Autor: María José
Gómez Silva

CAPÍTULO 3. PREPROCESADO

3. 3. Cambios de Formato, Normalización y Agrupación de los Datos

CAMBIOS DE FORMATO

En ocasiones es necesario cambiar el formato de los datos, para que sean mejor entendidos tanto por los analistas como por la máquina.

```
In [8]: def mayusculas(cadena):
            return cadena.upper()

        orders=pd.read_excel('Superstore_Dataset.xlsx', converters={'Customer Name':mayusculas})
        orders.head(3)
```

Out[8]:

	Row ID+O6G3A1:R6	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Region	Product ID	Category
0	1	CA-2019-152156	2019-11-08	2019-11-11	Second Class	CG-12520	CLAIRE GUTE	Consumer	United States	Henderson	Kentucky	South	FUR-BO-10001798	Furniture
1	2	CA-2019-152156	2019-11-08	2019-11-11	Second Class	CG-12520	CLAIRE GUTE	Consumer	United States	Henderson	Kentucky	South	FUR-CH-10000454	Furniture
2	3	CA-2019-138688	2019-06-12	2019-06-16	Second Class	DV-13045	DARRIN VAN HUFF	Corporate	United States	Los Angeles	California	West	OFF-LA-10000240	Office Supplies

```
In [7]: def titulo(cadena):
            return cadena.title()

        orders['Customer Name']=orders['Customer Name'].apply(titulo)
        orders.head(3)
```

Out[7]:

	Row ID+O6G3A1:R6	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Region	Product ID	Category
0	1	CA-2019-152156	2019-11-08	2019-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	South	FUR-BO-10001798	Furniture
1	2	CA-2019-152156	2019-11-08	2019-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	South	FUR-CH-10000454	Furniture
2	3	CA-2019-138688	2019-06-12	2019-06-16	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	West	OFF-LA-10000240	Office Supplies

Figura 1. Conversión de formato de una columna de tipo cadena de caracteres

En la Figura 1 se muestra un ejemplo en el que se aplican dos conversiones de formato a los datos de la columna "*Customer Name*" de la hoja de cálculo "*Superstore_Dataset.xlsx*". Se emplea una función para pasar a mayúsculas todos los nombres guardados en la columna y se aplica esa función en el momento de su lectura como *DataFrame*. Posteriormente se vuelve a aplicar otra función de conversión para guardar los nombres en formato *Título*.

```
In [56]: orders['Order Date'] = orders['Order Date'].astype(str)
         orders.dtypes

Out[56]: Row ID+O6G3A1:R6             int64
         Order ID                    object
         Order Date                  object
         Ship Date           datetime64[ns]
         Ship Mode                   object
         Customer ID                 object
         Customer Name               object
         Segment                     object
         Country                     object
         City                        object
         State                       object
         Region                      object
         Product ID                  object
         Category                    object
         Sub-Category                object
         Product Name                object
         Sales                      float64
         Quantity                     int64
         Profit                     float64
         dtype: object

In [57]: orders['Order Date'] = pd.to_datetime(orders['Order Date'], format='%Y-%m-%d')
         orders.head(2)

Out[57]:
```

	Row ID+O6G3A1:R6	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Region	Product ID	Category
0	1	CA-2019-152156	2019-11-08	2019-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	South	FUR-BO-10001798	Furniture
1	2	CA-2019-152156	2019-11-08	2019-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	South	FUR-CH-10000454	Furniture

```
In [58]: orders.dtypes

Out[58]: Row ID+O6G3A1:R6             int64
         Order ID                    object
         Order Date          datetime64[ns]
         Ship Date           datetime64[ns]
         Ship Mode                   object
         Customer ID                 object
         Customer Name               object
         Segment                     object
         Country                     object
         City                        object
         State                       object
         Region                      object
         Product ID                  object
         Category                    object
         Sub-Category                object
         Product Name                object
         Sales                      float64
         Quantity                     int64
         Profit                     float64
         dtype: object
```

