Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 2. INGESTA DE DATOS.

2. 5 Análisis y Visualización de los Datos

ANÁLISIS DE VARIABLES

El análisis o exploración de los datos permite entender mejor la información contenida en cada variable y detectar posibles errores antes de emplear los datos para realizar cálculos o entrenar un modelo.

Cuando tenemos datos contenidos en una estructura *DataFrame*, podemos considerar que cada columna es una variable de un tipo determinado. La estructura *DataFrame* ofrece varios métodos (comandos) para realizar el análisis de cada una de las variables que contiene.

El método *info* muestra información relativa a cada variable, o lo que es lo mismo, a cada columna, como el nombre de la columna, el tipo de dato, y el número de valores distintos de NaN que contiene. NaN son las siglas de *Not a Number*, y es la forma de tipificar los valores vacíos (celdas vacías de una tabla) o desconocidos y que no son computables. Por lo tanto, el método *info* nos permite comprobar que los datos de cada columna se hayan almacenado con el tipo correcto y así evitar situaciones como por ejemplo en la que los datos numéricos son reconocidos como texto o viceversa.

El método *info*, además, proporciona información general sobre la tabla como el número de filas (*entries*) y columnas y la cantidad de memoria empleada. El método *shape* también nos da información de las dimensiones del *DataFrame*, devuelve el número de filas y de columnas.

Por otro lado, el método *describe* muestra los primeros estadísticos de cada variable/columna, permitiendo así su análisis. El análisis estadístico sólo es realizado para las columnas que contengan datos de tipo numérico, y las métricas mostradas son el número de valores empleados en el análisis (*count*), la media (*mean*), la desviación típica (*std*), el mínimo (*min*) y máximo (*max*) y los cuartiles (el cuartil 50% es la

mediana). La Figura 1 muestra el resultado de emplear los métodos *info, shape* y *describe*.

```
In [5]: cosecha_finca1= pd.DataFrame([[300, 400], [100, 500], [400, 200], [600, 100]],
                                     columns=('trigo', 'maiz'), index=[2018, 2017, 2021, 2022])
        cosecha_finca1
```

Out[5]:

	trigo	maiz
2018	300	400
2017	100	500
2021	400	200
2022	600	100

```
In [6]: cosecha_finca1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4 entries, 2018 to 2022
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   trigo   4 non-null      int64
 1   maiz    4 non-null      int64
dtypes: int64(2)
memory usage: 96.0 bytes
```

```
In [8]: cosecha_finca1.shape
```

```
Out[8]: (4, 2)
```

```
In [9]: cosecha_finca1.describe()
```

Out[9]:

	trigo	maiz
count	4.0000	4.000000
mean	350.0000	300.000000
std	208.1666	182.574186
min	100.0000	100.000000
25%	250.0000	175.000000
50%	350.0000	300.000000
75%	450.0000	425.000000
max	600.0000	500.000000

Figura 1. Métodos de Pandas para el análisis de cada variable de un DataFrame.

VISUALIZACIÓN

La librería Pandas junto con la librería Matplotlib proporcionan los métodos necesarios para generar los gráficos más comunes en estadística descriptiva, como los diagramas de barras (*bar*) y de tartas (*pie*).

En el ejemplo de la Figura 2 se ha empleado la base de datos de pedidos del archivo "Superstore_Dataset.xlsx". Se ha generado un nuevo *DataFrame* llamado *ventas* que contine el número de artículos (columna *Quantity*) de cada producto vendido en los cinco primeros pedidos. Se ha añadido el tipo de producto (columna *Sub-Category*) como índice de la nueva tabla. Finalmente, se ha representado una única figura con dos gráficos (ax1 y ax2), mediante el método *subplot*.

```
In [44]: ventas = orders['Quantity'][0:5]
         ventas.index=orders['Sub-Category'][0:5]
         ventas

Out[44]: Sub-Category
         Bookcases    2
         Chairs       3
         Labels       2
         Paper        3
         Binders      3
         Name: Quantity, dtype: int64

In [45]: import matplotlib.pyplot as plt
         figura, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(9,3))
         ventas.plot(ax=ax1, kind='bar', title='Nº de patas');
         ventas.plot(ax=ax2, kind='pie', title='Clase');
```

