Python con Excel II Machine Learning, Analisis de una muestra del censo de la población americana

Introducción al Machine Learning con Python en Excel, uso de las principales librerías. Marco conceptual básico. Aplicación a la muestra de población del censo americano, sobre la relación de variables que condicionan el nivel ingresos de los individuos.

Jose Ignacio González Gómez
Departamento de Economía, Contabilidad y Finanzas - Universidad de La Laguna
www.jggomez.eu
Ejercicio adaptado de: jggomez

V. 2.5

Archivos fuentes:

Contenido

Cuestiones técnicas relacionadas I. Introducción a Marching Learning con Python

La biblioteca scikit-learn, módulo preprocessing y clases de codificadores

La librería Scikit-learn

Scikit-learn es una de las bibliotecas más populares de Python para aprendizaje automático (machine learning) que proporciona herramientas simples y eficientes para análisis predictivo de datos. Incluye algoritmos para:

- Clasificación
- Regresión
- Clustering
- Reducción de dimensionalidad
- Selección de modelos
- Preprocesamiento de datos

Módulo preprocessing

El módulo preprocessing se encarga de **preprocesar los datos** antes de alimentar un modelo, contiene funciones y clases para **transformar los datos**, es como la cocina donde se preparan los ingredientes antes de cocinar el modelo.

¿Por qué es importante? Porque los algoritmos de machine learning no entienden texto, categorías o escalas diferentes, preprocessing ayuda a:

- Normalizar
- Escalar datos numéricos (StandardScaler, MinMaxScaler)
- Codificar variables categóricas
- Imputación de valores faltantes
- Transformaciones polinómicas

Clases de codificadores de variables categóricas

Disponemos en este modulo de tres codificadores LabelEncoder, OneHotEncoder y OrdinalEncoder y para uso debemos cargar siempre la librería y el proceso, por ejemplo:

- from sklearn.preprocessing import **LabelEncoder**
- from sklearn.preprocessing import **OneHotEncoder**
- from sklearn.preprocessing import **OrdinalEncoder**

Reglas prácticas para la elección del codificador

Como primera aproximación presentamos a modo de orientas las consideraciones a tener en cuenta para la elección del codificador.

- Si tus datos tienen **categorías sin orden lógico** → usa **OneHotEncoder**.
- Si tus datos tienen **categorías con orden lógico** → usa **OrdinalEncoder**.
- Si estás codificando la **variable objetivo (y)** → usa **LabelEncoder**.
- Si estás usando modelos como **árboles de decisión o random forests**, que **no se ven afectados por el orden numérico**, puedes usar OrdinalEncoder incluso para variables nominales (aunque no es lo ideal).

Para modelos lineales o basados en distancia (como regresión logística, SVM, KNN), evita codificadores que introduzcan orden artificial.

Codificación de variables categóricas, de texto a numero

Sobre variables independientes (x) y dependientes (y)

En cualquier estudio o investigación la definición de las hipótesis es un proceso de gran importancia que supone identificar y clasificar las variables que intervienen. Y → X

Partiendo de las hipótesis (enunciado) debemos trasformar o esquematizarlo identificando las variables que intervienen asi como la relación entre ellas, las que son causa y las que son efecto.

Por tanto, podemos distinguir dos tipos de variables:

- Variable/s de entrada o predictor, tambien se le conoce como explicativa/s independiente/s, causa/s, input, estimulo/s (**x**). Viene representada por una X, son factores o atributos que se usan para predecir, explicar o influir en el valor de otra variable (la variable de respuesta o dependiente).
  Es la variable/s que se selecciona para determinar su relación con el fenómeno/s observados. En la relación más simple, un investigador estudia qué le sucedería a la variable efecto cuando cambia los valores de la variable causa o variable independiente.
- Variable dependiente o de respuesta, tambien se le conoce como de efecto, salida, output. (**y**). Viene representada por una Y. La variable dependiente es el factor que se observa o mide para determinar el efecto de la variable independiente o variable causa. A la variable dependiente se le considera así porque sus valores van a depender de los valores de la variable independiente. Representa la consecuencia de los cambios en el sujeto bajo estudio o en la situación que se está estudiando

Necesidad de transformación de las variables

Como hemos señalado la mayoría de los algoritmos de machine learning requieren que las variables de entrada sean numéricas, por tanto, es necesario llevar a cabo esta conversión, pasar las etiquetas (labels) con valores categóricos (texto, categorías) a números.

Por otro lado también es necesario considerar que la variable objetivo (y) sin es categórica debe convertirse a numérica y para ello se debe usar la clase LabelEncoder.

Asi como primer aproximación destacar que antes de entrenar el modelo debemos transformar las columnas categóricas (de texto, categorías,,) a números antes de entrenar un modelo, asi:

- Codificar la **variable objetivo (y).** Si la variable objetivo (y) es categóricas debe transformarse en numérica, asignándole un numero entero a cada categoría, donde el orden no importa, solo se codifica la salida, introduce un orden artifical. En este caso se debe usar la clase de codificación LabelEncoder.
- Codificar las **variables de salida** (x). Para las variables de salida (x) que sean categóricas igualmente requieren esta conversión, pasar las etiquetas (labels) con valores categóricos (texto, categorías) a números y según el caso la clase de codificación OrdinalEncoder o OneHotEncoder.

Variables categóricas no ordinales (nominales) vs ordinales

Las **variables categóricas no ordinales** (también llamadas *nominales*) y las **variables ordinales** son dos tipos de variables categóricas, pero se diferencian en si existe o no un **orden natural** entre sus categorías.

Característica	Variables nominales	Variables ordinales
Definición	Clasifican en categorías sin jerarquía.	Clasifican en categorías con un orden lógico.
Orden	No existe orden entre categorías.	Sí existe un orden o jerarquía.
Ejemplos	Color de ojos (azul, verde, marrón), tipo de mascota (perro, gato, pez).	Nivel educativo (primaria, secundaria, universidad), grado de satisfacción (bajo, medio, alto).

Operaciones posibles	Solo se puede contar la frecuencia de cada categoría.	Además de contar, se pueden comparar posiciones relativas (mayor/menor).
Codificación numérica	Los números asignados son solo etiquetas sin significado cuantitativo.	Los números asignados reflejan el orden, pero no la magnitud exacta de la diferencia.

Ejemplos

Contexto	Variable nominal	Variable ordinal
Encuesta de satisfacción	Tipo de producto comprado (ropa, electrónica, comida)	Nivel de satisfacción (muy insatisfecho, insatisfecho, neutral, satisfecho, muy satisfecho)
Educación	Carrera universitaria (Medicina, Derecho, Ingeniería)	Nivel educativo (primaria, secundaria, bachillerato, licenciatura, posgrado)
Salud	Grupo sanguíneo (A, B, AB, O)	Gravedad del dolor (leve, moderado, intenso)
Marketing	Marca preferida (Nike, Adidas, Puma)	Frecuencia de compra (nunca, rara vez, a veces, frecuentemente)
Deportes	Deporte practicado (fútbol, baloncesto, natación)	Posición en un torneo (1º, 2º, 3º...)
Gastronomía	Tipo de cocina favorita (italiana, japonesa, mexicana)	Picante de un plato (suave, medio, fuerte)

Importancia de diferenciarlas

Distinguir entre **variables nominales** y **ordinales** es clave porque influye directamente en **cómo analizas los datos y qué conclusiones puedes sacar**. Si las tratas igual, puedes cometer errores estadísticos o interpretar mal los resultados.

🎯 *Razones por las que es importante diferenciarlas*

1. Elección del análisis estadístico
   - Las **nominales** requieren pruebas que no asumen orden (ej. Chi-cuadrado, tablas de contingencia).
   - Las **ordinales** permiten análisis que consideran jerarquía (ej. correlación de Spearman, pruebas de rangos).
2. Codificación correcta de datos
   - En nominales, los números son solo etiquetas (1 = perro, 2 = gato).
   - En ordinales, los números reflejan un orden (1 = bajo, 2 = medio, 3 = alto).
3. Interpretación de resultados
   - Si tratas una nominal como ordinal, podrías inventar un orden que no existe.
   - Si tratas una ordinal como nominal, pierdes información valiosa sobre la jerarquía.
4. Visualización adecuada
   - Nominal: gráficos de barras sin orden específico.
   - Ordinal: gráficos que respeten la secuencia lógica (de menor a mayor, por ejemplo).
5. Modelos predictivos
   - En machine learning, el tipo de variable determina si se usa *one-hot encoding* (nominal) o *ordinal encoding* (ordinal) para que el modelo no aprenda relaciones falsas.

💡 En resumen: Saber si una variable es nominal u ordinal no es un detalle técnico menor; es la base para elegir el método estadístico correcto, evitar sesgos y comunicar resultados de forma precisa.

Truco rápido para identificarlas:

- Si **puedes ordenar** las categorías de forma lógica → **ordinal**.
- Si **no hay un orden natural** y solo son etiquetas → **nominal**.

Cómo usarlos en análisis

- Si analizas **nominales**, solo puedes contar cuántas veces aparece cada categoría.
- Si analizas **ordinales**, además puedes ver tendencias en función del orden (por ejemplo, si la mayoría de clientes están "muy satisfechos" o "satisfechos").

Aspectos técnicos. La clase LabelEconceder.

Presentacion

En Python, contamos con la biblioteca **scikit-learn**, una de las más utilizadas para tareas de *machine learning*. Dentro de su módulo **preprocessing**, se encuentra la clase **LabelEncoder**, que nos permite transformar etiquetas categóricas en valores numéricos.

Por tanto, la clase LabelEncoder de la librería **scikit-learn**, que se usa para **convertir etiquetas categóricas en valores numéricos**.

📌 ¿Qué hace exactamente?

- **Toma etiquetas (y)** — por ejemplo, nombres de ciudades, colores, tipos de producto.
- **Asigna un número entero único a cada categoría**.
- **Transforma** esas etiquetas a números para que el modelo pueda procesarlas.
- También permite **hacer el proceso inverso**: volver de números a las etiquetas originales.