Figura 2. Conversión a tipo datatime de una columna de un DataFrame

Cuando los datos incorporan columnas con información de fechas, resulta muy útil que tales columnas sean de tipo *objeto datetime*, ya que esto permite realizar una ordenación

cronológica. El *DataFrame* del ejemplo anterior contenía dos columnas con información de fechas "*Orders Date*" y "*Ship Date*". Supongamos que por algún motivo la información en "*Orders Date*" se ha almacenado como cadenas de caracteres, en lugar de como objetos *datetime*. Para simular tal situación, en la Figura 2 se fuerza a que esa columna esté formada por cadenas de caracteres (str). Para recuperar su formato *datetime*, se puede emplear la función de Pandas *pd.to_datetime*, indicando el formato de fecha o fecha y hora apropiado. Una vez que la columna es de tipo *datetime*, ésta puede emplearse para ordenar cronológicamente el *DataFrame*, como en la Figura 3.

```
In [59]: orders.sort_values(by='Order Date')
```

Out[59]:

	Row ID+O6G3A1:R6	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Region
2861	4919	CA-2019-160304	2019-01-02	2019-01-07	Standard Class	BM-11575	Brendan Murry	Corporate	United States	Gaithersburg	Maryland	East
2862	4920	CA-2019-160304	2019-01-02	2019-01-07	Standard Class	BM-11575	Brendan Murry	Corporate	United States	Gaithersburg	Maryland	East
5608	9495	CA-2019-105207	2019-01-03	2019-01-08	Standard Class	BO-11350	Bill Overfelt	Corporate	United States	Broken Arrow	Oklahoma	Central

Figura 3. Ordenación cronológica de un DataFrame

NORMALIZACIÓN DE LOS DATOS

Especialmente en el caso de variables numéricas, la aplicación de algoritmos de Machine Learning, no sólo requiere que los datos aparezcan en un determinado formato, sino también el escalado de sus valores. Si los datos tienen campos numéricos con distintas escalas, como por ejemplo datos en una columna con valores de 0 a 1, y en otra de 100 a 1000, entonces es necesario convertirlos a una misma escala común. Ese proceso es denominado *normalización*.

La representación de las distribuciones estadísticas de los datos de cada columna de tipo numérico de un DataFrame nos permitirá observar los rangos de valores que presenta cada variable, como muestra el ejemplo de la Figura 4.

```
In [45]: viviendas = pd.read_csv('HousesDataset.csv', sep=",")
         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"]

         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.distplot(datos[colum], hist= True, rug = True,
                          color = (list(plt.rcParams['axes.prop_cycle'])*2)[i]["color"],
                          kde_kws = {'shade': True, 'linewidth': 1}, ax  = axes[i] )
             axes[i].set_title(colum, fontsize = 7, fontweight = "bold")
             axes[i].tick_params(labelsize = 6)
             axes[i].set_xlabel("")

         fig.tight_layout()
         plt.subplots_adjust(top = 0.9)
         fig.suptitle('Distribución variables numéricas', fontsize = 10, fontweight = "bold");
```

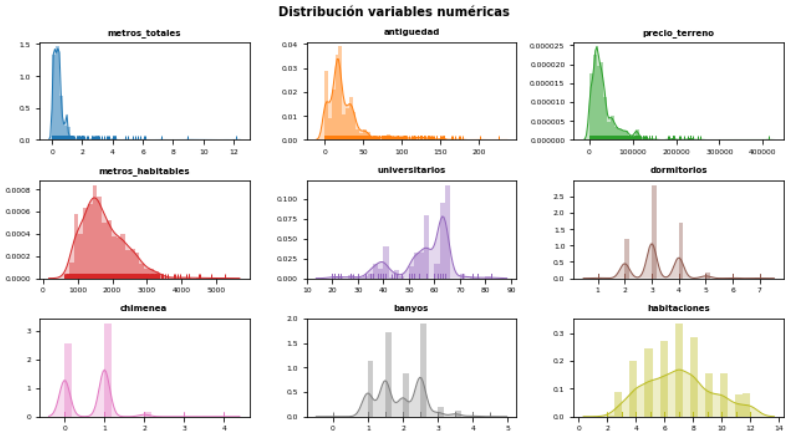

Figura 4. Representación de las distribuciones estadísticas de varias variables numéricas.

```
In [53]: viviendas.chimenea.value_counts()

Out[53]: 1    942
         0    740
         2     42
         4      2
         3      2
         Name: chimenea, dtype: int64

In [54]: viviendas.chimenea = viviendas.chimenea.astype("str")
         viviendas.chimenea

Out[54]: 0       1
         1       0
         2       1
         3       1
         4       0
                ..
         1723    1
         1724    1
         1725    0
         1726    1
         1727    0
         Name: chimenea, Length: 1728, dtype: object
```

Figura 5. Cambio de formato de la columna 'chimenea'.

