ETAPAS DEL CICLO DE VIDA DE CIENCIA DE DATOS.

1. Formular una pregunta:

A pesar del creciente uso de la bicicleta compartida en Madrid, el sistema BiciMAD presenta concentración de viajes en zonas específicas del centro urbano, mientras que otras áreas —particularmente residenciales o periféricas— muestran una baja frecuencia de uso o ausencia de estaciones. Esta desigualdad no se analiza habitualmente desde una perspectiva espacio-temporal integrada.

Además, se desconoce si la variación horaria (p. ej., viajes en hora punta vs. horas valle, entre semana vs. fines de semana) está alineada con el tipo de uso del suelo de cada zona. Sin esta información, es difícil adaptar la distribución de bicicletas, mejorar la equidad del servicio y reducir la congestión o infrautilización.

¿Los viajes tienen registro de entrada y salida? ¿Cuanto duran los que no se registran en la ida y/o en la vuelta? ¿Que relacion tiene el suelo con el uso de las bicicletas?

```
# linear algebra, probability
import numpy as np
# data manipulation
import pandas as pd
# visualization
import matplotlib.pyplot as plt
import seaborn as sns
## interactive visualization library
import plotly.offline as py
py.init_notebook_mode()
import plotly.graph_objs as go
import plotly.figure_factory as ff
import plotly.express as px
from datetime import datetime
import geopandas as gpd
```

2. Adquisición de datos:

```
# Para leer un archivo del drive:
# 1. Importamos la biblioteca drive para montar la unidad.
# 2. Montamos la carpeta que queremos acceder
from google.colab import drive
drive.mount('/content/drive/')
trips = pd.read_csv("/content/drive/MyDrive/MOVILIDAD/trips_febrero_2023.csv",sep=";",
    skip_blank_lines=True,
    engine="python")
suelo = gpd.read_file("/content/drive/MyDrive/MOVILIDAD/SIOSE_Madrid_2014.gpkg", layer="T_POLIGONOS")
valor = gpd.read_file("/content/drive/MyDrive/MOVILIDAD/SIOSE_Madrid_2014.gpkg", layer="T_VALORES")
```

```
Mounted at /content/drive/
```

3. Análisis Exploratorio de Datos Se revisan los datos e inferimos su estructura. Entonces, podremos empezar a responder las preguntas que se plantearon en la etapa 1.

```
trips.head(10)
```

	fecha	idBike	fleet	trip_minutes	geolocation_unlock	address_unlock	unlock_date	locktype	unlocktype	geolocation_lock	addr
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	2023-02-01	7337.0	1.0	5.52	{'type': 'Point', 'coordinates': [-3.6956178, ...	'Calle Jesús nº 1'	2023-02-01T00:00:10	STATION	STATION	{'type': 'Point', 'coordinates': [-3.7088337, ...	'P Ceba
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	2023-02-01	5098.0	1.0	0.32	{'type': 'Point', 'coordinates': [-3.7022591, ...	'Glorieta de Embajadores nº 2'	2023-02-01T00:00:25	STATION	STATION	{'type': 'Point', 'coordinates': [-3.7022591, ...	'G Eml
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
5	2023-02-01	6519.0	1.0	0.27	{'type': 'Point', 'coordinates': [-3.6894193, ...	'Calle Antonio Maura nº 15'	2023-02-01T00:00:36	STATION	STATION	{'type': 'Point', 'coordinates': [-3.6894193, ...	'Call Ma
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
7	2023-02-01	2551.0	1.0	8.58	{'type': 'Point', 'coordinates': [-3.7022591, ...	'Glorieta de Embajadores nº 2'	2023-02-01T00:00:53	STATION	STATION	{'type': 'Point', 'coordinates': [-3.6991147, ...	'Cal
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
9	2023-02-01	6519.0	1.0	0.20	{'type': 'Point', 'coordinates': [-3.6894193, ...	'Calle Antonio Maura nº 15'	2023-02-01T00:00:57	STATION	STATION	{'type': 'Point', 'coordinates': [-3.6894193, ...	'Call Ma

T_POLIGONOS: contiene los polígonos geográficos (zonas) y campos como ID_POLYGON, SIOSE_CODE, CODIIGE (cobertura) y HILUCS (uso).

