INDICE

Preprocesamiento - Explorar dataFrame

Bibliotecas Pandas

Pandas es una de las bibliotecas más populares en Python para el análisis y manipulación de datos. Proporciona estructuras de datos flexibles y eficientes, así como herramientas para leer, escribir y procesar datos de manera rápida y sencilla.

Estructuras de datos: Pandas proporciona dos estructuras de datos fundamentales: Series y DataFrame.
Series: Es una estructura de datos unidimensional similar a un arreglo o una columna en una tabla. Cada elemento de una Serie tiene una etiqueta, conocida como índice.
DataFrame: Es una estructura de datos bidimensional similar a una tabla. Está compuesto por una colección de columnas, donde cada columna puede ser de un tipo diferente. Los DataFrames también tienen un índice para etiquetar filas y columnas.

Implementacion

Primero importamos la biblioteca Pandas y le asignamos un alias para mejor manipulación:

```
import pandas as pd
```

Segundo cargamos nuestros datos especificando el tipo de datos (csv, xlsx, etc) y se lo asignamos a una variable para poder manipularlos.
Formato CSV:

```
df_titanic = pd.read_csv('MI_RUTA/titanic.csv')
```

Formato EXCEL:

```
df_titanic = pd.read_excel('MI_RUTA/titanic.xlsx)
```

Ya estamos listos para trabajar con nuestros datos.

Preprocesamiento - Explorar dataFrame

Metodo head():

Es un método proporcionado por la biblioteca Pandas en Python que se utiliza para mostrar las primeras filas de un DataFrame o una Serie. Proporciona una vista previa rápida de los datos para comprender su estructura y contenido, este método por defecto muestra los 5 primeros registros del dataFrame, pero si le pasamos un número mayor como parámetro tomará éste como referencia de la cantidad de registro que debe mostrar.

```
data.head()
```

	Pclass	Sex	Age	Survived	Parch	SibSp
0	3	male	22.0	0	0	1
1	1	female	38.0	1	0	1
2	3	female	26.0	1	0	0
3	1	female	35.0	1	0	1
4	3	male	35.0	0	0	0

Método info(): Es un método proporcionado por la biblioteca Pandas en Python que se utiliza para obtener información detallada sobre un DataFrame. Proporciona un resumen de la estructura del DataFrame, incluyendo el tipo de datos de cada columna, el número de valores no nulos y el uso de memoria.

Ejemplo

```
df_titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Preprocesamiento - Explorar dataFrame

Método describe(): Se utiliza para calcular estadísticas descriptivas de un DataFrame o una Serie. Proporciona un resumen de los valores numéricos en el conjunto de datos, incluyendo el recuento de valores, la media, la desviación estándar, los valores mínimo y máximo, y los cuartiles. La función describe() es útil para obtener una visión general de los datos numéricos en el DataFrame y entender su distribución y dispersión.

```
df.describe(include='all')
```

	car_ID	symboling	CarName	fueltype
count	205.000000	205.000000	205	205
unique	NaN	NaN	147	2
top	NaN	NaN	toyota corona	gas
freq	NaN	NaN	6	185
mean	103.000000	0.834146	NaN	NaN
std	59.322565	1.245307	NaN	NaN
min	1.000000	-2.000000	NaN	NaN
25%	52.000000	0.000000	NaN	NaN
50%	103.000000	1.000000	NaN	NaN
75%	154.000000	2.000000	NaN	NaN
max	205.000000	3.000000	NaN	NaN

11 rows × 26 columns

Método shape(): Se utiliza para obtener la forma (dimensiones) de un DataFrame o una Serie. Devuelve una tupla que indica el número de filas y columnas en el conjunto de datos.

