

Título de la formación: MLOps y DataOps

Presentación.

Ángel Delgado Panadero

- Machine Learning Engineer - Paradigma
- Investigador Independiente
- Tutor de Máster
- Físico



Índice.

Viernes

1. Introducción.
2. Problemas comunes.

Preguntas

3. ¿Qué es el MLOps?
4. Pycaret.

Ejercicio de Pycaret
5. MLFlow.

Ejercicio de MLFlow

Sábado

1. Introducción
2. Diseño de Código.

Ejercicio de ML

3. Docker.

Ejercicio de Docker
4. Pipelines con Airflow.

Ejercicio de Pipelines de ML.
5. Pipelines con Sagemaker.

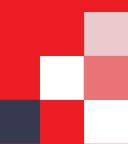
Disclaimer.

- Si la clase es **interactiva** mejor!
- **MLOps** no tiene un standard.
- Cada uno tiene un **background** distinto.

Viernes.



1 - Introducción.



¿Qué es Machine Learning?

¿Qué es Machine Learning?

El Machine Learning es el área de la **Inteligencia Artificial**, que permite a las máquinas “aprender” a partir del procesamiento de datos.

Los procesos de cómputo que permiten realizar este aprendizaje se denominan **entrenamiento**, mientras que los que usan ese aprendizaje para inferir resultados, se denominan **predicción**.



¿En qué se diferencia?

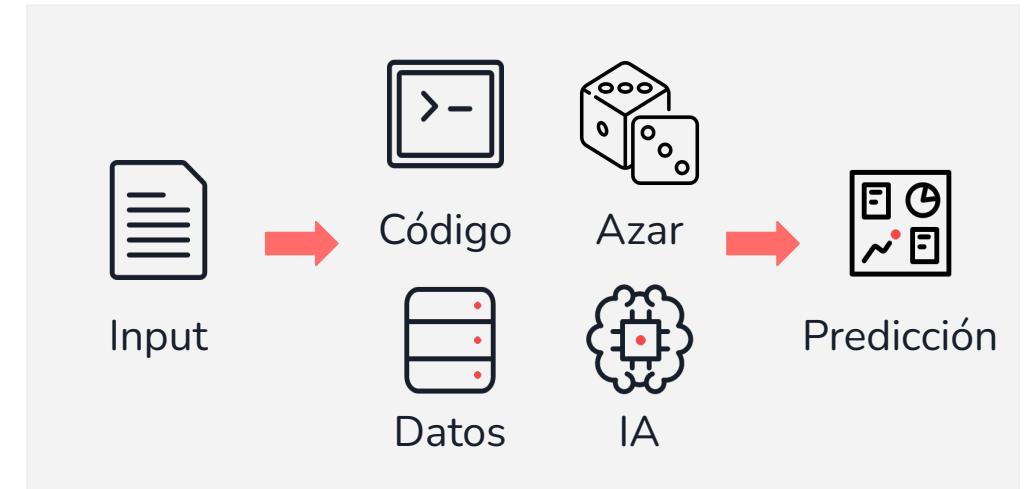
Desarrollo de Software vs Desarrollo de ML

La diferencia entre un proyecto de Machine Learning y uno de desarrollo de software es que hay **más componentes además del código** que es necesario desarrollar.

Desarrollo de Software



Desarrollo de Machine Learning



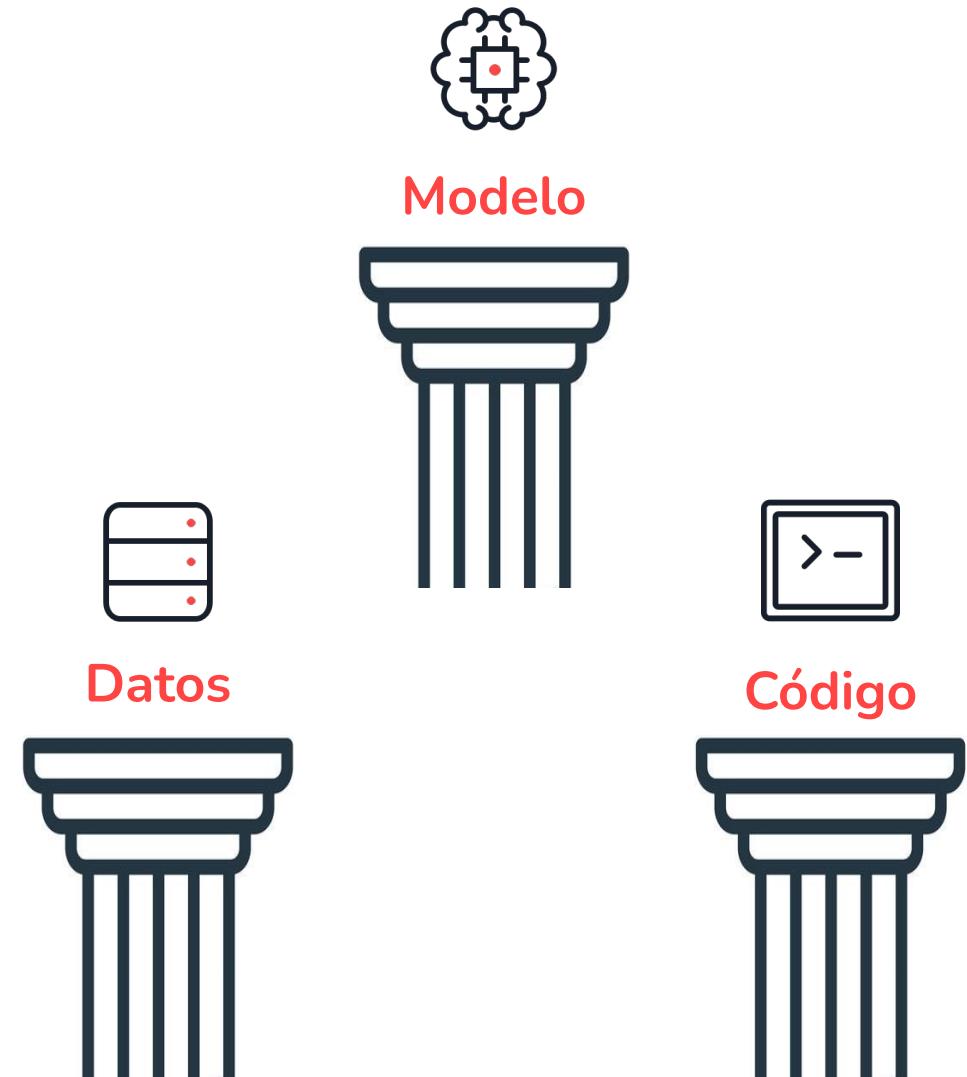
¿De qué componentes está formado?

Componentes y artefactos.

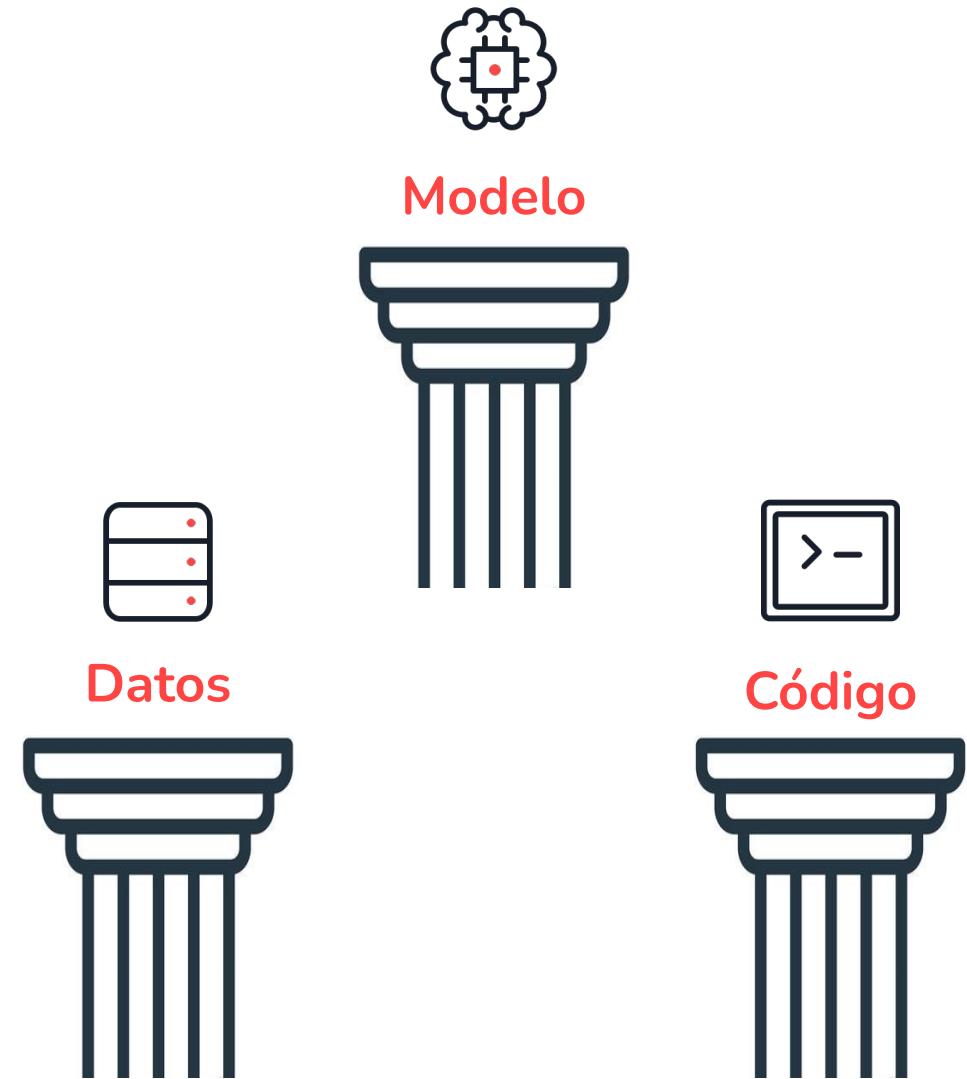
Código: Lógica que contiene el algoritmo de ML así como los procesos de entrenamiento y predicción

Datos: Información que ha utilizado la IA para crearse. A mismo código pero diferentes datos, la IA generada es distinta.

Modelo: Resultado de ejecutar un código de ML sobre unos datos de entrenamiento. (Dos ejecuciones distintas pueden dar dos IA distintas!!!)



Componentes y artefactos.



Componentes y artefactos.



Google Maps



Modelo



Datos



Código



Componentes y artefactos.



Modelo



Datos



Código



¿Cómo se desarrollan estas componentes?

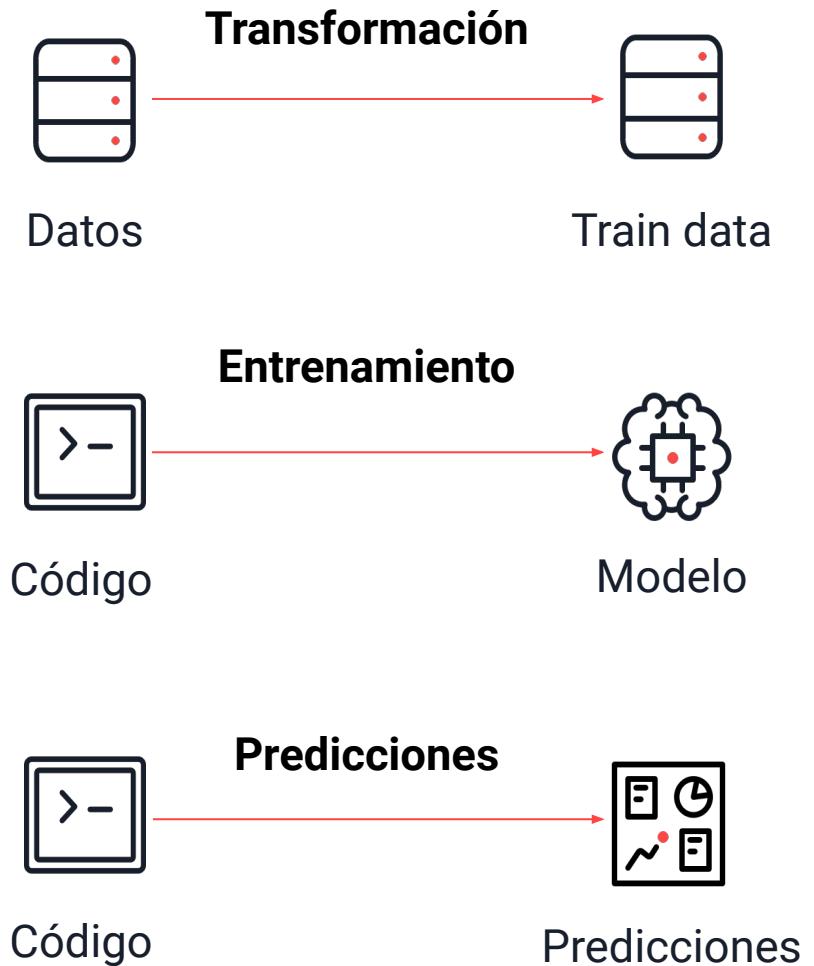
Procesos de ML.

Cada proceso de ML tiene un propósito a la hora de crear todos los componentes. Los principales son:

Transformación: Proceso que prepara los datos para realizar el entrenamiento.

Entrenamiento: Proceso que genera un modelo entrenado a partir de unos datos de entrenamiento.

Predicción: Proceso que usa un modelo entrenado para realizar predicciones sobre nuevos datos.



¿Y cómo se crean estos procesos?

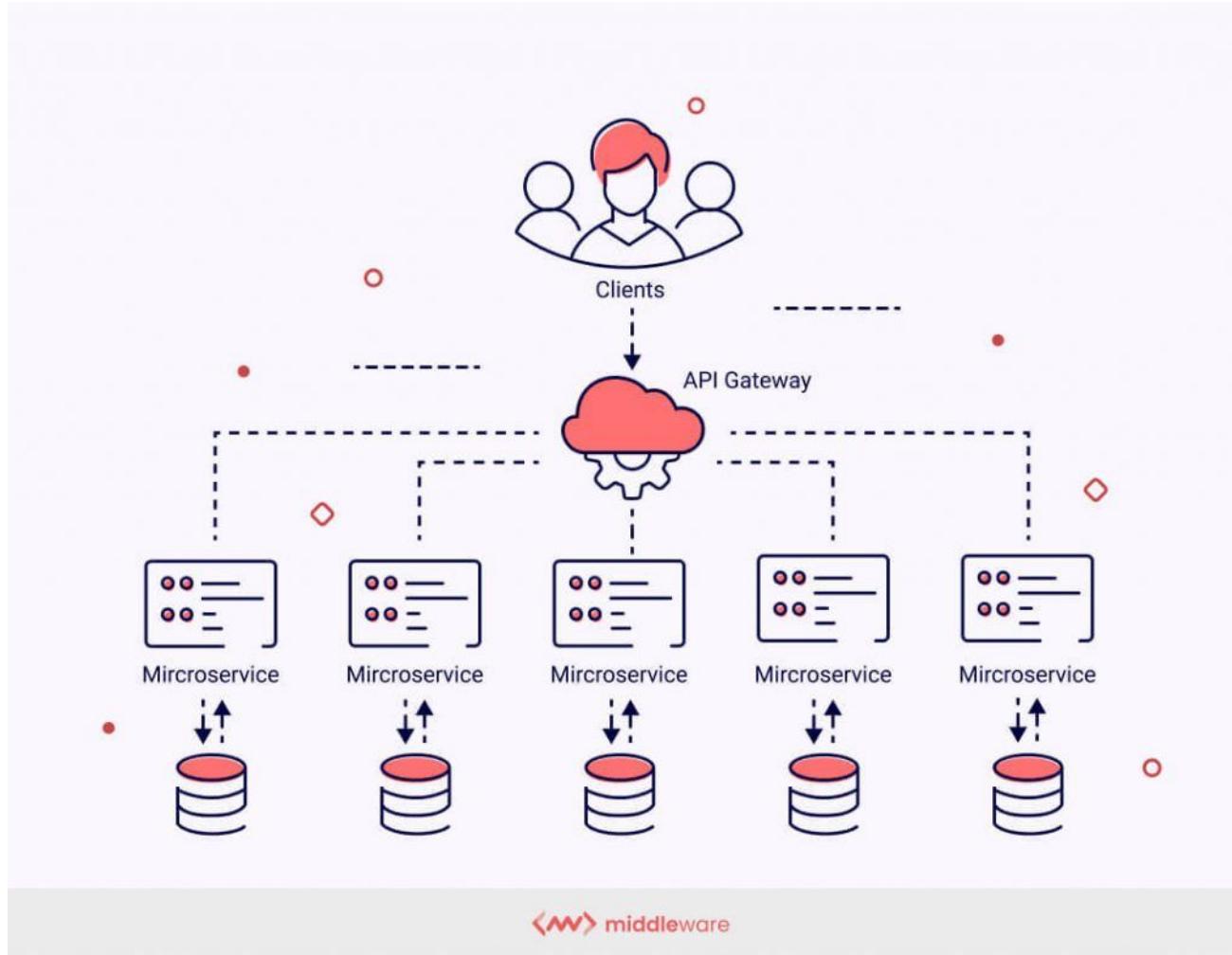
Orquestación de procesos

En arquitecturas de software como **microservicios** tenemos un orquestador que se encarga de gestionar el funcionamiento de todos en conjunto.

En los proyectos de ML necesitamos un **orquestador de procesos** que permita gestionar todos los procesos que se ejecutan.



Orquestación de procesos



Orquestación de procesos



Y además... Hay muchos más procesos!!!

Más procesos de ML

Además de los procesos anteriores, es común que también existan otros procesos que complementen a los anteriores. Existe gran diversidad de ellos, pero algunos son:

Testeo

FeatureStore

AutoML

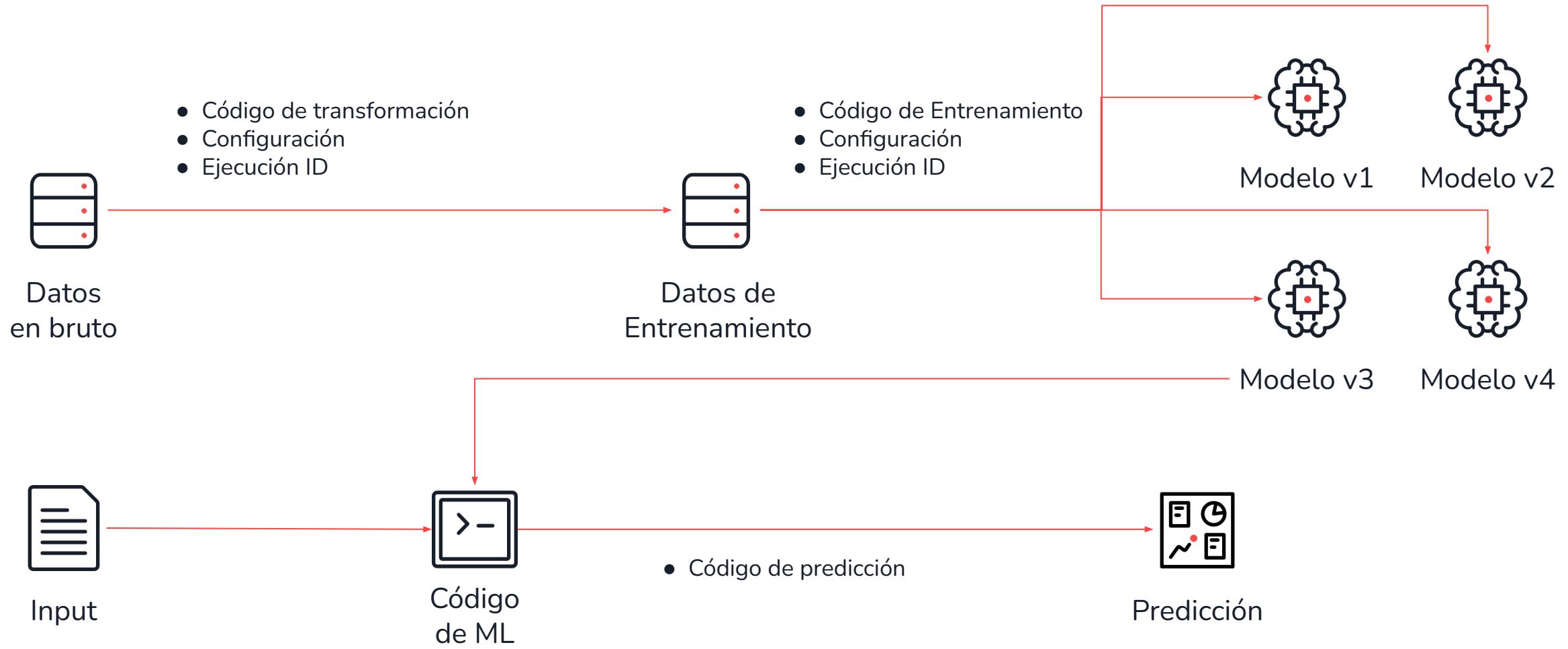
Cross-Validation

A/B Testing

Explicabilidad

Vamos a juntarlo todo

Diagrama de flujo de ML



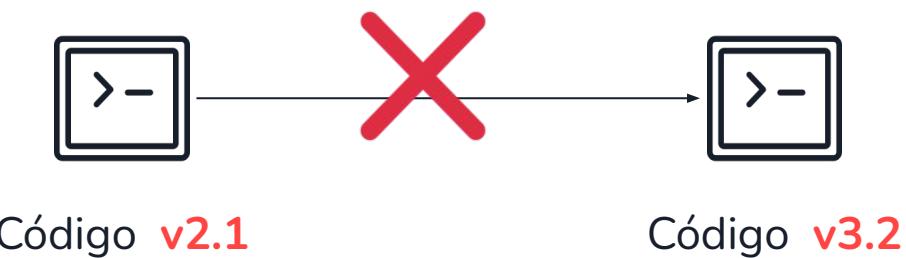
Preguntas?



2 - Problemas comunes.

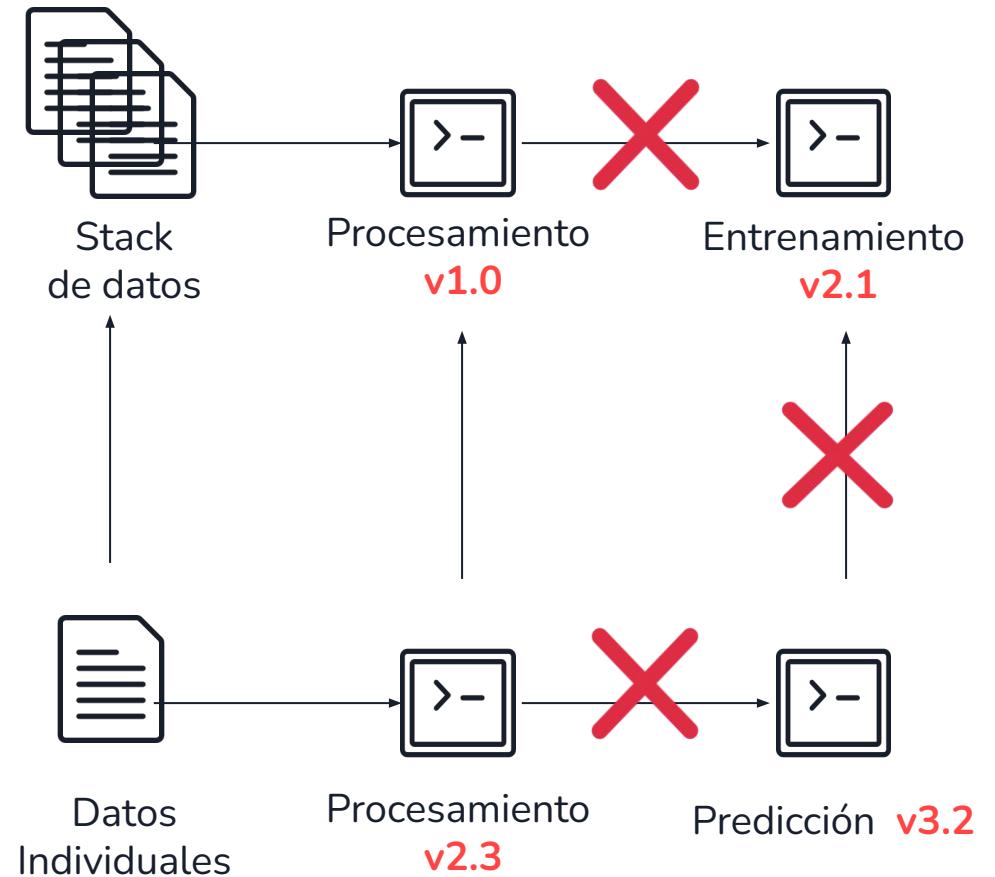


Falta de concordancia entre procesos



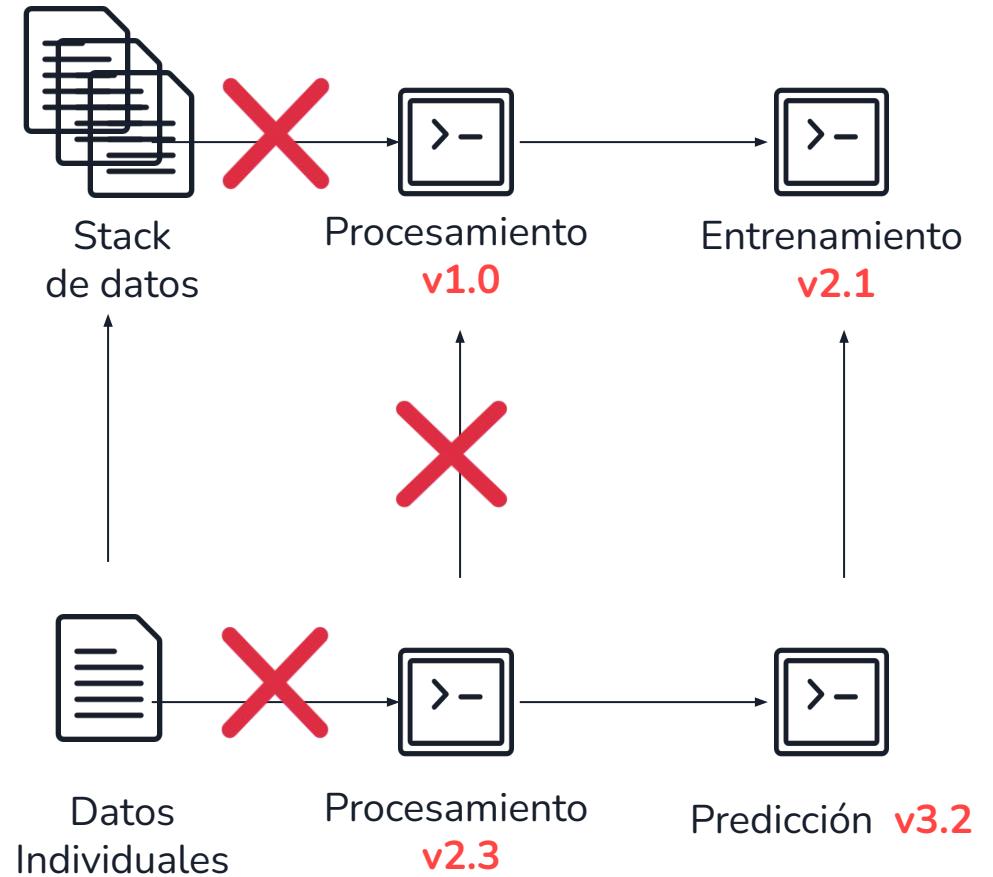
Falta de concordancia entre procesos

- Cambiar el preprocessamiento en entrenamiento y no cambiarlo en predicción.



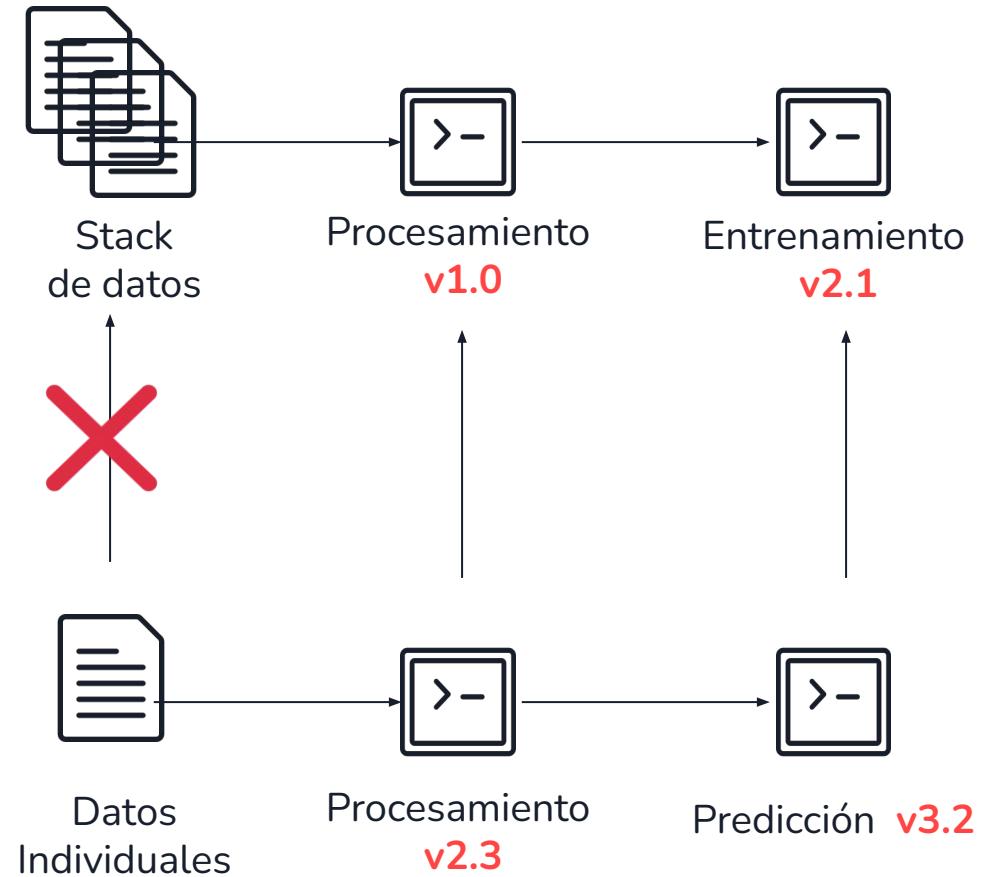
Falta de concordancia entre procesos

- Cambiar el preprocessamiento en entrenamiento y no cambiarlo en predicción.
- Cambiar el esquema de los datos en algunas funciones y en otras no.