Ejemplo

```
# Importamos la clase
   from sklearn.preprocessing import LabelEncoder

# Creamos el codificador
   le = LabelEncoder()

# Lista de etiquetas
   ciudades = ["París", "París", "Tokio", "Ámsterdam"]

# Ajusta y transforma
   numeros = le.fit_transform(ciudades)
   print(le.classes_)       # ['Ámsterdam' 'París' 'Tokio']
   print(numeros)           # [1 1 2 0]

# Volver a texto
   original = le.inverse_transform(numeros)
   print(original)          # ['París' 'París' 'Tokio' 'Ámsterdam']
```

Cosas importantes a tener en cuenta

- LabelEncoder **está pensado para codificar la variable objetivo (y)**, no las columnas de características (X). Para variables categóricas en X, se recomienda usar OrdinalEncoder o OneHotEncoder.
- El mapeo es **ordinal** (0, 1, 2…), por lo que **introduce un orden artificial** que no siempre es deseable.

- Si trabajas con datos en Excel y Python, puedes usarlo para transformar columnas categóricas antes de entrenar un modelo.

Limitaciones de LabelEncoder

Como hemos destacado el LabelEncoder de *scikit-learn* es muy útil para convertir etiquetas categóricas en números, pero tiene varias **limitaciones** que conviene conocer antes de usarlo, principalmente:

1. **Orden artificial**
   - Asigna números enteros consecutivos (0, 1, 2…) a las categorías.
   - Esto **induce un orden que no existe** en datos nominales (por ejemplo, "rojo" < "verde" < "azul"), lo que puede confundir a modelos que interpretan relaciones numéricas.
2. **No apto para variables de entrada (X)**
   - Está diseñado para codificar **la variable objetivo (y)**, no las características de entrada.
   - Si lo usas en columnas de entrada, algunos algoritmos pueden interpretar erróneamente que hay una relación ordinal.
3. **Problemas con algoritmos basados en distancias**
   - Modelos como *k-NN*, *SVM* o regresión lineal pueden interpretar que la diferencia entre códigos numéricos tiene un significado real, distorsionando el resultado.
4. **Sensibilidad a categorías nuevas**
   - Si en predicción aparece una categoría que no estaba en el entrenamiento, el LabelEncoder no sabrá cómo codificarla y lanzará un error.
5. **No soporta multietiqueta o multiclase compleja**
   - No puede representar casos donde una observación pertenezca a varias categorías simultáneamente
6. **Escalabilidad limitada**
   - Con muchas categorías distintas, los números asignados pueden ser grandes y poco manejables para ciertos modelos.

Alternativas según el caso, tabla comparativa LabelEncoder, OneHotEncoder y OrdinalEncoder

- OneHotEncoder → Para variables categóricas sin orden, crea columnas binarias por categoría.
- OrdinalEncoder → Para variables categóricas con orden natural (por ejemplo, "bajo" < "medio" < "alto").
- Codificadores especializados (*Target Encoding*, *Frequency Encoding*, etc.) → Para datos con alta cardinalidad.

Presentamos la siguiente tabla comparativa entre **LabelEncoder, OneHotEncoder y OrdinalEncoder**, con ejemplos y cuándo usar cada uno, para que lo tengas como referencia rápida en tus proyectos de Python y Excel

Codificador	¿Qué hace?	Ejemplo de entrada	Ejemplo de salida	Cuándo usarlo	Ventajas	Limitaciones

LabelEncoder	Asigna un número entero único a cada categoría.	["rojo", "azul", "verde"]	[2, 0, 1]	- Codificar la variable objetivo (y)- Casos donde el orden no importa pero solo se codifica la salida.	- Simple y rápido. - No aumenta la dimensionalidad.	Introduce un orden artificial.- No recomendado para variables de entrada (X) nominales.
OneHotEncoder	Crea una columna binaria por cada categoría.	["rojo", "azul", "verde"]	[[1,0,0],[0,1,0],[0,0,1]]	- Variables categóricas sin orden. - Modelos que no asumen relación ordinal (Regresión logística, Redes neuronales, KNN).	- Evita orden artificial. - Fácil de interpretar.	- Aumenta la dimensionalidad. - Puede ser costoso con muchas categorías.
OrdinalEncoder	Asigna números enteros respetando un orden definido.	["bajo", "medio", "alto"]	[0, 1, 2]	- Variables categóricas con orden natural. - Ej.: niveles de satisfacción, tallas de ropa.	- Mantiene el orden real. - Compacto en memoria.	- Si el orden no es real, introduce sesgo. - No apto para datos nominales.

Regla de oro

- **Nominal (sin orden)** → OneHotEncoder
- **Ordinal (con orden)** → OrdinalEncoder
- **Variable objetivo (y)** → LabelEncoder

Guía de elección del codificador adecuado

Elegir entre **LabelEncoder**, **OneHotEncoder** y **OrdinalEncoder** depende del **tipo de variable categórica** que tienes y del **modelo de machine learning** que vas a usar. Aquí te dejo una guía clara para ayudarte a decidir:

Codificador	¿Para qué tipo de variable?	¿Cuándo usarlo?	¿Ventajas?	¿Precauciones?
LabelEncoder	Variable **dependiente** (target)	Cuando la variable objetivo es categórica (clasificación).	Simple y directo.	No usar en variables independientes (features), ya que introduce orden falso.
OneHotEncoder	Variable **independiente nominal**	Cuando las categorías **no tienen orden** (ej. color, ciudad, país).	No introduce orden, ideal para modelos lineales.	Aumenta la dimensionalidad si hay muchas categorías.
OrdinalEncoder	Variable **independiente ordinal**	Cuando las categorías **tienen un orden lógico** (ej. bajo < medio < alto).	Preserva el orden, útil para modelos que lo aprovechan.	

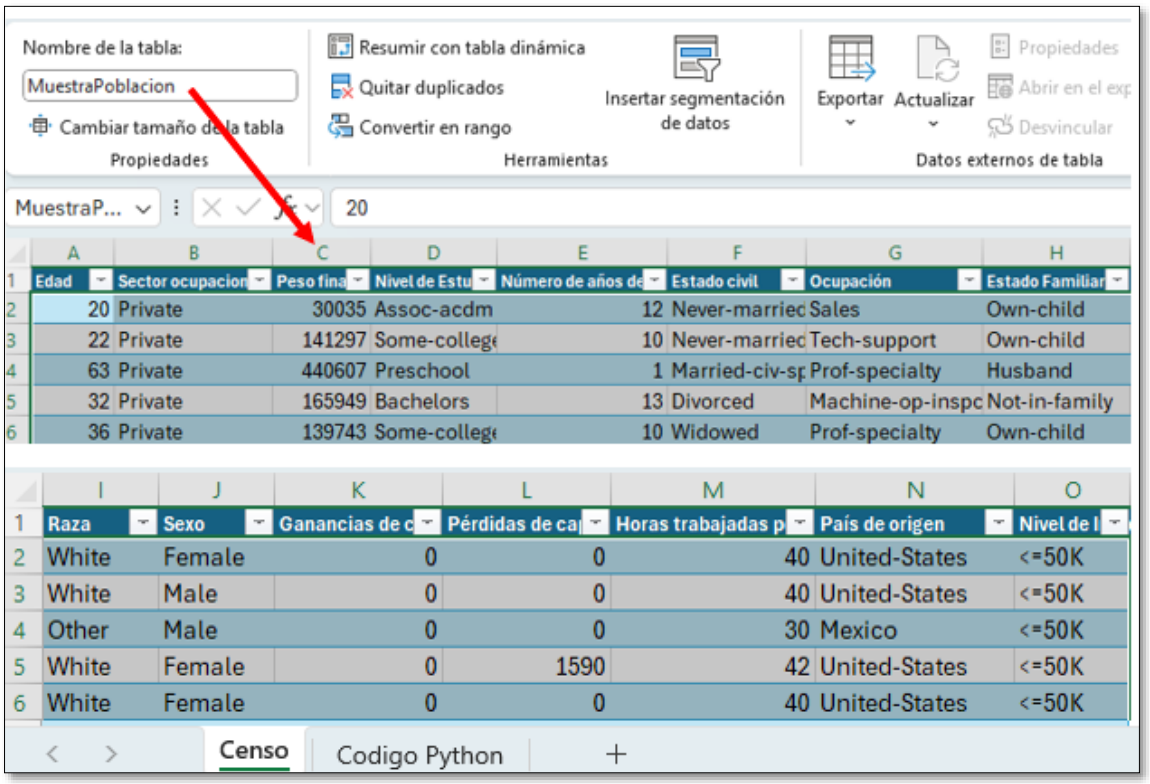
Caso Machine Learning I. Análisis del Censo (David Langer)

Fuente: David Langer, Machine Learning [Machine Learning with Microsoft Excel? Yes, PLEASE!]

Presentacion

Vamos a trabajar con el libro de Excel MachineLearning que contiene una muestra de registros (15.016 personas censadas) provenientes de la Oficina del Censo de Estados Unidos, en la tabla MuestraPoblacion.

Cada fila es un ciudadano estadounidense que proporcionó los datos a la Oficina del Censo las columnas son diversas características de estos ciudadanos estadounidenses individuales,



	A	B	C	D	E	F	G	H
1	Edad	Sector ocupacion	Peso fina	Nivel de Estu	Número de años de	Estado civil	Ocupación	Estado Familiar
2	20	Private	30035	Assoc-acdm	12	Never-marriec	Sales	Own-child
3	22	Private	141297	Some-college	10	Never-marriec	Tech-support	Own-child
4	63	Private	440607	Preschool	1	Married-civ-sp	Prof-specialty	Husband
5	32	Private	165949	Bachelors	13	Divorced	Machine-op-inspc	Not-in-family
6	36	Private	139743	Some-college	10	Widowed	Prof-specialty	Own-child

	I	J	K	L	M	N	O
1	Raza	Sexo	Ganancias de c	Pérdidas de ca	Horas trabajadas p	País de origen	Nivel de I
2	White	Female	0	0	40	United-States	<=50K
3	White	Male	0	0	40	United-States	<=50K
4	Other	Male	0	0	30	Mexico	<=50K
5	White	Female	0	1590	42	United-States	<=50K
6	White	Female	0	0	40	United-States	<=50K

Ilustración 1

Disponemos igualmente de una hoja que hemos denominado “Codigo Python” cuyo objetivo es disponer del codigo desarrollado relacionado con el analisis.

