



Machine Learning Models

Supervisado - Clasificación



# ML Supervisado Clasificación

- ☐ 1. Preprocesado Previo
  - ☐ 2. Introducción
  - ☐ 3. Regresión Logística
  - ☐ 4. Árbol de Decisión (Decision Tree)
- 

# 1. Preprocesado Previo

# Preprocesado previo

Los modelos de ML en general y los supervisados de clasificación en particular necesitan de todo el preprocesamiento que habéis estudiado hasta ahora. O sea:

- ☐ Data cleaning

- ☐ Eliminación de duplicados
- ☐ Eliminación o imputación de valores nulos
- ☐ Tratamiento de Outliers

- ☐ Transformación

- ☐ Conversión de variables categóricas a numéricas
- ☐ Normalización de todas las variables numéricas (excepto la variable a predecir). OJO! LO HAREMOS DENTRO DEL ENTRENAMIENTO!!

## 2. Introducción

# Introducción

## Modelo supervisado:

- ❑ Hay una variable que queremos predecir
- ❑ Para generar el modelo necesitamos datos ya clasificados (Labeled data)

## Modelo no supervisado:

- ❑ NO Hay una variable que queremos predecir
- ❑ No tenemos datos ya clasificados (Unlabeled data)

## Modelo supervisado:

- ❑ **Clasificación:** La variable a predecir es categórica (nos centraremos en dicotómica, 2 categorías)
- ❑ **Regresión:** La variable a predecir es numérica (continua)

## Data in Supervised vs. Unsupervised Learning

Supervised Learning

Labeled Data

Unsupervised Learning

Unlabeled Data



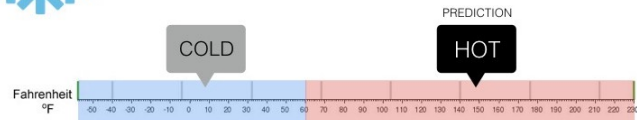
### Regression

What is the temperature going to be tomorrow?



### Classification

Will it be Cold or Hot tomorrow?

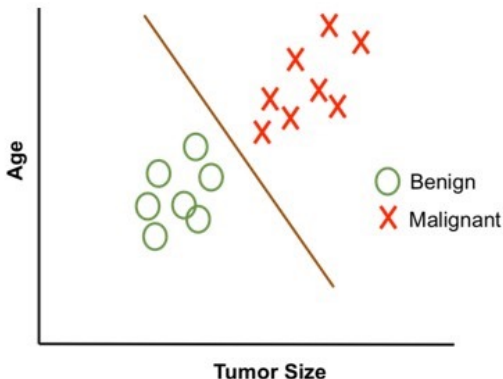


# Introducción

## Ejemplo modelo supervisado de clasificación:

- ❑ Hay una variable que queremos predecir (Tumor Benigno / Tumor Maligno)
- ❑ Para generar el modelo necesitamos datos ya clasificados

|           |   |
|-----------|---|
| Feature   | Tumor Age and Tumor Size  |
| Label     | Tumor (Benign or Malignant)   |
| Goal/ Aim | We want to predict whether a tumor is benign or malignant from the given age and tumor size |



