

**Iniciamos**

**6:10 am**

**:::Gracias:::**

**Mentor: Jonnatan Arias**

**Garcia**



# Machine Learning Operations II



by Jonnatan Arias Garcia

12/09/2024

# Contenido I

## Modulo 1: Introducción MLOps

### MLOps

- MLOps y Devs
- Importancia MLOps
- Ciclo de vida MLOps

### Entorno de desarrollo

- Google Colab
- Configuración de Python para MLOps
- Control de Versiones Git y Github

## Modulo 2: Fundamentos de Machine Learning

### ML básico

- Aprendizaje Supervisado – No Supervisado
- Evaluación de Modelos: Accuracy, precisión recall, F1 score

### Gestión de Datasets

- Preprocesamiento
- Librerías Pandas y Numpy

## Módulo 3: Automatización de procesos

### Pipeline de ML

- Definición y usos
- Pipelines en Scikit-learn
- Implementación en Colab

### Control de Versiones de datos

- DVC colab
- Versionado de Datasets.

## Módulo 4. Modelos en Producción

### Modelos Reproducibles

- Reproducibilidad y Semilla
- Tracking con MLflow
- Mlflow en colab

### Modelos y entrenamiento

- Modelos con tensorflow y keras
- Guardado y Exportación de modelo entrenado

# Contenido II

## Módulo 5. Pruebas y Monitoreo de Modelos

### Pruebas

- Validación Cruzada
- Pruebas unitarias y de integración de pipelines en ML
- Pytest

### Monitoreo

- Introducción al monitoreo de modelos
- Herramientas (Prometheus, Grafana)
- Implementación de Alertas Básicas

## Módulo 6. Gestión de Recursos

### Optimización de Recursos

- CPU, GPU, TPU
- Uso eficiente y paralelaje

### Optimización de Parámetros

- GridSearch y RandomSearch

## Módulo 7. Despliegue en la Nube

### Google Cloud AI

- Integración de Colab con Google Cloud
- Despliegue en Google Cloud AI

### Integración CI/CD

- CI/CD: Integración continua + entrega e implementación continua
- Github Actions

## Módulo 8. Documentación y Buenas Practicas

### Sphinx

- Documentación de pipelines, modelos y librerías

### Buenas Prácticas

- Recomendaciones, Seguridad y privacidad



# Modulo IV

Modelos: semillas, guardado, carga...

# Modelos Reproducibles Reproducibles

La reproducibilidad es un concepto fundamental y se enfoca en que se debe permitir que los resultados de un experimento se repliquen de manera consistente.





# Reproducibilidad y Semillas con keras-tensorflow

## 1 Semilla

Generan secuencias de números pseudoaleatorios. Para inicialización inicialización de pesos, lotes,...

```
import numpy as np
import tensorflow as tf
import random

# Fijar semillas para reproducibilidad
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
```

## 2

## Aleatoriedad en división de datos

Manejo de variación determinista en cuanto a la elección y manipulación de datos como “shuffle”.

```
from sklearn.model_selection import train_test_split

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

## 3

## Inicialización de parámetros

Al utilizar semillas, se controla el comportamiento aleatorio en el ajuste de Hiperparámetros del modelo e inicialización de pesos.

```
def crear_entrenar_modelo(xtr, ytr):
    # Fijar semilla del generador aleatorio
    tf.keras.utils.set_random_seed(123)

    # Crear modelo
    modelo = tf.keras.models.Sequential([
        tf.keras.layers.Flatten(input_shape=(3,1)),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ])

    # Compilarlo
    modelo.compile(optimizer='sgd',
                    loss='BinaryCrossentropy',
                    metrics = ['accuracy'])

    # Entrenarlo sin imprimir iteraciones en pantalla. No validar
    modelo.fit(xtr,ytr,verbose=0,
               epochs=5)

    return modelo
```

# Modelos con TensorFlow y Keras

## TensorFlow

Una biblioteca de código abierto para el desarrollo de modelos de machine learning.

## Keras

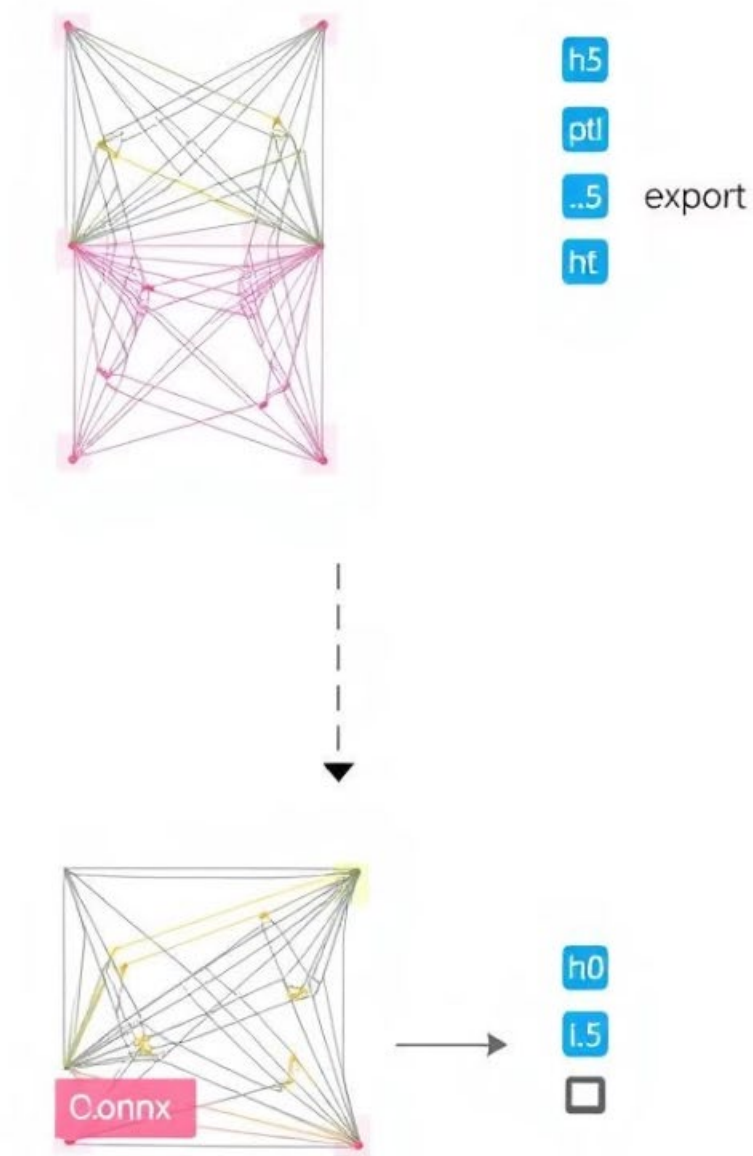
Una API de alto nivel que simplifica la construcción y entrenamiento de modelos con TensorFlow.

## Modelos Reproducibles

Al integrar MLflow, se pueden rastrear los experimentos y garantizar la reproducibilidad.



# Guardado y Exportación de Modelo Entrenado



1

## Guardado del Modelo

Los modelos entrenados se guardan en formato binario o como archivos de texto.

2

## Exportación del Modelo

Se puede exportar el modelo para su uso en otras plataformas o aplicaciones.

3

## Reproducibilidad del Modelo

Al guardar el modelo, se garantiza la reproducibilidad del resultado del entrenamiento.

# Funciones útiles (guardar modelo)

## Guardar Checkpoints

Para guardar el modelo o los pesos del modelo con cierta frecuencia.

Se usa junto al entrenamiento para guardar un modelo o pesos (en un archivo) en algún intervalo, de modo que posteriormente podamos cargarlos para continuar con el entrenamiento desde una punto dado.