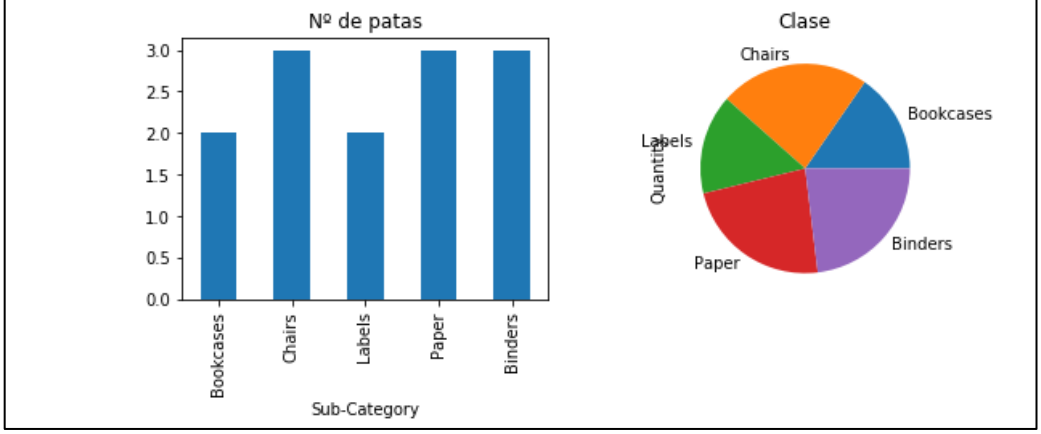

Figura 2. Diagrama de barras y de tartas a partir de los datos de un DataFrame

El método *plot* de los *DataFrames* permite representar múltiples tipos de diagramas, dando distintos valores al argumento *kind.* Entre los diagramas más empleados encontramos los de barras (*bar*) y tarta (*pie*), mostrados en el ejemplo anterior y el de líneas (*line*), que es el representado por defecto, el histograma (*hist*), el diagrama de caja (*box*), o el de dispersión (*scatter*), entre otros. Puede encontrarse más documentación disponible en https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.html .

ANÁLISIS MULTI-VARIABLE Y VISUALIZACIÓN

Además del análisis de cada una de las variables contenidas en un *DataFrame*, Pandas permite estudiar las relaciones entre ellas, o lo que es lo mismo, permite realizar un análisis multi-variable. Este tipo de análisis ayuda a obtener una visión inicial de qué variables son más o menos descriptivas y por lo tanto más o menos adecuadas para ser empleadas como datos de entrada de un modelo predictivo de Machine Learning.

Dos de los estadísticos más empleados para analizar la relación lineal entre dos variables son la covarianza y la correlación.

- La *covarianza* indica como los cambios en una variable provocan cambios en la otra. Su signo indica si ambas variables varían en el mismo sentido (covarianza positiva) o en sentido opuesto (covarianza negativa). Si la covarianza es cercana a cero o cero, no se puede definir si los cambios en una variable se relacionan con algún tipo de cambio en la otra.
- La *correlación* nos indica la magnitud de la relación o similitud entre las variables. Cuanto mayor sea su valor absoluto, mayor es la relación entre las variables.

```
In [104]: tabla = pd.DataFrame([('Enero', 31, 'inv', 2, 4),('Febrero', 28, 'inv', 3, 2),('Marzo', 31, 'inv', 7, 5),
                               ('Abril', 30, 'pri', 7, 9),('Mayo', 31, 'pri', 9, 10),('Junio', 30, 'pri', 15, 14),
                               ('Julio', 31, 'ver', 20, 24),('Agosto', 31, 'ver', 27, 26),('Septiembre', 30, 'ver', 25, 18),
                               ('Octubre', 31, 'oto', 20, 14),('Noviembre', 30, 'oto', 11, 10),('Diciembre', 31, 'oto', 6, 7),],
                              columns=['Mes', 'Días', 'Estación', 'Temp.2021', 'Temp.2022'],
                              index=['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'])
          tabla
```