```
print('Train:',X_train_processed.shape)
print('Test:',X_test_processed.shape)
```

```
Train: (153, 55)
Test: (52, 55)
```

La primera posición de la tupla representa la cantidad de filas y la segunda posición las columnas.

Preprocesamiento - Trabajar datos faltantes

Eliminar filas: Para eliminar filas de un DataFrame en Pandas, puedes utilizar el método drop() se indica en el parámetro “index “ la posición de la fila que deseamos eliminar y como segundo parámetro le indicamos que confirme la acción pasándole el parámetro “inplace=True”.

Supongamos que tienes un DataFrame llamado df

Eliminar una fila por índice

```
df.drop(index=3, inplace=True)
```

Eliminar varias filas por índices

```
df.drop(index=[1, 4, 6], inplace=True)
```

Eliminar una fila por condición

```
df.drop(df[df['columna'] == valor].index, inplace=True)
```

Eliminar varias filas por condición

```
df.drop(df[df['columna'].isin([valor1, valor2, valor3])].index,
inplace=True)
```

Eliminar columnas:

Eliminar una columna por nombre

```
df.drop(columns='nombre_columna', inplace=True)
```

Eliminar varias columnas por nombres

```
df.drop(columns=['nombre_columna1', 'nombre_columna2'], inplace=True)
```

Recuerda que en todos los casos, puedes usar el parámetro inplace=True o asignar el resultado a un nuevo DataFrame para guardar los cambios realizados.

```
df = df[['columna1', 'columna2', 'columna3']]
```

Técnicas de imputación según el tipo de dato de la columna

La técnica de imputación "**mean()**" se utiliza para reemplazar los valores faltantes en una variable numérica utilizando el valor promedio de los datos existentes.

Preprocesamiento - Trabajar datos faltantes

```
# Crear un dataframe de ejemplo
data = {'Nombre': ['Juan', 'María', 'Pedro', 'Luis', 'Ana'],
        'Edad': [25, 30, None, 28, 35]}
df = pd.DataFrame(data)

# Imprimir el dataframe original
print("Dataframe original:")
print(df)

# Imputar los valores faltantes utilizando mean()
df_imputed = df.fillna(df['Edad'].mean())
print(df_imputed )
```

Dataframe original:

	Nombre	Edad
0	Juan	25.0
1	María	30.0
2	Pedro	NaN
3	Luis	28.0
4	Ana	35.0

Dataframe imputado:

	Nombre	Edad
0	Juan	25.0
1	María	30.0
2	Pedro	29.5
3	Luis	28.0
4	Ana	35.0

En el ejemplo anterior, podemos ver cómo seleccionamos los valores nulos del dataframe luego le pasamos el filtro con la columna que vamos a trabajar y por último la estrategia de imputación mean() ideal en este caso, ya que nos referimos a las edades de atletas en una competencia, tomando en cuenta que existen rango de edad para poner competir en esta.

Preprocesamiento - Trabajar datos faltantes

La técnica de imputación "mode()" se utiliza para reemplazar los valores faltantes en variables numéricas como categóricas reemplazandolos por el valor más se repita en la columna que seleccionemos para trabajar.

Dataframe original:

	A	B	C
0	Rojo	Gato	None
1	Azul	Perro	None
2	None	Perro	Pájaro
3	Verde	None	Gato
4	Rojo	Gato	Perro

Dataframe imputado:

	A	B	C
0	Rojo	Gato	Gato
1	Azul	Perro	Gato
2	Rojo	Perro	Pájaro
3	Verde	Gato	Gato
4	Rojo	Gato	Perro

En el ejemplo pasado vemos que en el columna A el valor que más se repite es el Rojo en efecto reemplaza los valores nulos de esa columna con Rojo, en la columna C vemos algo en particular, todo los datos sólo tienen una sola coincidencia, por eso el algoritmo toma el valor que está alfabéticamente primero y lo imputa en los datos faltantes.

Imputación de datos con Simple Imputer de la bibliotecas sklearn.impute

El SimpleImputer es una clase de la biblioteca scikit-learn.impute que se utiliza para imputar valores faltantes en conjuntos de datos de manera más automatizada. Proporciona estrategias predefinidas, como la imputación de la media, mediana o moda, para reemplazar los valores faltantes.

Preprocesamiento - Trabajar datos faltantes

Tomaremos el mismo ejemplo de imputación con la estrategia mean() aplicandolo a través del simple imputer.