En resumen, las variables disponibles son:

Edad	Raza
Sector ocupacional	Sexo
Peso final	Ganancias de capital
Nivel de Estudios	Pérdidas de capital
Número de años de educación	Horas trabajadas por semana
Estado civil	País de origen
Ocupación	Nivel de Ingreso/año
Estado Familiar	

Tabla 1

Objetivo. Aplicación de Machine Learning

De los valores disponibles, destacar la variable o característica “**Nivel de Ingresos/Año**” que categoriza a los ciudadanos según el nivel de renta anual en dos categorías, altos ingresos y bajos

ingresos que serán aquellos inferiores a <= 50K (ingresos anuales menor o igual a 50.000 $) o altos ingresos cuando superan esa barrera > 50K anuales.

Se intenta analizar si existe alguna relación predictiva entre el nivel de ingresos anuales y el resto de variables disponibles y para ello aplicaremos Machine Learning (aprendizaje automático) con Python en Excel con el fin de encontrar algún patrón en los datos, algún tipo de asociación.

Técnicamente y de forma simplificada Machine Learning es una rama de la inteligencia artificial que permite a las computadoras aprender por sí solas y su funcionamiento consiste en:

1. Se le da información (datos) a un modelo.
2. El modelo encuentra patrones en esos datos.
3. Aprende a hacer predicciones o tomar decisiones basadas en esos patrones.
4. Cuanto más datos tiene, mejor aprende.

Por tanto y para nuestro caso, queremos determinar que variables o columnas influyen mas en el nivel de ingresos y visualizar estas relaciones o cómo cambia el ingreso según el nivel de estudios, edad, etc..

Es decir, queremos conocer que columnas o factores afectan al nivel de ingresos bajos o ingresos altos y poder encontrar algún tipo de patrón o relación predictiva precisa entre todas estas otras columnas y este resultado concreto de interés.

Preparando el análisis con Python para Excel

Introducción, definiendo el Dataframe

En primer lugar comenzamos creando en una hoja de calculo nueva dentro del libro de trabajo en la que vamos a llevar el analisis con Python para Excel, comenzando por crear cada codigo o scrip en una fila, asi definimos el Dataframe que llamaremos como df al ser el nombre más corriente.

Ilustración 2

Para ello simplemente en la celda B3 seleccionamos la tabla origen de los datos y creamos el nombre a la variable como df.

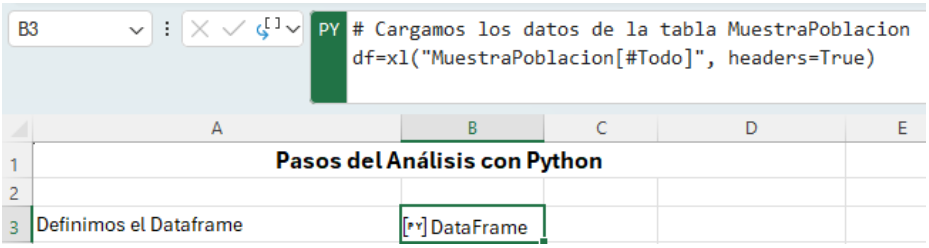


Ilustración 3

Explorando el Dataframe, función info()

Información general del DataFrame. Proporciona información sobre la tabla, muestra el número de entradas, nombres de columnas, tipo de datos, y cuántos valores nulos en el Dataframe.

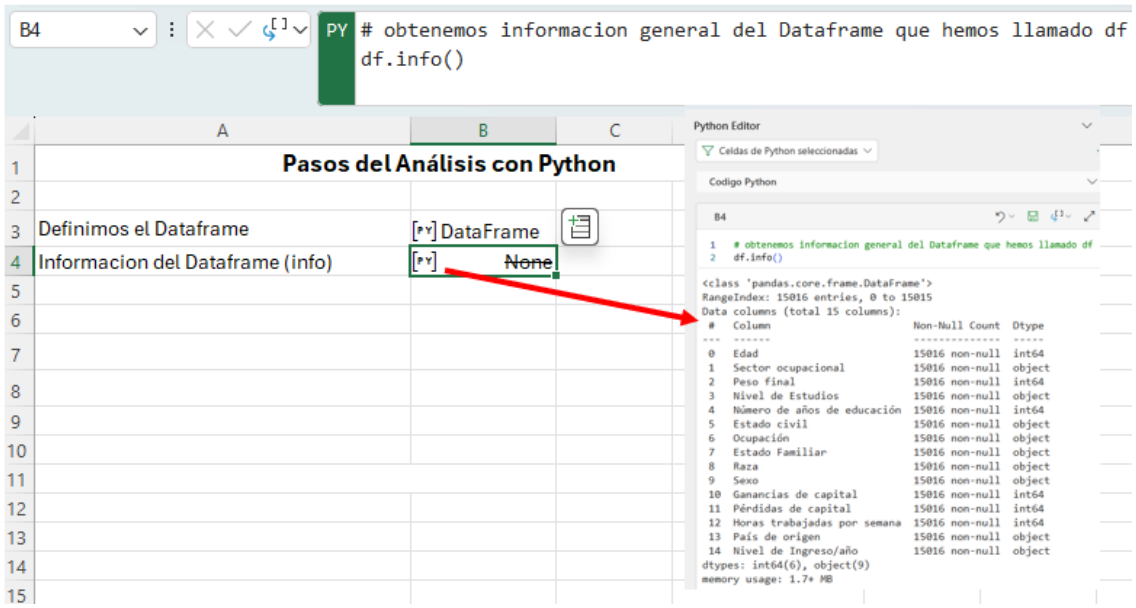


Ilustración 4

El resultado de esta función se muestra en el editor de Python.

Explorando el Dataframe. Estadisticas básicas, función describe()

Estadísticas descriptivas. Devuelve estadísticas como media, desviación estándar, mínimo, máximo, etc., para columnas numéricas. El resultado se muestra en el editor de Python.

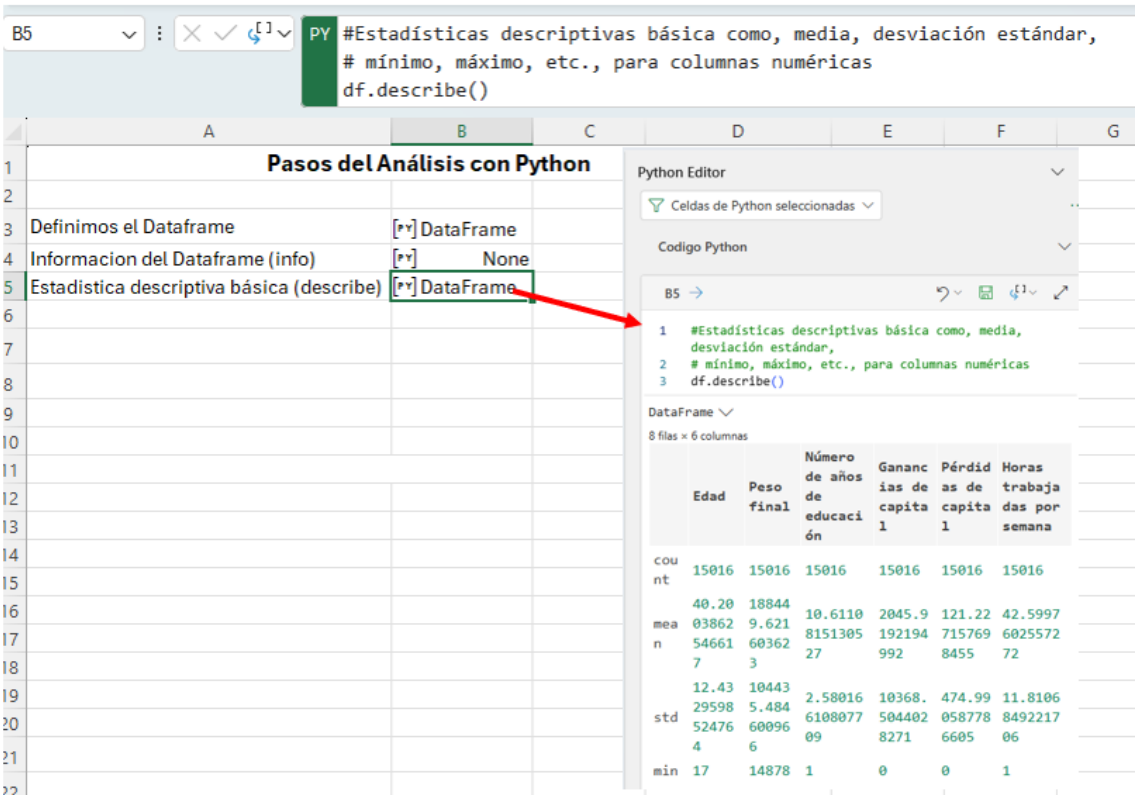


Ilustración 5

Selección y transformación de las variables independientes (x) y dependiente (y)

Introducción

Recordamos que la mayoría de los algoritmos de machine learning requieren que las variables de entrada (x) y de salida (y) sean numéricas, por tanto, en algunos casos es necesario llevar a cabo esta conversión, pasar las variables categóricas (con texto o etiquetas) a número lo que implica, como se ha señalado en el marco conceptual donde se dispone de tres clases para esta transformación LabelEncoder, OneHotEncoder y OrdinalEncoder y dependiendo del tipo de variable categórica de la que disponemos y del modelo de machine learning que a aplicar se recomienda la clase a emplear. En general

- Transformar o codificar la **variable objetivo (y).** Si la variable objetivo (y) es categóricas debe transformarse en numérica, asignándole un numero entero a cada categoría, donde el orden no importa, solo se codifica la salida, introduce un orden artificial. En este caso se debe usar la clase de codificación LabelEncoder.
- Transformar o codificar las **variables de salida** (x). Para las variables de salida (x) que sean categóricas igualmente requieren esta conversión, pasar las etiquetas (labels) con valores categóricos (texto, categorías) a números y según el caso la clase de codificación OrdinalEncoder o OneHotEncoder.

Pasamos a aplicarlo al caso de estudio.

Selección y transformación de los predictores (x), creacion de la lista en Python

A continuación, vamos a definir los predictores, es decir que variables (columnas) son las explicativas, representada por una X, son los factores o atributos que se usan para predecir, explicar o influir en el valor de otra variable Y (la variable de respuesta o dependiente). De esta selección construiremos una lista de Python aunque previamente debemos garantizar que, como hemos comentado, el carácter numérico de sus valores.

Tal y como se muestra en la Tabla 1(variables en azul) seleccionaremos las características: edad, número de años de educación, ocupación y horas trabajadas por semana y estas características seleccionadas entendemos que quizás podría predecir el nivel de ingresos.