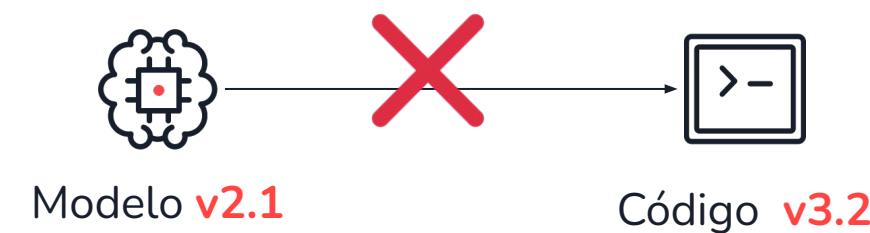


Falta de concordancia entre procesos

- Cambiar el preprocessamiento en entrenamiento y no cambiarlo en predicción.
- Cambiar el esquema de los datos en algunas funciones y en otras no.
- Que la entrada al modelo sea en batch y las predicciones sean atómicas.

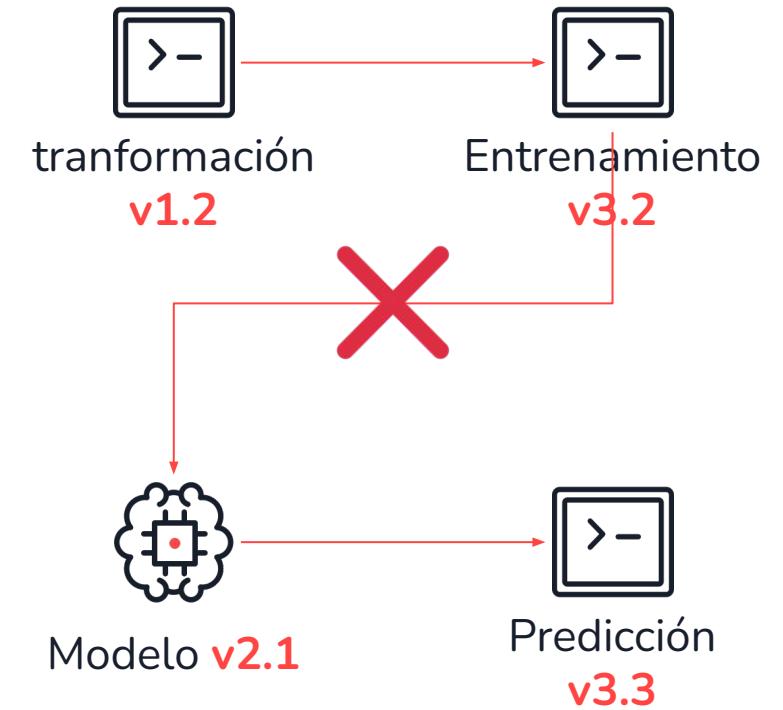


Falta de concordancia entre código y modelos.



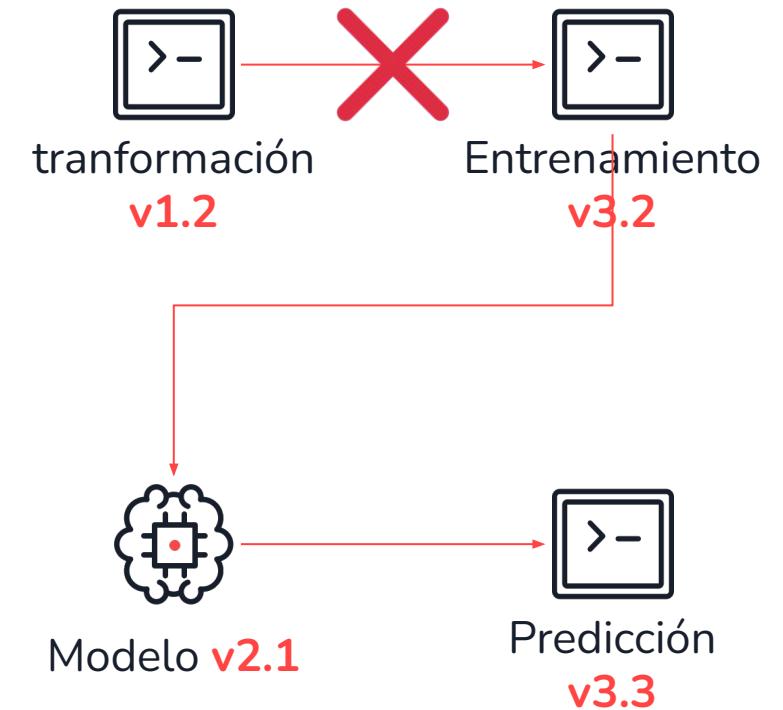
Falta de concordancia entre código y modelos.

- ¿Con qué versión de código se ejecutó el entrenamiento?



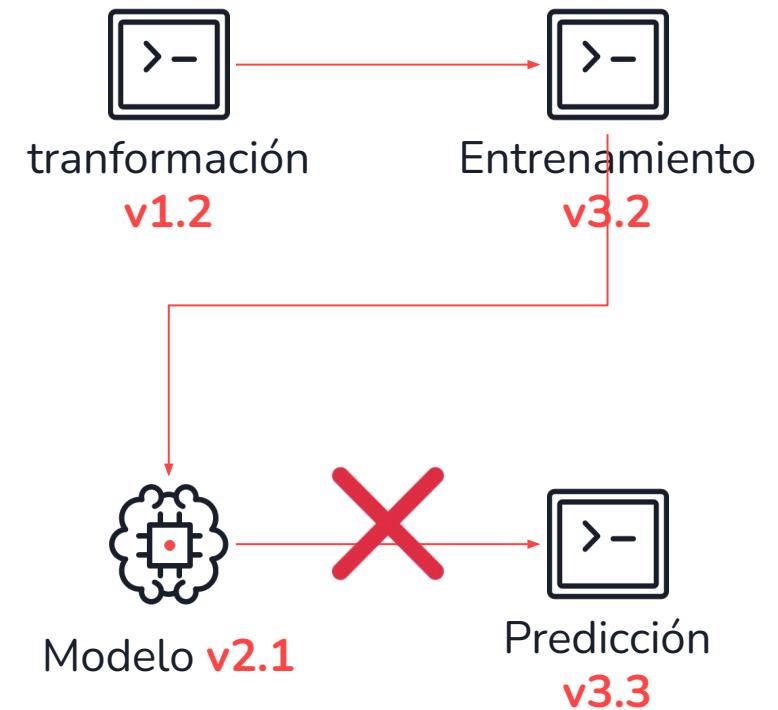
Falta de concordancia entre código y modelos.

- ¿Con qué versión de código se ejecutó el entrenamiento?
- ¿Con qué preprocessamiento se creó ese modelo?



Falta de concordancia entre código y modelos.

- ¿Con qué versión de código se ejecutó el entrenamiento?
- ¿Con qué preprocessamiento se creó ese modelo?
- ¿Con qué código se ejecuta este modelo?

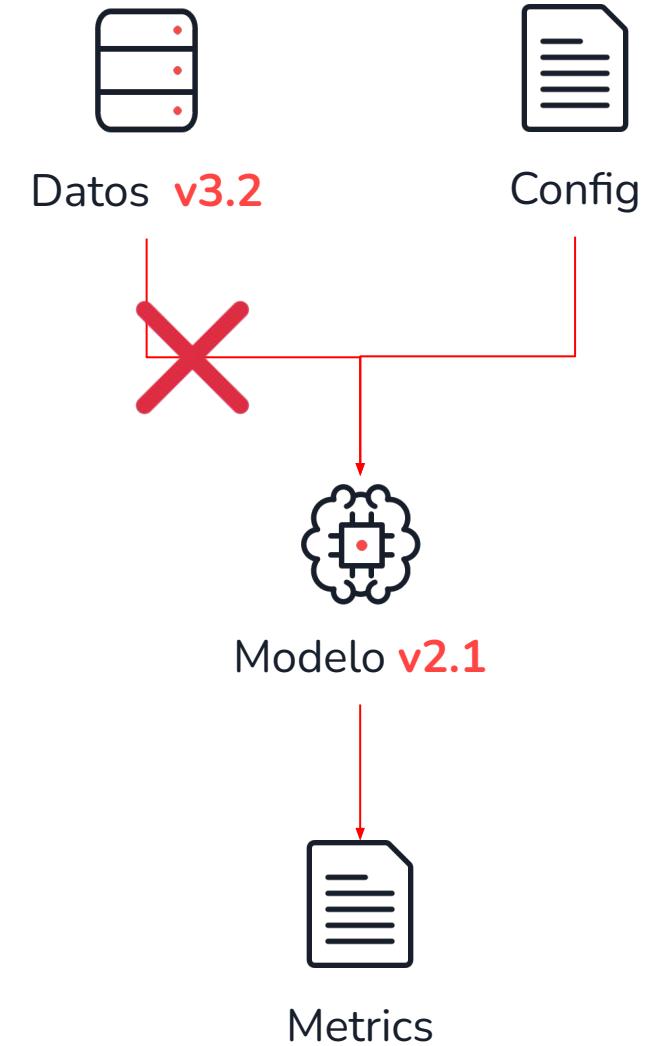


Falta de concordancia entre datos y modelos.



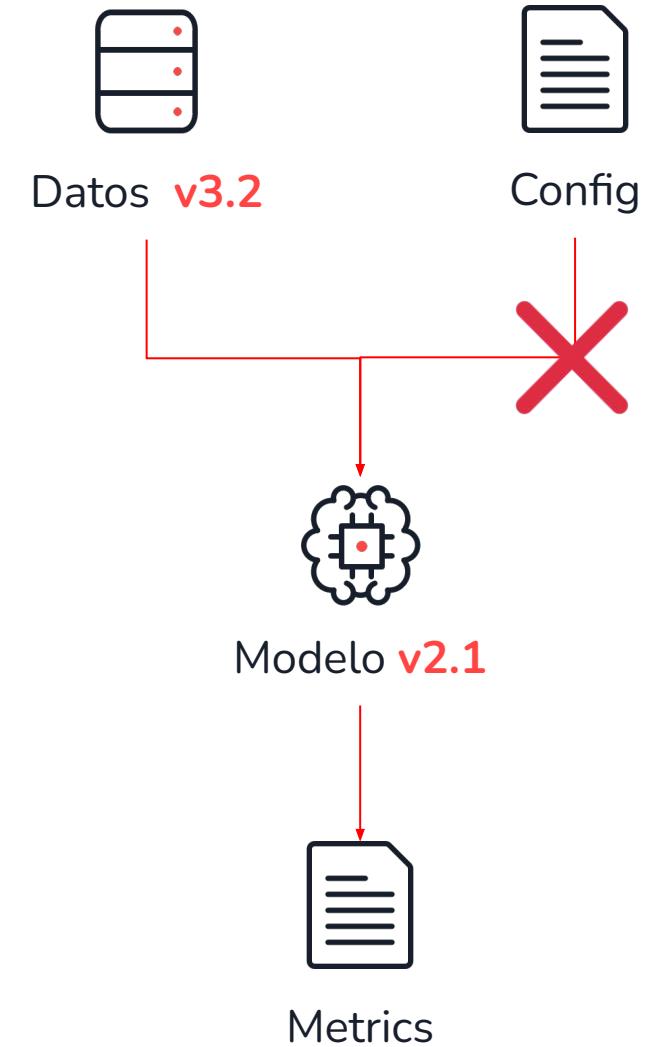
Falta de concordancia entre datos y modelos.

- ¿Con qué datos se entrenó este modelo?



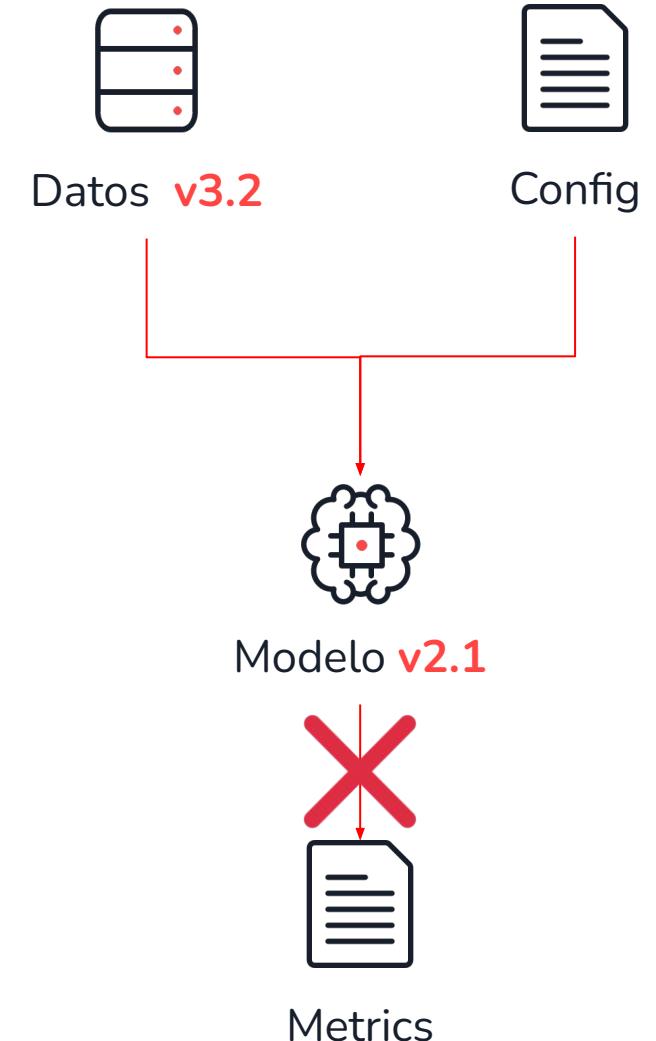
Falta de concordancia entre datos y modelos.

- ¿Con qué datos se entrenó este modelo?
- ¿Con qué configuración se entrenó el modelo?



Falta de concordancia entre datos y modelos.

- ¿Con qué datos se entrenó este modelo?
- ¿Con qué configuración se entrenó el modelo?
- ¿En qué ejecución se entrenó? ¿Qué resultados dió?



¿Cómo se resuelven todos estos problemas?

Falta de concordancia entre código y modelos.

- Definición de un conjunto de buenas prácticas.
- Herramientas que nos permitan versionar y relacionar componentes.
- Herramientas que nos permitan asegurar que los procesos se ejecutan correctamente.

Ejemplos de errores



Ejemplo 1

Ejemplo 1

¿Qué problema tiene este código?

```
import pandas as pd
from sklearn.linear_model import
LogisticRegression

class MyModel():

    def __init__(self):
        self.model = LogisticRegression()

    def preprocess(self, X):
        x_range = X.max(axis=1)-X.min(axis=1)
        x_min = X.min(axis=1)
        return (X-x_min) / x_range

    def fit(self, X, y):
        X = self.preprocess(X)
        model.fit(X, y)
        return self

    def predict(self, X):
        X = self.preprocess(X)
        return model.predict(X)
```

Ejemplo 1

¿Qué problema tiene este código?

Congruencia entre preprocessamiento en entrenamiento y en predicción.

El preprocessamiento no es congruente en el entrenamiento con respecto a la predicción.

Si los datos que llegan en entrenamiento y en predicción son distintos, el preprocessamiento da unos resultados distintos.

```

import pandas as pd
from sklearn.linear_model import
LogisticRegression

class MyModel():

  def __init__(self):
    self.model = LogisticRegression()

  def preprocess(self, X):
    x_range = X.max(axis=1)-X.min(axis=1)
    x_min = X.min(axis=1)
    return (X-x_min) / x_range

  def fit(self, X, y):
    X = self.preprocess(X)
    model.fit(X, y)
    return self

  def predict(self, X):
    X = self.preprocess(X)
    return model.predict(X)
  
```

Ejemplo 2

Ejemplo 2

¿Qué problema tiene este código?

```
import pandas as pd
from sklearn.linear_model import
LogisticRegression
```

```
class MyModel():
```

```
def __init__(self):
    self.model = LogisticRegression(
        l1_ratio = 0.2
    )
```

```
def fit(self, X, y):
    model.fit(X, y)
    score = model.score(X, y)
    return self
```

```
def predict(self, X):
    y = model.predict(X)
    return y
```

Ejemplo 2

¿Qué problema tiene este código?

Congruencia entre modelo y configuración de entrenamiento.

Los parámetros de entrenamiento no deberían estar puestos como literales, sino que tendrían que ser configurables para cada entrenamiento.

Además las métricas de entrenamiento deberían recogerse de alguna forma para que puedan ser almacenadas junto con el modelo.

```
import pandas as pd
from sklearn.linear_model import
LogisticRegression
```

```
class MyModel():

    def __init__(self):
        self.model = LogisticRegression(
            l1_ratio = 0.2
        )
```

```
    def fit(self, X, y):
        model.fit(X, y)
        score = model.score(X, y)
        return self
```

```
    def predict(self, X):
        y = model.predict(X)
        return y
```

Ejemplo 3

Ejemplo 3

¿Qué problema tiene este código?

```
import pickle
from sklearn.linear_model import LogisticRegression
```

```
class MyModel():
```

```
    def __init__(self):
        self.model = LogisticRegression()
```

```
    def fit(self, X, y):
        model.fit(X, y)
        return self
```

```
    def predict(self, X):
        y = model.predict(X)
        return y
```

```
    def save_model(self):
        file = open("model.pkl", "wb")
        pickle.dump(self.model, file)
```

```
    def load_model(self, X):
        file = open("model.pkl", "rb")
        self.model = pickle.load(file)
```

Ejemplo 3

¿Qué problema tiene este código?

Congruencia entre versión de modelos y versión de código.

El nombre del modelo debería ser configurable para que permitiese definir diferentes versiones y fechas de ejecución.

Nunca debería ser un literal estático.

```
import pickle  
from sklearn.linear_model import LogisticRegression
```

```
class MyModel():
```

```
    def __init__(self):  
        self.model = LogisticRegression()
```

```
    def fit(self, X, y):  
        model.fit(X, y)  
        return self
```

```
    def predict(self, X):  
        y = model.predict(X)  
        return y
```

```
    def save_model(self):  
        file = open("model.pkl", "wb")  
        pickle.dump(self.model, file)
```

```
    def load_model(self, X):  
        file = open("model.pkl", "rb")  
        self.model = pickle.load(file)
```

Ejercicio



Ejercicio - Crear un modelo custom

1. Descarga el dataset de California House Pricing de [aquí](#) y crea un experimento de regresión a partir de él usando como target “median_house_value”.
2. Crea una clase en python que se llame “HousePricingModel” y con los métodos:
 - def __init__(self, n_estimators, max_depth)
 - def fit (X, y)
 - def predict(X)
 - def save(model_name)
 - def load(model_name)
3. La clase HousePricingModel deberá servir como interfaz para preprocesar las variables categóricas a numéricas y entrenar un RandomForest de Scikit-Learn.

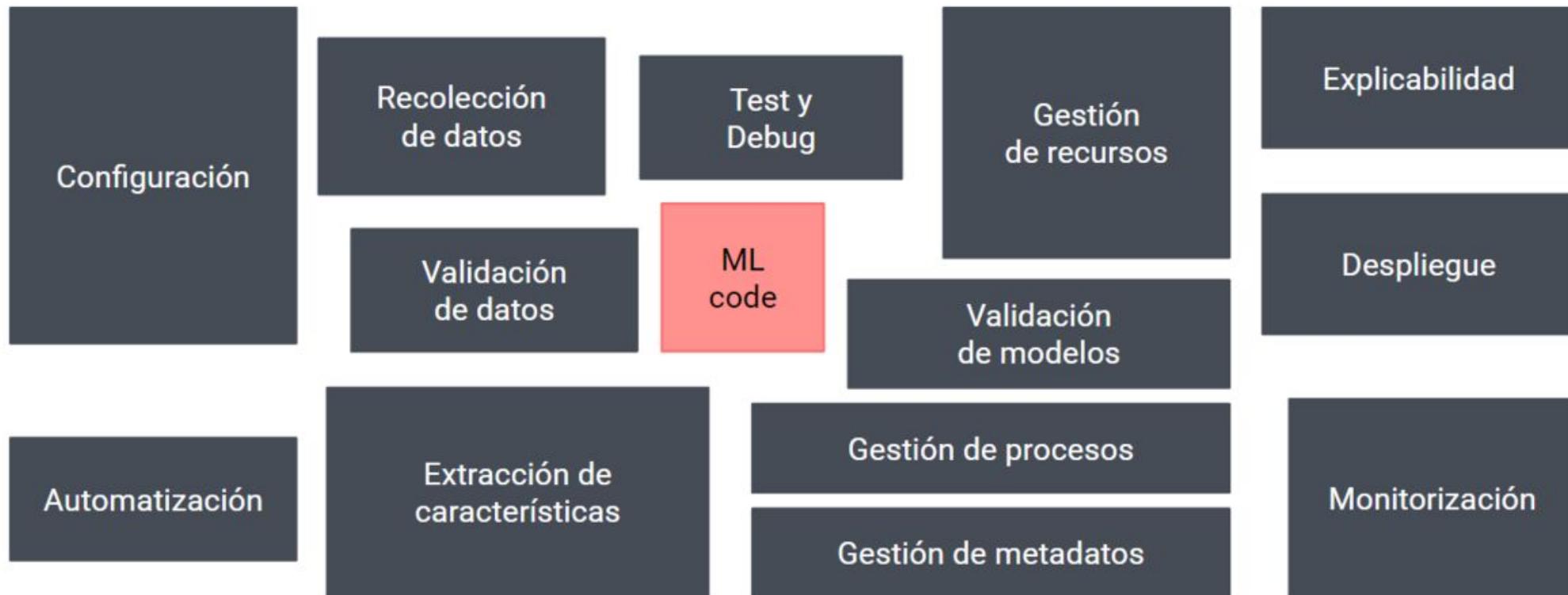
3 - MLOps



¿Qué es el MLOps?

¿Qué es el MLOps?

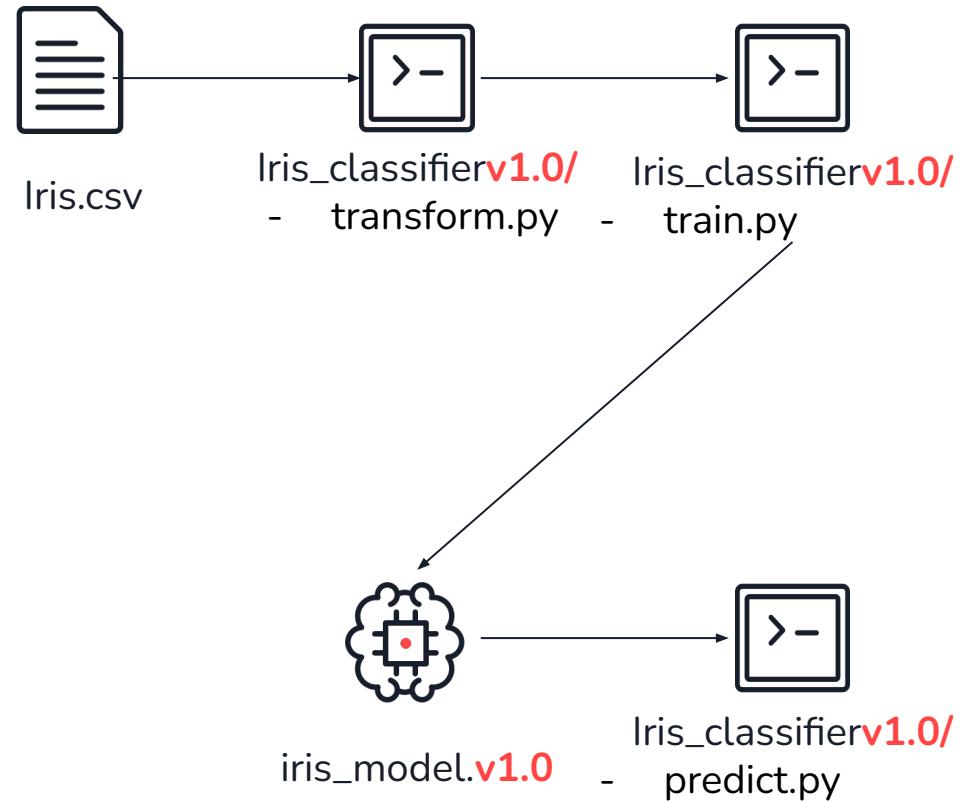
Conjunto de buenas prácticas y herramientas que nos permiten automatizar todos los procesos de un proyecto de ML, así como versionar y gobernar sus componentes.



Buenas prácticas.

Buenas prácticas

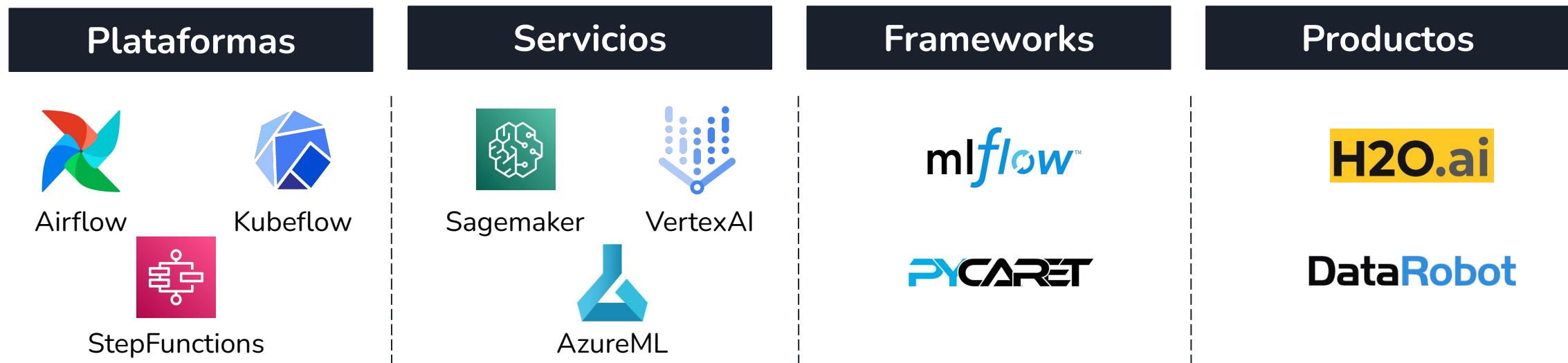
- **Naming:** Definir una política de nombrado que permita relacionar los modelos con códigos y datos.
- **Versiones:** Definir una política de versiones que permita versionar datos, código y ejecuciones de entrenamiento.
- **Flujos:** Definir las secuencias de ejecuciones como un flujo único de procesamiento.



Herramientas.

Herramientas.

Existe una gran diversidad según el grado de automatización y versatilidad que ofrecen.
A grandes rasgos, una posible clasificación es la siguiente:



- Alto nivel de mantenimiento
- Bajo nivel de lock-in

- Bajo nivel de mantenimiento
- Alto nivel de lock-in

Herramientas.

Existe una gran diversidad según el grado de automatización y versatilidad que ofrecen.
A grandes rasgos, una posible clasificación es la siguiente:



- Alto nivel de mantenimiento
- Bajo nivel de lock-in

- Bajo nivel de mantenimiento
- Alto nivel de lock-in

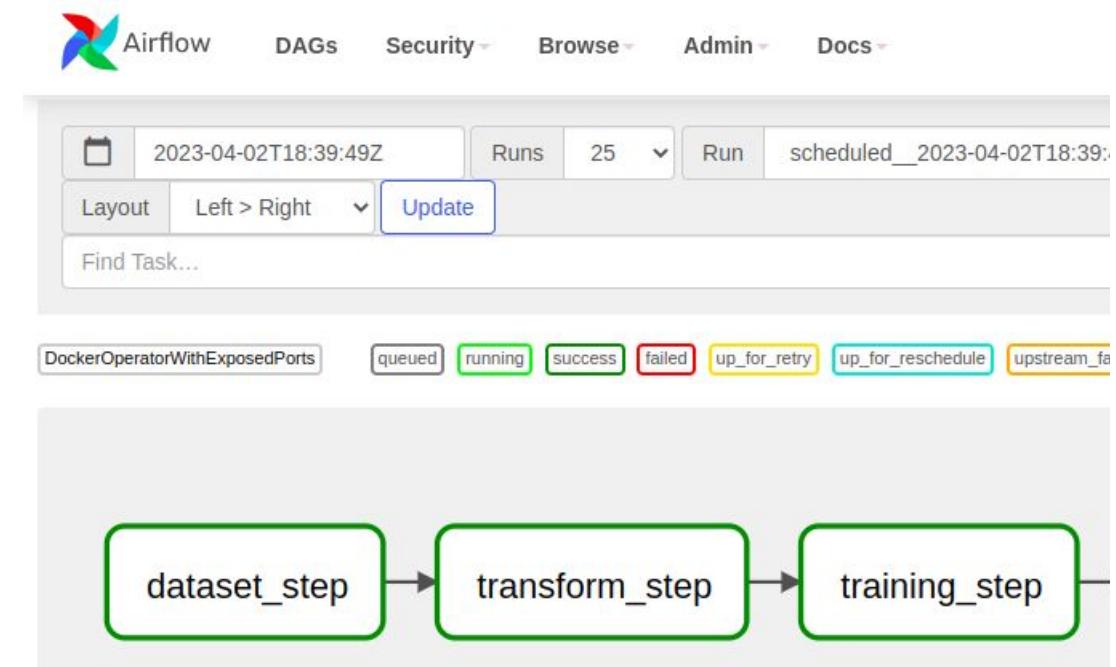
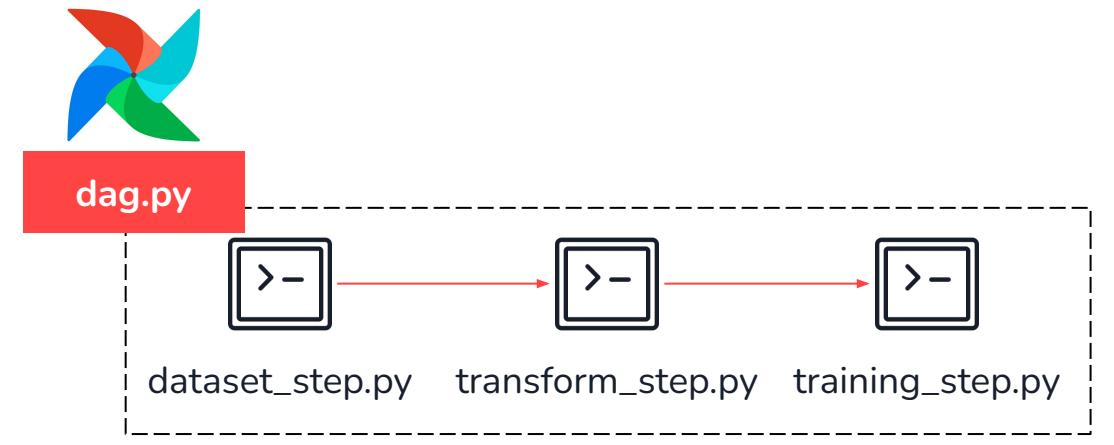
Herramientas basadas en plataforma

Herramientas de plataforma

De propósito general orquestación de procesos. Pero se usan para proyectos de ML.