Out[104]:

	Mes	Días	Estación	Temp.2021	Temp.2022
Ene	Enero	31	inv	2	4
Feb	Febrero	28	inv	3	2
Mar	Marzo	31	inv	7	5
Abr	Abril	30	pri	7	9
May	Mayo	31	pri	9	10
Jun	Junio	30	pri	15	14
Jul	Julio	31	ver	20	24
Ago	Agosto	31	ver	27	26
Sep	Septiembre	30	ver	25	18
Oct	Octubre	31	oto	20	14
Nov	Noviembre	30	oto	11	10
Dic	Diciembre	31	oto	6	7

```
In [108]: tabla.corr()
```

Out[108]:

	Días	Temp.2021	Temp.2022
Días	1.000000	0.256108	0.349989
Temp.2021	0.256108	1.000000	0.928141
Temp.2022	0.349989	0.928141	1.000000

Figura 3. Cálculo de la matriz de correlación de las variables de un DataFrame

El cálculo de la covarianza y la correlación de todas las variables con respecto a cada una de ellas permite obtener la matriz de covarianza y la matriz de correlación,

respectivamente. La matriz de covarianza para todas las variables numéricas de un *DataFrame*, se obtiene con el método *cov*, y la matriz de correlación, con el método *corr* (véase un ejemplo en la Figura 3).

La librería de Python *Seaborn* permite representar la matriz de correlación mediante un mapa de color, con el método *heatmap*, tal y como se muestra en la Figura 4. Además, una vez identificados los pares de variables correlacionadas, es posible representar su gráfico de dispersión (*scatter*) gracias a la librería *Matplotlib*. El gráfico de dispersión de dos variables es la representación de una nube de puntos cuyas coordenadas son dichas variables.

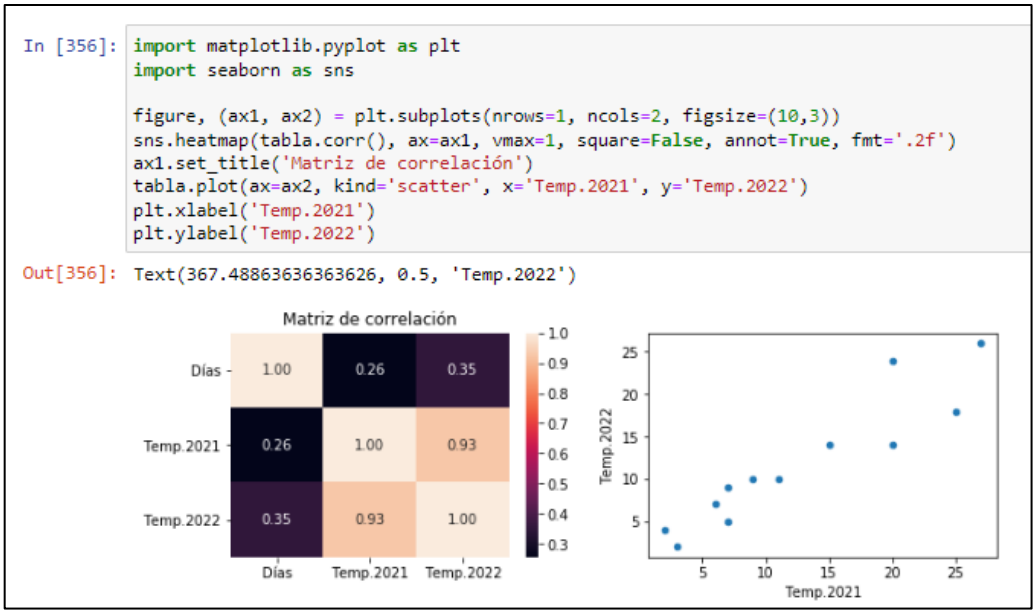

Figura 4. Representación de la matriz de correlación con un mapa de color y el gráfico de dispersión de dos variables correlacionadas.