Todas las columnas seleccionadas tiene formato numérico, excepto la columna Ocupación que contiene valores categóricos relacionados con las diversas ocupaciones, 14 valores o categorías de ocupación. Por tanto esta columna o variable es necesaria transformarla para pasarla de categórica a numérica, obteniendo asi una vez realizada la transformación en 17 columnas, 3 correspondientes a la edad, número de años de educación, y horas trabajadas que no han necesitado transformación y las otras 14 a la columna transformada ocupación.

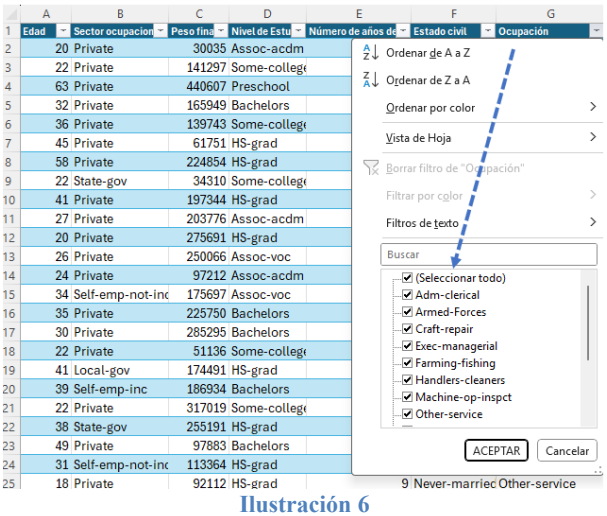


Ilustración 6

Comenzamos por tanto el trabajo en nuestro libro de trabajo Excel, tal y como se muestra en la Ilustración 7creamos una variable que llamamos Predictores1 para seleccionar las 4 columnas (características) de nuestro Dataframe.

Esto implica que consideramos que las características Edad, Nº de años de educación, la ocupación asi como horas trabajadas por semana influyen en el nivel de ingresos anuales. Es decir

la hipótesis que se plantean es que las personas mayores, con alto nivel de educación, que trabajan en cierta ocupación y que trabajan 40 horas semanales podría estar relacionado con altos ingresos.

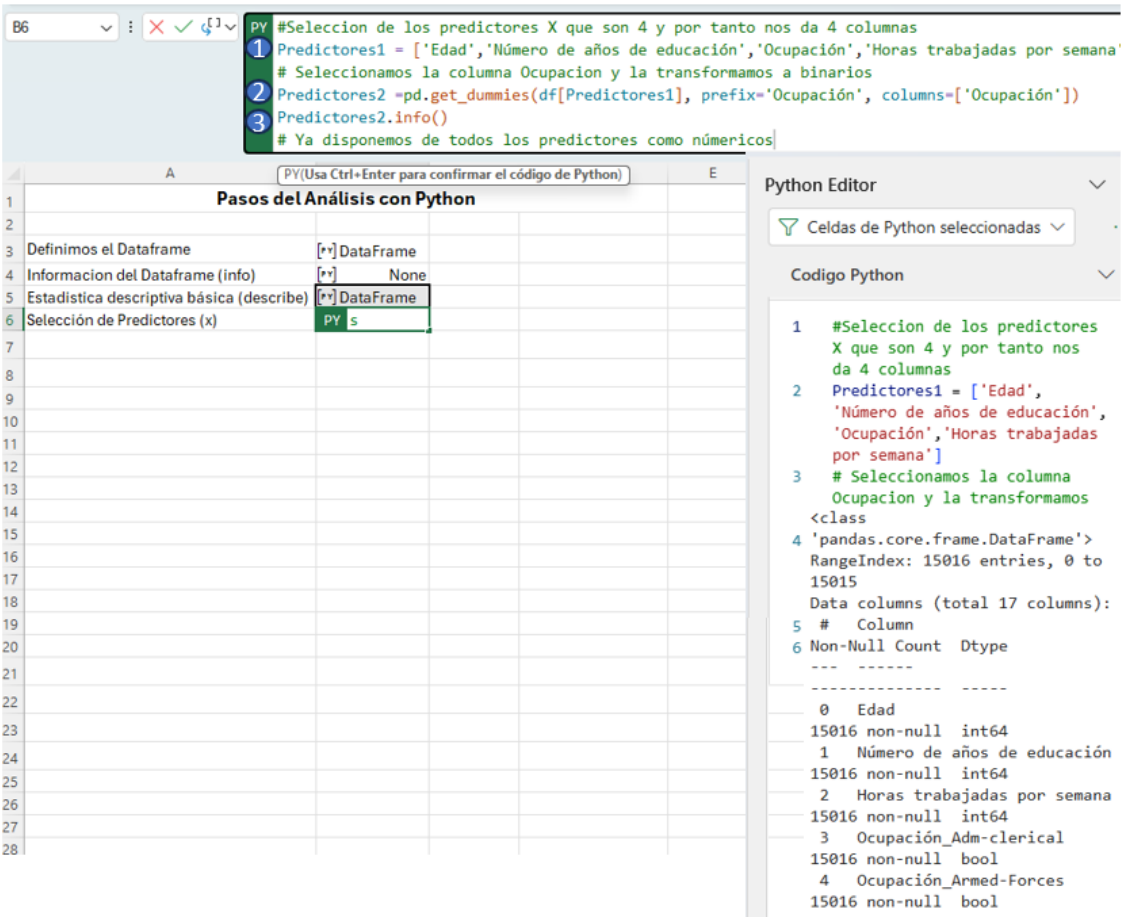


Ilustración 7

Pero como hemos señalado es necesario transformar la variable "Ocupación" en categoría y para ello definimos la variable Predictores2 que recupera la lista creada anteriormente (Predictores1), y transforma la columna "Ocupación" de categórica a binarias (0 o 1), generando asi 14 columnas nuevas que comienza con prefijo Ocupación_Categoria tal y como se muestra en la Ilustración 7. Finalmente disponemos de la lista seleccionada de predictores formada por 17 columnas o características (14+3)

Del codigo de la Ilustración 7 destacamos lo siguiente:

1. Crea una lista que llamamos Predictores1

```
Predictores1 = ['Edad','Número de años de educación','Ocupación','Horas trabajadas por semana']
```

En este caso selecciona del Dataframe (df) las columnas señaladas y crear la lista de Python.

2. Modificar la lista crea una nueva que llamamos Predictores2 seleccionado todos los de la lista anterior (Predictores1) y transforma los valores de "Ocupación" que son valores categóricos nominales en binarios (0 o 1), con la funcion pd.get_dummies

```
Predictores2 =pd.get_dummies(df[Predictores1], prefix='Ocupación', columns=['Ocupación'])
```

La función pd.get_dummies() de pandas permite convertir variables categóricas en variables dummy (también conocidas como variables indicadoras o variables ficticias). Es una técnica común en machine learning y modelado estadístico para trabajar con datos categóricos.

Desglose paso a paso:

a) df[Predictores1]:

- Aquí se está seleccionando un subconjunto del DataFrame df usando una lista de nombres de columnas almacenada en Predictores1.
- Es decir, Predictores1 probablemente sea algo como ['Ocupación', 'Edad', 'Salario'], y esta parte selecciona esas columnas.

b) columns=['Ocupación']:

- Indica que solo se deben convertir en variables dummy las categorías de la columna 'Ocupación'.

c) prefix='Ocupación':

- Especifica el prefijo que se usará para nombrar las nuevas columnas dummy. Por ejemplo, si 'Ocupación' tiene valores como 'Ingeniero', 'Doctor', 'Profesor', las nuevas columnas serán:
  - Ocupación_Ingeniero
  - Ocupación_Doctor
  - Ocupación_Profesor

d) pd.get_dummies(...):

- Esta función crea una nueva columna para cada categoría única en 'Ocupación', con valores 0 o 1 indicando si esa fila pertenece a esa categoría.

e) Predictores2 = ...:

- El resultado se guarda en una nueva variable llamada Predictores2, que ahora contiene las columnas originales más las columnas dummy generadas derivada de ocupacion.

Selección y transformación de la variable objetivo (y)

Si analizamos la variable dependiente o de efecto "Nivel de Ingresos/año" vemos que esta es categórica en concreto dos, <=50k y >50k y es necesario transformarla a numérica o binarias (0 o 1), como hemos comentado en el apartado técnico.

En la librería de Python **scikit-learn** (una de las más usadas en Python para machine learning) y en el módulo **preprocessing** disponemos de la clase **LabelEncoder** para transformar la variable objetivo de categórica a numérica, por tanto vamos a aplicar esta transformación.

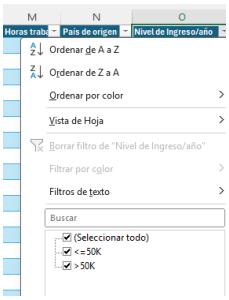


Ilustración 8

¿Por qué es útil esto en Machine Learning?

Los algoritmos de ML como regresión logística, árboles de decisión, etc., **no pueden trabajar directamente con texto**. Necesitan que las variables categóricas estén en formato numérico. LabelEncoder es una forma rápida de hacerlo cuando las categorías **no tienen un orden específico** (aunque si sí lo tienen, podrías considerar OrdinalEncoder).

Comenzamos por comprobar si cuando cargamos el Excel disponemos de la librería necesaria scikit-learn, es decir si ha sido importada en los parametros por defecto de Python y como se muestra en Ilustración 9 esta no esta por defecto activada, por lo tanto debemos comenzar por importarla para trabajar con el módulo preproccesing y la clase LabelEncoder.

