PYTHON

Y

CIENCIA DE DATOS

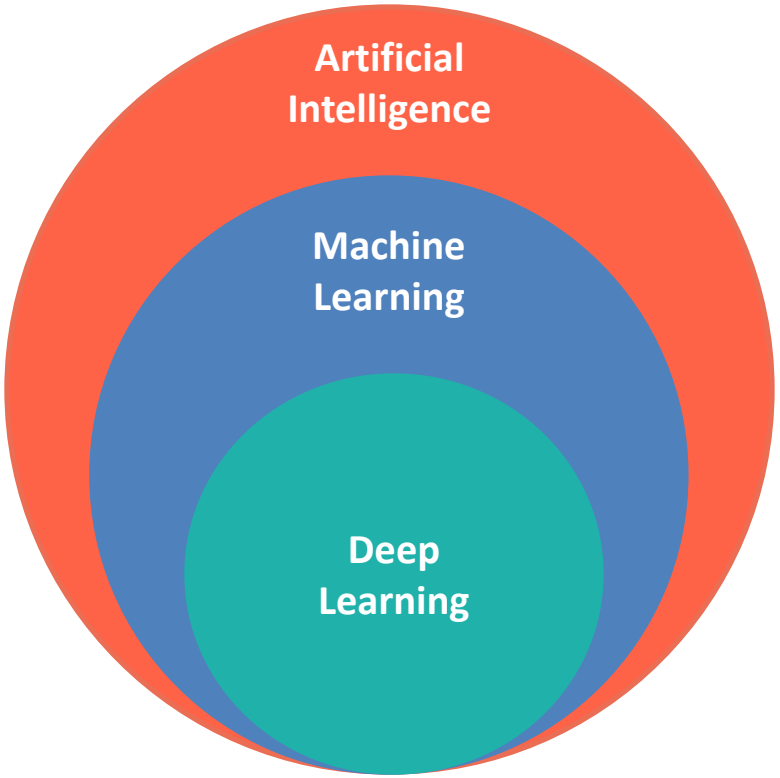
INTRODUCCIÓN

Los datos están en todas partes y desde el nacimiento de la era digital han empezado a crecer exponencialmente. La ciencia de los datos es un campo que lo que intenta hacer es extraer ideas, aportar información útil, etc.

Etapas del análisis de datos

Las etapas que se distinguen son:

Etapa 1 → Detectar el problema: Qué queremos estimar o predecir?
Etapa 2 → Obtener y preparar los datos: Qué recursos tenemos?, qué información es relevante? limpiar los datos.
Etapa 3 → Explorar los datos: visualizarlos y localizar en los gráficos posibles tendencias, correlaciones o patrones.
Etapa 4 → Modelizar y evaluar los datos: crear el modelo con un algoritmo innovador. Evaluar el modelo.
Etapa 5 → Comunicar los resultados: Qué resultados hemos obtenido?, Los resultados tienen sentido?



ARTIFICIAL INTELLIGENCE (Inteligencia Artificial)
Ciencia que faculta a las computadoras para imitar la inteligencia humana.

MACHINE LEARNING (Aprendizaje automático)
El Machine Learning es una rama de la Inteligencia Artificial que permite a las máquinas mejorar en una tarea determinada con la experiencia. Es importante aclarar que todas las técnicas de Machine Learning se clasifican como técnicas de Inteligencia Artificial. Sin embargo, no toda la Inteligencia Artificial puede contar como Machine Learning ya que algunos motores básicos basados en reglas pueden ser clasificados como IA pero no aprenden de la experiencia por lo que no pertenecen a la categoría de Machine Learning.

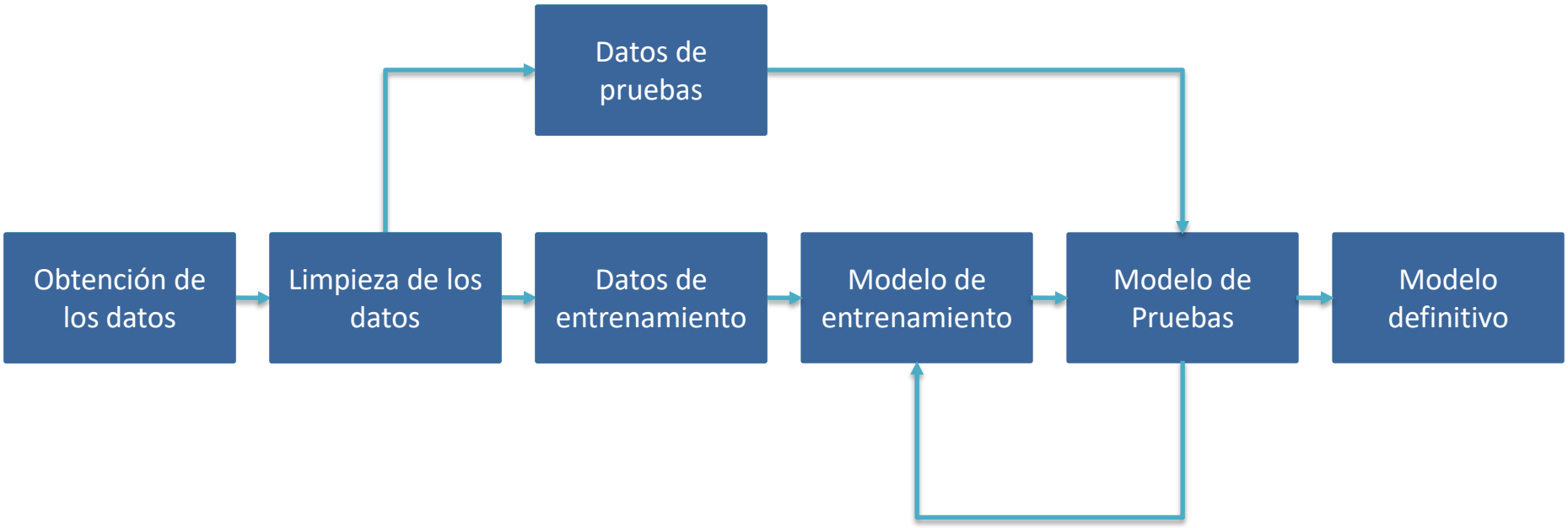
DEEP LEARNING (Aprendizaje Profundo)
El Deep Learning es una rama especializada del Machine Learning que se basa en el entrenamiento de Redes Neuronales Artificiales Profundas usando un gran conjunto de datos como imágenes o textos. Son modelos de procesamiento de información inspirados en el cerebro humano.

