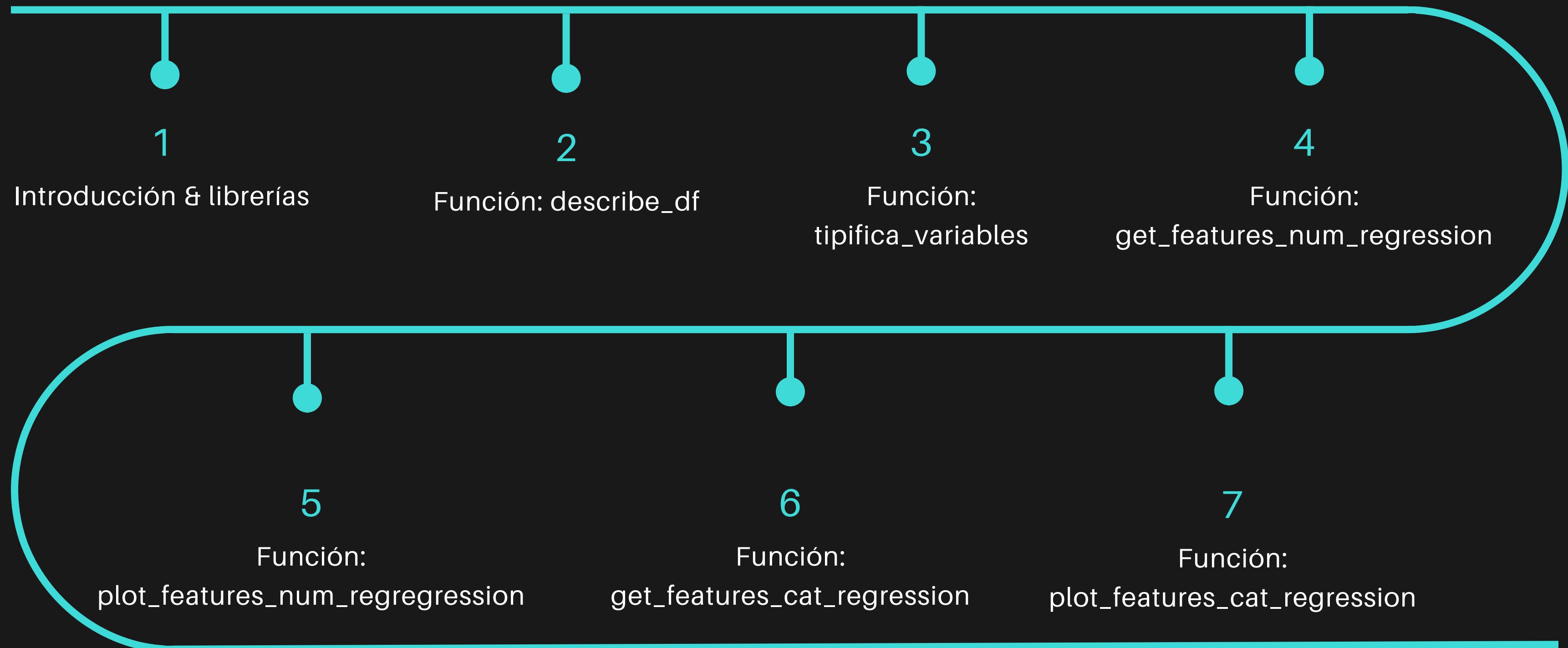


Team Challenge - Toolbox

Machine Learning por Jesús González, Víctor Caballero y Viridiana Espinosa



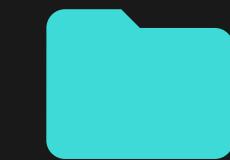


TOOLBOX



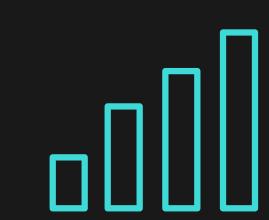
OBJETIVO

Construcción de un modelo de herramientas básicas.