```
keras.callbacks.ModelCheckpoint(  
    filepath,  
    monitor="val_loss",  
    verbose=0,  
    save_best_only=False,  
    save_weights_only=False,  
    mode="auto",  
    save_freq="epoch",  
    initial_value_threshold=None,  
)
```

# Funciones útiles (guardar modelo)

Algunas opciones que ofrece esta devolución de llamada incluyen:

- Ya sea para conservar únicamente el modelo que ha logrado el "mejor rendimiento" hasta el momento o para guardar el modelo al final de cada época, independientemente del rendimiento.
- Definición de "mejor"; qué cantidad **monitorear** y si se debe maximizar o minimizar.
- La frecuencia con la que debería guardarse. Actualmente, la devolución de llamada admite guardar al final de cada **época** o después de una cantidad fija de lotes de entrenamiento.
- Si solo se guardan los **pesos** o se guarda **todo** el modelo.
- La opción **mode** 'auto' puede ser max o min dependiendo la métrica, loss='min' accuracy='max'

```
keras.callbacks.ModelCheckpoint(  
    filepath,  
    monitor="val_loss",  
    verbose=0,  
    save_best_only=False,  
    save_weights_only=False,  
    mode="auto",  
    save_freq="epoch",  
    initial_value_threshold=None,  
)
```

# Funciones útiles (guardar modelo)

**Checkpoint:** **Callback** que carga los datos del modelo incluyendo los pesos (en la dirección y archivo 'C:\Mis documento\best\_model.h5')

```
from keras.callbacks import ModelCheckpoint
```

```
model_checkpoint = ModelCheckpoint('best_model.h5', monitor='val_accuracy',  
save_best_only=True)
```

```
model.fit(x_train, y_train, validation_data=(x_val, y_val),  
callbacks=[model_checkpoint])
```

.

```
model = alexnet3d_v2() #alexnet v2

# Compilar el modelo
model.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics=['accuracy'])
model.summary()

# Definir el callback ModelCheckpoint
checkpoint = ModelCheckpoint("alexnetv2_run1.h5", monitor='val_accuracy', save_best_only=True, mode='max', verbose=1)

# Preprocesar los datos de entrenamiento
y_train_categorical = to_categorical(y_train, 2)
y_test_categorical = to_categorical(y_test, 2)

# Entrenar el modelo
history = model.fit(X_train, y_train_categorical,
                  batch_size=10,
                  callbacks=[checkpoint],
                  epochs=100,
                  verbose='auto',
                  validation_split=0.3)
```

# Funciones útiles (Carga modelo)

**Igualmente podemos cargar un modelo pre-entrenado para predecir.**

```
model = create_model()
```

```
keras.models.load_model('best_model.h5')
```

**O solo los pesos**

```
model.load_weights('best_model.h5')
```

**Quiero guardar mi modelo entrenado como un zip**

```
model.save('my_model.keras')
```

**Y si quiero cargarlos**

```
new_model = tf.keras.models.load_model('my_model.keras')
```

.

```
from tensorflow import keras

# Cargar el modelo desde un archivo .h5
loaded_model = keras.models.load_model('alexnetv2_run2.h5')

# Opcional: Cargar únicamente los pesos desde un archivo .h5
# loaded_model = create_model() # Crea una instancia vacía del modelo
loaded_model.load_weights('alexnetv2_run2.h5')
loaded_model.summary()