Cuando existen variables que presentan un rango de posibles valores discreto y muy reducido, como es el caso de la columna *chimenea* (véase la figura 4), suele ser conveniente transformar la variable en cuestión de numérica a categórica. Esto se consigue modificando el formato o tipo de los datos de numérico a cadena de caracteres (*str*), como muestra la Figura 5. En Pandas, el tipo *object* se refiere a datos de tipo *string* (cadenas de caracteres).

La normalización o escalado de las variables se consigue aplicando transformaciones aritméticas para comprender sus valores entre unos valores mínimos y máximos comunes. De este modo se homogenizan los datos, todas las variables se mueven en unos mismos rangos de valores y eso favorece el aprendizaje automático, ya que facilita que el peso de cada variable dependa de su relevancia para generar la predicción del modelo, y no de la escala de sus valores.

Existen dos formas principales de normalizar los datos:

- La estandarización. Para cada valor de una variable le resta la media y lo divide entre la desviación típica de la variable. De esta forma, la variable pasa a tener una distribución normal. Se puede realizar con la función *StandardScaler* del módulo *preprocessing* de *Scikit-learn*.

$$z = \frac{x - \mu}{\sigma}$$

- El escalado. Para cada valor de una variable le aplica la siguiente fórmula. De esta forma, la variable pasa a tener un valor mínimo de cero y un valor máximo de uno. Se puede realizar con la función *MinMaxScaler* del módulo *preprocessing* de *Scikit-learn*.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Por su parte, el centrado consiste en únicamente restar a cada valor el valor de la media de la variable en cuestión. Se puede realizar con la función *StandardScaler* del módulo *preprocessing* de *Scikit-learn*, poniendo el argumento *with_std* a *False*.

BINARIZACIÓN DE LAS VARIABLES CATEGÓRICAS

La mayoría de los algoritmos de Machine Learning requieren que las variables categóricas sean binarizas. Para ello, se emplea el formato conocido como one-hot-encoding. Se trata de indicar con valores binarios (0 o 1) si la variable toma un determinado valor o no. Por ejemplo, si tenemos una columna de un dataframe que toma valores de colores de un deteminado producto y los únicos valores que puede tomar son 'rojo', 'verde' y 'azul', su binarización implica la creación de tres nuevas columnas que sustituirán a la columna original. Será necesaria una columna por cada valor de la variable, en este caso una columna para cada color. En cada fila se indicará el color que tomaba la variable original, colocando un 1 en la columna que corresponda y 0 en las demás.

```
In [163]: from sklearn.compose import ColumnTransformer
          from sklearn.preprocessing import OneHotEncoder
          from sklearn.preprocessing import StandardScaler
          from sklearn.compose import make_column_selector
          numeric_cols = viviendas.select_dtypes(include=['float64', 'int64']).columns.to_list()
          cat_cols = viviendas.select_dtypes(include=['object', 'category']).columns.to_list()
          preprocessor = ColumnTransformer( [('scale', StandardScaler(), numeric_cols),
                                             ('onehot', OneHotEncoder(handle_unknown='ignore'), cat_cols)],
                                              remainder='passthrough')
          viviendas_prep = preprocessor.fit_transform(viviendas)
          encoded_cat = preprocessor.named_transformers_['onehot'].get_feature_names(cat_cols)
          labels = np.concatenate([numeric_cols, encoded_cat])
          datos_viviendas_prep = preprocessor.transform(viviendas)
          datos_viviendas_prep = pd.DataFrame(datos_viviendas_prep, columns=labels)
          datos_viviendas_prep.info()

          <class 'pandas.core.frame.DataFrame'>
          RangeIndex: 1728 entries, 0 to 1727
          Data columns (total 27 columns):
           #   Column                        Non-Null Count  Dtype
          ---  ------                        --------------  -----
           0   precio                        1728 non-null   float64
           1   metros_totales                1728 non-null   float64
           2   antiguedad                    1728 non-null   float64
           3   precio_terreno                1728 non-null   float64
           4   metros_habitables             1728 non-null   float64
           5   universitarios                1728 non-null   float64
           6   dormitorios                   1728 non-null   float64
           7   banyos                        1728 non-null   float64
           8   habitaciones                  1728 non-null   float64
           9   chimenea_0                    1728 non-null   float64
           10  chimenea_1                    1728 non-null   float64
           11  chimenea_2_mas                1728 non-null   float64
           12  calefaccion_electric          1728 non-null   float64
           13  calefaccion_hot air           1728 non-null   float64
           14  calefaccion_hot water/steam   1728 non-null   float64
           15  consumo_calefacion_electric   1728 non-null   float64
           16  consumo_calefacion_gas        1728 non-null   float64
           17  consumo_calefacion_oil        1728 non-null   float64
           18  desague_none                  1728 non-null   float64
           19  desague_public/commercial     1728 non-null   float64
           20  desague_septic                1728 non-null   float64
           21  vistas_lago_No                1728 non-null   float64
           22  vistas_lago_Yes               1728 non-null   float64
           23  nueva_construccion_No         1728 non-null   float64
           24  nueva_construccion_Yes        1728 non-null   float64
           25  aire_acondicionado_No         1728 non-null   float64
           26  aire_acondicionado_Yes        1728 non-null   float64
          dtypes: float64(27)
          memory usage: 364.6 KB
```