LIBRERIAS

```
import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
from scipy.stats import pearsonr, f_oneway, ttest_ind
```



FINALIDAD

Preparar la creación de modelos de MachineLearning de forma sencilla.

def describe (df):

```
1 toolbox_ML.describe_df(df_precios)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT	MEDV
Tipo de Dato	float64												
% Valores Nulos	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Valores Únicos	504	26	76	2	81	446	356	412	9	66	46	455	229
% Cardinalidad	99.6	5.14	15.02	0.4	16.01	88.14	70.36	81.42	1.78	13.04	9.09	89.92	45.26

```
def describe_df(df):  
  
    resumen = {  
        'Tipo de Dato': df.dtypes,  
        '% Valores Nulos': df.isnull().mean() * 100,  
        'Valores Únicos': df.nunique(),  
        '% Cardinalidad': (df.nunique() / len(df)) * 100  
    }  
  
    resumen_df = pd.DataFrame(resumen)  
  
    # Ajustar el formato de la salida (por ejemplo, redondear los porcentajes)  
    resumen_df['% Valores Nulos'] = resumen_df['% Valores Nulos'].round(2)  
    resumen_df['% Cardinalidad'] = resumen_df['% Cardinalidad'].round(2)  
  
    return resumen_df.T
```

Genera un resumen informativo sobre las columnas

1

Tipo de dato



2

Porcentaje de valores Nulos



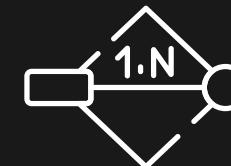
3

Número de valores únicos por columna



4

Porcentaje de cardinalidad



`def tipifica_variables(df, umbral_categoria, umbral_continua):`

```
1 toolbox_ML.tipifica_variables(df_precios,10,0.90)

  nombre_variable tipo_sugerido
0      CRIM  Numerica Continua
1        ZN  Numerica Continua
2     INDUS  Numerica Continua
3       CHAS      Binaria
4      NOX  Numerica Continua
5       RM  Numerica Continua
6      AGE  Numerica Continua
7       DIS  Numerica Continua
8       RAD      Categórica
9       TAX  Numerica Continua
10    PTRATIO  Numerica Continua
11     LSTAT  Numerica Continua
12     MEDV  Numerica Continua

def tipifica_variables(df, umbral_categoria, umbral_continua):
    # Asegúrate de que umbral_categoria es un entero
    try:
        umbral_categoria = int(umbral_categoria)
    except ValueError:
        print(f"Error: umbral_categoria debe ser un número, se recibió: {umbral_categoria}")
        return None

    # Inicializar una lista para almacenar el resultado
    sugerencias = []

    # Recorrer cada columna del DataFrame
    for col in df.columns:
        # Calcular la cardinalidad (número de valores únicos)
        cardinalidad = df[col].nunique()
        # Calcular el porcentaje de cardinalidad
        porcentaje_cardinalidad = (cardinalidad / len(df)) * 100

        # Determinar el tipo sugerido
        if cardinalidad == 2:
            tipo_sugerido = "Binaria"
        elif cardinalidad < umbral_categoria:
            tipo_sugerido = "Categórica"
        elif porcentaje_cardinalidad >= umbral_continua:
            tipo_sugerido = "Numerica Continua"
        else:
            tipo_sugerido = "Numerica Discreta"

        # Añadir la sugerencia a la lista
        sugerencias.append({
            'nombre_variable': col,
            'tipo_sugerido': tipo_sugerido
        })

    # Convertir la lista de sugerencias en un DataFrame
    resultado_df = pd.DataFrame(sugerencias)

    return resultado_df
```

1

2

3

Sugiere el tipo de c/columna basándose en la cardinalidad y % de valores únicos vs tamaño de la muestra

Tipo de variable:

- Binaria
- Categórica
- Numérica Discreta
- Numérica Continua

umbral_continua:
float - 0.90

..... ➡ >= Número continua

 umbral_categoria:
int - 10
< Número discreta

> entonces categórica 

< entonces numérica 

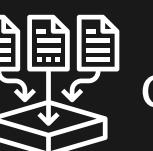
```
def get_features_num_regression(df,  
target_col, umbral_corr, pvalue=None):
```

```
1 lista_num = toolbox_ML.get_features_num_regression(df_precios, "MEDV", 0.5, 0.05)  
2 print(lista_num)  
[7] ✓ 0.0s  
... ['RM', 'PTRATIO', 'LSTAT']
```

```
def get_features_num_regression(df, target_col, umbral_corr, pvalue=None):  
  
    # Verifica que la columna objetivo existe y es numérica  
    if target_col not in df.columns:  
        print("Error: target_col debe ser una columna existente en el DataFrame.")  
        return None  
  
