tratamiento_datos

March 2, 2023

[ ]:

[126]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import OneHotEncoder
import matplotlib.pyplot as plt
```

[ ]:

1 0. Basics

[ ]:

[210]:
```
# Generamos un dataframe

# Crear un DataFrame con diferentes variables y ciudades repetidas
data = {'Ciudad': ['Madrid', 'Barcelona', 'Sevilla', 'Valencia', 'Madrid',␣
↪'Sevilla', 'Valencia', 'Barcelona', 'Sevilla', 'Madrid'],
        'Edad': [25, 32, 45, 28, 37, 22, 29, 31, 24, 36],
        'Ingresos': [50000, 80000, 65000, 45000, 70000, 55000, 48000, 82000,␣
↪60000, 75000],
        'Hijos': [2, 0, 1, 0, 3, 2, 1, 0, 2, 1]}

df = pd.DataFrame(data)

# Imprimir el DataFrame
df
```

[210]:

	Ciudad	Edad	Ingresos	Hijos
0	Madrid	25	50000	2
1	Barcelona	32	80000	0
2	Sevilla	45	65000	1
3	Valencia	28	45000	0

```
4     Madrid    37     70000      3
5    Sevilla    22     55000      2
6   Valencia    29     48000      1
7  Barcelona    31     82000      0
8    Sevilla    24     60000      2
9     Madrid    36     75000      1
```

[241]:
```
# Dimension

print(f"El numero de filas es {df.shape[0]} unidades")
print(f"El numero de columnas es {df.shape[1]} unidades")
```

```
El numero de filas es 10 unidades
El numero de columnas es 5 unidades
```

[245]:
```
# Cambiamos el nombre de las columnas para tener todas en miniculas

for j in df.columns:
    nm = str(j).lower()
    df[f"{nm}_lower"] = df[j]

df
```

[245]:

	Ciudad	Edad	Ingresos	Hijos	prueba	ciudad_lower	edad_lower	\
0	Madrid	25	50000	2	25	Madrid	25	
1	Barcelona	32	80000	0	32	Barcelona	32	
2	Sevilla	45	65000	1	45	Sevilla	45	
3	Valencia	28	45000	0	28	Valencia	28	
4	Madrid	37	70000	3	37	Madrid	37	
5	Sevilla	22	55000	2	22	Sevilla	22	
6	Valencia	29	48000	1	29	Valencia	29	
7	Barcelona	31	82000	0	31	Barcelona	31	
8	Sevilla	24	60000	2	24	Sevilla	24	
9	Madrid	36	75000	1	36	Madrid	36	

	ingresos_lower	hijos_lower	prueba_lower
0	50000	2	25
1	80000	0	32
2	65000	1	45
3	45000	0	28
4	70000	3	37
5	55000	2	22
6	48000	1	29
7	82000	0	31
8	60000	2	24
9	75000	1	36

[252]:
```
# Una forma mas facil de hacer este tipo de cambios

cols_to_update = df.filter(like='lowe').columns # que colum queremos cambiar
new_cols = [str(col).upper() for col in cols_to_update] # tratamiento a los␣
 ↪nombres especificos
df.rename(columns=dict(zip(cols_to_update, new_cols)), inplace=True) # el rename
df
```

[252]:

	Ciudad	Edad	Ingresos	Hijos	prueba	CIUDAD_LOWER	EDAD_LOWER	\
0	Madrid	25	50000	2	25	Madrid	25	
1	Barcelona	32	80000	0	32	Barcelona	32	
2	Sevilla	45	65000	1	45	Sevilla	45	
3	Valencia	28	45000	0	28	Valencia	28	
4	Madrid	37	70000	3	37	Madrid	37	
5	Sevilla	22	55000	2	22	Sevilla	22	
6	Valencia	29	48000	1	29	Valencia	29	
7	Barcelona	31	82000	0	31	Barcelona	31	
8	Sevilla	24	60000	2	24	Sevilla	24	
9	Madrid	36	75000	1	36	Madrid	36	

	INGRESOS_LOWER	HIJOS_LOWER	PRUEBA_LOWER
0	50000	2	25
1	80000	0	32
2	65000	1	45
3	45000	0	28
4	70000	3	37
5	55000	2	22
6	48000	1	29
7	82000	0	31
8	60000	2	24
9	75000	1	36

[254]:
```
cols_to_update = df.filter(like='LOWER').columns
df.drop(cols_to_update, axis = 1, inplace=True)
df
```

[254]:

	Ciudad	Edad	Ingresos	Hijos	prueba
0	Madrid	25	50000	2	25
1	Barcelona	32	80000	0	32
2	Sevilla	45	65000	1	45
3	Valencia	28	45000	0	28
4	Madrid	37	70000	3	37
5	Sevilla	22	55000	2	22
6	Valencia	29	48000	1	29
7	Barcelona	31	82000	0	31
8	Sevilla	24	60000	2	24
9	Madrid	36	75000	1	36

[213]:
```
# Manejamos los tipos de datos

df['prueba'] = df.Edad.to_string()
print(df.dtypes)
df['prueba'] = df.Edad.astype(int)
print(df.dtypes)
```

```
Ciudad      object
Edad         int64
Ingresos     int64
Hijos        int64
prueba      object
dtype: object
Ciudad      object
Edad         int64
Ingresos     int64
Hijos        int64
prueba       int64
dtype: object
```

[217]:
```
# Contamos cuantas ciudades diferentes hay

df.Ciudad.unique()
```

[217]:
```
array(['Madrid', 'Barcelona', 'Sevilla', 'Valencia'], dtype=object)
```

[224]:
```
# Cuantas veces se repite cada ciudad
#help(pd.DataFrame.value_counts)

df.Ciudad.value_counts(normalize=True) # como %
```

[224]:
```
Madrid       0.3
Sevilla      0.3
Barcelona    0.2
Valencia     0.2
Name: Ciudad, dtype: float64
```

[230]:
```
# Cuantas obs hay si ciudad en madrid

len(df[df['Ciudad'] == "Madrid"])
```

[230]:
```
3
```

[255]:
```
# Cuantas veces ciudad es missing

