2.2.1_Preprocesamiento

June 13, 2025

Creado por:

Isabel Maniega

1 Pasos que realizar para el preprocesamiento:

1. Comprender los datos estructurados y no estructurados y sus implicaciones en el análisis de datos (ídem 1.1.4)¶

Los **datos estructurados** (o "limpios") son los datos clásicos de una hoja de cálculo: todo está bien y limpio (no quiere decir que no pueda haber datos faltantes o un formato incorrecto) y los datos están organizados en una estructura similar a una tabla. Las bases de datos que almacenan este tipo de datos se denominan bases de datos relacionales: usamos SQL para administrar los datos en esas bases de datos. Ejemplos de datos estructurados son los archivos .csv o Excel.

Los **datos semiestructurados**, como sugiere el nombre, incorporan algunos elementos de datos estructurados, aunque no están organizados en una estructura tabular. Sin embargo, contienen etiquetas y elementos para organizar los datos de una manera significativa y crear jerarquías. Las bases de datos que almacenan datos semiestructurados se denominan bases de datos no relacionales, como MongoDB.

Datos no estructurados, que son la mayoría de los datos del mundo, son datos sin procesar que no siguen ningún esquema. Son los más ricos en información, pero en la mayoría de los casos deben limpiarse para que sean significativos. Algunos ejemplos de datos no estructurados son los archivos de video y audio, así como las fotos.

Limpieza y preprocesamiento de datos

La limpieza y el preprocesamiento de datos son pasos esenciales para garantizar que sus datos sean precisos y estén listos para el análisis.

- Manejo de datos faltantes: Desarrolle estrategias para lidiar con los datos faltantes, como la imputación o la eliminación, según la naturaleza de sus datos y los objetivos de la investigación.
- Detección de valores atípicos: Identifique y aborde los valores atípicos que pueden sesgar los resultados del análisis. Considere si los valores atípicos deben corregirse, eliminarse o conservarse según su importancia.
- Normalización y escalamiento: Normalice o escale los datos para que estén dentro de un rango común, haciéndolos adecuados para ciertos algoritmos y modelos.

- Transformación de datos: Aplique transformaciones de datos, como escalamiento logarítmico o codificación categórica, para preparar los datos para tipos específicos de análisis.
- Desequilibrio de datos: Aborde problemas de desequilibrio de clases en conjuntos de datos, en particular aplicaciones de aprendizaje automático, para evitar un entrenamiento de modelos sesgado.

1.1 Preprocesando datos con texto en el dataset

En general, los algoritmos de machine learning no trabajan bien con datos con valores que no sean numéricos. Es por ello, que aquellos datos con valores que contengan un texto o sean una categoría, debemos transformarlos a valores numéricos.

Ejemplo de columnas no numéricas: «Color» que nos indica el color en inglés (Red, Blue, etc) y «Spectral_Class» que es otro dato categórico (M, O, A, etc).

Tipos de datos categóricos

Los datos categóricos se pueden clasificar en términos generales en dos tipos:

- **Datos nominales:** este tipo de datos representan categorías sin ningún orden inherente. Los ejemplos incluyen género (masculino, femenino), color (rojo, azul, verde) y país (EE. UU., India, Reino Unido).
- **Datos ordinales:** este tipo de datos representa categorías con un orden o clasificación significativo. Los ejemplos incluyen el nivel educativo (escuela secundaria, licenciatura, maestría, doctorado) y la satisfacción del cliente (baja, media, alta).

Técnicas de codificación Exploremos las técnicas más utilizadas:

1. Codificación de etiquetas (Label encoding)

La codificación de etiquetas es un método simple y directo que asigna un número entero único a cada categoría. Este método es adecuado para datos ordinales donde el orden de las categorías es significativo. Caso de Uso: Aplicable para casos en los que el ordenamiento de categorías tiene relevancia analítica.

```
data = ['red', 'blue', 'green', 'blue', 'red']

0 => 'blue'
1 => 'green'
2 => 'red'

data = [2 0 1 0 2]
```

2. Codificación en caliente (One-hot Encoding)

One-Hot Encoding convierte datos categóricos en una matriz binaria, donde cada categoría está representada por un vector binario. Este método es adecuado para datos nominales. Caso de uso: Más apropiado para aquellas situaciones en las que las categorías no tienen un orden inherente o existe una distinción clara entre ellas.

red	blue	green
0	1	0
1	0	0
0	0	1

```
data = ['red', 'blue', 'green', 'blue', 'red']

data = [[0. 1. 0.]
         [1. 0. 0.]
         [0. 0. 1.]
         [1. 0. 0.]
         [0. 1. 0.]]
```

Ventajas y desventajas de cada técnica de codificación

Técnica de codificación	Ventajas	Desventajas
Codificación de etiquetas	- Simple y fácil de implementar - Adecuado para datos ordinales	- Introduce relaciones ordinales arbitrarias para datos nominales- Puede no funcionar bien con valores atípicos
Codificación en caliente	- Adecuado para datos nominales - Evita introducir relaciones ordinales- Mantiene información sobre los valores de cada variable	- Puede provocar un aumento de la dimensionalidad y la escasez. - Puede provocar un sobreajuste, especialmente con muchas categorías y tamaños de muestra pequeños.