ANÁLISIS DE LA VARIABLE A PREDECIR

El objetivo de entrenar un modelo de Machine Learning es que dicho modelo finalmente sea capaz de predecir el valor de una determinada variable. A esta variable se le llama *salida, respuesta* o *predicción* del modelo o también, variable *target*, y en las ecuaciones es simbolizada con la letra *y*.

Antes de entrenar el modelo es conveniente analizar la distribución de la variable de salida y su relación con el resto de las variables de entrada. Este análisis arrojará información útil que permitirá tomar decisiones en cuanto a la elección del modelo y los métodos empleados para preparar su entrenamiento.

En la figura 5 se muestra la cabecera de un DataFrame que contiene información sobre características de un listado de viviendas y además se incluye su precio de venta. Estos datos se han cargado desde un archivo csv, 'HouseDataset.csv'.

```
In [55]: viviendas = pd.read_csv('HousesDataset.csv', sep=",")

         # Renombramos las columnas
         viviendas.columns = ["precio", "metros_totales", "antiguedad", "precio_terreno", "metros_habitables",
                              "universitarios", "dormitorios", "chimenea", "banyos", "habitaciones", "calefaccion",
                              "consumo_calefacion", "desague", "vistas_lago", "nueva_construccion", "aire_acondicionado"]
         datos.head(3)
```

Out[55]:

	precio	metros_totales	antiguedad	precio_terreno	metros_habitables	universitarios	dormitorios	chimenea	banyos	habitaciones
0	132500	0.09	42	50000	906	35	2	1	1.0	5
1	181115	0.92	0	22300	1953	51	3	0	2.5	6
2	109000	0.19	133	7300	1944	51	4	1	1.0	8

Figura 5. Creación de un DataFrame con características de viviendas a partir de un archivo csv.

```
In [56]: fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(6, 6))

         sns.distplot( viviendas.precio, hist = False, rug = True, color = "blue",
                       kde_kws = {'shade': True, 'linewidth': 1}, ax = axes[0])
         axes[0].set_title("Distribución original", fontsize = 'medium')
         axes[0].set_xlabel('precio', fontsize='small')
         axes[0].tick_params(labelsize = 6)

         sns.distplot( np.sqrt(viviendas.precio), hist = False, rug = True, color = "blue",
                       kde_kws = {'shade': True, 'linewidth': 1}, ax = axes[1])
         axes[1].set_title("Transformación raíz cuadrada", fontsize = 'medium')
         axes[1].set_xlabel('sqrt(precio)', fontsize='small')
         axes[1].tick_params(labelsize = 6)

         sns.distplot( np.log(viviendas.precio), hist = False, rug = True, color = "blue",
                       kde_kws = {'shade': True, 'linewidth': 1}, ax = axes[2])
         axes[2].set_title("Transformación logarítmica", fontsize = 'medium')
         axes[2].set_xlabel('log(precio)', fontsize='small')
         axes[2].tick_params(labelsize = 6)

         fig.tight_layout()
```