```
# Crear un dataframe de ejemplo
data = {'A': [1, 2, None, 4, 5], 'B': [6, None, 8, None, 10],'C': [11, 12, 13, 14, 15]}
df = pd.DataFrame(data)

# Imprimir el dataframe original
print("Dataframe original:")
print(df)

# Crear una instancia de SimpleImputer con la estrategia de imputación de la media
imputer = SimpleImputer(strategy='mean')

# Imputar los valores faltantes
df_imputed = imputer.fit_transform(df)

# Convertir el resultado en un dataframe
df_imputed = pd.DataFrame(df_imputed, columns=df.columns)

# Imprimir el dataframe imputado
print("\nDataframe imputado:")
print(df_imputed)
```

Dataframe original:

	A	B	C
0	1.0	6.0	11
1	2.0	NaN	12
2	NaN	8.0	13
3	4.0	NaN	14
4	5.0	10.0	15

Dataframe imputado:

	A	B	C
0	1.0	6.0	11.0
1	2.0	8.0	12.0
2	3.0	8.0	13.0
3	4.0	8.0	14.0
4	5.0	10.0	15.0

Codificación ordinal a variables categóricas con jerarquía

Antes de llegar a la práctica debemos saber la definición de codificación ordinal y sus características.

La codificación ordinal es utilizada para variables categóricas ordenadas, donde los diferentes niveles de la variable tienen un orden específico o jerarquía. La codificación ordinal asigna un valor numérico a cada nivel de la variable categórica, de manera que se preserve el orden relativo entre los niveles.

Ejemplo de variable categoría:
Tamaño: Pequeño, mediano, grande.
En este mismo orden le asignaremos a pequeño: 0, mediano:1 y grande:2, de manera que no pierdan la jerarquía que los categoriza.

Método LabelEncoder(): Es una función proporcionada por la biblioteca scikit-learn en Python que se utiliza para codificar variables categóricas en valores numéricos. Convierte las etiquetas de las categorías en valores numéricos enteros, asignando un número único a cada categoría. Es importante tener en cuenta que el método LabelEncoder() asume que no hay un orden o jerarquía específica entre las categorías. Simplemente asigna un número único a cada categoría sin considerar su relación relativa.

Ejemplos:

```
# Crear un DataFrame de ejemplo
data = {'Tamaño': ['Pequeño', 'Mediano', 'Grande', 'Mediano', 'Grande', 'Pequeño']}
df = pd.DataFrame(data)

# Crear una instancia de LabelEncoder
encoder = LabelEncoder()
# Codificar la variable categórica ordinal
df['Tamaño_ordinal'] = encoder.fit_transform(df['Tamaño'])
```

Codificación ordinal a variables categóricas con jerarquía

Resultado:

	Tamaño	Tamaño_ordinal
0	Pequeño	1
1	Mediano	2
2	Grande	0
3	Mediano	2
4	Grande	0
5	Pequeño	1

Método replace(): Para realizar una codificación ordinal personalizada. La idea es asignar valores numéricos a las categorías de acuerdo con un orden específico utilizando un diccionario de reemplazo.

Ejemplos:

```
data = {'Tamaño': ['Pequeño', 'Mediano', 'Grande', 'Mediano', 'Grande', 'Pequeño']}
df = pd.DataFrame(data)
```

```
# Definir el orden de los niveles
orden = {'Pequeño': 1, 'Mediano': 2, 'Grande': 3}
```

```
# Aplicar la codificación ordinal utilizando replace()
df['Tamaño_ordinal'] = df['Tamaño'].replace(orden)
```

	Tamaño	Tamaño_ordinal
0	Pequeño	1
1	Mediano	2
2	Grande	3
3	Mediano	2
4	Grande	3
5	Pequeño	1

Ejemplos:

Codificación ordinal a variables categóricas con jerarquía

```
# Crear un DataFrame de ejemplo
data = {'Tamaño': ['Pequeño', 'Mediano', 'Grande', 'Mediano', 'Grande', 'Pequeño']}
df = pd.DataFrame(data)