	Aprendizaje automático	Aprendizaje profundo
Formato de datos	Datos estructurados	Datos no estructurados
Base de datos	Base de datos manejable	Más de un millón de puntos de datos
Entrenamiento	Se necesita un entrenador humano	El sistema aprende por sí solo
Algoritmo	Algoritmo variable	Red neuronal de algoritmos
Aplicación	Tareas rutinarias sencillas	Tareas complejas
	Algunos ámbitos de aplicación	
	Marketing online	Seguridad de sistemas informáticos
	Atención al cliente	Atención al cliente
	Ventas	Creación de contenidos
	Inteligencia empresarial	Asistentes de voz
	Además de los campos mencionados, ambas tecnologías también se utilizan en muchos otros ámbitos de la vida, como la medicina, la ciencia o la movilidad.	

Machine Learning (Aprendizaje automático)

✓Es una rama de la IA cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan.
✓Es un método de análisis de datos que automatiza la construcción de un modelo analítico.
✓Permite a las computadoras encontrar soluciones a problemas, sin ser explícitamente programadas para ello, mediante el uso de algoritmos que aprenden de los datos.

Proceso de Machine Learning



Aprendizaje Automático

Supervisado:
Necesita datos previamente etiquetados para aprender a realizar el trabajo. En base a estos es capaz de aprender a resolver problemas futuros similares.

Regresión:
Ej: Partiendo de una lista de precios de casas, puede predecirse el precio de una casa en función de sus características (zona, tamaño, número de habitaciones, etc.)

Clasificación:
Ej: Partiendo de una lista de hombres y mujeres con su altura y peso, puede predecirse el sexo de una nueva persona en función de esos datos.

No supervisado:
Necesita indicaciones previas que le enseñan a comprender y analizar la información para resolver problemas futuros similares. No necesita datos previamente etiquetados.

Segmentación:
Ej: Partiendo de datos de perros de diversas razas se crea un algoritmo para predecir la raza de un nuevo perro según datos de altura y peso. En este caso, no hay etiquetas, no hay solución correcta a los datos iniciales, el objetivo es formar grupos similares para predecir a qué grupo pertenece el nuevo elemento.

Por refuerzo:
Aprende por su cuenta en base a conocimientos previamente introducidos y a la práctica que realiza sobre los problemas, aprendiendo en función del éxito o fracaso que obtiene al resolver los problemas.

En estos algoritmos se aprende a base de prueba y error, en función de los datos obtenidos, intentando maximizar el valor de alguna variable que interese, como conseguir máxima puntuación en un videojuego, ganar una partida de ajedrez, etc

PREPROCESADO DE DATOS

Análisis previo de los datos a preprocesar

```
# ejemplo 1
import pandas as pd

dataset = pd.read_csv('<ruta de archivo>/Datos.csv')
print(dataset)
```

```
>>> print(dataset)
        Pais  Edad  Salario Compro
0     Brasil  44.0  72000.0     No
1  Argentina  27.0  48000.0     Si
2      Chile  30.0  54000.0     No
3  Argentina  38.0  61000.0     No
4      Chile  40.0      NaN     Si
5     Brasil  35.0  58000.0     Si
6  Argentina   NaN  52000.0     No
7     Brasil  48.0  79000.0     Si
8      Chile  50.0  83000.0     No
9     Brasil  37.0  67000.0     Si
```

1	Pais,Edad,Salario,Compro
2	Brasil,44,72000,No
3	Argentina,27,48000,Si
4	Chile,30,54000,No
5	Argentina,38,61000,No
6	Chile,40,,Si
7	Brasil,35,58000,Si
8	Argentina,,52000,No
9	Brasil,48,79000,Si
10	Chile,50,83000,No
11	Brasil,37,67000,Si

Analizar la planilla de datos, distinguiendo que nos está mostrando:

- 4 columnas: Pais, Edad, Salario y Compro (variables)
- 10 observaciones
- Clientes de una empresa que han comprado un determinado producto.
- Las 3 primeras columnas tienen información personal, origen del cliente, edad y salario.
- La 4ta columna contiene información acerca de cómo ese cliente se relaciona con la empresa (compró o no compró el producto?). Entonces qué diferencia hay entre las variables?
- Distinguir variables:
  1. Primeras tres columnas: datos que soy capaz de observar, en este caso personas y características, serán las variable independientes. Pais, Edad y Salario, forman una matriz que hay que crear.
  2. La última columna, Compro, es la que el algoritmo de ML va a intentar predecir a partir de las variables independientes.