Ventajas: Pueden adaptarse a cualquier tipo de flujo e integrarse con otras herramientas sin necesidad de modificar el código de Machine Learning.

Inconvenientes: Requieren un nivel de mantenimiento y una curva de aprendizaje más alta que el resto de soluciones.



Herramientas de plataforma

De propósito general orquestación de procesos. Pero se usan para proyectos de ML.

Ventajas: Pueden adaptarse a cualquier tipo de flujo e integrarse con otras herramientas sin necesidad de modificar el código de Machine Learning.

Inconvenientes: Requieren un nivel de mantenimiento y una curva de aprendizaje más alta que el resto de soluciones.

```
from airflow import DAG  
from airflow.decorators import task  
from airflow.operators import PythonOperator
```

```
@task(task_id="train", requirements=["sklearn"])  
def train(X, y):  
    import pickle  
    from sklearn.tree import DecisionTreeClassifier  
    model = DecisionTreeClassifier().fit(X,y)  
    pickle.dump(model, open("model.pkl", "wb"))
```

```
@task(task_id="predict", dependencies=["sklearn"])  
def predict(X):  
    import pickle  
    model = pickle.load(open("model.pkl", "rb"))  
    return model.predict(X)
```

```
with DAG("mlops_dag") as dag:
```

```
train = PythonOperator(task_id="train")  
predict = PythonOperator(task_id='deploy_model')
```

```
train >> predict
```

Herramientas basadas en servicios

Servicios: Sagemaker, VertexAI, AzureML

Los cloud vendors, ofrecen herramientas de orquestación específicas para procesos de Machine Learning.

Ventajas: Pueden adaptarse a la mayoría de Machine Learning de manera más sencilla que las herramientas de orquestación general.

Inconvenientes: Al ser servicios específicos de ML requieren un nivel más bajo de configuración que las plataformas de carácter general.

Sagemaker



Datos



Lambda



Modelos



Logs

Otros recursos de AWS

Servicios: Sagemaker, VertexAI, AzureML

Los cloud vendors, ofrecen herramientas de orquestación específicas para procesos de Machine Learning.

Ventajas: Pueden adaptarse a la mayoría de Machine Learning de manera más sencilla que las herramientas de orquestación general.

Inconvenientes: Al ser servicios específicos de ML requieren un nivel más bajo de configuración que las plataformas de carácter general.

```
import sagemaker
from sagemaker.sklearn.estimator import SKLearn

role = sagemaker.get_execution_role()
session = sagemaker.Session()

sklearn = SKLearn(
    source_dir='./src',
    entry_point='train.py',
    framework_version='0.23-1',
    instance_type="ml.c4.xlarge",
    role=role,
    sagemaker_session=session,
    hyperparameters={
        "min_leaf_nodes": 3,
        "n_estimators": 10,
        "target": "Species"
    }
)
sklearn.fit(inputs={"train": s3_path})

predictor = sklearn.deploy(
    initial_instance_count=1,
    instance_type="ml.m5.xlarge"
)
```

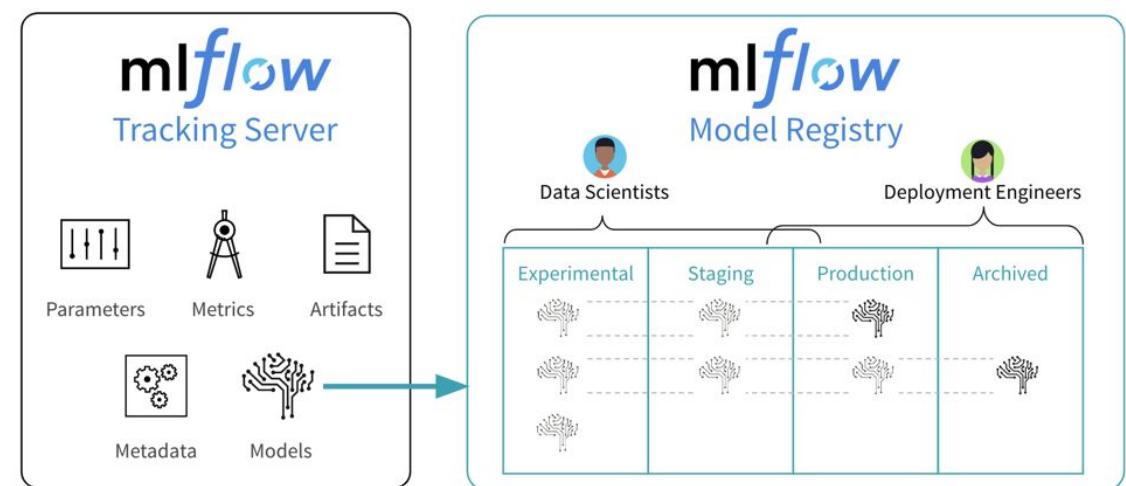
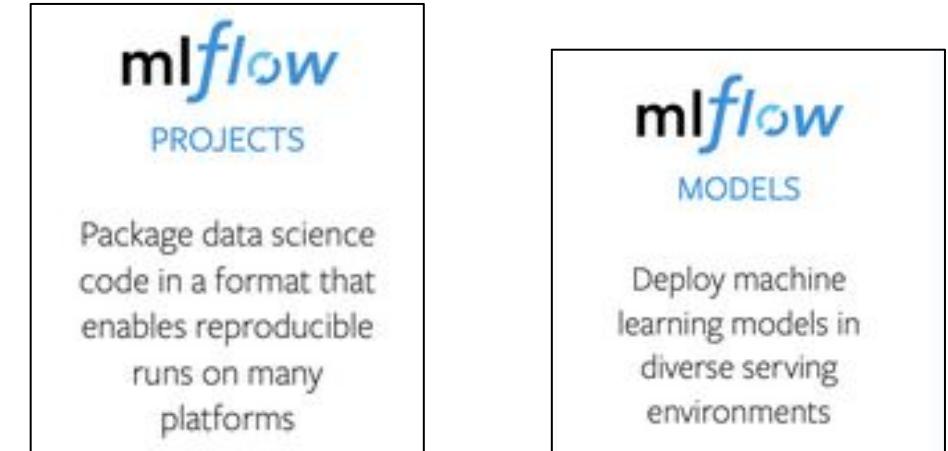
Herramientas basadas en frameworks

Frameworks: MLFlow PyCaret

Una alternativa a las plataformas de orquestación son usar frameworks de ML que permiten automatizar y trazar los procesos de ML.

Ventajas: Requieren una curva de aprendizaje muy baja y no requieren de una persona que configure y gestione el sistema de MLOps como en el caso de los servicios.

Inconvenientes: Hacen que el código de Machine Learning sea dependiente del código de framework que se ha utilizado.



Frameworks: MLFlow Pycaret

Una alternativa a las plataformas de orquestación son usar frameworks de ML que permiten automatizar y trazar los procesos de ML.

Ventajas: Requieren una curva de aprendizaje muy baja y no requieren de una persona que configure y gestione el sistema de MLOps como en el caso de los servicios.

Inconvenientes: Hacen que el código de Machine Learning sea dependiente del código de framework que se ha utilizado.

```
import mlflow
import pandas as pd
from sklearn.linear_model import LinearRegression
```

```
l1_ratio = 0.1
```

```
df_train = pd.read_csv(args.data_path)
X_train = df_train.drop(["Species"], axis=1)
y_train = df_train["Species"]
```

```
model = LinearRegression(l1_ratio=l1_ratio)
model.fit(X_train, y_train)
```

```
score = model.score(X_train, y_train)
```

```
mlflow.log_param('l1_ratio', l1_ratio)
```

```
mlflow.log_metric('score', score)
```

```
mlflow.sklearn.log_model(model, 'model')
```

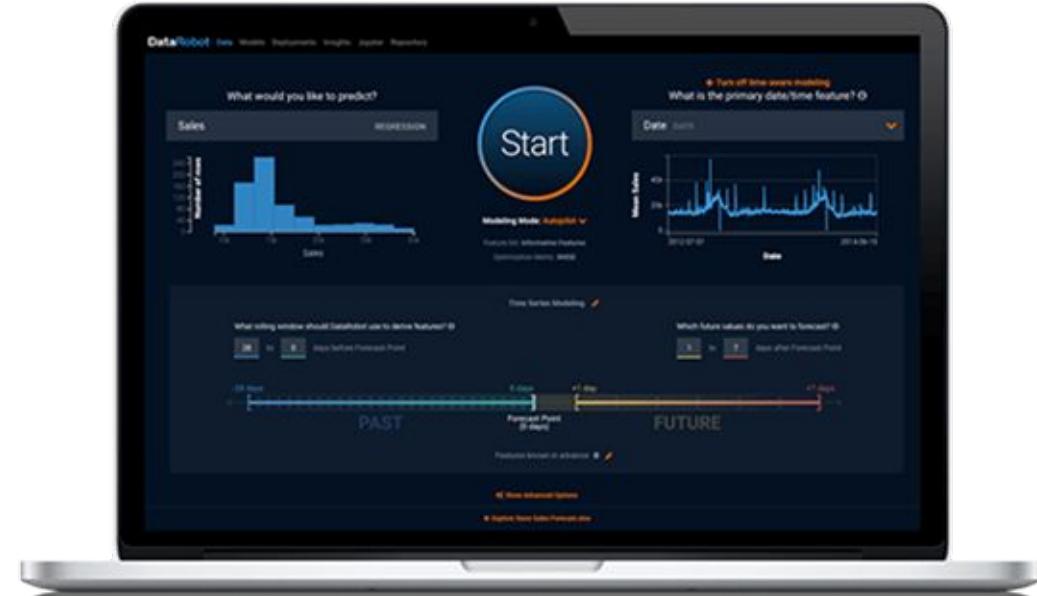
Herramientas basadas en productos

Productos: H2O, DataRobot

Similar a los frameworks solo que además también suelen proveer su propia plataforma de ejecución.

Ventajas: Requiere poco conocimiento de ML y de plataforma para desarrollar un flujo de Machine Learning.

Inconvenientes: La solución depende completamente del producto. Generalmente ni el código de ML ni los flujos de MLOps se pueden migrar a otra plataforma o producto.



¿Cuándo elegir cada una?

Cuándo elegir cada herramienta.

Plataformas

Si queremos una herramienta que sea suficientemente **versatil** como para adaptarse a cualquier tipo de flujo.

Servicios

Si queremos que el flujo tenga **integración** fácil con otros servicios (p.e. bases de datos, endpoints...).

Frameworks

Si queremos una herramienta **específica** de MLOps menos versatil que una plataforma pero más sencilla.

Productos

Si queremos una solución **low-code** que nos permita simplificar todo el desarrollo.

Preguntas



4 - Pycaret.



¿Qué es Pycaret?

Qué es Pycaret

Pycaret es una herramienta **low-code** de Machine Learning que permite crear, desplegar y monitorizar modelos por medio de un framework a alto nivel.



Data
Preparation



Model
Training



Hyperparameter
Tuning



Analysis &
Interpretability



Model
Selection



Experiment
Logging

Instalación.

Instalación.

Pycaret está compuesto por diferentes submódulos para diferentes funcionalidades.

vanilla installation

\$ pip install pycaret

install analysis extras

\$ pip install pycaret[analysis]

models extras

\$ pip install pycaret[models]

install tuner extras

\$ pip install pycaret[tuner]

install mlops extras

\$ pip install pycaret[mlops]

install parallel extras

\$ pip install pycaret[parallel]

install test extras

\$ pip install pycaret[test]

Experimentos de Pycaret.

Experimentos.

En Pycaret, el objeto principal son los **experimentos**. Un experimento entrena múltiples modelos en paralelo. Contiene la siguiente información:

- experiment.dataset
- experiment.models()
- experiment.get_logs()
- experiment.get_metrics()

```
from pycaret.datasets import get_data
from pycaret.classification import
ClassificationExperiment
```

```
data = get_data('diabetes')
```

```
experiment = ClassificationExperiment()
```

Ejecutamos el experimento

```
experiment.setup(
    data,
    target = 'Class variable',
    session_id = 123)
```

Configuracion del experimento

```
print(experiment.dataset)
print(experiment.models())
print(experiment.get_logs())
print(experiment.get_metrics())
```

Extraemos el mejor modelo de todos

```
best_model = experiment.compare_models()
```

Modelos de Pycaret.

Modelos.

Los modelos de PyCaret son el objeto principal que permite realizar predicciones

Usando **experiment.compare_models()** se entranan múltiples modelos y se usa el que ha obtenido mejores resultados.

Usando **experiment.create_model()** se puede seleccionar un modelo concreto sin necesidad de compararlo con otros.

Extraemos el mejor modelo de todos

```
best_model = experiment.compare_models()
```

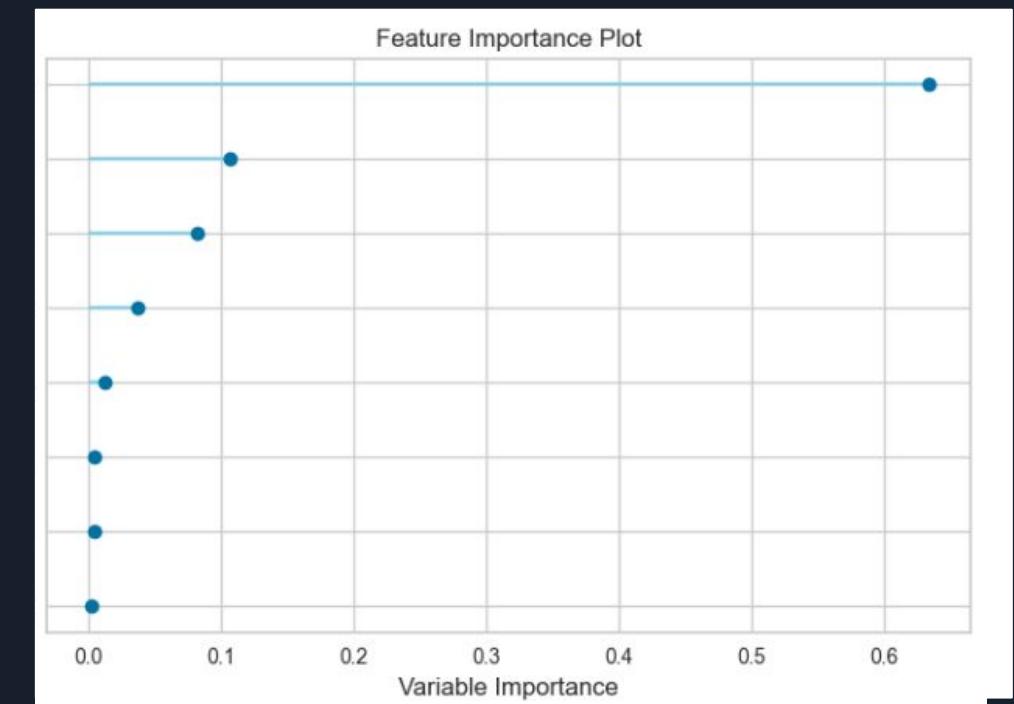
| | Model | Accuracy | AUC | Recall | Prec. |
|----------|---------------------------------|----------|--------|--------|--------|
| lr | Logistic Regression | 0.7689 | 0.8047 | 0.5602 | 0.7208 |
| ridge | Ridge Classifier | 0.7670 | 0.0000 | 0.5497 | 0.7235 |
| lda | Linear Discriminant Analysis | 0.7670 | 0.8055 | 0.5550 | 0.7202 |
| rf | Random Forest Classifier | 0.7485 | 0.7911 | 0.5284 | 0.6811 |
| nb | Naive Bayes | 0.7427 | 0.7955 | 0.5702 | 0.6543 |
| catboost | CatBoost Classifier | 0.7410 | 0.7993 | 0.5278 | 0.6630 |
| gbc | Gradient Boosting Classifier | 0.7373 | 0.7918 | 0.5550 | 0.6445 |
| ada | Ada Boost Classifier | 0.7372 | 0.7799 | 0.5275 | 0.6585 |
| et | Extra Trees Classifier | 0.7299 | 0.7788 | 0.4965 | 0.6516 |
| qda | Quadratic Discriminant Analysis | 0.7282 | 0.7894 | 0.5281 | 0.6558 |
| lightgbm | Light Gradient Boosting Machine | 0.7133 | 0.7645 | 0.5398 | 0.6036 |
| knn | K Neighbors Classifier | 0.7001 | 0.7164 | 0.5020 | 0.5982 |
| dt | Decision Tree Classifier | 0.6928 | 0.6512 | 0.5137 | 0.5636 |
| xgboost | Extreme Gradient Boosting | 0.6853 | 0.7516 | 0.4912 | 0.5620 |
| dummy | Dummy Classifier | 0.6518 | 0.5000 | 0.0000 | 0.0000 |
| svm | SVM - Linear Kernel | 0.5954 | 0.0000 | 0.3395 | 0.4090 |

Analizando un modelo.

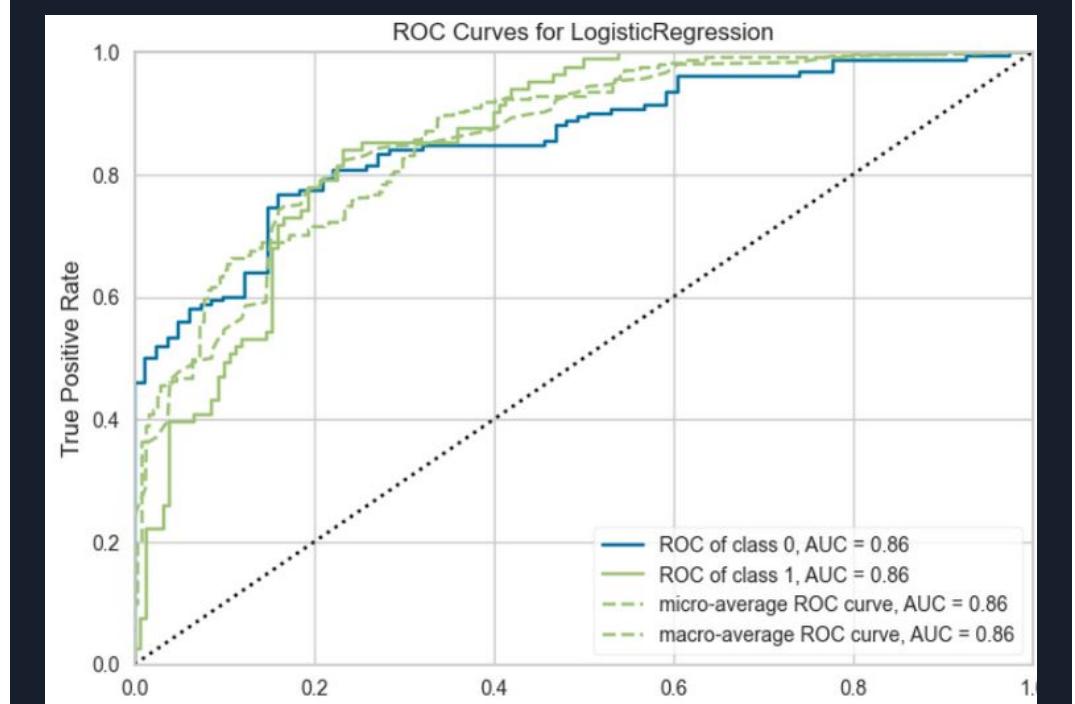
Analizando un modelo.

Además, para un experimento Pycaret te permite analizar los resultados de los modelos de manera estadística y visual.

Elegimos el mejor modelo en base a las metricas
`best_model = experiment.compare_models()`



Vemos la curva ROC del mejor modelo
`experiment.plot_model(best, plot = 'auc')`



Guardado del modelo.

Guardado de modelos.

Para **guardar un modelo** concreto podemos usar la función `save_model()`.

Si queremos guardar también la información relativa al entrenamiento y experimento, tenemos que **guardar el experimento** usando la función `save_experiment()`.

Guardamos el modelo

```
experiment.save_model(best_model, 'model')
```

Pipeline(

```
    memory=FastMemory(location=/tmp/joblib),  
    steps=[  
        ('numerical_imputer',  
            TransformerWrapper(exclude=None, include=[  
                'Number of times pregnant',...])  
        TransformerWrapper(exclude=None,  
            include=None,  
            transformer=CleanColumnNames(  
                match='[\\"\\\[\\]\\{\\}\\:\\]+'))),  
        ('trained_model',  
            LogisticRegression(C=1.0,  
                class_weight=None,  
                dual=False,  
                fit_intercept=True, intercept_scaling=1,  
                l1_ratio=None,  
                max_iter=1000,  
                penalty='l2',  
                random_state=123,  
                solver='lbfgs',  
                tol=0.0001)  
    ], verbose=False)
```

Predicciones del modelo.

Predicciones del modelo.

Teniendo un modelo ya guardado, podemos cargarlo para generar predicciones.

Los modelos que genera Pycaret, automáticamente ya **contienen toda la secuencia de procesos** para su despliegue (preprocesamiento, función de predicción, guardado, cargado,...).

```
from pycaret.datasets import get_data
from pycaret.classification import
ClassificationExperiment
```

```
data = get_data('diabetes')
```

```
experiment = ClassificationExperiment()
```

Recreamos el experimento

```
experiment.setup(
    data,
    target = 'Class variable',
    session_id = 123)
```

Ejecutamos el experimento

```
model = experiment.load_model('model')
```

Calculamos las predicciones

```
pred = experiment.predict_model(model, data)
```

Servicio de predicción.

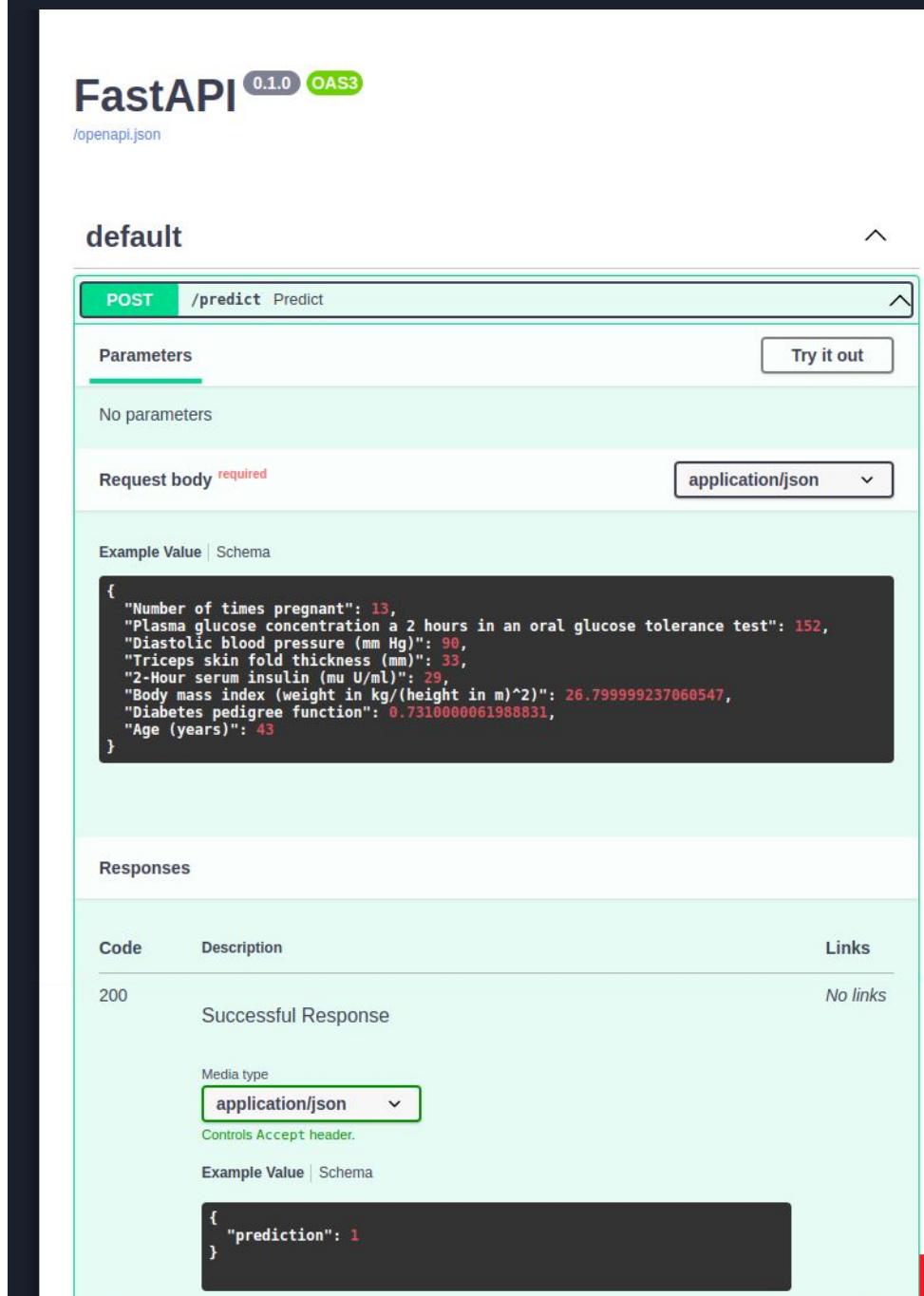
Servicio de predicción.

Pycaret permite crear un servicio de predicción de manera automática usando:

```
experiment.create_app(model, "prediction_api")
```

Esto crea un código de predicción que se puede llamar para levantar el servicio.

```
$ python prediction_api.py
```



The screenshot shows the FastAPI documentation for the `/predict` endpoint. The endpoint is a POST method with the path `/predict`. It is labeled `Predict`. The request body is required and has a schema of `application/json`. An example value is provided as a JSON object:

```
{  
    "Number of times pregnant": 13,  
    "Plasma glucose concentration a 2 hours in an oral glucose tolerance test": 152,  
    "Diastolic blood pressure (mm Hg)": 90,  
    "Triceps skin fold thickness (mm)": 33,  
    "2-Hour serum insulin (mu U/ml)": 29,  
    "Body mass index (weight in kg/(height in m)^2)": 26.799999237060547,  
    "Diabetes pedigree function": 0.7310000061988831,  
    "Age (years)": 43  
}
```

The responses section shows a successful response (200) with a description of "Successful Response". The media type is set to `application/json`, which controls the Accept header. An example value is also provided as a JSON object:

```
{  
    "prediction": 1  
}
```

Ejercicio de Pycaret.



Preguntas

1. Descarga el dataset de California House Pricing de [aquí](#) y crea un experimento de regresión a partir de él usando como target “median_house_value”.
2. La variable “ocean_proximity” es una variable categórica. Accede al atributo “experiment.pipeline” y comprueba cómo la está preprocesando. Crea un nuevo experimento que la procese de otra forma.
3. Esta vez en vez de entrenar muchos modelos (experiment.compare_models()) entrenaremos solo un RandomForest con la función experiment.create_model()

Preguntas

4. Analiza los resultados del modelo para ver qué tal funciona
5. Guarda el modelo entrenado y cargalo de nuevo para hacer predicciones sobre el mismo dataset.
6. Crea un servicio de predicción usando la función `experiment.create_api()`
7. Crea una aplicación usando la función `expetiment.create_app()`

Conclusión.



Ventajas e inconvenientes.

Ventajas.

- **Congruencia entre proceso de ML como transformación, entrenamiento y despliegue.**
- **Almacenamiento de los parámetros** de configuración del entrenamiento.
- **Medida de las métricas** de los modelos en el entrenamiento y la comparativa entre ellos.

Inconvenientes.

- **Versionado de modelos, datasets y código:** Podemos guardar el modelo, pero no gestionar las versiones ni los dataset con los que se entrenó.
- **Gestión de dependencias:** Las dependencias (librerías, archivos...) necesarios no van almacenados con el modelo.
- **Falta de versatilidad:** Es muy fácil crear un modelo de Pycaret, pero si quieres crear tu propia lógica no es tan sencillo.

Solución.

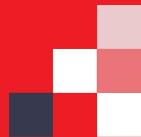


Solución

Para poder resolver los problemas del versionado de componentes así como poder añadir más versatilidad a la solución, podemos combinar Pycaret con MLFlow.



5 - MLFlow.

