

---

**Máster en Big Data y Data Science**

**05MBID - Minería de datos - Actividad guiada 2**

**Nombre: Antonio Manuel Palma Bautista**

**Fecha: 30 de junio de 2023**

**Curso 2023 - Ed. Abril**

---

## 1 Motivación y problema a resolver

**Contexto de partida:** una empresa de gestión de viviendas de alquiler turístico, sin presencia en Andalucía, quiere introducirse en la región invirtiendo en algunos municipios.

**Problema a resolver:** quiere mejorar su conocimiento del mercado turístico en Andalucía.

**Objetivo específico buscado:** elaborar un perfil del sector de las viviendas de alquiler turístico de cada municipio de la región, para comparar unos perfiles con otros y así detectar oportunidades de negocio.

**Resultado final pretendido en el KDD:** cuadros y gráficos que sirvan para que los directivos de la empresa, ajenos al mundo del análisis de datos, puedan entender la situación del mercado y comprender los patrones y tendencias que se siguen en él, identificando también aquellos municipios que se desvían de esos patrones (lo cual puede suponer la existencia de un nicho de mercado en esos municipios).

**Resultado buscado en la actividad:** se obtendrá la vista minable, quedando pendientes el resto de pasos hasta completar el KDD y obtener el conocimiento.

## ▼ 2 Pasos previos

### ▼ 2.1 Importación de librerías y funciones

```
import pandas as pd
import numpy as np
import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib
plt.style.use('ggplot')
from matplotlib.pyplot import figure

%matplotlib inline
matplotlib.rcParams['figure.figsize'] = (12,8)

pd.options.mode.chained_assignment = None
```

### ▼ 2.2 Importación del dataset

La tabla de datos que se va a utilizar se corresponde con el dataset completo en formato CSV de OpenRTA, el Registro de Turismo de Andalucía. Es un registro de acceso público disponible en el Portal de Datos Abiertos de la Junta de Andalucía, en <https://www.juntadeandalucia.es/datosabiertos/portal/dataset/openrta>.

El archivo con el que se va a trabajar se descargó comprimido en ZIP el día 11 de junio de 2023 y con datos actualizados a 25 de mayo de 2023 (según la fecha del fichero allopenRTA\_csv.csv contenido en el interior del fichero comprimido). Está publicada bajo la licencia [Reconocimiento 4.0 Internacional](#) (CC BY 4.0) de Creative Commons, aprobada para "obras culturales libres".

El archivo se importa desde Google Drive y se crea una copia para trabajar sobre ella:

```
from google.colab import drive
drive.mount('/content/drive')

# lectura de los datos
df_original = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/all-openRTA_csv.csv', low_memory=False)
df = df_original.copy()

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive"
```

### ▼ 3 Análisis inicial del dataset. Selección

Haz doble clic (o pulsa Intro) para editar

Se comprueba la cantidad de instancias de las que dispone el dataset:

```
print(df.shape)

(140048, 76)
```

Se observa que el conjunto de datos de partida tiene **140.048** filas y **76** columnas.

En la actividad 1 se agruparon los atributos en varias categorías:

- Atributos identificativos del establecimiento
- Atributos de contacto del establecimiento (con sus datos de contacto)
- Atributos exclusivos de algún (o algunos) tipo concreto de establecimiento
- Atributos de localización geográfica del establecimiento (coordenadas)
- Atributos sobre el seguro obligatorio del establecimiento
- Atributos identificativos del titular del establecimiento

Puesto que no aportan información para lograr el objetivo planteado, se eliminan los atributos identificativos del titular del establecimiento, los atributos sobre el seguro y los exclusivos de campamentos de turismo, oficinas de turismo u otros tipos de establecimiento que tampoco serán estudiados aquí.

```
to_drop_titular = ['ID_TIPOVIA', 'COD_VIA', 'NOMBRE_VIA', 'KM', 'CALIF_NUM', 'BLOQUE', 'PORTAL',
                  'ESCALERA', 'PISO', 'PUERTA', 'REF_CATASTRAL', 'IND_PUB_OPEN_RTA', 'COMPLEMENTODOM',
                  'ID_NUCLEO', 'KM_NUM', 'TIPO_NUMERACION', 'NUM_DOC_IDENTIFICATIVO', 'TITULAR',
                  'GRUPO_ID', 'CATEGORIA_ID', 'MODALIDAD_ID', 'LISTA_ESPEC']
to_drop_seguro = ['FEC_PRESENT_RTADSEG', 'ESTADO_PRESENT_RTADSEG', 'FEC_VERIF_RTADSEG',
                  'ESTADO_VERIF_RTADSEG', 'RESULT_VERIF_RTADSEG']
to_drop_exclusivos = ['IND_COMPARTIDA', 'ACTIVIDADES_TURISMO_ACTIVADO', 'IND_USO_PRIVADO',
                     'NUM_PARCELAS_ACAMPADA', 'NP_PARCELAS_ACAMPADA', 'SUP_ZONA_ACAMPADA',
                     'NUM_INSTALACIONES_FIJAS', 'NP_INSTALACIONES_FIJAS', 'SUPERFICIE_INSTALACIONES_FIJAS',
                     'TOTAL_PLAZAS_CAMPAMENTO', 'CAPACIDAD_MAXIMA_CAMPAMENTO', 'IND_OFICINA_TUR_INTEGRADA_RED',
                     'TITULARIDAD_OFICINA_TURISMO', 'PADRE_ID', 'IND_ESPECIFICO_ZONAL', 'IND_FIJO_MOVIL',
                     'IND_ON_LINE', 'URL', 'IDIOMAS']
```

```
df = df.drop(columns=to_drop_titular).drop(columns=to_drop_seguro).drop(columns=to_drop_exclusivos)
```

Una vez obtenido el conjunto de datos objetivo, se analiza la cantidad y el tipo de los datos de las instancias de que dispone:

```
print(df.shape)
print(df.dtypes)

(140048, 30)
RN                int64
ID                int64
COD_REGISTRO      object
NOMBRE            object
DESCRIPCION       object
FEC_INSCRIPCION   int64
FEC_INICIO_ACTIVIDAD float64
TIPO_OBJETO       object
TIPO_OBJETO_ID    int64
GRUPO             object
CATEGORIA         object
MODALIDAD         object
ESPECIALIDADES    object
DOMICILIO_ESTAB   object
CODIGO_POSTAL     object
LOCALIDAD         object
ID_MUNICIPIO      float64
MUNICIPIO         object
ID_PROVINCIA      float64
PROVINCIA         object
TELEFONO          object
MOVIL             object
FAX               object
CORREO_ELECTRONICO object
TOT_GEN_UA        float64
TOT_GEN_PLAZAS    float64
IND_TIPO_ALQUILER object
COORD_X           object
COORD_Y           object
SRID              float64
dtype: object
```

Ahora tiene **140.048** filas y **30** columnas.