Las variables independientes son las que suministramos al algoritmo de ML y la dependiente es la que se quiere intentar predecir. Podría ser una categoría como [Compro] o un número , es decir que con Pais, Edad y Salario intentamos predecir si hubo o no hubo compra.

Separación de variables independientes de la variable dependiente

```
# ejemplo 2
import pandas as pd

dataset = pd.read_csv('<ruta de archivo>/Datos.csv')
print(dataset)

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 3].values
```

```
>>> print(dataset)
       Pais  Edad  Salario Compro
0    Brasil  44.0  72000.0     No
1  Argentina  27.0  48000.0     Si
2     Chile  30.0  54000.0     No
3  Argentina  38.0  61000.0     No
4     Chile  40.0      NaN     Si
5    Brasil  35.0  58000.0     Si
6  Argentina   NaN  52000.0     No
7    Brasil  48.0  79000.0     Si
8     Chile  50.0  83000.0     No
9    Brasil  37.0  67000.0     Si
```

```
>>> print(X)
[['Brasil' 44.0 72000.0]
 ['Argentina' 27.0 48000.0]
 ['Chile' 30.0 54000.0]
 ['Argentina' 38.0 61000.0]
 ['Chile' 40.0 nan]
 ['Brasil' 35.0 58000.0]
 ['Argentina' nan 52000.0]
 ['Brasil' 48.0 79000.0]
 ['Chile' 50.0 83000.0]
 ['Brasil' 37.0 67000.0]]
>>> print(y)
['No' 'Si' 'No' 'No' 'Si' 'Si' 'No' 'Si' 'No' 'Si']
```

Observar que para X ([[]]) devuelve una matriz y para y un vector([])

VALORES DESCONOCIDOS

Resolver el problema de los datos faltantes

En particular existen diferentes enfoques que se pueden llevar a cabo para resolver los NaN es decir, decidir qué se debe hacer con esos datos faltantes. Por ejemplo , en el conjunto de datos original tal cual lo tenemos aquí, podemos ver claramente que faltan dos valores, hay un dato que falta en la columna Edad y luego tenemos una persona de la cual desconocemos su Salario.

1	Pais,Edad,Salario,Compro
2	Brasil,44,72000,No
3	Argentina,27,48000,Si
4	Chile,30,54000,No
5	Argentina,38,61000,No
6	Chile,40,,Si
7	Brasil,35,58000,Si
8	Argentina,,52000,No
9	Brasil,48,79000,Si
10	Chile,50,83000,No
11	Brasil,37,67000,Si

```
>>> print(dataset)
       Pais  Edad  Salario Compro
0    Brasil  44.0  72000.0     No
1  Argentina  27.0  48000.0     Si
2     Chile  30.0  54000.0     No
3  Argentina  38.0  61000.0     No
4     Chile  40.0      NaN     Si
5    Brasil  35.0  58000.0     Si
6  Argentina   NaN  52000.0     No
7    Brasil  48.0  79000.0     Si
8     Chile  50.0  83000.0     No
9    Brasil  37.0  67000.0     Si
```

a. Hay que encontrar una mejor solución que el eliminar toda la fila
b. Una posibilidad, para este caso, es reemplazar el valor desconocido de la columna Edad por la media de todos los valores de la columna de Edad.
c. Con el mismo criterio se resolvería el valor desconocido de la columna Salario.

Resolver el problema de los datos faltantes:
→ **pip install -U scikit-learn** → https://scikit-learn.org/stable/

```
# ejemplo 3
import pandas as pd

dataset = pd.read_csv('<ruta de archivo>/Datos.csv')
print(dataset)
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 3].values

from sklearn.impute import SimpleImputer
import numpy as np
imputer = SimpleImputer(missing_values = np.nan, strategy="mean")
imputer = imputer.fit(X[:,1:3])
X[:,1:3] = imputer.transform(X[:,1:3])
```