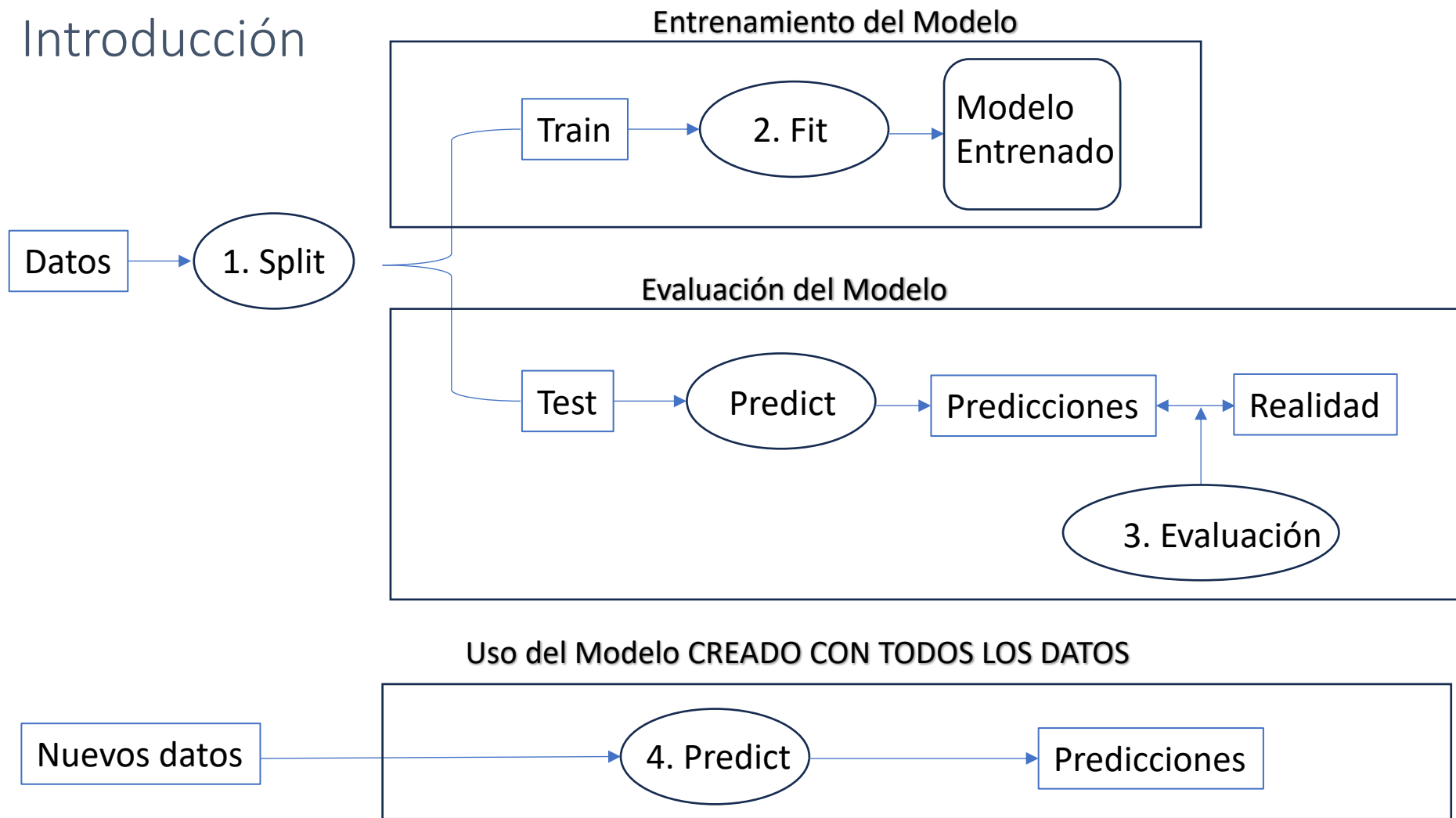
# Introducción

## PASOS PROCESO DE ML SUPERVISADO

- ☐ 0. Separamos las variables que sirven para predecir (X) de la que queremos predecir (y)
- ☐ 1. Split. Separamos los datos en dos conjuntos
  - ☐ Train (Conjunto para entrenar el modelo)
  - ☐ Test (Conjunto para Evaluar el modelo)
- ☐ 2. Preprocesado de estandarización controlado. Lo fitamos en el train y lo aplicamos en el train y el test
- ☐ 3. Fit. Entrenamos el modelo
  - ☐ Creamos el modelo que vamos a usar
  - ☐ Entrenamos el modelo sobre el conjunto de Train y obtenemos el Modelo ya entrenado (preparado para hacer predicciones)
- ☐ 4. Evaluación. Predecimos sobre el conjunto de test para evaluar hasta qué punto nuestro modelo predice bien.
- ☐ 5. Uso del modelo. Creamos el modelo con todos los datos y lo usamos para predecir sobre nuevos datos



# Introducción



# 3. Regresión Logística



# Regresión Logística

**Función Logística:** La regresión logística utiliza una función logística, también conocida como sigmoide, para transformar la salida del modelo en un rango entre 0 y 1. Esta función es fundamental y se expresa como:

$$\frac{1}{1+e^{-z}}$$

Donde  $z$  es la combinación lineal de las variables independientes.