    # Verificar que la columna objetivo es numérica  
    if df[target_col].dtype not in ['int64', 'float64']:  
        print("Error: target_col debe ser numérica.")  
        return None  
  
    # Filtrar las columnas numéricas  
    numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns  
  
    # Excluir la columna objetivo de las columnas a analizar  
    numeric_cols = [col for col in numeric_cols if col != target_col]  
  
    # Lista para almacenar las columnas seleccionadas.  
    selected_columns = []  
  
    # Para cada columna numérica, calcular la correlación con la columna objetivo  
    for col in numeric_cols:  
        correlacion, p_val = pearsonr(df[col].dropna(), df[target_col].dropna())  
  
        # Si la correlación supera el umbral  
        if abs(correlacion) >= umbral_corr:  
            # Verificar si el p-value es significativo si se especifica  
            if pvalue is not None:  
                if p_val <= pvalue:  
                    selected_columns.append(col)  
            else:  
                selected_columns.append(col)  
  
    return selected_columns
```

Identifica columnas numéricas que estén correlacionadas con la columna (target_col), a través de umbrales de correlación y p-value.

Argumentos de la función:



1

df (pd.DataFrame): datos



2

target_col (str):
columna objetivo

3

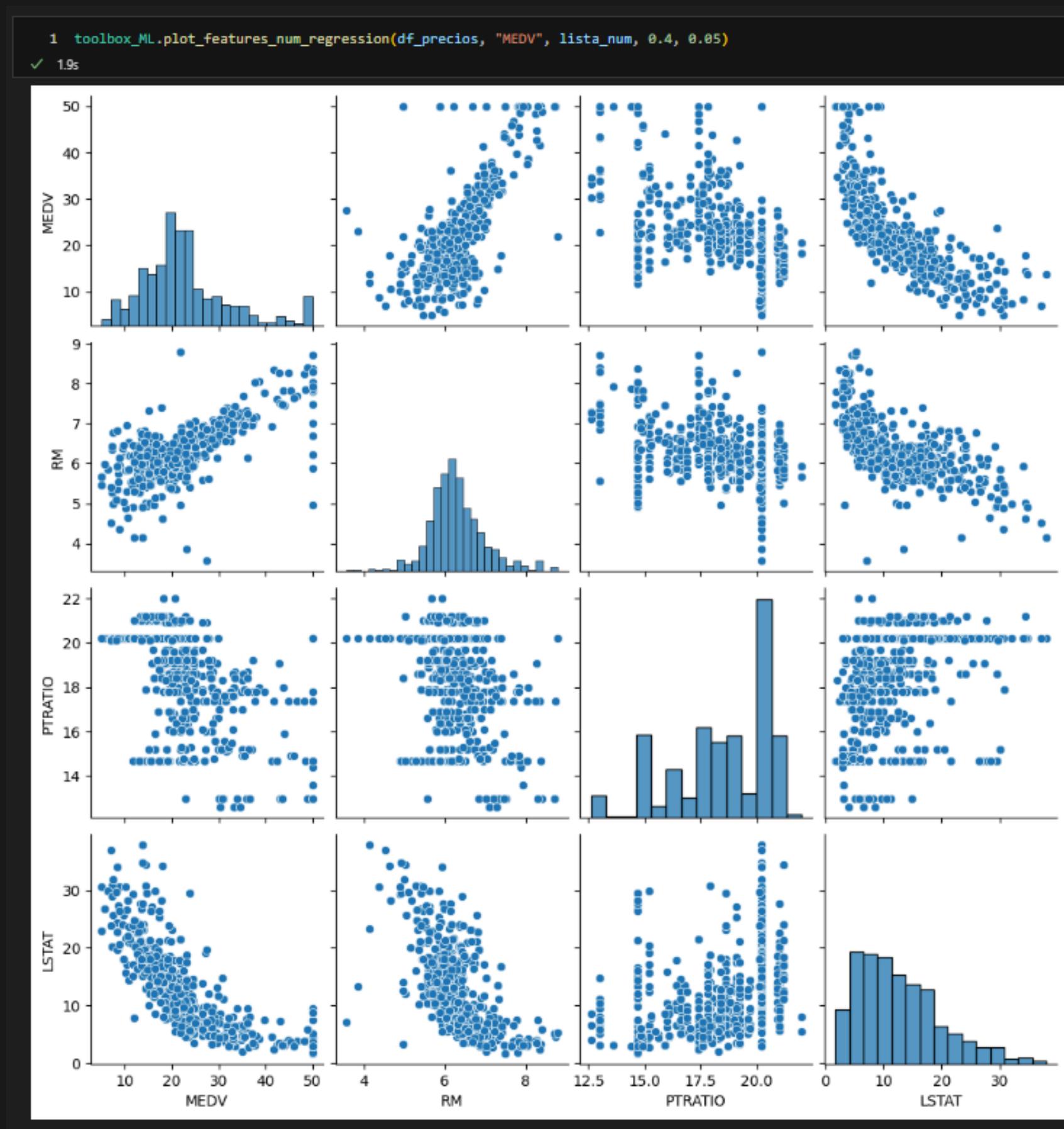
umbral_corr (float): valor
absoluto mínimo de
correlación para seleccionar
las características.

4

pvalue (float, opcional):
filtra según la significancia
de la correlación.
*Si es None, no se realiza
este test.* ✗

5

Retorna:
Una lista con los nombres de las columnas que están
correlacionadas con la columna objetivo por encima del umbral.