len(df[df['Ciudad'].isnull()])
```

[255]: 0

[ ]:

2 1. Estandarización

En Python, el proceso de estandarización se refiere a la transformación de los datos de tal manera que tengan una media igual a cero y una desviación estándar igual a uno. Esto se logra aplicando la siguiente fórmula a cada valor del conjunto de datos:

z = (x - u) / s

donde z es el valor estandarizado, x es el valor original, u es la media del conjunto de datos y s es la desviación estándar del conjunto de datos.

En Python, se puede utilizar la biblioteca Scikit-learn para realizar el proceso de estandarización. La clase StandardScaler de Scikit-learn se utiliza para realizar la estandarización de las características de un conjunto de datos. Esta clase ajusta y transforma los datos de manera que tengan una media igual a cero y una desviación estándar igual a uno.

En este ejemplo, la columna 'A' del DataFrame se estandariza utilizando la clase StandardScaler. La columna 'A' se selecciona utilizando el doble corchete [[ ]] para indicar que queremos una matriz de características en lugar de una serie. La función fit_transform() se utiliza para ajustar y transformar los datos de la columna 'A' en una sola operación. La columna 'A' del DataFrame original se reemplaza por la columna estandarizada.

[29]:
```
# Definimos un df

df = pd.DataFrame({'A': [1, 2, 3], 'B': [10, 20, 30], 'C': [100, 200, 300]})
df
```

[29]:

	A	B	C
0	1	10	100
1	2	20	200
2	3	30	300

[30]:
```
# Creamos una variable nueva que es es la A estandarizada: media = 0. y dt = 1.

scaler = StandardScaler()
df['A2'] = scaler.fit_transform(df[['A']])
df
```

[30]:

	A	B	C	A2
0	1	10	100	-1.224745
1	2	20	200	0.000000
2	3	30	300	1.224745

[31]:
```
# Para todas la columnas

scaler = StandardScaler()

for j in list(df.columns):
    df[j + "2"] = scaler.fit_transform(df[[j]])

df
```

[31]:

	A	B	C	A2	B2	C2	A22
0	1	10	100	-1.224745	-1.224745	-1.224745	-1.224745
1	2	20	200	0.000000	0.000000	0.000000	0.000000
2	3	30	300	1.224745	1.224745	1.224745	1.224745

3 2. Normalización

En Python, el proceso de normalización se refiere a la transformación de los datos de tal manera que estén en un rango específico o tengan una distribución específica. Normalmente, la normalización se utiliza para escalar los datos a un rango específico, por ejemplo, entre 0 y 1, o para hacer que la suma de los valores de los datos sea igual a 1.

En Python, se pueden utilizar varias técnicas de normalización, como la normalización por escala mínima y máxima (MinMaxScaler), la normalización por norma (Normalizer) y la normalización por sumas (sumas parciales) acumuladas (QuantileTransformer), entre otras.

La técnicas Normalizer: Es importante tener en cuenta que la normalización con la clase Normalizer escala cada fila de la columna para que su norma (longitud) sea igual a 1. En otras palabras, cada valor de la columna se divide por la norma de la fila a la que pertenece. El resultado es una columna normalizada donde cada valor tiene una longitud de 1.

[32]:
```
scaler = MinMaxScaler()
df['A3'] = scaler.fit_transform(df[['A']])
df
```

[32]:

	A	B	C	A2	B2	C2	A22	A3
0	1	10	100	-1.224745	-1.224745	-1.224745	-1.224745	0.0
1	2	20	200	0.000000	0.000000	0.000000	0.000000	0.5
2	3	30	300	1.224745	1.224745	1.224745	1.224745	1.0

[33]:
```
normalizer = Normalizer()
df['A_normalized'] = normalizer.fit_transform(df[['A']])
df
```

[33]:

	A	B	C	A2	B2	C2	A22	A3	A_normalized
0	1	10	100	-1.224745	-1.224745	-1.224745	-1.224745	0.0	1.0
1	2	20	200	0.000000	0.000000	0.000000	0.000000	0.5	1.0
2	3	30	300	1.224745	1.224745	1.224745	1.224745	1.0	1.0

[ ]:

4 3. Variables categóricas

[ ]:

4.1 3.1 Variable categorica a binaria

[41]:
```
# Generamos un dataframe

# Crear un DataFrame con diferentes variables y ciudades repetidas
data = {'Ciudad': ['Madrid', 'Barcelona', 'Sevilla', 'Valencia', 'Madrid',␣
↪'Sevilla', 'Valencia', 'Barcelona', 'Sevilla', 'Madrid'],
        'Edad': [25, 32, 45, 28, 37, 22, 29, 31, 24, 36],
        'Ingresos': [50000, 80000, 65000, 45000, 70000, 55000, 48000, 82000,␣
↪60000, 75000],
        'Hijos': [2, 0, 1, 0, 3, 2, 1, 0, 2, 1]}

df = pd.DataFrame(data)

# Imprimir el DataFrame
df
```

[41]:

	Ciudad	Edad	Ingresos	Hijos
0	Madrid	25	50000	2
1	Barcelona	32	80000	0
2	Sevilla	45	65000	1
3	Valencia	28	45000	0
4	Madrid	37	70000	3
5	Sevilla	22	55000	2
6	Valencia	29	48000	1
7	Barcelona	31	82000	0
8	Sevilla	24	60000	2
9	Madrid	36	75000	1

[42]:
```
# Crear una instancia de OneHotEncoder
encoder = OneHotEncoder(sparse=False)