Se analiza qué atributos son numéricos:

```
df_numeric = df.select_dtypes(include=[np.number])
numeric_cols = df_numeric.columns.values
print(numeric_cols)

['RN' 'ID' 'FEC_INSCRIPCION' 'FEC_INICIO_ACTIVIDAD' 'TIPO_OBJETO_ID'
 'ID_MUNICIPIO' 'ID_PROVINCIA' 'TOT_GEN_UA' 'TOT_GEN_PLAZAS' 'SRID']
```

Y cuáles no son numéricos:

```
df_non_numeric = df.select_dtypes(exclude=[np.number])
non_numeric_cols = df_non_numeric.columns.values
print(non_numeric_cols)

['COD_REGISTRO' 'NOMBRE' 'DESCRIPCION' 'TIPO_OBJETO' 'GRUPO' 'CATEGORIA'
 'MODALIDAD' 'ESPECIALIDADES' 'DOMICILIO_ESTAB' 'CODIGO_POSTAL'
 'LOCALIDAD' 'MUNICIPIO' 'PROVINCIA' 'TELEFONO' 'MOVIL' 'FAX'
 'CORREO_ELECTRONICO' 'IND_TIPO_ALQUILER' 'COORD_X' 'COORD_Y']
```

Ahora se puede revisar la lista de tipos de datos "sucios" y arreglarlos uno por uno.

## ▼ 4 Datos perdidos o faltantes

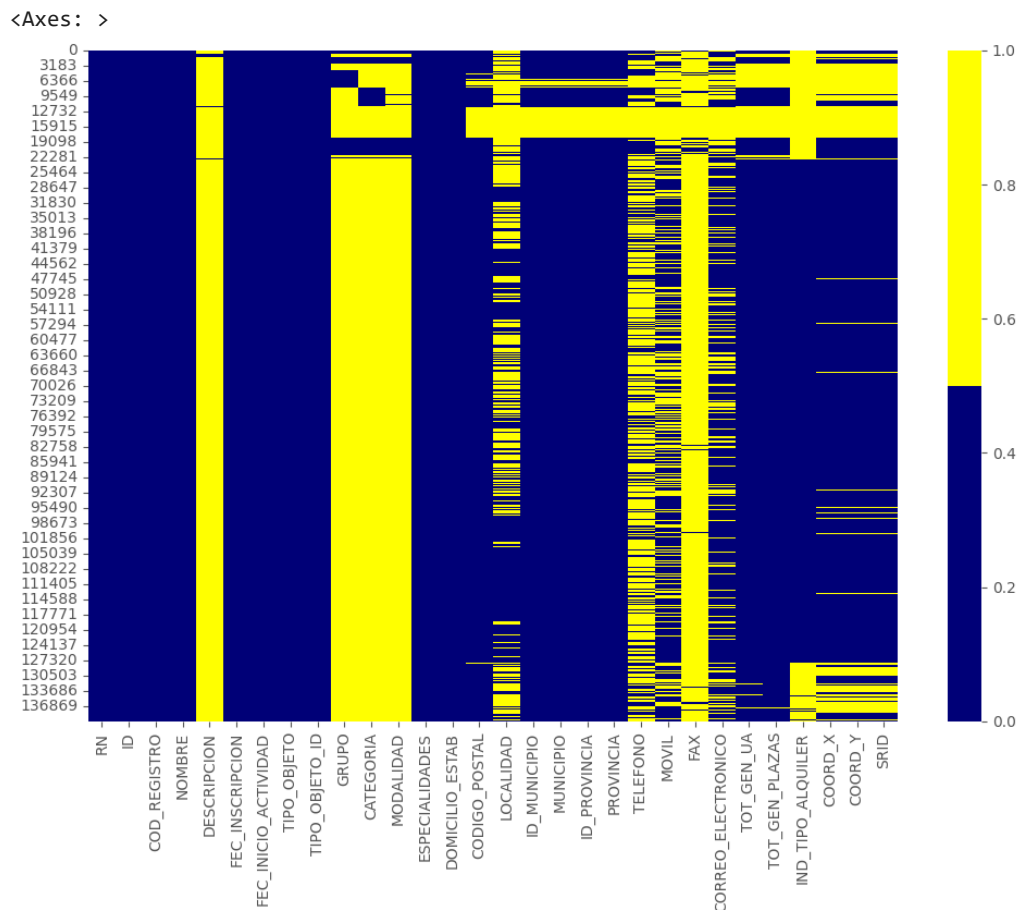
### ▼ 4.1 Análisis de los datos perdidos

Se analiza el conjunto de datos objetivo mediante tres técnicas para aprender más sobre los datos que le faltan:

#### ▼ 4.1.a Mapa de calor de datos perdidos

Como el número de atributos no es muy grande, 30, se pueden visualizar los datos que faltan a través de un mapa de calor:

```
colours = ['#000077', '#FFFF00'] # colores para el mapa de calor: azul, está / amarillo, es nulo
sns.heatmap(df.isnull(), cmap=sns.color_palette(colours))
```



El gráfico de arriba muestra los patrones de datos que faltan en el dataset objetivo, con el nombre del atributo de entrada en el eje horizontal y el número de registro en el vertical. el color amarillo representa los datos que faltan, mientras que el color azul los datos que sí están. Por ejemplo, se ve que el atributo GRUPO tiene valores perdidos en la mayoría filas mientras que NOMBRE sólo tiene algunos perdidos.

#### ▼ 4.1.b Lista de porcentaje de datos perdidos

Completando la información visual que ofrece el mapa de calor, en esta lista se muestra el porcentaje de valores faltantes para cada uno de los atributos:

```
for col in df.columns:
    pct_missing = np.mean(df[col].isnull())
    print('{} - {}'.format(col, round(pct_missing*100)))
```

```
RN - 0%
ID - 0%
COD_REGISTRO - 0%
NOMBRE - 1%
DESCRIPCION - 99%
FEC_INSCRIPCION - 0%
FEC_INICIO_ACTIVIDAD - 0%
TIPO_OBJETO - 0%
TIPO_OBJETO_ID - 0%
GRUPO - 93%
CATEGORIA - 93%
MODALIDAD - 95%
ESPECIALIDADES - 0%
DOMICILIO_ESTAB - 0%
CODIGO_POSTAL - 6%
LOCALIDAD - 51%
ID_MUNICIPIO - 5%
MUNICIPIO - 5%
ID_PROVINCIA - 5%
PROVINCIA - 5%
TELEFONO - 67%
MOVIL - 41%
FAX - 96%
CORREO_ELECTRONICO - 30%
TOT_GEN_UA - 10%
TOT_GEN_PLAZAS - 9%
IND_TIPO_ALQUILER - 25%
COORD_X - 17%
COORD_Y - 17%
SRID - 17%
```

Se ve que al atributo DESCRIPCION le falta el 99% de los datos, a GRUPO le falta el 93%, mientras que a NOMBRE sólo le falta el 1%.