```

def plot_features_num_regression(df,
target_col="", columns=[], umbral_corr=0,
pvalue=None):

```

Relación entre variables numéricas y una variable objetivo numérica.

- Dependiendo del nivel de correlación y el p-value.

```

def plot_features_num_regression(df, target_col="", columns=[], umbral_corr=0, pvalue=None):

    # Paso 1: Verificar que 'target_col' está en el DataFrame
    if target_col not in df.columns:
        print("Error: La columna objetivo no está en el DataFrame.")
        return None

    # Paso 2: Verificar que 'target_col' es numérica
    if not (df[target_col].dtype == 'int64' or df[target_col].dtype == 'float64'):
        print("Error: La columna objetivo debe ser numérica (entero o decimal).")
        return None

    # Paso 3: Si no hay columnas especificadas, seleccionar todas las numéricas
    if not columns:
        columns = [col for col in df.columns if df[col].dtype in ['int64', 'float64']]

    # Paso 4: Excluir la columna objetivo de las columnas a analizar
    if target_col in columns:
        columns.remove(target_col)

    # Lista para guardar las columnas que cumplen con los criterios
    selected_columns = []

    # Paso 5: Calcular la correlación para cada columna numérica
    for col in columns:
        correlacion, p_val = pearsonr(df[col].dropna(), df[target_col].dropna())

        if abs(correlacion) >= umbral_corr:
            if pvalue is not None:
                if p_val <= pvalue:
                    selected_columns.append(col)
            else:
                selected_columns.append(col)

    # Crear gráficos pairplot en bloques
    for i in range(0, len(selected_columns), 5):
        subset_columns = [target_col] + selected_columns[i:i+5]
        sns.pairplot(df[subset_columns])
        plt.show()

    # Retornar las columnas seleccionadas
    return selected_columns

```

```
def  
get_features_cat_regression(df,target_col,  
pvalue=0.05, umbral_cardinalidad=10):
```

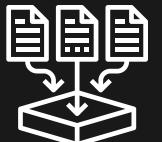
```
1 lista_cat = toolbox_ML.get_features_cat_regression(df_precios, "MEDV")  
2 print(lista_cat)  
✓ 0.0s  
['CHAS', 'RAD']
```

```
def get_features_cat_regression(df, target_col, pvalue=0.05, umbral_cardinalidad=10):  
  
    # Comprobaciones de entrada  
    if target_col not in df.columns:  
        print("Error: La columna 'target_col' no existe en el DataFrame.")  
        return None  
  
    if not pd.api.types.is_numeric_dtype(df[target_col]):  
        print("Error: La columna 'target_col' no es una variable numérica.")  
        return None  
  
    if not isinstance(pvalue, float) or not (0 < pvalue < 1):  
        print("Error: El valor de 'pvalue' debe ser un float entre 0 y 1.")  
        return None  
  
    if not isinstance(umbral_cardinalidad, int) or umbral_cardinalidad <= 0:  
        print("Error: El 'umbral_cardinalidad' debe ser un entero positivo.")  
        return None  
  
    # Identificar columnas categóricas basadas en la cardinalidad  
    columnas_categoricas = [col for col in df.columns if df[col].nunique() < umbral_cardinalidad]  
  
