 0

    def test_all_nulls(self):
        """Verifica que DataFrame con todos nulls queda vacío."""
        df = pd.DataFrame({
            'a': [np.nan, np.nan],
            'b': [np.nan, np.nan]
        })
        result = clean_data(df)

        assert len(result) == 0


class TestValidateSchema:
    """Tests para la función validate_schema."""

    def test_valid_schema(self):
        """Verifica schema válido."""
        df = pd.DataFrame({
            'feature1': [1, 2],
            'feature2': [3, 4],
            'target': [0, 1]
        })
        expected = ['feature1', 'feature2', 'target']

        assert validate_schema(df, expected) is True

    def test_extra_columns_ok(self):
        """Verifica que columnas extra no causan error."""
        df = pd.DataFrame({
            'feature1': [1, 2],
            'feature2': [3, 4],
            'extra': [5, 6]
        })
        expected = ['feature1', 'feature2']

        assert validate_schema(df, expected) is True

    def test_missing_column_raises(self):
        """Verifica que columna faltante lanza error."""
        df = pd.DataFrame({
            'feature1': [1, 2]
        })
        expected = ['feature1', 'feature2', 'target']

        with pytest.raises(ValueError, match="Columnas faltantes"):
            validate_schema(df, expected)

    def test_empty_expected(self):
        """Verifica que lista vacía de expected es válida."""
        df = pd.DataFrame({'a': [1]})

        assert validate_schema(df, []) is True
```

Ejecución:

```
$ pytest tests/test_data.py -v

========================= test session starts ==========================
collected 8 items

tests/test_data.py::TestCleanData::test_removes_rows_with_nulls PASSED
tests/test_data.py::TestCleanData::test_preserves_complete_rows PASSED
tests/test_data.py::TestCleanData::test_empty_dataframe PASSED
tests/test_data.py::TestCleanData::test_all_nulls PASSED
tests/test_data.py::TestValidateSchema::test_valid_schema PASSED
tests/test_data.py::TestValidateSchema::test_extra_columns_ok PASSED
tests/test_data.py::TestValidateSchema::test_missing_column_raises PASSED
tests/test_data.py::TestValidateSchema::test_empty_expected PASSED

========================== 8 passed in 0.45s ===========================
```

Módulo 10: Docker

Solución 10.1 - Dockerfile Optimizado

```dockerfile
# Dockerfile optimizado con multi-stage build
# Stage 1: Builder
FROM python:3.10-slim as builder

WORKDIR /app

# Instalar dependencias de compilación
RUN apt-get update && apt-get install -y \
    gcc \
    python3-dev \
    && rm -rf /var/lib/apt/lists/*

# Crear virtualenv
RUN python -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

# Instalar dependencias Python
COPY requirements.txt .
RUN pip install --no-cache-dir --upgrade pip && \
    pip install --no-cache-dir -r requirements.txt

# Stage 2: Runtime
FROM python:3.10-slim as runtime

WORKDIR /app

# Copiar virtualenv del builder
COPY --from=builder /opt/venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

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

# Crear usuario no-root
RUN useradd --create-home appuser && \
    chown -R appuser:appuser /app
USER appuser

# Variables de entorno
ENV PYTHONPATH=/app
ENV PYTHONUNBUFFERED=1
ENV MODEL_PATH=/app/models/model.pkl

# Puerto
EXPOSE 8000

# Healthcheck
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:8000/health || exit 1

# Comando
CMD ["uvicorn", "src.api.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Verificación de tamaño:

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

# Verificar tamaño
docker images ml-model:optimized
# REPOSITORY          TAG         SIZE
# ml-model           optimized    ~350MB  (vs ~1.2GB sin optimizar)
```

Módulo 11: FastAPI

Solución 11.1 - API Completa con Tests

src/api/main.py:

```python
"""API REST para modelo ML."""
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field
import joblib
import numpy as np
from pathlib import Path
import logging