# Ajustar y transformar el codificador a la columna "Ciudad"
ciudades_encoded = encoder.fit_transform(df[["Ciudad"]])

# Crear un DataFrame con las variables codificadas
df_encoded = pd.DataFrame(ciudades_encoded, columns=encoder.
↪get_feature_names(['Ciudad']))
df_encoded
```

[42]:

	Ciudad_Barcelona	Ciudad_Madrid	Ciudad_Sevilla	Ciudad_Valencia
0	0.0	1.0	0.0	0.0
1	1.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	1.0
4	0.0	1.0	0.0	0.0
5	0.0	0.0	1.0	0.0
6	0.0	0.0	0.0	1.0
7	1.0	0.0	0.0	0.0
8	0.0	0.0	1.0	0.0
9	0.0	1.0	0.0	0.0

[43]:
```
# Concatenar los DataFrames "df" y "df_encoded"
df_final = pd.concat([df, df_encoded], axis=1)
df_final # hacemos un concaat de axis columna, unir por el lado, no por abajo.
# si queremos borrar ciudad: df_final.drop('Ciudad', axis=1, inplace=True)
```

[43]:

	Ciudad	Edad	Ingresos	Hijos	Ciudad_Barcelona	Ciudad_Madrid	\
0	Madrid	25	50000	2	0.0	1.0	
1	Barcelona	32	80000	0	1.0	0.0	
2	Sevilla	45	65000	1	0.0	0.0	
3	Valencia	28	45000	0	0.0	0.0	
4	Madrid	37	70000	3	0.0	1.0	
5	Sevilla	22	55000	2	0.0	0.0	
6	Valencia	29	48000	1	0.0	0.0	
7	Barcelona	31	82000	0	1.0	0.0	
8	Sevilla	24	60000	2	0.0	0.0	
9	Madrid	36	75000	1	0.0	1.0	

	Ciudad_Sevilla	Ciudad_Valencia
0	0.0	0.0
1	0.0	0.0
2	1.0	0.0
3	0.0	1.0
4	0.0	0.0
5	1.0	0.0
6	0.0	1.0
7	0.0	0.0
8	1.0	0.0
9	0.0	0.0

[ ]:

4.2 3.2 Variable categorica a continua

[46]:
```
# Generamos el df

data = {'Edad': [30, 25, 42, 37, 21, 46, 29, 38, 31, 27],
        'Ingresos': [75000, 30000, 45000, 80000, 20000, 90000, 55000, 72000,␣
 ↪68000, 36000],
        'Hijos': [2, 0, 1, 3, 0, 2, 1, 2, 1, 0],
        'Nivel económico': np.random.choice(['rico', 'pobre', 'normal'],␣
 ↪size=10)}

df = pd.DataFrame(data)

# Imprimir el DataFrame
df
```

[46]:

	Edad	Ingresos	Hijos	Nivel económico
0	30	75000	2	pobre
1	25	30000	0	rico
2	42	45000	1	pobre
3	37	80000	3	normal
4	21	20000	0	normal
5	46	90000	2	rico
6	29	55000	1	pobre
7	38	72000	2	pobre
8	31	68000	1	rico
9	27	36000	0	pobre

[47]:
```
# Lo convertimos en variable categorica numerica
mapping = {'pobre': 1, 'normal': 2, 'rico': 3}

# Utilizar el método map() para convertir la variable "Nivel económico" a␣
 ↪numérica
df['Nivel económico2'] = df['Nivel económico'].map(mapping)
df
```

[47]:

	Edad	Ingresos	Hijos	Nivel económico	Nivel económico2
0	30	75000	2	pobre	1
1	25	30000	0	rico	3
2	42	45000	1	pobre	1
3	37	80000	3	normal	2
4	21	20000	0	normal	2
5	46	90000	2	rico	3
6	29	55000	1	pobre	1
7	38	72000	2	pobre	1
8	31	68000	1	rico	3
9	27	36000	0	pobre	1

[48]:
```
# LO vamos a hacer igual pero sin elegir nosotros el orden: da orden segun␣
↪aparezca en el df
df['Nivel económico3'], _ = pd.factorize(df['Nivel económico'])
df
```

[48]:

	Edad	Ingresos	Hijos	Nivel económico	Nivel económico2	Nivel económico3
0	30	75000	2	pobre	1	0
1	25	30000	0	rico	3	1
2	42	45000	1	pobre	1	0
3	37	80000	3	normal	2	2
4	21	20000	0	normal	2	2
5	46	90000	2	rico	3	1
6	29	55000	1	pobre	1	0
7	38	72000	2	pobre	1	0
8	31	68000	1	rico	3	1
9	27	36000	0	pobre	1	0

[ ]:
```

```

5 4. Valores Missing

[65]:
```
# Generamos un dataframe

# Definir los valores para la variable 'año' y 'vendedor'
años = [2018, 2019, 2020, 2021]
vendedores = ['Juan', 'Pedro', 'María', 'Laura', 'Paco']

# Crear una lista de diccionarios con los datos para cada fila
data = []
for i in range(20):
    fila = {'año': np.random.choice(años),
            'vendedor': np.random.choice(vendedores),
            'ventas': np.random.choice([100, 200, np.nan])}
    data.append(fila)

# Crear el DataFrame a partir de la lista de diccionarios
df = pd.DataFrame(data)

# Imprimir el DataFrame resultantedf
df = df.sort_values(by=['vendedor', 'año'])
df ## como vemos este df no vale, pero vamos a colapsarle para aprender cosas␣
↪nuevas
```

[65]:

	año	vendedor	ventas
12	2020	Juan	200.0
13	2020	Juan	NaN

```
5   2018    Laura   100.0
6   2019    Laura   200.0
0   2020    Laura     NaN
11  2021    Laura   200.0
8   2018    María     NaN
16  2019    María   100.0
10  2020    María   100.0
17  2020    María   100.0
18  2020    María   200.0
15  2021    María   200.0
4   2019     Paco   100.0
2   2020     Paco   100.0
7   2020     Paco   200.0
14  2020     Paco     NaN
19  2020     Paco     NaN
1   2021     Paco   200.0
3   2018    Pedro   100.0
9   2018    Pedro   200.0
```

[66]:
```
# Vemos que en el caso de arriba, si se vuleve a eje utar cambiare, la fila 10
 ↪y 17 es la misma
# eliminamos duplicados en termino de todas las variables

df = df.drop_duplicates()
#df = df.drop_duplicates(subset=['variable1', 'variable2'], inplace=True) //
 ↪esto para hacerlo en termino de varias variables
df ## solo ha quedado la fila 10
```

[66]:
```
      año vendedor  ventas
12  2020     Juan   200.0
13  2020     Juan     NaN
5   2018    Laura   100.0
6   2019    Laura   200.0
0   2020    Laura     NaN
11  2021    Laura   200.0
8   2018    María     NaN
16  2019    María   100.0
10  2020    María   100.0
18  2020    María   200.0
15  2021    María   200.0
4   2019     Paco   100.0
2   2020     Paco   100.0
7   2020     Paco   200.0
14  2020     Paco     NaN
1   2021     Paco   200.0
3   2018    Pedro   100.0
9   2018    Pedro   200.0
```

[60]:
```
# Hacemos el collapse