# Definir el orden de los niveles
orden = ['Pequeño', 'Mediano', 'Grande']

# Aplicar la codificación ordinal
df['Tamaño_ordinal'] = df['Tamaño'].apply(lambda x: orden.index(x))
```

En el ejemplo anterior le estamos agregando una nueva columna a nuestro dataFrame llamada 'Tamaño_ordinal' que contendrá el valor ordinal que la función lambda le asignará. Dentro de la función lambda podemos ver que el array “orden” con ayuda del método “index” que recibe un valor el cual en este caso se lo pasamos por la variable “x” que representa los valores de cada fila del dataFrame, en donde confirmamos si existe dentro dicho valor dentro del array “orden” este devolverá como resultado el índice de ese valor. En efecto, se le asignan a la nueva columna y se muestra en el dataFrame de la siguiente manera.

	Tamaño	Tamaño_ordinal
0	Pequeño	0
1	Mediano	1
2	Grande	2
3	Mediano	1
4	Grande	2
5	Pequeño	0

Codificación one-hot a variables categóricas sin jerarquía

Las características nominales representan distintas categorías o grupos, pero no existe una relación de orden o magnitud entre ellas. Algunos ejemplos comunes de características nominales son el color de los ojos (azul, verde, marrón), el género (masculino, femenino), el estado civil (soltero, casado, divorciado), entre otros.

Método One Encoder: La codificación one-hot crea nuevas columnas binarias para cada categoría única en la variable categórica original. Estas columnas binarias indican la presencia o ausencia de cada categoría en una instancia de datos.