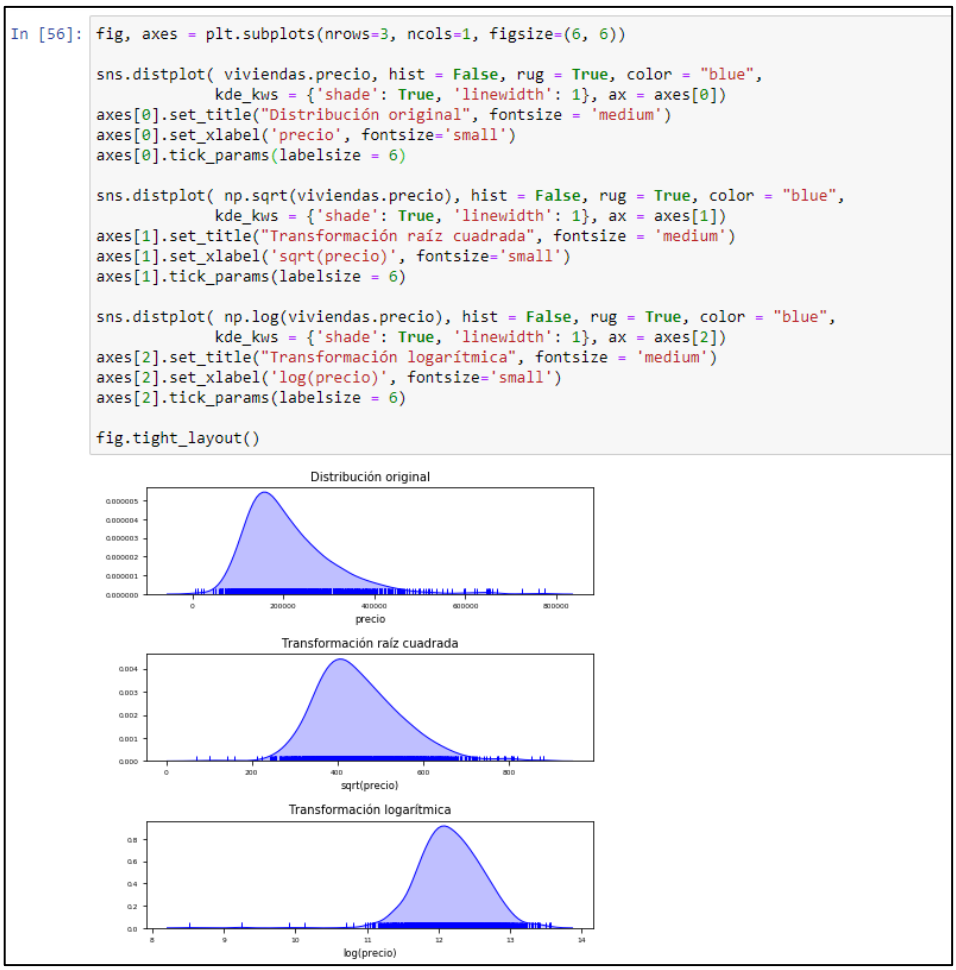

Figura 6. Representación de la distribución de los datos de precio de las viviendas.

Supongamos que queremos entrenar un modelo que sea capaz de predecir el precio de venta de una vivienda en función de sus características y vamos a usar los datos del csv anterior para entrenar el modelo. La variable de salida será el precio de las viviendas y como primer paso analizaremos la distribución de sus valores. La figura 6 muestra un ejemplo en el que se representa la distribución de los valores de la columna "precio", y además se muestran dos distribuciones más, las resultantes de calcular la raíz cuadrada de los datos, y el logaritmo. Este análisis es interesante porque existen algunos métodos de Machine Learning que sólo funcionan, o funcionan mejor, para variables de salida con distribución Gaussiana o normal. En ocasiones puede ocurrir que la distribución de los datos no sea de tipo Gaussiana, pero sí la de la raíz o la del logaritmo de los datos.

Además del análisis visual que ofrecen las gráficas anteriores, la librería *fitter* de Python permite identificar la distribución que mejor se ajusta a nuestros datos. La Figura 7 muestra los resultados obtenidos para el ajuste de los datos de la columna 'precio' con nueve tipos de distribuciones estadísticas distintas. Los resultados se muestran ordenados en función del error total de cada ajuste. Por lo tanto, la primera distribución es la que mejor se ajusta a los datos.

```
In [58]: from fitter import Fitter, get_common_distributions
         distribuciones = ['cauchy', 'chi2', 'expon',  'gamma', 'norm', 'beta', 'logistic']

         fitter = Fitter(viviendas.precio, distributions=distribuciones)
         fitter.fit()
         fitter.summary(Nbest=10, plot=False)

         Fitting 7 distributions: 100%|██████████| 7/7 [00:00<00:00, 12.81it/s]
```