df_co = df.groupby(['vendedor', 'año']).agg({'ventas' : 'sum'}) # vemos que el␣
 ↪NA lo trata como 0
df_co
```

[60]:

		ventas
vendedor	año	
Juan	2019	0.0
	2020	0.0
	2021	0.0
Laura	2018	100.0
	2019	100.0
María	2019	100.0
	2020	500.0
	2021	500.0
Paco	2018	200.0
	2019	0.0
	2021	200.0
Pedro	2018	200.0
	2019	300.0

[62]:
```
# Ponemos bien el indice

df = df_co.reset_index()
df
```

[62]:

	vendedor	año	ventas
0	Juan	2019	0.0
1	Juan	2020	0.0
2	Juan	2021	0.0
3	Laura	2018	100.0
4	Laura	2019	100.0
5	María	2019	100.0
6	María	2020	500.0
7	María	2021	500.0
8	Paco	2018	200.0
9	Paco	2019	0.0
10	Paco	2021	200.0
11	Pedro	2018	200.0
12	Pedro	2019	300.0

[ ]:
```

```

[78]:
```
# Ahora si definimos un dataframe para trabjar con missing
import pandas as pd
import numpy as np
```

```
# Generamos un DataFrame con 10 filas y 3 columnas
df = pd.DataFrame(np.random.randn(10, 3), columns=['A', 'B', 'C'])

# Colocamos algunos valores como "missing"
df.iloc[0:3, 1] = np.nan
df.iloc[4, 2] = np.nan
df.iloc[7:9, 0] = np.nan

df
```

[78]:

	A	B	C
0	0.237728	NaN	-0.927959
1	-0.379252	NaN	0.144797
2	-0.483554	NaN	1.030567
3	0.921447	0.410015	-0.144146
4	0.617002	-0.653735	NaN
5	-0.834674	0.770177	-0.480188
6	-1.637227	-0.503851	-0.781750
7	NaN	0.311699	-0.364551
8	NaN	-0.449389	-0.427029
9	0.477334	-0.556738	-0.885308

[79]:
```
# Vemos cuantos missing hay en cada avariable
df.isnull().sum()
```

[79]:
```
A    2
B    3
C    1
dtype: int64
```

[80]:
```
# que me diga el % en cada variable
df.isnull().sum(axis=0) / len(df)
```

[80]:
```
A    0.2
B    0.3
C    0.1
dtype: float64
```

[81]:
```
# identificamos los valores missing de la variable B
df['A_missing'] = df['A'].isnull().astype(int)
df
```

[81]:

	A	B	C	A_missing
0	0.237728	NaN	-0.927959	0
1	-0.379252	NaN	0.144797	0
2	-0.483554	NaN	1.030567	0

```
3  0.921447  0.410015 -0.144146          0
4  0.617002 -0.653735       NaN          0
5 -0.834674  0.770177 -0.480188          0
6 -1.637227 -0.503851 -0.781750          0
7       NaN  0.311699 -0.364551          1
8       NaN -0.449389 -0.427029          1
9  0.477334 -0.556738 -0.885308          0
```

[ ]:

[83]:
```
# Eliminamos todos los missing de un df
df_clean = df.dropna()
df_clean
```

[83]:
```
          A         B         C  A_missing
3  0.921447  0.410015 -0.144146          0
5 -0.834674  0.770177 -0.480188          0
6 -1.637227 -0.503851 -0.781750          0
9  0.477334 -0.556738 -0.885308          0
```

[86]:
```
# eliminar solo los missing de una variable
df.dropna(subset=['C'], inplace=True)
df
```

[86]:
```
          A         B         C  A_missing
0  0.237728       NaN -0.927959          0
1 -0.379252       NaN  0.144797          0
2 -0.483554       NaN  1.030567          0
3  0.921447  0.410015 -0.144146          0
5 -0.834674  0.770177 -0.480188          0
6 -1.637227 -0.503851 -0.781750          0
7       NaN  0.311699 -0.364551          1
8       NaN -0.449389 -0.427029          1
9  0.477334 -0.556738 -0.885308          0
```

[87]:
```
# remplazamos los valores de B por la mediana de esa variable
median_B = df['B'].median()
df_imputed = df.fillna(value={'B': median_B})
df_imputed
```

[87]:
```
          A         B         C  A_missing
0  0.237728 -0.068845 -0.927959          0
1 -0.379252 -0.068845  0.144797          0
2 -0.483554 -0.068845  1.030567          0
3  0.921447  0.410015 -0.144146          0
5 -0.834674  0.770177 -0.480188          0
6 -1.637227 -0.503851 -0.781750          0
```

```
7       NaN  0.311699 -0.364551          1
8       NaN -0.449389 -0.427029          1
9  0.477334 -0.556738 -0.885308          0
```

[ ]:

5.1 4.2 Un poco mas complejo

[106]:
```
import pandas as pd
import numpy as np

# Crear un DataFrame con todas las combinaciones de nombres, años y meses
nombres = ['Juan', 'Pedro', 'María', 'Ana']
anios = range(2020, 2023)
meses = range(1, 13)
df = pd.DataFrame([(nombre, np.random.choice([np.nan, np.random.normal(loc=100,␣
 ↪scale=20)], p=[0.1, 0.9]), anio, mes)
                   for nombre in nombres
                   for anio in anios
                   for mes in meses],
                  columns=['nombre', 'ventas', 'anio', 'mes'])

# Asegurarnos de que solo hay un registro por mes para cada nombre
df = df.drop_duplicates(subset=['nombre', 'anio', 'mes'])
df
```

[106]:
```
    nombre      ventas  anio  mes
0     Juan  102.591798  2020    1
1     Juan  129.116054  2020    2
2     Juan  106.800738  2020    3
3     Juan   81.658448  2020    4
4     Juan   97.426364  2020    5
..     …         …    …   …
139    Ana   94.305569  2022    8
140    Ana  127.727286  2022    9
141    Ana  101.720330  2022   10
142    Ana   83.071366  2022   11
143    Ana   83.365152  2022   12