# Configurar logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Crear app
app = FastAPI(
    title="Iris Classifier API",
    description="API para clasificación de flores Iris",
    version="1.0.0"
)

# Cargar modelo
MODEL_PATH = Path("models/model.pkl")
if MODEL_PATH.exists():
    model = joblib.load(MODEL_PATH)
    logger.info(f"Modelo cargado desde {MODEL_PATH}")
else:
    model = None
    logger.warning("Modelo no encontrado")

# Schemas
class IrisInput(BaseModel):
    """Input para predicción."""
    sepal_length: float = Field(..., ge=0, le=10, description="Longitud del sépalo (cm)")
    sepal_width: float = Field(..., ge=0, le=10, description="Ancho del sépalo (cm)")
    petal_length: float = Field(..., ge=0, le=10, description="Longitud del pétalo (cm)")
    petal_width: float = Field(..., ge=0, le=10, description="Ancho del pétalo (cm)")

    class Config:
        json_schema_extra = {
            "example": {
                "sepal_length": 5.1,
                "sepal_width": 3.5,
                "petal_length": 1.4,
                "petal_width": 0.2
            }
        }

class PredictionResponse(BaseModel):
    """Respuesta de predicción."""
    prediction: str
    confidence: float
    probabilities: dict

class HealthResponse(BaseModel):
    """Respuesta de health check."""
    status: str
    model_loaded: bool

# Endpoints
@app.get("/health", response_model=HealthResponse)
def health_check():
    """Verifica el estado del servicio."""
    return HealthResponse(
        status="healthy",
        model_loaded=model is not None
    )

@app.post("/predict", response_model=PredictionResponse)
def predict(input_data: IrisInput):
    """Realiza una predicción para una flor Iris."""
    if model is None:
        raise HTTPException(status_code=503, detail="Modelo no disponible")

    try:
        # Preparar features
        features = np.array([[
            input_data.sepal_length,
            input_data.sepal_width,
            input_data.petal_length,
            input_data.petal_width
        ]])

        # Predecir
        prediction = model.predict(features)[0]
        probabilities = model.predict_proba(features)[0]

        # Mapear clases
        classes = model.classes_
        prob_dict = {str(c): float(p) for c, p in zip(classes, probabilities)}

        logger.info(f"Predicción: {prediction}, Confianza: {max(probabilities):.2f}")

        return PredictionResponse(
            prediction=str(prediction),
            confidence=float(max(probabilities)),
            probabilities=prob_dict
        )

    except Exception as e:
        logger.error(f"Error en predicción: {e}")
        raise HTTPException(status_code=500, detail=str(e))
```

tests/test_api.py:

```python
"""Tests para la API."""
import pytest
from fastapi.testclient import TestClient
from src.api.main import app

client = TestClient(app)

class TestHealthEndpoint:
    """Tests para /health."""

    def test_health_returns_200(self):
        response = client.get("/health")
        assert response.status_code == 200

    def test_health_response_format(self):
        response = client.get("/health")
        data = response.json()
        assert "status" in data
        assert "model_loaded" in data

class TestPredictEndpoint:
    """Tests para /predict."""

    def test_predict_valid_input(self):
        response = client.post("/predict", json={
            "sepal_length": 5.1,
            "sepal_width": 3.5,
            "petal_length": 1.4,
            "petal_width": 0.2
        })
        assert response.status_code == 200
        data = response.json()
        assert "prediction" in data
        assert "confidence" in data

    def test_predict_invalid_input(self):
        response = client.post("/predict", json={
            "sepal_length": -1,  # Inválido
        })
        assert response.status_code == 422  # Validation error
```

Notas Finales

Las soluciones proporcionadas son ejemplos de referencia. Tu implementación puede variar y seguir siendo correcta.

Lo importante es que: - El código funcione correctamente - Siga las mejores prácticas - Esté documentado - Tenga tests

Ejercicios | Rúbrica | Syllabus