    # Lista para almacenar las columnas que cumplen con el criterio  
    columnas_significativas = []  
  
    # Iterar sobre las columnas categóricas y realizar el test adecuado  
    for col in columnas_categoricas:  
        categorias = df[col].dropna().unique()  
  
        if len(categorias) == 2:  
            # Prueba t de Student para variables binarias  
            grupo1 = df[df[col] == categorias[0]][target_col].dropna()  
            grupo2 = df[df[col] == categorias[1]][target_col].dropna()  
            stat, p = ttest_ind(grupo1, grupo2)  
        elif len(categorias) > 2:  
            # ANOVA para variables categóricas con más de dos categorías  
            grupos = [df[df[col] == categoria][target_col].dropna() for categoria in categorias]  
            stat, p = f_oneway(*grupos)  
        else:  
            continue # Si la columna no tiene suficientes categorías, pasar a la siguiente  
  
        # Verificar si el p-valor es menor que el umbral  
        if p < pvalue:  
            columnas_significativas.append(col)  
  
    return columnas_significativas
```

Identifica las columnas categóricas que tienen una relación estadísticamente significativa con una columna numérica objetivo - mediante pruebas estadísticas como ANOVA o la prueba t de Student.

Argumentos de la función:

1



df (pd.DataFrame): datos

2



target_col (str):
columna objetivo

3

pvalue (float, 0.05):
filtra según la significancia de
la correlación.

5

< variable categórica vs
objetivo = estadísticamente
significativa

4

umbral_cardinalidad (int, 10):
Límite máximo de valores
únicos para considerar una
categórica.



Retorna:

Lista con los nombres de las columnas categóricas que tienen una
relación estadísticamente significativa con la variable objetivo.

```

def plot_features_cat_regression(df, target_col, columns, pvalue=0.05, with_individual_plot=False):

    # Verificaciones de entrada
    if target_col == "":
        print("Error: Debes especificar una columna objetivo 'target_col'.")
        return None

    if target_col not in df.columns:
        print(f"Error: La columna objetivo '{target_col}' no existe en el DataFrame.")
        return None

    if not pd.api.types.is_numeric_dtype(df[target_col]):
        print(f"Error: La columna objetivo '{target_col}' no es numérica.")
        return None

    if not isinstance(columns, list):
        print("Error: El argumento 'columns' debe ser una lista de strings.")
        return None

    if len(columns) == 0:
        # Si la lista está vacía, seleccionamos las columnas categóricas del DataFrame
        columns = df.select_dtypes(include=['object', 'category']).columns.tolist()

    if not isinstance(pvalue, float) or not (0 < pvalue < 1):
        print("Error: El valor de 'pvalue' debe ser un float entre 0 y 1.")
        return None

    columnas_significativas = []

    # Iterar sobre las columnas para analizar la relación con la variable 'target_col'
    for col in columns:
        if col not in df.columns:
            print(f"Advertencia: La columna '{col}' no existe en el DataFrame. Se omite.")
            continue

        # Eliminar los valores nulos de los datos
        df_clean = df[[col, target_col]].dropna()

        categorias = df_clean[col].unique()

        if len(categorias) < 2:
            print(f"Advertencia: La columna '{col}' tiene menos de 2 categorías, se omite.")
            continue

        # Si la columna tiene dos categorías, aplicamos la prueba t de Student
        if len(categorias) == 2:
            grupo1 = df_clean[df_clean[col] == categorias[0]][target_col]
            grupo2 = df_clean[df_clean[col] == categorias[1]][target_col]
            stat, p = ttest_ind(grupo1, grupo2, equal_var=False) # Prueba t para dos grupos independientes
            print(p)

        # Si la columna tiene más de dos categorías, aplicamos ANOVA
        else:
            grupos = [df_clean[df_clean[col] == cat][target_col] for cat in categorias]
            stat, p = f_oneway(*grupos)

        # Si la relación es significativa, añadimos la columna a la lista
        if p < pvalue:
            columnas_significativas.append(col)

        # Si se debe plotear el histograma
        if with_individual_plot:
            plt.figure(figsize=(10, 6))
            sns.histplot(data=df_clean, x=target_col, hue=col, multiple="stack", kde=False)
            plt.title(f"Histograma de '{target_col}' agrupado por '{col}'")
            plt.show()