Out[58]:

	sumsquare_error	aic	bic	kl_div	ks_statistic	ks_pvalue
beta	2.497420e-11	3068.852573	-55037.908642	inf	0.051211	2.229230e-04
logistic	4.913880e-11	3147.967019	-53883.297831	inf	0.071310	4.368612e-08
cauchy	5.221450e-11	2956.669693	-53778.388707	inf	0.121728	9.001774e-23
chi2	5.776892e-11	3321.818880	-53596.249282	inf	0.093520	1.326966e-13
norm	6.947514e-11	3324.534158	-53284.856663	inf	0.104149	8.972690e-17
expon	2.915346e-10	2824.103160	-50806.577128	inf	0.316530	2.028906e-154
gamma	4.841645e-10	inf	-49922.566370	3.958212	0.947917	0.000000e+00

Figura 7. Ajuste de los datos de precio de las viviendas con varios tipos de distribuciones.

Además, como se ha mencionado anteriormente, una práctica muy útil es la exploración de la relación entre la variable de salida y el resto de las variables del *DataFrame*. Esto permite extraer la información necesaria para realizar la selección de las características que serán empleadas como datos de entrada del modelo que queremos entrenar.

La librería *Seaborn* con la función *regplot* ofrece la representación del diagrama de dispersión entre dos variables, acompañado por la visualización del ajuste de una regresión lineal entre esas dos variables. La Figura 8 muestra el diagrama de dispersión y la regresión lineal entre cada variable numérica y la variable de salida *precio*.

```
In [63]: import matplotlib.ticker as ticker

fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(9, 5))
axes = axes.flat
columnas_numeric = viviendas.select_dtypes(include=['float64', 'int64']).columns
columnas_numeric = columnas_numeric.drop('precio')

for i, colum in enumerate(columnas_numeric):
    sns.regplot( x = viviendas[colum], y = viviendas['precio'], color = "green", marker = '.',
        scatter_kws = {"alpha":0.4}, line_kws = {"color":"r","alpha":0.7}, ax = axes[i])
    axes[i].set_title(f"precio vs {colum}", fontsize = 7, fontweight = "bold")
    axes[i].yaxis.set_major_formatter(ticker.EngFormatter())
    axes[i].xaxis.set_major_formatter(ticker.EngFormatter())
    axes[i].tick_params(labelsize = 6)
    axes[i].set_xlabel("")
    axes[i].set_ylabel("")

fig.tight_layout()
plt.subplots_adjust(top=0.9)
fig.suptitle('Correlación con precio', fontsize = 10, fontweight = "bold");
```

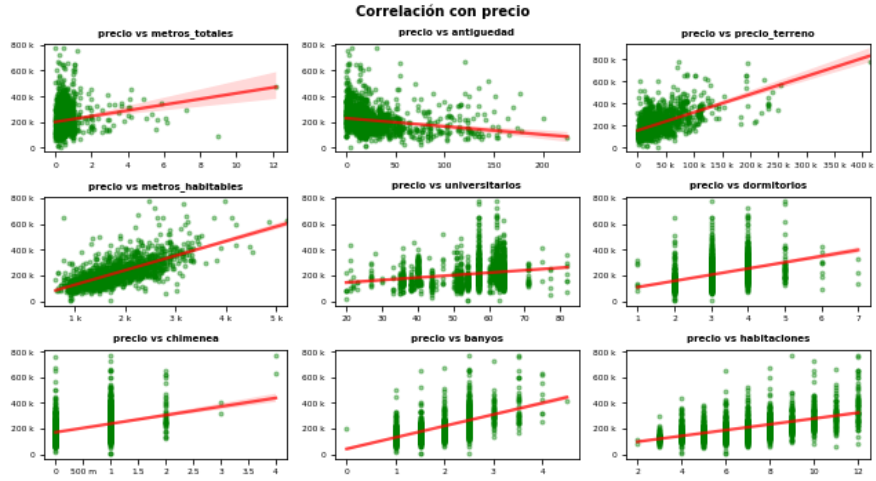

Figura 8. Relación entre la variable de salida y el resto de las variables numéricas.

Además de la exploración visual realizada sobre las gráficas anteriores, es posible realizar un análisis cuantitativo de la relación entre la variable de salida y el resto de las variables numéricas gracias al cálculo y la representación de la matriz de correlación, como muestra la Figura 9.