# Preprocesar los datos de entrenamiento
y_train_categorical = to_categorical(y_train, 2)
y_test_categorical = to_categorical(y_test, 2)
# Evaluate the model on test set
score = loaded_model.evaluate(X_test, y_test_categorical)
# Print test accuracy
print('\n', 'Test accuracy:', score[1])
# Re-evaluate the model
loss, acc = loaded_model.evaluate(X_train, y_train_categorical, verbose=2)
print("Restored model, accuracy: {:.2f}%".format(100 * acc))
```



# Beneficios de la Reproducibilidad

## 1 Confiabilidad

Aumenta la confianza en los resultados del modelo, ya que se pueden reproducir de manera consistente.

## 2 Eficiencia

Reduce el tiempo dedicado a la depuración y la búsqueda de errores, facilitando la reutilización del código.

## 3 Colaboración

Permite que otros investigadores o equipos puedan replicar los resultados y contribuir al desarrollo del modelo.

## 4 Mejora Continua

Facilita la iteración y la mejora del modelo, ya que se puede rastrear el impacto de los cambios.



# MLflow Models

  
TensorFlow  
PYTORCH

  
learn

  
R

  
APACHE  
Spark  
dmlc  
XGBoost

ML  
Frameworks

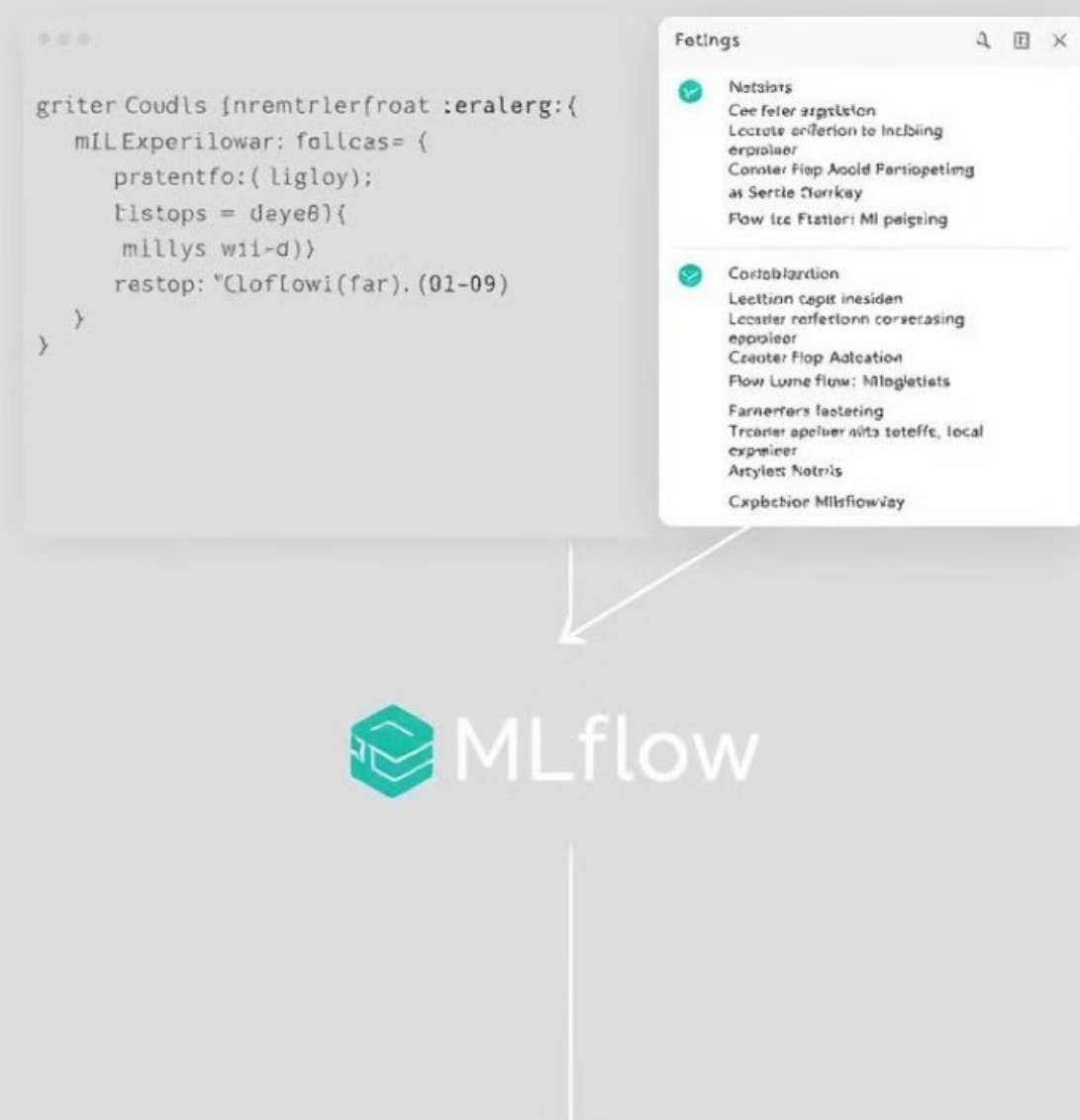


Standard for ML  
models

 docker   
Inference Code

  
APACHE  
Spark  
Batch & Stream  
Scoring

  
Serving Tools



# mlflow

Herramienta para gestión y seguimiento de flujo de trabajo y experimentos

## Integración Fácil

MLflow se integra de manera fluida en Google Colab, simplificando el proceso de seguimiento.

## Flujo de Trabajo Optimizado

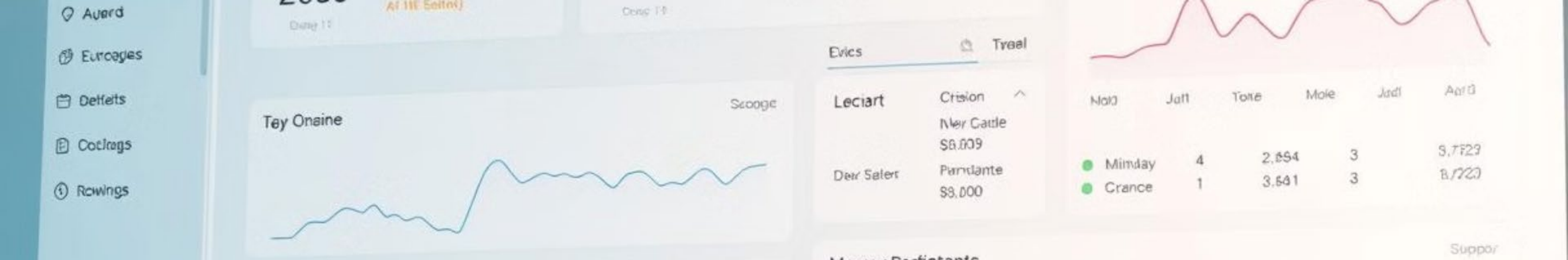
MLflow facilita el seguimiento de experimentos y la gestión de modelos en Google Colab.

## Reproducibilidad en la Nube

Los experimentos se almacenan en la nube, asegurando la reproducibilidad y la colaboración.



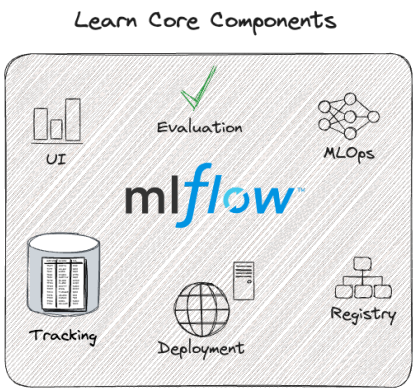
mlflow



# Tracking con MLflow

## Registro de Experimentos

MLflow permite registrar parámetros, métricas y artefactos de los experimentos.



## Model Deployment

Despliegue de modelos con APIs para AWS



## Registro del Modelo

Manejo de Logs. Y modelos de machine learning