```
# Crear un DataFrame de ejemplo
data = {'Color': ['Rojo', 'Verde', 'Azul', 'Verde', 'Rojo']}
df = pd.DataFrame(data)
# Crear una instancia de OneHotEncoder
encoder = OneHotEncoder(sparse=False)
# Realizar la codificación one-hot
encoded_features = encoder.fit_transform(df)
# Crear un nuevo DataFrame con las características codificadas
df_encoded =
pd.DataFrame(encoded_features,columns=encoder.get_feature_names())
```

	x0_Azul	x0_Rojo	x0_Verde
0	0.0	1.0	0.0
1	0.0	0.0	1.0
2	1.0	0.0	0.0
3	0.0	0.0	1.0
4	0.0	1.0	0.0

Codificación one-hot a variables categóricas sin jerarquía

Hiperparametros para optimización del algoritmo OneHotEncoder

Hiperparametros:

categories (opcional): Especifica las categorías únicas de la variable categórica. Puedes pasar una lista de listas que contengan las categorías para cada variable categórica si tienes múltiples columnas. Si no se especifica, se inferirán automáticamente las categorías de los datos de entrada.

drop (opcional): Determina si se debe eliminar una columna de cada conjunto de características codificadas. Si se establece en "first", se eliminará la primera columna de cada conjunto de características codificadas para evitar la multicolinealidad. El valor predeterminado es None, lo que significa que no se eliminará ninguna columna.

sparse (opcional): Indica si la salida codificada debe ser una matriz dispersa (True) o una matriz densa (False). El valor predeterminado es False, lo que produce una matriz densa.

handle_unknown (opcional): Especifica cómo manejar las categorías desconocidas durante la transformación. Puede tomar los valores "error" (lanzará un error), "ignore" (ignorará las categorías desconocidas) o "value" (asignará un valor especificado). El valor predeterminado es "error".

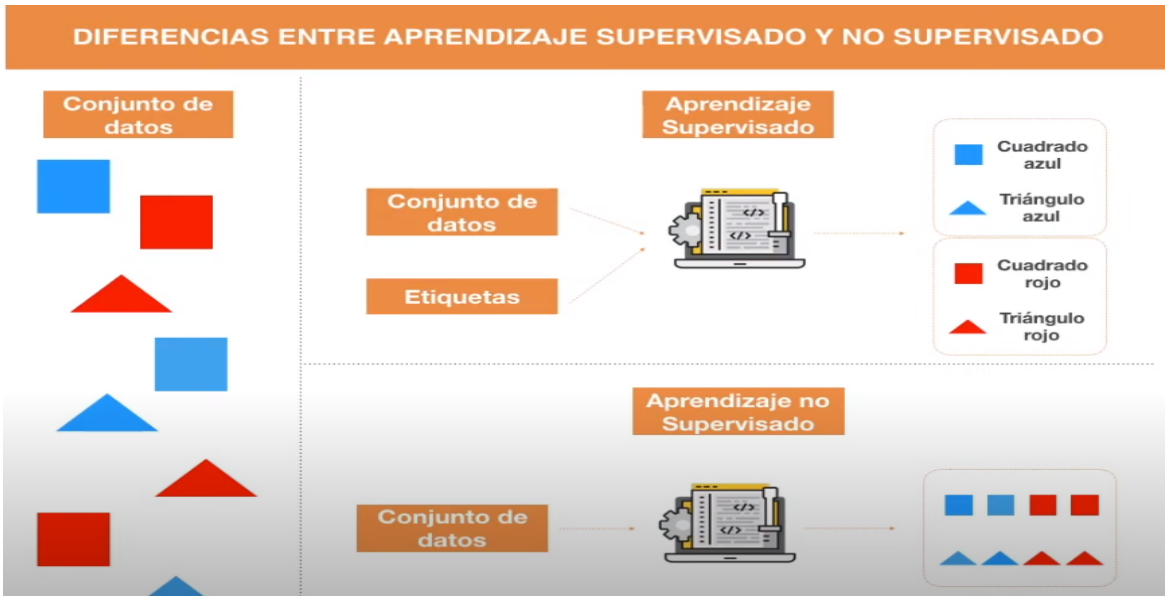
Tipos de aprendizajes

Supervisado

El aprendizaje supervisado es una técnica de aprendizaje automático en la que se utilizan datos etiquetados para entrenar un modelo y realizar predicciones o tomar decisiones basadas en esos datos. En el aprendizaje supervisado, se cuenta con un conjunto de datos de entrenamiento que consiste en características (variables independientes) y los valores correspondientes del objetivo (variable dependiente). Es decir, la salida del algoritmo ya es conocida.

No supervisado

El aprendizaje no supervisado es una técnica de aprendizaje automático en la que se utilizan datos no etiquetados para descubrir patrones, estructuras o relaciones ocultas en los datos. Está estrechamente más ligado a la inteligencia artificial, ya que el algoritmo aprende de manera independiente en base a los datos que le van llegando. No existe un conjunto de datos de entrenamiento y los resultados son desconocidos, el algoritmo entra a ciegas al problema y con solo operaciones lógicas para guiarlo. Algunos ejemplos de algoritmos son clustering, k-mean y reglas de asociación.



Fuente: AprendeIA con Ligdi Gonzalez

En el ejemplo anterior, vemos como en el aprendizaje supervisado la máquina etiqueta las diferentes categorías por forma geométrica y color, mientras que en el aprendizaje no supervisado la máquina agrupa por forma geométrica y color sin etiquetar ni categorizar los datos que encuentra.

Definir las características (X) y el objetivo (y)

Definir objetivo para modelos de regresión

Las características X, también conocidas como variables independientes o predictores, son las variables que se utilizan para predecir o explicar el valor de la variable objetivo "y". Estas características pueden ser numéricas o categóricas y se representan como una matriz o un dataframe. Cada fila representa una observación o instancia, mientras que cada columna representa una característica específica.

Por ejemplo, si estás construyendo un modelo de regresión para predecir el precio de una casa, las características podrían incluir el tamaño de la casa, el número de habitaciones, la ubicación, etc.

El objetivo Y, también conocido como variable dependiente o variable a predecir, es la variable que se desea predecir o estimar utilizando las características (X). En el contexto de un modelo de regresión, el objetivo generalmente es una variable numérica continua.

Continuando con el ejemplo anterior, el objetivo podría ser el precio de la casa. Cada valor del objetivo se asocia con una observación específica en los datos de características.

Definir objetivo para modelos clasificación

Tomando en cuenta la definición de características X, la cuales representan el mismo significado para los modelos de clasificación, tienen como diferencia que estas características pueden ser numéricas, categóricas o incluso **textuales (correo electrónico, etc…)**. Se representan igual que modelos de regresión en el dataframe, donde cada fila representa una instancia o ejemplo, y cada columna representa una característica específica.

Definir las características (X) y el objetivo (y)

Por ejemplo, si estás construyendo un modelo de clasificación para predecir si un correo electrónico es spam o no, las características podrían incluir la frecuencia de palabras clave, la longitud del correo electrónico, la presencia de enlaces sospechosos, etc.

El objetivo Y en este tipo de modelos, no deja de ser la variable dependiente de las características o más bien el valor a predecir en base al resultado de las características. En un modelo de clasificación, el objetivo es una variable categórica discreta. Puede ser binario (dos clases) o multiclase (más de dos clases). Siguiendo con el ejemplo anterior, el objetivo sería la categoría "spam" = 1 o "no spam" = 0 para cada correo electrónico en los datos.

Realizar un train test split a los datos para prepararlos para el aprendizaje automático supervisado

El train_test_split es una función que se utiliza para dividir un conjunto de datos en dos conjuntos más pequeños: un conjunto de entrenamiento y un conjunto de prueba. Esta división es fundamental en el aprendizaje automático para evaluar el rendimiento de un modelo en datos no vistos durante el entrenamiento.

El conjunto de entrenamiento se utiliza para ajustar los parámetros del modelo y enseñarle a hacer predicciones. El modelo aprenderá patrones y relaciones en los datos a medida que se ajuste a ellos.
El conjunto de prueba, por otro lado, se utiliza para evaluar qué tan bien el modelo generaliza a nuevos datos. Al hacer predicciones en el conjunto de prueba y compararlas con las etiquetas verdaderas, se puede medir la precisión y el rendimiento del modelo.

La función train_test_split toma como entrada los datos originales y el objetivo (si corresponde) y devuelve cuatro conjuntos de datos: X_train, X_test, y_train y y_test. Los

Definir las características (X) y el objetivo (y)

conjuntos X representan las características (variables independientes), mientras que los conjuntos y representan la variable objetivo (variable dependiente). Por lo tanto, los conjuntos X_train y y_train se utilizan para entrenar el modelo, mientras que los conjuntos X_test y y_test se utilizan para evaluar su rendimiento.

Ejemplo:

Cargar el conjunto de datos Iris
```
iris = load_iris()
```

Separar las características (X) y el objetivo (y)
```
X = iris.data
y = iris.target
```

Realizar el split de entrenamiento y prueba
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

A continuación, utilizamos la función train_test_split para realizar el split de entrenamiento y prueba. En este caso, especificamos que queremos que el 20% de los datos se utilicen como conjunto de prueba (test_size=0.2) y utilizamos el parámetro random_state para asegurar que el split sea reproducible.

Finalmente, los conjuntos de entrenamiento y prueba se asignan a las variables X_train, X_test, y_train y y_test, que puedes utilizar para entrenar y evaluar tu modelo de aprendizaje automático.

Escalar cualquier característica numérica

StandardScaler de la bibliotecas sklearn.preprocessing

StandardScaler es una técnica comúnmente utilizada en el preprocesamiento de datos en el aprendizaje automático para estandarizar las características de un conjunto de datos. Su objetivo principal es transformar las características de manera que tengan una media de cero y una desviación estándar de uno.

La estandarización de características es importante porque muchos algoritmos de **aprendizaje automático** asumen que los datos están distribuidos normalmente y que las características están en la misma escala. Al estandarizar las características, se elimina la media y se escala la varianza de manera que todas las características tengan una distribución similar y contribuyan de manera equitativa al modelo.

Ejemplo:

```
# Crear una instancia de StandardScaler
scaler = StandardScaler()

# Ajustar el scaler a los datos de entrenamiento y transformar los datos
X_train_scaled = scaler.fit_transform(X_train)

# Aplicar la misma transformación a los datos de prueba
X_test_scaled = scaler.transform(X_test)
```

Valores original:

	Feature1	Feature2
4	5	10
2	3	8
0	1	6
3	4	9

Valores escalados:

```
array([[ 1.18321596,  1.18321596],
       [-0.16903085, -0.16903085],
       [-1.52127766, -1.52127766],
       [ 0.50709255,  0.50709255]])
```

Concatenar todas las características de vuelta a un DataFrame

En NumPy, puedes concatenar arrays utilizando la función numpy.concatenate(). Esta función permite unir arrays a lo largo de un eje especificado.

```
# Crear dos DataFrames
df1 = pd.DataFrame({'A': [1, 2, 3],
                    'B': ['a', 'b', 'c']})

df2 = pd.DataFrame({'A': [4, 5, 6],
                    'B': ['d', 'e', 'f']})

# Concatenar los DataFrames
concatenated_df = pd.concat([df1, df2], axis=0)
print(concatenated_df)
```

Resultado:

```
  A B
0 1 a
1 2 b
2 3 c
0 4 d
1 5 e
2 6 f
```

Los índices de las filas se mantienen de forma predeterminada, por lo que en el DataFrame concatenado se generan nuevos índices. Si deseas restablecer los índices, puedes utilizar el método reset_index():

```
concatenated_df = pd.concat([df1, df2], axis=0).reset_index(drop=True)
```