[144 rows x 4 columns]
```

[107]:
```
# Creamos variable fecha_mes
df['fecha_mes'] = df.apply(lambda row: str(row['anio']) + '-' +␣
 ↪str(row['mes']), axis=1)
df['fecha_mes'] = pd.to_datetime(df['fecha_mes'], format='%Y-%m')
print(df.dtypes)
```

```
df
```

```
nombre               object
ventas              float64
anio                  int64
mes                   int64
fecha_mes    datetime64[ns]
dtype: object
```

[107]:

	nombre	ventas	anio	mes	fecha_mes
0	Juan	102.591798	2020	1	2020-01-01
1	Juan	129.116054	2020	2	2020-02-01
2	Juan	106.800738	2020	3	2020-03-01
3	Juan	81.658448	2020	4	2020-04-01
4	Juan	97.426364	2020	5	2020-05-01
..	…	…	…	…	…
139	Ana	94.305569	2022	8	2022-08-01
140	Ana	127.727286	2022	9	2022-09-01
141	Ana	101.720330	2022	10	2022-10-01
142	Ana	83.071366	2022	11	2022-11-01
143	Ana	83.365152	2022	12	2022-12-01

[144 rows x 5 columns]

[108]:
```
# % de missing en ventas
df['ventas'].isnull().sum(axis=0) / len(df)
```

[108]: 0.125

[112]:
```
# Identificamos donde están los missing
df['ventas_missing'] = df['ventas'].isnull().astype(int)
df[df['nombre'] == "Ana"]
```

[112]:

	nombre	ventas	anio	mes	fecha_mes	ventas_missing
108	Ana	107.739415	2020	1	2020-01-01	0
109	Ana	104.044413	2020	2	2020-02-01	0
110	Ana	133.356275	2020	3	2020-03-01	0
111	Ana	103.731302	2020	4	2020-04-01	0
112	Ana	NaN	2020	5	2020-05-01	1
113	Ana	129.125971	2020	6	2020-06-01	0
114	Ana	121.192110	2020	7	2020-07-01	0
115	Ana	67.627211	2020	8	2020-08-01	0
116	Ana	80.283064	2020	9	2020-09-01	0
117	Ana	99.667184	2020	10	2020-10-01	0
118	Ana	NaN	2020	11	2020-11-01	1
119	Ana	84.587330	2020	12	2020-12-01	0
120	Ana	88.387078	2021	1	2021-01-01	0

121	Ana	71.584039	2021	2	2021-02-01	0
122	Ana	80.742256	2021	3	2021-03-01	0
123	Ana	NaN	2021	4	2021-04-01	1
124	Ana	137.534574	2021	5	2021-05-01	0
125	Ana	92.778526	2021	6	2021-06-01	0
126	Ana	111.742369	2021	7	2021-07-01	0
127	Ana	71.681623	2021	8	2021-08-01	0
128	Ana	86.075937	2021	9	2021-09-01	0
129	Ana	105.211592	2021	10	2021-10-01	0
130	Ana	NaN	2021	11	2021-11-01	1
131	Ana	85.836705	2021	12	2021-12-01	0
132	Ana	52.652914	2022	1	2022-01-01	0
133	Ana	114.110300	2022	2	2022-02-01	0
134	Ana	103.898079	2022	3	2022-03-01	0
135	Ana	91.140682	2022	4	2022-04-01	0
136	Ana	75.803545	2022	5	2022-05-01	0
137	Ana	NaN	2022	6	2022-06-01	1
138	Ana	NaN	2022	7	2022-07-01	1
139	Ana	94.305569	2022	8	2022-08-01	0
140	Ana	127.727286	2022	9	2022-09-01	0
141	Ana	101.720330	2022	10	2022-10-01	0
142	Ana	83.071366	2022	11	2022-11-01	0
143	Ana	83.365152	2022	12	2022-12-01	0

[122]:
```
df['imputacion'] = df.groupby(['nombre', 'anio'])['ventas'].transform('mean')
df[(df['nombre'] == "Ana") & (df['anio'] == 2020)] ## no cuenta los missing␣
 ↪para el calculo de la media
```

[122]:

	nombre	ventas	anio	mes	fecha_mes	ventas_missing	imputacion
108	Ana	107.739415	2020	1	2020-01-01	0	103.135428
109	Ana	104.044413	2020	2	2020-02-01	0	103.135428
110	Ana	133.356275	2020	3	2020-03-01	0	103.135428
111	Ana	103.731302	2020	4	2020-04-01	0	103.135428
112	Ana	NaN	2020	5	2020-05-01	1	103.135428
113	Ana	129.125971	2020	6	2020-06-01	0	103.135428
114	Ana	121.192110	2020	7	2020-07-01	0	103.135428
115	Ana	67.627211	2020	8	2020-08-01	0	103.135428
116	Ana	80.283064	2020	9	2020-09-01	0	103.135428
117	Ana	99.667184	2020	10	2020-10-01	0	103.135428
118	Ana	NaN	2020	11	2020-11-01	1	103.135428
119	Ana	84.587330	2020	12	2020-12-01	0	103.135428

[124]:
```
df['ventas'] = df['ventas'].fillna(df['imputacion'])
df[(df['nombre'] == "Ana") & (df['anio'] == 2020)]
```

[124]:

	nombre	ventas	anio	mes	fecha_mes	ventas_missing	imputacion
108	Ana	107.739415	2020	1	2020-01-01	0	103.135428