```

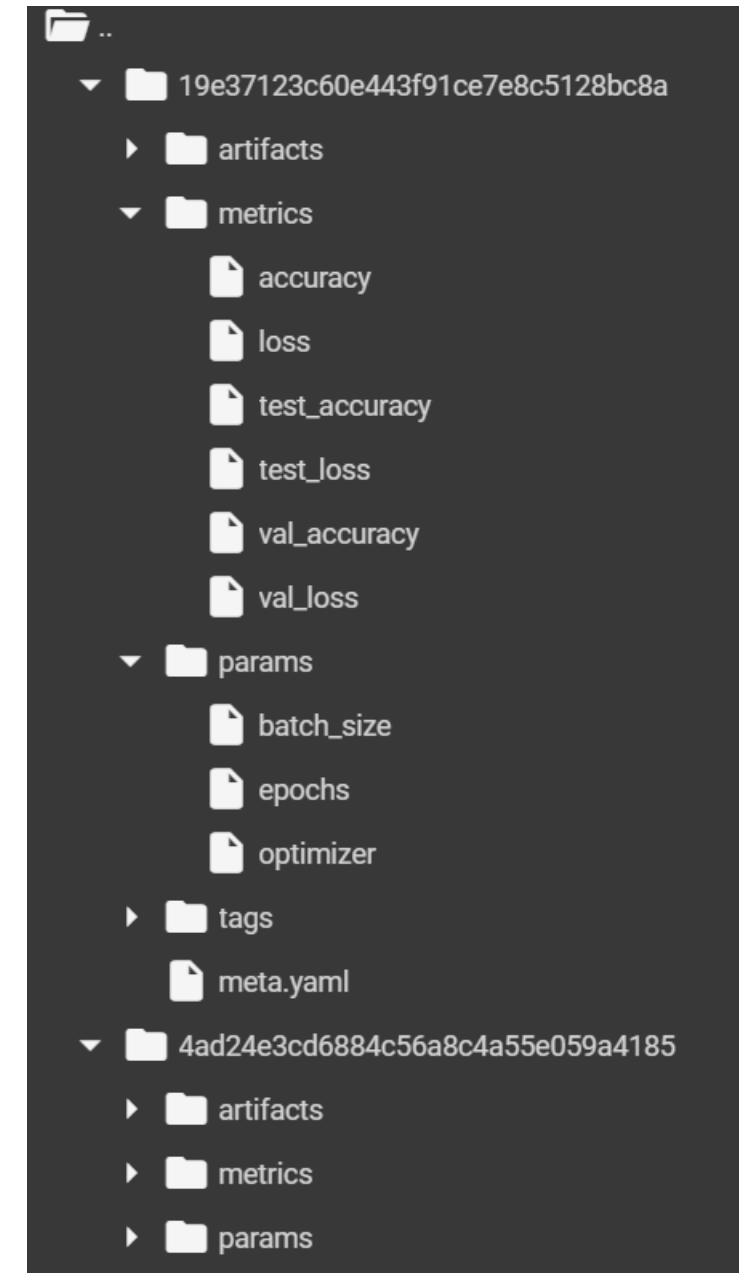
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Iniciar el seguimiento del experimento en MLflow
with mlflow.start_run():
    # Registrar hiperparámetros
    mlflow.log_param("optimizer", "adam")
    mlflow.log_param("batch_size", 32)
    mlflow.log_param("epochs", 5)
    # Entrenar el modelo
    history = model.fit(X_train, y_train, batch_size=32, epochs=5, validation_split=0.2)
    # Registrar métricas
    for epoch in range(5):
        mlflow.log_metric("loss", history.history['loss'][epoch], step=epoch)
        mlflow.log_metric("val_loss", history.history['val_loss'][epoch], step=epoch)
        mlflow.log_metric("accuracy", history.history['accuracy'][epoch], step=epoch)
        mlflow.log_metric("val_accuracy", history.history['val_accuracy'][epoch], step=epoch)

    # Evaluar el modelo en los datos de prueba
    test_loss, test_accuracy = model.evaluate(X_test, y_test)
    # Registrar las métricas de prueba
    mlflow.log_metric("test_loss", test_loss)
    mlflow.log_metric("test_accuracy", test_accuracy)
    # Guardar el modelo
    mlflow.keras.log_model(model, "mnist_model")

