Importación de bibliotecas:

Importas las bibliotecas necesarias, incluyendo pandas para la manipulación de datos, DecisionTreeClassifier de scikit-learn para el modelo de árbol de decisión, train_test_split para dividir los datos en conjuntos de entrenamiento y prueba, y métricas de scikit-learn para evaluar el rendimiento del modelo.

```
# Load libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
```

Definición de nombres de columnas:

Defines los nombres de las columnas del conjunto de datos en col_names.

Carga de datos:

Cargas el conjunto de datos de diabetes desde un archivo CSV llamado "diabetes.csv" y lo almacenas en un DataFrame llamado pima. Los nombres de las columnas se asignan según col_names.

```
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv("diabetes.csv", header=None, names=col_names)
```

Preprocesamiento de datos:

Eliminas la primera fila de pima con pima = pima.iloc[1:, :] (esto parece ser para eliminar una fila de encabezado adicional que puede haber quedado después de cargar el archivo CSV).

```
pima.head()
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0

```
pima=pima.iloc[1:,:]
```

```
#split dataset in features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable
```

División de datos:

Defines las características (X) y la variable objetivo (y) y luego divides los datos en conjuntos de entrenamiento (X_train, y_train) y prueba (X_test, y_test) utilizando train_test_split. El 70% de los datos se utiliza para entrenar el modelo, mientras que el 30% se utiliza para evaluarlo.

```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30% test
```

Creación y entrenamiento del primer árbol de decisión:

Creas un objeto DecisionTreeClassifier llamado clf. Entrenas el modelo de árbol de decisión en el conjunto de entrenamiento (X_train, y_train) utilizando fit.

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

Predicciones y métricas del primer modelo:

Utilizas el modelo entrenado para hacer predicciones en el conjunto de prueba (X_test) y almacenas las predicciones en y_pred. Calculas métricas de evaluación, como la precisión, la recuperación y la puntuación F1, utilizando las funciones de metrics.

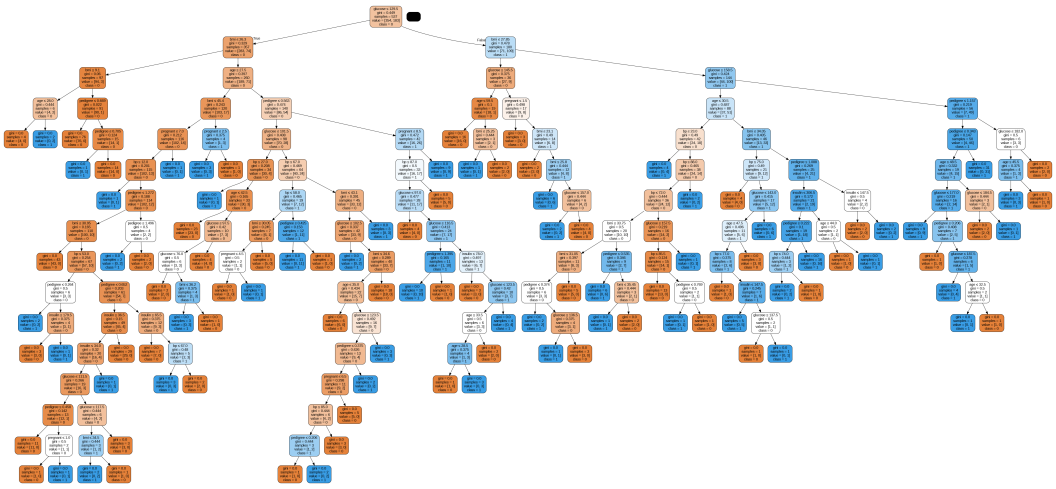
```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred,labels=['0','1'],pos_label='1'))
print("Recall:",metrics.recall_score(y_test, y_pred,labels=['0','1'],pos_label='1'))
print("F1:",metrics.f1_score(y_test, y_pred,labels=['0','1'],pos_label='1'))
```

```
Accuracy: 0.6796536796536796
Precision: 0.5714285714285714
Recall: 0.5176470588235295
F1: 0.5432098765432098
```

Visualización del primer árbol de decisión:

Exportas el árbol de decisión entrenado como un gráfico PNG utilizando export_graphviz y lo almacenas en un archivo llamado "diabetes.png". Luego, visualizas el gráfico utilizando pydotplus y Image

```
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```



Creación y entrenamiento del segundo árbol de decisión (con entropía):

Creas un segundo objeto DecisionTreeClassifier llamado clf con el criterio de "entropía" y una profundidad máxima de 3. Entrenas el nuevo modelo de árbol de decisión en el conjunto de entrenamiento (X_train, y_train) utilizando fit.

Creación y entrenamiento del segundo árbol de decisión (con entropía):

Creas un segundo objeto DecisionTreeClassifier llamado clf con el criterio de "entropía" y una profundidad máxima de 3. Entrenas el nuevo modelo de árbol de decisión en el conjunto de entrenamiento (X_train, y_train) utilizando fit.

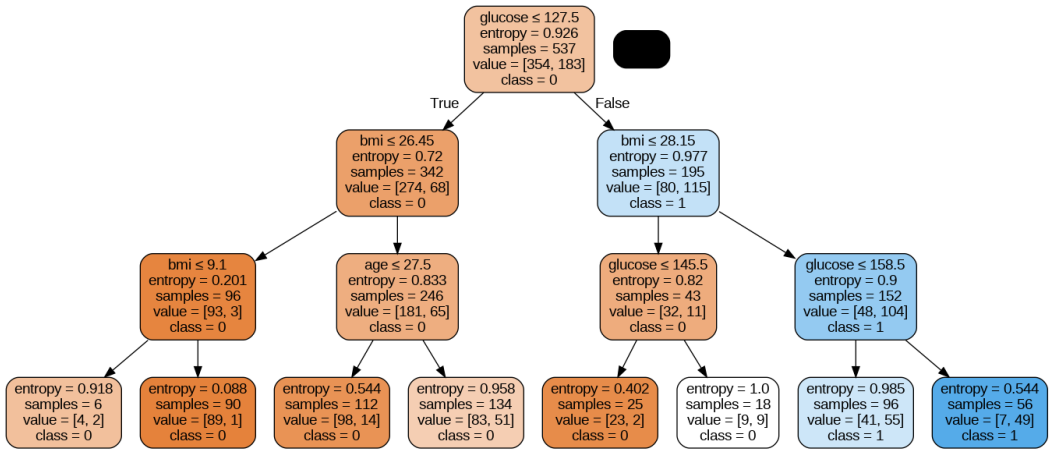
```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred,labels=['0','1'],pos_label='1'))
print("Recall:",metrics.recall_score(y_test, y_pred,labels=['0','1'],pos_label='1'))
print("F1:",metrics.f1_score(y_test, y_pred,labels=['0','1'],pos_label='1'))
```

```
Accuracy: 0.7705627705627706
Precision: 0.7105263157894737
Recall: 0.6352941176470588
F1: 0.6708074534161491
```

Visualización del segundo árbol de decisión (con entropía):

Exportas el segundo árbol de decisión entrenado como un gráfico PNG y lo visualizas de la misma manera que el primer árbol.

```
from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```



Este código carga el conjunto de datos de diabetes, entrena dos modelos de árbol de decisión con diferentes configuraciones (criterio de división y profundidad máxima) y evalúa su rendimiento. También visualiza ambos árboles de decisión para comprender cómo toman

decisiones.

✓ 0s completed at 12:59 PM