Las variables poco correlacionas con la variable de salida pueden no funcionar bien como descriptores de la situación y por lo tanto colaborar poco en la tarea de predicción del modelo. Sin embargo, las variables altamente correlacionadas con la de salida en ocasiones también pueden perjudicar el entrenamiento ya que hacen que el algoritmo de aprendizaje aparte el foco de la información que ofrecen el resto de las variables, perdiendo así capacidad descriptiva.

```
In [67]: fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(6, 6))

sns.heatmap( viviendas.corr(), annot = True, cbar = False, annot_kws = {"size": 8},
    vmin = -1, vmax = 1, center = 0, cmap = sns.diverging_palette(20, 220, n=200),
    square = True, ax = ax )
ax.set_xticklabels( ax.get_xticklabels(), rotation = 45, horizontalalignment = 'right',)
ax.tick_params(labelsize = 8)
```

Figura 9. Representación de la matriz de correlación entre las columnas de un DataFrame

ANÁLISIS DE VARIABLES CATEGÓRICAS

Consideramos como variables categóricas aquellas que, en lugar de tomar un valor numérico, toman como valor una etiqueta o clase entre varias opciones disponibles. Las columnas de este tipo normalmente son designadas en *Pandas* como de tipo *objeto*.

Es posible emplear el método describe sobre las columnas de este tipo, como muestra la Figura 10. La información que se obtiene es el número de registros en cada columna (*count*), el número de valores distintos que toma (*unique*), el valor que más se repite (*top*) y el número de veces que se repite (*freq*).

```
In [68]: viviendas.select_dtypes(include=['object']).describe()
```

Out[68]:

	calefaccion	consumo_calefacion	desague	vistas_lago	nueva_construccion	aire_acondicionado
count	1728	1728	1728	1728	1728	1728
unique	3	3	3	2	2	2
top	hot air	gas	public/commercial	No	No	No
freq	1121	1197	1213	1713	1647	1093

Figura 10. Método describe aplicado sobre columnas de tipo 'object'.

Además, también es posible visualizar la distribución de este tipo de variables mediante la representación del número de apariciones de cada valor posible (realizado con el método *counts()*), utilizando diagramas de barras (con el método *plot.barh*), como se muestra en el ejemplo de la Figura 11.

```
In [71]: fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(9, 5))
         axes = axes.flat
         columnas_object = viviendas.select_dtypes(include=['object']).columns

         for i, colum in enumerate(columnas_object):
             datos[colum].value_counts().plot.barh(ax = axes[i])
             axes[i].set_title(colum, fontsize = 8, fontweight = "bold")
             axes[i].tick_params(labelsize = 8)
             axes[i].set_xlabel("")

         fig.tight_layout()
         plt.subplots_adjust(top=0.9)
```