# Finaliza el seguimiento del experimento

```

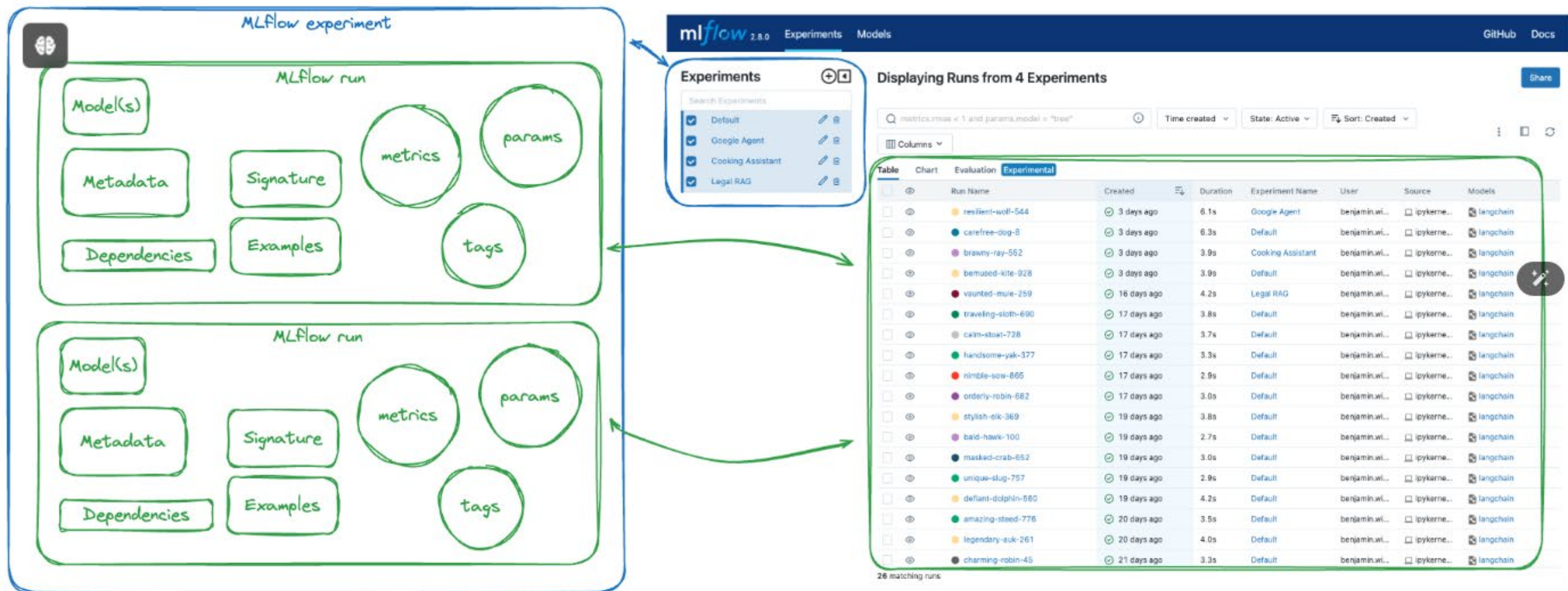


Cargamos en [Inicio, fin] además decimos que datos vamos a registrar toma de datos.

Podemos almacenarlos localmente o en servidor web de pago.

MLflows También tiene gestión automática de API's





In these introductory guides to MLflow Tracking, you will learn how to leverage MLflow to:

- **Log** training statistics (loss, accuracy, etc.) and hyperparameters for a model
- **Log** (save) a model for later retrieval
- **Register** a model using the [MLflow Model Registry](#) to enable deployment
- **Load** the model and use it for inference



# Modulo V

Pruebas y Monitoreo



# Pruebas en Machine Learning

Las pruebas son un componente fundamental del desarrollo de modelos de machine learning. Permiten garantizar la calidad, robustez y confiabilidad de los modelos.

## 1. Conjunto de Entrenamiento (Train Set)

### Para entrenar el modelo

El modelo ajusta sus pesos u otros parámetros internos en función de los datos que se le proporcionan.

Cuanto más grande sea este conjunto, más capacidad tendrá el modelo para aprender patrones generales de los datos.

## 2. Conjunto de Validación (Validation Set)

### Evaluar el modelo durante el entrenamiento

No se utilizan para ajustar los pesos directamente.

Su función es ayudar a ajustar **hiperparámetros** (como el número de capas en una red neuronal, el tipo de optimizador, etc.) y detectar problemas como el **sobreajuste** (overfitting).

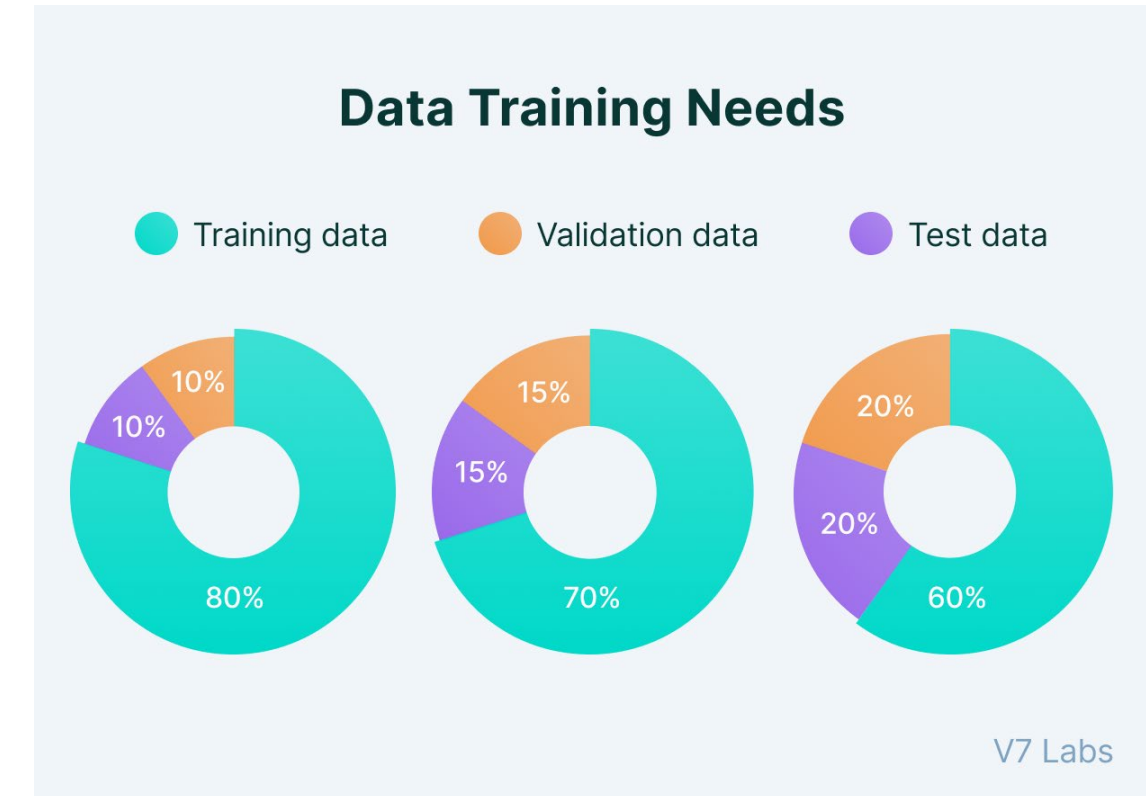
En cada época de entrenamiento, se evalúa el modelo en el conjunto de validación y se monitoriza su rendimiento, lo que permite ajustes.

## 3. Conjunto de Prueba (Test Set)

### Evalúan el rendimiento final del modelo.

Sirve para obtener una estimación objetiva de cómo se comportará el modelo con datos nuevos y no vistos.

Ninguna información de este conjunto debe usarse durante el entrenamiento o la validación para evitar sesgos.