**Proceso de Funcionamiento:**

**Combinación Lineal:** La regresión logística realiza una combinación lineal de las variables independientes ponderadas por los coeficientes del modelo.

$$z = b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + \dots + b_n \cdot x_n$$

Donde  $b_0, b_1, \dots, b_n$  son los coeficientes del modelo y  $x_1, x_2, \dots, x_n$  son las variables independientes.

La combinación lineal  $z$  se introduce en la función logística, que produce una salida en el rango de 0 a 1.

**Umbral de Decisión:** Se establece un umbral (generalmente 0.5) para decidir a qué clase pertenece la instancia. Si la probabilidad calculada es mayor que el umbral, la instancia se clasifica en la clase 1; de lo contrario, se clasifica en la clase 0.

# Regresión Logística

❑ Objetivo: Modelizar la predicción de una variable dicotómica

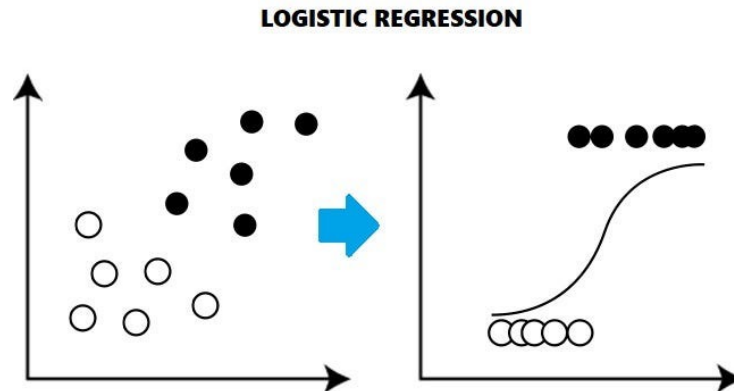
❑ Tumor Benigno / Tumor Maligno

❑ Diabético / No diabético

❑ Sobrevive / No sobrevive

❑ Resultado

Ecuación (Como en regresión lineal)



# Regresión Logística

## EJEMPLO. DATOS DE COMPRA

### ▼ Importar los datos

✓  
0 s



```
df = pd.read_csv('Social_Network_Ads.csv')  
df.sample(8)
```



|     | Age | EstimatedSalary | Purchased |
|-----|-----|-----------------|-----------|
| 196 | 30  | 79000           | 0         |
| 113 | 37  | 55000           | 0         |
| 143 | 30  | 89000           | 0         |
| 308 | 36  | 125000          | 1         |
| 362 | 47  | 50000           | 1         |
| 193 | 19  | 70000           | 0         |
| 385 | 56  | 60000           | 1         |
| 336 | 58  | 144000          | 1         |



# Regresión Logística

## PASOS ML SUPERVISADO CON PYTHON. REGRESIÓN LOGÍSTICA

- ✓ 0. Separar la variable a predecir y las predictoras

```
[ ] X=df.drop(columns=['Purchased'],inplace=False)
    y=df['Purchased']
```

- ✓ 1. Split. Separar los datos en conjunto de entrenamiento (train) y conjunto de evaluación o test(test)

```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

# Regresión Logística

## EJEMPLO. DATOS DE COMPRA

### ✓ 2. Estandarización controlada. Transformación a numéricas no hace falta

- Fit en el train
- Transform en el train
- Transform en el test

```
✓  
s  ▶ from sklearn.preprocessing import StandardScaler  
    estandarizador = StandardScaler()  
    estandarizador.fit(X_train)  
    X_train_std=estandarizador.transform(X_train)  
    X_test_std=estandarizador.transform(X_test)
```

```
✓  
s  ▶ print("Tamaño X_train",X_train.shape)  
    print("Tamaño X_test",X_test.shape)  
    print("Tamaño X_train std",X_train_std.shape)  
    print("Tamaño X_test std",X_test_std.shape)
```

```
Tamaño X_train (300, 2)  
Tamaño X_test (100, 2)  
Tamaño X_train std (300, 2)  
Tamaño X_test std (100, 2)
```

# Regresión Logística

## PASOS ML SUPERVISADO CON PYTHON. REGRESIÓN LOGÍSTICA

### ✓ 3. Fit. Entrenar el modelo

- Creamos el modelo
- Entrenamos el modelo sobre los datos de train y obtenemos el modelo entrenado

```
✓ [9] # Cargamos el modelo y lo creamos  
s     from sklearn.linear_model import LogisticRegression  
      LR = LogisticRegression()  
      # Ahora LR ya es un modelo que se puede entrenar (fit)
```

```
✓ [10] # Entrenamos el modelo dtree  
s      LR.fit(X_train_std,y_train)  
      # Ahora LR es un modelo entrenado capaz de hacer predicciones
```

```
▼ LogisticRegression  
LogisticRegression()
```



# Regresión Logística

## PASOS ML SUPERVISADO CON PYTHON. REGRESIÓN LOGÍSTICA

### ✓ 4. Evaluar el Modelo

- Hacemos predicciones sobre el conjunto de test
- Comparamos esas predicciones con los valores reales. Calculamos la precisión (accuracy)

```
✓ 0 s ▶ from sklearn.metrics import accuracy_score
# Hacemos predicciones sobre el conjunto de test
predictions = LR.predict(X_test_std)

# Calculamos la accuracy (porcentaje de observaciones con predicción correcta)
accuracy_score(y_test, predictions)
# Vemos que tenemos una accuracy del 89%. Es el porcentaje de observaciones con predicción correcta
```

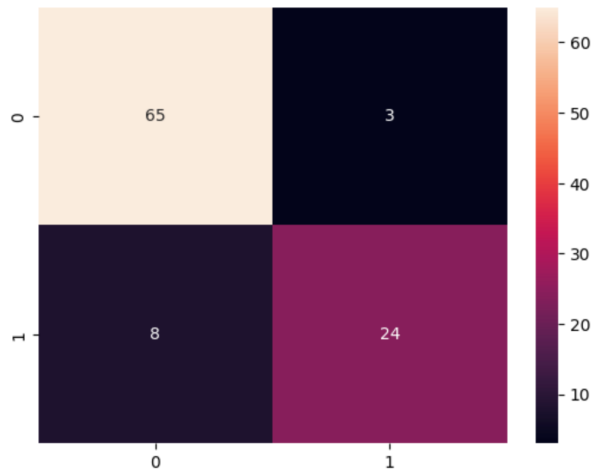
0.89

# Regresión Logística

- Hay otras métricas que podemos usar. Todas aparecen a partir de la matriz de confusión

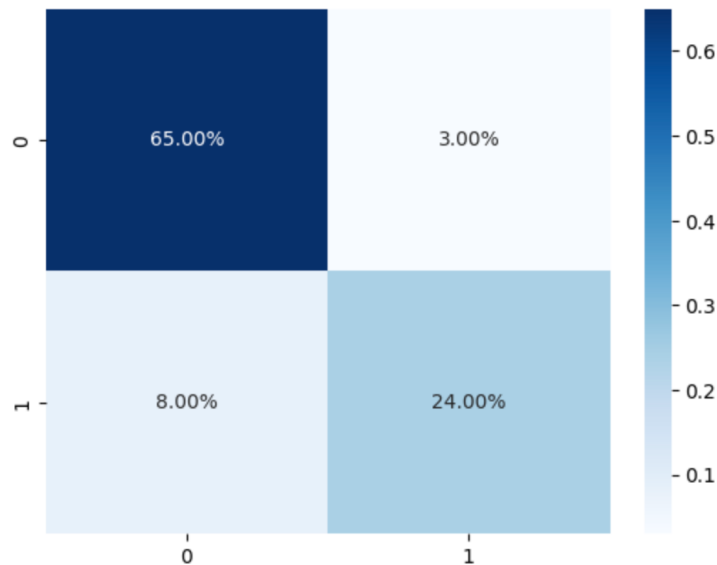
```
[63] cm=confusion_matrix(y_test, predictions)
import seaborn as sns
sns.heatmap(cm, annot=True)
```

<Axes: >



```
0 s sns.heatmap(cm/np.sum(cm), annot=True,
fmt='.2%', cmap='Blues')
```

<Axes: >



Las predicciones 0 1 están en columnas y la realidad 0 1 en filas. Por lo tanto, de la matriz de confusión podemos decir que:

- Hay 65 valores que se han clasificado bien como 0 y realmente eran cero (Verdaderos positivos)
- Hay 8 valores que se han clasificado mal como 0 y en realidad eran 1 (Falsos negativos)
- Hay 3 valores que se han clasificado mal como 1 y en realidad eran 0 (Falsos positivos)
- Hay 24 valores que se han clasificado bien como 1 y realmente eran 1 (Verdaderos positivos)

# Regresión Logística

Conceptos importantes en modelos supervisados de clasificación

| Predicted class      |  | Actual class |
|----------------------|--|--------------|
| True Positives (TP)  | False Negatives (FN)                   |              |
| False Positives (FP) | True Negatives (TN)                    |              |
| Measure              | formula                                |              |
| Accuracy             | (TP+TN)/(TP+FP+FN+TN)                  |              |
| Precision            | TP/ (TP+FP)                            |              |
| Recall               | TP/ (TP+FN)                            |              |
| F-Measure            | 2*Precision*Recall/ (Precision+Recall) |              |

```
[65] # Para hacerlos en Python
      from sklearn.metrics import precision_score
      precision_score(y_test, predictions)
```

0.8888888888888888

```
[66] from sklearn.metrics import recall_score
      recall_score(y_test, predictions)
```

0.75

```
[67] from sklearn.metrics import f1_score
      f1_score(y_test, predictions)
```

0.8135593220338982

# Regresión Logística

## ✓ 5. Creamos el modelo final

- Creamos una nueva estandarización con todos los datos
- Creamos el modelo con todos los datos
- Guardamos el estandarizador y el modelo

```
▶ # Creamos una nueva estandarización con todos los datos
estandarizador2 = StandardScaler()
estandarizador2.fit(X)
X_std=estandarizador2.transform(X)
#Creamos /fitamos el modelo con todos los datos
LR.fit(X_std,y)
```



```
▼ LogisticRegression
LogisticRegression()
```

✓  
0 s

```
[15] # Guardamos el estandarizador
from joblib import dump
dump(estandarizador2, 'estandarizador2.std') # Guardamos el estandarizador
# Guardamos el modelo
dump(LR, 'logistic_regression.joblib')
```

```
['logistic_regression.joblib']
```

# Regresión Logística

## 6. Usamos el modelo

- Volvemos a cargar el estandarizador y el modelo simulando que lo recuperamos
- Creamos los nuevos datos sobre los que queremos usar / hacer predicciones
- Los estandarizamos con el estandarizador recuperado
- Predecimos la variable 'Purchased' para esos nuevos datos

```
[16] # Recuperamos el estandarizador y el modelo
      from joblib import load
      estandarizador = load('estandarizador2.std')
      regression_model = load('logistic_regression.joblib')
```

```
[18] # Creamos unos datos nuevos para hacer predicción
      X_new={'Age': [39,56], 'EstimatedSalary': [20000,80000]}
      datos_nuevos=pd.DataFrame(X_new)
      datos_nuevos
```

|   | Age | EstimatedSalary |
|---|-----|-----------------|
| 0 | 39  | 20000           |
| 1 | 56  | 80000           |

```
[19] # Estandarizamos los nuevos datos
      datos_nuevos_std=estandarizador.transform(datos_nuevos)
      datos_nuevos_std
```

```
array([[ 0.12846516, -1.46068138],
       [ 1.75218836,  0.30121002]])
```

```
[21] # Predecimos la variable 'Purchased' para esos valores
      new_predictions = regression_model.predict(datos_nuevos_std)
      print(new_predictions)
      # Nos predice que el primer caso no compran y el segundo si
```

[0 1]

# 4. Decision Tree

A dark blue, abstract, curved shape that starts from the bottom left and extends towards the bottom right, resembling a stylized wave or a decorative footer element.

# Decision Tree

## Conceptos Básicos:

- 1.Nodos y Ramas:** Un árbol de decisión consta de nodos y ramas. Cada nodo representa una característica o una pregunta sobre los datos, y cada rama representa una posible respuesta a esa pregunta.
- 2.Raíz y Hojas:** El nodo superior se llama la raíz, y los nodos finales sin ramas se llaman hojas. Cada hoja representa una decisión o una clasificación.
- 3.Características de Entrada:** En cada nodo, se realiza una pregunta sobre una característica específica de los datos.
- 4.División:** Los datos se dividen en subconjuntos en función de las respuestas a las preguntas. Esto se repite recursivamente para crear subárboles hasta que se alcanza una condición de parada.
- 5.Etiquetas y Predicciones:** Cada hoja tiene una etiqueta que representa la decisión o la clasificación que se asigna a las instancias de datos que llegan a esa hoja. Durante la predicción, una instancia de datos desciende por el árbol, siguiendo las ramas según sus características, hasta llegar a una hoja y recibir la etiqueta de esa hoja como la predicción final.

# Decision Tree

## Proceso de Funcionamiento:

**1. Selección de Características:** En cada nodo, se selecciona la característica que mejor separa los datos en función de algún criterio, como la ganancia de información o la impureza de Gini.

**2. División de Datos:** Los datos se dividen en subconjuntos en función de los valores de la característica seleccionada.

**3. Recursividad:** Se repite el proceso para cada subconjunto, creando subárboles.

**4. Condición de Parada:** Se define una condición de parada para detener la construcción del árbol, como alcanzar una profundidad máxima o tener un número mínimo de instancias en un nodo.

**5. Predicción:** Durante la predicción, una instancia de datos desciende por el árbol según sus características hasta llegar a una hoja, donde se realiza la predicción.

## Ventajas:

- Fácil de entender y visualizar.
- Puede manejar datos numéricos y categóricos.
- No requiere normalización de datos.

*En resumen, un árbol de decisión es como un conjunto de preguntas "si-sino" que se hacen sobre las características de los datos para llegar a una decisión final*



# Decision Tree

❑ Objetivo: Modelizar la predicción de una variable dicotómica

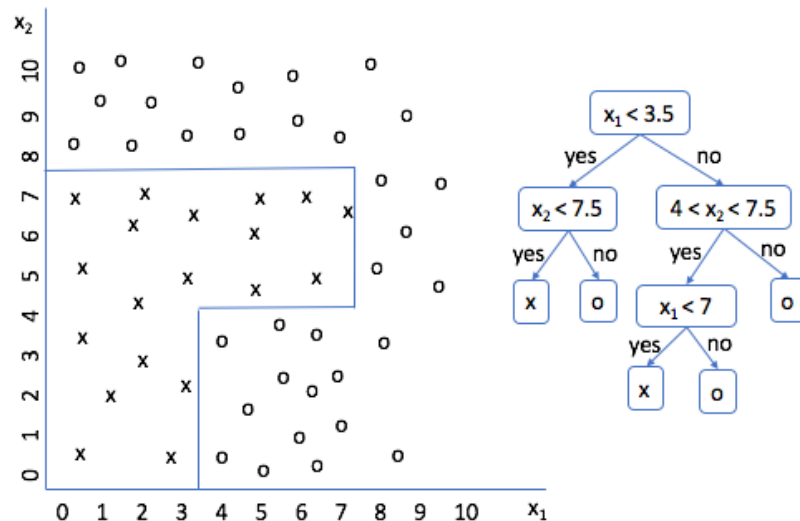
❑ Tumor Benigno / Tumor Maligno

❑ Diabético / No diabético

❑ Sobrevive / No sobrevive

❑ Resultado

Árbol con reglas



# Decision Tree

✓ DECISION TREE. No usamos la estandarización

✓ 0. Separar la variable a predecir y las predictoras

```
✓ [15] X=df.drop(columns=['Purchased'], inplace=False)  
0s y=df['Purchased']
```

✓ 1. Split. Separar los datos en conjunto de entrenamiento (train) y conjunto de evaluación o test(test)

```
✓ [16] from sklearn.model_selection import train_test_split  
0s X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

## 2. Estandarización

- En este caso no es necesario y no lo hacemos

# Decision Tree

## ✓ 3. Fit. Entrenar el modelo

- Creamos el modelo
- Entrenamos el modelo sobre los datos de train y obtenemos el modelo entrenado

✓  
0 s

```
[19] # Cargamos el modelo y lo creamos
      from sklearn.tree import DecisionTreeClassifier
      dtree = DecisionTreeClassifier(max_depth=3)
      # Ahora dtree ya es un modelo que se puede entrenar (fit)
```

✓  
0 s

```
[20] # Entrenamos el modelo dtree
      dtree.fit(X_train,y_train)
      # Ahora dtree es un modelo entrenado capaz de hacer predicciones
```

▼ DecisionTreeClassifier  
DecisionTreeClassifier(max\_depth=3)

✓  
0 s



```
# Podemos ver diferentes aspectos de la configuración una vez el modelo está fitado
print(dtree.criterion)
print(dtree.classes_)
```



```
gini
[0 1]
```

# Decision Tree

## ✓ 4. Evaluar el Modelo

- Hacemos predicciones sobre el conjunto de test
- Comparamos esas predicciones con los valores reales. Calculamos la precisión (accuracy)

✓  
0 s

```
from sklearn.metrics import confusion_matrix, accuracy_score
# Hacemos predicciones sobre el conjunto de test
predictions = dtree.predict(X_test)

# Calculamos la accuracy (porcentaje de observaciones con predicción correcta)
accuracy_score(y_test, predictions)
# Vemos que tenemos una accuracy del 94%. Es el porcentaje de observaciones con predicción correcta
```

0.94

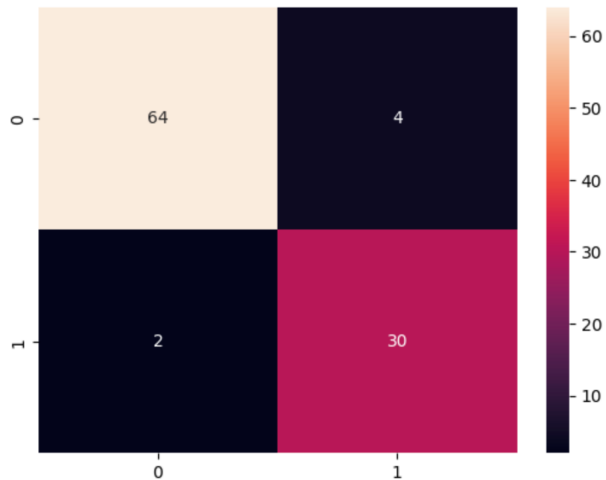
# Decision Tree

- Hay otras métricas que podemos usar. Todas aparecen a partir de la matriz de confusión

✓  
0 s

```
cm=confusion_matrix(y_test, predictions)
import seaborn as sns
sns.heatmap(cm, annot=True)
```

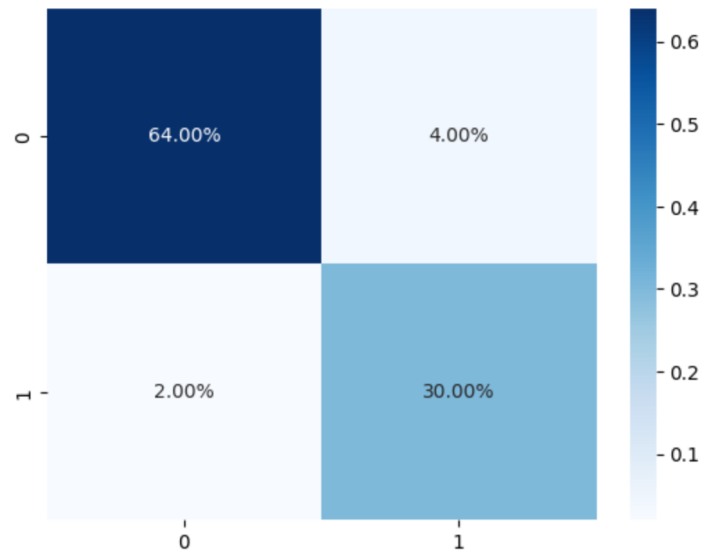
<Axes: >



✓  
0 s

```
sns.heatmap(cm/np.sum(cm), annot=True,
             fmt='.2%', cmap='Blues')
```

<Axes: >



Las predicciones 0 1 están en columnas y la realidad 0 1 en filas. Por lo tanto, de la matriz de confusión podemos decir que:

- Hay 64 valores que se han clasificado como 0 y realmente eran cero (Verdaderos positivos)
- Hay 2 valores que se han clasificado como 0 y en realidad eran 1 (Falsos negativos)
- Hay 4 valores que se han clasificado como 1 y en realidad eran 0 (Falsos positivos)
- Hay 30 valores que se han clasificado como 1 y realmente eran 1 (Verdaderos positivos)

# Decision Tree

Conceptos importantes en modelos supervisados de clasificación

|              |  | Predicted class       |   |         |         |          |                       |           |             |        |             |           |  |
|--------------|--|-----------------------|---|---------|---------|----------|-----------------------|-----------|-------------|--------|-------------|-----------|--|
| Actual class | True Positives (TP)                    | False Negatives (FN)  |   |         |         |          |                       |           |             |        |             |           |  |
|              | False Positives (FP)                   | True Negatives (TN)   |   |         |         |          |                       |           |             |        |             |           |  |
|              |  |                       | <table><tr><th>Measure</th><th>formula</th></tr><tr><td>Accuracy</td><td>(TP+TN)/(TP+FP+FN+TN)</td></tr><tr><td>Precision</td><td>TP/ (TP+FP)</td></tr><tr><td>Recall</td><td>TP/ (TP+FN)</td></tr><tr><td>F-Measure</td><td>2*Precision*Recall/ (Precision+Recall)</td></tr></table> | Measure | formula | Accuracy | (TP+TN)/(TP+FP+FN+TN) | Precision | TP/ (TP+FP) | Recall | TP/ (TP+FN) | F-Measure | 2*Precision*Recall/ (Precision+Recall) |
|              | Measure                                | formula               |   |         |         |          |                       |           |             |        |             |           |  |
|              | Accuracy                               | (TP+TN)/(TP+FP+FN+TN) |   |         |         |          |                       |           |             |        |             |           |  |
| Precision    | TP/ (TP+FP)                            |                       |   |         |         |          |                       |           |             |        |             |           |  |
| Recall       | TP/ (TP+FN)                            |                       |   |         |         |          |                       |           |             |        |             |           |  |
| F-Measure    | 2*Precision*Recall/ (Precision+Recall) |                       |   |         |         |          |                       |           |             |        |             |           |  |

```
[82] # Para hacerlos en Python
      from sklearn.metrics import precision_score
      precision_score(y_test, predictions)
```

0.8823529411764706

```
from sklearn.metrics import recall_score
recall_score(y_test, predictions)
```

0.9375

```
[84] from sklearn.metrics import f1_score
      f1_score(y_test, predictions)
```

0.9090909090909091

# Decision Tree

## ✓ 5. Creamos el modelo final

- Creamos el modelo con todos los datos
- Guardamos el modelo

```
✓ [100] #Creamos /fitamos el modelo con todos los datos  
0 s    dtree.fit(X,y)  
        # Guardamos el modelo  
        dump(dtree, 'decision_tree.joblib')  
  
['decision_tree.joblib']
```

# Decision Tree

## 6. Usamos el modelo

- Volvemos a cargar el modelo simulando que lo recuperamos
- Creamos los nuevos datos sobre los que queremos usar / hacer predicciones
- Predecimos la variable 'Purchased' para esos nuevos datos

```
✓ [101] # Recuperamos el modelo  
0 s from joblib import load  
dt_model = load('decision_tree.joblib')
```

```
✓ ▶ # Creamos unos datos nuevos para hacer predicción  
0 s X_new={'Age': [39,56], 'EstimatedSalary': [20000,80000]}  
datos=pd.DataFrame(X_new)  
datos
```

|   | Age | EstimatedSalary |
|---|-----|-----------------|
| 0 | 39  | 20000           |
| 1 | 56  | 80000           |

```
✓ ▶ # Predecimos la variable 'Purchased' para esos valores  
0 s new_predictions = dt_model.predict(datos)  
print(new_predictions)  
# Nos predice que el primer caso no compran y el segundo si
```

```
[0 1]
```



# Decision Tree