```
109    Ana  104.044413  2020    2 2020-02-01              0  103.135428
110    Ana  133.356275  2020    3 2020-03-01              0  103.135428
111    Ana  103.731302  2020    4 2020-04-01              0  103.135428
112    Ana  103.135428  2020    5 2020-05-01              1  103.135428
113    Ana  129.125971  2020    6 2020-06-01              0  103.135428
114    Ana  121.192110  2020    7 2020-07-01              0  103.135428
115    Ana   67.627211  2020    8 2020-08-01              0  103.135428
116    Ana   80.283064  2020    9 2020-09-01              0  103.135428
117    Ana   99.667184  2020   10 2020-10-01              0  103.135428
118    Ana  103.135428  2020   11 2020-11-01              1  103.135428
119    Ana   84.587330  2020   12 2020-12-01              0  103.135428
```

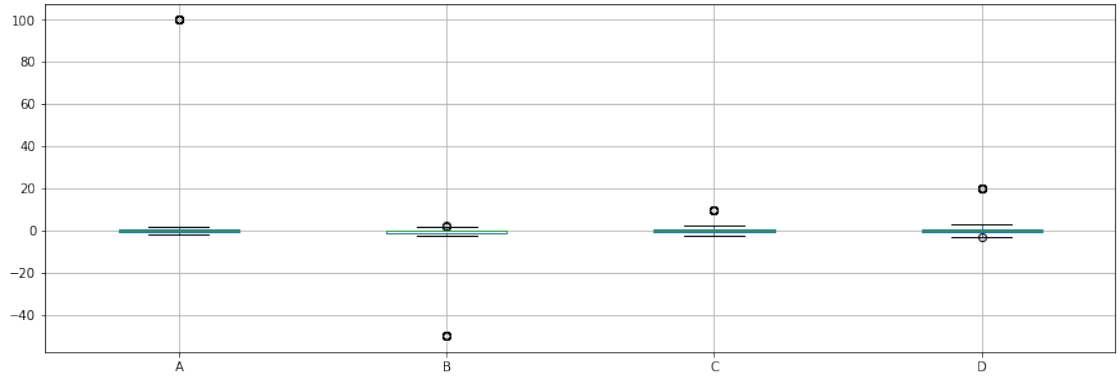
[ ]:

6 5. Valores atipicos

[176]:
```
# Generamos el dataframe

import pandas as pd
import numpy as np

# Creamos un DataFrame con 4 columnas: 'A', 'B', 'C' y 'D'
np.random.seed(123)
data = pd.DataFrame(np.random.randn(100, 4), columns=['A', 'B', 'C', 'D'])

# Agregamos algunos valores atípicos
data.iloc[0:10, 0] = 100
data.iloc[10:20, 1] = -50
data.iloc[20:30, 2] = 10
data.iloc[30:40, 3] = 20
data
```

[176]:
```
             A         B         C         D
0   100.000000  0.997345  0.282978 -1.506295
1   100.000000  1.651437 -2.426679 -0.428913
2   100.000000 -0.866740 -0.678886 -0.094709
3   100.000000 -0.638902 -0.443982 -0.434351
4   100.000000  2.186786  1.004054  0.386186
..         …       …         …         …
95    0.789828 -1.007509 -1.305786 -0.882829
96   -0.346090  0.109403 -0.772584  0.744819
97    0.251464 -0.694798  0.888993  1.161068
98   -0.098685 -0.214983 -1.773771 -0.407513
99   -0.291507  0.245379 -0.168426  0.244027

[100 rows x 4 columns]
```

[177]:
```
# Visualizamos la distribución
plt.figure(figsize=(15, 5))
data.boxplot()
plt.show() # vemos que hay outliers
```



[178]:
```
# A través de estadisticos
print(data.mean())
print(data.std())
```

```
A    10.009487
B    -5.134458
C     0.904557
D     1.891563
dtype: float64
A    30.159221
B    15.054684
C     3.210436
D     6.153953
dtype: float64
```

[179]:
```
# Detección por mas de 2 devia iones tipicaS. Sería mejor poner 3 pero en este␣
 ↪caso no hay
# IMPORTANTEEEEEEEEEEEEEE: mira como crear el valor de cada columna␣
 ↪f"{col}_outlier"
for col in data.columns:
    mean = data[col].mean()
    std = data[col].std()
    data[f"{col}_outlier"] = ((data[col] < mean - 2 * std) | (data[col] > mean␣
 ↪+ 2 * std)).astype(int)
data[data['A_outlier'] == 1]
```

[179]:

	A	B	C	D	A_outlier	B_outlier	C_outlier	\
0	100.0	0.997345	0.282978	-1.506295	1	0	0	

```
1  100.0  1.651437 -2.426679 -0.428913          1          0          0
2  100.0 -0.866740 -0.678886 -0.094709          1          0          0
3  100.0 -0.638902 -0.443982 -0.434351          1          0          0
4  100.0  2.186786  1.004054  0.386186          1          0          0
5  100.0  1.490732 -0.935834  1.175829          1          0          0
6  100.0 -0.637752  0.907105 -1.428681          1          0          0
7  100.0 -0.861755 -0.255619 -2.798589          1          0          0
8  100.0 -0.699877  0.927462 -0.173636          1          0          0
9  100.0  0.688223 -0.879536  0.283627          1          0          0

   D_outlier
0          0
1          0
2          0
3          0
4          0
5          0
6          0
7          0
8          0
9          0
```

[180]:
```
# Borramos las obs donde A es outlier
data.drop(data[data['A_outlier'] == 1].index, inplace=True)
data.drop('A_outlier', axis =1, inplace=True)
data
```

[180]:
```
           A          B         C         D  B_outlier  C_outlier  D_outlier
10 -0.805367 -50.000000 -0.390900  0.573806          1          0          0
11  0.338589 -50.000000  2.392365  0.412912          1          0          0
12  0.978736 -50.000000 -1.294085 -1.038788          1          0          0
13  1.743712 -50.000000  0.029683  1.069316          1          0          0
14  0.890706 -50.000000  1.495644  1.069393          1          0          0
..       …        …        …        …         …         …          …
95  0.789828  -1.007509 -1.305786 -0.882829          0          0          0
96 -0.346090   0.109403 -0.772584  0.744819          0          0          0
97  0.251464  -0.694798  0.888993  1.161068          0          0          0
98 -0.098685  -0.214983 -1.773771 -0.407513          0          0          0
99 -0.291507   0.245379 -0.168426  0.244027          0          0          0