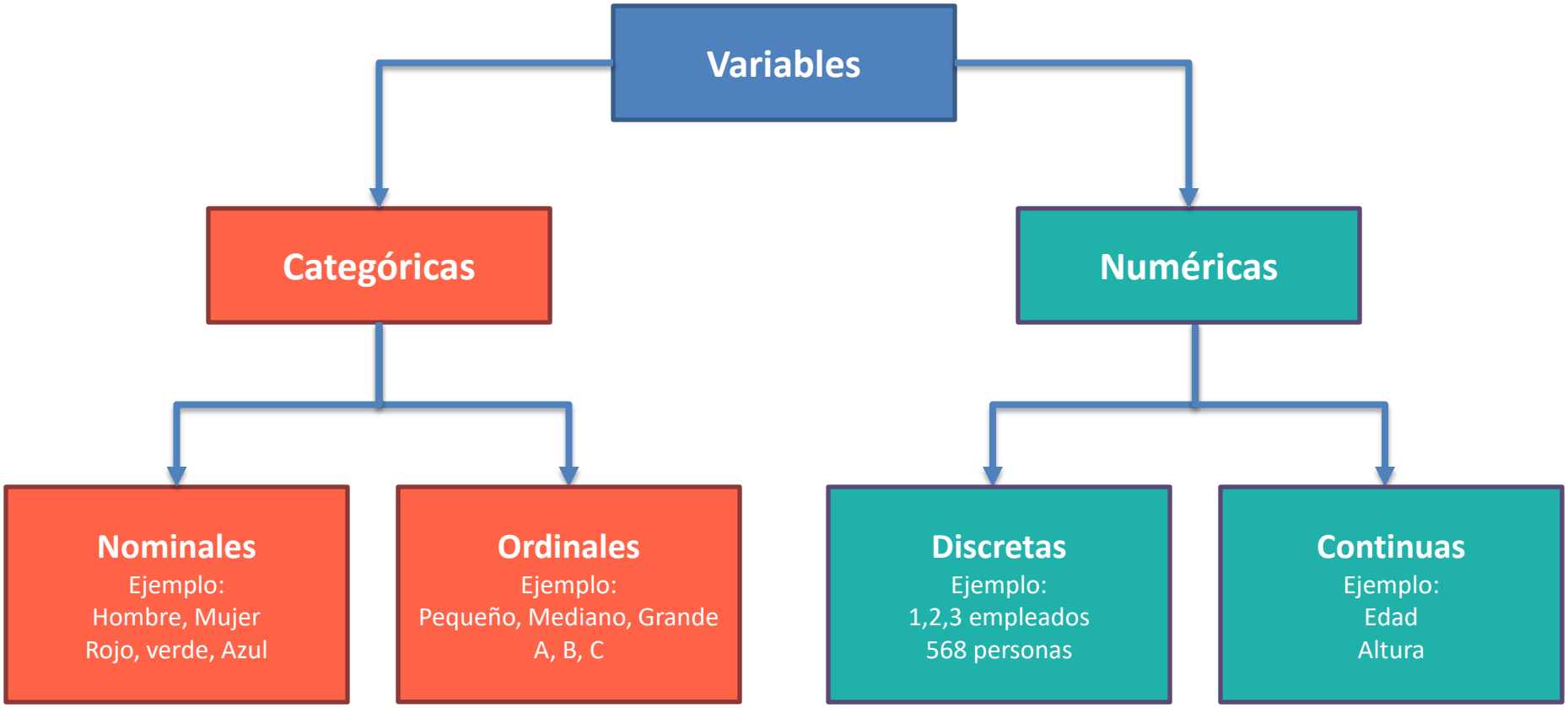
```
>>> print(X[:,1:3])
[[44.0 72000.0]
 [27.0 48000.0]
 [30.0 54000.0]
 [38.0 61000.0]
 [40.0 nan]
 [35.0 58000.0]
 [nan 52000.0]
 [48.0 79000.0]
 [50.0 83000.0]
 [37.0 67000.0]]
```

fit() : es utilizado para generar parámetros de modelo de aprendizaje a partir de datos de entrenamiento.
transform(): parámetros generados a partir del método fit() son aplicados al modelo para generar un conjunto de datos transformado.
Estos métodos se utilizan para centrar / escalar características de un dato dado. Básicamente ayuda a normalizar los datos dentro de un rango particular.

```
>>> from sklearn.impute import SimpleImputer
>>> imputer = SimpleImputer(strategy = "mean")
>>> imputer = imputer.fit(X[:,1:3])
>>> X[:,1:3] = imputer.transform(X[:,1:3])
>>> print(X)
[['Brasil' 44.0 72000.0]
 ['Argentina' 27.0 48000.0]
 ['Chile' 30.0 54000.0]
 ['Argentina' 38.0 61000.0]
 ['Chile' 40.0 63777.77777777778]
 ['Brasil' 35.0 58000.0]
 ['Argentina' 38.77777777777778 52000.0]
 ['Brasil' 48.0 79000.0]
 ['Chile' 50.0 83000.0]
 ['Brasil' 37.0 67000.0]]
>>>
```

DATOS CATEGÓRICOS

Tipos de variables



Datos categóricos: son las columnas Pais y Compro, no son datos numéricos. Entonces la primera face de limpieza es traducir a números estos datos

```
# ejemplo 4
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer

dataset = pd.read_csv('<ruta de archivo>/Datos.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, 3].values
imputer = SimpleImputer(missing_values = np.nan, strategy="mean")
imputer = imputer.fit(X[:,1:3])
X[:,1:3] = imputer.transform(X[:,1:3])

from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
```

Cualquier modelo tomará esos números en forma ordinal (0 < 1 < 2) pero los países no son comparables. Aparece el concepto de variables **dummy o One Hot Encode**

fit_transform(): combinación de fit()y transform() en el mismo conjunto de datos .
LabelEncoder (): codifica etiquetas de una característica categórica en valores numéricos

```
>>> print(dataset)
       Pais  Edad  Salario Compro
0    Brasil  44.0  72000.0     No
1  Argentina  27.0  48000.0     Si
2     Chile  30.0  54000.0     No
3  Argentina  38.0  61000.0     No
4     Chile  40.0      NaN     Si
5    Brasil  35.0  58000.0     Si
6  Argentina   NaN  52000.0     No
7    Brasil  48.0  79000.0     Si
8     Chile  50.0  83000.0     No
9    Brasil  37.0  67000.0     Si
```

```
>>> print(X[:, 0])
['Brasil' 'Argentina' 'Chile' 'Argentina' 'Chile' 'Brasil' 'Argentina'
 'Brasil' 'Chile' 'Brasil']
>>> labelencoder_X = LabelEncoder()
>>> X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
>>> print(X[:, 0])
[1 0 2 0 2 1 0 1 2 1]
>>> print(X)
[[1 44.0 72000.0]
 [0 27.0 48000.0]
 [2 30.0 54000.0]
 [0 38.0 61000.0]
 [2 40.0 63777.77777777778]
 [1 35.0 58000.0]
 [0 38.77777777777778 52000.0]
 [1 48.0 79000.0]
 [2 50.0 83000.0]
 [1 37.0 67000.0]]
>>>
```