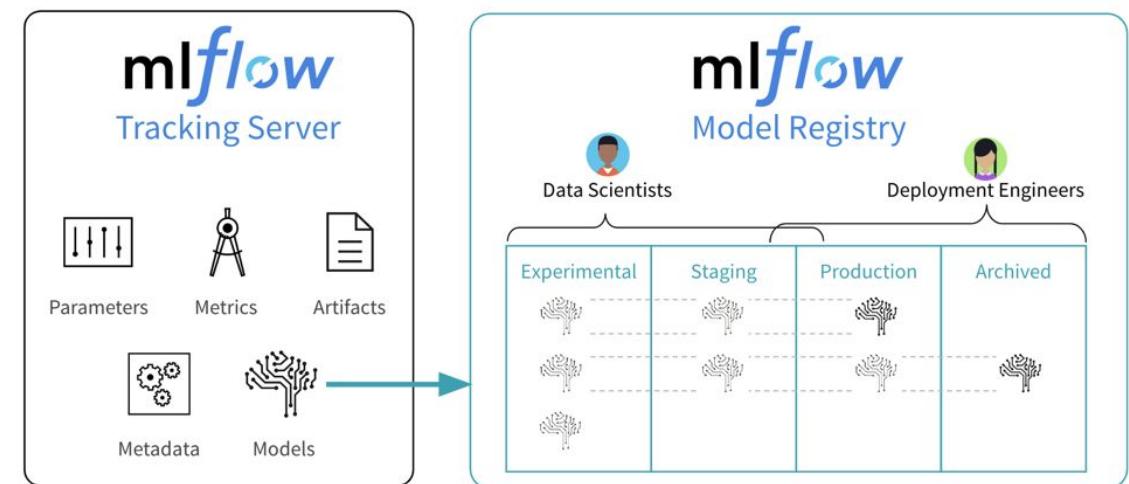
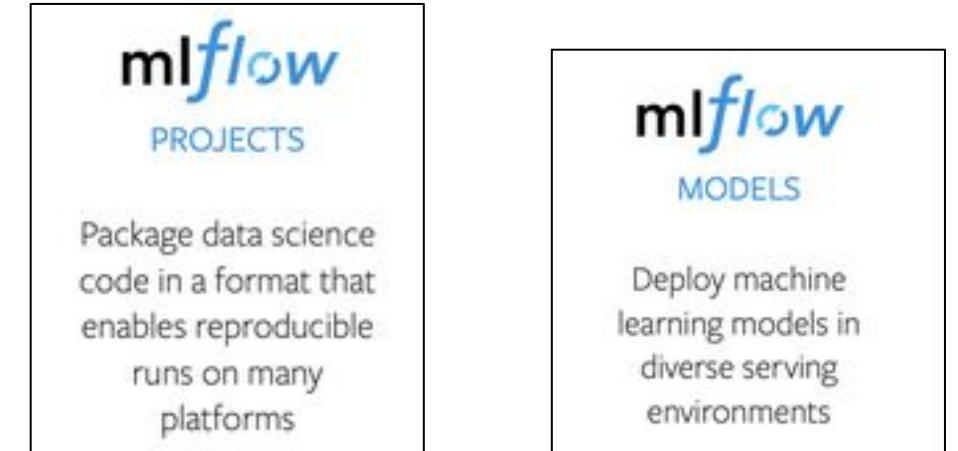


¿Qué es MLFlow?

¿Qué es MLFlow?

MLFlow es un framework de MLOps que permite crear, desplegar y versionar modelos usando su librería.

Para hacer todo esto, MLOps funciona con una base de datos relaciones y un sistema de ficheros en el que almacena toda la información.



Componentes de MLFlow.

Componentes de MLFlow.

MLFlow está compuesto por los siguientes elementos:

Traking

Es una API y una interfaz de usuario para registrar parámetros, versiones de código, métricas y artefactos.

Project

Estándar para empaquetar un proyecto de ML en un directorio usando un archivo yml que gestiona las dependencias.

Model

Convección y plantilla de código que permite desarrollar modelos de ML que sean compatibles con la plataforma de MLFlow.

ModelRegistry

Ofrece una interfaz de usuario para administrar de forma colaborativa el ciclo de vida de un modelo y su linaje

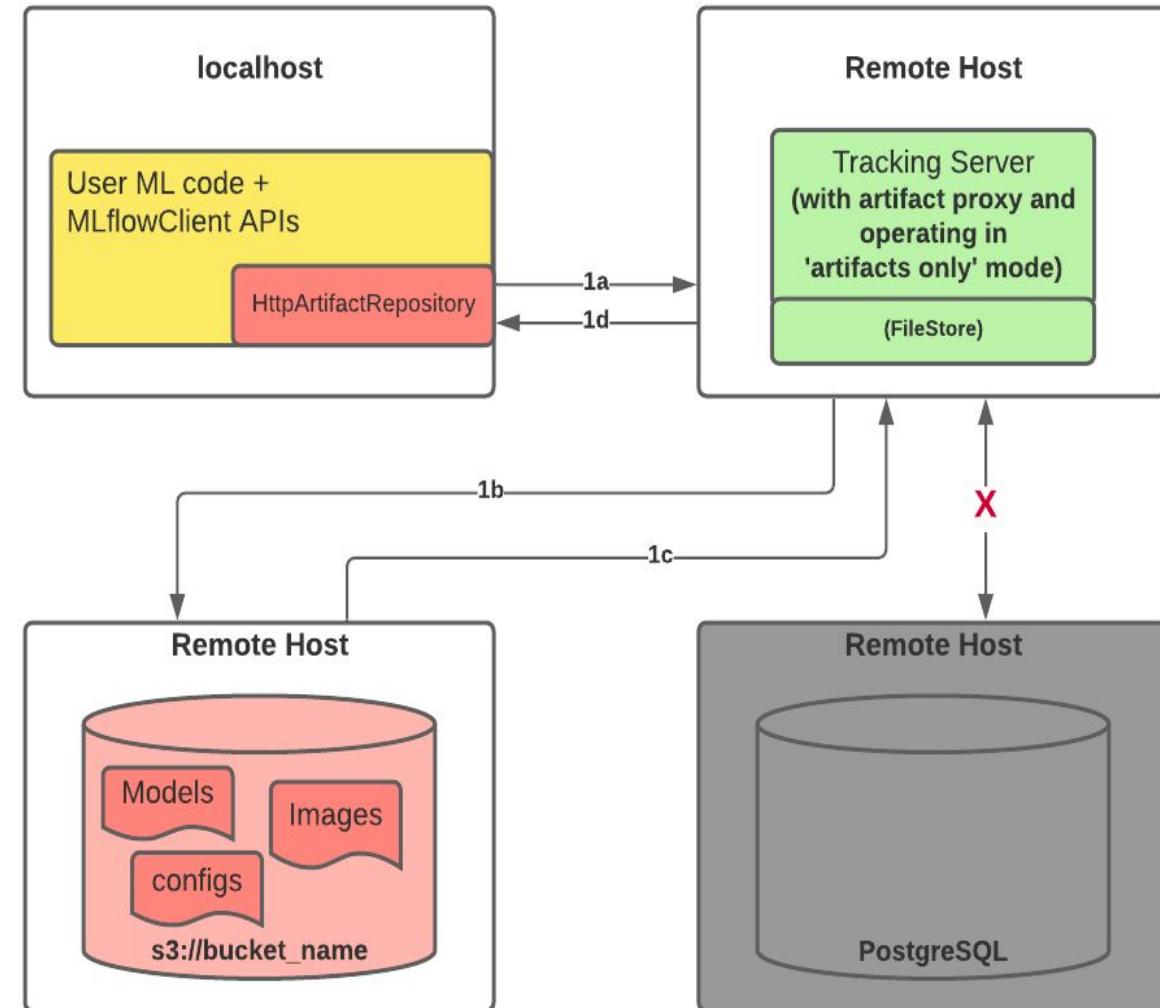
Instalación de MLFlow

Instalación de MLFlow

MLFlow se puede instalar de manera local por medio del gestor de paquetes pip.

```
host@user:~/ $ pip install mlflow
```

Sin embargo, si queremos acceder a toda la funcionalidad de MLFlow es necesario setear un server de MLFlow junto con una base de datos PostgreSQL y un filestorage.



MLFlow - DockerCompose

```
version: '3.7'
```

```
services:
```

```
minio:
  restart: always
  image: minio/minio:RELEASE.2023-04-28T18-11-17Z.fips
  container_name: mlflow_s3
```

```
ports:
```

```
- "9000:9000"
- "9001:9001"
```

```
entrypoint:
```

```
- bash
```

```
--c
```

```
- |
```

```
mkdir -p /data/mlflow
```

```
/usr/bin/docker-entrypoint.sh minio server /data --console-address ':9001' --address ':9000'
```

```
environment:
```

```
- MINIO_ROOT_USER=${AWS_ACCESS_KEY_ID:-minio}
```

```
- MINIO_ROOT_PASSWORD=${AWS_SECRET_ACCESS_KEY:-minio123}
```

```
db:
```

```
restart: always
```

```
image: mysql/mysql-server:8.0.32-1.2.11-server
```

```
container_name: mlflow_db
```

```
ports:
```

```
- "3306:3306"
```

```
environment:
```

```
- MYSQL_DATABASE=${MYSQL_DATABASE:-mlflow}
```

```
- MYSQL_USER=${MYSQL_USER:-mysql}
```

```
- MYSQL_PASSWORD=${MYSQL_PASSWORD:-mysql123}
```

```
- MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD:-root}
```

```
web:
```

```
restart: always
```

```
image: ghcr.io/mlflow/mlflow:v2.3.1
```

```
container_name: mlflow_server
```

```
depends_on:
```

```
- minio
```

```
- db
```

```
ports:
```

```
- "5000:5000"
```

```
environment:
```

```
- MLFLOW_S3_ENDPOINT_URL=http://minio:9000
```

```
- AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID:-minio}
```

```
- AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY:-minio123}
```

```
entrypoint:
```

```
- bash
```

```
--c
```

```
- |
```

```
pip install cryptography==39.0.2 &&
```

```
pip install boto3==1.26.91 &&
```

```
pip install mlflow==2.2.2 &&
```

```
pip install pymysql==1.0.2 &&
```

```
mlflow server --backend-store-uri
```

```
mysql+pymysql://${MYSQL_USER:-mysql}:${MYSQL_PASSWORD:-mysql123}@
```

```
db:3306/${MYSQL_DATABASE:-mlflow} --default-artifact-root s3://mlflow/ --host
```

```
0.0.0.0
```

MLFlow Tracking.

MLFlow Tracking.

Por medio de MLFlow tracking, podemos definir, desde código, que tipo de información queremos almacenar junto con el modelo.

Este tipo de información se almacenan como **metadatos** junto con el modelos entrenado.

```
import mlflow
from sklearn.datasets import load_iris
from sklearn.linear_model import LinearRegression
```

```
l1_ratio = 0.1
```

```
X, y = load_iris(return_X_y = True)
```

```
model = LinearRegression(l1_ratio=l1_ratio)
model.fit(X, y)
score = model.score(X, y)
```

```
# Usamos las funcionalidades de MLFlow
# tracking para almacenar los resultados
mlflow.log_param('l1_ratio', l1_ratio)
mlflow.log_metric('score', score)
```

```
# Ademas tambien podemos guardar archivos
with open("outputs/test.txt", "w") as f:
    f.write("hello world!")
mlflow.log_artifacts("outputs")
```

MLFlow Project.

MLFlow Project.

Un proyecto de MLFlow es un directorio con todo el **código de ML** que contiene un archivo con el nombre “MLproject” que define las dependencias y la forma de ejecución.

Esto permite además definir los comandos de **ejecución de los procesos** de ML que componen el proyecto.

```
user@user:~/project$ tree .
```

```
├── MLproject
├── my_model/
└── python_env.yaml
├── train.py
└── validate.py
└── requirements.txt
```

```
user@user:~/project$ cat MLproject
```

```
name: My Project
python_env: python_env.yaml
```

```
entry_points:
```

```
main:
```

```
parameters:
```

```
    data_file: path
```

```
    r2: {type: float, default: 0.1}
```

```
    command: "python train.py -r2 {r2} {data_file}"
```

```
validate:
```

```
parameters:
```

```
    data_file: path
```

```
    command: "python validate.py {data_file}"
```

```
user@user:~/project$ mlflow run train --r2 0.4
```

MLFlow Model.

MLFlow Model.

Un modelo de MLFlow se corresponde con un directorio que contiene un **artefacto de** modelo entrenado con todos los metadatos y dependencias que lo definen.

Esto permite la **ejecución del modelo** entrenado para hacer nuevas predicciones así como su despliegue.

```
user@user:~/project$ tree my_model
my_model
├── MLmodel
├── model.pkl
├── conda.yaml
└── python_env.yaml
    └── requirements.txt
```

```
user@user:~/project$ cat my_model/MLmodel
time_created: 2018-05-25T17:28:53.35
```

```
flavors:
  sklearn:
    sklearn_version: 0.19.1
    pickled_model: model.pkl
    python_function:
      loader_module: mlflow.sklearn
```

```
user@user:~/project$ mlflow models serve -m
my_model
```

MLFlow Model Registry.

MLFlow Model Registry.

Interfaz centralizado que permite almacenar y acceder a todos los modelos creados de manera conjunta.

Además, se encarga del **versionado** y del lineage de los modelos así como de los datasets que se han usado para su creación.

Podemos acceder a él por medio de la función “log_model” y “register_model”

```
import mlflow
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
```

```
l1_ratio = 0.1
```

```
X, y = load_iris(return_X_y = True)
```

```
model = LogisticRegression(l1_ratio=l1_ratio)
model.fit(X, y)
score = model.score(X,y)
```

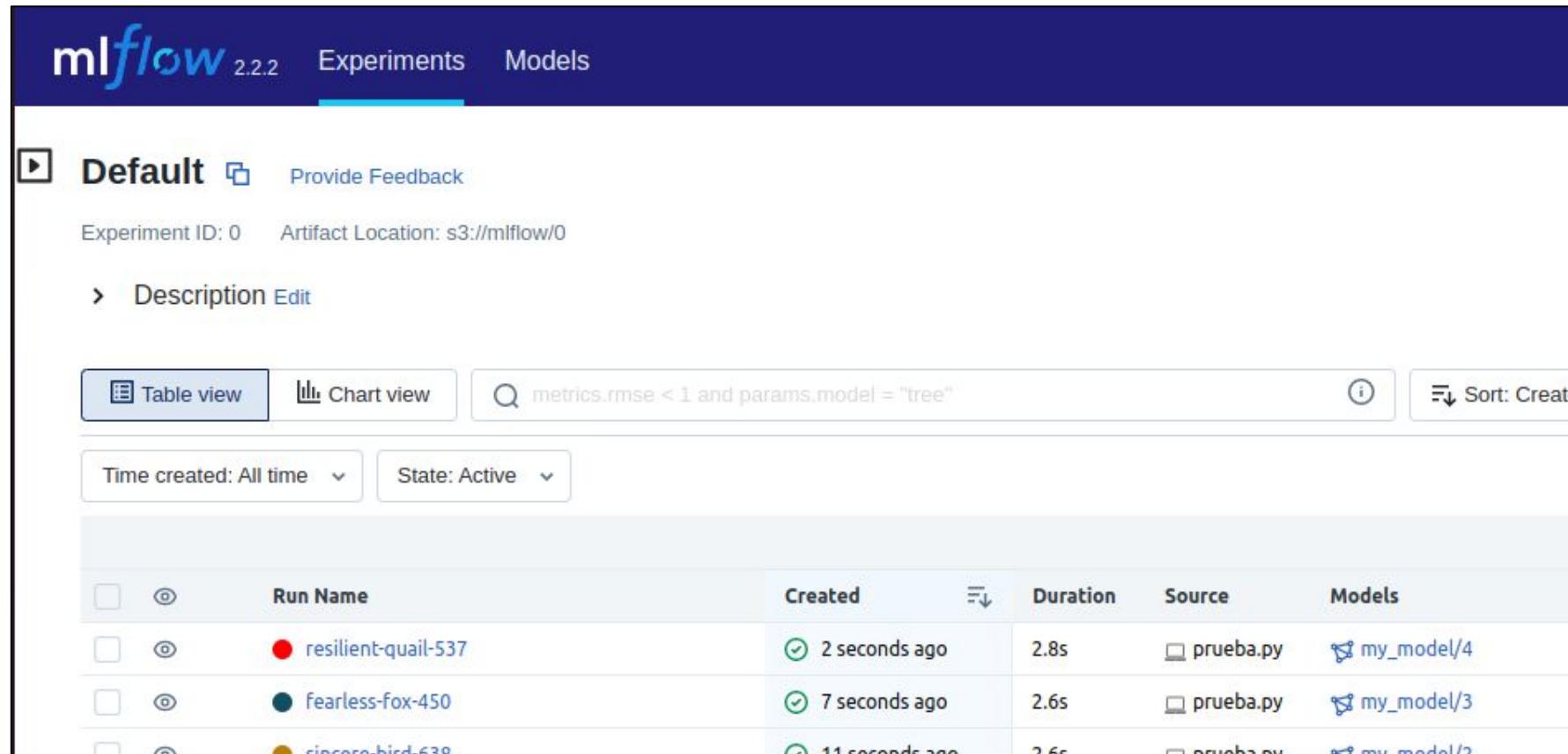
```
mlflow.log_param('l1_ratio', l1_ratio)
mlflow.log_metric('score', score)
```

```
# Para registrar un modelo nuevo podemos usar
# lod_model definido "registered_model_name"
mlflow.sklearn.log_model(
    sk_model=model,
    artifact_path='my_model',
    registered_model_name='my_model')
```

¿Cómo podemos ver todas las métricas?

Experimentos de MLFlow

Cada ejecución de un proceso de python que logea algo a MLFlow se denomina “experimento” todos los **experimentos** junto con su información son visibles desde el UI



The screenshot shows the MLflow UI interface. At the top, there's a dark blue header with the "mlflow 2.2.2" logo, followed by tabs for "Experiments" and "Models". Below the header, the "Experiments" tab is active, showing a list of runs under the "Default" experiment. Each run is represented by a row in a table with columns: Run Name, Created, Duration, Source, and Models. The first run, "resilient-quail-537", is highlighted with a red dot next to its name. The second run, "fearless-fox-450", is highlighted with a teal dot. The third run, "cincero-bird-638", is highlighted with an orange dot. The table also includes filters at the top: "Table view", "Chart view", a search bar ("metrics.rmse < 1 and params.model = 'tree'"), and buttons for "Sort: Create" and filtering by "Time created: All time" and "State: Active".

| | Run Name | Created | Duration | Source | Models |
|--------------------------|---------------------|----------------|----------|-----------|------------|
| <input type="checkbox"/> | resilient-quail-537 | 2 seconds ago | 2.8s | prueba.py | my_model/4 |
| <input type="checkbox"/> | fearless-fox-450 | 7 seconds ago | 2.6s | prueba.py | my_model/3 |
| <input type="checkbox"/> | cincero-bird-638 | 11 seconds ago | 2.6s | prueba.py | my_model/2 |

¿Cómo realizamo predicciones con un modelo?

Predicciones con MLFlow.

Una vez hemos registrado el modelo, podemos usar su nombre y su versión para realizar predicciones.

También se pueden realizar predicciones usando modelos que no se han registrado. Sin embargo, en estos casos perdemos la capacidad de versionarlos.

```
import mlflow
from sklearn.datasets import load_iris
```

```
X, y = load_iris(return_X_y = True)
```

```
name = "my_model"
version = 1
uri = f"models:{name}/{version}"
```

```
# Podemos obtener una referencia al modelo a
# partir de su URI
model = mlflow.pyfunc.load_model(
    model_uri=uri)
```

```
# Llamamos al modelo como si fuese un modelo
# de sklearn, sin embargo, esto solo realiza una
# llamada al server. Las predicciones se hacen
# en el server
model.predict(X)
```

¿Y si queremos crear nuestros propios modelos?

CustomModels

MLFlow permite registrar y ejecutar modelos de muchos frameworks distintos (sklearn, keras,...). Los adaptadores de MLFlow que permiten hacer esto se denominan **Flavours**.

Es posible crear nuevos flavours en el caso de que queramos inventarnos nuevos modelos. Esto se puede hacer creando un objeto de tipo **PythonModel**.

```
import mlflow
from xgboost import XGBClassifier
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y = True)
model = XGBClassifier().fit(X, y)
model.save_model("xgb_model.pth")

class XGBWrapper(mlflow.pyfunc.PythonModel):

    def load_context(self, context):
        from xgboost import XGBClassifier
        self.model = XGBClassifier().load_model(
            context.artifacts["model"])

    def predict(self, context, model_input):
        return self.model.predict(
            model_input.values)

mlflow.pyfunc.log_model(
    path="custom_model",
    python_model=XGBWrapper(),
    artifacts={"model": "xgb_model.pth"},
    registered_model_name="custom_model"
)
```

Ejercicio de MLFlow.



Preguntas

1. Descarga el dataset de California House Pricing de [aquí](#) y crea un modelo de RandomForest usando la librería Scikit-Learn. Haz un log del número de estimadores del modelo y crea un **modelo de MLFlow** usando “mlflow.sklearn.log_model()”
2. Encuentra el experimento anterior usando mlflow desde línea de comando (mlflow experiment list) y busca en los directorios de mlflow el archivo de modelo entrenado
3. Crea un **proyecto de MLflow** que permita ejecutar el entrenamiento de un modelo pasándole como variable de entrada el número de estimadores del random forest

Preguntas

4. Levanta MLFlow server con docker compose y accede a la dirección 127.0.0.1:5000 para comprobar que puedes ver la interfaz
5. Setea MLFlow para que use el server de Docker-compose y ejecuta el proyecto anterior para crear un modelo en el server (pista: tienes que setear estas variables)

```
export MLFLOW_TRACKING_URI=http://localhost:5000
export MLFLOW_S3_ENDPOINT_URL=http://localhost:9000
export AWS_ACCESS_KEY_ID=minio
export AWS_SECRET_ACCESS_KEY=minio123
```

6. Ve al interfaz de usuario y mira el modelo creado en el último experimento. Usa el código para hacer predicciones.

Conclusión.



Ventajas e inconvenientes.

Ventajas.

- **Congruencia entre proceso de ML como transformación, entrenamiento y despliegue.**
- **Almacenamiento de los parámetros y métricas** de cada entrenamiento y model.
- **Versionado de modelos, datasets y código.**
- **Gestión de dependencias**

Inconvenientes.

- **Solución ligada a su framework:** Dependemos completamente de conocer y usar MLFlow.
- **Limitaciones de plataforma:** MLFlow funciona bien como sistema de versionado, pero somo servicio de desplieuge no escala si hay muchos modelos.
- **Falta de integración** con otros servicios que no son de ML

Solución.



Solución

Es una práctica común usar MLFlow como servicio de versionado y registro de modelos y experimentos. Sin embargo, como plataforma tiene limitaciones de escalabilidad e integración.

La solución es usar una herramienta de plataforma que se integre con MLFlow para gestionar todos esos procesos.

Gracias!



Sábado.



Título de la formación: MLOps y DataOps

Presentación.

Ángel Delgado Panadero

- Machine Learning Engineer - Paradigma
- Investigador Independiente
- Tutor de Máster
- Físico



Índice.

Viernes

1. Introducción.
2. Problemas comunes.

Preguntas

3. ¿Qué es el MLOps?
4. Pycaret.

Ejercicio de Pycaret
5. MLFlow.

Ejercicio de MLFlow

Sábado

1. Introducción
2. Diseño de Código.

Ejercicio de ML

3. Docker.

Ejercicio de Docker
4. Pipelines con Airflow.

Ejercicio de Pipelines de ML.
5. Pipelines con Sagemaker.

Disclaimer.

- Si la clase es **interactiva** mejor!
- **MLOps** no tiene un standard.
- Cada uno tiene un **background** distinto.

1 - Introducción.

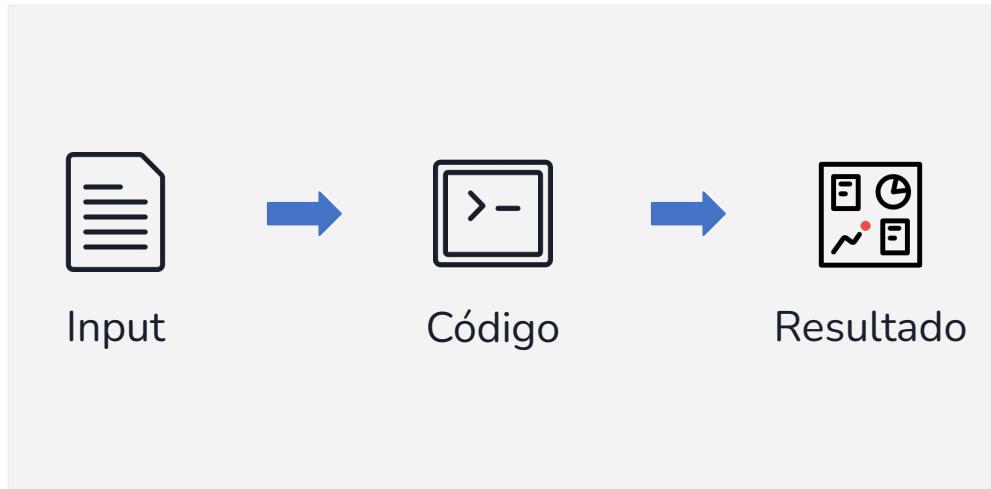


¿En qué se diferencia el Machine Learning?

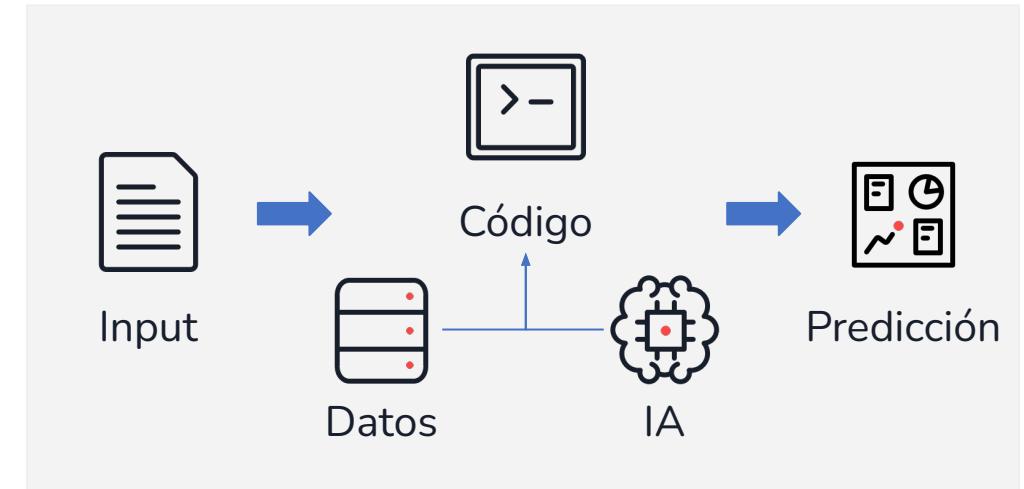
Desarrollo de Software vs Desarrollo de ML

La diferencia entre un proyecto de Machine Learning y uno de desarrollo de software es que hay **más componentes además del código** que es necesario desarrollar.

Desarrollo de Software



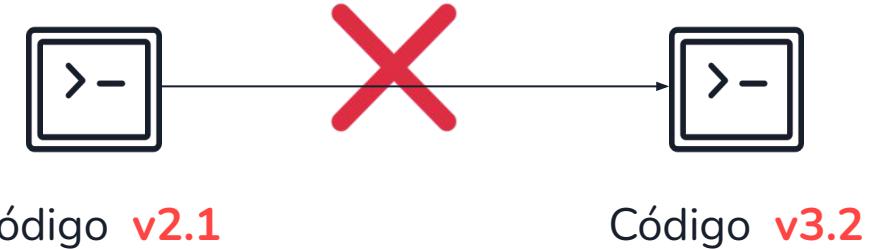
Desarrollo de Machine Learning



¿Cuáles son los problemas?

¿Cuáles son los problemas?

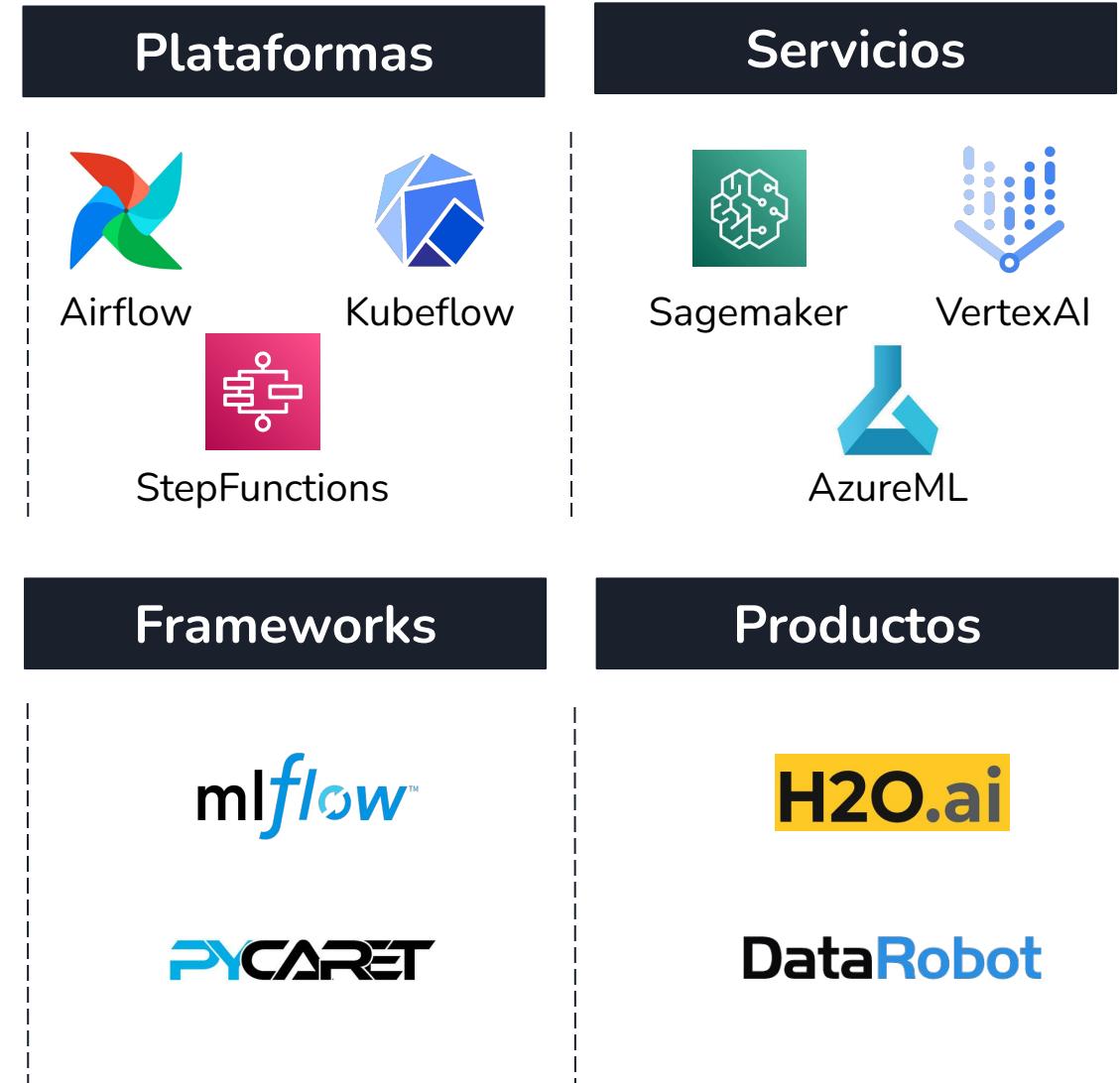
- Falta de concordancia entre procesos
- Falta de concordancia entre código y modelos.
- Falta de concordancia entre datos y modelos.