1.2 Escalado de los datos

Este será una de las principales tareas que realicemos en el preprocesamiento de datasets. Los principales algoritmos de machine learning que existen no funcionan muy bien cuando existen una gran diferencia entre los valores de una columna. Si tenemos una columna «L» que contiene valores muy distante, por ejemplo, tiene un valor mínimo de 0 y máximo de 849820. Esto es algo que debemos evitar, ya que los algoritmos de aprendizaje no funcionan bien en estos casos.

Existen dos formas claras de solucionar estos problemas de escalas: la normalización de valores y la estandarización.

Normalización

El escalado min-max o normalización, es una técnica común a la hora de solucionar el problema de tener diferentes escalas en los valores de una columna. El objetivo que se consigue con esta técnica es que todos los valores de una columna estén comprendido en el intervalo [0-1]. De forma matemática, lo que estamos haciendo es a cada valor le restamos el mínimo y lo dividimos entre el valor máximo.

Otras maneras de utilizar la normalización, además de min-max son:

- Normalización Z-score
- Normalizado por escala decimal

Otras maneras de utilizar la normalización, además de min-max son:

- **Normalización min-max:** se calcula de la siguiente fórmula

$$x_s = \frac{x - x_{min}}{x_{max} - x_{min}} * (max - min) + min$$

donde:

- x es la variable en su escala original
- xmax y xmin son el máximo y el mínimo
- *max* y *min* son el máximo y el mínimo pre-definidos (es decir que queremos obtener tras el escalamiento)
- xs es la variable obtenida tras el escalamiento

¿Cuándo NO usar MinMaxScaler?

- Cuando la distribución tiene un sesgo Una distribución con sesgo es cuando su forma se aleja demasiado de una distribución normal o gaussiana (campana simétrica).

En este caso NO se recomienda el uso de MinMaxScaler pues al escalar los datos comprimimos la distribución de los datos a un rango más pequeño que el original. Es decir que desaprovechamos todo el rango de valores disponible tras el escalamiento

- Cuando los datos tienen valores extremos (outliers) Los valores extremos (u outliers) son simplemente datos cuyos valores se encuentran excepcionalmente fuera del rango de valores de la mayoría de nuestros datos.

En este caso tampoco se recomienda el uso de minmaxscaler. Y esto se debe a que por tratarse de valores extremos, estos outliers generalmente corresponderán a los valores xmax y xmin que aparecen en la ecuación de escalamiento de nuestra variable original.

Así que al hacer el escalamiento estos valores extremos quedarán mapeados al máximo y mínimo en el rango resultante, mientras que los valores no extremos (que es la mayoría de los datos) quedarán mapeados dentro de un rango resultante muchísimo menor.

Es decir que si usamos minmaxscaler cuando tenemos valores extremos en nuestros datos hace que la mayoría de nuestros datos (que no son extremos) queden como resultado comprimidos a un rango de valores muy pequeño.

¿Cuándo podemos usar MinMaxScaler? Teniendo en cuenta lo anterior, se sugiere el uso de MinMaxScaler en estas situaciones:

1. Cuando tenemos claro el rango de valores esperado a la salida del escalamiento
2. Cuando la distribución de los datos NO tiene demasiado sesgo
3. Cuando los datos NO contienen outliers

Estandarización

La otra forma de realizar el escalado de valores que vamos a ver se denomina estandarización. Matemáticamente hablando, lo que estamos haciendo en este proceso es restar la media de los valores y dividir por la desviación estándar de los mismos. De esta forma los valores obtenidos tendrán una media de cero y una varianza de uno.

Existen algunas diferencias notables con respecto a la normalización. En primer lugar los valores obtenidos por la estandarización no están acotados en ningún rango ([0-1] por ejemplo).

En segundo lugar, este método consigue solucionar el problema de valores atípicos u outliers que presenta la normalización. Por ejemplo, supongamos que un atributo de temperatura contiene valores entre 0 y 100 por regla general. Por un error de medición, tenemos un valor de 10000 que se consideraría un outlier. Si aplicamos la normalización, la mayor parte de valores estarían en el rango 0-0.1. Sin embargo, esto no se da con la estandarización.

En sklearn tenemos el método StandardScaler() para calcularlo usamos los siguiente:

- **Normalización Z-score:** sigue la siguiente fórmula:

$$z = (x - u)/s$$

donde:

- x el valor de la muestra
- u es la media de los datos
- s es la desviación estandar de los datos.

Así pues podemos encontrarnos con distintas etapas del preprocesamiento:

- **Data cleaning:** la limpieza de datos elimina ruido y resuelve las inconsistencias en los datos.
- **Data integration:** con la Integración de datos se migran datos de varias fuentes a una fuente coherente como un Data Warehouse.
- **Data transformation:** la transformación de datos sirve para normalizar datos de cualquier tipo.
- **Data reduction:** la reducción de datos reduce el tamaño de los datos agregandolos.

1.3 ETL - Extract, Transform, Load

Las herramientas ETL, ya poseen la mayoría de las técnicas de procesamiento de datos mencionadas anteriormente como la migración de datos y la transformación de datos, esto hace que el seguimiento de estas prácticas de limpieza de datos resulte mucho más conveniente. Además, tales herramientas ETL permiten a los usuarios especificar los tipos de transformaciones que desean realizar con sus datos.

1.3.1 LIBRERÍA SCIKIT-LEARN (sklearn)

Para realizar este paso podemos ir a la librería sklearn para ver los distintos tipos disponibles: https://scikit-learn.org/stable/modules/preprocessing.html

[1]:
```
# pip install pandas
```

[2]:
```
# pip install scikit-learn
```

[3]:
```
# pip install seaborn
```

[4]:
```
import pandas as pd
from sklearn.datasets import fetch_openml
import seaborn as sns
from sklearn.impute import SimpleImputer
import numpy as np
```

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

2 Cargar el titanic con sklearn

[5]:
```
df = fetch_openml("titanic", version=1, as_frame=True)["data"]
df.head()
```

[5]:
```
   pclass                                            name     sex      age  \
0       1                    Allen, Miss. Elisabeth Walton  female  29.0000
1       1                   Allison, Master. Hudson Trevor    male   0.9167
2       1                     Allison, Miss. Helen Loraine  female   2.0000
3       1             Allison, Mr. Hudson Joshua Creighton    male  30.0000
4       1  Allison, Mrs. Hudson J C (Bessie Waldo Daniels)  female  25.0000

   sibsp  parch  ticket      fare    cabin embarked boat   body  \
0      0      0   24160  211.3375       B5        S    2    NaN
1      1      2  113781  151.5500  C22 C26        S   11    NaN
2      1      2  113781  151.5500  C22 C26        S  NaN    NaN
3      1      2  113781  151.5500  C22 C26        S  NaN  135.0
4      1      2  113781  151.5500  C22 C26        S  NaN    NaN

                        home.dest
0                    St Louis, MO
1  Montreal, PQ / Chesterville, ON
2  Montreal, PQ / Chesterville, ON
3  Montreal, PQ / Chesterville, ON
4  Montreal, PQ / Chesterville, ON
```

[6]:
```
# df.to_csv("Titanic_all.csv")
```

3 Mostrar las columnas sin datos

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 13 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   pclass     1309 non-null   int64
 1   name       1309 non-null   object
 2   sex        1309 non-null   category
 3   age        1046 non-null   float64
 4   sibsp      1309 non-null   int64
```

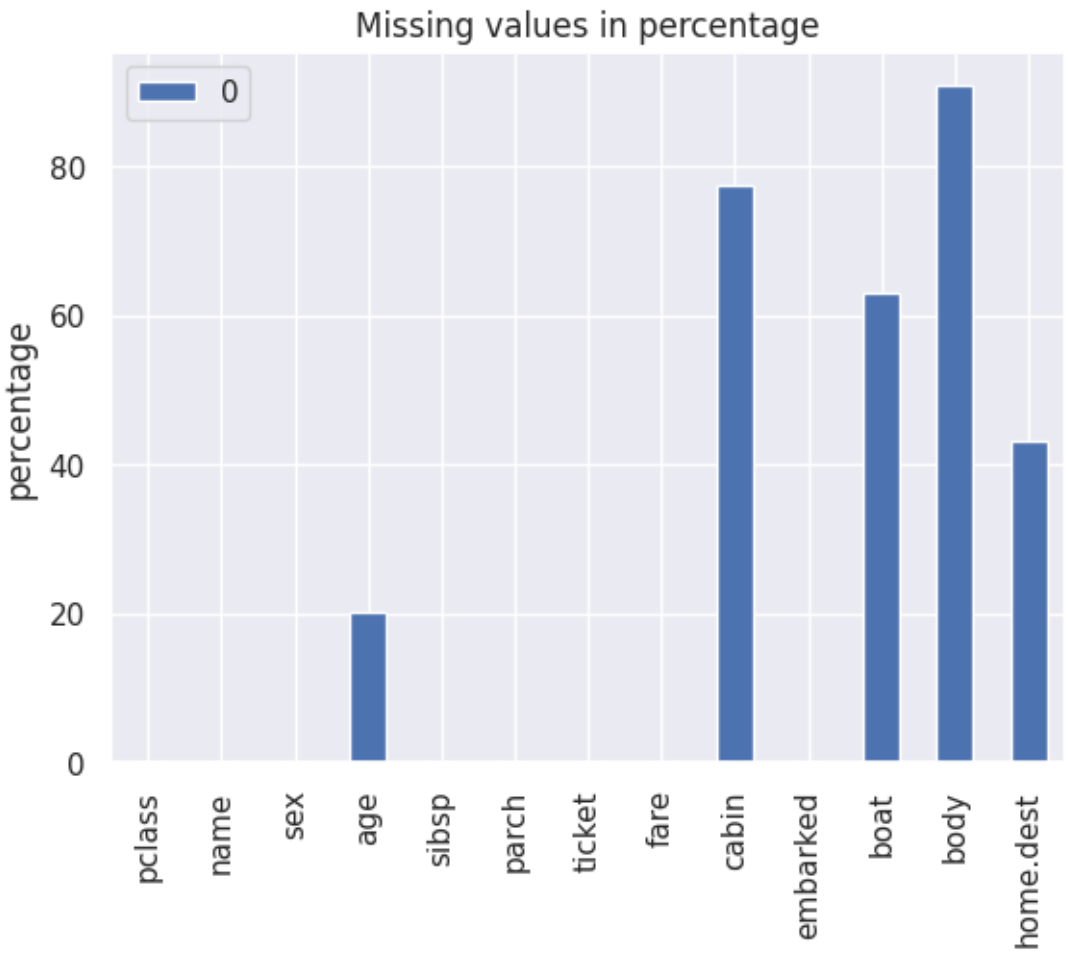
```
 5   parch      1309 non-null   int64
 6   ticket     1309 non-null   object
 7   fare       1308 non-null   float64
 8   cabin      295 non-null    object
 9   embarked   1307 non-null   category
 10  boat       486 non-null    object
 11  body       121 non-null    float64
 12  home.dest  745 non-null    object
dtypes: category(2), float64(3), int64(3), object(5)
memory usage: 115.4+ KB
```

[8]:
```
df.isnull().sum()
```

[8]:
```
pclass          0
name            0
sex             0
age           263
sibsp           0
parch           0
ticket          0
fare            1
cabin        1014
embarked        2
boat          823
body         1188
home.dest     564
dtype: int64
```

[9]:
```
# Visusalización de los datos

sns.set()
miss_vals = pd.DataFrame(df.isnull().sum()/ len(df)*100)
miss_vals.plot(kind="bar",
               title="Missing values in percentage",
               ylabel="percentage")
```

[9]:
```
<Axes: title={'center': 'Missing values in percentage'}, ylabel='percentage'>
```



3.1 Procedimiento para valores nulos

Existen dos maneras:

- Eliminar la columna
- Asignamos a los valores la media, mediana, moda, etc

Eliminación

- Eliminamos los valores nulos:

[10]:
```
print(f"Size of the dataset: {df.shape}")
df.drop(["cabin", "boat", "body", "home.dest"], axis=1, inplace=True)
df.dropna(inplace=True)
print(f"Size of the dataset: {df.shape}")
```

```
Size of the dataset: (1309, 13)
Size of the dataset: (1043, 9)
```

Sustitución

- Sustituir por el valor más común (media):

[11]:
```
df = fetch_openml("titanic", version=1, as_frame=True)["data"]
df.head()
```

[11]:
```
   pclass                                            name     sex      age  \
0       1                    Allen, Miss. Elisabeth Walton  female  29.0000
1       1                   Allison, Master. Hudson Trevor    male   0.9167
2       1                     Allison, Miss. Helen Loraine  female   2.0000
3       1             Allison, Mr. Hudson Joshua Creighton    male  30.0000
4       1  Allison, Mrs. Hudson J C (Bessie Waldo Daniels)  female  25.0000

   sibsp  parch  ticket      fare    cabin embarked boat   body  \
0      0      0   24160  211.3375       B5        S    2    NaN
1      1      2  113781  151.5500  C22 C26        S   11    NaN
2      1      2  113781  151.5500  C22 C26        S  NaN    NaN
3      1      2  113781  151.5500  C22 C26        S  NaN  135.0
4      1      2  113781  151.5500  C22 C26        S  NaN    NaN

                        home.dest
0                    St Louis, MO
1  Montreal, PQ / Chesterville, ON
2  Montreal, PQ / Chesterville, ON
3  Montreal, PQ / Chesterville, ON
4  Montreal, PQ / Chesterville, ON
```

[12]:
```
print(f"Número de valores nulos de la columna edad: {df.age.isnull().sum()}")
```

```
Número de valores nulos de la columna edad: 263
```

[13]:
```
df["age"].fillna(df["age"].mean())
print(f"Número de valores nulos de la columna edad: {df.age.isnull().sum()}")
```

```
Número de valores nulos de la columna edad: 263
```

- Otra opción: Simple transformación con Sklearn

[14]:
```
df = fetch_openml("titanic", version=1, as_frame=True)["data"]
df.head()
```

[14]:
```
   pclass                                            name     sex      age  \
0       1                    Allen, Miss. Elisabeth Walton  female  29.0000
1       1                   Allison, Master. Hudson Trevor    male   0.9167
2       1                     Allison, Miss. Helen Loraine  female   2.0000
3       1             Allison, Mr. Hudson Joshua Creighton    male  30.0000
4       1  Allison, Mrs. Hudson J C (Bessie Waldo Daniels)  female  25.0000

   sibsp  parch  ticket      fare    cabin embarked boat   body  \
0      0      0   24160  211.3375       B5        S    2    NaN
```

```
1      1      2  113781  151.5500  C22 C26        S   11    NaN
2      1      2  113781  151.5500  C22 C26        S  NaN    NaN
3      1      2  113781  151.5500  C22 C26        S  NaN  135.0
4      1      2  113781  151.5500  C22 C26        S  NaN    NaN

                        home.dest
0                     St Louis, MO
1  Montreal, PQ / Chesterville, ON
2  Montreal, PQ / Chesterville, ON
3  Montreal, PQ / Chesterville, ON
4  Montreal, PQ / Chesterville, ON
```

[15]:
```
print(f"Número de valores nulos de la columna edad: {df.age.isnull().sum()}")
```

```
Número de valores nulos de la columna edad: 263
```

[16]:
```
imp = SimpleImputer(strategy="mean")
df["age"] = imp.fit_transform(df[["age"]])
print(f"Número de valores nulos de la columna edad: {df.age.isnull().sum()}")
```

```
Número de valores nulos de la columna edad: 0
```

[17]:
```
print("Tipos de datos con valores Nulos:")
for col in df.columns[df.isnull().any()]:
    print(col, df[col][df[col].isnull()].values[0])
```

```
Tipos de datos con valores Nulos:
fare nan
cabin nan
embarked nan
boat nan
body nan
home.dest nan
```

- Modificamos los None:

[18]:
```
df = fetch_openml("titanic", version=1, as_frame=True)["data"]
df.head()
```

[18]:
```
   pclass                                             name     sex      age  \
0       1                    Allen, Miss. Elisabeth Walton  female  29.0000
1       1                   Allison, Master. Hudson Trevor    male   0.9167
2       1                     Allison, Miss. Helen Loraine  female   2.0000
3       1             Allison, Mr. Hudson Joshua Creighton    male  30.0000
4       1  Allison, Mrs. Hudson J C (Bessie Waldo Daniels)  female  25.0000

   sibsp  parch  ticket      fare    cabin embarked boat   body  \
0      0      0   24160  211.3375       B5        S    2    NaN
1      1      2  113781  151.5500  C22 C26        S   11    NaN
```

```
2      1      2  113781  151.5500  C22 C26        S  NaN    NaN
3      1      2  113781  151.5500  C22 C26        S  NaN  135.0
4      1      2  113781  151.5500  C22 C26        S  NaN    NaN

                        home.dest
0                     St Louis, MO
1  Montreal, PQ / Chesterville, ON
2  Montreal, PQ / Chesterville, ON
3  Montreal, PQ / Chesterville, ON
4  Montreal, PQ / Chesterville, ON
```

[19]:
```
cols_categoricas = ["pclass", "sex", "embarked"]

df[cols_categoricas] = df[cols_categoricas].astype("category")
df.dtypes
```

[19]:
```
pclass       category
name           object
sex          category
age           float64
sibsp           int64
parch           int64
ticket         object
fare          float64
cabin          object
embarked     category
boat           object
body          float64
home.dest      object
dtype: object
```

[20]:
```
df["pclass"] = pd.Categorical(df["pclass"],categories=[3, 2, 1],
                              ordered=True)
df.dtypes
```

[20]:
```
pclass       category
name           object
sex          category
age           float64
sibsp           int64
parch           int64
ticket         object
fare          float64
cabin          object
embarked     category
boat           object
body          float64
```

```
home.dest      object
dtype: object
```

[21]:
```
cols_numericas = ["age", "fare"]

df[cols_numericas] = df[cols_numericas].astype("float")
df.dtypes
```

[21]:
```
pclass       category
name           object
sex          category
age           float64
sibsp           int64
parch           int64
ticket         object
fare          float64
cabin          object
embarked     category
boat           object
body          float64
home.dest      object
dtype: object
```

df = df.drop(['boat', 'body', 'home.dest','name', 'ticket', 'cabin'], axis=1)

df.head()

[22]:
```
def get_parameters(df):
    parameters = {}
    for col in df.columns[df.isnull().any()]:
        if df[col].dtype == "float64" or df[col].dtype == "int64" or df[col].
 ↪dtype == "int32":
            strategy = "mean"
        else:
            strategy = "most_frequent"
        missing_values = df[col][df[col].isnull()].values[0]
        parameters[col] = {"missing_values": missing_values, "strategy":␣
 ↪strategy}
    return parameters
get_parameters(df)
```

[22]:
```
{'age': {'missing_values': np.float64(nan), 'strategy': 'mean'},
 'fare': {'missing_values': np.float64(nan), 'strategy': 'mean'},
 'cabin': {'missing_values': nan, 'strategy': 'most_frequent'},
 'embarked': {'missing_values': nan, 'strategy': 'most_frequent'},
 'boat': {'missing_values': nan, 'strategy': 'most_frequent'},
 'body': {'missing_values': np.float64(nan), 'strategy': 'mean'},
 'home.dest': {'missing_values': nan, 'strategy': 'most_frequent'}}
```

[23]:
```
parameters = get_parameters(df)

for col, param in parameters.items():
    missing_values = param["missing_values"]
    strategy = param["strategy"]
    imp = SimpleImputer(missing_values=missing_values, strategy=strategy)
    if strategy == "most_frequent":
        df[[col]] = imp.fit_transform(df[[col]])
    else:
        df[col] = imp.fit_transform(df[[col]])

df.isnull().sum()
```

[23]:
```
pclass       0
name         0
sex          0
age          0
sibsp        0
parch        0
ticket       0
fare         0
cabin        0
embarked     0
boat         0
body         0
home.dest    0
dtype: int64
```

[24]:
```
df.head()
```

[24]:

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
0	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375	B5	S	2	160.809917	St Louis, MO
1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500	C22 C26	S	11	160.809917	Montreal, PQ / Chesterville, ON
2	1	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781	151.5500	C22 C26	S	13	160.809917	
3	1	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781	151.5500	C22 C26	S	13	135.000000	
4	1	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1	2	113781	151.5500	C22 C26	S	13	160.809917	

```
2  Montreal, PQ / Chesterville, ON
3  Montreal, PQ / Chesterville, ON
4  Montreal, PQ / Chesterville, ON
```

3.2 Crear nuevas características (Feature Engineering)

- Sibsp: pasajeros que viajan con hermanos
- Parch: viajeros que viajan con niños

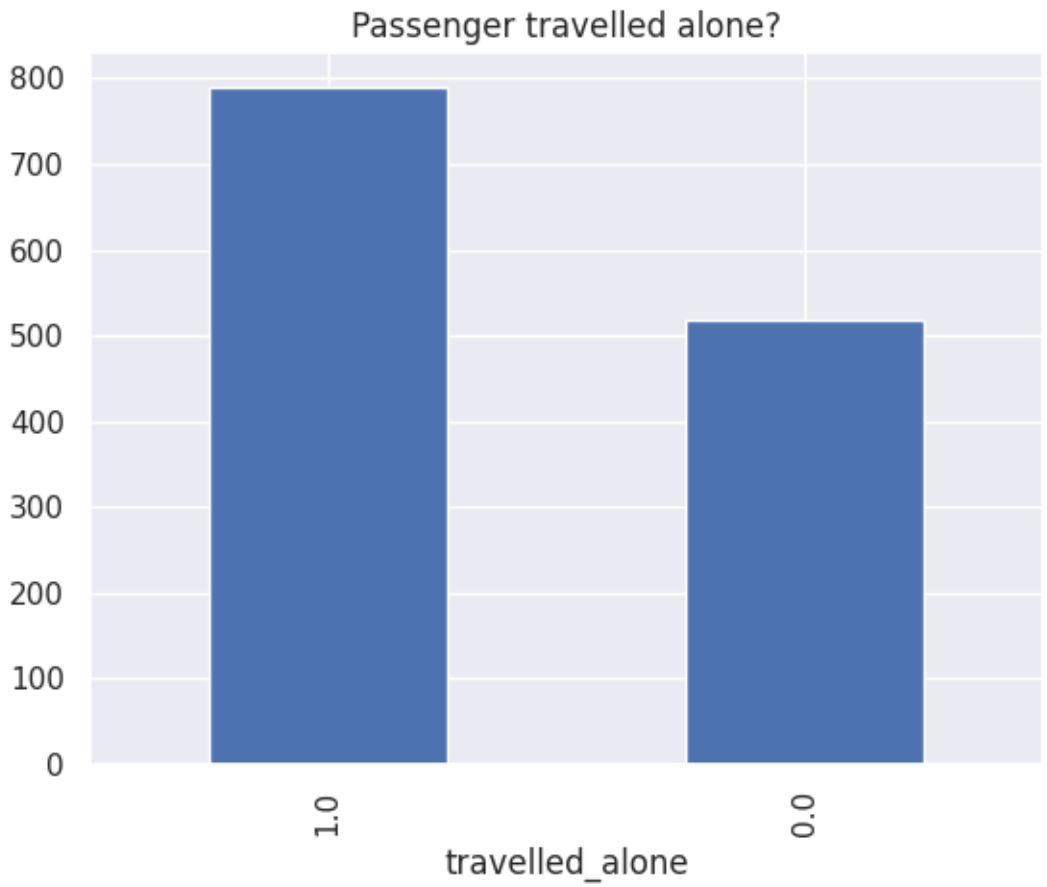
Calculamos el número de pasajeros que viajan solos:

[25]:
```
df = fetch_openml("titanic", version=1, as_frame=True)["data"]
df.head()
```

[25]:
```
   pclass                                            name     sex      age  \
0       1                    Allen, Miss. Elisabeth Walton  female  29.0000
1       1                   Allison, Master. Hudson Trevor    male   0.9167
2       1                     Allison, Miss. Helen Loraine  female   2.0000
3       1             Allison, Mr. Hudson Joshua Creighton    male  30.0000
4       1  Allison, Mrs. Hudson J C (Bessie Waldo Daniels)  female  25.0000

   sibsp  parch  ticket      fare    cabin embarked boat   body  \
0      0      0   24160  211.3375       B5        S    2    NaN
1      1      2  113781  151.5500  C22 C26        S   11    NaN
2      1      2  113781  151.5500  C22 C26        S  NaN    NaN
3      1      2  113781  151.5500  C22 C26        S  NaN  135.0
4      1      2  113781  151.5500  C22 C26        S  NaN    NaN

                        home.dest
0                    St Louis, MO
1  Montreal, PQ / Chesterville, ON
2  Montreal, PQ / Chesterville, ON
3  Montreal, PQ / Chesterville, ON
4  Montreal, PQ / Chesterville, ON
```

[26]:
```
df["family"] = df["sibsp"] + df["parch"]

df.loc[df["family"] > 0, "travelled_alone"] = 0
df.loc[df["family"] == 0, "travelled_alone"] = 1

df["travelled_alone"].value_counts().plot(title="Passenger travelled alone?",␣
↪kind="bar")
```

[26]:
```
<Axes: title={'center': 'Passenger travelled alone?'}, xlabel='travelled_alone'>
```



3.3 Encode categorical features

- scikit-learn: OneHotEncoder()
- pandas: get_dummies()

[27]:
```
df = fetch_openml("titanic", version=1, as_frame=True)["data"]
df.head()
```

[27]:

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body
0	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375	B5	S	2	NaN
1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500	C22 C26	S	11	NaN
2	1	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN
3	1	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN
4	1	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN

```
3      1      2  113781  151.5500  C22 C26        S  NaN  135.0
4      1      2  113781  151.5500  C22 C26        S  NaN    NaN

                        home.dest
0                     St Louis, MO
1  Montreal, PQ / Chesterville, ON
2  Montreal, PQ / Chesterville, ON
3  Montreal, PQ / Chesterville, ON
4  Montreal, PQ / Chesterville, ON
```

[28]:
```
df[["female", "male"]] = OneHotEncoder().fit_transform(df[['sex']]).toarray()
df[["sex", "female", "male"]]
```

[28]:
```
          sex  female  male
0     female     1.0   0.0
1       male     0.0   1.0
2     female     1.0   0.0
3       male     0.0   1.0
4     female     1.0   0.0
…       …      …     …
1304  female     1.0   0.0
1305  female     1.0   0.0
1306    male     0.0   1.0
1307    male     0.0   1.0
1308    male     0.0   1.0

[1309 rows x 3 columns]
```

Eliminaremos uno de las columnas para evitar la colinealidad

[29]:
```
df = fetch_openml("titanic", version=1, as_frame=True)["data"]
df["sex"] = OneHotEncoder().fit_transform(df[['sex']]).toarray()[:, 1]
df.head()
```

[29]:
```
   pclass                                            name  sex      age  \
0       1                    Allen, Miss. Elisabeth Walton  0.0  29.0000
1       1                   Allison, Master. Hudson Trevor  1.0   0.9167
2       1                     Allison, Miss. Helen Loraine  0.0   2.0000
3       1             Allison, Mr. Hudson Joshua Creighton  1.0  30.0000
4       1  Allison, Mrs. Hudson J C (Bessie Waldo Daniels)  0.0  25.0000

   sibsp  parch  ticket      fare    cabin embarked boat   body  \
0      0      0   24160  211.3375       B5        S    2    NaN
1      1      2  113781  151.5500  C22 C26        S   11    NaN
2      1      2  113781  151.5500  C22 C26        S  NaN    NaN
3      1      2  113781  151.5500  C22 C26        S  NaN  135.0
4      1      2  113781  151.5500  C22 C26        S  NaN    NaN
```

```
                        home.dest
0                     St Louis, MO
1  Montreal, PQ / Chesterville, ON
2  Montreal, PQ / Chesterville, ON
3  Montreal, PQ / Chesterville, ON
4  Montreal, PQ / Chesterville, ON
```

0 == female; 1== male

- Pandas:

[30]:
```
df = fetch_openml("titanic", version=1, as_frame=True)["data"]

df["sex"] = pd.get_dummies(df["sex"], drop_first=True, dtype=int)
df.head()
```

[30]:
```
   pclass                                            name  sex      age  \
0       1                    Allen, Miss. Elisabeth Walton    0  29.0000
1       1                   Allison, Master. Hudson Trevor    1   0.9167
2       1                     Allison, Miss. Helen Loraine    0   2.0000
3       1             Allison, Mr. Hudson Joshua Creighton    1  30.0000
4       1  Allison, Mrs. Hudson J C (Bessie Waldo Daniels)    0  25.0000

   sibsp  parch  ticket      fare    cabin embarked boat   body  \
0      0      0   24160  211.3375       B5        S    2    NaN
1      1      2  113781  151.5500  C22 C26        S   11    NaN
2      1      2  113781  151.5500  C22 C26        S  NaN    NaN
3      1      2  113781  151.5500  C22 C26        S  NaN  135.0
4      1      2  113781  151.5500  C22 C26        S  NaN    NaN

                        home.dest
0                     St Louis, MO
1  Montreal, PQ / Chesterville, ON
2  Montreal, PQ / Chesterville, ON
3  Montreal, PQ / Chesterville, ON
4  Montreal, PQ / Chesterville, ON
```

0 == female; 1== male

3.4 Encoding all categorical features

[31]:
```
df = fetch_openml("titanic", version=1, as_frame=True)["data"]

cat_cols = df.select_dtypes(include=["category"]).columns
print(f"Columnas Categoricas: {cat_cols}")
```

```
Columnas Categoricas: Index(['sex', 'embarked'], dtype='object')
```

[32]:
```
for col in cat_cols:
    fill_value = df[col].mode()[0]
    df[col].fillna(fill_value)

    append_to = list(df[col].unique())

    print(append_to)

    df[append_to] = OneHotEncoder().fit_transform(df[[col]]).toarray()

    df.drop(col, axis=1, inplace=True)
    df.drop(append_to[0], axis=1, inplace=True)

print(df.columns)
df[["male", "C", "Q"]].head()
```

```
['female', 'male']
['S', 'C', nan, 'Q']
Index([   'pclass',      'name',       'age',     'sibsp',     'parch',
          'ticket',      'fare',     'cabin',      'boat',      'body',
       'home.dest',      'male',         'C',         nan,         'Q'],
      dtype='object')
```

[32]:

	male	C	Q
0	0.0	0.0	0.0
1	1.0	0.0	0.0
2	0.0	0.0	0.0
3	1.0	0.0	0.0
4	0.0	0.0	0.0

[33]:
```
df.head()
```

[33]:

	pclass	name	age	sibsp
0	1	Allen, Miss. Elisabeth Walton	29.0000	0
1	1	Allison, Master. Hudson Trevor	0.9167	1
2	1	Allison, Miss. Helen Loraine	2.0000	1
3	1	Allison, Mr. Hudson Joshua Creighton	30.0000	1
4	1	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	25.0000	1

	parch	ticket	fare	cabin	boat	body
0	0	24160	211.3375	B5	2	NaN
1	2	113781	151.5500	C22 C26	11	NaN
2	2	113781	151.5500	C22 C26	NaN	NaN
3	2	113781	151.5500	C22 C26	NaN	135.0
4	2	113781	151.5500	C22 C26	NaN	NaN

	home.dest	male	C	NaN	Q
0	St Louis, MO	0.0	0.0	1.0	0.0

```
1  Montreal, PQ / Chesterville, ON   1.0  0.0  1.0  0.0
2  Montreal, PQ / Chesterville, ON   0.0  0.0  1.0  0.0
3  Montreal, PQ / Chesterville, ON   1.0  0.0  1.0  0.0
4  Montreal, PQ / Chesterville, ON   0.0  0.0  1.0  0.0
```

3.5 MinMaxScaler

MinMaxScaler() pone todos los valores númericos de 0 a 1:

[34]:
```
df = fetch_openml("titanic", version=1, as_frame=True)["data"]

num_cols = df.select_dtypes(include=["int64", "int32", "float64"]).columns
print(num_cols)
```

```
Index(['pclass', 'age', 'sibsp', 'parch', 'fare', 'body'], dtype='object')
```

[35]:
```
for col in num_cols:
    fill_value = df[col].mean()
    df[col].fillna(fill_value)

minmax = MinMaxScaler()

df[num_cols] = minmax.fit_transform(df[num_cols])
df[num_cols]
```

[35]:

	pclass	age	sibsp	parch	fare	body
0	0.0	0.361169	0.000	0.000000	0.412503	NaN
1	0.0	0.009395	0.125	0.222222	0.295806	NaN
2	0.0	0.022964	0.125	0.222222	0.295806	NaN
3	0.0	0.373695	0.125	0.222222	0.295806	0.409786
4	0.0	0.311064	0.125	0.222222	0.295806	NaN
…	…	…	…	…	…	…
1304	1.0	0.179540	0.125	0.000000	0.028213	1.000000
1305	1.0	NaN	0.125	0.000000	0.028213	NaN
1306	1.0	0.329854	0.000	0.000000	0.014102	0.926606
1307	1.0	0.336117	0.000	0.000000	0.014102	NaN
1308	1.0	0.361169	0.000	0.000000	0.015371	NaN

[1309 rows x 6 columns]

3.6 StandardScaler

StandardScaler() poner todos los valores tengan una media de 0 y de desviación de 1

[36]:
```
df = fetch_openml("titanic", version=1, as_frame=True)["data"]

num_cols = df.select_dtypes(include=["int64", "int32", "float64"]).columns
print(num_cols)
```

```
Index(['pclass', 'age', 'sibsp', 'parch', 'fare', 'body'], dtype='object')
```

[37]:
```
for col in num_cols:
    fill_value = df[col].mean()
    df[col].fillna(fill_value)

ss = StandardScaler()

df[num_cols] = ss.fit_transform(df[num_cols])
df[num_cols].head()
```

[37]:

	pclass	age	sibsp	parch	fare	body
0	-1.546098	-0.061162	-0.479087	-0.445000	3.441165	NaN
1	-1.546098	-2.010496	0.481288	1.866526	2.285603	NaN
2	-1.546098	-1.935302	0.481288	1.866526	2.285603	NaN
3	-1.546098	0.008251	0.481288	1.866526	2.285603	-0.265282
4	-1.546098	-0.338812	0.481288	1.866526	2.285603	NaN

[38]:
```
df[num_cols].describe()
```

[38]:

	pclass	age	sibsp	parch	fare	body
count	1.309000e+03	1.046000e+03	1.309000e+03	1.309000e+03	1.308000e+03	1.210000e+02
mean	-1.737003e-16	-1.358590e-16	-8.142201e-18	1.628440e-17	-8.691654e-17	-8.074349e-17
std	1.000382e+00	1.000478e+00	1.000382e+00	1.000382e+00	1.000382e+00	1.004158e+00
min	-1.546098e+00	-2.062556e+00	-4.790868e-01	-4.449995e-01	-6.435292e-01	-1.642574e+00
25%	-3.520907e-01	-6.164626e-01	-4.790868e-01	-4.449995e-01	-4.909206e-01	-9.128147e-01
50%	8.419164e-01	-1.305744e-01	-4.790868e-01	-4.449995e-01	-3.641609e-01	-5.971606e-02
75%	8.419164e-01	6.329641e-01	4.812878e-01	-4.449995e-01	-3.905147e-02	9.783920e-01
max	8.419164e-01	3.478880e+00	7.203909e+00	9.956864e+00	9.258680e+00	1.718429e+00

Usando el método de describe() podemos ver la media y la desvicación estandar de las columnas escaladas.

La media no parece ser igual a 0 pero, de hecho 4.342507e-17 es igual 0,00000000000000043425. Esto es tan cercano a 0 que puede considerarse igual a 0. Lo mismo ocurre con la desviación estándar que es tan cercana a 1 que puede considerarse igual a 1.

Creado por:

Isabel Maniega