[90 rows x 7 columns]
```

[181]:
```
# Ahora queremos borrar todo donde haya outlier
cols_out = data.filter(like='out').columns
data['out'] = data[cols_out].sum(axis=1)
data.drop(data[data['out'] > 0].index, inplace=True)
data
```

[181]:

	A	B	C	D	B_outlier	C_outlier	D_outlier
40	0.020316	-0.193964	0.134027	0.704474	0	0	0
41	0.665653	-0.898423	1.523664	-1.095026	0	0	0
42	0.079227	-0.274397	-1.048992	-0.075121	0	0	0
43	-0.740814	0.072907	0.403086	1.471929	0	0	0
44	0.307384	-0.611225	-0.391620	0.139978	0	0	0
45	0.093461	1.459589	1.395353	-0.358936	0	0	0
46	-0.548642	-2.557055	-0.548920	-0.978058	0	0	0
47	-0.354824	0.391584	0.177192	-0.029968	0	0	0
48	0.199582	-0.126118	0.197019	-3.231055	0	0	0
49	-0.269293	-0.110851	-0.341262	-0.217946	0	0	0
50	0.703310	-0.598105	2.200702	0.688297	0	0	0
51	-0.006307	-0.206662	-0.086522	-0.915307	0	0	0
52	-0.095203	0.278684	0.579542	0.579690	0	0	0
53	-0.274878	-1.416082	-0.669103	1.612193	0	0	0
54	0.896058	0.369620	-0.761294	0.003645	0	0	0
55	-1.255669	-0.551937	-0.245203	-0.361640	0	0	0
56	0.956602	-1.418726	-0.865432	-1.374688	0	0	0
57	-1.237353	0.124056	-1.600441	0.753869	0	0	0
58	-0.246816	0.068788	0.322577	-0.434167	0	0	0
59	1.032480	-0.194343	0.594070	-0.199112	0	0	0
60	0.290874	0.279663	0.249970	-0.974308	0	0	0
61	0.435876	-0.318957	0.630488	-2.152493	0	0	0
62	-1.465116	0.363446	1.862928	0.835059	0	0	0
63	-0.682451	-1.692052	0.742686	-0.080583	0	0	0
64	0.590704	0.115299	0.029643	2.958625	0	0	0
65	-0.006130	-0.159245	-0.121449	-0.583537	0	0	0
66	0.990133	-0.353754	0.635943	0.284603	0	0	0
67	1.218986	0.420180	-1.213385	-1.326488	0	0	0
68	1.408369	-0.608711	-1.320603	-0.669619	0	0	0
69	1.264625	-1.420213	-0.866495	-0.666808	0	0	0
70	-1.251190	-1.184327	-1.518108	-0.461187	0	0	0
71	-0.354909	-0.682538	-1.653698	1.253336	0	0	0
72	-1.329079	0.278034	-1.074767	0.668317	0	0	0
73	0.955832	-0.877614	-1.923716	0.695787	0	0	0
74	1.875801	0.415695	0.160544	0.819761	0	0	0
75	0.765055	-0.828989	-0.659151	0.611124	0	0	0
76	-0.144013	1.316606	-0.704342	0.750610	0	0	0
77	0.342638	-0.126438	1.175911	0.680072	0	0	0
78	-1.004967	0.640219	1.374991	-0.130445	0	0	0
79	-0.248656	-0.669647	-0.013604	0.686201	0	0	0
80	-0.817668	-1.346358	-0.375750	-1.379725	0	0	0
81	0.523218	-0.426690	-1.755402	-0.348608	0	0	0
82	-0.192615	0.449136	-0.145364	1.868726	0	0	0
83	-0.518704	-0.062399	-0.102911	-0.282628	0	0	0
84	0.142426	0.541231	1.340099	-1.569256	0	0	0
85	-0.510343	-0.447771	0.937850	-0.356663	0	0	0

```
86 -1.895176  0.087730 -0.033689  0.179752          0          0          0
87 -1.040163  1.719035 -0.323860 -0.188297          0          0          0
88 -0.900009 -0.931002 -1.222737 -0.393311          0          0          0
89 -0.957582  2.056467 -1.888492 -1.128331          0          0          0
90 -0.401414  0.673491 -0.413757  0.675963          0          0          0
91 -0.986804  0.059291  1.744041 -0.967744          0          0          0
92  0.419568  0.206928 -2.251535 -0.588971          0          0          0
93  1.131152  0.135078 -1.212269  0.690777          0          0          0
94 -0.479123  0.360051  0.376920 -1.118696          0          0          0
95  0.789828 -1.007509 -1.305786 -0.882829          0          0          0
96 -0.346090  0.109403 -0.772584  0.744819          0          0          0
97  0.251464 -0.694798  0.888993  1.161068          0          0          0
98 -0.098685 -0.214983 -1.773771 -0.407513          0          0          0
99 -0.291507  0.245379 -0.168426  0.244027          0          0          0
```

```
    out
40    0
41    0
42    0
43    0
44    0
45    0
46    0
47    0
48    0
49    0
50    0
51    0
52    0
53    0
54    0
55    0
56    0
57    0
58    0
59    0
60    0
61    0
62    0
63    0
64    0
65    0
66    0
67    0
68    0
69    0
70    0
```