#### ▼ 4.1.c Histograma de datos perdidos

Este histograma ayuda a identificar las situaciones de valores perdidos de las 140.048 observaciones y sus posibles patrones.

```
# Creación de una copia para no añadir nuevas columnas en df
df_missing = df.copy()

# Creación del indicador de faltantes para atributos con datos perdidos
for col in df.columns:
    missing = df[col].isnull()
    num_missing = np.sum(missing)

    if num_missing > 0:
```

```
print('...creado indicador de faltantes para: {}'.format(col))
df_missing['{}_ismissing'.format(col)] = missing
```

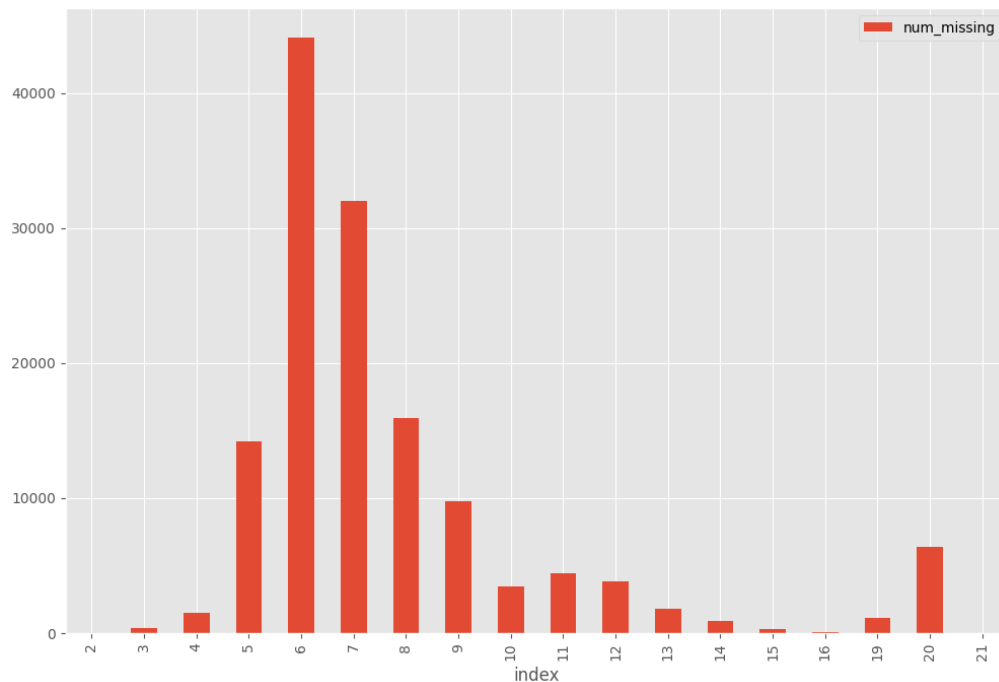
# Basado en ese indicador se muestra el histograma de datos perdidos

```
ismissing_cols = [col for col in df_missing.columns if 'ismissing' in col]
```

```
df_missing['num_missing'] = df_missing[ismissing_cols].sum(axis=1)
```

```
df_missing['num_missing'].value_counts().reset_index().sort_values(by='index').plot.bar(x='index', y='num_missir
```

```
...creado indicador de faltantes para: NOMBRE
...creado indicador de faltantes para: DESCRIPCION
...creado indicador de faltantes para: FEC_INICIO_ACTIVIDAD
...creado indicador de faltantes para: GRUPO
...creado indicador de faltantes para: CATEGORIA
...creado indicador de faltantes para: MODALIDAD
...creado indicador de faltantes para: CODIGO_POSTAL
...creado indicador de faltantes para: LOCALIDAD
...creado indicador de faltantes para: ID_MUNICIPIO
...creado indicador de faltantes para: MUNICIPIO
...creado indicador de faltantes para: ID_PROVINCIA
...creado indicador de faltantes para: PROVINCIA
...creado indicador de faltantes para: TELEFONO
...creado indicador de faltantes para: MOVIL
...creado indicador de faltantes para: FAX
...creado indicador de faltantes para: CORREO_ELECTRONICO
...creado indicador de faltantes para: TOT_GEN_UA
...creado indicador de faltantes para: TOT_GEN_PLAZAS
...creado indicador de faltantes para: IND_TIPO_ALQUILER
...creado indicador de faltantes para: COORD_X
...creado indicador de faltantes para: COORD_Y
...creado indicador de faltantes para: SRID
<Axes: xlabel='index'>
```



Se ve que no hay ningún registro con menos de dos valores perdidos y que hay cerca de 50.000 registros con seis valores perdidos y más de 5.000 con veinte valores perdidos (les falta más de la mitad de los valores).

## ▼ 4.2 Soluciones ante los datos perdidos

### ▼ 4.2.a Dejar o no dejar el registro

En el mapa de calor se observa que alrededor de la fila 15.000 hay gran número de registros con mucha información nula (en el histograma de datos faltantes también se observan esos registros: hay más de 5.000 registros con veinte valores perdidos).

Se podría crear un nuevo conjunto de datos eliminando esos registros a los que les faltan datos en veinte registros, pero viendo la información de algunos de ellos, se observa que todos son de TIPO\_OBJETO "Guía de turismo", tipo que no aporta información a la resolución del problema planteado.

Se analizan los valores posibles de TIPO\_OBJETO para ver cuáles sí aportan información:

```
print("Valores de TIPO_OBJETO: ", df["TIPO_OBJETO"].unique())

Valores de TIPO_OBJETO: ['Apartamento turístico' 'AIAT. Actividades deportivas'
'AIAT. Actividades dirigidas a prestar servicios de recepción a las personas usuarias turísticas'
'AIAT. Actividades relacionadas con el conocimiento de la lengua castellana por personas extranjeras'
'AIAT. Transporte turístico, autobuses, coches de caballo, alquiler de bicicletas u otros'
'AIAT. Ocio, entretenimiento y esparcimiento'
'AIAT. Actividades de intermediación de servicios turísticos no incluidas en la letra b) ap 1 art 28'
'AIAT. Puertos deportivos' 'AIAT. Campos de golf' 'AIAT. Campos de Polo'
'AIAT. Actividades relacionadas con el bienestar personal'
'AIAT. Estaciones de esquí' 'Empresa de turismo activo'
'Agencias de viajes. Establecimiento y punto de venta'
'Agencia de viajes' 'Establecimiento/pto.venta AAVV fuera de Andalucía'
'Campamento de turismo' 'Casa rural' 'Complejo turístico rural'
'EC. Turismo ecológico' 'Guías Libre Prestación Temporal'
'Guía de turismo' 'Establecimiento Hotelero'
'Organización de congresos, convenciones u otro tipo de eventos empresariales'
'REST. Restauración y catering turístico' 'Oficina de turismo'
'Punto de información turística' 'Clasificación de Proyectos'
'Vivienda con fines turísticos' 'Vivienda turística de alojamiento rural'
'Establecimiento/servicio turístico Clandestino']
```

Ya que al problema planteado sólo afectan las entidades relacionadas con el alojamiento turístico, se mantienen en el dataset sólo los registros con TIPO\_OBJETO igual a uno de los siguientes valores: "Apartamento turístico", "Campamento de turismo", "Casa rural", "Establecimiento Hotelero", "Vivienda con fines turísticos", "Vivienda turística de alojamiento rural".

Y se reinicia el índice.

```
print('Antes de eliminar los registros:', df.shape)
condicion = (df['TIPO_OBJETO'] == "Apartamento turístico") \
| (df['TIPO_OBJETO'] == "Campamento de turismo") \
| (df['TIPO_OBJETO'] == "Casa rural") \
| (df['TIPO_OBJETO'] == "Establecimiento Hotelero") \
| (df['TIPO_OBJETO'] == "Vivienda con fines turísticos") \
| (df['TIPO_OBJETO'] == "Vivienda turística de alojamiento rural")
df = df.loc[condicion,]
print('Después de eliminar los registros:', df.shape)

# Se reinicia el índice
df = df.reset_index(drop=True)

Antes de eliminar los registros: (140048, 30)
Después de eliminar los registros: (126869, 30)
```

### ▼ 4.2.b Dejar o no el atributo

En cuanto a las columnas, se observa tanto en el mapa de calor como en la lista de porcentajes que prácticamente todos los datos de los atributos DESCRIPCION y FAX son nulos.

También faltan muchos de los datos en TELEFONO, MOVIL y CORREO\_ELECTRONICO. Los tres son datos de contacto de cada registro concreto.

Como no aportan información a la resolución del problema, se eliminan todas esas columnas.

```
df.drop(columns=['DESCRIPCION', 'TELEFONO', 'MOVIL', 'FAX', 'CORREO_ELECTRONICO'], inplace=True)
print('Después de eliminar los atributos:', df.shape)
```

```
Después de eliminar los atributos: (126869, 25)
```

Los datos de GRUPO, CATEGORIA y MODALIDAD también tienen muchos nulos, pero los datos no nulos sí aportan información (se encuentran en las filas con variables TIPO\_OBJETO que se clasifican por categorías y modalidades, como los hoteles, apartamentos o pensiones). Es por ello por lo que se mantienen estas columnas.

#### ▼ 4.2.c Imputar a los desaparecidos

Cuando el rasgo es una variable numérica se puede llevar a cabo algún computo con los datos que faltan, como asignarles el valor medio o mediano de los datos que sí existen en ese mismo atributo. Cuando la característica es una variable categórica se les puede asignar la moda (el valor más frecuente).

Se analiza en primer lugar qué cantidad de datos faltan en cada atributo numérico:

```
print('Número de datos faltantes en los atributos numéricos:')
for col in numeric_cols:
    print('...en', col + ':', len(df.loc[df[col].isnull()]))

print()
print('Registros con ID_MUNICIPIO nulo:')
print(df[['NOMBRE', 'ID_MUNICIPIO', 'MUNICIPIO', 'ID_PROVINCIA', 'PROVINCIA']].loc[df['ID_MUNICIPIO'].isnull()])

print()
print('Valores de SRID:', df["SRID"].unique())
```

```
Número de datos faltantes en los atributos numéricos:
```

```
...en RN: 0
...en ID: 0
...en FEC_INSCRIPCION: 0
...en FEC_INICIO_ACTIVIDAD: 1
...en TIPO_OBJETO_ID: 0
...en ID_MUNICIPIO: 2
...en ID_PROVINCIA: 2
...en TOT_GEN_UA: 446
...en TOT_GEN_PLAZAS: 112
...en SRID: 11167
```

```
Registros con ID_MUNICIPIO nulo:
```

	NOMBRE	ID_MUNICIPIO	MUNICIPIO	ID_PROVINCIA	PROVINCIA
6006	prueba clandestina	NaN	NaN	NaN	NaN
6007	PRUEBA CLANDESTINO	NaN	NaN	NaN	NaN

```
Valores de SRID: [25830. nan]
```

Se realizan las siguientes acciones en los que faltan:

- En FEC\_INICIO\_ACTIVIDAD se asigna a los faltantes el valor de FEC\_INSCRIPCION de ese registro
- Se eliminan los registros con ID\_MUNICIPIO e ID\_PROVINCIA nulos (se verifica que son registros de prueba, sin información)
- En TOT\_GEN\_UA y en TOT\_GEN\_PLAZAS se asigna la media de los datos existentes para esos atributos



- En SRID se asigna el valor "25830." (es el identificador del sistema de coordenadas en que están expresadas las coordenadas, que para toda Andalucía es el mismo)

```
df['FEC_INICIO_ACTIVIDAD'] = df['FEC_INICIO_ACTIVIDAD'].fillna(df['FEC_INSCRIPCION'])

ind_missing = df[df['ID_MUNICIPIO'].isnull()].index
df = df.drop(ind_missing, axis=0)

mediaUA = round(df['TOT_GEN_UA'].median())
df['TOT_GEN_UA'] = df['TOT_GEN_UA'].fillna(mediaUA)

mediaPLAZAS = round(df['TOT_GEN_PLAZAS'].median())
df['TOT_GEN_PLAZAS'] = df['TOT_GEN_PLAZAS'].fillna(mediaPLAZAS)

df['SRID'] = df['SRID'].fillna(25830.)

print('Número de datos faltantes en los atributos numéricos:')
for col in numeric_cols:
    print('...en', col + ':', len(df.loc[df[col].isnull()]))

Número de datos faltantes en los atributos numéricos:
...en RN: 0
...en ID: 0
...en FEC_INSCRIPCION: 0
...en FEC_INICIO_ACTIVIDAD: 0
...en TIPO_OBJETO_ID: 0
...en ID_MUNICIPIO: 0
...en ID_PROVINCIA: 0
...en TOT_GEN_UA: 0
...en TOT_GEN_PLAZAS: 0
...en SRID: 0
```

## ▼ 4.2.d Reemplazar valores

En este punto ya no hay atributos numéricos con datos nulos, pero sí hay dos campos con algunos datos iguales a cero, cuando deben tener datos mayores a cero ya que contienen el número total de habitaciones (o unidades de alojamiento en general) y el número total de plazas: TOT\_GEN\_UA y TOT\_GEN\_PLAZAS.

```
indice1 = df.loc[ (df.TOT_GEN_UA == 0) & (df.TOT_GEN_PLAZAS != 0) ].index
indice2 = df.loc[ (df.TOT_GEN_PLAZAS == 0) & (df.TOT_GEN_UA != 0) ].index
indice3 = df.loc[ (df.TOT_GEN_PLAZAS == 0) & (df.TOT_GEN_UA == 0) ].index
print('Número de registros con habitaciones == 0 y plazas != 0:', len(indice1))
print('Número de registros con habitaciones != 0 y plazas == 0:', len(indice2))
print('Número de registros con habitaciones == 0 y plazas == 0:', len(indice3))

Número de registros con habitaciones == 0 y plazas != 0: 30
Número de registros con habitaciones != 0 y plazas == 0: 2
Número de registros con habitaciones == 0 y plazas == 0: 15
```

Se realiza un reemplazo de los valores iguales a cero con el criterio siguiente:

CASO 1: Si falta el dato en el atributo TOT\_GEN\_UA pero sí existe el dato en TOT\_GEN\_PLAZAS para ese mismo registro, se insertará como valor “número de habitaciones” un valor igual al redondeo del “número de plazas” dividido entre la media de plazas por habitación CASO 2: De forma análoga, si falta el dato en el atributo TOT\_GEN\_PLAZAS pero sí existe el dato en TOT\_GEN\_UA para ese mismo registro, se insertará como valor “número de plazas” un valor igual al redondeo del “número de habitaciones” multiplicado por la media de plazas por habitación CASO 3: En el resto (en los que tanto el número de plazas como el número de habitaciones son cero), mayoritariamente de TIPO\_OBJETO "Vivienda turística de alojamiento rural", se inserta la media de esos valores para ese TIPO\_OBJETO

```
mediaPlazasHab = df.TOT_GEN_PLAZAS.sum() / df.TOT_GEN_UA.sum()
```

```
# CASO 1
```

```

df.loc[indice1, 'TOT_GEN_UA'] = round(df['TOT_GEN_PLAZAS']/mediaPlazasHab,0)

# CASO 2
df.loc[indice2, 'TOT_GEN_PLAZAS'] = round(df['TOT_GEN_UA']*mediaPlazasHab,0)

# CASO 3
df_caso3 = df.loc[df.TIPO_OBJETO == 'Vivienda turística de alojamiento rural']
df.loc[indice3, 'TOT_GEN_UA'] = round(df_caso3['TOT_GEN_UA'].mean(),0)
df.loc[indice3, 'TOT_GEN_PLAZAS'] = round(df_caso3['TOT_GEN_PLAZAS'].mean(),0)

indice1 = df.loc[ (df.TOT_GEN_UA == 0) & (df.TOT_GEN_PLAZAS != 0) ].index
indice2 = df.loc[ (df.TOT_GEN_PLAZAS == 0) & (df.TOT_GEN_UA != 0) ].index
indice3 = df.loc[ (df.TOT_GEN_PLAZAS == 0) & (df.TOT_GEN_UA == 0) ].index
print('Número de registros con habitaciones == 0 y plazas != 0:', len(indice1))
print('Número de registros con habitaciones != 0 y plazas == 0:', len(indice2))
print('Número de registros con habitaciones == 0 y plazas == 0:', len(indice3))

    Número de registros con habitaciones == 0 y plazas != 0: 0
    Número de registros con habitaciones != 0 y plazas == 0: 0
    Número de registros con habitaciones == 0 y plazas == 0: 0

```

Para los atributos categóricos se sustituyen los valores nulos por el valor "--SIN DATOS--".

```

df_non_numeric = df.select_dtypes(exclude=[np.number])
non_numeric_cols = df_non_numeric.columns.values
for col in non_numeric_cols:
    df[col] = df[col].fillna('--SIN DATOS--')

print('Número de datos faltantes en los atributos categóricos:')
for col in non_numeric_cols:
    print('...en', col + ':', len(df.loc[df[col].isnull())))

    Número de datos faltantes en los atributos categóricos:
    ...en COD_REGISTRO: 0
    ...en NOMBRE: 0
    ...en TIPO_OBJETO: 0
    ...en GRUPO: 0
    ...en CATEGORIA: 0
    ...en MODALIDAD: 0
    ...en ESPECIALIDADES: 0
    ...en DOMICILIO_ESTAB: 0
    ...en CODIGO_POSTAL: 0
    ...en LOCALIDAD: 0
    ...en MUNICIPIO: 0
    ...en PROVINCIA: 0
    ...en IND_TIPO_ALQUILER: 0
    ...en COORD_X: 0
    ...en COORD_Y: 0

```

## ▼ 5 Outliers (datos atípicos o fuera de rango)

### ▼ 5.1 Histograma / Box Plot

Cuando el atributo es numérico, para detectar los valores atípicos se puede usar un histograma o un diagrama de caja.

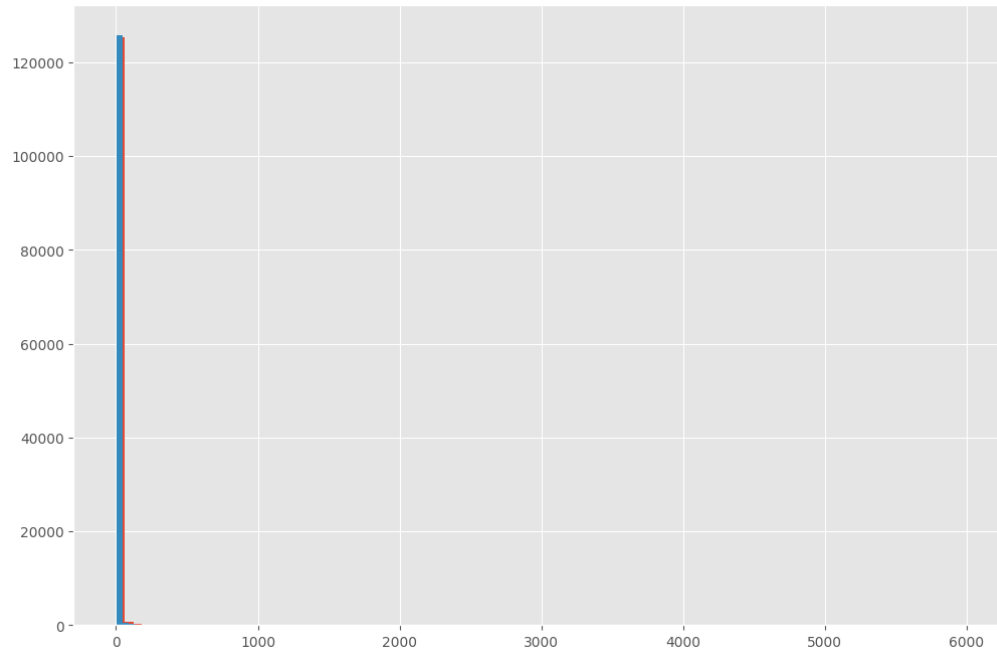
Se muestran superpuestos los histogramas de los atributos TOT\_GEN\_PLAZAS y TOT\_GEN\_UA:

```

# histogram of life_sq.
df['TOT_GEN_PLAZAS'].hist(bins=100)
df['TOT_GEN_UA'].hist(bins=100)

```

<Axes: >



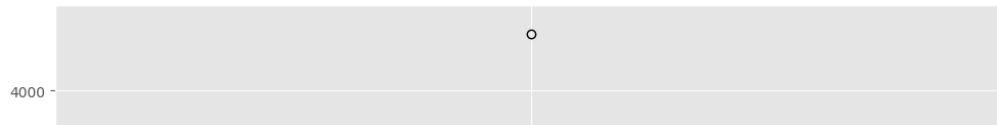
En ambos casos los datos parecen muy sesgados con la posible existencia de valores atípicos. Están centrados en un rango muy pequeño de valores.

A continuación se muestra el diagrama de caja para ambos atributos.

Para TOT\_GEN\_UA:

```
df.boxplot(column=[ 'TOT_GEN_UA' ])
```

<Axes: >




Se muestra un dato atípico, un registro con más de 4000 habitaciones.

Se analiza el resto de información sobre ese registro para decidir si se trata de un verdadero valor atípico o se trata de un error:

```
df.loc[df['TOT_GEN_UA'] == df['TOT_GEN_UA'].describe()['max']]
```

	RN	ID	COD_REGISTRO	NOMBRE	FEC_INSCRIPCION	FEC_INICIO_AC
42815	55979	180512	VFT/GR/08448	PISO EN LQA HEERRADUIRA	20230509	202

1 rows × 25 columns



¿Un piso en La Herradura con 4436 habitaciones y sólo cuatro plazas?

Se ve que se trata de un error, no un valor atípico.

Se corrige de acuerdo a la media de plazas por habitación calculada anteriormente:

```
df.loc[[42815], 'TOT_GEN_UA'] = round(df['TOT_GEN_PLAZAS']/mediaPlazasHab,0)
```

Para TOT\_GEN\_PLAZAS:

```
df.boxplot(column=['TOT_GEN_PLAZAS'])
```

<Axes: >

6000

o

Se observan varios valores con datos atípicos.

Se analizan los registros con TOT\_GEN\_PLAZAS mayor a 2500:

4000

```
df.loc[df['TOT_GEN_PLAZAS']>2500]
```

	RN	ID	COD_REGISTRO	NOMBRE	FEC_INSCRIPCION	FEC_INICIO_ACTIVIDAD
<b>6</b>	7	3138	A/AL/00058	PUEBLO INDALO	19891204	19891204.0
<b>2181</b>	7799	5530	CM/HU/00005	PLAYA DE MAZAGON	19781115	19781115.0
<b>2182</b>	7800	11534	CM/HU/00006	DOÑANA PLAYA	19830707	19830707.0

3 rows × 25 columns



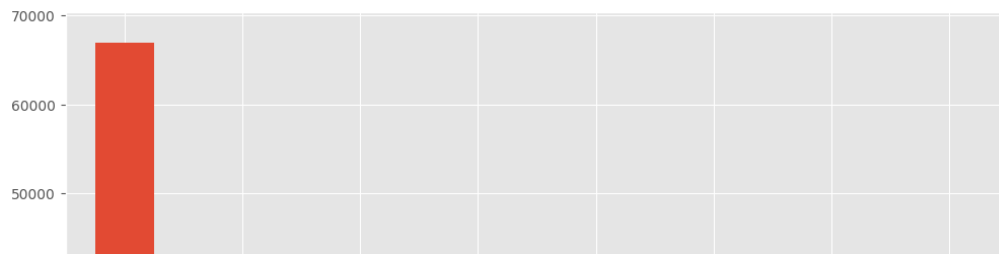
Todo correcto: el primer registro se corresponde con un complejo de apartamentos turísticos en Mojácar (Almería), con esa capacidad (<https://www.andalucia.org/es/mojacar-alojamientos-apartamentos-best-pueblo-indalo>) y los otros dos son grandes campings en Huelva.

## ▼ 5.2 Bar Chart

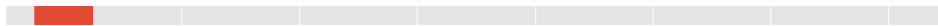
Para conocer la distribución del atributo PROVINCIA, al ser un atributo categórico, se usa un gráfico de barras para analizar sus valores posibles (deberían ser sólo las ocho provincias andaluzas) y distribución.

```
df['PROVINCIA'].value_counts().plot.bar()
```

<Axes: >



Todo correcto.



## ▼ 6 Datos innecesarios

Después del trabajo realizado para los datos faltantes y los valores atípicos se pasa a analizar los datos innecesarios, que son aquellos datos que no añaden valor por tres razones:



### ▼ 6.1 Desinformativos o repetitivos

Por tener demasiadas filas con el mismo valor.

Para identificar estos datos se crea una lista de atributos con un alto porcentaje del mismo valor.

A continuación se muestran los atributos con más del 90% de filas con el mismo valor:

```
num_rows = len(df.index)
low_information_cols = []

for col in df.columns:
    cnts = df[col].value_counts(dropna=False)
    top_pct = (cnts/num_rows).iloc[0]

    if top_pct > 0.9:
        low_information_cols.append(col)
        print('{0}: {1:.5f}%'.format(col, top_pct*100))
        print(cnts)
        print()

GRUPO: 95.35971%
--SIN DATOS--          120980
Hotel                  1810
Edificio/Complejo     1209
Conjunto               852
Pensión               799
Hostal                 798
Camping               177
Hotel-Apartamento    120
Albergue              96
Área pernocta autocaravanas 26
Name: GRUPO, dtype: int64

CATEGORIA: 92.43066%
--SIN DATOS--          117264
Básica                 3304
2 Llaves               1012
Única                  895
1 Llave                881
1 Estrella             832
2 Estrellas            811
4 Estrellas            616
3 Estrellas            579
Superior               438
3 Llaves               143
5 Estrellas            47
5 Estrellas con calificativo Gran Lujo 20
```

```

3 Llaves (D.A. Tercera D.194/2010)      1/
4 Llaves                                  7
4 Llaves (D.A. Tercera D.194/2010)      1
Name: CATEGORIA, dtype: int64

MODALIDAD: 95.12718%
--SIN DATOS--      120685
Ciudad              2915
Playa               1662
Rural               1427
Carretera           178
Name: MODALIDAD, dtype: int64

ESPECIALIDADES: 99.54283%
No disponible                               126287
Cortijo                                     142
Familiars                                   94
Monumentos e Inmuebles Protegidos         31
Casa Cueva                                29
...
Agroturismo, Cortijo, Deportivos, Granja-Escuela  1
Albergue, De Naturaleza, Familiares, Refugio      1
Albergue, Cortijo                                1
Agroturismo, Cortijo                              1
De Congresos y Negocios, Familiares, Monumentos e Inmuebles Protegidos  1
Name: ESPECIALIDADES, Length: 84, dtype: int64

SRID: 100.000000%
25830.0      126867
Name: SRID, dtype: int64

```

Ya se explicitó que los tres primeros atributos (GRUPO, CATEGORIA y MODALIDAD) se decidían mantener porque a pesar de tener muchos nulos sí aportan información útil en aquellos registros en los que sí están presentes (se encuentran en las filas con variables TIPO\_OBJETO que se clasifican por categorías y modalidades, como los hoteles, apartamentos o pensiones).

Si se elimina el atributo ESPECIALIDAD, porque más del 99,5% de sus datos son "No disponible" y el resto tienen valores muy diferentes entre sí.

El atributo SRID, el sistema de coordenadas en que están expresadas las coordenadas, es el mismo para todos los registros, por lo que se opta por eliminarlo. Habría que mantenerlo o volver a incluirlo si se quisiera comparar los datos de este registro con otros con diferente SRID.

```

df.drop(columns=['ESPECIALIDADES', 'SRID'], inplace=True)
print('Después de eliminar los registros:', df.shape)

```

```

Después de eliminar los registros: (126867, 23)

```

## ▼ 6.2 Irrelevantes

Ya se han eliminado algunos de los atributos irrelevantes para nuestro problema, como los referentes al titular del establecimiento o al seguro obligatorio, o los atributos DESCRIPCION, TELEFONO, MOVIL, FAX o CORREO\_ELECTRONICO.

También se han eliminado algunos registros, como aquellos con TIPO\_OBJETO no relacionados con el alojamiento turístico (guías turísticos, oficinas de turismo,...).

Se ha comprobado que todos los datos pertenecen a alguna de las provincias de Andalucía (si algún registro hubiera sido de fuera, se habría eliminado por irrelevante).

Hay otro atributo más que no aporta información a nuestro problema: las fechas de inscripción (la de inicio de actividad nos informa sobre la veteranía del negocio). Por lo tanto, se elimina el atributo FEC\_INSCRIPCION.

```
df.drop(columns=['FEC_INSCRIPCION'], inplace=True)
print('Después de eliminar el atributo:', df.shape)
```

Después de eliminar el atributo: (126867, 22)

## ▼ 7 Datos inconsistentes

### ▼ 7.1 Conversión de tipos

El atributo CODIGO\_POSTAL, de tipo objeto, se limpia reemplazando los valores vacíos por "--SIN DATOS--", eliminando los puntos y reemplazando las oes mayúsculas por ceros.

```
df.loc[df['CODIGO_POSTAL']=='']

df.loc[df['CODIGO_POSTAL']==''] = '--SIN DATOS--'
df['CODIGO_POSTAL'] = df['CODIGO_POSTAL'].str.replace('\.', '', regex=True) # eliminar los puntos
df['CODIGO_POSTAL'] = df['CODIGO_POSTAL'].str.replace('O', '0') # cambiar las oes mayúsculas por ceros
```

ID\_MUNICIPIO, ID\_PROVINCIA, TOT\_GEN\_UA y TOT\_GEN\_PLAZAS se convierten a enteros (int32)

```
lista = ['RN', 'ID', 'TIPO_OBJETO_ID', 'ID_MUNICIPIO', 'ID_PROVINCIA', 'TOT_GEN_UA', 'TOT_GEN_PLAZAS']
for col in lista:
    df[col] = df[col].astype('int32')
```

### ▼ 7.2 Capitalización

Se convierten a mayúsculas los atributos categóricos salvo TIPO\_OBJETO, GRUPO, CATEGORIA y MODALIDAD (se comprueba que estos cuatro atributos tienen un conjunto de valores posibles muy limitado).

```
print('Valores de TIPO_OBJETO:', df.TIPO_OBJETO.unique())
print('Valores de GRUPO:', df.GRUPO.unique())
print('Valores de CATEGORIA:', df.CATEGORIA.unique())
print('Valores de MODALIDAD:', df.MODALIDAD.unique())

to_upper = ['COD_REGISTRO', 'NOMBRE', 'DOMICILIO_ESTAB', 'LOCALIDAD', 'MUNICIPIO', 'PROVINCIA', 'IND_TIPO_ALQUIL']
for col in to_upper:
    df[col] = df[col].str.upper()

Valores de TIPO_OBJETO: ['Apartamento turístico' 'Campamento de turismo' 'Casa rural'
 'Establecimiento Hotelero' 'Vivienda con fines turísticos'
 'Vivienda turística de alojamiento rural']
Valores de GRUPO: ['Edificio/Complejo' 'Conjunto' 'Camping' 'Área pernocta autocaravanas'
 '--SIN DATOS--' 'Pensión' 'Hotel' 'Hostal' 'Hotel-Apartamento' 'Albergue']
Valores de CATEGORIA: ['2 Llaves' '1 Llave' '3 Llaves' '4 Llaves'
 '3 Llaves (D.A. Tercera D.194/2010)'
 '4 Llaves (D.A. Tercera D.194/2010)' '3 Estrellas' '2 Estrellas'
 '1 Estrella' '4 Estrellas' '--SIN DATOS--' 'Básica' 'Superior' 'Única'
 '5 Estrellas' '5 Estrellas con calificativo Gran Lujo']
Valores de MODALIDAD: ['Playa' 'Ciudad' 'Rural' 'Carretera' '--SIN DATOS--']
```

### ▼ 7.3 Formato

Las fechas en FEC\_INICIO\_ACTIVIDAD están almacenadas como número con formato AAAAMMDD, donde AAAA son los cuatro dígitos del año, MM los dos del mes y DD los dos del día. Por ejemplo, el 30/06/2023 estaría almacenado como 20230630.



Se convierte a cadena de texto y se extraen los valores del día, mes y año y se almacenan en tres nuevos atributos llamados respectivamente DIA\_INICIO\_ACT, MES\_INICIO\_ACT y ANYO\_INICIO\_ACT.

Se observa que hay algunos registros con años erróneos (con años menores que 1900 o mayores que 2023). Se eliminan esos registros y se reinicia el índice.