# Validación Cruzada

La validación cruzada es una técnica clave para evaluar el rendimiento de los modelos de machine learning. Divide el conjunto de datos en varios subconjuntos, utiliza un subconjunto para entrenamiento y otro para validación.

## 1 Ventajas

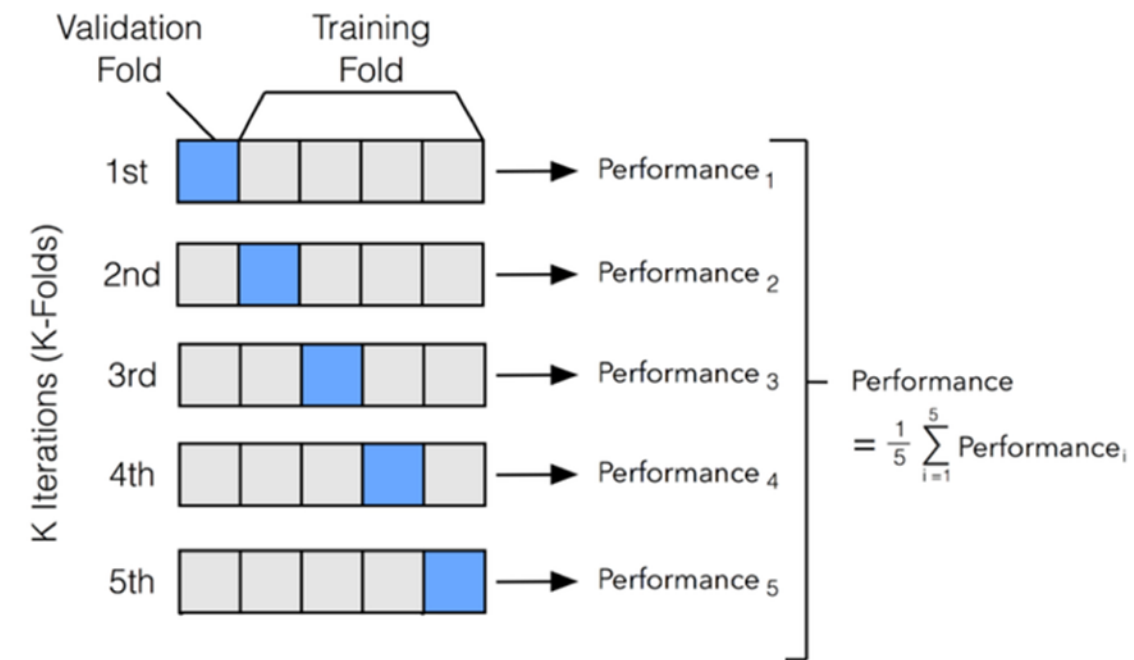
Reduce el sesgo y mejora la generalización del modelo.  
Proporciona una estimación más precisa del rendimiento real.

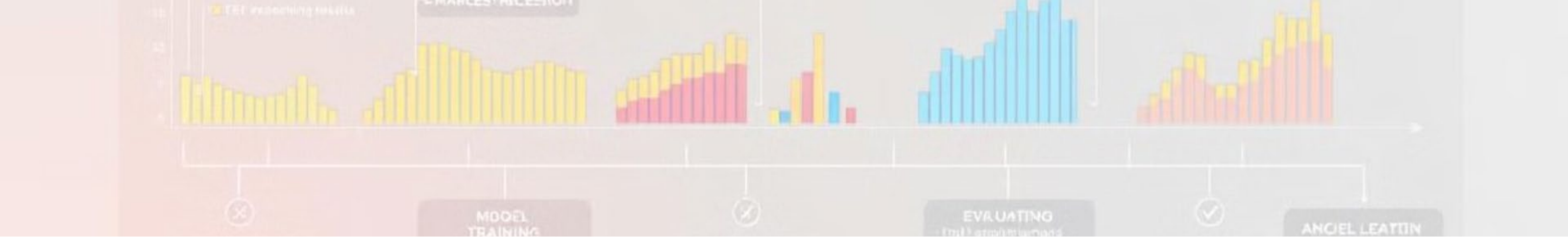
## 2 Tipos

Existen diversos tipos de validación cruzada: k-fold, leave-one-out, etc. La elección del tipo depende del tamaño del conjunto de datos y la complejidad del modelo.

## 3 Consideraciones

Es importante elegir el tipo de validación cruzada adecuado y ajustar los parámetros según el problema.





# Pruebas Unitarias y de Integración en Pipelines de ML

Las pruebas unitarias y de integración son esenciales para garantizar la calidad y confiabilidad de los pipelines de machine learning.

Las pruebas unitarias **verifican el funcionamiento de cada componente individual** del pipeline

Las pruebas de **integración aseguran la interacción correcta** entre los componentes.

## Pruebas Unitarias

Verifican la funcionalidad de cada componente del pipeline, como la carga de datos, el preprocesamiento o el entrenamiento del modelo.

1

2

## Pruebas de Integración

Aseguran el correcto funcionamiento del pipeline en su conjunto, incluyendo la interacción entre los diferentes componentes.

## Beneficios

Detectan errores en etapas tempranas, facilitan la depuración y mejoran la confiabilidad del pipeline.

3

# Pytest

Pytest es un framework de pruebas popular en Python, ideal para pruebas unitarias y de integración en pipelines de ML. Ofrece una sintaxis sencilla, flexible y amigable para escribir pruebas.

## Características

Admite parametrización, fixtures, mocks, y un amplio conjunto de plugins.

1. Parametrización
2. Fixtures
3. Mocks


## Ventajas


Facilita la escritura de pruebas, mejora la legibilidad del código y promueve la reutilización de código.

## Ejemplos

Puedes encontrar ejemplos de pruebas con pytest en la documentación oficial.