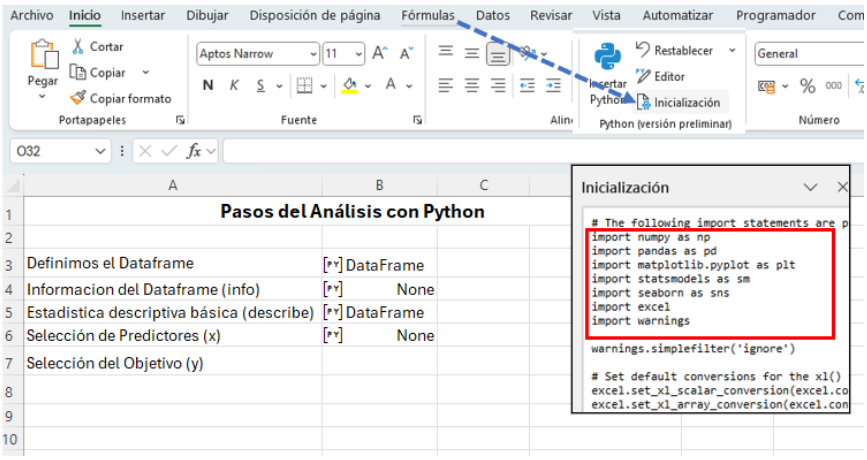


Ilustración 9

El proceso de selección y transformación que llevamos a cabo sobre la variable objetivo lo tenemos en la celda B7 tal y como se muestra en la Ilustración 10 y que pasamos a comentar a continuación.

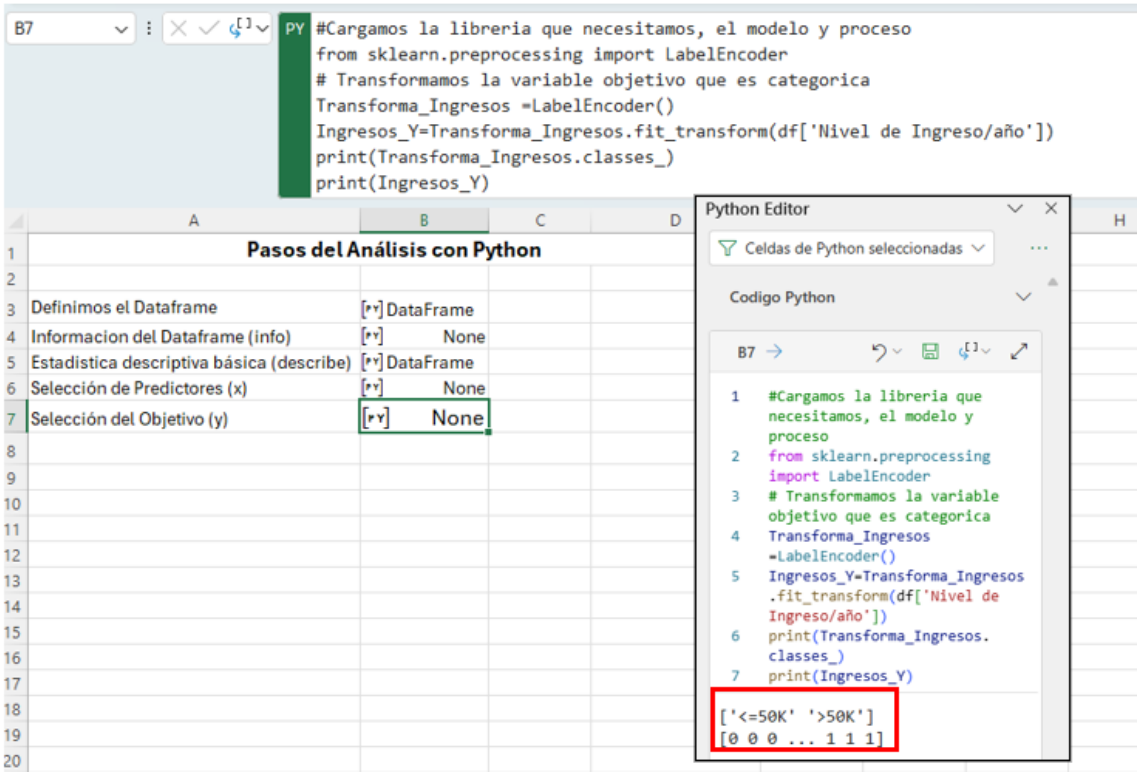


Ilustración 10

El código desarrollado utiliza la librería **scikit-learn** para transformar una columna con valores categóricos a valores numéricos. Esto es un paso común en el **preprocesamiento de datos** para algoritmos de **Machine Learning**, ya que la mayoría de ellos requieren datos numéricos para funcionar correctamente.

1 from sklearn.preprocessing import LabelEncoder

Comenzamos cargando la librería que necesitamos, el modelo y proceso. Importa la clase LabelEncoder de la biblioteca sklearn.preprocessing, que se usa para convertir etiquetas categóricas (strings) en números. Es útil cuando los modelos de Marchine Learning (ML) requieren datos numéricos. Esta clase es la herramienta principal que se usará para codificar las etiquetas categóricas en valores numéricos.

2. Transforma_Ingresos = LabelEncoder()

Aquí se crea una instancia de la clase LabelEncoder y se le asigna el nombre Transforma_Ingresos. Este objeto es el que contendrá toda la lógica y el "aprendizaje" necesario para realizar la transformación. Es como si estuvieras creando un "codificador" que se especializará en entender los niveles de ingreso.

Este objeto se usará para transformar los valores categóricos de la columna 'Nivel de Ingreso/año'.

3. Ingresos_Y = Transforma_Ingresos.fit_transform(df['Nivel de Ingreso/año'])

Esta es la línea más importante, aquí ocurre la transformación. Se realiza una doble operación en un solo paso:

- fit_transform() hace dos cosas:
    - **fit**: Aprende todas las categorías únicas en la columna 'Nivel de Ingreso/año'. El codificador Transforma_Ingresos "aprende" los valores únicos que existen en la columna 'Nivel de Ingreso/año' de tu DataFrame df. Por ejemplo, si los valores son "Bajo", "Medio" y "Alto", el codificador los identifica y les asigna un número a cada uno.
    - **transform**: Asigna un número entero a cada categoría y transforma la columna en un array de números. Una vez que ha "aprendido" la correspondencia, el codificador transforma la columna original, reemplazando cada valor categórico por su número asignado. El resultado de esta transformación se guarda en una nueva variable llamada Ingresos_Y

Por ejemplo, si la columna tiene:
['Bajo', 'Medio', 'Alto', 'Medio', 'Bajo']
El encoder podría asignar:
'Alto' → 0, 'Bajo' → 1, 'Medio' → 2
Y el resultado sería:
[1, 2, 0, 2, 1]

Las Línea 4 y 5: Verificación de la Transformación, estas líneas son para verificar el resultado de la codificación

4. print(Transforma_Ingresos.classes_)

Muestra las clases originales (categorías) que fueron codificadas, es decir Muestra los valores categóricos que LabelEncoder ha identificado y su orden de asignación. Esto es útil para saber qué número corresponde a qué categoría. Por ejemplo, podría imprimir ['Alto', 'Bajo', 'Medio'].

5. print(Ingresos_Y)

Imprime el array transformado, es decir, los valores numéricos que representan cada categoría en la columna 'Nivel de Ingreso/año', es decir muestra el nuevo array (lista de valores) que contiene la columna 'Nivel de Ingreso/año' pero ahora con los valores numéricos correspondientes. Si la columna original tenía los valores ['Alto', 'Medio', 'Bajo'], esta línea podría imprimir [0, 2, 1], dependiendo del orden en que el codificador asignó los números

En resumen, este código prepara tu columna objetivo categórica 'Nivel de Ingreso/año' para que pueda ser utilizada en un modelo de Machine Learning, convirtiendo texto en números que el modelo pueda entender y procesar.

Creacion y entrenamiento del modelo, con Árbol de Decisión

Concepto teóricos

Alcanzado este punto estamos en condiciones de entrenar el modelo, aplicando un tipo de técnica especifica llamada “Árbol de Decisión” que es una técnica (algoritmo) de **aprendizaje supervisado** que se usa para **clasificación** y **regresión**. Funciona como un diagrama de flujo jerárquico:

- **Nodo raíz** → el punto de inicio, donde se evalúa la primera característica de los datos.
- **Nodos de decisión** → cada uno plantea una pregunta o condición sobre una variable (por ejemplo: *¿La temperatura es mayor a 30 °C?*).
- **Ramas** → representan las posibles respuestas a esa pregunta.
- **Hojas** → el resultado final o predicción (por ejemplo: *Sí, va a llover* o *No, no va a llover*).

🔍 **Cómo aprende**: El algoritmo divide los datos en subconjuntos cada vez más homogéneos siguiendo una estrategia de *divide y vencerás*, eligiendo en cada paso la característica que mejor separa las clases o predice el valor.

✂️ **Poda** (*pruning*): Se eliminan ramas poco útiles para evitar que el modelo se vuelva demasiado complejo y se sobreajuste a los datos de entrenamiento.

💡 **Ventajas**:

- Fácil de interpretar y visualizar.
- Maneja datos numéricos y categóricos.
- No requiere suposiciones sobre la distribución de los datos.

Aplicando a nuestro caso tenemos tres apartados a considerar.

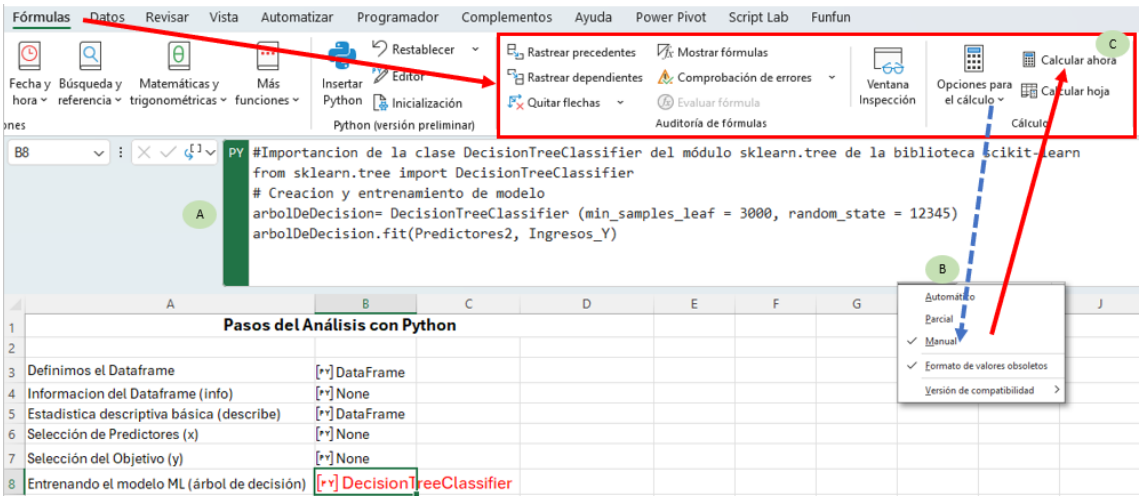


Ilustración 11

A. La parte del codigo donde creamos y entrenamos al modelo de machine learning a traves de la técnica de árbol de decisión y que pasaremos a explicar.
B. En este apartado es necesario tener en consideración el modo manual, es decir pasar de automático a manual, la razón fundamental es que todas estas celdas se ejecutan una y otra vez desde cero, que es el valor predeterminado y resulta un poco molesto, así que es mejor desactivarlo y para ello, vamos a la pestaña Fórmulas de la cinta de opciones tal y como se muestra en la Ilustración 11 y pasamos a Manua.
C. Cada vez que queramos entrenar o simular el modelo necesitamos (al estar en modo manual) ejecutar la opcion calcular ahora.