Figura 6. Binarización de variables categóricas y estandarización de variables numéricas.

En el ejemplo de la Figura 6 se muestra como la librería *Scikit-learn* permite binarizar todas las variables categóricas de un *DataFrame*, empleando el método *ColumnTransformer* y la clase *OneHotEncoder*. En este ejemplo también se emplea *ColumnTransformer* para estandarizar las variables numéricas.

Por defecto, la clase *OneHotEncoder* binariza todas las variables, por lo que hay que aplicarlo únicamente a las variables categóricas. Con el argumento *drop='first'* se elimina una de las nuevas columnas generadas para evitar redundancias. En el ejemplo anterior de las columnas para representar tres colores, no son necesarias las tres columnas, ya que con dos de ellas tendremos toda la información necesaria. Si las dos están a cero, por descarte el color es el de la columna eliminada.

En ciertos escenarios puede ocurrir que, en los datos de test, aparezca un nuevo nivel que no estaba en los datos de entrenamiento. Si no se conoce de antemano cuáles son todos los posibles niveles, se puede evitar errores en las predicciones indicando OneHotEncoder(handle_unknown='ignore').

TRANSFORMACIONES CON SCIKIT-LEARN

La librería *Scikit-learn* permite realizar operaciones de preprocesado sobre las columnas de un *DataFrame* gracias a las clases *ColumnTransformer* y *make_column_transformer* del módulo *compose.* Es posible realizar varias transformaciones diferentes sobre los datos especificando a qué columnas se aplica cada una. Estas clases son *transformers*, eso significa que tienen un método de entrenamiento (*fit*) y otro de transformación (*transform*). Esto permite que el aprendizaje de las transformaciones se haga únicamente con datos de entrenamiento, y se puedan aplicar después a cualquier conjunto de datos, al de entrenamiento, validación y test.

En el ejemplo de la Figura 6 se realiza una selección de columnas por tipo, y luego se define la transformación a aplicar sobre cada grupo seleccionado. El resultado devuelto por *ColumnTransformer* es un array de *numpy*, por lo que se pierden los nombres de las columnas. Por ese motivo hay que volver a generar un *DataFrame* con los nombres de las nuevas columnas y los datos generados por *ColumnTransformer*.

