Árbol de Decisión con dataset Titanic

Importamos las librerías necesarias para el manejo del dataset, así como las librerías para realizar gráficas.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Después realizamos la carga del dataset. Nota: la ruta cambia

```
# Cargando el dataset de Titanic
data_original = pd.read_csv('D:/DESARROLLO/Python/Sistemas_Apoyo_Decision/Arbol_Decision/dataset/titanic.csv')
```

Una vez que cargamos el dataset, mostramos la información que nos arroja dicho dataset para empezar hacer la manipulación del mismo.

```
data_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 887 entries, 0 to 886
Data columns (total 8 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Survived                 887 non-null    int64
 1   Pclass                   887 non-null    int64
 2   Name                     887 non-null    object
 3   Sex                      887 non-null    object
 4   Age                      887 non-null    float64
 5   Siblings/Spouses Aboard  887 non-null    int64
 6   Parents/Children Aboard  887 non-null    int64
 7   Fare                     887 non-null    float64
dtypes: float64(2), int64(4), object(2)
memory usage: 55.6+ KB
```

Observamos que el dataset cuenta con 8 columnas y 887 registros de entrada a analizar.

El dataset cuenta con la siguiente descripción

- Survived (0 = No y 1 = Sí)
- PClass (1 = 1st, 2 = 2nd, 3 = 3rd)
- Name (Nombre)
- Sex (Male = Hombre y Female = Femenino)
- Age (Edad)
- Siblings/Spouses (Hermanos/Esposas)
- Parents/Children (Parientes/Hijos)
- Fare (Tarifa del boleto)

Mostramos los primeros registros para analizar la información del dataset

```
data_original.head()
```

```
   Survived  Pclass  ... Parents/Children Aboard     Fare
0         0       3  ...                      0   7.2500
1         1       1  ...                      0  71.2833
2         1       3  ...                      0   7.9250
3         1       1  ...                      0  53.1000
4         0       3  ...                      0   8.0500
```

Ahora cambiaremos el nombre de unas columnas, acortando el nombre para próximas manipulaciones. Las columnas a cambiar son (*Siblings/Spouses Aboard* y *Parents/Children Aboard*).

```
data_original.rename(columns={'Siblings/Spouses Aboard':'SibSp','Parents/Children Aboard':'Parent'}, inplace=True)
```

```
data_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 887 entries, 0 to 886
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  887 non-null    int64
 1   Pclass    887 non-null    int64
 2   Name      887 non-null    object
 3   Sex       887 non-null    object
 4   Age       887 non-null    float64
 5   SibSp     887 non-null    int64
 6   Parent    887 non-null    int64
 7   Fare      887 non-null    float64
dtypes: float64(2), int64(4), object(2)
memory usage: 55.6+ KB
```

Ahora revisaremos si hay valores nulos en nuestro Dataset.

```
data_original.isnull().sum()
```

```
In [8]: data_original.isnull().sum()
Out[8]:
Survived    0
Pclass      0
Name        0
Sex         0
Age         0
SibSp       0
Parent      0
Fare        0
dtype: int64
```

Podemos observar que en este dataset no hay valores nulos. En el caso de que hubiera valores nulos tendríamos que aplicar un método de llenado (Mediana o por Distribución de Probabilidad). Si hubiera muchos campos nulos se podría descartar esa columna. Un ejemplo podría ser la siguiente línea

data_original["Age"].fillna(data_original["Age"].median(), inplace = True)

Ahora que no tenemos ningún valor nulo. Podemos ir preparando el modelo (dataset). En el caso de la columna “Sex”, tenemos como respuestas “male” y “female”. Y para verificarlo hacemos lo siguiente:

```
print (data_original['Sex'].unique())
```

```
['male' 'female']
```

Ahora convertiremos este campo en valores de 0 y 1. Donde 0 = Female y 1 = Male.

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
data_original["Sex"]=le.fit_transform(data_original["Sex"])

print (data_original['Sex'].unique())
```

```
['male' 'female']
[1 0]
```

En el caso de la columna “Name” podemos eliminarla. Ya que este campo no será necesario para determinar una clasificación o predicción.

```
data_original = data_original.drop(columns=['Name'])
data_original.info()
```

```
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  887 non-null    int64
 1   Pclass    887 non-null    int64
 2   Sex       887 non-null    int32
 3   Age       887 non-null    float64
 4   SibSp     887 non-null    int64
 5   Parent    887 non-null    int64
 6   Fare      887 non-null    float64
dtypes: float64(2), int32(1), int64(4)
memory usage: 45.2 KB
```

Ahora toca tendremos que discretar la columna de “Age”, ya que los rangos de edades son muy variadas, por lo tanto, haremos la siguiente discretación:

Rango actual	Valor discretado
0 a 12 años	1
12 a 25 años	2
25 a 45 años	3
> 45 años	4

Nota: Esta discretación es personal. Nosotros podemos hacer los ajustes del rango que más les convenga.

Definidos los valores a discretar, ahora lo aplicaremos en nuestro dataset.

```
def valor_discretar(x):
    if x <= 12:
        return 1
    elif x > 12  and x <= 25:
        return 2
    elif x > 25 and x <= 45:
        return 3
    elif x > 45:
        return 4

data_original['Age_discre'] = data_original['Age'].apply(valor_discretar)
```

Visualizamos nuestro DataFrame para ver si se aplicó lo anteriormente realizado.

Index	Survived	Pclass	Sex	Age	SibSp	Parent	Fare	Age_discre
0	0	3	1	22	1	0	7.25	2
1	1	1	0	38	1	0	71.2833	3
2	1	3	0	26	0	0	7.925	3
3	1	1	0	35	1	0	53.1	3
4	0	3	1	35	0	0	8.05	3
5	0	3	1	27	0	0	8.4583	3
6	0	1	1	54	0	0	51.8625	4
7	0	3	1	2	3	1	21.075	1
8	1	3	0	27	0	2	11.1333	3
9	1	2	0	14	1	0	30.0708	2

Ahora podremos borrar la columna de ‘Age’ o bien al seleccionar mis variables dependientes e independientes simplemente no la tomamos en cuenta.

En este caso lo haremos seleccionando una a una las columnas a utilizar.

```
X = data_original[['Age_discre','SibSp','Parent','Fare','Pclass','Sex']]
y = data_original['Survived']
```

Ahora empezaremos a crear el modelo para realizar el árbol de decisión. Separando los datos de entrenamiento y validación. Y verificamos la precisión del modelo.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

from sklearn.tree import DecisionTreeClassifier
arbol = DecisionTreeClassifier()
arbol.fit(X_train, y_train)

y_train_pred = arbol.predict(X_train)
y_test_pred = arbol.predict(X_test)

from sklearn.metrics import accuracy_score

print('Accuracy de Train:', accuracy_score(y_train_pred,y_train))
print('Accuracy de Test:', accuracy_score(y_test_pred,y_test))
```

La precisión quedo de esta manera.

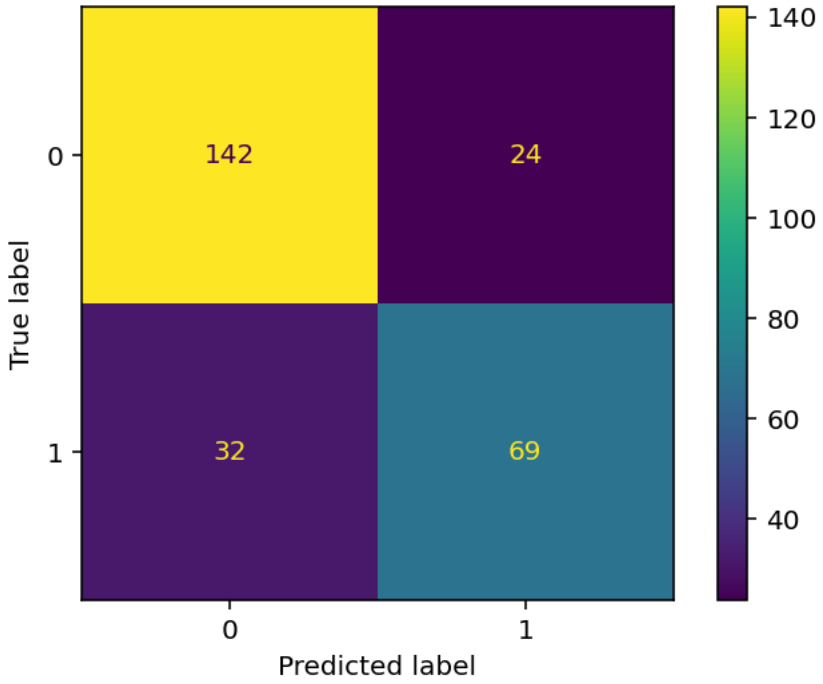
```
Accuracy de Train: 0.9467741935483871
Accuracy de Test: 0.7827715355805244
```

Ahora sacaremos una matriz de confusión, una **matriz de confusión** es una herramienta que permite visualizar el desempeño de un algoritmo de aprendizaje supervisado. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real., o sea en términos prácticos nos permite ver ***qué tipos de aciertos y errores está teniendo nuestro modelo a la hora de pasar por el proceso de aprendizaje con los datos***.

Realizaremos nuestra matriz de confusión para verificar la eficiencia y ver el comportamiento del modelo.

```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(
    arbol, X_test, y_test)

plt.show()
```



Podemos observar que en el modelo de los 166 supuestos positivos, 142 si lo fueron y 24 nos arroja como falso positivo. Y de los 101 supuestos negativos, 69 si fueron negativos y 32 fueron falsos negativos.

El modelo como esta es valido ya que tiene una efectividad del Test del 78% y un 94% del entrenamiento. Podemos ir ajustando los porcentajes de las variables de entrenamiento y test para ir subiendo la efectividad.