Aplicación al caso, codigo desarrollado

El codigo propuesto es

```
#Importancion de la clase DecisionTreeClassifier del módulo sklearn.tree de la biblioteca scikit-learn
    from sklearn.tree import DecisionTreeClassifier
# Creacion y entrenamiento de modelo
    arbolDeDecision= DecisionTreeClassifier (min_samples_leaf = 3000, random_state = 12345)
    arbolDeDecision.fit(Predictores2, Ingresos_Y)
```

El objetivo de este codigo es usando **la clase** **DecisionTreeClassifier** del **módulo** **sklearn.tree** de la **biblioteca** **scikit-learn** (de aprendizaje automático – machine learning), crear y entrenar un modelo para clasificar a los ciudadanos según sus ingresos.

Paso 1. Importación de la clase

```
from sklearn.tree import DecisionTreeCalssifier
```

Esta línea **importa la clase DecisionTreeClassifier** que contiene toda la lógica necesaria para construir y entrenar un modelo de árbol de decisión. Piénsalo como si estuvieras trayendo una herramienta específica (el DecisionTreeClassifier) de una caja de herramientas (sklearn).

Este modelo se usa para tareas de clasificación (por ejemplo, predecir si un cliente ganará más o menos de cierto ingreso)

Paso 2. Creacion del modelo

Aquí creamos una instancia de la clase DecisionTreeClassifier y la guardamos en la variable arbolDeDecision. Al hacerlo, configuramos dos parámetros clave del modelo de decisión.

```
arbolDeDecision= DecisionTreeCalssifier (min_samples_leaf = 3000, random_state = 12345)
```

- min_samples_leaf=3000: significa que **cada hoja del árbol debe contener al menos 3000 muestras u observaciones**.
  Esto evita que el modelo genere hojas demasiado pequeñas (sobreajuste), haciendo que el árbol sea más general y menos sensible a pequeñas variaciones en los datos.
  Un árbol de decisión crece dividiendo los datos en ramas, y cada rama termina en una "hoja". El parámetro min_samples_leaf establece que cada hoja (o nodo final) del árbol debe contener al menos **3000 muestras**. Si una división resultara en una hoja con menos de 3000 muestras, esa división no se realiza. Esto evita que el árbol se vuelva demasiado complejo y memorice los datos de entrenamiento en lugar de aprender patrones generales.
- random_state=12345: Este parámetro asegura la **reproducibilidad** de los resultados. El algoritmo de árbol de decisión tiene un componente aleatorio en su proceso de entrenamiento. Al fijar random_state con un número, te aseguras de que, si ejecutas el código varias veces con los mismos datos, **obtendrás exactamente el mismo árbol** en todas las ejecuciones

Paso 3. Entrenamiento del modelo

```
arbolDeDecision.fit(Predictores2, Ingresos_Y)
```

Estas la línea más importante, aquí alimentamos el modelo con nuestros datos y entrenamos al modelo. El método .fit() es el que "enseña" al árbol a tomar decisiones.

- Predictores2: son las **variables independientes** (features) los valores de entrada, es decir, lo que usas para predecir, en nuestro ejemplo es la matriz o DataFrame con las **variables independientes** (edad, educación, ocupación, etc.)
- Ingresos_Y: es la **variable dependiente** (target) que contiene las **etiquetas o resultados** que el modelo debe aprender a predecir. En un problema de clasificación, podría ser una columna con valores como "Sí" o "No" (por ejemplo, si una persona va a comprar un producto), en nuestro caso son las categorías de ingresos (<= 50k y > 50K).

Lo que hace .fit() es **ajustar el árbol**: buscar las divisiones que mejor separen las clases en función de los predictores.

El modelo aprende patrones en los datos para poder hacer predicciones futuras.

En resumen, el código **crea un modelo de árbol de decisión**, le dice que no genere ramas con menos de 3000 datos y se asegura de que sus resultados sean siempre los mismos. Finalmente, **entrena este modelo** usando los datos de `Predictores2` para predecir los valores en `Ingresos_Y`. Una vez que esta línea se ha ejecutado, el objeto `arbolDeDecision` está "entrenado" y listo para hacer predicciones sobre nuevos datos que no haya visto antes.

¿Qué podrías hacer después?

Una vez entrenado el modelo, puedes usar:

```
predicciones = arbolDeDecision.predict(nuevos_datos)
```

Para predecir ingresos en nuevos casos.

Visualización del modelo, con Árbol de Decisión

Creado y entrenado el modelo el siguiente paso es visualizarlo, lo que nos facilita el análisis y entendimiento de lo que esta pasando en el modelo. Asi, el siguiente código nos visualiza el árbol de decisión en Excel usando matplotlib y scikit-learn:

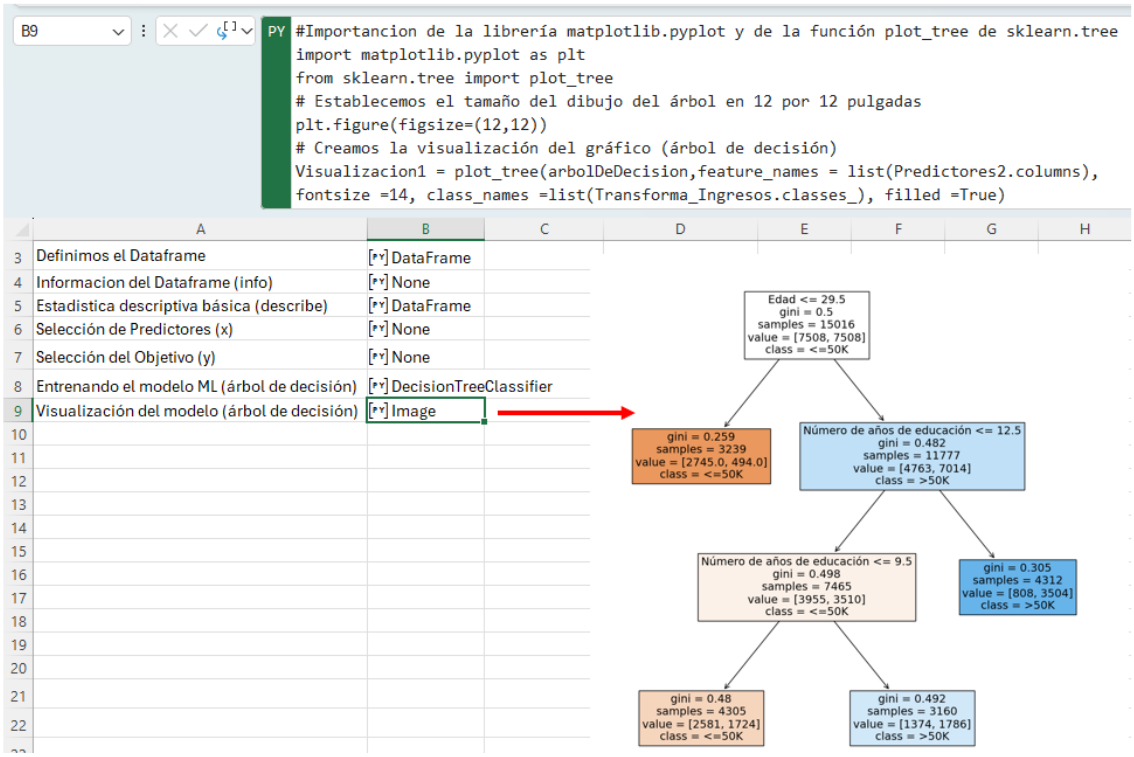


Ilustración 12

```
#Importancion de la librería matplotlib.pyplot y de la función plot_tree de sklearn.tree
    import matplotlib.pyplot as plt
    from sklearn.tree import plot_tree
# Establecemos el tamaño del dibujo del árbol en 12 por 12 pulgadas
    plt.figure(figsize=(12,12))
# Creamos la visualización del gráfico (árbol de decisión)
    Visualizacion1 = plot_tree(arbolDeDecision,feature_names = list(Predictores2.columns), fontsize
    =14, class_names =list(Transforma_Ingresos.classes_), filled =True)
```

Por tanto, este código sirve para **visualizar un árbol de decisión** entrenado con scikit-learn, dentro de Excel gracias a la integración con Python.

Explicación paso a paso.

Paso 1. Importación de librerías y funciones necesarias

```
import matplotlib.pyplot as plt
```

- matplotlib.pyplot: Es la librería más común para gráficos en Python. Le asigna el alias plt para que sea más fácil de usar en el código. plt se utiliza en la siguiente línea para configurar el tamaño del gráfico

- plot_tree de sklearn.tree: Esta función es la que genera el gráfico del árbol de decisión, es una función específica para visualizar árboles de decisión entrenados con DecisionTreeClassifier o DecisionTreeRegressor.

Paso 2. Definimos el tamaño del gráfico

```
plt.figure(figsize=(12,12))
```

Esto define el tamaño del gráfico en pulgadas. Un tamaño grande como 12x12 es útil para árboles complejos con muchas ramas y nodos.