ColumnTransformer aplica las transformaciones en paralelo, no de forma secuencial. Por lo tanto, solo puede aplicar una transformación a una misma columna. Si se necesita aplicar más de una transformación sobre una columna, se puede emplear el método *pipeline*, como muestra la Figura 7. Este método también agrupa transformaciones, pero las ejecutan de forma secuencial, de forma que la salida de una operación es la entrada de la siguiente.

```
In [165]: from sklearn.pipeline import Pipeline
          from sklearn.compose import ColumnTransformer
          from sklearn.impute import SimpleImputer
          from sklearn.preprocessing import OneHotEncoder
          from sklearn.preprocessing import StandardScaler
          from sklearn.compose import make_column_selector
          numeric_cols = viviendas.select_dtypes(include=['float64', 'int64']).columns.to_list()
          cat_cols = viviendas.select_dtypes(include=['object', 'category']).columns.to_list()
          numeric_transformer = Pipeline( steps=[ ('imputer', SimpleImputer(strategy='median')),
                                    ('scaler', StandardScaler()) ])
          categorical_transformer = Pipeline(steps=[ ('imputer', SimpleImputer(strategy='most_frequent')),
                                        ('onehot', OneHotEncoder(handle_unknown='ignore')) ])
          preprocessor = ColumnTransformer( transformers=[ ('numeric', numeric_transformer, numeric_cols),
                                ('cat', categorical_transformer, cat_cols) ], remainder='passthrough' )
          viviendas_prep = preprocessor.fit_transform(viviendas)
          encoded_cat = preprocessor.named_transformers_['cat']['onehot'].get_feature_names(cat_cols)
          labels = np.concatenate([numeric_cols, encoded_cat])
          datos_viviendas_prep = preprocessor.transform(viviendas)
          datos_viviendas_prep = pd.DataFrame(datos_viviendas_prep, columns=labels)
          datos_viviendas_prep.info()

          <class 'pandas.core.frame.DataFrame'>
          RangeIndex: 1728 entries, 0 to 1727
          Data columns (total 27 columns):
           #   Column                       Non-Null Count  Dtype
          ---  ------                       --------------  -----
           0   precio                       1728 non-null   float64
           1   metros_totales               1728 non-null   float64
           2   antiguedad                   1728 non-null   float64
           3   precio_terreno               1728 non-null   float64
           4   metros_habitables            1728 non-null   float64
           5   universitarios               1728 non-null   float64
           6   dormitorios                  1728 non-null   float64
           7   banyos                       1728 non-null   float64
           8   habitaciones                 1728 non-null   float64
           9   chimenea_0                   1728 non-null   float64
           10  chimenea_1                   1728 non-null   float64
           11  chimenea_2_mas               1728 non-null   float64
           12  calefaccion_electric         1728 non-null   float64
           13  calefaccion_hot air          1728 non-null   float64
           14  calefaccion_hot water/steam  1728 non-null   float64
           15  consumo_calefacion_electric  1728 non-null   float64
           16  consumo_calefacion_gas       1728 non-null   float64
           17  consumo_calefacion_oil       1728 non-null   float64
           18  desague_none                 1728 non-null   float64
           19  desague_public/commercial    1728 non-null   float64
           20  desague_septic               1728 non-null   float64
           21  vistas_lago_No               1728 non-null   float64
           22  vistas_lago_Yes              1728 non-null   float64
           23  nueva_construccion_No        1728 non-null   float64
           24  nueva_construccion_Yes       1728 non-null   float64
           25  aire_acondicionado_No        1728 non-null   float64
           26  aire_acondicionado_Yes       1728 non-null   float64
```

Figura 7. Agrupación de transformaciones en pipelines

AGRUPACIÓN DE DATOS EN PANDAS

En algunos casos es muy útil agrupar los datos en función de alguna característica, especialmente si se trata de la característica que queremos que nuestro modelo aprenda a predecir. De este modo, podremos conocer, por ejemplo, el número de clases diferentes en nuestro conjunto de datos, y el número de muestras de cada una de ellas.

Los *DataFrames* ofrecen el método *groupby* que permite dividir los datos en función de los valores de una o más variables, y permite hacer grupos de filas (*axis=0*) o de columnas (*axis=1*). El resultado es un objeto de tipo *Groupby* que tiene propiedades

como la de *ngroups* o *size* que permite conocer el número de grupos y el número de elementos de cada grupo. En la Figura 8 se muestra un ejemplo de agrupación por estación sobre un *DataFrame* con datos de meses.

```
In [8]: 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[8]:

	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 [11]: g = tabla.groupby(['Estación'])
         g.groups
Out[11]: {'inv': Index(['Ene', 'Feb', 'Mar'], dtype='object'),
          'oto': Index(['Oct', 'Nov', 'Dic'], dtype='object'),
          'pri': Index(['Abr', 'May', 'Jun'], dtype='object'),
          'ver': Index(['Jul', 'Ago', 'Sep'], dtype='object')}

In [12]: g.size()
Out[12]: Estación
         inv    3
         oto    3
         pri    3
         ver    3
         dtype: int64
```

Figura 8. Ejemplo de división en grupos de los datos de un DataFrame.

La división en grupos permite la aplicación de determinadas operaciones sobre los elementos de cada grupo, por ejemplo, en la Figura 9 se muestra como calcular la temperatura media de cada estación en 2022.

```
In [15]: tabla.groupby(['Estación'])['Temp.2022'].mean()
Out[15]: Estación
         inv     3.666667
         oto    10.333333
         pri    11.000000
         ver    22.666667
         Name: Temp.2022, dtype: float64
```

Figure 9. Resultado de aplicar una operación estadística por grupos.

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/