```
[1] 1 !pip install pytest
```

 **Mostrar el resultado oculto**

 1 %%writefile test\_fail.py  
2  
3 def func(x):  
4 | | return x + 2 # Error intencional, debería ser x + 1  
5  
6 def test\_answer():  
7 | | assert func(3) == 4 # Esto debería fallar

 **Mostrar el resultado oculto**

```
[5] 1 !pytest test_fail.py
```

 ===== test session starts =====  
platform linux -- Python 3.10.12, pytest-7.4.4, pluggy-1.5.0  
rootdir: /content  
plugins: anyio-3.7.1, typeguard-4.3.0  
collected 1 item

test\_fail.py F [100%]


===== FAILURES =====  
\_\_\_\_\_ test\_answer \_\_\_\_\_

```
def test_answer():  
> assert func(3) == 4 # Esto debería fallar  
E      assert 5 == 4  
E      + where 5 = func(3)
```


test\_fail.py:6: AssertionError

===== short test summary info =====  
FAILED test\_fail.py::test\_answer - assert 5 == 4  
===== 1 failed in 0.09s =====

```
[2] 1 %%writefile test_sample.py  
2  
3 def func(x):  
4 | | return x + 1  
5  
6 def test_answer():  
7 | | assert func(3) == 4
```

 Writing test\_sample.py