¿Cuáles son las soluciones?

¿Cuáles son las soluciones?

- **Buenas prácticas**
 - Naming
 - Versionado
 - Flujos
- **Herramientas**
 - Plataformas
 - Servicios
 - Frameworks
 - Productos



Problemas de los frameworks

Problemas de los frameworks

Desarrollos muy ligados a su framework:
Dependemos completamente de conocer y usar MLFlow.

Solución con herramientas de plataforma:
Permiten delegar la parte de MLOps en la plataforma y no en el desarrollo.



Herramientas de plataforma

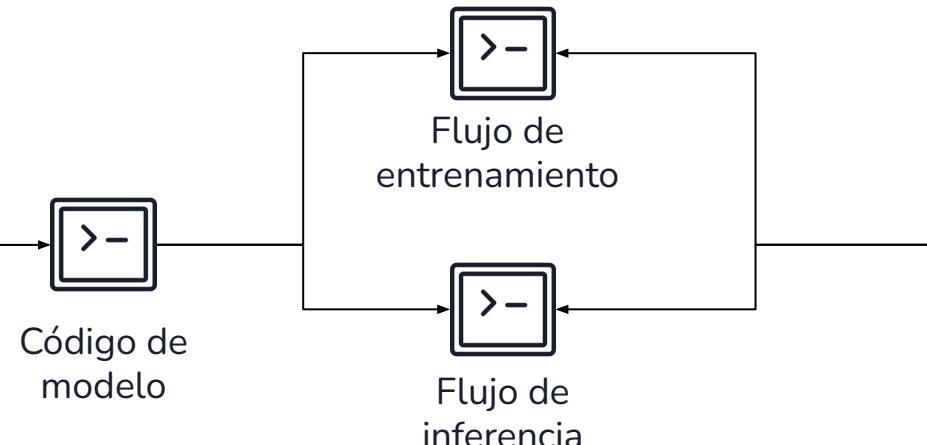
Herramientas de plataforma.

Separación entre la lógica de automatización de procesos y del código de Machine Learning.



Data Scientist

Desarrolla el **código** de Machine Learning sin tener que preocuparse de los flujos.



Código de ML

Para que sea reutilizable tiene que seguir un patrón de diseño concreto.



MLOps

Desarrolla los **flujos** de Machine Learning sin preocuparse del algoritmo.

2 - Diseño de código.

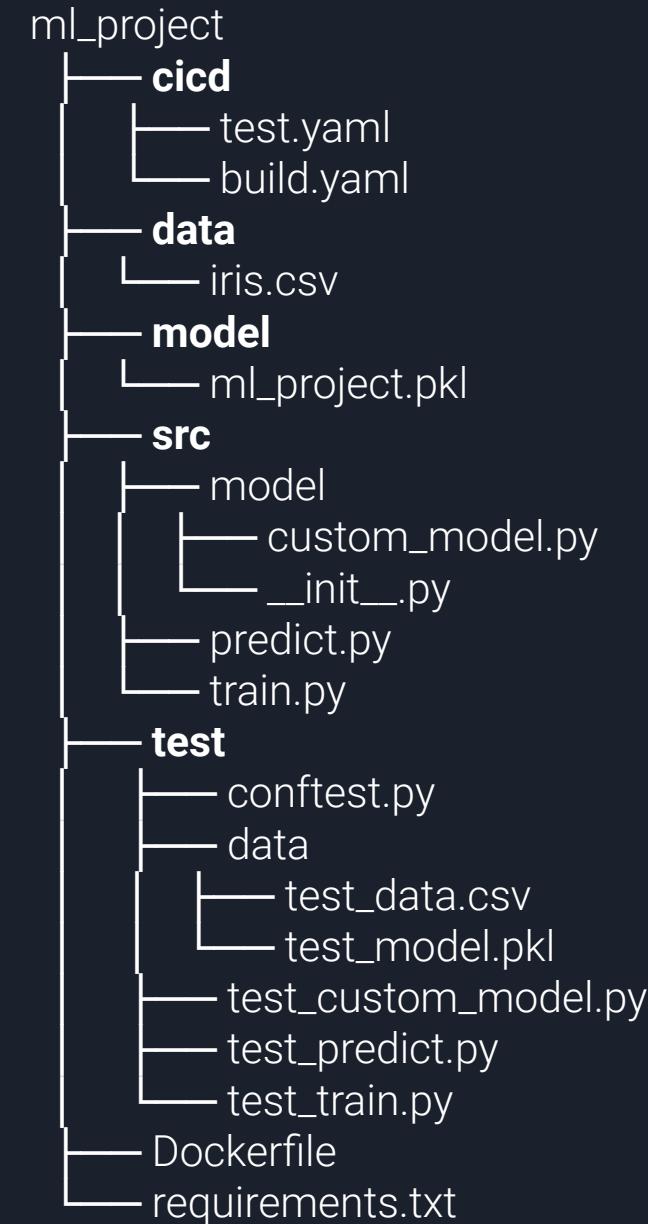


Programación orientada a ML

Estructura de proyecto

Un proyecto de Machine Learning debería tener una estructura que tenemos a la derecha.

Además es importante diferenciar el desarrollo de la lógica de ML (custom_model.py) de la lógica de ejecución de los procesos de ML (predict.py y train.py)



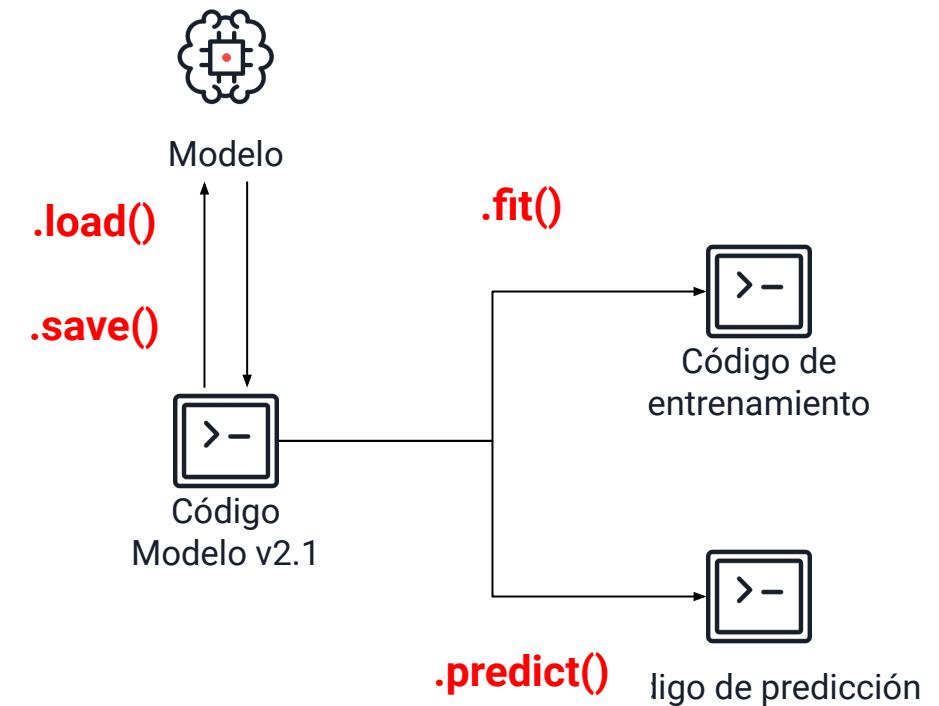
Interfaz de modelo

Interfaz de modelo.

Homogeneidad entre modelos: Para que diferentes algoritmos sean ejecutables desde diferentes procesos de manera análoga deben tener el mismo interfaz.

Lógica de Machine Learning: El hecho de crear una clase con el algoritmo nos permite separar el código de ML del código de ejecución del proceso.

Congruencia entre procesos: Al unificar las funciones de predicción y de entrenamiento en una misma función, nos permite asegurar la congruencia entre entrenamiento y predicción.



Interfaz de modelo.

Homogeneidad entre modelos: Para que diferentes algoritmos sean ejecutables desde diferentes procesos de manera análoga deben tener el mismo interfaz.

Lógica de Machine Learning: El hecho de crear una clase con el algoritmo nos permite separar el código de ML del código de ejecución del proceso.

Congruencia entre procesos: Al unificar las funciones de predicción y de entrenamiento en una misma función, nos permite asegurar la congruencia entre entrenamiento y predicción.

```
import argparse
import pandas as pd
from .model_logic import ModelLogic
```

```
class ModelLogic():
```

```
    def __init__(self, **parameters):
        self.parameters = parameters
```

```
    def fit(self, X, y):
        ...
        return self
```

```
    def predict(self, X):
        ...
        return prediction
```

```
    def save_model(self, model_path):
        ...
        return self
```

```
    def load_model(self, model_path):
        ...
        return self
```

Diseño de comandos

Diseño de comandos.

Parámetros de ejecución: El uso de comando del sistema permite ejecutar el mismo proceso con diferentes parámetros. Esto facilita la experimentación a la hora de ejecutar los procesos con diferentes parámetros.

Lógica de ejecución: Este tipo de scripts contienen toda la lógica necesaria para la ejecución de cada uno de los procesos. Lectura de datos, entrenamiento, guardado del modelo entrenado, lectura del modelo entrenado...

```
$ python train.py \  
--data_path $DATA_PATH \  
--model_path $MODEL_PATH \  
--param_lr 0.1 \  
--param_max_iter 100
```



Código de entrenamiento

```
$ python predict.py \  
--model_path $MODEL_PATH \  
--port 8080
```



Código de predicción



Código
Modelo

Diseño de comandos.

Parámetros de ejecución: El uso de comando del sistema permite ejecutar el mismo proceso con diferentes parámetros. Esto facilita la experimentación a la hora de ejecutar los procesos con diferentes parámetros.

Lógica de ejecución: Este tipo de scripts contienen toda la lógica necesaria para la ejecución de cada uno de los procesos. Lectura de datos, entrenamiento, guardado del modelo entrenado, lectura del modelo entrenado...

```
import argparse
import pandas as pd
from .model import ModelLogic

def train(args):
    df_train = pd.read_csv(args.data_path)
    X_train = df_train.drop(["Species"], axis=1)
    y_train = df_train["Species"]

    ModelLogic(learning_rate=args.lr
               ).fit(X_train, y_train
                     ).save(args.model_path)

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--lr", type=int)
    parser.add_argument("--data_path", type=str)
    parser.add_argument("--model_path", type=str)
    args = parser.parse_args()

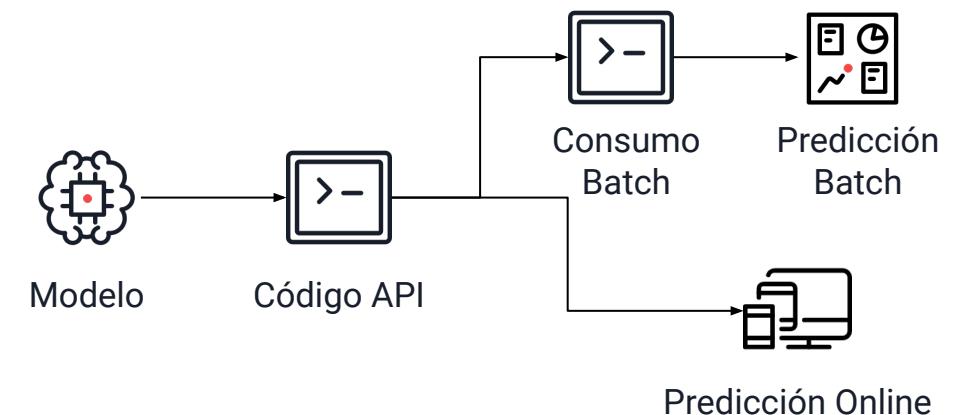
    train(args)
```

Servicio de predicción

Servicio de predicción

Predicción online vs batch: En proyectos de real time queremos que las predicciones se ejecuten bajo demanda usando una API, pero si no, lo común es que se haga en procesos batch.

Unificación del proceso de predicción: Para que no existan dos lógicas distintas de predicción, la práctica recomendable es que siempre se desarrolle como una API y que se consuma en batch o bajo demanda según la necesidad.



Servicio de predicción

Predicción online vs batch: En proyectos de real time queremos que las predicciones se ejecuten bajo demanda usando una API, pero si no, lo común es que se haga en procesos batch.

Unificación del proceso de predicción: Para que no existan dos lógicas distintas de predicción, la práctica recomendable es que siempre se desarrolle como una API y que se consuma en batch o bajo demanda según la necesidad.

```
import argparse
from fastapi import FastAPI
from model_logic import ModelLogic

def run(args):
    model = ModelLogic().load(args.model_path)
    app = FastAPI()

    @app.post("/predict")
    def predict(request: dict):
        predictions = []
        for instance in request["instances"]:
            pred = model.predict(instance)
            predictions.append(pred)

        return {"predictions": predictions}
    return app

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--port', type=int)
    parser.add_argument('--model_path', type=str)
    args = parser.parse_args()

    uvicorn.run(run(args), host='0.0.0.0', port=args.port)
```

Testing de código

Testing de código

Test del modelo: El modelo de Machine Learning se puede validar por medios de test unitarios. El objetivo es validar cada uno de los métodos que componen el modelo independientemente.

Test de comandos: Los diferentes comandos de ejecución se pueden validar por medios de test funcionales. El objetivo es testear la ejecución de cada comando de manera completa.

Datos de testeo



Datos de test
entrenamiento



Datos de test
predicción



Modelo
de test

Test funcionales



Test código
entrenamiento



Test código
predicción



Test código
de modelo

Test unitarios

Testing de código

Test del modelo: El modelo de Machine Learning se puede validar por medios de test unitarios. El objetivo es validar cada uno de los métodos que componen el modelo independientemente.

Test de comandos: Los diferentes comandos de ejecución se pueden validar por medios de test funcionales. El objetivo es testear la ejecución de cada comando de manera completa.

```
import os, tempfile, pytest
from src.model import ModelLogic
```

```
class TestModelLogic():
```

```
    def test_fit(self, X, y):
        model = ModelLogic().fit(X,y)
```

```
    def test_predict(self, X, y):
        y = ModelLogic().fit(X,y).predict(X)
```

```
    def test_save(self,):
        with tempfile.TemporaryDirectory() as tmp:
            ModelLogic().save(f'{tmp}/model.pkl')
```

```
    def test_load(self):
        with tempfile.TemporaryDirectory() as tmp:
            ModelLogic().load(f'{tmp}/model.pkl')
```

Logging

Logging



El logging permite monitorizar, no únicamente la lógica del software, sino también los resultados de ML

Logging de software: Logging del estado del proceso de ejecución (principalmente en los scripts de los comandos de ejecución).

Logging de Machine Learning: Logging de las métricas y parámetros del proceso de ML. Para separarlos del resto de logs, es recomendable sacarlos por un canal aparte (p.e. INFO).

Código de procesamiento.

INFO: Read file dataset.
CRITICAL: Dataset is empty.
ERROR: Wrong file path.

Código de modelo.

INFO: Params: lr=0.1.
INFO: Accuracy: 0.87.
INFO: Vanish gradients.

Logging

El logging permite monitorizar, no únicamente la lógica del software, sino también los resultados de ML

Logging de software: Logging del estado del proceso de ejecución (principalmente en los scripts de los comandos de ejecución).

Logging de Machine Learning: Logging de las métricas y parámetros del proceso de ML. Para separarlos del resto de logs, es recomendable sacarlos por un canal aparte (p.e. INFO).

```
import logging
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
```

```
class ModelLogic():
```

```
    def __init__(self, max_iter = 100):
```

```
        logging.info(f'Params: max_iter: {max_iter}')
        self._processor = StandardScaler()
        self._predictor = LogisticRegression(max_iter)
```

```
    def fit(self, X, y):
```

```
        _X = self._processor.fit_transform(X)
        self._predictor.fit(_X,y)
```

```
        score = self._predictor.score(_X, y)
        logging.info(f'Accuracy: {score}')
```

```
    return self
```

Ejercicio de diseño de código.



Preguntas

1. Crear un proyecto de Machine Learning similar al visto anteriormente pero usando el algoritmo de LogisticRegression con la siguiente estructura de ficheros.

```
project/
    data/iris.csv
    test/test_model.py
    src/model.py
    model/
        model.pkl
    train.py
```

El script `train.py` deberá generar un modelo entrenado en el directorio “`model/`” y deberá ejecutarse como

```
python train.py --n-iters 100
```

Preguntas

2. Entrenar el modelo.
3. Crear un script de test unitario para el model con test unitarios para cada uno de los métodos. El test deberá comprobar lo siguiente:
 - El método `.fit()` crea dentro del objeto un modelo entrenado
 - El método `.fit()` lanza una excepción si la variable “y” no es categórica
 - El método `.predict()` devuelve un array de predicciones con tantas predicciones como líneas tiene el argumento “X”
 - El método `.predict()` lanza una excepción si las columnas de “X” no coinciden con las columnas con las que se entrenó el modelo.

Preguntas

4. Crear el archivo de predicción (predict.py) como un servicio que cargue el modelo y acepte peticiones de predicción en el endpoint “/predict”.

Levantar el servicio de predicción y hacer una predicción

```
curl -X POST \  
  -H 'Content-Type: application/json' \  
  -d '{: [{"SepalLengthCm": [1],"SepalWidthCm" : [1],"PetalLengthCm": [1],"PetalWidthCm" : [1]}]}' \  
  http://localhost:8000/predict
```

Conclusión.



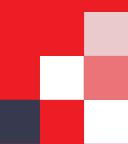
Diseño de código.

- Separar el código en código de modelo de te procesamiento nos permite diferenciar la lógica de ML de la lógica de ejecución.
- Seguir un estandar e interfaz de modelo concreto nos permite poder testear el código así como logear las métricas y resultados del modelo.

Siguiente objetivo.

- Encapsular todo el software en un “artefacto” que sea distribuible

3 - Artefactos.



Docker.



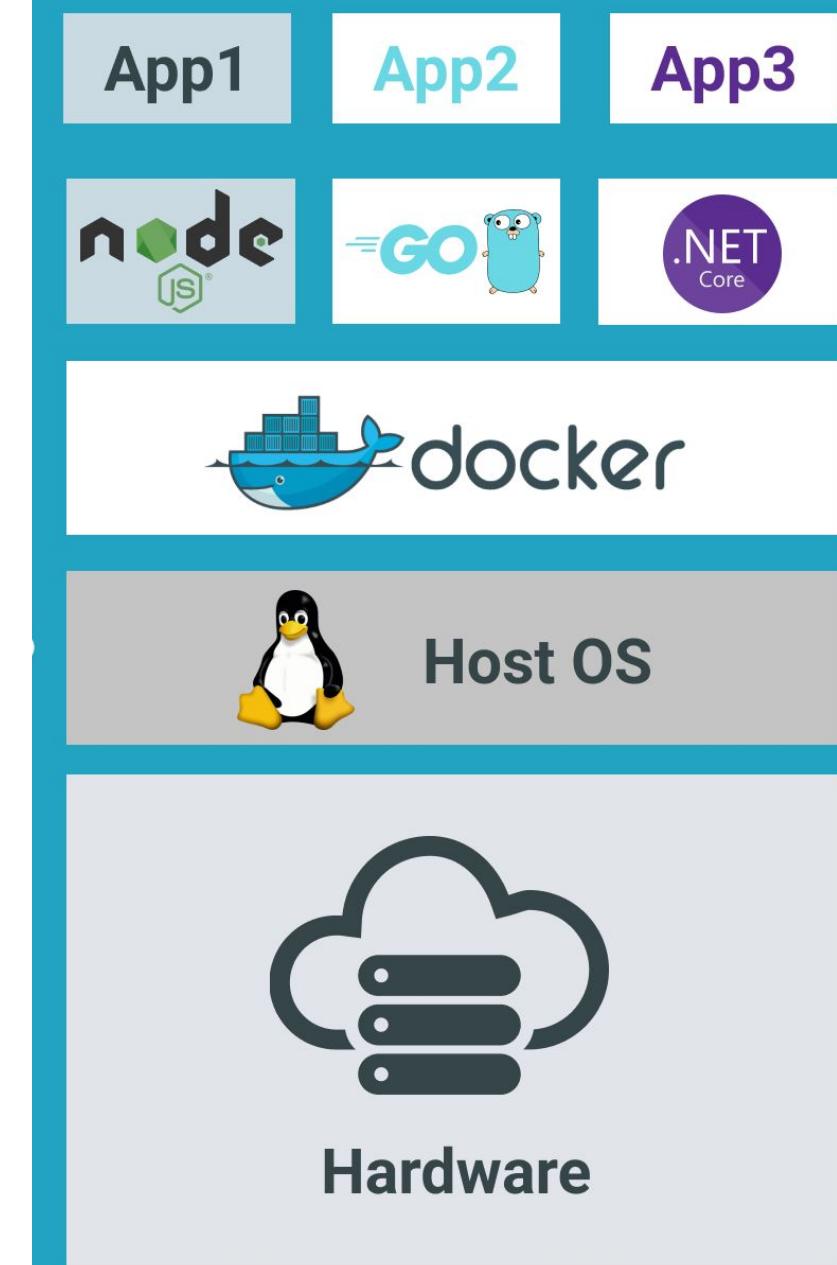
¿Qué es Docker?

¿Qué es Docker?

Docker es un sistema de virtualización ligera que permite encapsular software para ser ejecutado en cualquier plataforma (como una máquina virtual pero más ligera).

Las ventajas que ofrece esto son las siguientes:

- Multiplataforma.
- Encapsulado.
- Comandos de ejecución.

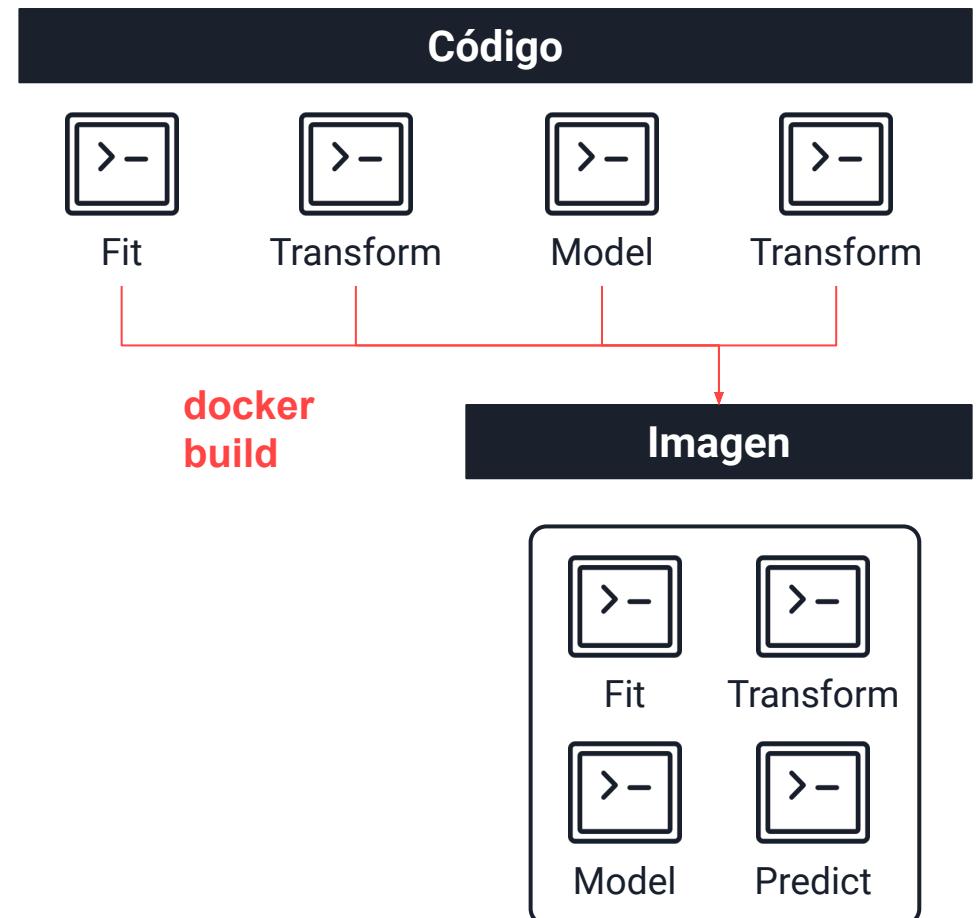


¿Qué es Docker?

Docker es un sistema de virtualización ligera que permite encapsular software para ser ejecutado en cualquier plataforma (como una máquina virtual pero más ligera).

Las ventajas que ofrece esto son las siguientes:

- **Multiplataforma.**
- **Encapsulado.**
- **Comandos de ejecución.**



¿Cómo se crea una imagen Docker?

¿Cómo se crea una imagen Docker?

Mediante un archivo que se denomina **Dockerfile**. Este archivo contiene una secuencia de sentencias en un lenguaje específico que definen cómo construir la imagen.

Una vez creado el archivo se puede ejecutar con el comando:

```
docker build -t <NOMBRE_IMAGEN> .
```

FROM: Este comando te permite definir la imagen que se usa como base antes de empezar (p.e.: ubuntu:20)

RUN: Este comando te permite ejecutar una sentencia de código de línea de comandos durante la construcción de la imagen

COPY: Este comando te permite copiar archivos de la máquina anfitrión a dentro de la imagen

ENTRYPOINT: Permite definir el comando que se va a ejecutar cuando se ejecute la imagen

CMD: Permite definir las opciones con las que se va a ejecutar el comando (o sea, se añade al entrypoint)

¿Cómo se crea una imagen Docker?

Mediante un archivo que se denomina **Dockerfile**. Este archivo contiene una secuencia de sentencias en un lenguaje específico que definen cómo construir la imagen.

Una vez creado el archivo se puede ejecutar con el comando:

docker build -t <NOMBRE_IMAGEN> .

```
# Imagen base sobre la que construir la nuestra
FROM python:3.8
```

```
# Instalacion de dependencias del sistema
RUN apt-get upgrade && apt-get update -y
```

```
# Instalación de las dependencias del proyecto
COPY ./requirements.txt /home
WORKDIR /home
RUN pip install -r requirements.txt
```

```
# Copiamos el proyecto dentro de la imagen
COPY ./ /home
```

```
# Creamos un comando de ejecucion
ENTRYPOINT ["python"]
CMD ["train.py", "--n-estimators", "100"]
```

¿Cómo se ejecuta una imagen Docker?

¿Cómo se ejecuta una imagen Docker?

Podemos ver las imágenes creadas con:

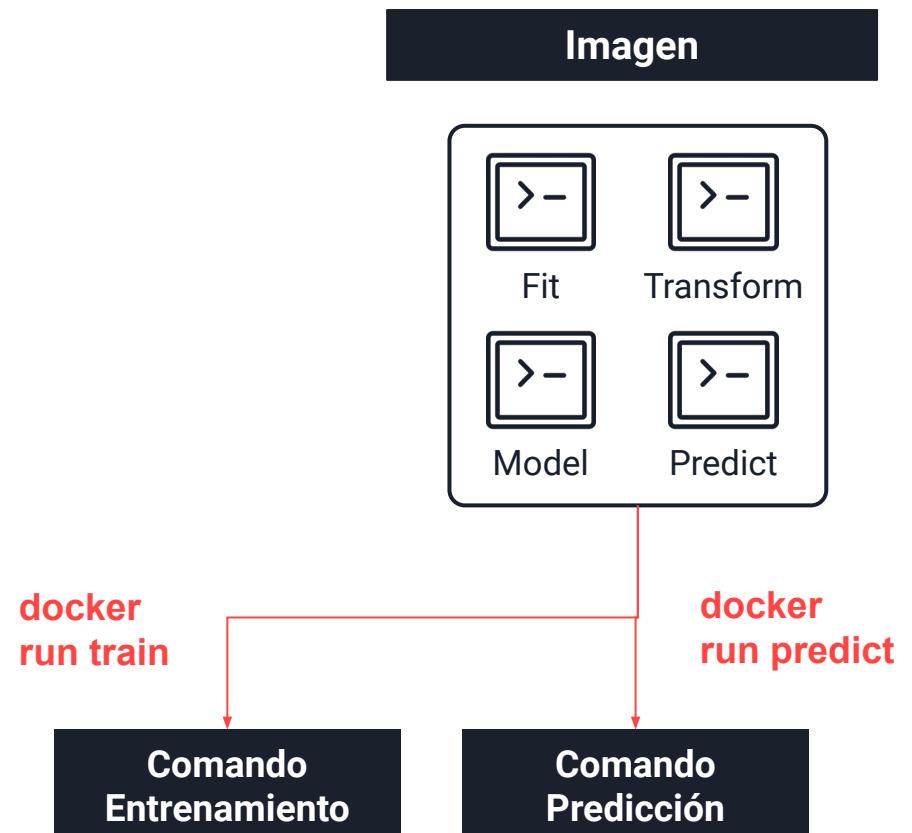
`docker imagen`

Para ejecutar una imagen de las creadas, podemos ejecutar el comando. Una imagen en ejecución se llama **contenedor**.

`docker run <IMAGEN>`

Incluso podemos pasárle comandos

`docker run <IMAGEN> python train.py`



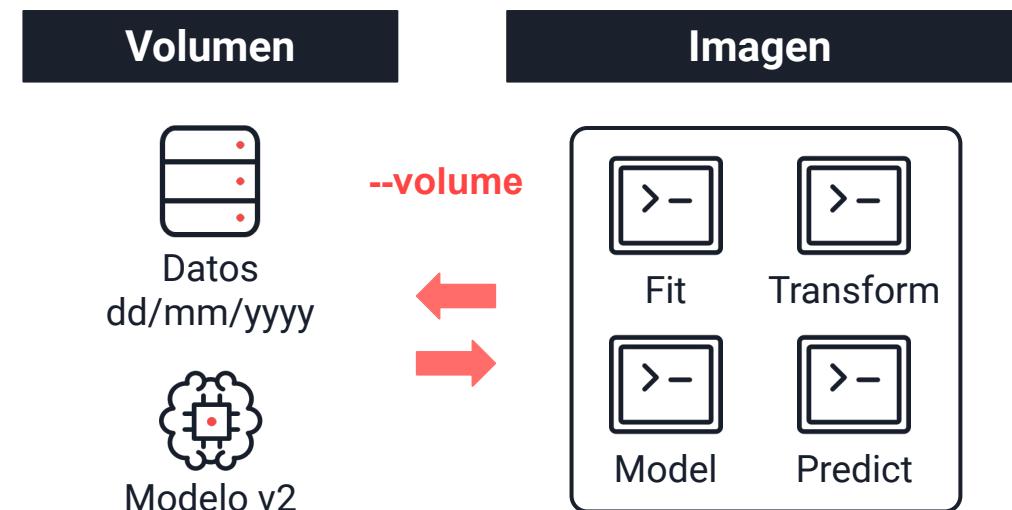
¿Podemos leer y escribir archivos de fuera?

¿Podemos leer y escribir archivos de fuera?

Docker permite montar archivos de fuera de la máquina anfitrión a dentro del contenedor por medio de **volumenes**.

Para ejecutar crear un contenedor en un volumen hay que ejecutar el siguiente comando:

```
docker run -v /path_source:/path_target  
<IMAGEN>
```



Despliegue.



¿Qué es desplegar?

¿Qué es desplegar?

Desplegar consiste en llevar la aplicación del entorno/máquina donde se ha desarrollado al entorno/máquina donde se va a ejecutar.

En un proyecto de ML hay que llevar a producción: **datos, modelo y código.**

La forma más sencilla de llevar el código a producción es llevando a **producción la imagen docker** como un todo

Registro de datos y modelos

Registro de datos y modelos

A la hora de llevar un modelo de Machine Learning a producción es necesario registrar la configuración con la que se entrenó el modelo, así como los datos que se usaron.

Registro de datos

Repositorio donde se almacenan los datos de cada entrenamiento. Esto permite asegurar la reproducibilidad

Registro de modelos

Repositorio donde se almacenan todos los modelos entrenados junto con los metadatos necesarios para auditarlo.

Naming y Versionado

Naming y Versionado

Naming

El naming hace referencia al conjunto de nombre que se le pone a cada uno de los componentes de la aplicación.

Es importante que haya una política común para que todas las componentes de un mismo proyecto tengan la misma “raíz” en el nombre

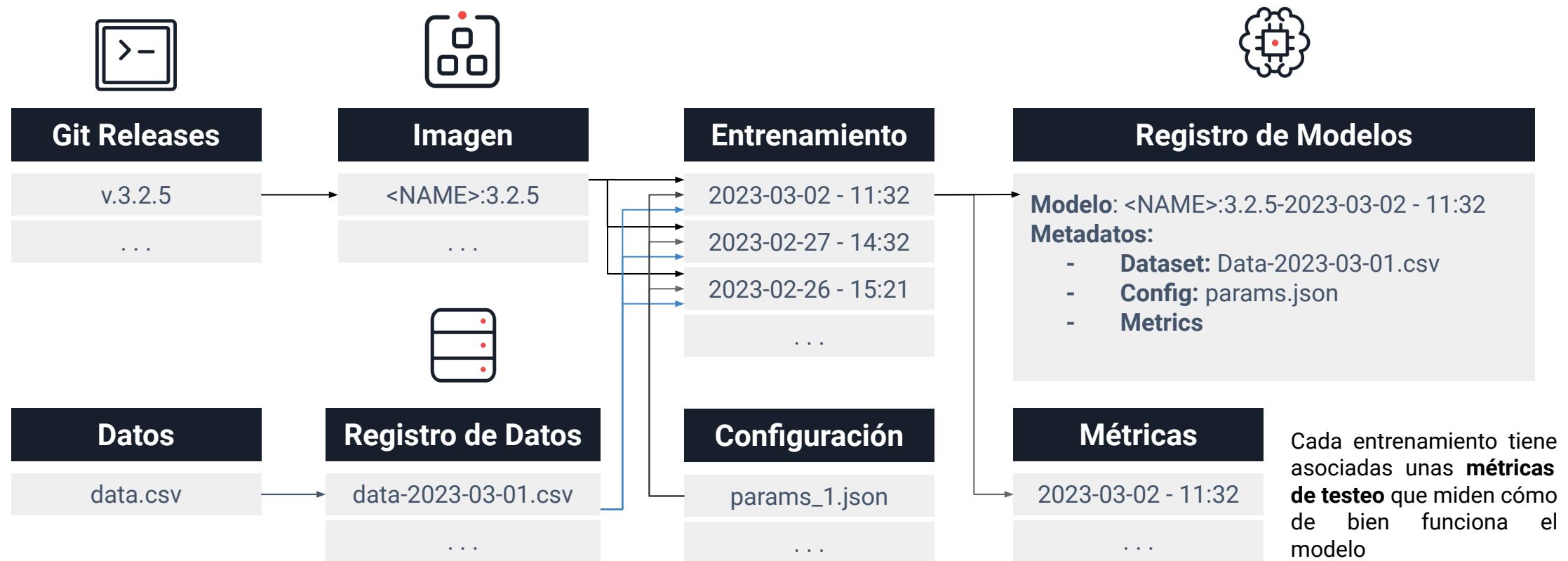
Versionado

Dado un proyecto con un naming concreto, el versionado hace referencia a la forma de nombrar nuevas versiones del mismo proyecto.

Generalmente se suele hacer siguiente el semantic versioning (p.e. 3.12.2).

¿Cómo quedaría todo junto?

Naming y versionado.



CI / CD

CI / CD

El CI/CD es el conjunto de técnicas procesos y herramientas que permiten desplegar de manera automática el proyecto

Continuous Integration

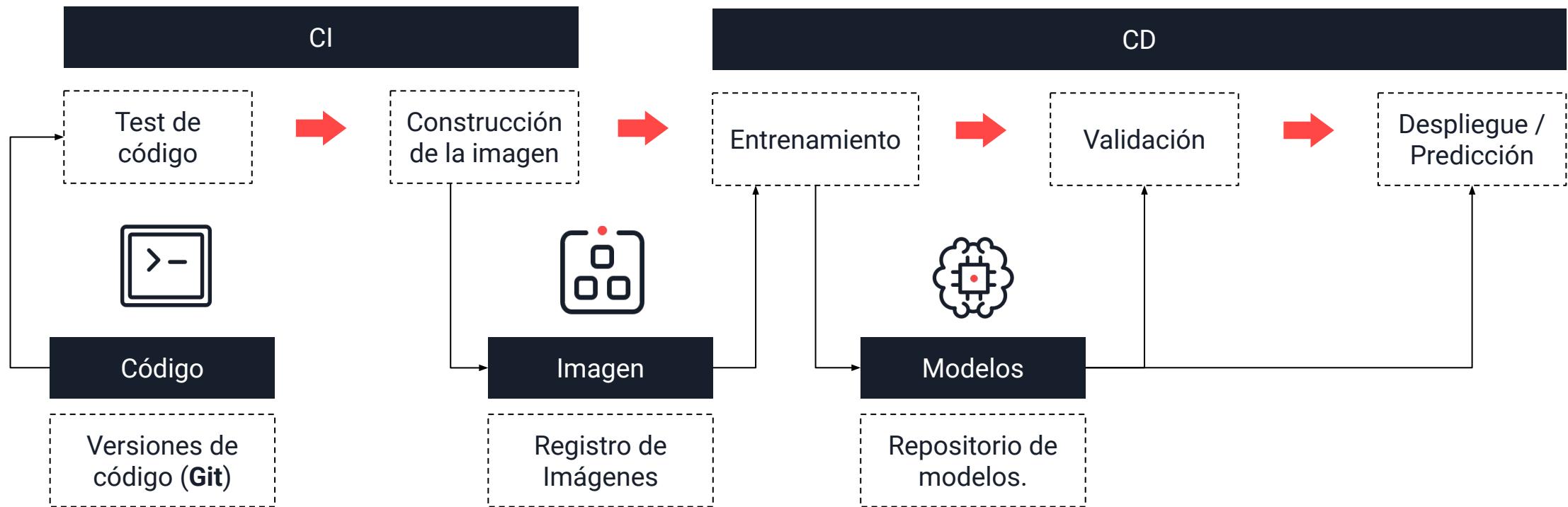
Automatizar el proceso de integración del proyecto en el entorno productivo (p.e. construir la imagen docker y llevarla a producción)

Continuous Deployment

Automatizar el proceso de ejecución del proyecto una vez integrado (p.e. ejecutar la imagen docker que se ha subido)

CI / CD

El proceso de integración y despliegue del artefacto docker se puede automatizar con el siguiente flujo.



Herramientas de CI/CD

Herramientas de CI/CD

Existen múltiples herramientas que permiten crear estos procesos de CI/CD. La mayoría de ellas se configuran a partir de un fichero YML donde se definen la secuencia de pasos a ejecutar durante el proceso de CI/CD.

Algunos ejemplos de ellas son: Jenkins, Code Build, Github Actions,...

```
# Ejemplo de pipeline de test de código.
name: Code Testing
on:
  push:
    branches:
      - main
jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      fail-fast: True
    steps:
      - uses: actions/checkout@v2
      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.x'
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          python -m pip install .
      - name: Test with unittest
        run: |
          python -m pytest test/
```

Herramientas de CI/CD

Existen múltiples herramientas que permiten crear estos procesos de CI/CD. La mayoría de ellas se configuran a partir de un fichero YML donde se definen la secuencia de pasos a ejecutar durante el proceso de CI/CD.

Algunos ejemplos de ellas son: Jenkins, Code Build, Github Actions,...

```
# Ejemplo de pipeline de construcción de imagen
name: Build and Push image
on:
  push:
    branches:
      - main
jobs:
  build:
    name: Build Image
    runs-on: ubuntu-latest
    steps:
      - name: Check out code
        uses: actions/checkout@v2
      - name: Configure AWS credentials
        uses:
          aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ap-south-1
      - name: Login to Amazon ECR
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v1
```

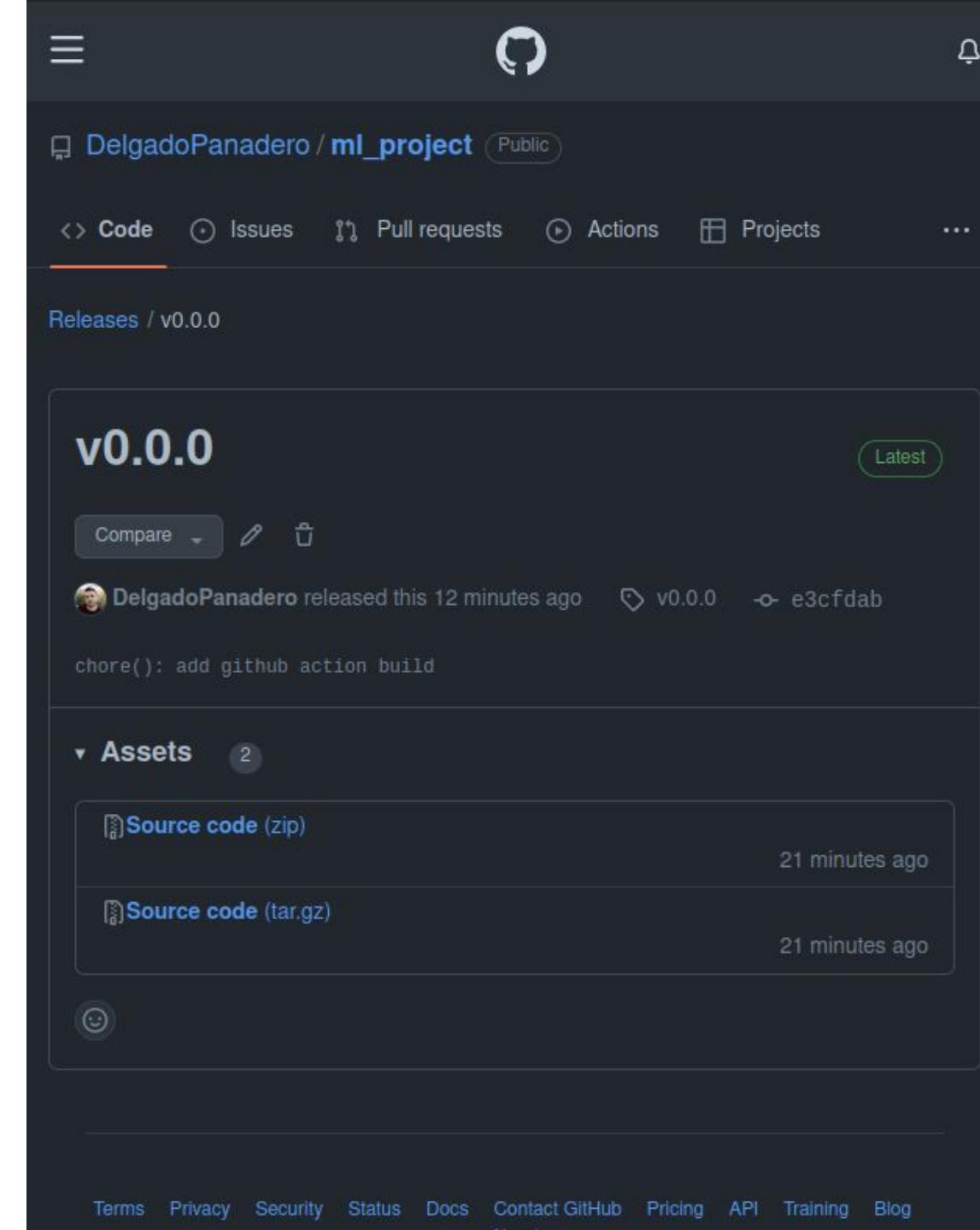
Release



Release

En git podemos crear releases a partir de un commit pusheado con un **tag**. Es conveniente usar el standard de **semantic versioning**.

Una **release** construye un código paquetizado del proyecto, sin embargo a nosotros nos interesa que sirva como **trigger** para ejecutar el CI.



Integración

Integración

Nuestro proceso de integración, en la mayoría de los casos tendrá únicamente dos procesos:

- testeo de software (ojo!! no testeo de modelo)
- construcción de la imagen docker

Get started with GitHub Actions

Build, test, and deploy your code. Make code reviews, branch

Skip this and [set up a workflow yourself →](#)

 Search workflows

Suggested for this repository

Docker image

By GitHub Actions



Build a Docker image to deploy, run, or push to a registry.

Configure

Dockerfile 

Django

By GitHub Actions



Build and Test a Django Project

Integración

¿Qué procesos no serían parte de un flujo de CI?

- Entrenamiento del modelo
- Testeo del modelo
- Procesos de ML en general

Estos procesos serían parte del proceso de CD

Get started with GitHub Actions

Build, test, and deploy your code. Make code reviews, branch

Skip this and [set up a workflow yourself →](#)

 Search workflows

Suggested for this repository

Docker image

By GitHub Actions



Build a Docker image to deploy, run, or push to a registry.

Configure

Dockerfile 

Django

By GitHub Actions

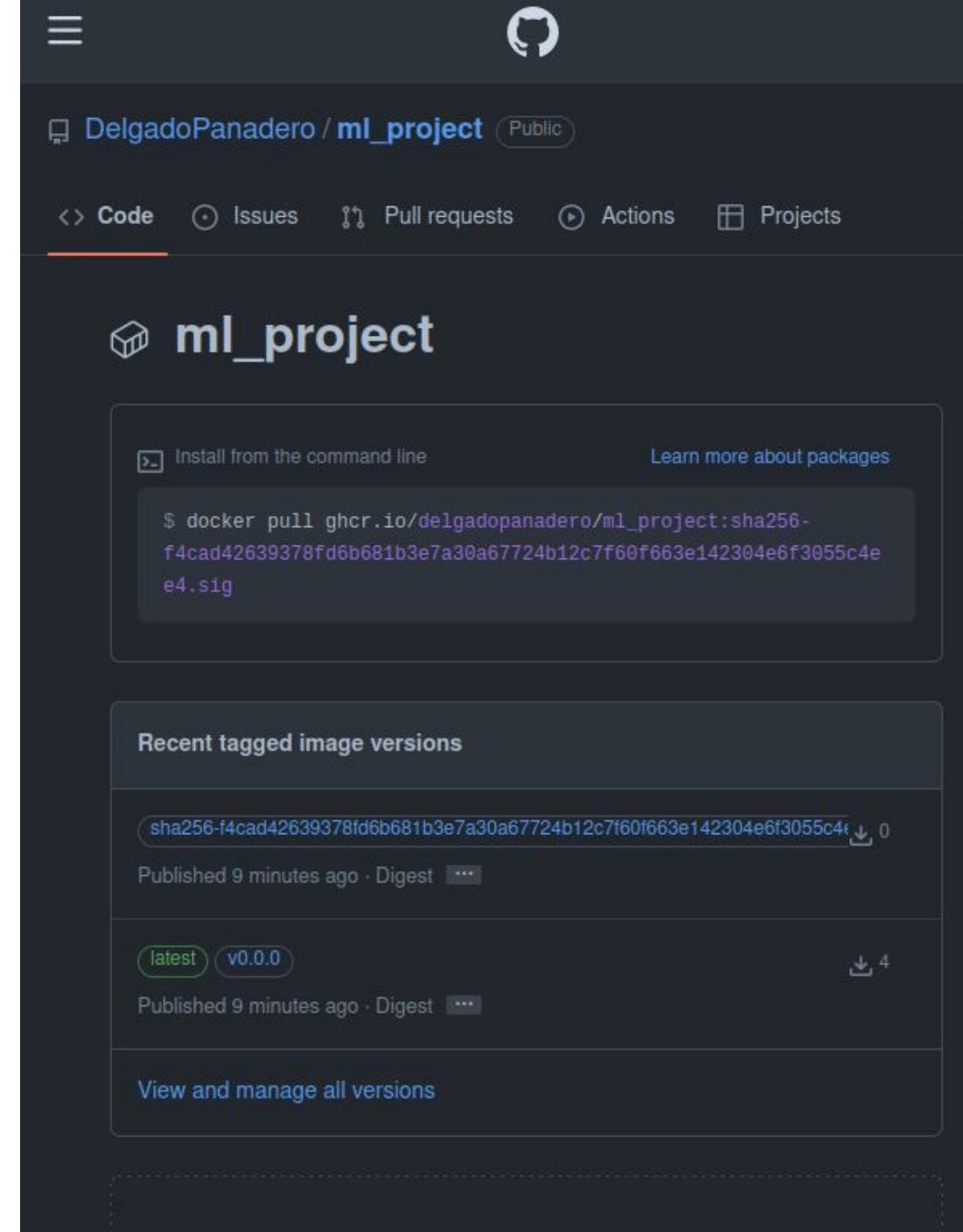


Build and Test a Django Project

Repositorio de imágenes

Repositorio de imágenes

El repositorio de imágenes sigue el mismo tagging que el software, esto nos permite no solo usar ese mismo tag a la hora de **elegir qué imágenes desplegamos**, sino también nos servirá para el artefacto de modelo entrenado.



The screenshot shows a GitHub repository page for 'ml_project'. At the top, there's a navigation bar with 'Code' (highlighted in orange), 'Issues', 'Pull requests', 'Actions', 'Projects', and a '...' button. The repository name 'DelgadoPanadero / ml_project' and its status as 'Public' are displayed. Below the navigation, the repository name 'ml_project' is shown next to a 3D cube icon. A section titled 'Install from the command line' contains a command-line snippet:

```
$ docker pull ghcr.io/delgadopanadero/ml_project:sha256-f4cad42639378fd6b681b3e7a30a67724b12c7f60f663e142304e6f3055c4ee4.sig
```

Below this, a section titled 'Recent tagged image versions' lists two entries:

- sha256-f4cad42639378fd6b681b3e7a30a67724b12c7f60f663e142304e6f3055c4e · Published 9 minutes ago · Digest · ...
- latest · v0.0.0 · Published 9 minutes ago · Digest · ...

At the bottom of this section is a link 'View and manage all versions'.

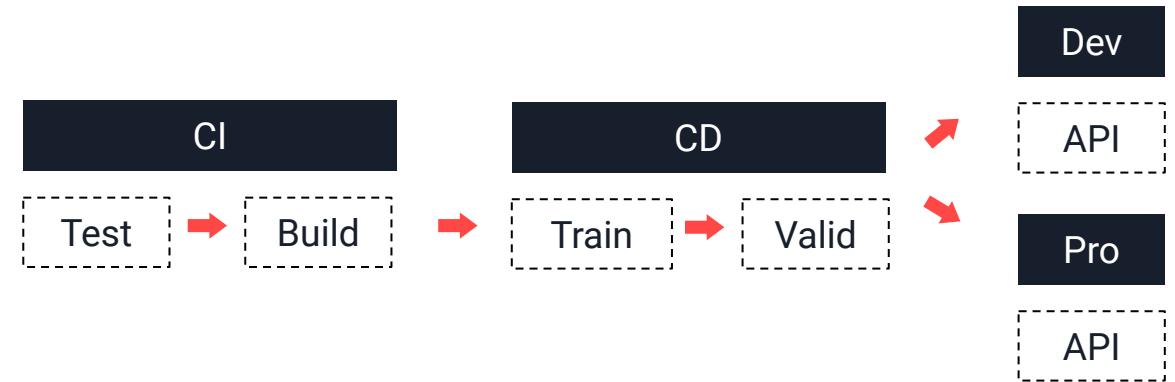
Paso entre entornos

Paso entre entornos

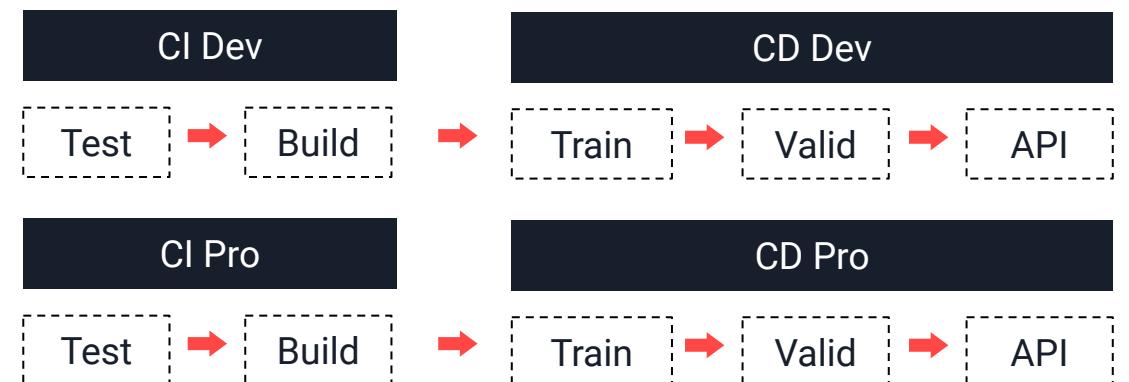
El paso entre entornos se puede hacer en un proyecto de Machine Learning de dos formas distintas:

- **A nivel de modelo:** Si queremos desplegar un modelo ya entrenado.
- **A nivel de código:** Si queremos desplegar el artefacto en un nuevo entorno y replicar todos los procesos.

A nivel de modelo



A nivel de código



Paso entre entornos

El paso entre entornos se puede hacer en un proyecto de Machine Learning de dos formas distintas:

- **A nivel de modelo:** Si queremos desplegar un modelo ya entrenado.
- **A nivel de código:** Si queremos desplegar el artefacto en un nuevo entorno y replicar todos los procesos.

A nivel de código

Si queremos priorizar que el modelo se entrene y genere con los datos de pro (con los que luego va a usar en el entorno real).

A nivel de modelo

Si queremos priorizar el hecho de poder llevar a producción un modelo que ya hemos visto qué funciona bien en Dev (recordemos que los entrenamientos tienen siempre una componente aleatoria, igual la próxima vez no funciona igual de bien)

Ejercicio de Docker.



Preguntas

1. Construir una imagen Docker a partir de ejercicio de la sección anterior, para ello la estructura del proyecto debería ser la siguiente.

```
project/  
  - data/  
  - test/  
  - src/  
  - model/  
  - train.py  
  - Dockerfile
```

2. Ejecutar el comando de entrenamiento del proyecto usando el comando **docker run <IMAGEN> python train.py**
3. Ejecutarlo de nuevo creando un volumen en la carpeta data/ y en la carpeta model/ para que lea los datos de la máquina anfitrión.

Conclusión.



Docker.

- Construir una imagen nos permite encapsular todo el código de ML y procesos como un “todo”.
- Además también nos permite ejecutar comandos de manera independiente y sin necesidad de depender de la plataforma.

Siguiente objetivo.

- Orquestar la secuencia de comandos a ejecutar
- Registrar el conjunto de archivos que se leen y escriben.

Pipelines

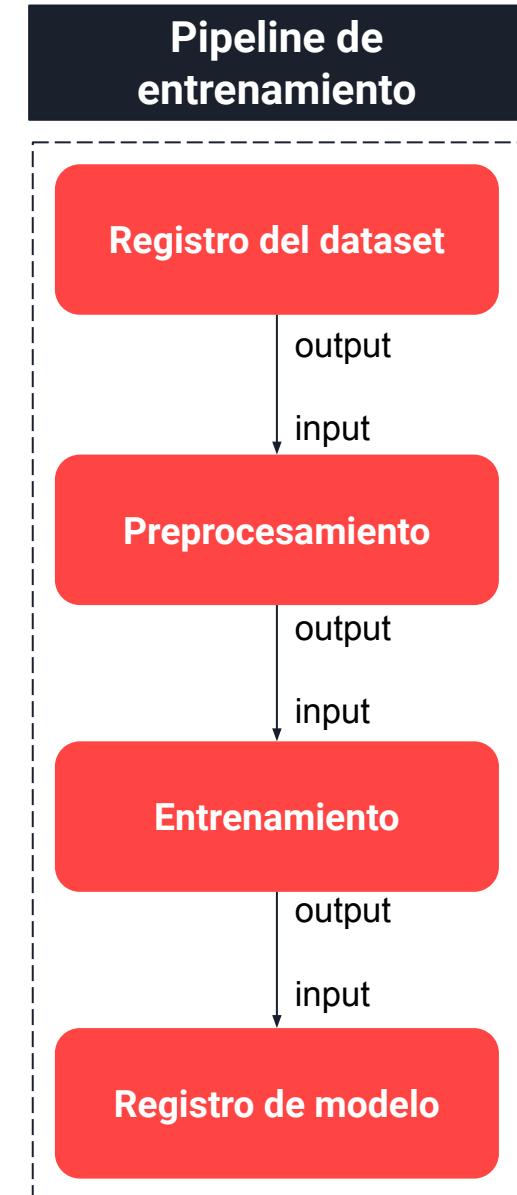


¿Qué es un pipeline?

¿Qué es un pipeline?

Generalmente, en un proyecto de Machine Learning, no se ejecutan cada uno de los procesos de manera individual e independiente sino que se ejecutan secuencias de pasos. Esta **secuencias de pasos es lo que se denomina Pipeline**.

Por ejemplo, un pipeline de Entrenamiento tendría que descargar los datos de entrenamiento, luego un preprocessamiento, la ejecución del entrenamiento, guardar el modelo en un repositorio,...

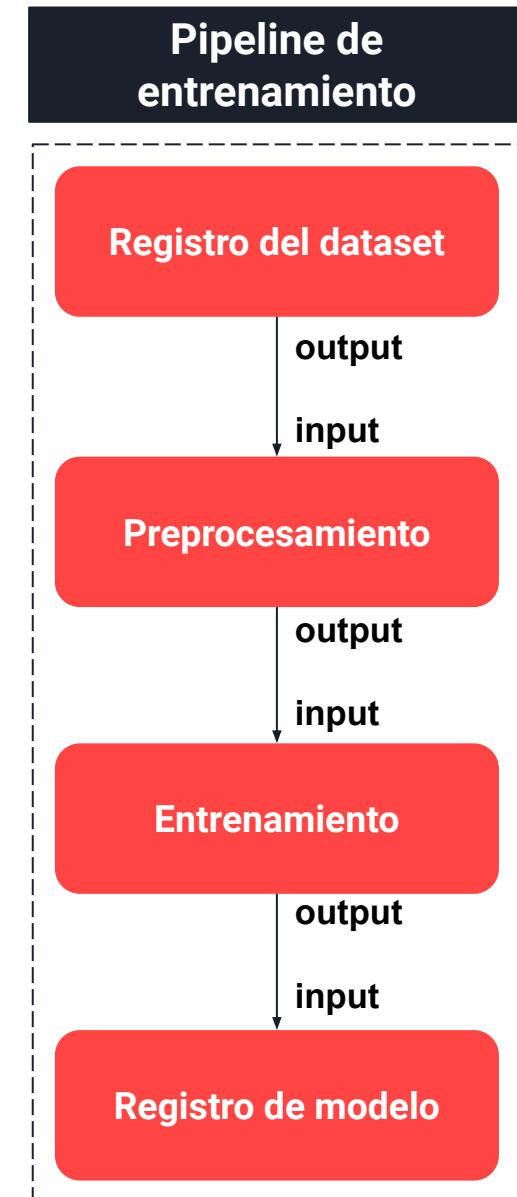


¿Cómo se relacionan los procesos?

¿Cómo se relacionan los procesos?

Estructura de grafo: Muchas veces los pipelines tienen una estructura secuencial, sin embargo hay veces que puede ser más compleja. Cada proceso tiene un parent, y puede tener uno o varios hijos.

Estado: Es importante que cada proceso pueda utilizar el resultado de los procesos anteriores, para ello tiene que estar accesible el estado del pipeline en cada ejecución de cada proceso



Airflow.



¿Qué es Airflow?



¿Cómo se relacionan los procesos?

Airflow es un orquestador de procesos, que nos permite definir una secuencia de ejecuciones (p.e. Docker run). Las ventajas que tiene son:

Automatización: Permite programar la lógica del flujo de ejecuciones, así como encadenar la salida de un proceso con la entrada del siguiente.

Monitorización: Los logs de la ejecución, así como los resultados, configuración, metadatos... quedan registrados de manera automática en cada ejecución.

DAGs

| All 26 | Active 10 | Paused 16 | Filter DAGs |
|---|-----------|-----------|-------------|
| DAG | Owner | Runs | i |
| example_bash_operator <small>example example2</small> | airflow | | |
| example_branch_dop_operator_v3 <small>example</small> | airflow | | |
| example_branch_operator <small>example example2</small> | airflow | | |
| example_complex <small>example example2 example3</small> | airflow | | |
| example_external_task_marker_child | airflow | | |
| example_external_task_marker_parent | airflow | | |
| example_kubernetes_executor <small>example example2</small> | airflow | | |
| example_kubernetes_executor_config <small>example3</small> | airflow | | |
| example_nested_branch_dag <small>example</small> | airflow | | |
| example_passing_params_via_test_command <small>example</small> | airflow | | |

Componentes de Airflow

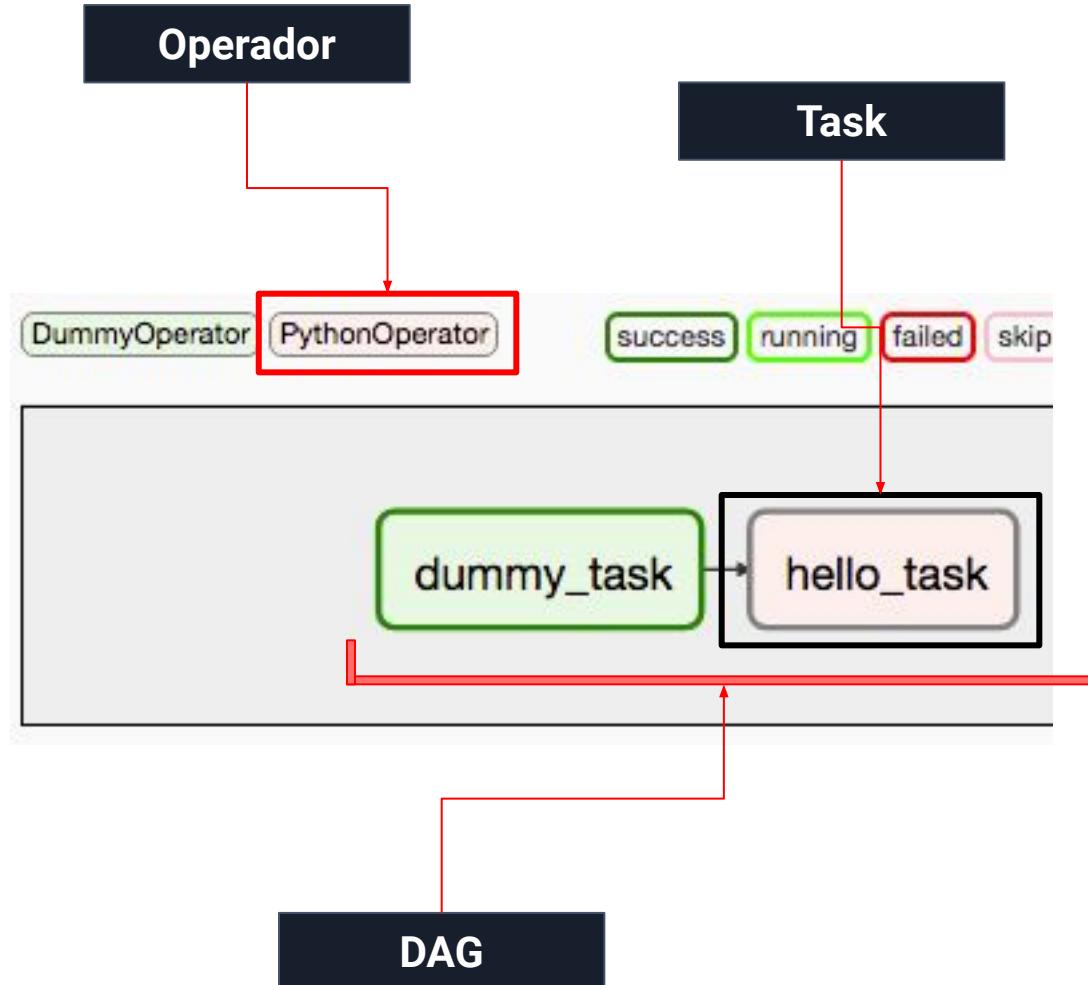
¿Cómo se relacionan los procesos?

Los componentes que tiene airflow son los siguientes:

DAG: Definición de una secuencia de ejecuciones siguiendo una estructura de Grafo Acíclico Directo.

Task: Cada uno de los nodos del un DAG. La unidad básica de ejecución dentro de Airflow.

Operator: Plantillas predefinidas para crear Tasks.



Y podemos crear contenedores Docker!

Docker con Airflow

Los componentes que tiene airflow son los siguientes:

DAG: Definición de una secuencia de ejecuciones siguiendo una estructura de Grafo Acíclico Directo.

Task: Cada uno de los nodos del un DAG. La unidad básica de ejecución dentro de Airflow.

Operator: Plantillas predefinidas para crear Tasks.

```
from docker.types import Mount
from plugins.operators import DockerOperator
```

```
t1 = DockerOperator(
    task_id="training_step",
    image="{{ params.image_name }}",
    auto_remove=True,
    entrypoint=[ "python", "train.py" ],
    mount_tmp_dir=True,
    mounts=[
        Mount(type="bind",
              source = input_dir
              target='/input/'),
        Mount(type="bind",
              source = target_dir
              target="/output/")
    ],
    environment={
        "LR" : "{{ params.param_lr }}",
        "DATA_PATH" : "/input/{{ params.dataset }}",
        "MODEL_PATH" : "/output/{{ params.model }}"
    }
)
```

Pipelines con Airflow.



¿Podemos crear Pipelines con Airflow?

¿Podemos crear Pipelines con Airflow?

Como una secuencia de ejecución de procesos Docker que se puede crear con un **DAG**.

Como hemos visto, cada uno de los procesos se podría ejecutar como un comando Docker independiente definiendo los task como **operator** de tipo DockerOperator.

Todos los procesos juntos se pueden orquestar usando Airflow.

```
from airflow import DAG
from docker.types import Mount
from plugins.operators import RegisterModel
from plugins.operators import DockerOperator
```

```
params = { ... }
```

```
with DAG('TrainingPipeline', params=params) as dag:
```

```
t1 = DockerOperator(  
    task_id="preprocess_step",  
    entrypoint=[ "python", "tranform.py" ],  
    ...)
```

```
t2 = DockerOperator(  
    task_id="training_step",  
    entrypoint=[ "python", "train.py" ],  
    ...)
```

```
t3 = RegisterModel(  
    task_id="register_step",  
    ...)
```

```
t1 >> t2 >> t3
```

¿Cómo crear los Pipelines con Airflow?

¿Cómo crear los Pipelines con Airflow?

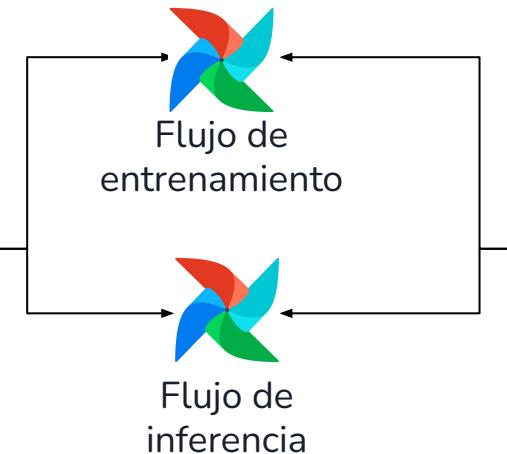
Separación entre la lógica de automatización de procesos y del código de Machine Learning.



Data Scientist



Código de modelo



MLOps

Crea imágenes Docker con sus modelos siguiendo el diseño que hemos visto

Crea flujos de Airflow que usan la imagen Docker con el modelo

¿Qué ventajas tiene usar Airflow?

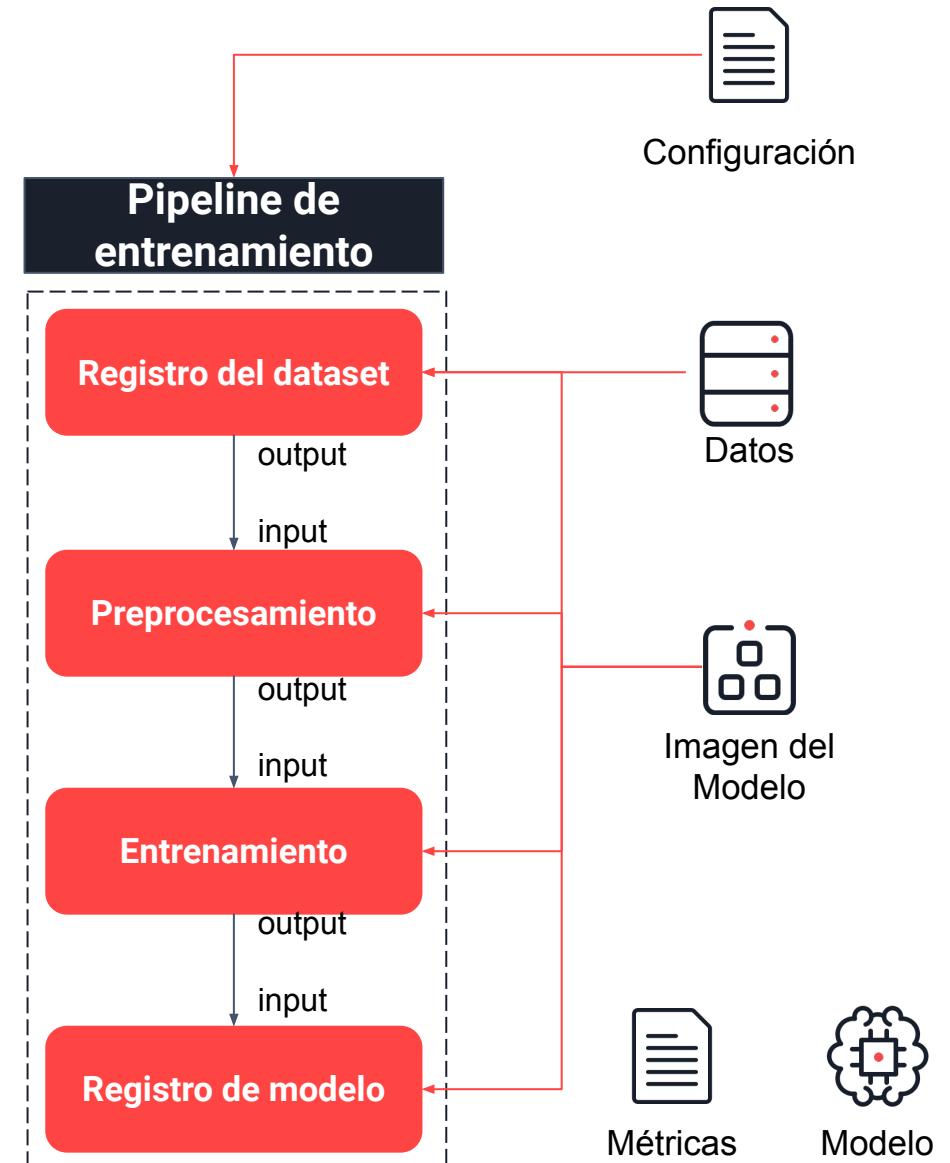
¿Qué ventajas tiene usar Airflow?

Las ventajas que ofrece Airflow para programar pipelines son:

Configuración: En cada ejecución se almacenan los parámetros con los que se ejecutó.

Experimentos: Los resultados de cada proceso se almacenan como un identificador único.

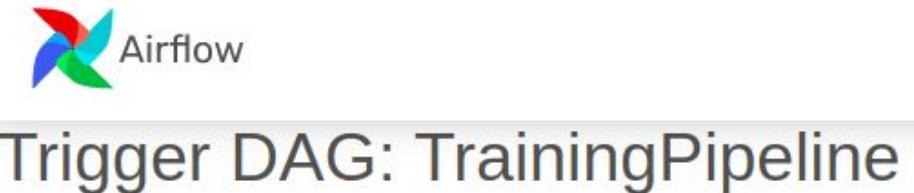
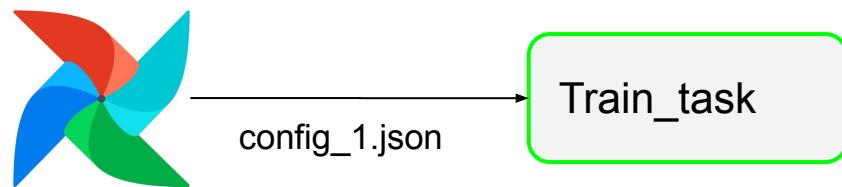
Métricas: Podemos almacenar las métricas asociadas a cada entrenamiento en los logs del DAG.



Configuración del Pipeline

Configuración del Pipeline

Cada vez que ejecutamos un pipeline, podemos pasarle una configuración específica con los parámetros de entrada que espera cada uno de los procesos de Machine Learning que va a ejecutar.



Airflow

Trigger DAG: TrainingPipeline

2023-04-03 19:13:00+00:00

Configuration JSON (Optional, must be a dict object)

```
1 {  
2     "image_name": "ml_project",  
3     "dataset_name": "iris.csv",  
4     "model_name": "ml_project.pkl",  
5     "project_dir": "/home/ml_project",  
6     "data_dir_path": "/home/ml_project/data",  
7     "model_dir_source": "/home/ml_project/model",  
8     "param_test_size": 0.2,  
9     "param_random_state": 0,  
10    "param_max_iter": 100  
11 }
```

To access configuration in your DAG use `{{ dag_run.conf }}`. As `core.dag_run_conf_overrides_params` is set to `True`, so passing any configuration here will override task params which can be accessed via `{{ params }}`.

Unpause DAG when triggered

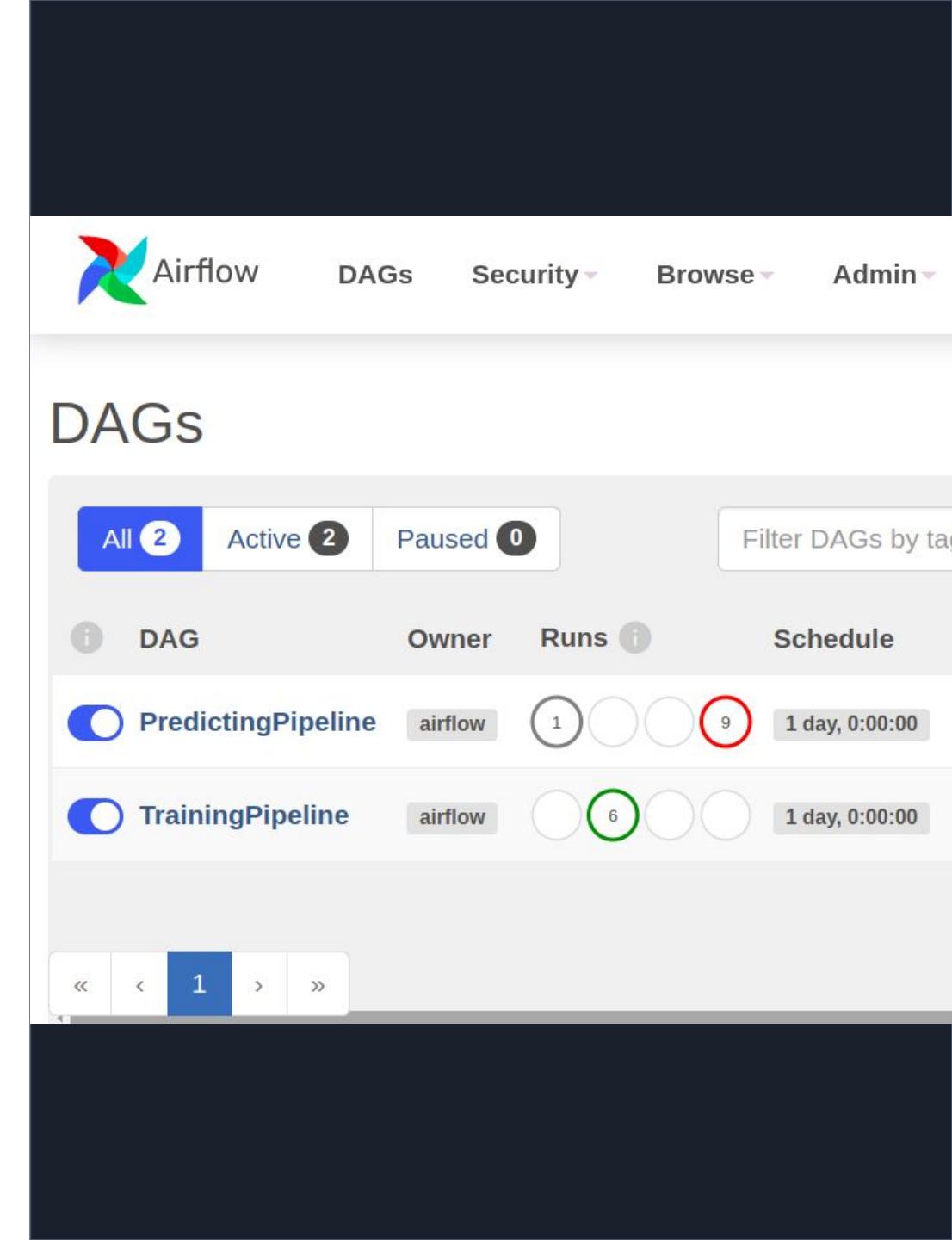
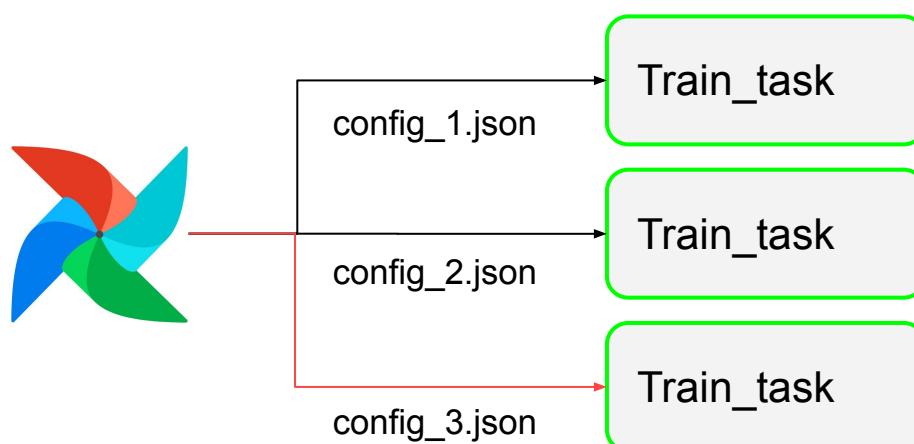
Trigger

Cancel

Ejecución de experimentos.

Ejecución de experimentos.

Con Airflow podemos ejecutar múltiples experimentos simultáneamente y ver el estado de cada uno de ellos en cada momento.



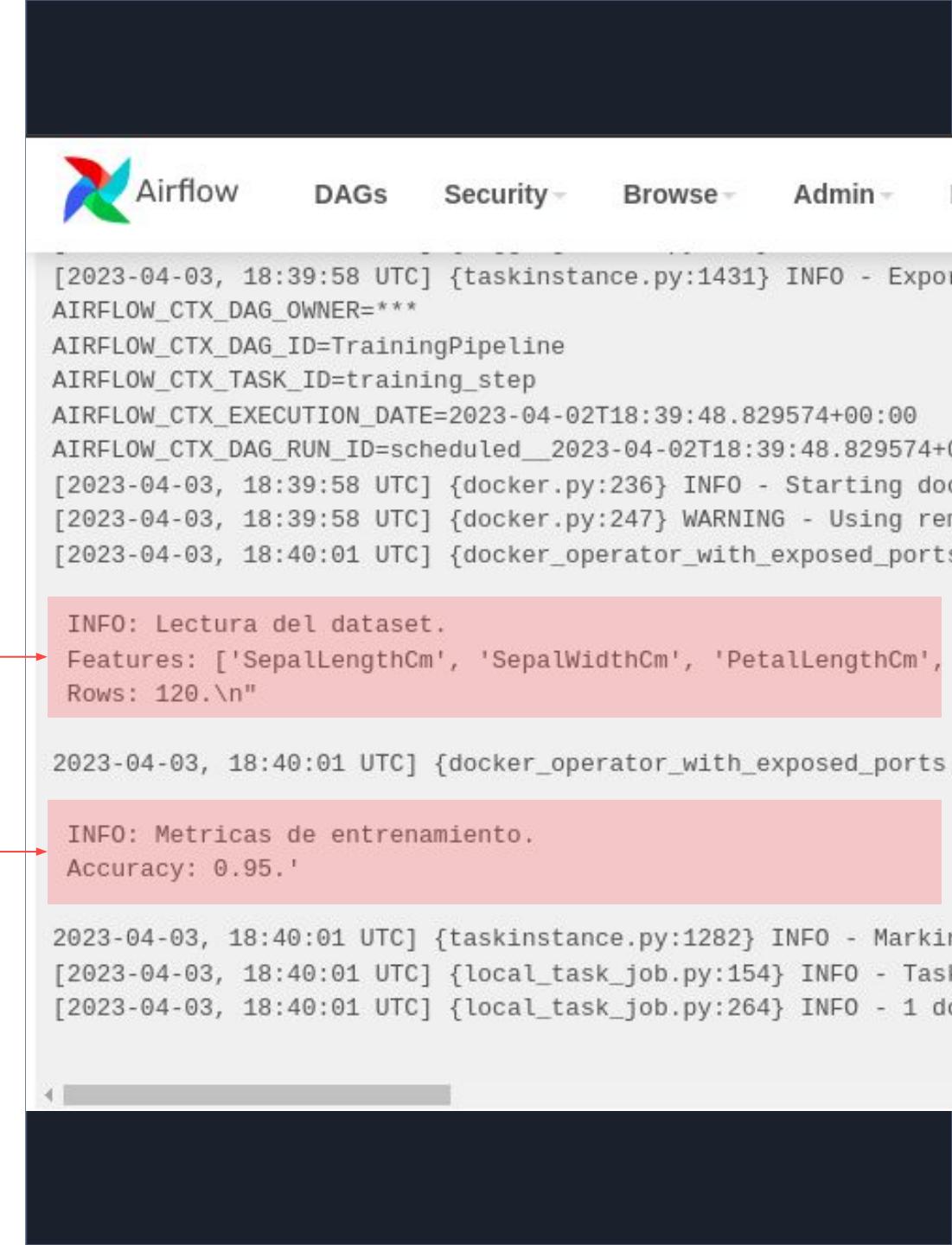
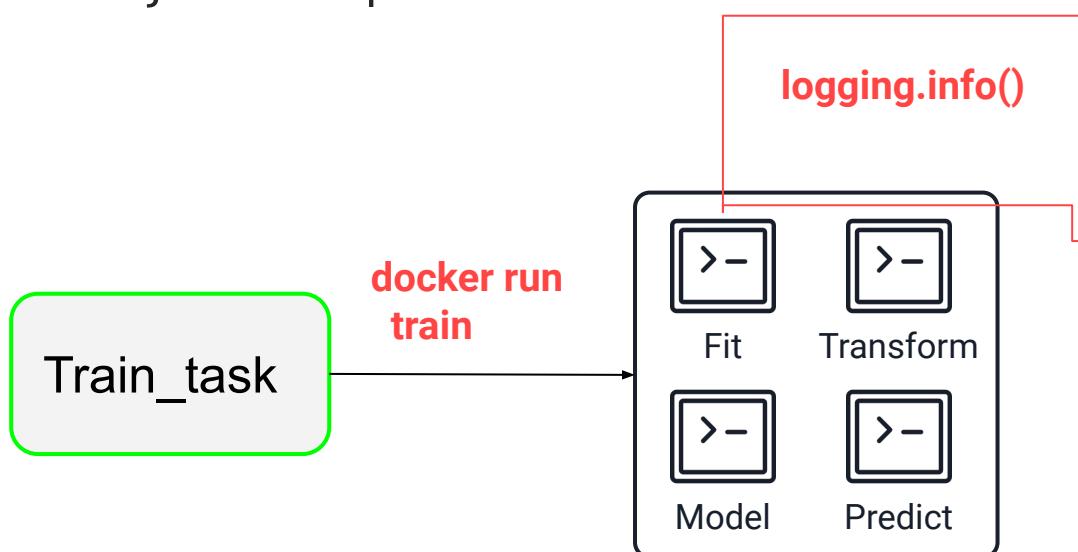
The screenshot shows the Airflow web interface with the following details:

- Header:** Airflow, DAGs, Security, Browse, Admin.
- Section:** DAGs
- Filter:** Filter DAGs by tag
- Status Buttons:** All (2), Active (2), Paused (0)
- DAG List:**
 - PredictingPipeline:** Owner: airflow, Runs: 1 (green circle), Last run: 1 day, 0:00:00. The number 9 is highlighted with a red circle.
 - TrainingPipeline:** Owner: airflow, Runs: 6 (green circle), Last run: 1 day, 0:00:00.
- Pagination:** <<, <, 1, >, >>

Monitorización del Pipeline.

Monitorización del Pipeline.

Podemos ver los logs de cada ejecución de cada proceso docker por medio de los logs de Airflow asociados al task donde se ejecutó el proceso.



A screenshot of the Airflow web interface showing the logs for a task named "training_step" in the "TrainingPipeline" DAG. The logs are displayed in a red-bordered box, indicating they are associated with the "logging.info()" call in the code. The logs show the process of reading a dataset and calculating accuracy.

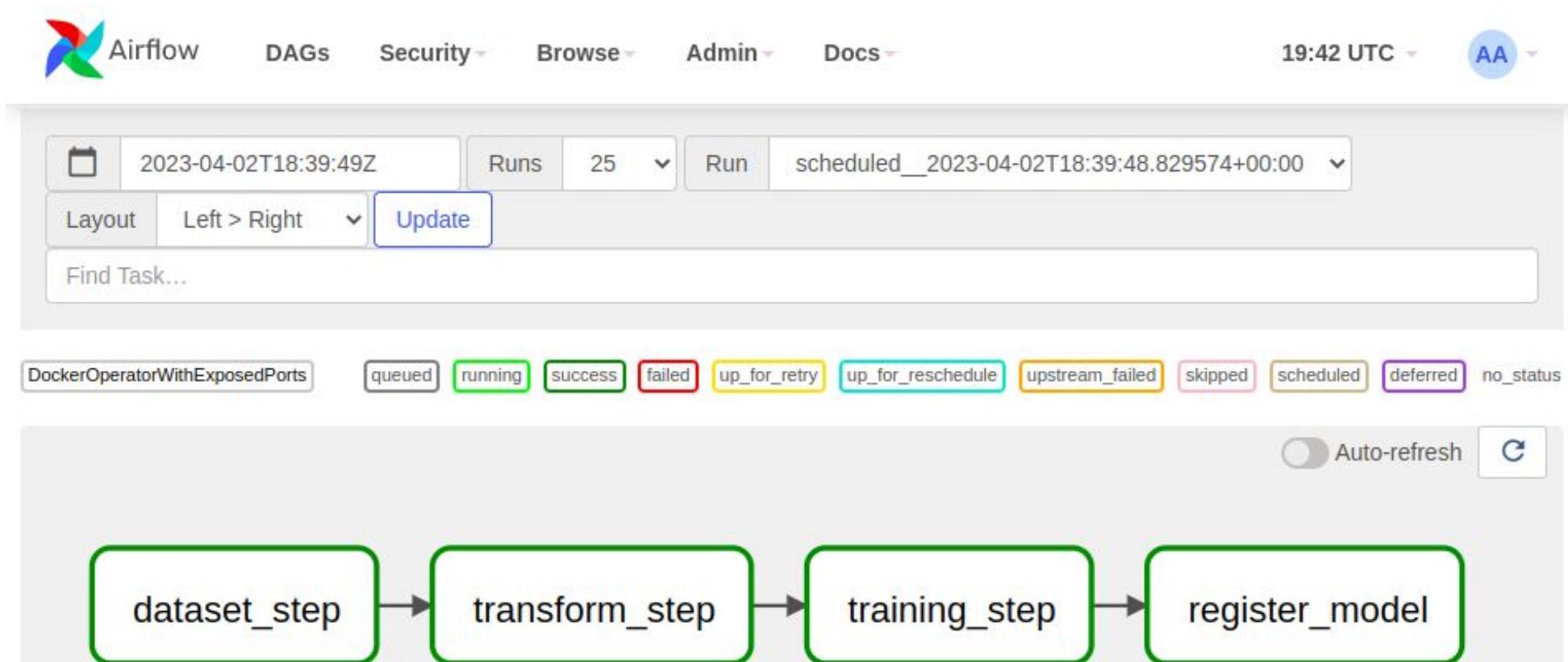
```
[2023-04-03, 18:39:58 UTC] {taskinstance.py:1431} INFO - Exporting context to Airflow_CTX_DAG_OWNER=***  
AIRFLOW_CTX_DAG_ID=TrainingPipeline  
AIRFLOW_CTX_TASK_ID=training_step  
AIRFLOW_CTX_EXECUTION_DATE=2023-04-02T18:39:48.829574+00:00  
AIRFLOW_CTX_DAG_RUN_ID=scheduled__2023-04-02T18:39:48.829574+00:00  
[2023-04-03, 18:39:58 UTC] {docker.py:236} INFO - Starting docker run  
[2023-04-03, 18:39:58 UTC] {docker.py:247} WARNING - Using remote Docker host  
[2023-04-03, 18:40:01 UTC] {docker_operator_with_exposed_ports}  
  
INFO: Lectura del dataset.  
Features: ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm',  
Rows: 120.\n]  
  
2023-04-03, 18:40:01 UTC] {docker_operator_with_exposed_ports}  
  
INFO: Metricas de entrenamiento.  
Accuracy: 0.95.'
```

Ejemplo Pipeline Entrenamiento.