T_VALORES: contiene el detalle de todas las coberturas y atributos para cada polígono (uno a muchos).

Campo clave: SIOSE_CODE, que incluye etiquetas tipo "A(50CNF_50MTR)" que indican composiciones del uso del suelo.

```
suelo.head(10)
```

	ID_POLYGON	SIOSE_CODE	SIOSE_XML	SUPERF_HA	CODIIGE	HILUCS	SELLADO	FCC	CODBL
0	98678c0f-70e2-49f6-85d0-40b7cb9292d4	PMX(80ZEV_10SNE_05EDFnv_05VAP)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	41.602492	123	130	10.0	NaN	13
1	ae8c5627-affd-4a5d-8022-3b5e5ff31bbb	MTR	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	2.836093	330	631	NaN	NaN	13
2	cae746a8-2196-4f6e-9e4f-65d3ce3efec0	A(75MTR_25FDP)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	22.202730	340	631	NaN	25.0	13
3	febe7109-4d88-4b87-b3b3-043bb22ea29f	UDS(65ZAU_15EDFva_10VAP_05LAA_05SNE)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	1.938984	113	500	25.0	NaN	13
4	443f3abb-daa3-4b60-9ec0-a50496f88ab1	DHS(60PST_40FDP)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	41.033844	311	631	NaN	40.0	13
5	63e32573-bb96-4e49-a829-e8c265632cbc	A(35FDCfr_35MTRfr_30ACU)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	11.372221	311	631	NaN	35.0	13
	23e31c2f-6dbe-4fe1-		<?xml version="1.0"						

```
a=suelo['CODBLQ'].unique()
a
```

```
array([13], dtype=int16)
```

```
b=trips['geolocation_unlock'].unique()
b
```

```
array([nan, "{'type': 'Point', 'coordinates': [-3.6956178, 40.4132798]}",
       "{'type': 'Point', 'coordinates': [-3.7022591, 40.4056107]}",
       "{'type': 'Point', 'coordinates': [-3.6894193, 40.4166834]}",
       "{'type': 'Point', 'coordinates': [-3.6840229, 40.4211802]}",
       "{'type': 'Point', 'coordinates': [-3.6758555, 40.4323655]}",
       "{'type': 'Point', 'coordinates': [-3.6758383, 40.4239447]}",
       "{'type': 'Point', 'coordinates': [-3.6989925, 40.4607502]}",
       "{'type': 'Point', 'coordinates': [-3.6689089, 40.4114475]}",
       "{'type': 'Point', 'coordinates': [-3.6784838, 40.4082805]}",
       "{'type': 'Point', 'coordinates': [-3.708439, 40.4338516]}",
       "{'type': 'Point', 'coordinates': [-3.71183, 40.4168]}",
       "{'type': 'Point', 'coordinates': [-3.6954615, 40.4192342]}",
       "{'type': 'Point', 'coordinates': [-3.6780555, 40.4515833]}",
       "{'type': 'Point', 'coordinates': [-3.6892368, 40.4355143]}",
       "{'type': 'Point', 'coordinates': [-3.71019, 40.4364]}",
       "{'type': 'Point', 'coordinates': [-3.7036825, 40.4146755]}",
       "{'type': 'Point', 'coordinates': [-3.6996502, 40.4207773]}",
       "{'type': 'Point', 'coordinates': [-3.7095084, 40.4156057]}",
       "{'type': 'Point', 'coordinates': [-3.7020842, 40.4239757]}",
       "{'type': 'Point', 'coordinates': [-3.6772222, 40.43725]}",
       "{'type': 'Point', 'coordinates': [-3.6782777, 40.4290555]}",
       "{'type': 'Point', 'coordinates': [-3.7007111, 40.4083061]}",
       "{'type': 'Point', 'coordinates': [-3.7088337, 40.4112744]}",
       "{'type': 'Point', 'coordinates': [-3.7040344, 40.4418402]}",
       "{'type': 'Point', 'coordinates': [-3.678492, 40.4345973]}",
       "{'type': 'Point', 'coordinates': [-3.688398, 40.419752]}",
       "{'type': 'Point', 'coordinates': [-3.7036675, 40.4463667]}",
       "{'type': 'Point', 'coordinates': [-3.7064516, 40.4035988]}",
       "{'type': 'Point', 'coordinates': [-3.7043611, 40.401]}",
       "{'type': 'Point', 'coordinates': [-3.7188898, 40.4309797]}",
       "{'type': 'Point', 'coordinates': [-3.6887722, 40.4656779]}",
       "{'type': 'Point', 'coordinates': [-3.7172213778842007, 40.40765396169372]}",
       "{'type': 'Point', 'coordinates': [-3.66096, 40.43666]}",
       "{'type': 'Point', 'coordinates': [-3.6738865, 40.4159569]}",
       "{'type': 'Point', 'coordinates': [-3.6992759, 40.4060941]}",
       "{'type': 'Point', 'coordinates': [-3.7110513, 40.4070358]}",
       "{'type': 'Point', 'coordinates': [-3.7080833, 40.4411388]}",
       "{'type': 'Point', 'coordinates': [-3.6751388, 40.4441388]}",
       "{'type': 'Point', 'coordinates': [-3.7170448, 40.4253944]}",
       "{'type': 'Point', 'coordinates': [-3.6752024, 40.4272582]}",
       "{'type': 'Point', 'coordinates': [-3.6993465, 40.4309524]}",
       "{'type': 'Point', 'coordinates': [-3.697895, 40.4222862]}",
       "{'type': 'Point', 'coordinates': [-3.6991147, 40.4122047]}",
       "{'type': 'Point', 'coordinates': [-3.6915555, 40.4331111]}",
       "{'type': 'Point', 'coordinates': [-3.7038312, 40.4232649]}",
       "{'type': 'Point', 'coordinates': [-3.6935205, 40.4075606]}",
       "{'type': 'Point', 'coordinates': [-3.698447, 40.424148]}",
       "{'type': 'Point', 'coordinates': [-3.7103285, 40.4141931]}",
       "{'type': 'Point', 'coordinates': [-3.7025875, 40.4285524]}",
       "{'type': 'Point', 'coordinates': [-3.7061931, 40.4284246]}",
       "{'type': 'Point', 'coordinates': [-3.690756, 40.4232153]}",
       "{'type': 'Point', 'coordinates': [-3.6957355, 40.4162619]}",
       "{'type': 'Point', 'coordinates': [-3.713053, 40.423668]}",
       "{'type': 'Point', 'coordinates': [-3.7030833, 40.4165039]}",
       "{'type': 'Point', 'coordinates': [-3.6977715, 40.4251906]}",
       "{'type': 'Point', 'coordinates': [-3.6803874, 40.4051451]}",
       "{'type': 'Point', 'coordinates': [-3.7080429728445563, 40.39440607959559]}",
       "{'type': 'Point', 'coordinates': [-3.7040666, 40.4097617]}",
```

```
valor.head(10)
```

	ID_POLYGON	ID_COBERTURAS	ID_ANCESTROS	INTER_ID	INTER_ANCESTROS	ATRIBUTOS	SUPERF_HA	SUPERF_POR
0	00007cce-155b-40de-be0f-e0b4c841b720	858		1			1.5110	100.0
1	00007cce-155b-40de-be0f-e0b4c841b720	111	858	2	1		0.6800	45.0
2	00007cce-155b-40de-be0f-e0b4c841b720	101	858	3	1	21	0.4533	30.0
3	00007cce-155b-40de-be0f-e0b4c841b720	102	858	4	1		0.3022	20.0
4	00007cce-155b-40de-be0f-e0b4c841b720	104	858	5	1		0.0756	5.0
5	0001cccc-8b59-4f24-bce6-84582ca2f886	600		1		11	41.7011	100.0
6	0001cccc-8b59-4f24-bce6-84582ca2f886	313	600	2	1		18.7655	45.0
7	0001cccc-8b59-4f24-bce6-84582ca2f886	320	600	3	1		16.6804	40.0
8	0001cccc-8b59-4f24-bce6-84582ca2f886	300	600	4	1		6.2552	15.0
9	00026eeb-8064-4d02-a427-a8bd89282162	600		1		12	3.0053	100.0

P1. ¿Qué problema podríamos abordar en estos datos? Algunos nombres aparecen en mayúsculasy otros no. Esto puede ser un problema en un posterior análisis, así que podemos convertir todos los nombres a minúsculas.