```
[3] 1 !pytest test_sample.py
```

 ===== test session starts =====  
platform linux -- Python 3.10.12, pytest-7.4.4, pluggy-1.5.0  
rootdir: /content  
plugins: anyio-3.7.1, typeguard-4.3.0  
collected 1 item

test\_sample.py . [100%]

===== 1 passed in 0.01s =====



# Monitoreo

El monitoreo es crucial para garantizar el rendimiento y la estabilidad de los modelos de machine learning en producción. Permite detectar problemas, identificar áreas de mejora y tomar acciones oportunas.

## Rendimiento

Supervisar métricas clave como la precisión, loss o F1 score.

## Calidad de los Datos

Monitorear la calidad de los datos de entrada para detectar anomalías o cambios inesperados.

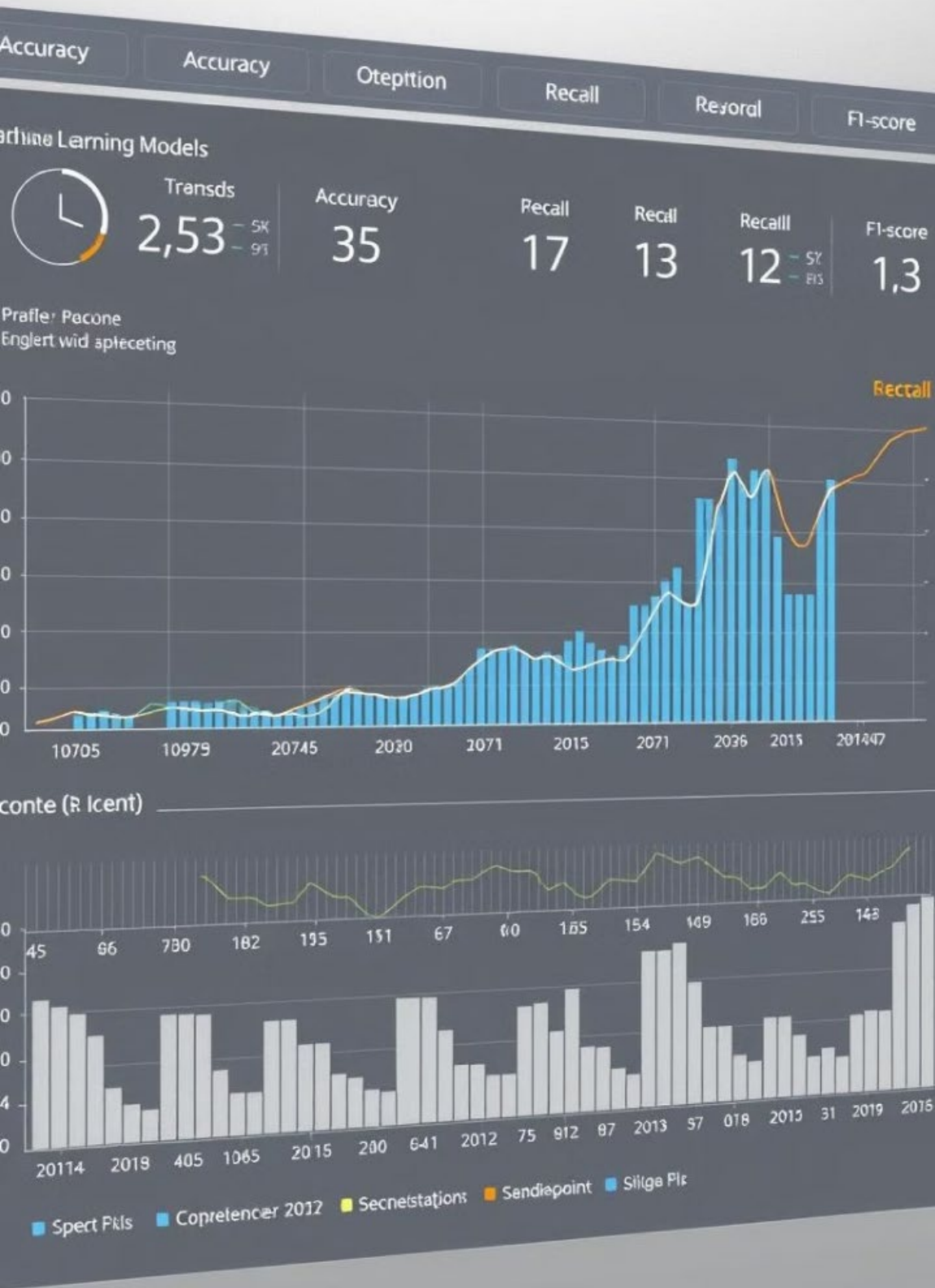
## Drift de Datos

Identificar cambios en la distribución de los datos que podrían afectar el rendimiento del modelo.





# Introducción al Monitoreo de Modelos



1

## Recopilación de Datos

Capturar métricas clave del modelo en tiempo real.

2

## Análisis de Datos

Evaluar las métricas para identificar anomalías o tendencias negativas.

3

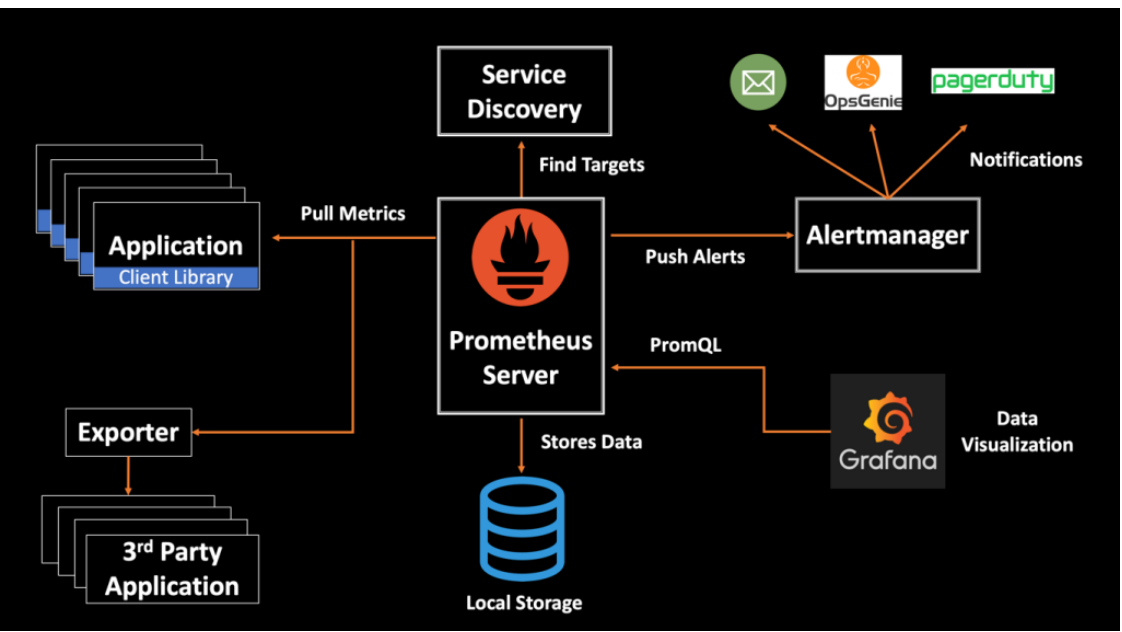
## Generación de Alertas

Enviar alertas cuando las métricas superen umbrales predefinidos.

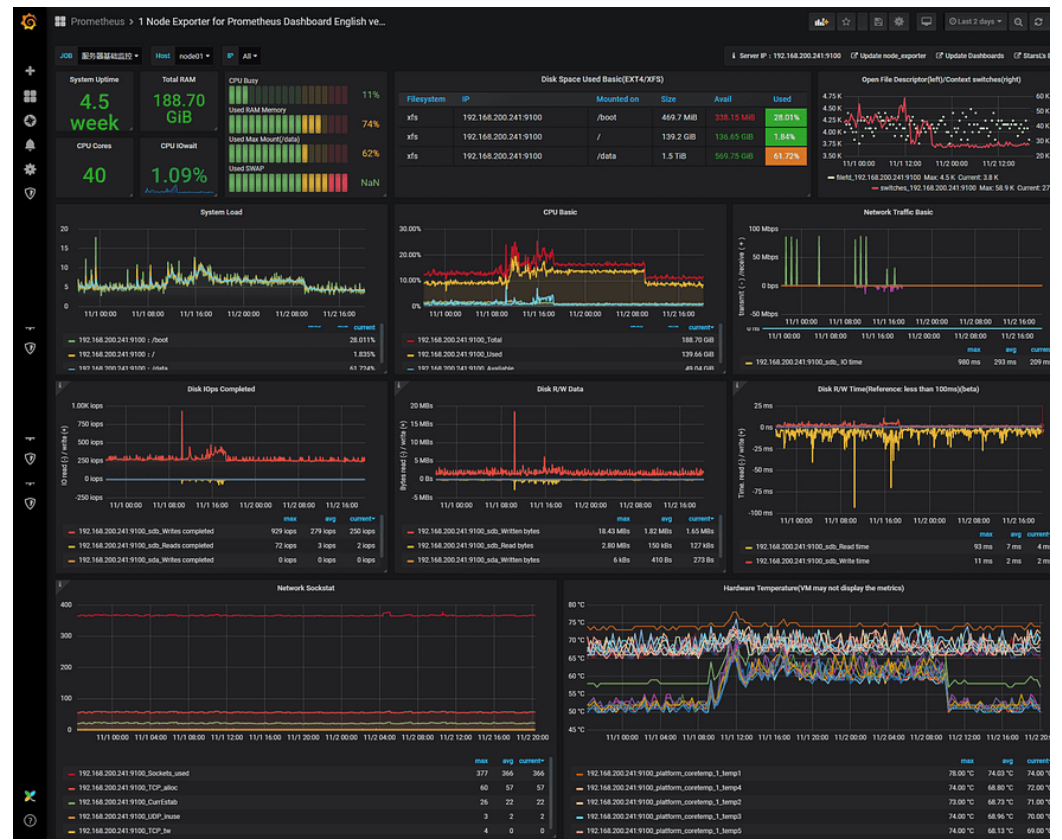
# Herramientas (Prometheus, Grafana)

Prometheus y Grafana son herramientas de código abierto populares para el monitoreo de modelos de Servidores.

Prometheus recopila métricas en tiempo real, mientras que Grafana permite visualizar y analizar los datos recopilados.



Herramienta	Descripción
Prometheus	Sistema de monitoreo de tiempo serie que recopila métricas de aplicaciones y sistemas.
Grafana	Herramienta de visualización y análisis que permite crear dashboards personalizados para monitorear métricas.











Las alertas son cruciales para la detección temprana de problemas. Las alertas básicas se basan en umbrales predefinidos para las métricas del modelo. Cuando las métricas superan los umbrales, se generan alertas que informan a los equipos de operaciones.



## Alertas de Calidad de Datos



Se activan cuando la distribución de los datos de entrada cambia significativamente, lo que puede afectar el rendimiento del modelo.





# Casos de Uso y Mejores Prácticas

El monitoreo de modelos tiene numerosos casos de uso, desde la detección temprana de problemas hasta la optimización del rendimiento del modelo. Las mejores prácticas incluyen la selección de las métricas adecuadas, la configuración de umbrales de alerta realistas y la integración del monitoreo en los procesos de desarrollo y operaciones.

## Detección de Problemas

Identificar problemas como el drift de datos, el deterioro del rendimiento o la aparición de sesgos.

## Optimización del Modelo

Identificar áreas de mejora en el modelo, como la selección de características o la optimización de hiperparámetros.

## Mejora de la Confiabilidad

Asegurar la estabilidad y confiabilidad del modelo en producción.



**Siguientes modulo:**  
**6-8 Viernes**

**Asistencia:**

**<https://tally.so/r/mD5PqR>**