### ¿Qué hacer?

```
df['FEC_INICIO_ACTIVIDAD'] = df['FEC_INICIO_ACTIVIDAD'].astype('str')

df['DIA_INICIO_ACT'] = df['FEC_INICIO_ACTIVIDAD'].str.slice(6, 8)
df['MES_INICIO_ACT'] = df['FEC_INICIO_ACTIVIDAD'].str.slice(4, 6)
df['ANYO_INICIO_ACT'] = df['FEC_INICIO_ACTIVIDAD'].str.slice(0, 4)

ind_bad_data = df[(df['ANYO_INICIO_ACT'].astype('int32') < 1900) | (df['ANYO_INICIO_ACT'].astype('int32') > 2023)]
print('Registros con días, meses o años erróneos:', len(ind_bad_data))
df = df.drop(ind_bad_data, axis=0)

# Se reinicia el índice
df = df.reset_index(drop=True)
print()

df['DIA_INICIO_ACT'] = df['DIA_INICIO_ACT'].astype('int32')
df['MES_INICIO_ACT'] = df['MES_INICIO_ACT'].astype('int32')
df['ANYO_INICIO_ACT'] = df['ANYO_INICIO_ACT'].astype('int32')

# Se comprueba que los datos obtenidos son válidos
print('Datos obtenidos:')
print('...días:', np.sort(df['DIA_INICIO_ACT'].unique()))
print('...meses:', np.sort(df['MES_INICIO_ACT'].unique()))
print('...años:', np.sort(df['ANYO_INICIO_ACT'].unique()))

Registros con días, meses o años erróneos: 12

Datos obtenidos:
...días: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26 27 28 29 30 31]
...meses: [ 1  2  3  4  5  6  7  8  9 10 11 12]
...años: [1910 1915 1926 1930 1932 1941 1945 1946 1947 1948 1950 1951 1952 1953
 1954 1955 1957 1958 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968
 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982
 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996
 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023]
```

## ▼ 7.4 Direcciones

En el atributo DOMICILIO\_ESTAB se ejecuta el siguiente código para eliminar los espacios en blanco al principio y al final, eliminar los puntos y estandarizar algunas palabras. En un paso anterior ya se pusieron todas sus letras en mayúsculas.

```
print('...antes:')
print(df['DOMICILIO_ESTAB'].head(10))

df['DOMICILIO_ESTAB'] = df['DOMICILIO_ESTAB'].str.strip() # eliminar espacios al principio y al final
df['DOMICILIO_ESTAB'] = df['DOMICILIO_ESTAB'].str.replace('\.', '', regex=True) # eliminar puntos
df['DOMICILIO_ESTAB'] = df['DOMICILIO_ESTAB'].str.replace('\bCALLE\b', 'CL ', regex=True) # reemplazar "CALLE"
df['DOMICILIO_ESTAB'] = df['DOMICILIO_ESTAB'].str.replace('C/', 'CL', regex=True) # reemplazar C/ por CL
df['DOMICILIO_ESTAB'] = df['DOMICILIO_ESTAB'].str.replace('\bAVDA\b', 'AV', regex=True) # reemplazar AVDA por AV
df['DOMICILIO_ESTAB'] = df['DOMICILIO_ESTAB'].str.replace('\bAVENIDA\b', 'AV', regex=True) # reemplazar AVENIDA por AV
df['DOMICILIO_ESTAB'] = df['DOMICILIO_ESTAB'].str.replace('\bPASEO\b', 'PO ', regex=True) # reemplazar "PASEO"
df['DOMICILIO_ESTAB'] = df['DOMICILIO_ESTAB'].str.replace('\bCAMINO\b', 'CNO ', regex=True) # reemplazar "CAMINO"
df['DOMICILIO_ESTAB'] = df['DOMICILIO_ESTAB'].str.replace('\bURBANIZACION\b', 'URB', regex=True) # reemplazar "URBANIZACION"
df['DOMICILIO_ESTAB'] = df['DOMICILIO_ESTAB'].str.replace('\bURBANIZACIÓN\b', 'URB', regex=True) # reemplazar "URBANIZACIÓN"
```

```
df['DOMICILIO_ESTAB'] = df['DOMICILIO_ESTAB'].str.replace('\\bCARRETERA\\b', 'CTRA', regex=True) # reemplazar CA

print()
print('...después:')
print(df['DOMICILIO_ESTAB'].head(10))

...antes:
0      CNO. VIEJO DE GARRUCHA, 10
1      PL. FLORES, 6
2      CL. TRAIÑA, S/N
3      AVDA. GAVIOTAS, S/N
4      PASEO MARITIMO, 99
5      CL. VICENTE ALEXANDRE, S/N
6      PASEO DEL MEDITERRANEO, 233
7      PLAYA SERENA
8      URB. PLAYASERENA
9      CL. CORREOS, S/N (SAN JOSE)
Name: DOMICILIO_ESTAB, dtype: object

...después:
0      CNO VIEJO DE GARRUCHA, 10
1      PL FLORES, 6
2      CL TRAIÑA, S/N
3      AV GAVIOTAS, S/N
4      PO MARITIMO, 99
5      CL VICENTE ALEXANDRE, S/N
6      PO DEL MEDITERRANEO, 233
7      PLAYA SERENA
8      URB PLAYASERENA
9      CL CORREOS, S/N (SAN JOSE)
Name: DOMICILIO_ESTAB, dtype: object
```

## ▼ 8 Comparativa entre los datos iniciales y los datos finales

```
print('Tamaño del conjunto de datos original:', df_original.shape)
print('Número total de nulos:', df_original.isnull().sum().sum())
```

```
Tamaño del conjunto de datos original: (140048, 76)
Número total de nulos: 5775080
```

Se partió de un conjunto de datos objetivo con **140.048** registros y **76** atributos, con datos nulos, datos perdidos, datos mal formateados y datos inconsistentes.

```
print('Tamaño del conjunto de datos transformados:', df.shape)
print('Número total de nulos:', df.isnull().sum().sum())
```

```
Tamaño del conjunto de datos transformados: (126855, 25)
Número total de nulos: 0
```

Mediante el trabajo realizado para la preparación, limpieza y transformación de los datos se ha obtenido un conjunto de datos transformados, una vista minable, con **126.855** registros y **25** atributos.

## 9 Data mining y utilización del conocimiento

El modelo que se plantea utilizar para realizar este proceso KDD descriptivo será un agrupamiento o clustering, pretendiéndose obtener grupos a partir de los datos para encontrar patrones en la distribución de las viviendas de alquiler turístico en los diferentes municipios en función de diferentes atributos.

En un primer momento se utilizaría un algoritmo de agrupamiento K-medias, al ser el algoritmo de aprendizaje no supervisado más simple.

En caso de que los datos no siguieran un formato circular y por lo tanto no se agruparan correctamente con un K-medias, se utilizaría un algoritmo de mezcla gaussiana (ya que es un algoritmo que no necesita datos con forma circular para que funcione bien).

El algoritmo de mezcla gaussiana calcula la probabilidad de que un punto de datos pertenezca a una distribución gaussiana específica y éste es el grupo en el que se ubicará.

### **Resultado final**

Como resultado final, para utilizar el conocimiento se pretende obtener cuadros y gráficos que describan los escenarios estudiados según las tipologías y condicionantes planteados.