VARIABLES DUMMY

Variables Dummy

```
# ejemplo 5
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder

dataset = pd.read_csv('<ruta de archivo>/Datos.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 3].values
imputer = SimpleImputer(missing_values = np.nan, strategy="mean")
imputer = imputer.fit(X[:,1:3])
X[:,1:3] = imputer.transform(X[:,1:3])

labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
onehotencoder = make_column_transformer((OneHotEncoder(), [0]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)

labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
```

```
>>> labelencoder_y = LabelEncoder()
>>> y = labelencoder_y.fit_transform(y)
>>> print(y)
[0 1 0 0 1 1 0 1 0 1]
```

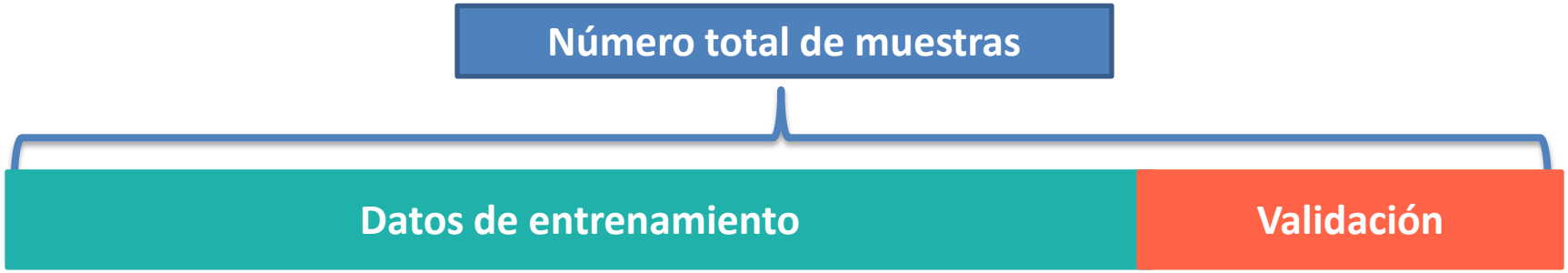
```
>>> print(y)
['No' 'Si' 'No' 'No' 'Si' 'Si' 'No' 'Si' 'No' 'Si']
```

make_column_transformer : permite aplicar transformaciones de datos de forma selectiva a diferentes columnas de su conjunto de datos

```
>>> print(X)
[[1 44.0 72000.0]
 [0 27.0 48000.0]
 [2 30.0 54000.0]
 [0 38.0 61000.0]
 [2 40.0 63777.77777777778]
 [1 35.0 58000.0]
 [0 38.77777777777778 52000.0]
 [1 48.0 79000.0]
 [2 50.0 83000.0]
 [1 37.0 67000.0]]
```

```
>>> print(X)
[[0.0 1.0 0.0 44.0 72000.0]
 [1.0 0.0 0.0 27.0 48000.0]
 [0.0 0.0 1.0 30.0 54000.0]
 [1.0 0.0 0.0 38.0 61000.0]
 [0.0 0.0 1.0 40.0 63777.77777777778]
 [0.0 1.0 0.0 35.0 58000.0]
 [1.0 0.0 0.0 38.77777777777778 52000.0]
 [0.0 1.0 0.0 48.0 79000.0]
 [0.0 0.0 1.0 50.0 83000.0]
 [0.0 1.0 0.0 37.0 67000.0]]
```

DIVIDIR EL DATASET



Aproximadamente entre el 70% y 80% del conjunto de datos se utilizará para la fase de entrenamiento, mientras que porcentaje restante será utilizado para evaluar, o sea para la fase de testing, definimos:

X_train : es el conjunto de entrenamiento, las variables independientes que se utilizarán para entrenar el algoritmo.
X_test : datos con los cuales se va a testear que el algoritmo funciona correctamente,
y_train: los valores de predicción que también se suministran al algoritmo para que aprenda a predecirlos .
y_test : se usa para validar si las predicciones de testing, son o no son correctas, para evaluar la performance, la eficacia del algoritmo.

Posibles problemas:

Overfitting o sobre ajuste: ocurre cuando se desempeña bien con los datos de entrenamiento, pero su precisión es notablemente más baja con los datos de evaluación; esto se debe a que el modelo ha memorizado los datos que ha visto y no pudo generalizar las reglas para predecir los datos que no ha visto.

Underfitting o subajuste: se refiere a un modelo que no puede modelar los datos de entrenamiento, no puede generalizar a nuevos datos, esto ocurre cuando el modelo es muy simple.

Dividir el dataset en conjunto de entrenamiento y conjunto de test

```
# ejemplo 6
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

dataset = pd.read_csv('<ruta de archivo>/Datos.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 3].values
imputer = SimpleImputer(missing_values = np.nan, strategy="mean")
imputer = imputer.fit(X[:,1:3])
X[:,1:3] = imputer.transform(X[:,1:3])

labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])

onehotencoder = make_column_transformer((OneHotEncoder(), [0]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)

labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```
>>> print(X_train)
[[0.0 0.0 1.0 40.0 63777.77777777778]
 [0.0 1.0 0.0 37.0 67000.0]
 [1.0 0.0 0.0 27.0 48000.0]
 [1.0 0.0 0.0 38.77777777777778 52000.0]
 [0.0 1.0 0.0 48.0 79000.0]
 [1.0 0.0 0.0 38.0 61000.0]
 [0.0 1.0 0.0 44.0 72000.0]
 [0.0 1.0 0.0 35.0 58000.0]]
>>> print(X_test)
[[0.0 0.0 1.0 30.0 54000.0]
 [0.0 0.0 1.0 50.0 83000.0]]
>>> print(y_train)
[1 1 1 0 1 0 0 1]
>>> print(y_test)
[0 0]
>>>
```

train_test_split(): permite dividir un dataset en bloques, típicamente bloques destinados al entrenamiento y validación del modelo.