Paso 3. Visualización del árbol

```
Visualizacion1 = plot_tree(arbolDeDecision,feature_names = list(Predictores2.columns), fontsize =14,
class_names =list(Transforma_Ingresos.classes_), filled =True)
```

Aquí se genera el gráfico del árbol de decisión. Los argumentos son:

- **arbolDeDecision**: Es el modelo de árbol de decisión que ya hemos entrenado anteriormente con DecisionTreeClassifier. La función plot_tree toma este objeto como base para dibujar el gráfico
- **feature_names = list(Predictores2.columns)**: Se le pasa los nombres de las variables predictoras para que el árbol muestre etiquetas en vez de índices (ej. “Edad”, “Ingreso”, etc.), es decir muestra los nombres de las variables predictoras en cada nodo del árbol (columnas del DataFrame Predictores2).
- **fontsize=14:** Define el tamaño de la letra en el gráfico.
- **class_names =list(Transforma_Ingresos.classes_)**: Muestra los nombres de las clases (etiquetas) en lugar de solo números. Esto es útil si usamos LabelEncoder para transformar etiquetas categóricas. Es decir etiquetas de clase como <=50 K o <50 K
- **filled=True:** Colorea los nodos del árbol según la clase predominante, lo que facilita la interpretación visual, es decir rellena los nodos con colores según la clase predominante.

El objeto Visualizacion1 que guardamos contiene los elementos gráficos creados (listas de textos, nodos, etc.), aunque lo normal es que no lo uses directamente, porque lo que importa es el gráfico en pantalla.

Resultado final:

Este código genera un gráfico visual del árbol de decisión que se puede ver directamente en Excel si estás usando Python integrado en Excel. cada nodo representa una condición sobre una variable, y las hojas indican la clase final. Es muy útil para interpretar modelos y explicar decisiones.

Veamos brevemente lo que muestra la Ilustración 13, en primer lugar, si tenemos 29.5 años o menos predice que ganas 50 000 dólares o menos al año.

Si tenemos 30 o mas años y tu nivel educativo es inferior a 13, lo que en Estados Unidos sería esencialmente inferior a la universidad, es decir, un título de secundaria o inferior, entonces bajas por este lado del árbol y entonces decimos: «Oye, ¿tu nivel educativo también es inferior al 10.º grado en Estados Unidos?», lo que sería inferior a la educación secundaria, entonces predecimos unos ingresos bajos; de lo contrario, predecimos unos ingresos altos y, de lo contrario, predecimos unos ingresos altos.

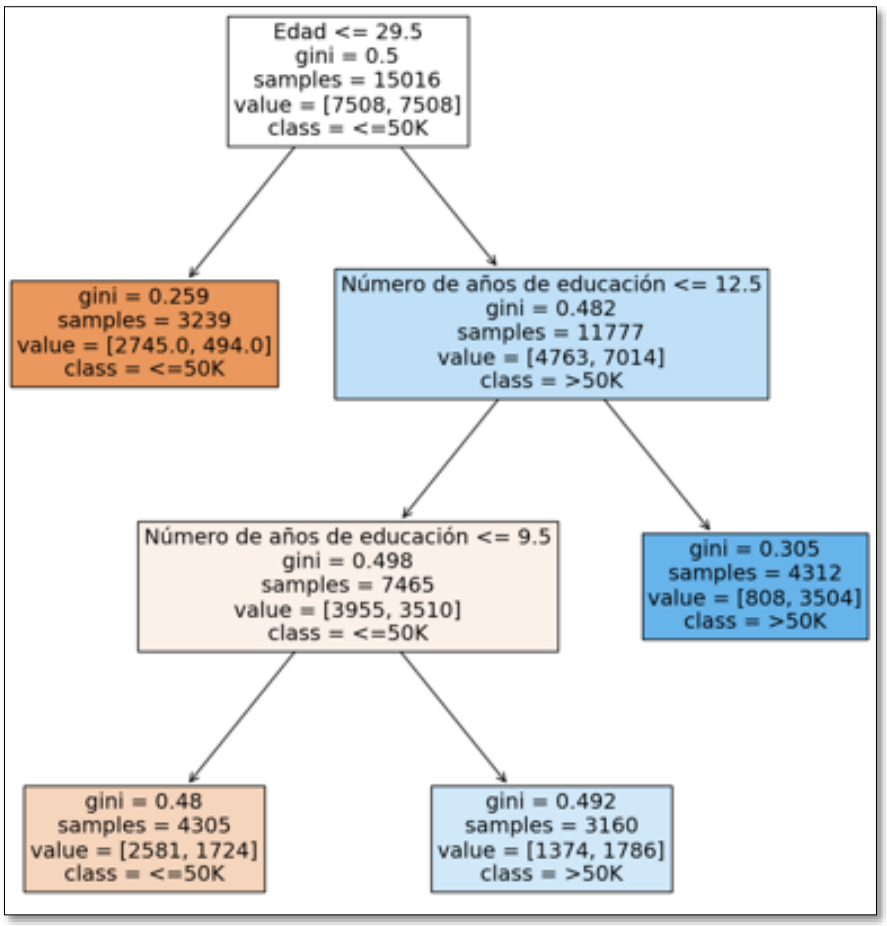


Ilustración 13

Ahora bien, de nuevo, este es un árbol muy simple, lo que dice es que estas son las características que realmente importan: la edad y la educación, de las cuatro que hemos elegido, la ocupación no aparece en ningún lugar del modelo de aprendizaje automático, lo cual es muy interesante, porque uno pensaría que la ocupación sería importante, que las horas semanales serían importantes, pero no, eso no es lo que muestra el modelo.

Este modelo puede ser bueno o puede no serlo, pero esto es lo que nos dice el modelo, esto es lo que ha aprendido, un patrón en los datos que ha aprendido para la predicción y el siguiente es el valor de confianza que ofrece el modelo.

Evaluación del modelo, matriz de confusión

Por tanto, vamos a evaluar que tan bueno es el modelo que hemos creado y para ello vamos a basarnos en la **matriz de confusion** que es una herramienta fundamental en Machine Learning para evaluar el rendimiento de un modelo de clasificación. Es especialmente útil cuando trabajas con algoritmos como árboles de decisión, Random Forest, SVM, redes neuronales, etc.

Básicamente es una tabla que compara las **predicciones del modelo** con los **valores reales** (etiquetas verdaderas) para un conjunto de datos de prueba. La matriz muestra cuántas veces el modelo clasificó correctamente o incorrectamente cada clase.

Estructura básica (para clasificación binaria):

	Predicción: Positivo	**Predicción: Negativo**
Real: Positivo	Verdadero Positivo (TP)	Falso Negativo (FN)
Real: Negativo	Falso Positivo (FP)	Verdadero Negativo (TN)

Es una tabla que permite visualizar el desempeño de un modelo de clasificación. Compara las **predicciones del modelo** con los **valores reales** y muestra cuántas veces acertó o se equivocó en cada clase

¿Qué significan estos valores?

- **TP (True Positive):** El modelo predijo "positivo" y realmente era "positivo".
- **TN (True Negative):** El modelo predijo "negativo" y realmente era "negativo".
- **FP (False Positive):** El modelo predijo "positivo" pero era "negativo" (error tipo I).
- **FN (False Negative):** El modelo predijo "negativo" pero era "positivo" (error tipo II).

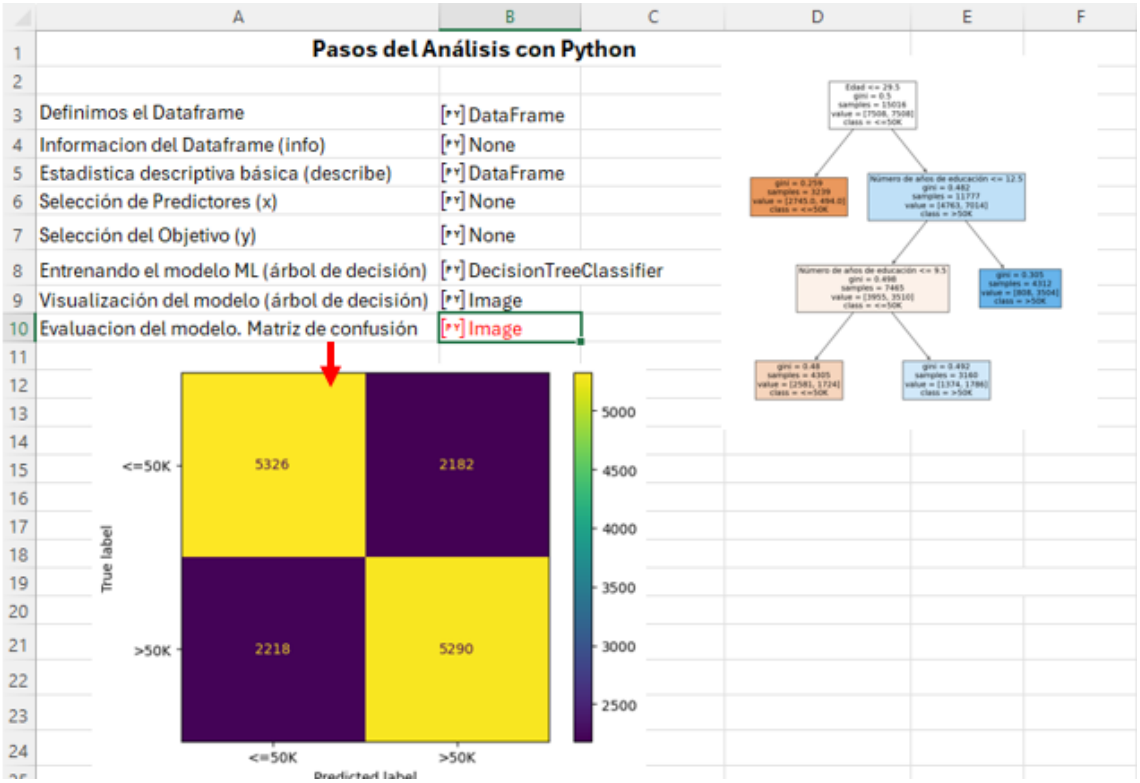


Ilustración 14

En concreto, vamos a evaluar el rendimiento del modelo de árbol de decisión (arbolDeDecision) utilizando una **matriz de confusión** para comparar las predicciones con los valores reales.

```
#Importancion de las librerías y funciones necesarias
    from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
# Recupera las clases predichas del modelo
    Valores_pred= arbolDeDecision.predict(Predictores2)
# Genera la matriz de confusión
    MatrizConfusion= confusion_matrix(Ingresos_Y, Valores_pred)
    VerMatConfus= ConfusionMatrixDisplay(MatrizConfusion, display_labels = Transforma_Ingresos.classes_)
# Crea la visualización de la matriz de confusión
    VerMatConfus.plot();
```

Por tanto, este código sirve para **visualizar un árbol de decisión** entrenado con scikit-learn, dentro de Excel gracias a la integración con Python.