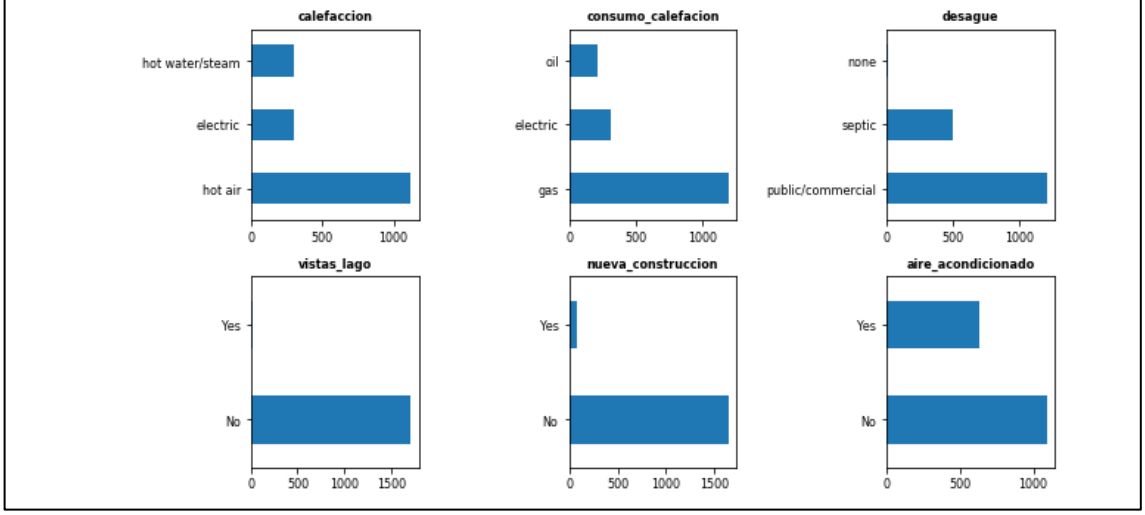

Figura 11. Representación de la distribución de valores en variables categóricas

También es posible analizar la relación de la variable de salida con las variables categóricas. Para ello, se emplea un gráfico de tipo violín que relacione cada variable categórica con la de salida, como se muestra en el ejemplo de la Figura 12, donde la variable de salida es *precio*. Este tipo de gráfico muestra la distribución de una variable numérica para cada una de las clases de una variable categórica.

```
In [84]: fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(9, 5))
         axes = axes.flat
         columnas_object = viviendas.select_dtypes(include=['object']).columns

         for i, colum in enumerate(columnas_object):
             sns.violinplot( x = colum, y = 'precio', data = datos, color = "white", ax = axes[i])
             axes[i].set_title(f"precio vs {colum}", fontsize = 8, fontweight = "bold")
             axes[i].yaxis.set_major_formatter(ticker.EngFormatter())
             axes[i].tick_params(labelsize = 8)
             axes[i].set_xlabel("")
             axes[i].set_ylabel("")

         fig.tight_layout()
```

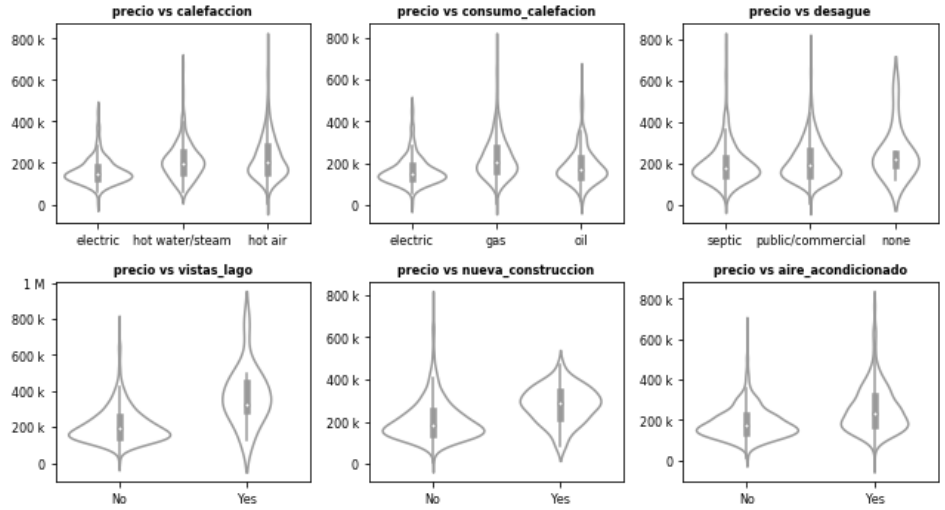

Figura 12. Relación entre cada variable categórica de un DataFrame y la variable numérica de salida.

REFERENCIAS

https://docs.python.org/3/tutorial/index.html

https://www.anaconda.com/products/individual

https://docs.jupyter.org/en/latest/

https://numpy.org/doc/stable/

https://pandas.pydata.org/docs

https://matplotlib.org/stable/index.html

Machine learning con Python y Scikit-learn by Joaquín Amat Rodrigo, available under a Attribution 4.0 International (CC BY 4.0) at https://www.cienciadedatos.net/documentos/py06_machine_learning_python_scikitlearn.html

https://www.drivendata.org/