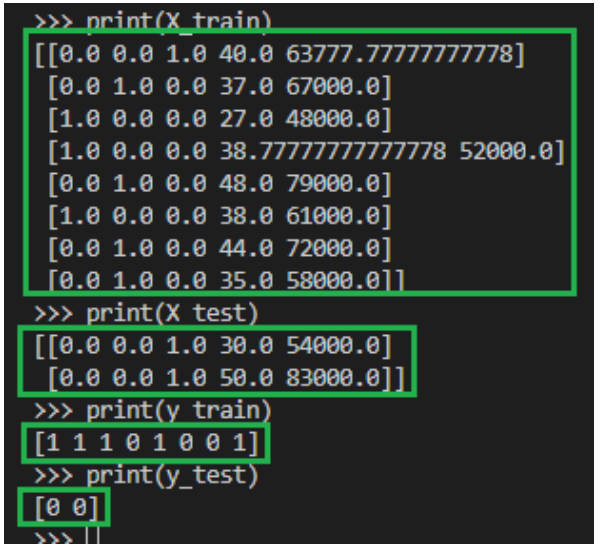
ESCALAR LOS DATOS

Ocurre que la columna de Edad y la columna de Salario no se encuentran dentro del mismo rango de valores. La diferencia es grande por lo tanto el algoritmo lo tiene que tener en cuenta. La solución a la diferencia es escalar los datos.

Escalar significa normalizar los datos para que estén definidos en el mismo rango de valores, ejemplo típico es escalar entre -1 y 1, de modo que la Edad más pequeña correspondería al valor -1 y la mayor a 1. Con el Salario (u otras columnas numéricas) es igual, el Salario menor corresponde a -1 y a partir de ahí se escalaría linealmente hasta 1 que representaría el Salario más elevado.

Esta normalización o estandarización es muy importante para evitar que unas variables dominen sobre otras, dentro del algoritmo de ML y que el propio algoritmo pueda discernir que peso le dará a cada una de las variables, no por tener un rango grande o pequeño sino porque aportan en el proceso de predicción o clasificación.

Hay que hacer lo mismo con y train, y_test ? en este caso no, porque lo que se quiere es clasificar compra o no compra. No obstante hay casos, como la regresión lineal , donde se recomienda que se realice la normalización del vector de datos a predecir para guardar una consistencia.



Escalar los datos

```
# ejemplo 7
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.model_selection import train_test_split

dataset = pd.read_csv('<ruta de archivo>/Datos.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 3].values

imputer = SimpleImputer(missing_values = np.nan, strategy="
mean")
imputer = imputer.fit(X[:,1:3])
X[:,1:3] = imputer.transform(X[:,1:3])

labelencoder_X = LabelEncoder()
#calcula los datos categóricos y sobreescribe
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])

onehotencoder = make_column_transformer((OneHotEncoder
(), [0]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)

labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.2, random_state = 0)
```

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```
>>> print(X_train)
[[-0.77459667 -1.          2.64575131  0.26306757  0.12381479]
 [-0.77459667  1.         -0.37796447 -0.25350148  0.46175632]
 [ 1.29099445 -1.         -0.37796447 -1.97539832 -1.53093341]
 [ 1.29099445 -1.         -0.37796447  0.05261351 -1.11141978]
 [-0.77459667  1.         -0.37796447  1.64058505  1.7202972 ]
 [ 1.29099445 -1.         -0.37796447 -0.0813118  -0.16751412]
 [-0.77459667  1.         -0.37796447  0.95182631  0.98614835]
 [-0.77459667  1.         -0.37796447 -0.59788085 -0.48214934]]
>>> print(X_test)
[[-0.77459667 -1.          2.64575131 -1.45882927 -0.90166297]
 [-0.77459667 -1.          2.64575131  1.98496442  2.13981082]]
```

```
[['Brasil' 44.0 72000.0]
 ['Argentina' 27.0 48000.0]
 ['Chile' 30.0 54000.0]
 ['Argentina' 38.0 61000.0]
 ['Chile' 40.0 63777.77777777778]
 ['Brasil' 35.0 58000.0]
 ['Argentina' 38.77777777777778 52000.0]
 ['Brasil' 48.0 79000.0]
 ['Chile' 50.0 83000.0]
 ['Brasil' 37.0 67000.0]]
```

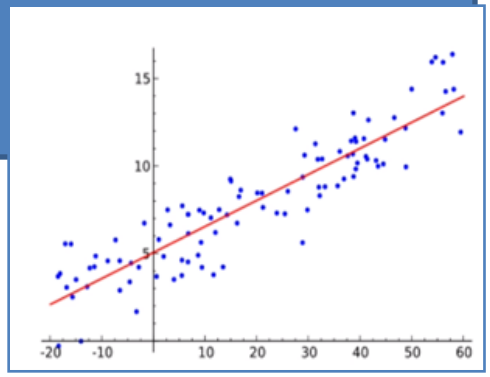
Tenemos los datos listos!!

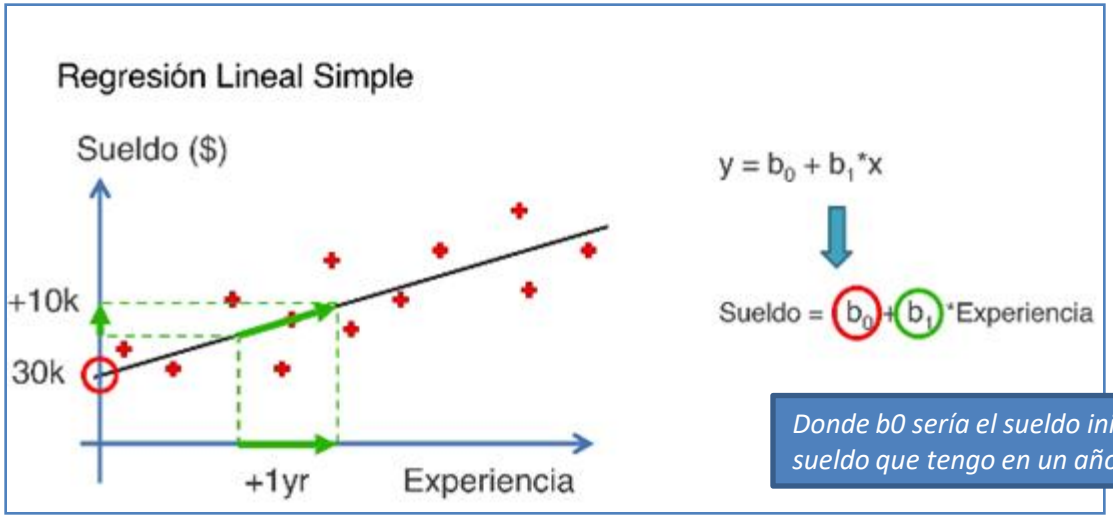
REGRESIÓN LINEAL SIMPLE

Regresión Lineal

La regresión lineal es un algoritmo de aprendizaje supervisado que se utiliza en Machine Learning y en estadística. La regresión lineal es una aproximación para modelar la relación entre una variable escalar dependiente "y" y una o más variables explicativas "X".

La idea es dibujar una recta que indicará la tendencia del conjunto de datos.





Donde b0 sería el sueldo inicial, 30k, y b1 es el incremento de sueldo que tengo en un año de experiencia, en el ejemplo es 10k

El algortimo de Regresión Lineal Simple utiliza el método de los mínimos cuadrados para hallar la mejor recta que se ajusta a los datos. Es decir, de todas las rectas se queda con aquella que minimiza los cuadrados de las diferencias entre el dato real y la predicción.

```
#Ejemplo de Regresión lineal simple
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

dataset = pd.read_csv('<ruta de archivo>/Datos_de_salarios.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

print(dataset.shape)
print(dataset.describe())
```

Aquí tendremos como variable independiente o matriz de características (X) los años de experiencia por un lado, y el sueldo es la variable dependiente (y) aquella que el modelo de regresión lineal va a intentar predecir.

```
>>> print(dataset)
    AniosExperiencia   Salario
0                1.1   39343.0
1                1.3   46205.0
2                1.5   37731.0
3                2.0   43525.0
4                2.2   39891.0
5                2.9   56642.0
6                3.0   60150.0
7                3.2   54445.0
8                3.2   64445.0
9                3.7   57189.0
10               3.9   63218.0
11               4.0   55794.0
12               4.0   56957.0
13               4.1   57081.0
14               4.5   61111.0
15               4.9   67938.0
```

```
>>> print(dataset.shape)
(30, 2)
>>> print(dataset.describe())
       AniosExperiencia        Salario
count         30.000000      30.000000
mean           5.313333   76003.000000
std            2.837888   27414.429785
min            1.100000   37731.000000
25%            3.200000   56720.750000
50%            4.700000   65237.000000
75%            7.700000  100544.750000
max           10.500000  122391.000000
```

```
# Ejemplo de Regresión lineal simple
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

dataset = pd.read_csv('<ruta de archivo>/Datos_de_salarios.csv')
print(dataset)
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
print(dataset.shape)
print(dataset.describe())

dataset.plot(x='AniosExperiencia', y='Salario', style='o')
plt.title('Salario vs Años de Experiencia')
plt.ylabel('Salario')
plt.xlabel('Años de Experiencia')
plt.show()
```

```
# Ejemplo de  Regresión lineal simple
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

dataset = pd.read_csv('<ruta de archivo>/Datos_de_salarios.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

print(dataset.shape)
print(dataset.describe())

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)
```

	X_test	y_test
0	1.5	37731.0
1	10.3	122391.0
2	4.1	57081.0
3	3.9	63218.0
4	9.5	116969.0
5	8.7	109431.0
6	9.6	112635.0
7	4.0	55794.0
8	5.3	83088.0
9	7.9	101302.0

Aquí tenemos los valores exactos de lo que ganan los individuos, los 20 que formarán parte del proceso de entrenamiento.

Con estos datos crearemos el modelo de regresión lineal y luego comprobaremos si el modelo se comporta bien o no, utilizando los elementos de testing.

X_test se suministrará al modelo que se ha creado con los elementos de entrenamiento y miraremos si la predicción que elabora el modelo con estos datos de test se asemeja o no al vector **y_test**. Es la idea global del algoritmo de regresión lineal simple.

	X_train	y_train
0	2.9	56642.0
1	5.1	66029.0
2	3.2	64445.0
3	4.5	61111.0
4	8.2	113812.0
5	6.8	91738.0
6	1.3	46205.0
7	10.5	121872.0
8	3.0	60150.0
9	2.2	39891.0
10	5.9	81363.0
11	6.0	93940.0
12	3.7	57189.0
13	3.2	54445.0
14	9.0	105582.0
15	2.0	43525.0
16	1.1	39343.0
17	7.1	98273.0
18	4.9	67938.0
19	4.0	56957.0

```
# Ejemplo de Regresión lineal simple
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

dataset = pd.read_csv('<ruta de archivo>/Datos_de_salarios.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

print(dataset.shape)
print(dataset.describe())

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)

from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(X_train, y_train)

y_pred = regression.predict(X_test)
```

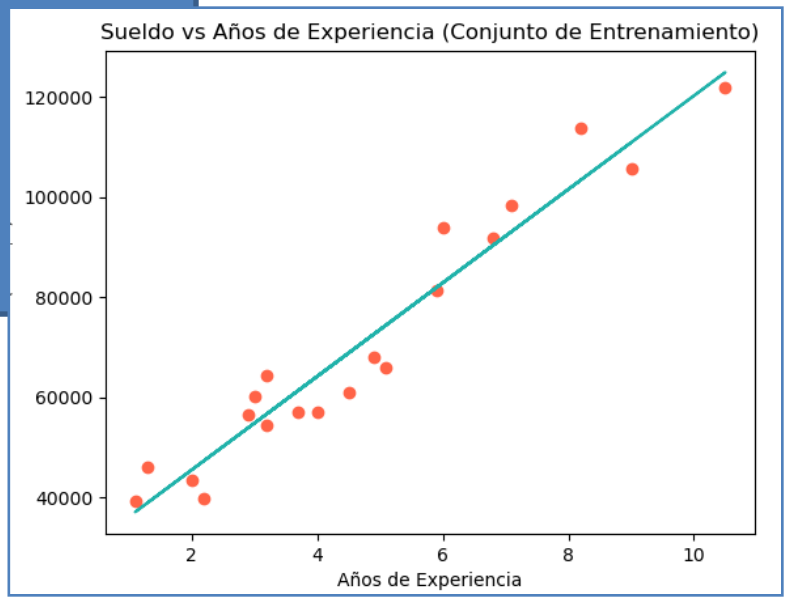
	X_train	y_train
0	2.9	56642.0
1	5.1	66029.0
2	3.2	64445.0
3	4.5	61111.0
4	8.2	113812.0
5	6.8	91738.0
6	1.3	46205.0
7	10.5	121872.0
8	3.0	60150.0
9	2.2	39891.0
10	5.9	81363.0
11	6.0	93940.0
12	3.7	57189.0
13	3.2	54445.0
14	9.0	105582.0
15	2.0	43525.0
16	1.1	39343.0
17	7.1	98273.0
18	4.9	67938.0
19	4.0	56957.0

Le damos al modelo los datos de entrenamiento. Entonces, utilizando la técnica de los mínimos cuadrados, basado en la experiencia de aprendizaje, se creó un modelo y este, será capaz de predecir con nuevos años de experiencia que tenga un trabajador en la empresa, cuál es el sueldo que le corresponde.

Ahora se pueden predecir las observaciones nuevas, que son las del conjunto de pruebas **X_test** y que no ha sido utilizado en la creación del modelo lineal. La función **predict** se encarga de hacer las predicciones.

```
# Ejemplo de Regresión Lineal Simple
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

dataset = pd.read_csv('<ruta de archivo>/Datos_de_salarios.csv')
print(dataset)
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
print(dataset.shape)
print(dataset.describe())
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)

from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(X_train, y_train)
y_pred = regression.predict(X_test)

df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicción': y_pred.flatten()})

plt.scatter(X_train, y_train, color = "#FF6347")
plt.plot(X_train, regression.predict(X_train), color = "#20B2AA")
plt.title("Sueldo vs Años de Experiencia (Conjunto de Entrenamiento)")
plt.xlabel("Años de Experiencia")
plt.ylabel("Sueldo")
plt.show()
```

```
>>> print(df)
      Actual     Predicción
0    37731.0   40835.105909
1   122391.0  123079.399408
2    57081.0   65134.556261
3    63218.0   63265.367772
4   116969.0  115602.645454
5   109431.0  108125.891499
6   112635.0  116537.239698
7    55794.0   64199.962017
8    83088.0   76349.687193
9   101302.0  100649.137545
```



```
# Ejemplo de Regresión Lineal Simple
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

dataset = pd.read_csv('<ruta de archivo>/Datos_de_salarios.csv')
print(dataset)
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
print(dataset.shape)
print(dataset.describe())
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(X_train, y_train)

y_pred = regression.predict(X_test)

df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicción': y_pred.flatten()})

plt.scatter(X_test, y_test, color =  "#FF6347"))
plt.plot(X_train, regression.predict(X_train), color = "#20B2AA")
plt.title("Sueldo vs Años de Experiencia (Conjunto de Testing)")
plt.xlabel("Años de Experiencia")
plt.ylabel("Sueldo")
plt.show()
```

```
# Ejemplo de Regresión Lineal Simple

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

dataset = pd.read_csv('<ruta de archivo>/Datos_de_salarios.csv')
print(dataset)
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
print(dataset.shape)
print(dataset.describe())
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(X_train, y_train)

y_pred = regression.predict(X_test)

from sklearn import metrics

print('Error Medio Absoluto (MAE):', metrics.mean_absolute_error(y_test, y_pred))
print('Error cuadrático medio (MSE):', metrics.mean_squared_error(y_test, y_pred))
print('Raíz cuadrada del error cuadrático medio (RMSE) :', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

Para los algoritmos de regresión, se utilizan comúnmente tres métricas de evaluación:
El error medio absoluto (MAE) es la media del valor absoluto de los errores.
El error cuadrático medio (MSE) es la media de los errores al cuadrado
Raíz cuadrada del error cuadrático medio (RMSE) es la raíz cuadrada de la media de los errores cuadráticos

FIN