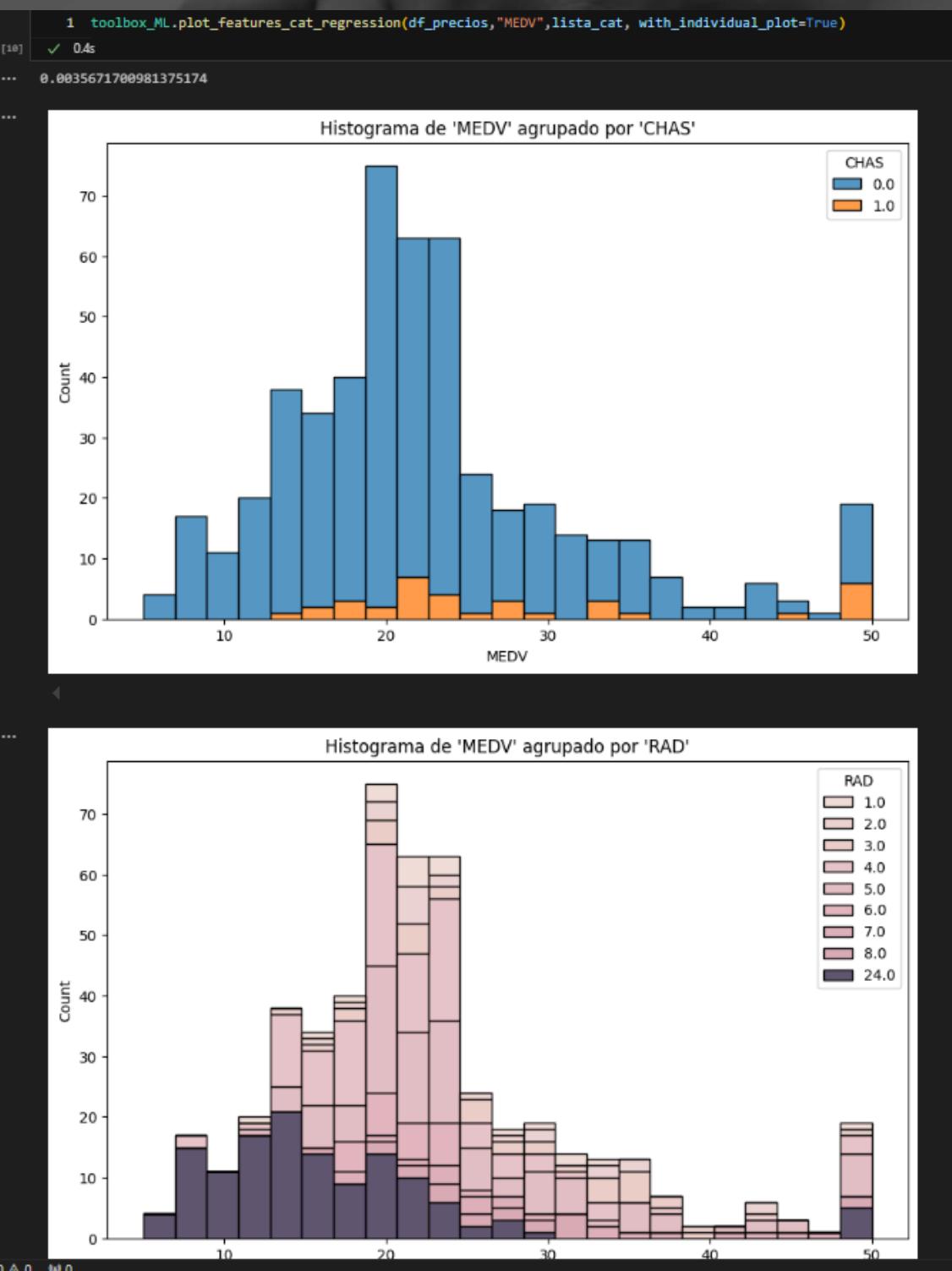
    # Retornar las columnas que tienen una relación significativa con 'target_col'
    return columnas_significativas

```

def plot_features_cat_regression(df, target_col, columns, pvalue=0.05, with_individual_plot=False):

Se analiza las variables categóricas o numéricas. Si el test estadístico de relación entre ellas es significativo:

- Se pinta histogramas agrupados para la variable 'target_col' en función de las variables de 'columns',



**Gracias por
su atención**