Explicación paso a paso.

Paso 1. Importación de librerías y funciones necesarias

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

Aquí, importamos dos funciones clave de la biblioteca scikit-learn: confusion_matrix y ConfusionMatrixDisplay. La primera se encarga de crear la matriz de confusión, y la segunda se utiliza para visualizarla de manera gráfica y legible

- confusion_matrix: calcula la matriz de confusión. Crea la matriz de confusión a partir de los valores reales y los predichos.
- ConfusionMatrixDisplay: sirve para graficar esa matriz de confusión, permite visualizarla gráficamente.

Paso 2. Recupera las clases predichas del modelo

```
Valores_pred = arbolDeDecision.predict(Predictores2)
```

Usa tu modelo de árbol de decisión (arbolDeDecision) para hacer predicciones sobre los datos de entrada (Predictores2).

- arbolDeDecision: es nuestro modelo previamente entrenado.
- predict(Predictores2): genera predicciones usando los datos de entrada Predictores2.
- Valores_pred: contiene las clases predichas por el modelo, contiene las etiquetas que el modelo predice (ej. “>50K” o “<=50K”)

El resultado (Valores_pred) es un array con las clases predichas.

Paso 3. Genera la matriz de confusión

```
MatrizConfusion= confusion_matrix(Ingresos_Y, Valores_pred)
```

Aquí comparas las **etiquetas reales** (Ingresos_Y) con las **predicciones** (Valores_pred) para generar la matriz de confusión MatrizConfusion, es decir crea la matriz de confusión, que muestra cuántas veces el modelo clasificó correctamente o incorrectamente cada clase. Es decir, aquí creas la **matriz de confusión (**Valores_pred:**)** comparando:

- Ingresos_Y: → los valores **reales** (la clase correcta de cada fila, etiquetas verdaderas).
- Valores_pred: los valores **predichos por el model (**las clases predichas).

El resultado es una matriz 2x2 (si son dos clases) donde cada celda cuenta cuántos casos cayeron en esa combinación, por ejemplo, si estás clasificando ingresos como <=50K o >50K, la matriz podría verse así:

	Predicho <=50K	**Predicho >50K**
Real <=50K	2300	300
Real >50K	200	1200

Diagonal principal: aciertos.
Fuera de la diagonal: errores.

Paso 4. Genera la visualización de la matriz de confusión y la muestra

```
VerMatConfus= ConfusionMatrixDisplay(MatrizConfusion, display_labels=Transforma_Ingresos.classes_)
VerMatConfus.plot();
```

VerMatConfus: es el objeto que prepara la visualización de la matriz de confusión, esto hace que el gráfico no muestre solo 0 y 1, sino los nombres de las categorías

- Transforma_Ingresos.classes_: contiene los nombres de las clases (por ejemplo, ['<=50K', '>50K']).
- VerMatConfus.plot : Muestra la matriz de confusión como una imagen, con etiquetas y colores para facilitar la interpretación.

El resultado final es la siguiente matriz que en síntesis viene mostrar, por ejemplo, cuántas veces el modelo predijo correctamente un ingreso alto (TP) y cuántas veces se equivocó al predecir un ingreso bajo cuando en realidad era alto (FN).

- *Diagonal principal: aciertos.*
- *Fuera de la diagonal: errores.*

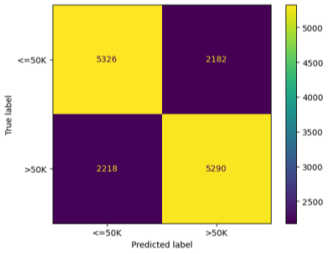


Ilustración 15

En resumen, este grafico nos permite ver:

- *Exactitud del modelo.*
- *Errores de clasificación (falsos positivos y falsos negativos).*
- *Desempeño por clase.*

Evaluación del modelo, métricas de la matriz de confusión

Para evaluar el rendimiento del modelo de clasificación de forma mas consistente o complementaria al anterior, podemos calcular métricas usando funciones de scikit-learn. En concreto las mas frecuentes son:

- **Precisión (Precision)**: Mide la proporción de verdaderos positivos (predicciones positivas correctas) sobre el total de predicciones positivas.

  *La **precisión** mide la exactitud de las predicciones positivas del modelo. En otras palabras, responde a la pregunta: "De todas las veces que el modelo predijo un resultado positivo, ¿cuántas fueron realmente correctas?". Es muy útil cuando el costo de un **falso positivo** (predecir positivo cuando en realidad es negativo) es alto. La fórmula es:*

$$Precisión = \frac{VerdaderosPositivos}{VerdaderosPositivos + FalsosPositivos}$$

- **Recall (Sensibilidad)**: Proporción de verdaderos positivos detectados correctamente.

  *El **recall**, o **sensibilidad**, mide la capacidad del modelo para encontrar todos los casos positivos reales. Responde a la pregunta: "De todos los casos positivos que existían en los datos, ¿cuántos fue capaz de capturar el modelo?". Esta métrica es importante cuando el costo de un **falso negativo** (predecir negativo cuando en realidad es positivo) es alto. La fórmula es:*

$$Recall = \frac{VerdaderosPositivos}{VerdaderosPositivos + FalsosNegativos}$$

- **Accuracy (Exactitud)**: Mide la proporción de predicciones correctas sobre el total de predicciones. Es la métrica más sencilla, pero puede ser engañosa en conjuntos de datos desequilibrados.

- **F1-score**: Media armónica entre precisión y recall.

  *El **F1-Score** es la media armónica de la precisión y el recall. Es una métrica útil cuando necesitas un equilibrio entre ambos, especialmente en conjuntos de datos desequilibrados (donde una clase tiene muchos más ejemplos que la otra). Un F1-Score alto significa que el modelo tiene un buen rendimiento tanto en precisión como en recall.*

Ejemplos

Imagina un modelo que diagnostica una enfermedad rara.

- **Precisión alta:** Significa que si el modelo dice que una persona está enferma, es muy probable que lo esté. Esto es crucial si un diagnóstico incorrecto (falso positivo) conlleva un tratamiento costoso o invasivo.
- **Recall alto:** Significa que el modelo es muy bueno detectando a todas las personas que realmente tienen la enfermedad. Esto es vital si no diagnosticar a un enfermo (falso negativo) es muy peligroso, como en el caso de una enfermedad grave.

Asi aplicado a nuestro caso tendríamos

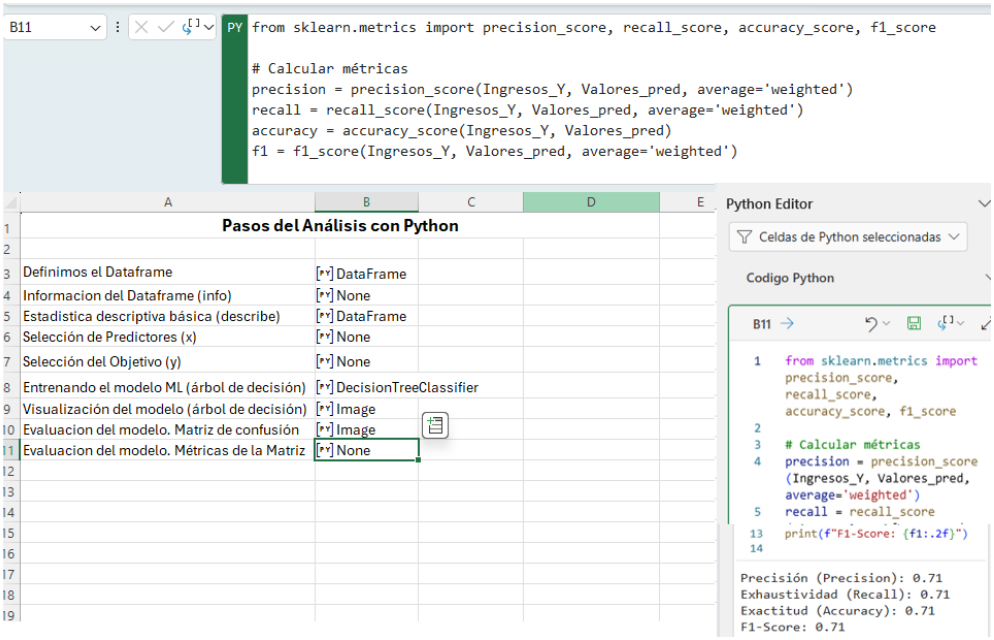


Ilustración 16

```
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score

# Calcular métricas
precision = precision_score(Ingresos_Y, Valores_pred, average='weighted')
recall = recall_score(Ingresos_Y, Valores_pred, average='weighted')
accuracy = accuracy_score(Ingresos_Y, Valores_pred)
f1 = f1_score(Ingresos_Y, Valores_pred, average='weighted')

# Mostrar resultados
print(f"Precisión (Precision): {precision:.2f}")
print(f"Exhaustividad (Recall): {recall:.2f}")
print(f"Exactitud (Accuracy): {accuracy:.2f}")
print(f"F1-Score: {f1:.2f}")
```

Explicación paso a paso.

Paso 1. Importación de librerías y funciones necesarias

```
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score
```

Aquí, importamos dos funciones clave de la biblioteca scikit-learn: confusion_matrix y ConfusionMatrixDisplay. La primera se encarga de crear la matriz de confusión, y la segunda se utiliza para visualizarla de manera gráfica y legible

- confusion_matrix: calcula la matriz de confusión. Crea la matriz de confusión a partir de los valores reales y los predichos.
- ConfusionMatrixDisplay: sirve para graficar esa matriz de confusión, permite visualizarla gráficamente.

Paso 2. Calculamos las métricas

```
# Calcular métricas

precision = precision_score(Ingresos_Y, Valores_pred, average='weighted')
recall = recall_score(Ingresos_Y, Valores_pred, average='weighted')
accuracy = accuracy_score(Ingresos_Y, Valores_pred)
f1 = f1_score(Ingresos_Y, Valores_pred, average='weighted')
```

Paso 3. Mostramos los resultados

```
    # Mostrar resultados
        print(f"Precisión (Precision): {precision:.2f}")
        print(f"Exhaustividad (Recall): {recall:.2f}")
        print(f"Exactitud (Accuracy): {accuracy:.2f}")
        print(f"F1-Score: {f1:.2f}")
"FlorIris[Sepalo_Anchura]))
```