```
71    0
72    0
73    0
74    0
75    0
76    0
77    0
78    0
79    0
80    0
81    0
82    0
83    0
84    0
85    0
86    0
87    0
88    0
89    0
90    0
91    0
92    0
93    0
94    0
95    0
96    0
97    0
98    0
99    0
```

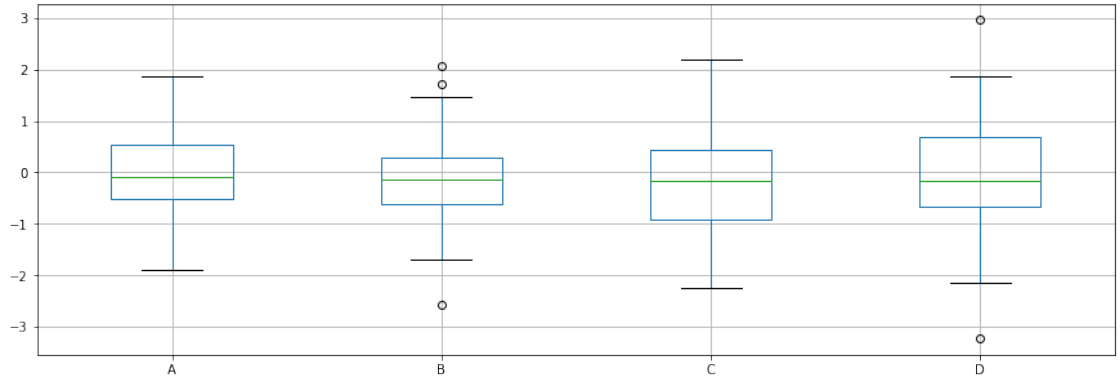
[182]:
```
data.drop(columns=data.filter(regex='out').columns, inplace=True)
data
```

[182]:

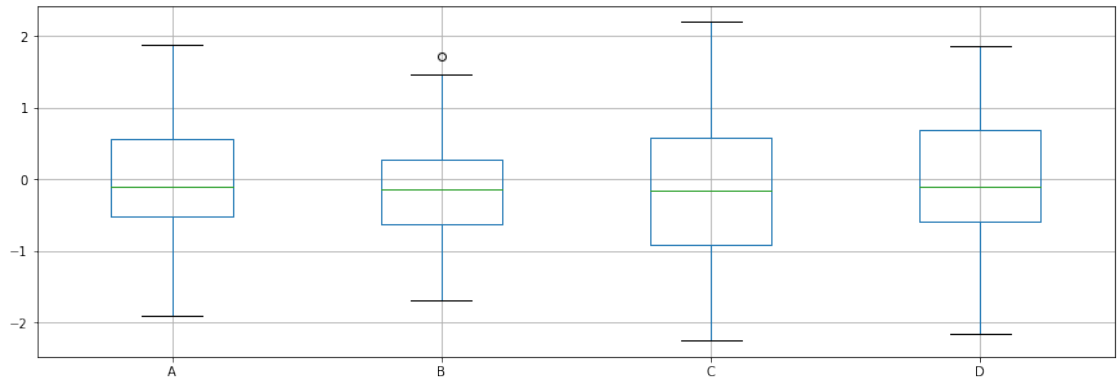
	A	B	C	D
40	0.020316	-0.193964	0.134027	0.704474
41	0.665653	-0.898423	1.523664	-1.095026
42	0.079227	-0.274397	-1.048992	-0.075121
43	-0.740814	0.072907	0.403086	1.471929
44	0.307384	-0.611225	-0.391620	0.139978
45	0.093461	1.459589	1.395353	-0.358936
46	-0.548642	-2.557055	-0.548920	-0.978058
47	-0.354824	0.391584	0.177192	-0.029968
48	0.199582	-0.126118	0.197019	-3.231055
49	-0.269293	-0.110851	-0.341262	-0.217946
50	0.703310	-0.598105	2.200702	0.688297
51	-0.006307	-0.206662	-0.086522	-0.915307
52	-0.095203	0.278684	0.579542	0.579690

```
53 -0.274878 -1.416082 -0.669103  1.612193
54  0.896058  0.369620 -0.761294  0.003645
55 -1.255669 -0.551937 -0.245203 -0.361640
56  0.956602 -1.418726 -0.865432 -1.374688
57 -1.237353  0.124056 -1.600441  0.753869
58 -0.246816  0.068788  0.322577 -0.434167
59  1.032480 -0.194343  0.594070 -0.199112
60  0.290874  0.279663  0.249970 -0.974308
61  0.435876 -0.318957  0.630488 -2.152493
62 -1.465116  0.363446  1.862928  0.835059
63 -0.682451 -1.692052  0.742686 -0.080583
64  0.590704  0.115299  0.029643  2.958625
65 -0.006130 -0.159245 -0.121449 -0.583537
66  0.990133 -0.353754  0.635943  0.284603
67  1.218986  0.420180 -1.213385 -1.326488
68  1.408369 -0.608711 -1.320603 -0.669619
69  1.264625 -1.420213 -0.866495 -0.666808
70 -1.251190 -1.184327 -1.518108 -0.461187
71 -0.354909 -0.682538 -1.653698  1.253336
72 -1.329079  0.278034 -1.074767  0.668317
73  0.955832 -0.877614 -1.923716  0.695787
74  1.875801  0.415695  0.160544  0.819761
75  0.765055 -0.828989 -0.659151  0.611124
76 -0.144013  1.316606 -0.704342  0.750610
77  0.342638 -0.126438  1.175911  0.680072
78 -1.004967  0.640219  1.374991 -0.130445
79 -0.248656 -0.669647 -0.013604  0.686201
80 -0.817668 -1.346358 -0.375750 -1.379725
81  0.523218 -0.426690 -1.755402 -0.348608
82 -0.192615  0.449136 -0.145364  1.868726
83 -0.518704 -0.062399 -0.102911 -0.282628
84  0.142426  0.541231  1.340099 -1.569256
85 -0.510343 -0.447771  0.937850 -0.356663
86 -1.895176  0.087730 -0.033689  0.179752
87 -1.040163  1.719035 -0.323860 -0.188297
88 -0.900009 -0.931002 -1.222737 -0.393311
89 -0.957582  2.056467 -1.888492 -1.128331
90 -0.401414  0.673491 -0.413757  0.675963
91 -0.986804  0.059291  1.744041 -0.967744
92  0.419568  0.206928 -2.251535 -0.588971
93  1.131152  0.135078 -1.212269  0.690777
94 -0.479123  0.360051  0.376920 -1.118696
95  0.789828 -1.007509 -1.305786 -0.882829
96 -0.346090  0.109403 -0.772584  0.744819
97  0.251464 -0.694798  0.888993  1.161068
98 -0.098685 -0.214983 -1.773771 -0.407513
99 -0.291507  0.245379 -0.168426  0.244027
```

[183]:
```
# Visualizamos la distribución
plt.figure(figsize=(15, 5))
data.boxplot()
plt.show() # vemos que hay outliers
```



[191]:
```
## Vemos que sigue habiendo puntos fuera en B y C

q = data[['B', 'D']].quantile(0.99)
p = data[['B', 'D']].quantile(0.01)
data = data[(data['B'] < q[0]) & (data['B'] > p[0] )]
data = data[(data['D'] < q[1]) & (data['D'] > p[1] )]
# Visualizamos la distribución
plt.figure(figsize=(15, 5))
data.boxplot()
plt.show() # vemos que hay outliers
```



[ ]: