ESPECIALIZACIÓN EN MACHINE LEARNING ENGINEER

Tema: Fundamentos de MLE – parte 1

Docente: Daniel Chavez Gallo

Perfil profesional

- Titulado en Ingeniería Estadística e Informática (UNALM / Universidad Agraria la Molina)
- Maestría en Data Science (ITAM – URP)
- Master en Diseño y desarrollo de productos y servicios de Inteligencia Artificial (MIT)
- Diplomado en businnes analytics (PUCC – UPC)

Experiencia

Ing. Daniel Chávez Gallo

Data Scientist leader

10 años de experiencia profesional

REGLAS

Se requiere **puntualidad** para un mejor desarrollo del curso.

Para una mayor concentración **mantener silenciado el micrófono** durante la sesión.

Las preguntas se realizarán **a través del chat** y en caso de que lo requieran **podrán activar el micrófono**.

Realizar las actividades y/o tareas encomendadas en **los plazos determinados**.

Identificarse en la sala Zoom con el primer nombre y primer apellido.

Evaluación

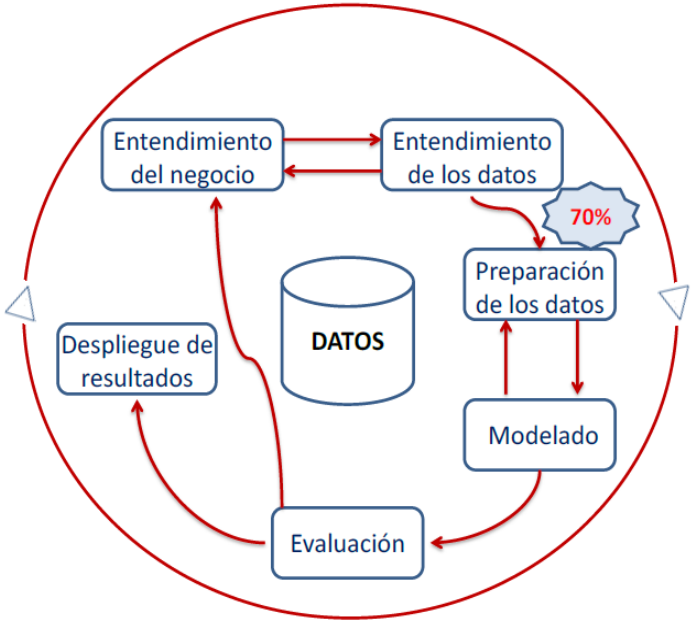
Participación por sesión	25%	1 pto por pregunta/respuesta en clase
Práctica calificada / parcial	25 %	Termino de la segunda clase
Trabajo final	50 %	Fecha de entrega: Una semana después de la tercera clase

Metodología CRISP : Historia

La metodología CRISP-DM, que significa **Cross-Industry Standard Process for Data Mining**, fue desarrollada en **1996** por un consorcio de empresas, incluyendo NCR, Daimler-Benz (ahora parte de Daimler AG), y SPSS Inc. (ahora parte de IBM).

El objetivo era crear un **estándar universal** que pudiera ser aplicado en diferentes industrias para proyectos de data mining. CRISP-DM se convirtió rápidamente en la metodología líder para **proyectos de minería de datos** debido a su **enfoque estructurado y fácil de seguir.**

A pesar de que la metodología no ha sido oficialmente actualizada desde la versión 1.0, su **marco conceptual sigue siendo ampliamente utilizado** y considerado como un estándar de facto en la industria.



Metodología CRISP: Etapas

Comprensión del Negocio

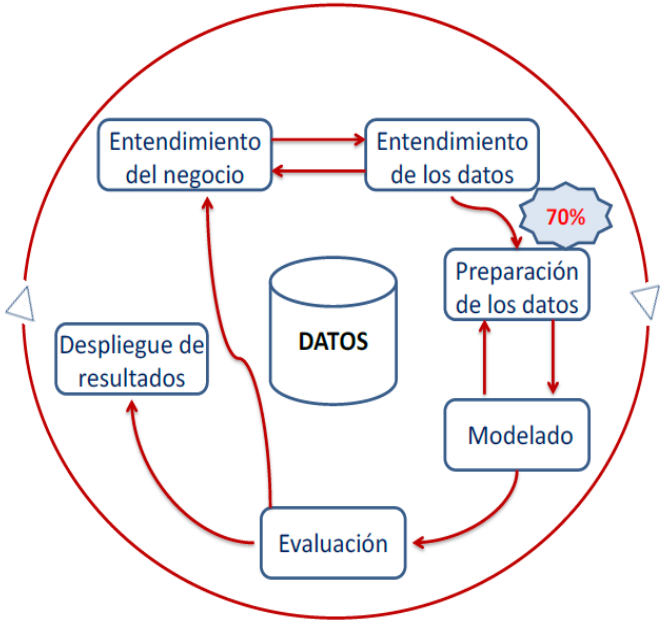
- Objetivos del negocio y requisitos desde una perspectiva de análisis de datos.
- Convertir estos objetivos en definiciones de problemas de minería de datos/Machine Learning.
- Establecer criterios de éxito.

Comprensión de los datos

- Recolección inicial de datos.
- Describir los datos, formular hipótesis, explorar la calidad de los datos.
- Identificar problemas de calidad de los datos y necesidades de preprocesamiento.

Preparación de los Datos

- Limpieza de datos, selección de características, transformaciones, creación de nuevas variables.
- Técnicas y herramientas para la preparación de datos.



Metodología CRISP: Etapas

Modelado

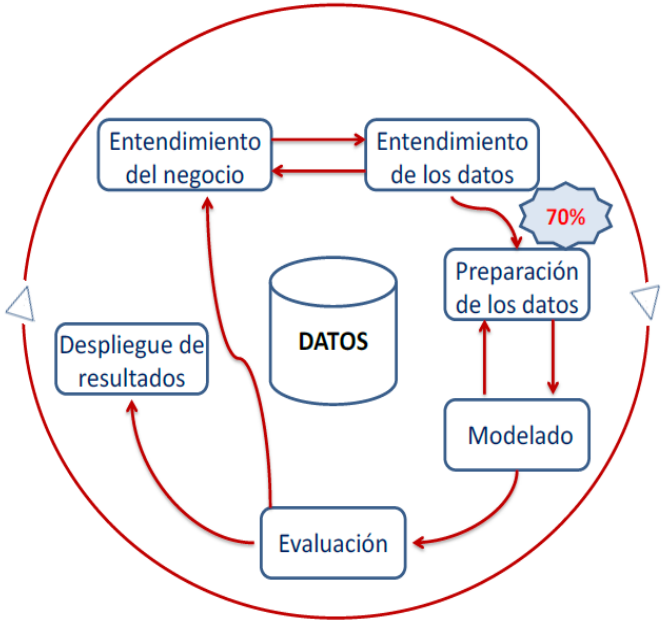
- Selección de técnicas de modelado adecuadas para el problema en cuestión.
- Diseño de pruebas y validación de modelos.
- Construcción de modelos y evaluación de su desempeño.

Evaluación

- Evaluación de los modelos en el contexto de los objetivos del negocio.
- Decisión sobre el despliegue del modelo.

Despliegue

- Planificación del despliegue.
- Monitoreo y mantenimiento de modelos en producción.
- Medición del rendimiento del modelo y feedback para futuras iteraciones.



Metodología CRISP: Evolución de DM a ML

Data Mining

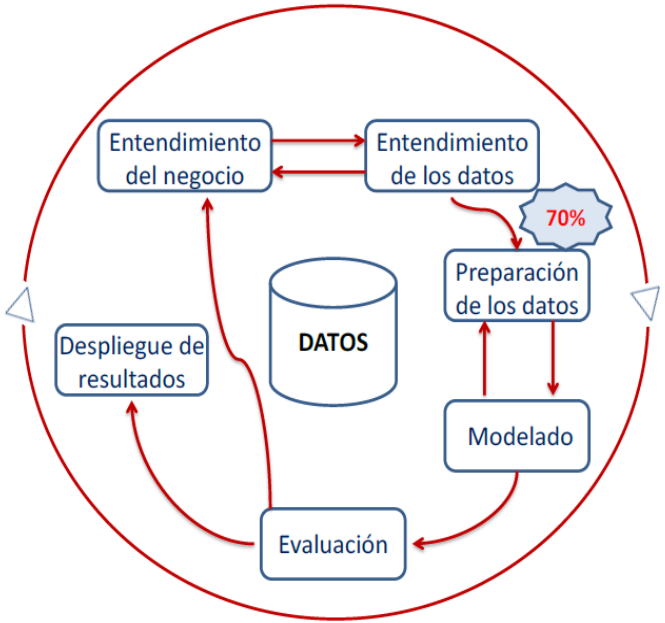
Es el proceso de **explorar grandes volúmenes de datos** para identificar patrones y relaciones significativas, validarlos y aplicarlos a nuevos datos. Su **objetivo** es **extraer información útil y convertirla en conocimiento comprensible para su aplicación**. Se centra en el análisis exploratorio y la extracción de valor de los datos.

Machine Learning

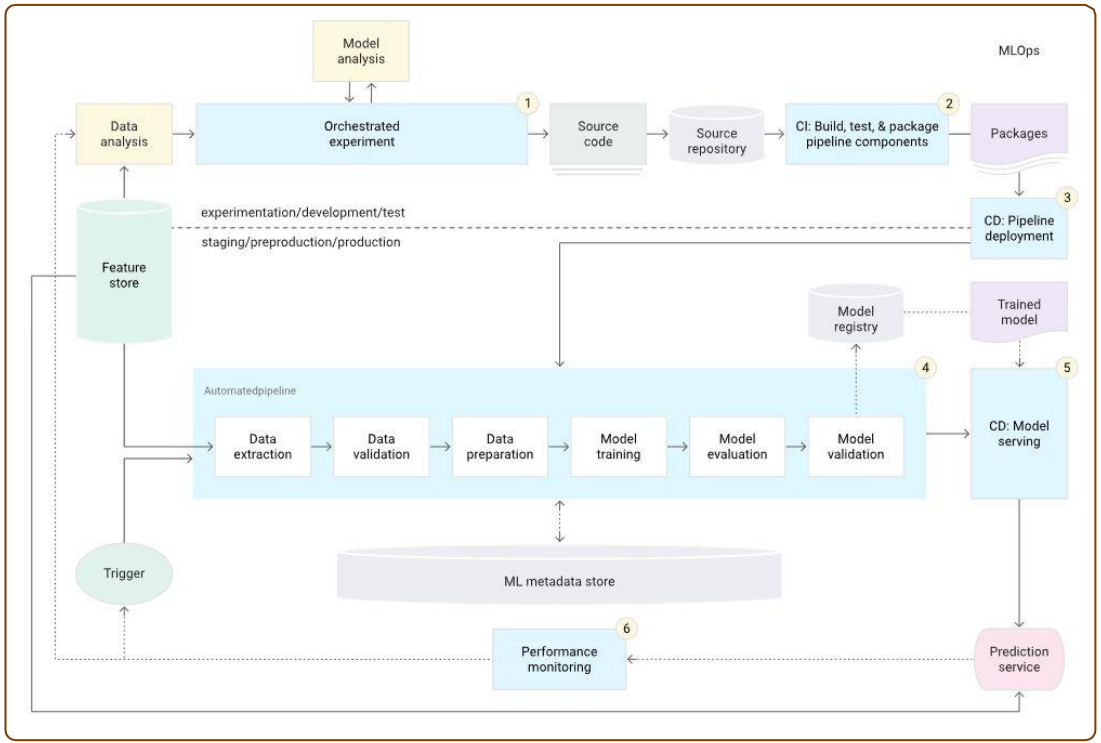
Es una **rama de la inteligencia artificial** que desarrolla algoritmos y modelos capaces de aprender de los datos para **hacer predicciones o tomar decisiones sin programación explícita**. A diferencia del Data Mining, se enfoca en crear sistemas que aprendan y mejoren de forma autónoma.

Las técnicas de **Machine Learning han ampliado las capacidades del Data Mining al permitir no solo extraer patrones, sino también predecir con mayor precisión**. Con algoritmos que aprenden de datos históricos y mejoran continuamente, ML incorpora avances como el **aprendizaje profundo**, que analiza grandes volúmenes de datos estructurados y no estructurados, automatizando y optimizando tareas complejas.

Metodología CRISP: Evolución de DM a ML



Framework de MLOps



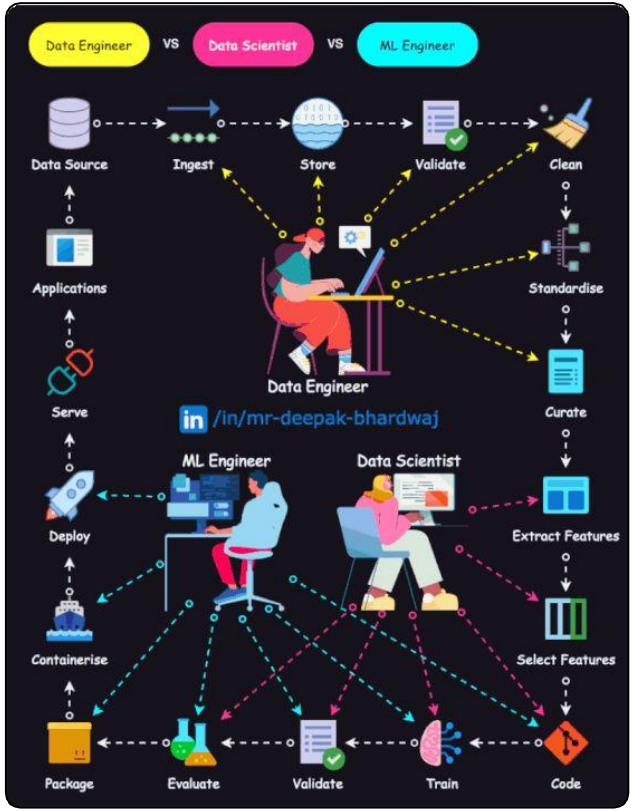
Data Engineer vs Data Science vs ML Engineer

Data Engineer

Un Data Engineer es un profesional especializado en **preparar la "infraestructura"** para el análisis de datos. Esto implica **diseñar, construir, y mantener la arquitectura de datos**, como bases de datos y grandes sistemas de procesamiento de datos.

Los ingenieros de datos se centran en la creación y mantenimiento de sistemas de datos que permitan recopilar, almacenar, acceder y analizar datos a **gran escala.** Usualmente trabajan con grandes sets de datos (big data) y con tecnologías como **Hadoop, Spark, Kafka, SQL**, y sistemas de **bases de datos NoSQL**.

Su trabajo es fundamental para asegurar que los **datos estén disponibles, precisos, y accesibles** para los científicos de datos y otros usuarios finales.



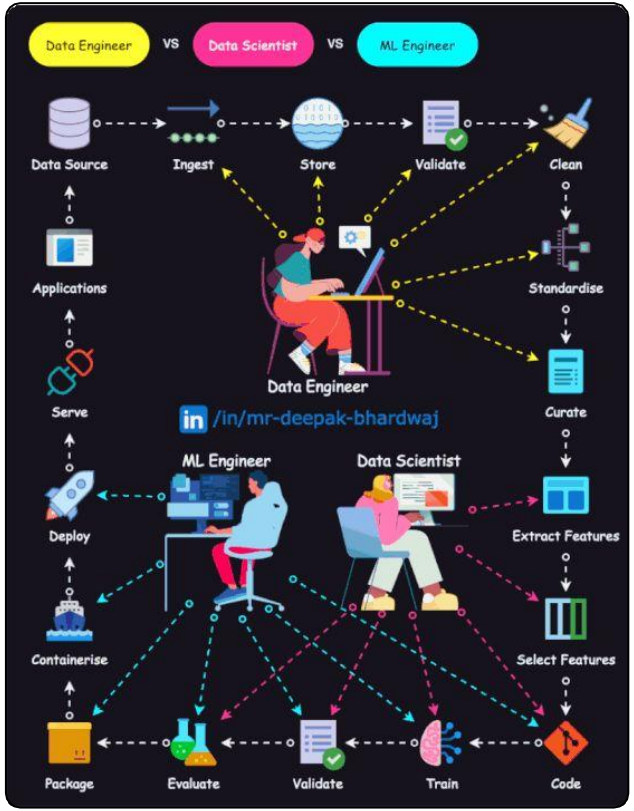
Data Engineer vs Data Science vs ML Engineer

Data Science

Un Data Scientist es alguien que **extrae conocimiento e insights de datos estructurados y no estructurados**. Utilizan una combinación de técnicas de **estadísticas**, **ciencia de datos** y **machine learning**, junto con el **entendimiento del negocio**, para analizar y encontrar patrones en los datos.

Los científicos de datos aplican modelos de ML para hacer **predicciones o clasificar información**, y comunican sus hallazgos a los stakeholders de la empresa para informar la toma de decisiones.

A menudo requieren una sólida base en **matemáticas, estadística y programación** (por lo general en **Python o R**) y habilidades para **visualizar datos** y comunicar resultados a un público no técnico.



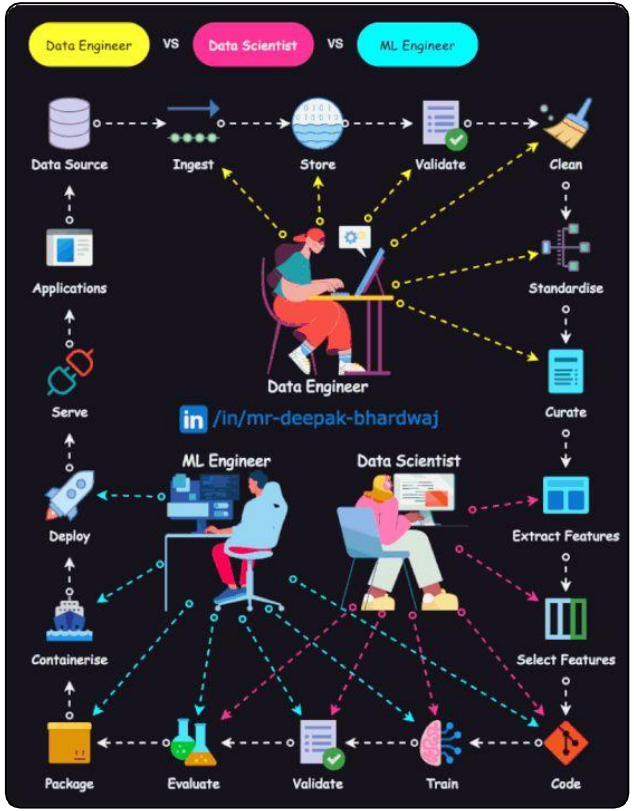
Data Engineer vs Data Science vs ML Engineer

ML Engineer

Un Machine Learning Engineer es un tipo de ingeniero especializado que **construye y despliega algoritmos y modelos de machine learning** diseñados por científicos de datos. La ingeniería de ML se centra más en la **aplicación práctica de modelos predictivos** y en la **creación de productos de ML**, como sistemas de recomendación o motores de búsqueda automatizados.

Los ML Engineers **trabajan estrechamente con los Data Scientists** para entender los modelos y algoritmos desarrollados y son **responsables de llevar estos modelos desde el prototipo hasta la producción.**

Esto incluye la **implementación, escalado, y mantenimiento** de los modelos dentro de sistemas existentes, y requiere un **conocimiento** profundo de las **plataformas de ML**, **programación**, y a menudo de **ingeniería de software** y **operaciones (DevOps)**, incluyendo prácticas de MLOps para **monitorear y gestionar sistemas de ML en producción**.



Data Engineer vs Data Science vs ML Engineer

Funciones y responsabilidades

Data Engineer

- **Diseñar, construir y mantener** la infraestructura de datos y arquitecturas de bases de datos.
- Implementar sistemas de **procesamiento de datos** y **pipelines ETL**.
- Asegurar la **calidad y la accesibilidad** de los datos.
- Optimizar soluciones de almacenamiento de datos.

Data Scientist

- **Analizar y extraer** información relevante de grandes conjuntos de datos.
- Utilizar **estadísticas y algoritmos de machine learning** para crear modelos predictivos.
- Interpretar y comunicar los resultados a los stakeholders.
- Realizar **experimentos científicos y pruebas A/B.**

ML Engineer

- **Implementar modelos** de machine learning en producción.
- **Optimizar algoritmos** para mejorar la eficiencia y el rendimiento.
- Trabajar en estrecha **colaboración con los científicos de datos** para entender los modelos y llevarlos a un entorno de producción.
- **Mantener y monitorear** sistemas de ML en producción.

Data Engineer vs Data Science vs ML Engineer

Herramientas y tecnologías

Data Engineer

- Herramientas de procesamiento de big data como **Hadoop, Spark y Kafka.**
- Bases de datos **SQL y NoSQL**.
- Lenguajes de programación como **Python, Java y Scala**.

Data Scientist

- Herramientas de análisis de datos y visualización como **Pandas, NumPy, Matplotlib y Seaborn.**
- Lenguajes de programación estadísticos como **R y Python.**
- Plataformas de ciencia de datos como **Jupyter Notebooks y RStudio**.
- Herramientas de machine learning como **Scikit-learn, TensorFlow y PyTorch.**

ML Engineer

- Frameworks de machine learning y deep learning como **TensorFlow, Keras y PyTorch.**
- Herramientas de desarrollo y despliegue como **Docker, Kubernetes y CI/CD.**
- Plataformas de cloud computing como **AWS SageMaker, Azure ML y VertexAI - GCP.**
- Monitoreo y operación de modelos ML en entornos de producción.

Data Engineer vs Data Science vs ML Engineer

Objetivos y metas

Data Engineer

- Garantizar la **fiabilidad, eficiencia y calidad** de los sistemas de datos.
- **Facilitar el acceso y la manipulación** de datos para los usuarios finales.

Data Scientist

- **Extraer insights y conocimientos** de los datos para resolver problemas de negocio.
- **Crear modelos predictivos y algoritmos** para entender tendencias, patrones y relaciones en los datos.

ML Engineer

- **Traducir modelos** de data science en **aplicaciones funcionales y escalables.**
- **Asegurar el rendimiento** y la escalabilidad de los modelos de machine learning en producción.

MLOps (Machine Learning Operations)

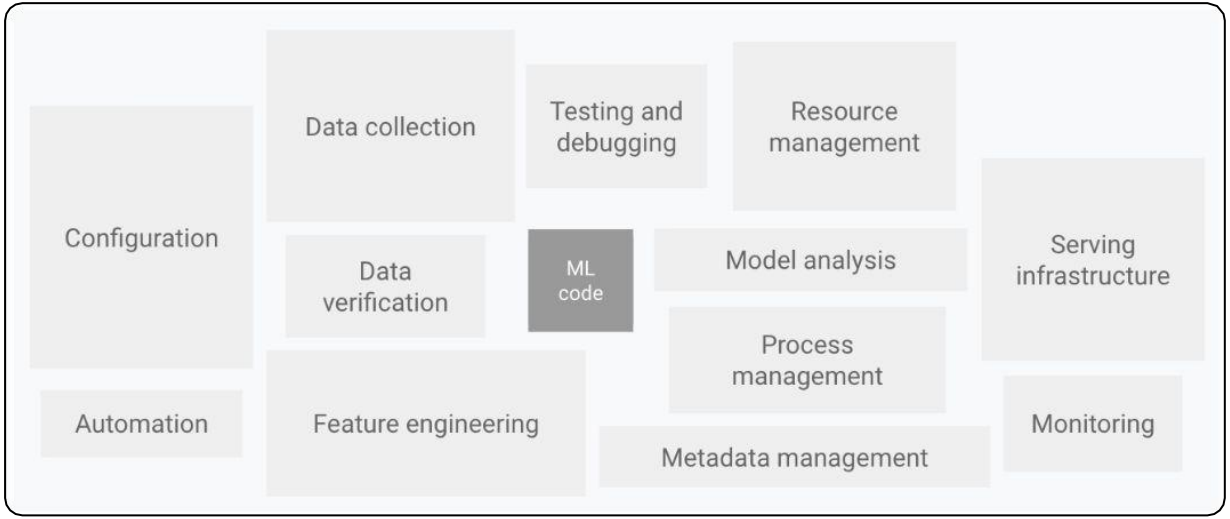
Contexto: Vivimos en una era de **datos masivos**, **procesamiento asequible** y **avances rápidos en ML**, lo que impulsa a las empresas a desarrollar modelos predictivos para generar valor comercial.

¿Qué es MLOps?
Un enfoque que **integra principios de DevOps en ML**, buscando la automatización y supervisión en todas las fases del ciclo de vida de ML, desde el desarrollo hasta la operación.

Desafíos: Más allá de desarrollar un modelo con buen rendimiento, el reto está en **crear y mantener sistemas de ML** que **operen eficientemente en producción.**

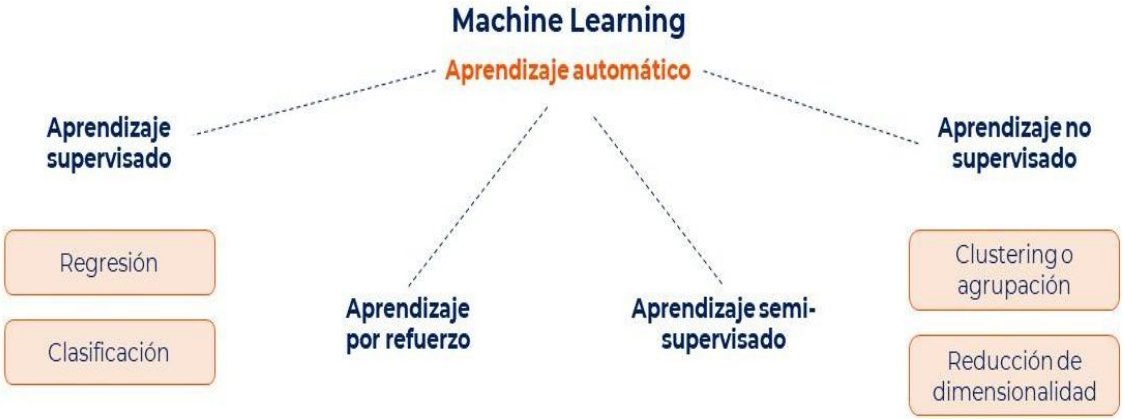
Un sistema de ML real **abarca mucho más que solo el código ML.** Los componentes críticos que lo rodean incluyen configuración, automatización, gestión de datos, pruebas, y mucho más.

Para manejar la complejidad de estos sistemas, se aplican principios de DevOps adaptados al ML, conocidos como MLOps. Esto incluye prácticas esenciales como:
CI (Integración Continua)
CD (Despliegue Continuo)
CT (Entrenamiento Continuo).



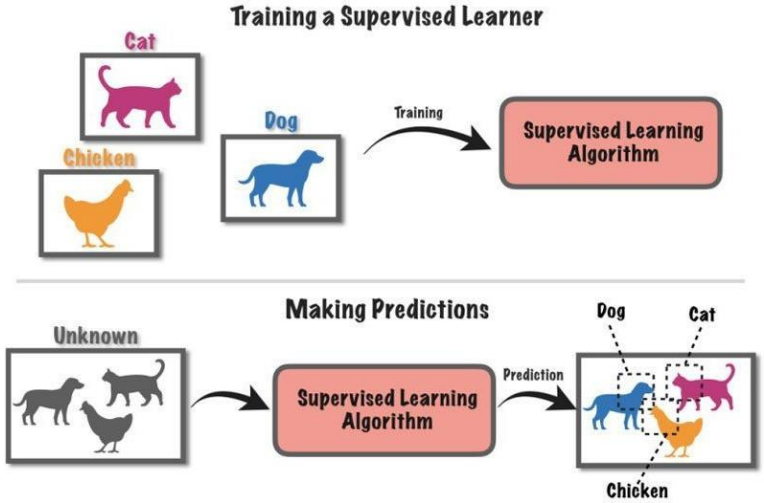
ML vs AutoML

Tipos de Machine Learning



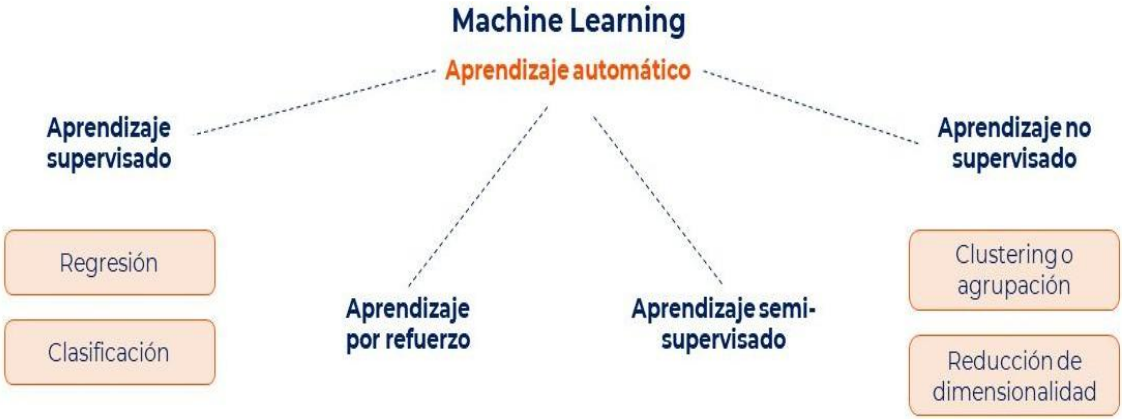
Aprendizaje Supervisado:

Este tipo de ML **utiliza datos etiquetados** para entrenar modelos, lo que significa que cada ejemplo de datos en el conjunto de entrenamiento está asociado con una respuesta correcta (etiqueta). Una vez entrenado, el modelo puede **predecir la etiqueta para nuevos datos no vistos.**
Ejemplos de casos de uso incluyen la clasificación de **correos electrónicos como spam o no spam** y la **predicción del precio** de una casa basada en sus características.



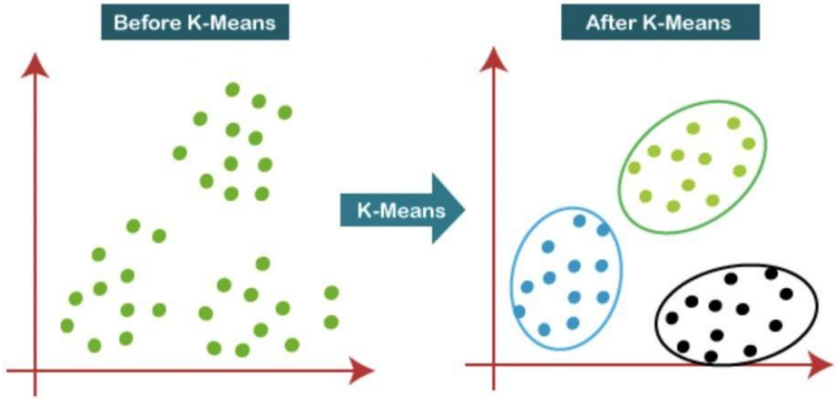
ML vs AutoML

Tipos de Machine Learning



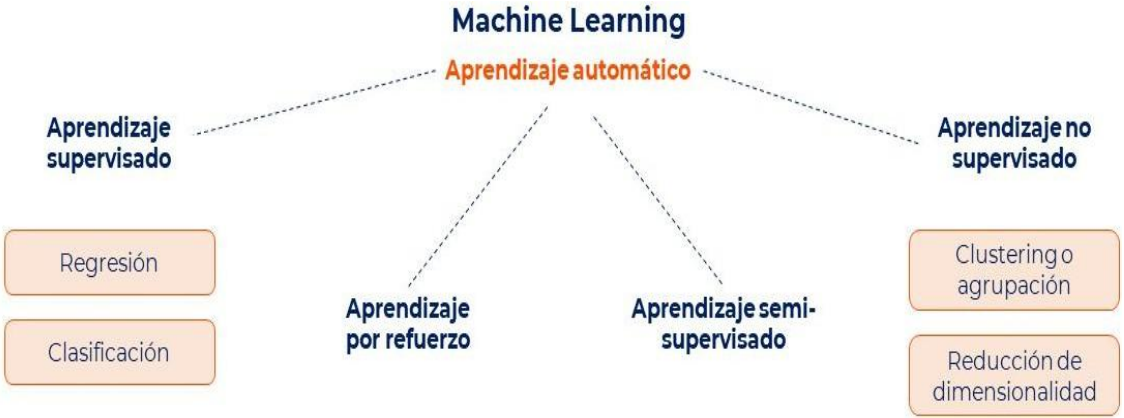
Aprendizaje No Supervisado:

A diferencia del aprendizaje supervisado, el aprendizaje no supervisado trabaja con **datos no etiquetados.** El objetivo aquí es explorar la estructura de los datos para **encontrar** algún tipo de **organización o patrones.** Los casos de uso comunes incluyen la **segmentación de clientes en marketing** y la **detección de anomalías** para identificar actividad fraudulenta en transacciones financieras.



ML vs AutoML

Tipos de Machine Learning



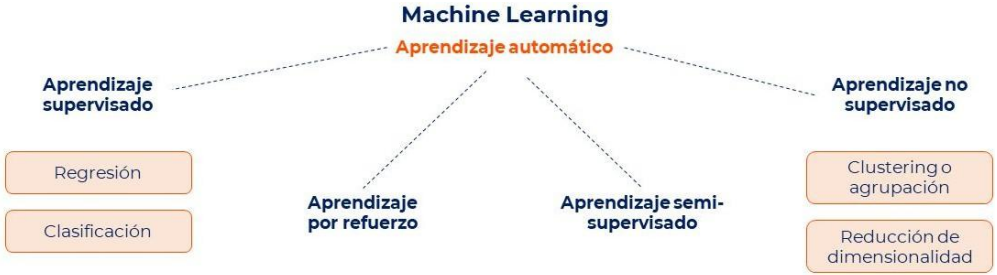
Aprendizaje Semi-Supervisado:

Esta aproximación utiliza una combinación de **datos etiquetados y no etiquetados** para entrenar modelos. Es útil cuando disponer de un **conjunto de datos completamente etiquetado es costoso o poco práctico.** Se utiliza en escenarios donde los datos no etiquetados pueden ayudar a mejorar la precisión del modelo.

- Uso de Datos Etiquetados
- Aplicación del Modelo a Datos No Etiquetados
- Selección de Muestras Confiables
- Reentrenamiento del Modelo
- Validación y Ajuste

ML vs AutoML

Tipos de Machine Learning



Aprendizaje por Refuerzo: En el aprendizaje por refuerzo, un "agente" **aprende a tomar decisiones** optimizando un "criterio de recompensa" a través de la **experimentación** y la interacción con un entorno. Se utiliza en áreas como los juegos (por ejemplo, AlphaGo), la robótica para la navegación autónoma y en sistemas de recomendación para maximizar ciertas métricas de interés.

Agente: En este contexto, el agente es el programa de IA que estás desarrollando para jugar al ajedrez.

Entorno: El tablero de ajedrez y las piezas constituyen el entorno. El estado del entorno incluye la posición de todas las piezas en el tablero.

Acciones: Las acciones son los movimientos legales que el agente puede realizar en el tablero de ajedrez, dependiendo del estado actual del juego.

Recompensas: Cada vez que el agente realiza un movimiento, recibe una recompensa (o penalización). Por ejemplo, puede recibir una recompensa positiva por capturar una pieza del oponente y una recompensa aún mayor por dar jaque mate. Por otro lado, podría recibir una penalización si pierde una de sus propias piezas o si realiza un movimiento que resulta en una posición desventajosa.

Política: La política es la estrategia que el agente utiliza para decidir qué movimientos realizar. Al principio, esta política podría ser aleatoria, pero con el tiempo, el agente aprende de sus experiencias. Utiliza las recompensas y penalizaciones que ha recibido de movimientos anteriores para formular una política que maximice las recompensas futuras.

Aprendizaje y mejora: A medida que el agente juega más y más partidas, recopila más datos sobre qué movimientos conducen a mejores resultados. Utiliza este conocimiento para actualizar su política, mejorando gradualmente su habilidad en el juego. Esto se logra a través de métodos como Q-learning, deep reinforcement learning, o algoritmos de optimización de políticas.

ML vs AutoML

Machine Learning: Ventajas

01	Personalización	• Alta capacidad de personalización para adaptarse a problemas específicos.
02	Transparencia y control	• Permite un control detallado sobre el proceso de modelado, lo que puede ofrecer una mejor interpretación y explicación de los resultados.
03	Precisión	• Potencial para alcanzar una mayor precisión a través de la afinación y optimización de modelos.
04	Adaptabilidad	• Capacidad para ajustar modelos a necesidades específicas y a restricciones de dominio.
05	Interpretabilidad	• Mayor capacidad de interpretar los modelos, especialmente con técnicas que priorizan la transparencia.

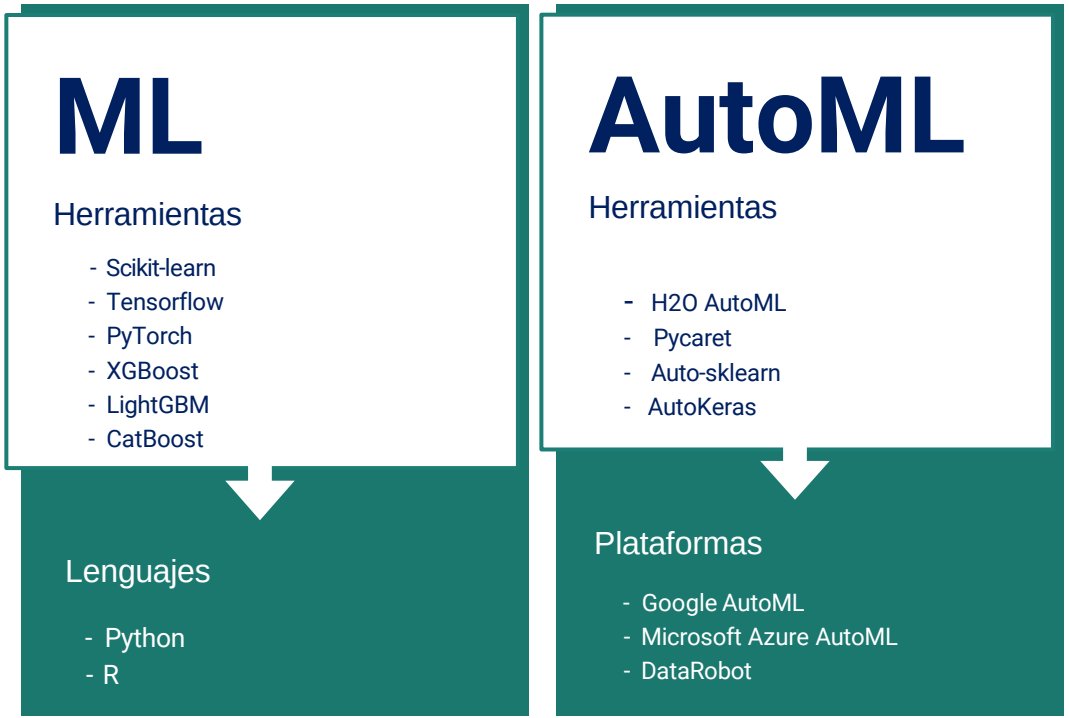
ML vs AutoML

AutoML: Ventajas

01	Eficiencia y Accesibilidad	• Hace que el ML sea más accesible para no expertos y reduce significativamente el tiempo y recursos necesarios para ejecutar modelos de ML.
02	Automatización	• Automatiza varios pasos del proceso de ML, incluyendo la selección de algoritmos, optimización de hiperparámetros, y validación cruzada.
03	Velocidad	• Reduce significativamente el tiempo desde la conceptualización hasta la implementación del modelo.
04	Democratización	• Hace que el ML sea más accesible para personas con menos formación en ciencia de datos.
05	Optimización	• Realiza automáticamente la búsqueda y evaluación de múltiples modelos y preprocesamientos.

ML vs AutoML

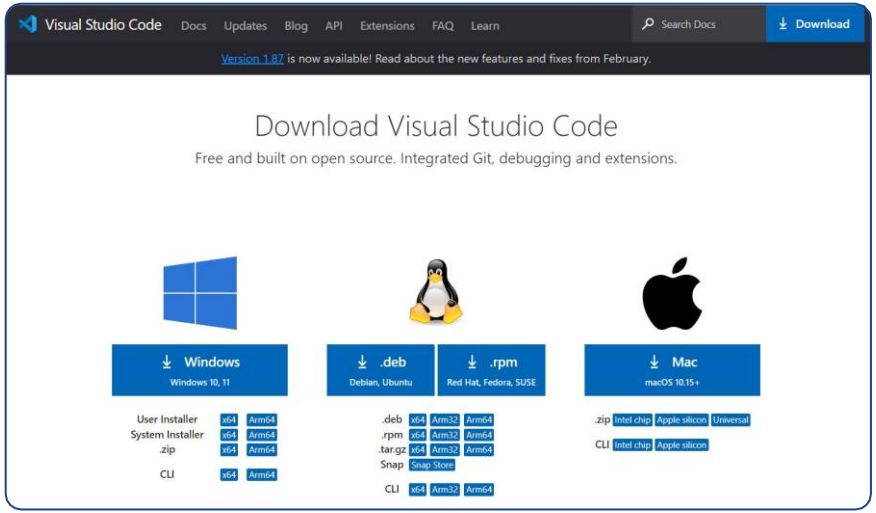
Herramientas

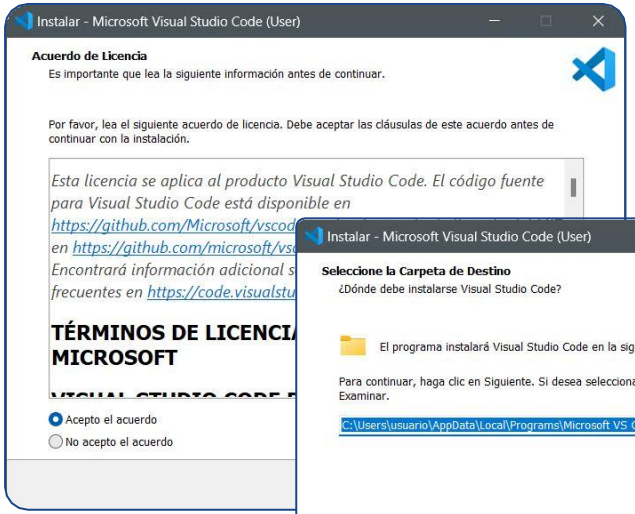


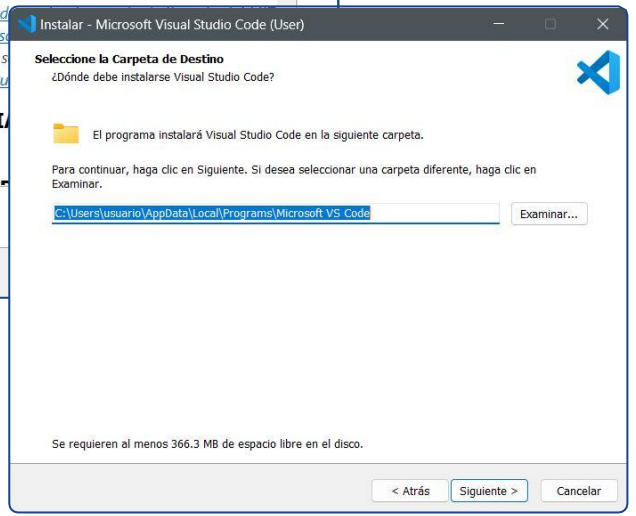
Visual Studio Code

Instalación

https://code.visualstudio.com/download

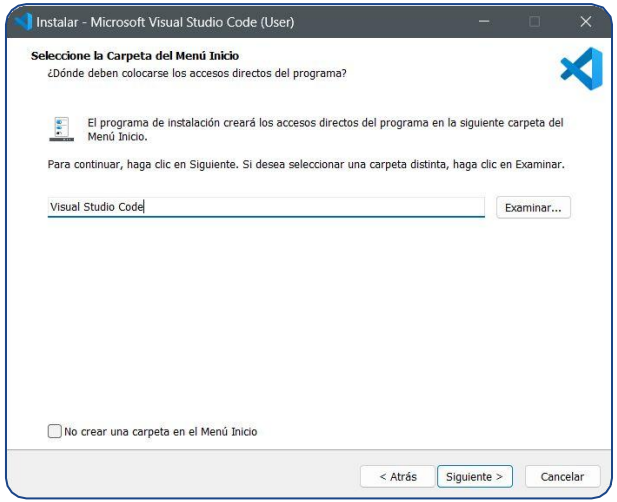


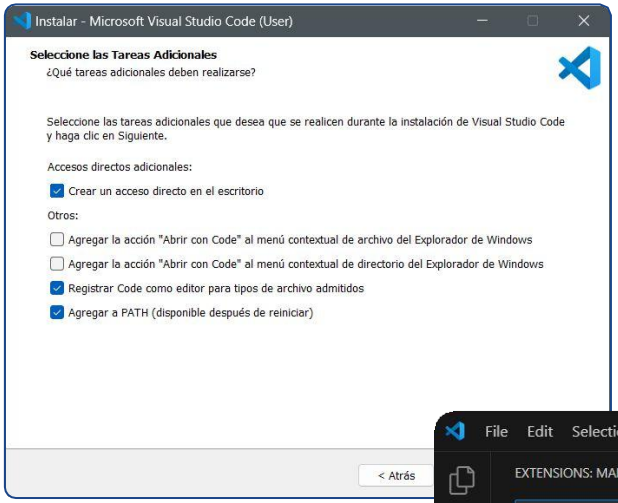


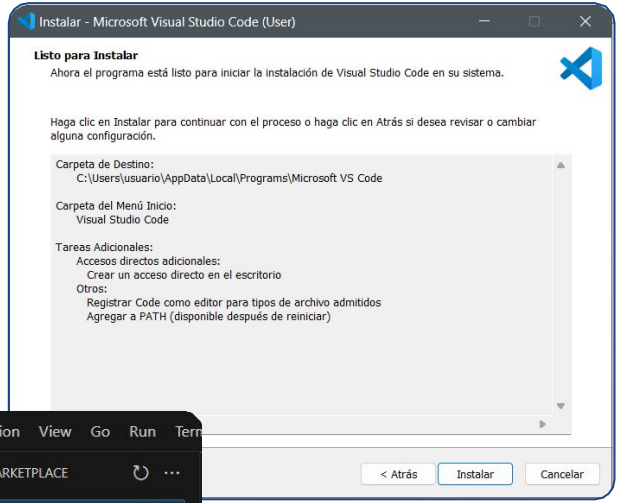


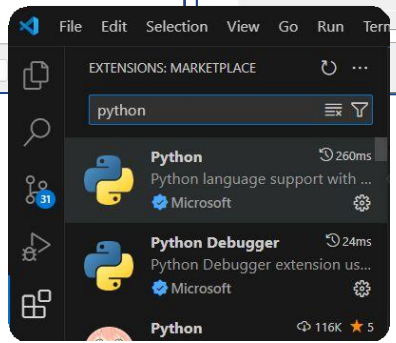
Visual Studio Code

Instalación





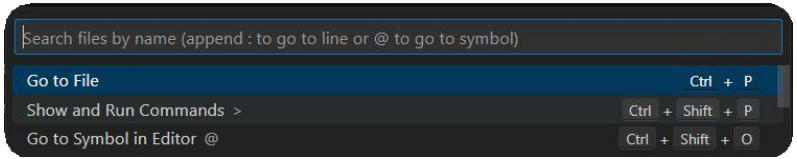




Visual Studio Code

Crear entorno

Un "entorno" se refiere a un conjunto de condiciones y configuraciones en las que se ejecuta un programa o aplicación de computadora.



Entornos Conda

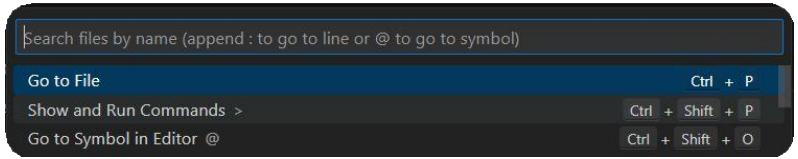
Conda es un administrador de paquetes binarios multiplataforma e independiente del idioma. Es conocido por simplificar la gestión de dependencias complejas y es particularmente popular en las comunidades de ciencia de datos e informática científica porque **puede gestionar paquetes de diferentes lenguajes (Python, R, etc.)** y gestionar dependencias de bibliotecas fuera del ecosistema Python.

Entornos Venv

Venv es un **módulo disponible de forma predeterminada en Python 3.3** y posteriores, diseñado para crear **entornos Python aislados**. El módulo venv no ofrece administración de paquetes y normalmente se usa con pip para instalar paquetes de Python.

Visual Studio Code

Seleccionar un intérprete



Un intérprete es un **programa que ejecuta instrucciones escritas en un lenguaje de programación**. En el caso de Python, el intérprete toma el código fuente, escrito en Python, y lo ejecuta directamente.

Programación con Python

Variables en Python

En Python, las variables se utilizan para **almacenar información que puede ser referenciada y manipulada en un programa.** No necesitas declarar el tipo de una variable explícitamente. El tipo es inferido automáticamente cuando asignas un valor a la variable.

```
# Asignación de variables
numero = 10              # Un entero (int)
nombre = "Alice"         # Una cadena de texto (str)
precio = 99.99           # Un número de punto flotante
(float)
esta_activo = True       # Un valor booleano (bool)

# Mostrar los valores de las variables
print(numero)
print(nombre)
print(precio)
print(esta_activo)
```

Programación con Python

Estructuras de control

Las estructuras de control en Python te **permiten dirigir el flujo del programa.** Python utiliza la indentación para definir bloques de código.

Condicional (**if, elif, else**)

```
edad = 20

if edad < 18:
    print("Menor de edad")
elif edad >= 18 and edad < 60:
    print("Adulto")
else:
    print("Adulto mayor")
```

Bucle **'For'**: Utilizado para iterar sobre una secuencia (lista, tupla, diccionario, conjunto o cadena).

```
# Iterar sobre una lista
frutas = ["manzana", "plátano", "cereza"]
for fruta in frutas:
    print(fruta)
```

Bucle **'While'**: Repite una instrucción o grupo de instrucciones mientras una condición dada es verdadera.

```
# Repetir mientras la condición sea verdadera
contador = 0
while contador < 5:
    print(contador)
    contador += 1
```

Programación con Python

Funciones

Las funciones en Python son **bloques de código que solo se ejecutan cuando se llaman**. Pueden recibir datos para trabajar (conocidos como parámetros) y pueden devolver datos como resultado.

```python
# Definir una función
def saluda(nombre):
    return "Hola, " + nombre

# Llamar a la función
mensaje = saluda("Alice")
print(mensaje)

# Función con varios parámetros
def suma(a, b):
    return a + b

# Llamar a la función y mostrar el resultado
print(suma(5, 7))
```

Programación con Python

POO

La POO es un **paradigma de programación** basado en el concepto de "**objetos**", que **pueden contener datos y código**: datos en forma de atributos (también conocidos como campos o propiedades) y código en forma de métodos (funciones asociadas a las clases).

Programación con Python

Conceptos de POO

Clases: Son los "planos" o plantillas para crear objetos. Definen un tipo de dato que incluye tanto variables de instancia (atributos) como funciones (métodos).

Objetos: Son instancias de clases. Cada objeto puede tener diferentes valores para sus atributos.

Atributos: Son variables asociadas a un objeto o clase.

Métodos: Son funciones definidas dentro de una clase y pueden ser utilizados por los objetos de esa clase.

```python
class Animal:
    def __init__(self, nombre, sonido):
        self.nombre = nombre   # Atributo
        self.sonido = sonido   # Atributo

    def hablar(self):   # Método
        return f"{self.nombre} dice {self.sonido}"

# Creando un objeto de la clase Animal
gato = Animal("Felix", "miau")
print(gato.hablar())   # Felix dice miau
```

Programación con Python

Conceptos de POO

Herencia: Es un mecanismo por el que una clase puede heredar atributos y métodos de otra clase.

```python
# Clase base o superclase
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def presentarse(self):
        return f"Yo soy un {self.nombre}"

# Clase derivada o subclase
class Perro(Animal):
    def hablar(self):
        return "Guau!"

# Uso de la clase derivada
my_dog = Perro("Perro")
print(my_dog.presentarse())  # Yo soy un Perro
print(my_dog.hablar())  # Guau!
```

Programación con Python

Conceptos de POO

Composición: La composición es un concepto fundamental en la Programación Orientada a Objetos (POO) que permite crear clases complejas combinando objetos más simples. Se refiere a la formación de clases compuestas utilizando objetos de otras clases como atributos.

```python
class Motor:
    def __init__(self, cilindros ):
        self.cilindros  = cilindros
        self.estado = 'apagado'

    def encender(self):
        self.estado = 'encendido'

class Coche:
    def __init__(self, modelo):
        self.modelo = modelo
        self.motor = Motor(4)   # Composición

# Uso de composición
my_car = Coche("Toyota")
my_car.motor.encender()
print(my_car.motor.estado)   # encendido
```

Programación con Python

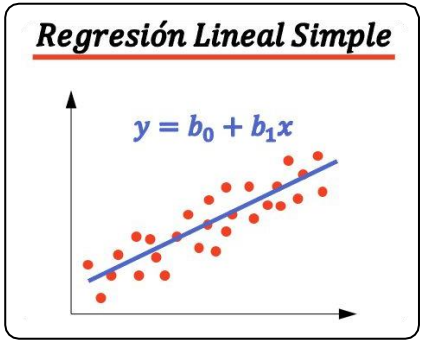
Clase para modelo de ML

```python
from sklearn.linear_model import LinearRegression as SKLinearRegression

class LinearRegression:
    def __init__(self):
        self.model = SKLinearRegression()

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

    def predict(self, X):
        # Verificamos si el modelo ha sido entrenado
        if self.model.coef_ is None:
            raise ValueError("Modelo no ha sido entrenado aún.")
        return self.model.predict(X)
```




```python
import numpy as np
# Creando datos de ejemplo
X = np.array([[1], [2], [3], [4]])   # Características
y = np.array([2, 3, 4, 5])           # Objetivo

# Usando la clase de regresión lineal
modelo = LinearRegression()
modelo.fit(X, y)   # Ajustamos el modelo
print(modelo.predict([[6]]))   # Hacemos una predicción
```

Programación con Python: Testing de código

El testing de código, o prueba de software, es el proceso de ejecutar un programa o aplicación con la intención de encontrar errores de software.

Importancia en el desarrollo de proyectos ML

- **Prevención de fallos en producción:** Permite identificar y solucionar problemas antes de que el software se despliegue, lo que reduce el riesgo de fallos que pueden ser costosos o peligrosos.
- **Confianza y seguridad:** Aumenta la confianza en la fiabilidad y seguridad del software, lo cual es especialmente crítico en aplicaciones sensibles y en sistemas de machine learning donde las decisiones pueden tener grandes repercusiones.
- **Eficiencia en el desarrollo:** Ayuda a detectar errores en las primeras etapas del desarrollo, reduciendo el tiempo y el costo asociados con su resolución.
- **Documentación y mantenibilidad:** Las pruebas bien diseñadas pueden servir como documentación del comportamiento esperado del software y facilitar futuras modificaciones y mantenimiento.

Programación con Python: Testing de código

Métodos de aserción

assertTrue y assertFalse

assertTrue(expr, msg=None): Verifica que expr sea verdadero. Si expr es falso, la prueba fallará.
assertFalse(expr, msg=None): Verifica que expr sea falso. Si expr es verdadero, la prueba fallará.

assertGreater, assertGreaterEqual, assertLess, y assertLessEqual

assertGreater(a, b, msg=None): Verifica que a sea mayor que b.
assertGreaterEqual(a, b, msg=None): Verifica que a sea mayor o igual que b.
assertLess(a, b, msg=None): Verifica que a sea menor que b.
assertLessEqual(a, b, msg=None): Verifica que a sea menor o igual que b.

Programación con Python: Testing de código

Métodos de aserción

Métodos de aserción para igualdad y diferencia

assertEqual(a, b, msg=None): Verifica que a y b sean iguales.
assertNotEqual(a, b, msg=None): Verifica que a y b no sean iguales.

Métodos de aserción para listas y colecciones

assertIn(member, container, msg=None): Verifica que member esté en container.
assertNotIn(member, container, msg=None): Verifica que member no esté en container.
assertListEqual(list1, list2, msg=None): Verifica que dos listas sean iguales.
assertDictEqual(d1, d2, msg=None): Verifica que dos diccionarios sean iguales.
assertTupleEqual(t1, t2, msg=None): Verifica que dos tuplas sean iguales.
assertSetEqual(set1, set2, msg=None): Verifica que dos conjuntos sean iguales.

Programación con Python: Testing de código

Métodos de aserción

Métodos de aserción para números y tipos

assertAlmostEqual(a, b, places=7, msg=None, delta=None): Verifica que a y b sean aproximadamente iguales hasta un número determinado de lugares decimales.
assertNotAlmostEqual(a, b, places=7, msg=None, delta=None): Verifica que a y b no sean aproximadamente iguales hasta un número determinado de lugares decimales.
assertIsInstance(obj, cls, msg=None): Verifica que obj sea una instancia de cls.
assertNotIsInstance(obj, cls, msg=None): Verifica que obj no sea una instancia de cls.

Métodos de aserción para identidad

assertIs(a, b, msg=None): Verifica que a es b (es decir, que ambos apuntan al mismo objeto).
assertIsNot(a, b, msg=None): Verifica que a no es b.

Programación con Python: Testing de código

Tipos de pruebas: Pruebas unitarias

Evalúan la funcionalidad de una parte específica del software, generalmente a nivel de función o método. Son esenciales para asegurar que cada componente funcione correctamente de manera aislada.

utils.py

```python
def calculate_average(numbers):
    return sum(numbers) / len(numbers)
```

test_unit.py

```python
import unittest
from utils import calculate_average

class TestCalculateAverage (unittest.TestCase):
    def test_with_positive_numbers (self):
        self.assertEqual (calculate_average ([10, 20, 30]), 20)

    def test_with_negative_numbers (self):
        self.assertEqual (calculate_average ([-10, -20, -30]), -20)

if __name__ == '__main__':
    unittest.main()
```

```
python -m unittest test_unit.py
```

Programación con Python: Testing de código

Tipos de pruebas: Pruebas de integración

Verifican que diferentes módulos o servicios del software funcionen juntos correctamente. Son críticas cuando se ensamblan las diferentes partes del sistema para formar un todo.

utils.py

```python
def calculate_average(numbers):
    return sum(numbers) / len(numbers)

def normalize_numbers(numbers):
    avg = calculate_average(numbers)
    return [(num - avg) / avg for num in numbers]
```

test_integration.py

```python
import unittest
from utils import calculate_average, normalize_numbers

class TestIntegration(unittest.TestCase):
    def test_normalize_numbers(self):
        numbers = [10, 20, 30]
        normalized = normalize_numbers(numbers)
        expected = [-0.5, 0, 0.5]  # Calculado manualmente
        for n, e in zip(normalized, expected):
            self.assertAlmostEqual(n, e)

if __name__ == '__main__':
    unittest.main()
```

Programación con Python: Testing de código

Tipos de pruebas: Pruebas funcionales

Se enfocan en examinar las características y la funcionalidad del software para asegurarse de que cumplan con los requisitos especificados. Estas pruebas se realizan desde la perspectiva del usuario final.

iris_processor.py

```python
from sklearn.datasets import load_iris
import pandas as pd

def process_iris_data():
    # Cargar los datos de Iris
    iris = load_iris()
    data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
    data['species'] = pd.Categorical.from_codes(iris.target,
iris.target_names)

    # Calcular la media de cada característica por especie
    mean_values = data.groupby('species').mean()
    return mean_values
```

test_functional.py

```python
import unittest
from iris_processor import process_iris_data
class TestIrisDataProcessor (unittest.TestCase):
    def test_process_iris_data (self):
        # Ejecutar la función de procesamiento
        result = process_iris_data ()
        # Verificar las columnas y filas esperadas
        self.assertEqual (list(result.columns), ['sepal length
(cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)'])
        self.assertEqual (len(result), 3)   # Debe haber 3 filas
        # Verificar que los valores medios no sean nulos
        for column in result.columns:
            self.assertFalse (result[column].isnull().any())

if __name__ == '__main__':
    unittest.main()
```

Programación con Python: Testing de código

Tipos de pruebas: Pruebas de sistema

Evalúan el comportamiento completo del sistema para verificar que cumpla con las especificaciones y requisitos. Estas pruebas consideran aspectos como el rendimiento, la seguridad y la comportabilidad del sistema.

system.py

```python
class MLSystem:
    def __init__(self):
        pass

    def load_data(self):
        # Simula la carga de un dataset
        data = load_iris()
        return data.data, data.target

    def preprocess_data(self, X):
        # Simula el preprocesamiento de los datos
        scaler = StandardScaler().fit(X)
        return scaler, scaler.transform(X)

    def train_model(self, X, y):
        # Simula el entrenamiento de un modelo
        model = LogisticRegression(random_state=42)
        model.fit(X, y)
        return model
```

```python
    def evaluate_model(self, model, X, y):
        # Simula la evaluación de un modelo
        predictions = model.predict(X)
        return accuracy_score(y, predictions)

    def run_entire_workflow(self, input_data_path):
        try:
            X, y = self.load_data()
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
            scaler, X_train_scaled = self.preprocess_data(X_train)
            model = self.train_model(X_train_scaled, y_train)
            X_test_scaled = scaler.transform(X_test)
            accuracy = self.evaluate_model(model, X_test_scaled, y_test)
            return {'success': True, 'accuracy': accuracy}
        except Exception as e:
            return {'success': False, 'message': str(e)}
```

Programación con Python: Testing de código

Tipos de pruebas: Pruebas de sistema

Evalúan el comportamiento completo del sistema para verificar que cumpla con las especificaciones y requisitos. Estas pruebas consideran aspectos como el rendimiento, la seguridad y la comportabilidad del sistema.

test_system.py

```
import unittest
from system import MLSystem

class TestMLSystem(unittest.TestCase):
    def test_entire_workflow(self):
        # Inicializa el sistema de ML
        system = MLSystem()
        # Ejecuta el flujo de trabajo completo y obtiene el
resultado
        result = system.run_entire_workflow(None)

        # Verifica que el flujo de trabajo se haya
completado con éxito
        self.assertTrue(result['success'], "The ML system
workflow should have completed successfully." )
```

```
# Verifica que la precisión del modelo sea razonable
# Nota: el umbral específico depende del caso de uso y
expectativas
        self.assertGreater(result['accuracy'], 0.7, "The
model accuracy should be above 0.7." )

if __name__ == '__main__':
    unittest.main()
```

¡GRACIAS!