Pipeline de Entrenamiento.

Un ejemplo del Pipeline de entrenamiento podría estar definido por las 4 tasks que se muestran a continuación.

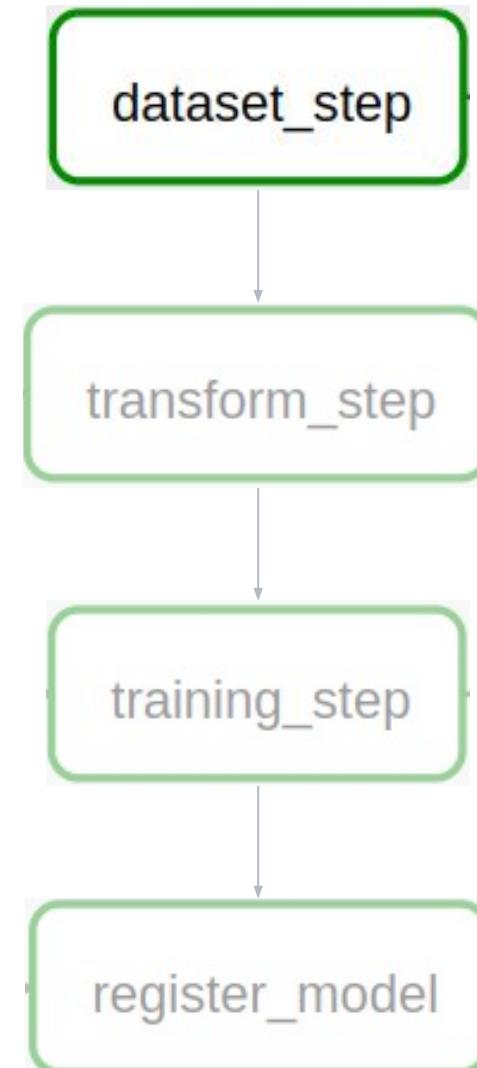


Pipeline de Entrenamiento.

La función de la Task “dataset_step” en el pipeline de entrenamiento es la siguiente:

Versionado de dataset: El objetivo de esta task del pipeline es la de guardar el dataset usado por el pipeline, en el momento de la ejecución.

Replicidad: Esto nos permite replicar los resultados del experimento aún en el caso de que el dato de origen cambie.

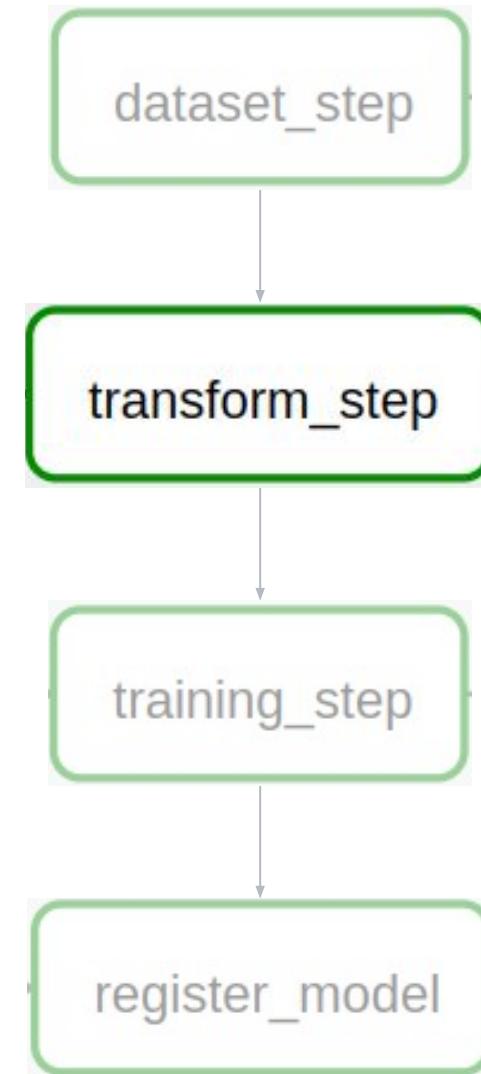


Pipeline de Entrenamiento.

La función de la Task “transform_step” en el pipeline de entrenamiento es la siguiente:

Registro del resultado: Hay veces que el proceso de transformación tiene un carácter aleatorio. Guardar los resultados permitirá auditar mejor el modelo entrenado.

Reutilización del resultado: Ejecutar el proceso de transformación como un proceso aparte del entrenamiento permite almacenar y reutilizar por otros modelos el resultado de la transformación.

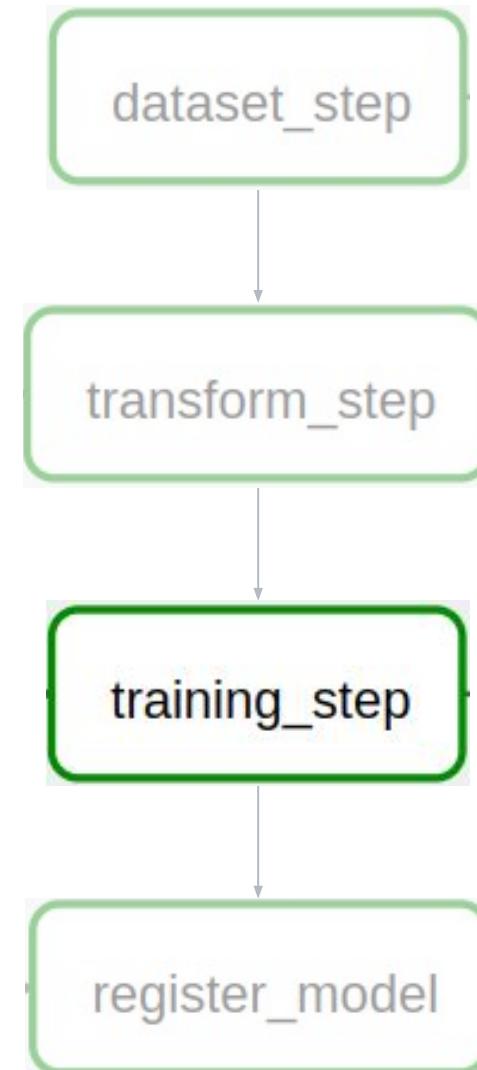


Pipeline de Entrenamiento.

La función de la Task “training_step” en el pipeline de entrenamiento es la siguiente:

Entrenamiento: Aunque el flujo entero sea de entrenamiento, la tarea “obligatoria” de este tipo de flujo es la que se corresponde con la ejecución del entrenamiento.

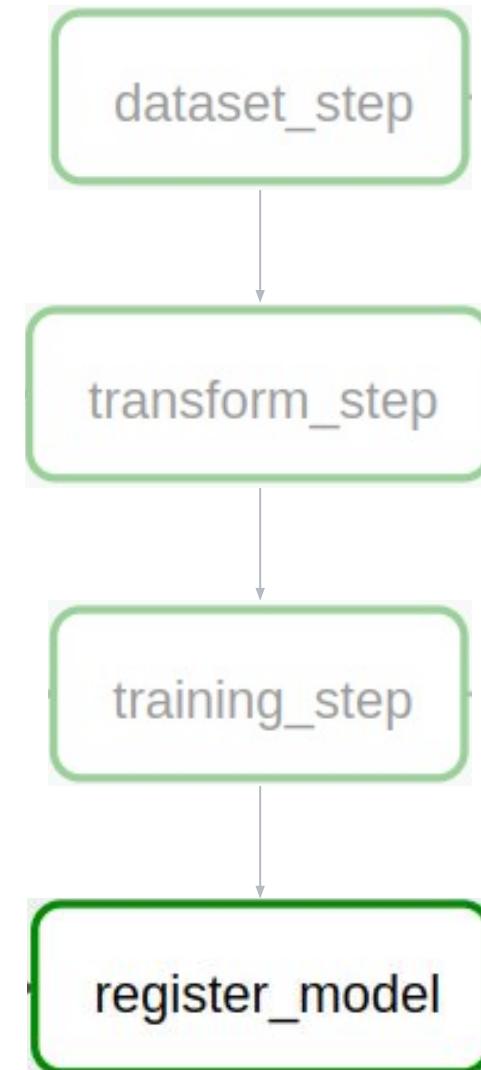
Validación: Hay veces que el proceso de validación se ejecuta como un step a parte. En nuestro caso lo hemos ejecutado en el mismo task que el script de procesamiento.



Pipeline de Entrenamiento.

La función de la Task “register_model” en el pipeline de entrenamiento es la siguiente:

Registro del modelo entrenado: Esta tarea guarda el resultado del modelo entrenado en el paso de entrenamiento junto con los metadatos de la ejecución, así como con los parámetros de configuración

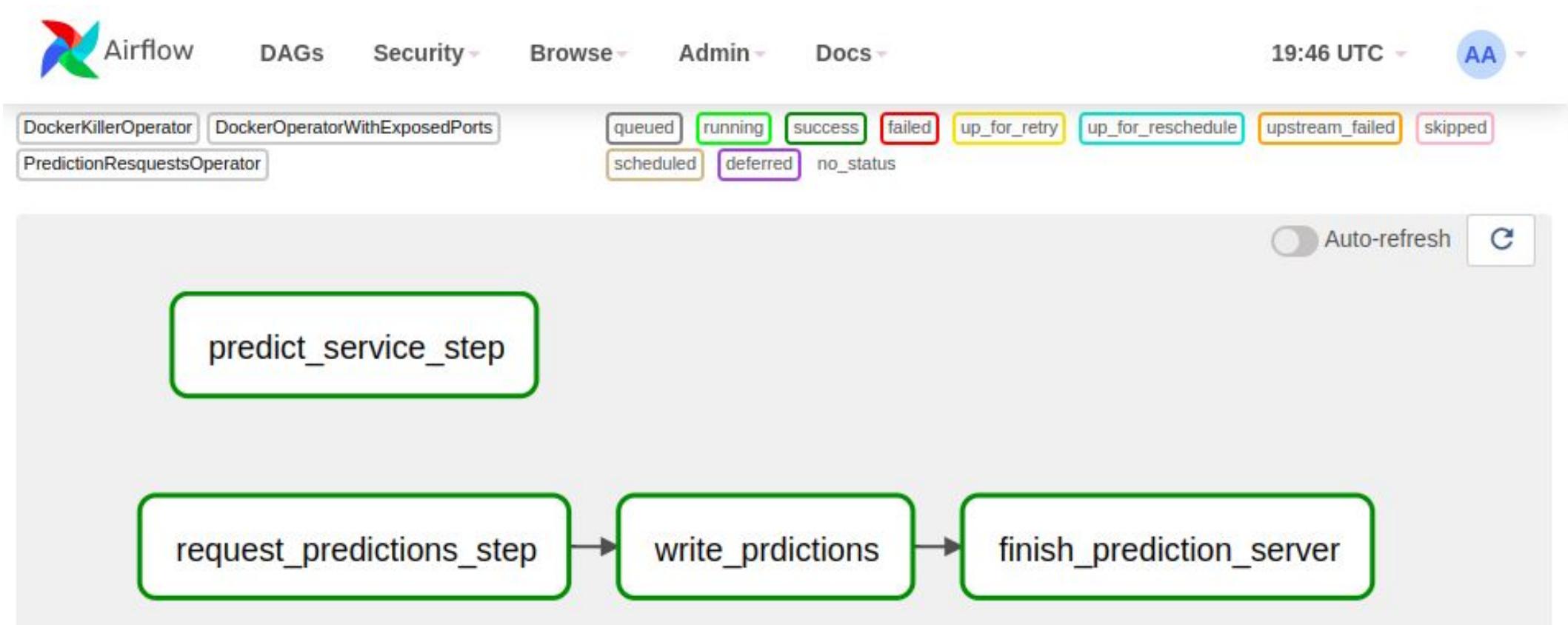


Ejemplo - Pipeline Predicción.



Pipeline de Predicción Batch.

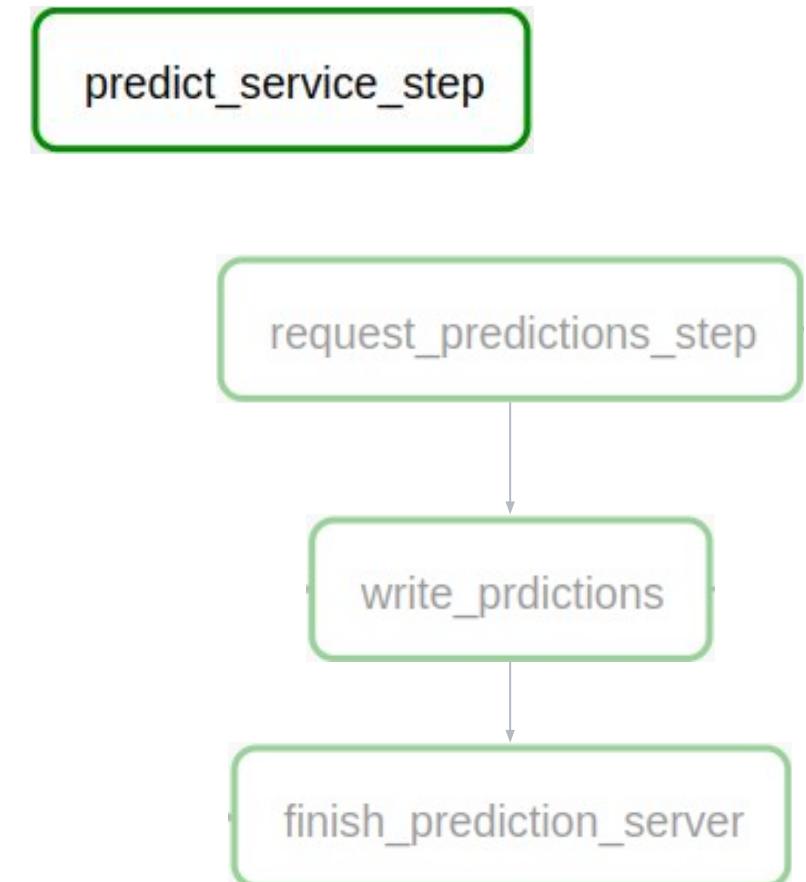
Un ejemplo del Pipeline de predicción podría estar definido por las 4 tasks que se muestran a continuación.



Pipeline de Predicción Batch.

La función de la Task “predict_service_step” en el pipeline de entrenamiento es la siguiente:

Levantar el servicio de predicción: Esta task levanta el servicio de predicción para que pueda ser usado por las task posteriores para construir las predicciones.



Pipeline de Predicción Batch.

Hacer peticiones de predicción: Dado un archivo de datos de entrada, realiza una petición para cada registro al servicio de predicción y las concatena en un archivo.

Escribir predicciones: Dado un archivo de datos de entrada, realiza una petición para cada registro al servicio de predicción y las concatena en un archivo.

Finalizar servidor : Esta task se encarga de finalizar el servicio de predicción una vez que se ha finalizado.

predict_service_step

request_predictions_step

write_predictions

finish_prediction_server

Ejercicio de Airflow.



Preguntas

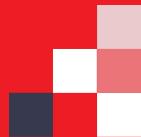
1. Ejecutar Airflow usando el archivo de docker-compose:

https://raw.githubusercontent.com/paradigmadigital/mentoring_mlops_airflow/main/airflow/docker-compose.yaml

2. Hacer un DAG de airflow que use la imagen Docker creada en ejercicio de la sección anterior y ejecute el proceso de entrenamiento. Esto será un pipeline de un solo paso.
3. Probar el ejecutar el proyecto:

https://github.com/paradigmadigital/mentoring_mlops_airflow

Sagemaker Pipelines



¿Qué es Sagemaker?

¿Qué es Sagemaker?

Sagemaker es el **servicio de AWS** de Machine Learning. A su vez está compuesto por muchos servicios como por ejemplo:

- **Sagemaker Notebooks:** Provisionamiento de notebook en cloud
- **Sagemaker Studio:** Interfaz de desarrollo y monitorización de modelos de ML
- **Sagemaker Pipelines:** Orquestación de flujos de ML
- Etc.

▼ Gobernanza

Panel de modelos NUEVO
Tarjetas de modelo NUEVO

► Ground Truth

► Bloc de notas

► Procesamiento

▼ Entrenamiento

Algoritmos
Trabajos de entrenamiento
Trabajos de ajuste de hiperparámetros

▼ Inferencia

Trabajos de compilación
Paquetes de modelos de Marketplace
Modelos
Configuraciones de punto de enlace
Puntos de enlace
Trabajos de transformación por lotes
Shadow tests

MACHINE LEARNING

Amazon S

Cree, entrene y despliegue modelos de machine learning de forma automática.

La forma más rápida y sencilla de llevar su modelo de concepción hasta producción.

Cómo funciona

¿Qué es Amazon SageMaker?

Amazon SageMaker proporciona los servicios necesarios para que los científicos de datos y desarrolladores creen y desplieguen modelos de ML de alta calidad.



Guía de incor

Comience a uti

¿Qué ventajas tiene Sagemaker?

¿Qué ventajas tiene Sagemaker?

Sencillez: Permite crear modelos y desplegar modelos de manera bastante sencilla desde su SDK.

Integración: Es compatible con muchos servicios de AWS (como AWS Lambda o AWS EMR).

Monitorización: Al ser un servicio desarrollado específicamente para Machine Learning, toda la monitorización y configuración está pensada para este tipo de procesos.

```
import sagemaker
from sagemaker.sklearn.estimator import SKLearn
session = sagemaker.Session()
sklearn = SKLearn(
    source_dir='./src',
    entry_point='train.py',
    framework_version='0.23-1',
    instance_type="ml.c4.xlarge",
    role=role,
    sagemaker_session=session,
    hyperparameters={
        "min_leaf_nodes": 3,
        "n_estimators": 10,
        "target": "Species"})
sklearn.fit(inputs={"train": S3_PATH})
predictor = sklearn.deploy(
    initial_instance_count=1,
    instance_type="ml.m5.xlarge")
response = predictor.predict(X_test)
```

¿Qué inconvenientes tiene Sagemaker?

¿Qué inconvenientes tiene Sagemaker?

Vendor lock-in: Al final los desarrollos de Machine Learning suelen estar hechos sobre el SDK de AWS (con Sagemaker Pipelines **podemos evitarlo**)

Curva de aprendizaje: Requiere conocer además de los algoritmos de Machine Learning, las características del Servicio de Sagemaker.

```
import sagemaker
from sagemaker.sklearn.estimator import SKLearn

session = sagemaker.Session()

sklearn = SKLearn(
    source_dir='./src',
    entry_point='train.py',
    framework_version='0.23-1',
    instance_type="ml.c4.xlarge",
    role=role,
    sagemaker_session=session,
    hyperparameters={
        "min_leaf_nodes": 3,
        "n_estimators": 10,
        "target": "Species"})
    
sklearn.fit(inputs={"train": S3_PATH})

predictor = sklearn.deploy(
    initial_instance_count=1,
    instance_type="ml.m5.xlarge"
)

response = predictor.predict(X_test)
```

¿Qué es Sagemaker Pipelines?

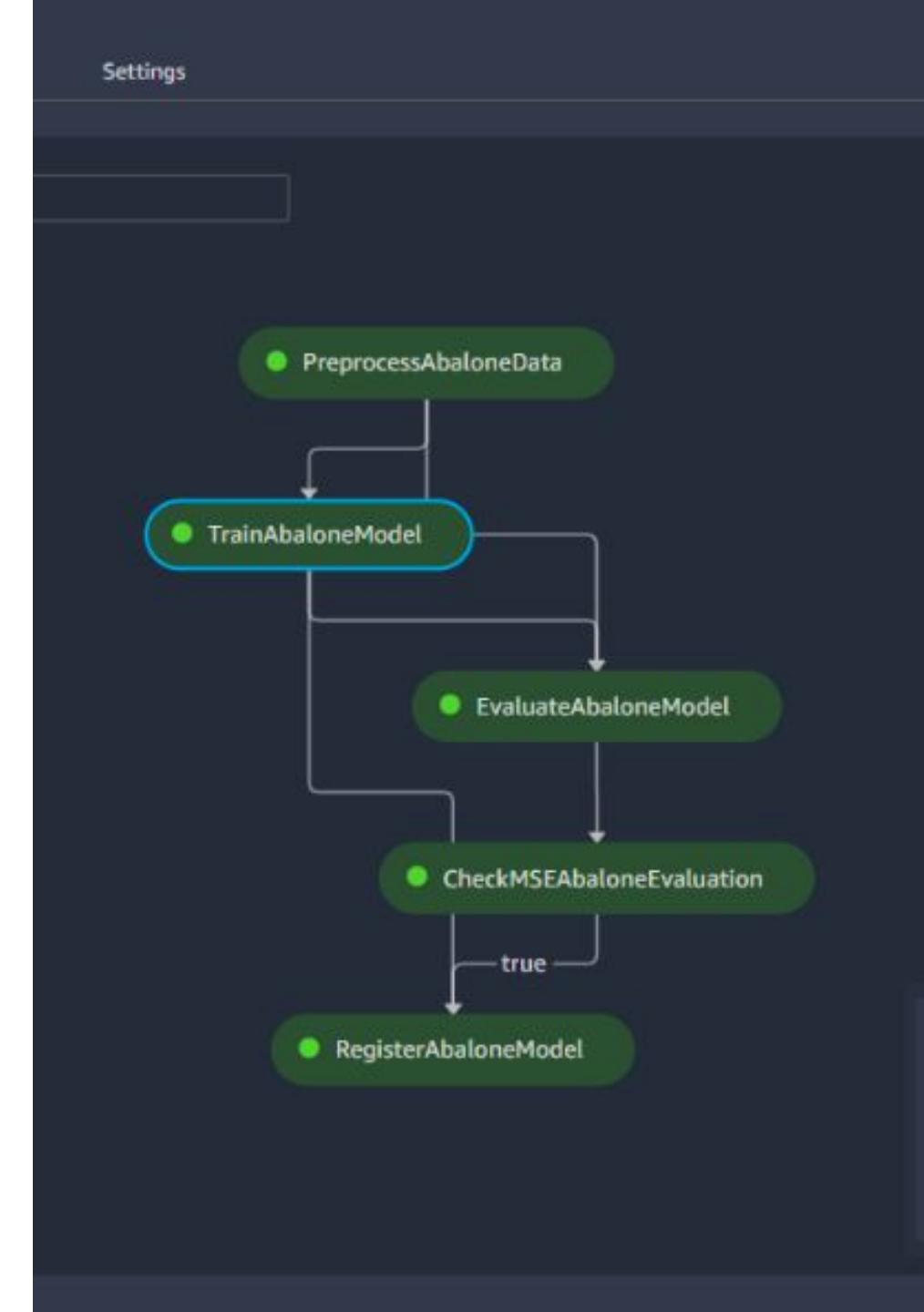
¿Qué es Sagemaker Pipelines?

Sagemaker Pipelines es un sistema de **orquestación de procesos** de Machine Learning usando como plataforma Sagemaker.

Permite ejecutar modelos Docker sin necesidad de tener que usar el SDK a nivel de código de ML. (**Evitamos vendor lock-in!**)

Documentación:

https://sagemaker.readthedocs.io/en/stable/amazon_sagemaker_model_building_pipeline.html



¿Cómo se crea un Pipeline de Sagemaker?

¿Cómo se crea un Pipeline de Sagemaker?

Basta con crear una imagen Docker, siguiendo las guías que marca la documentación de “Sagemaker Build your own image” y **usar el SDK** para crear un pipeline de Sagemaker.

El pipeline de Sagemaker está compuesto por **Steps de Sagemaker**. Estos Steps te permiten ejecutar procesos de entrenamiento e inferencia, así como acceder a otros servicios de AWS

Tipos de Steps de Sagemaker

- TrainingStep
- ProcessingStep
- TransformStep
- TuningStep
- AutoMLStep
- CreateModelStep
- LambdaStep
- CallbackStep
- QualityCheckStep
- ClarifyCheckStep
- EMRStep

¿Cómo se crea un Pipeline de Sagemaker?

Basta con crear una imagen Docker, siguiendo las guías que marca la documentación de “Sagemaker Build your own image” y **usar el SDK** para crear un pipeline de Sagemaker.

El pipeline de Sagemaker está compuesto por **Steps de Sagemaker**. Estos Steps te permiten ejecutar procesos de entrenamiento e inferencia, así como acceder a otros servicios de AWS

```
from sagemaker.model import Model
from sagemaker.estimator import Estimator
from sagemaker.workflow.pipeline import Pipeline
from sagemaker.workflow.steps import TrainingStep
from sagemaker.workflow.step_collections import CreateModelStep

step_train = TrainingStep(
    estimator = Estimator(
        image_uri=DOCKER_IMAGE
        ...
    )
    ...

step_create_model = CreateModelStep(
    model=Model(
        image_uri=DOCKER_IMAGE,
        ...
    )
    ...

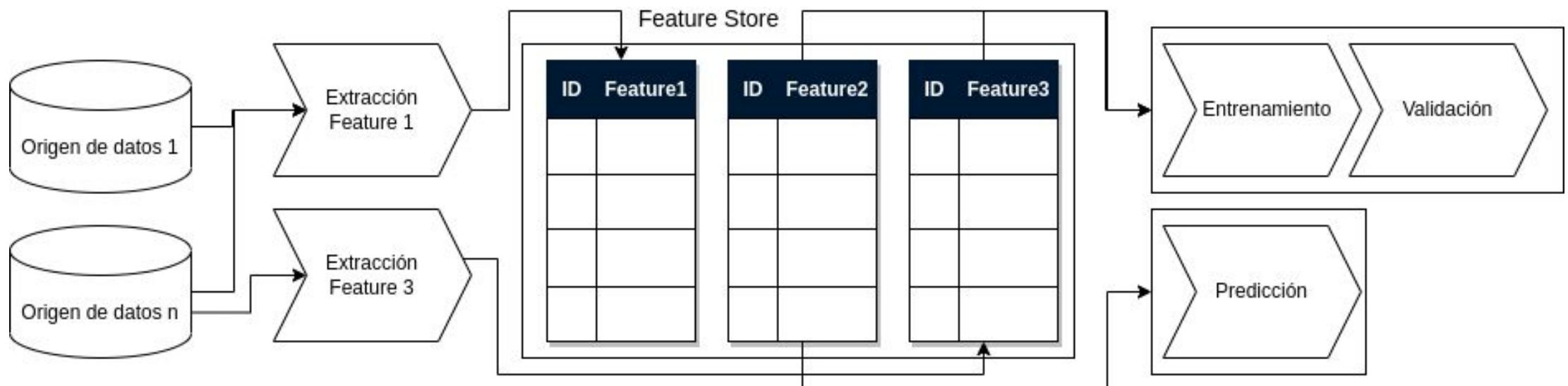
pipeline = Pipeline(
    name="SklearnIrisPipeline",
    steps=[
        step_train,
        step_create_model
    ]
)
```

Sagemaker FeatureStore

¿Qué es el FeatureStore?

El FeatureStore es un servicio que permite **preprocesar y almacenar las variables** que se van a utilizar en entrenamiento o inferencia de manera independiente.

Esto permite que cuando se ejecute el entrenamiento o inferencia ya estén precalculadas.



Sagemaker FeatureStore

Feature Group: Todas las características que se almacenan tienen que pertenecer a un grupo (esto permite juntar features para el mismo propósito)

Record Identifier Name: Cada registro de la feature tiene que tener un campo que se identifique como ID, esto permite extraer features usando el ID.

Online Store: Permite acceder a las features en tiempo real (es muy conveniente para predicciones online).

```
import sagemaker
from time import gmtime, strftime, sleep
from sagemaker.feature_store.feature_group import
FeatureGroup

current_timestamp = strftime(
    '%m-%d-%H-%M', gmtime())

iris_feature_group = FeatureGroup(
    f"iris-features-{current_timestamp}", session)

iris_feature_group.load_feature_definitions(
    data_frame=iris)

iris_feature_group.create(
    s3_uri=S3_URI,
    record_identifier_name='id',
    event_time_feature_name='event_time',
    role_arn=role,
    enable_online_store=True)

iris_feature_group.ingest(data_frame=iris_df)
```

Sagemaker ModelRegistry

Sagemaker ModelRegistry

El Model Registry es un servicio de AWS que dado un modelo entrenado, permite registrarlo junto con su imagen de ejecución y sus metadatos de creación (datos con los que se entrenó e id de ejecución).

El Model Registry es una forma de gobernar los modelos de ML, así como crear **nuevas versiones** de un mismo modelo sin que sean tratados como modelos diferentes.

```
from sagemaker.workflow.step_collections import
RegisterModel

register_step = RegisterModel(
    name="RegistroModelo",
    model=model,
    image_uri=IMAGE_URI,
    content_types=["text/csv"],
    response_types=["text/csv"],
    inference_instances=["ml.t2.medium"],
    transform_instances=["ml.m5.xlarge"],
    model_package_group_name="SklearnIris",
    approval_status="Approved",
)
```

Gracias!