Dar un solo formato a las columnas con contenido de texto

```
# Normalizar el formato de texto a minúsculas para columnas clave
cols_to_normalize = ['geolocation_unlock', 'address_unlock', 'locktype', 'unlocktype','address_lock']

for col in cols_to_normalize:
    trips[col] = trips[col].str.lower()
```

Posibles valores vacios en la tabla

```
len(trips)
```

```
336988
```

```
trips.isnull().sum()
```

	0
fecha	168494
idBike	168494
fleet	168494
trip_minutes	168494
geolocation_unlock	168494
address_unlock	168494
unlock_date	168494
locktype	168494
unlocktype	168494
geolocation_lock	168494
address_lock	168494
lock_date	168494
station_unlock	168874
dock_unlock	168874
unlock_station_name	168874
station_lock	168970
dock_lock	168970
lock_station_name	168970

dtype: int64

```
suelo.isnull().sum()
```

	0
ID_POLYGON	0
SIOSE_CODE	0
SIOSE_XML	0
SUPERF_HA	0
CODIIGE	0
HILUCS	0
SELLADO	27562
FCC	32656
CODBLQ	0
geometry	0

dtype: int64

```
valor.isnull().sum()
```

	0
ID_POLYGON	0
ID_COBERTURAS	0
ID_ANCESTROS	0
INTER_ID	0
INTER_ANCESTROS	0
ATRIBUTOS	0
SUPERF_HA	0
SUPERF_POR	0

dtype: int64

```
trips.dtypes
```

	0
fecha	object
idBike	float64
fleet	float64
trip_minutes	float64
geolocation_unlock	object
address_unlock	object
unlock_date	object
locktype	object
unlocktype	object
geolocation_lock	object
address_lock	object
lock_date	object
station_unlock	float64
dock_unlock	float64
unlock_station_name	object
station_lock	float64
dock_lock	float64
lock_station_name	object

dtype: object

```
suelo.dtypes
```

	0
ID_POLYGON	object
SIOSE_CODE	object
SIOSE_XML	object
SUPERF_HA	float64
CODIIGE	int16
HILUCS	int16
SELLADO	float64
FCC	float64
CODBLQ	int16
geometry	geometry

dtype: object

```
valor.dtypes
```

	0
ID_POLYGON	object
ID_COBERTURAS	int16
ID_ANCESTROS	object
INTER_ID	int16
INTER_ANCESTROS	object
ATRIBUTOS	object
SUPERF_HA	float64
SUPERF_POR	float64

dtype: object

P2. ¿Cuántos registros tenemos?

```
print(len(trips))
```

```
336988
```

```
print(len(suelo))
```

```
44858
```

```
print(len(valor))
```

```
152967
```

P3. ¿Cuál es el significado de los atributos o campos?

```
trips.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 336988 entries, 0 to 336987
Data columns (total 18 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   fecha                168494 non-null  object 
 1   idBike               168494 non-null  float64
 2   fleet                168494 non-null  float64
 3   trip_minutes         168494 non-null  float64
 4   geolocation_unlock   168494 non-null  object 
 5   address_unlock       168494 non-null  object 
 6   unlock_date          168494 non-null  object 
 7   locktype             168494 non-null  object 
 8   unlocktype           168494 non-null  object 
 9   geolocation_lock     168494 non-null  object 
 10  address_lock         168494 non-null  object 
 11  lock_date            168494 non-null  object 
 12  station_unlock       168114 non-null  float64
 13  dock_unlock          168114 non-null  float64
 14  unlock_station_name  168114 non-null  object 
 15  station_lock         168018 non-null  float64
 16  dock_lock            168018 non-null  float64
 17  lock_station_name    168018 non-null  object 
dtypes: float64(7), object(11)
memory usage: 46.3+ MB
```

```
suelo.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 44858 entries, 0 to 44857
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   ID_POLYGON  44858 non-null  object  
 1   SIOSE_CODE  44858 non-null  object  
 2   SIOSE_XML   44858 non-null  object  
 3   SUPERF_HA   44858 non-null  float64 
 4   CODIIGE     44858 non-null  int16   
 5   HILUCS      44858 non-null  int16   
 6   SELLADO     17296 non-null  float64 
 7   FCC         12202 non-null  float64 
 8   CODBLQ      44858 non-null  int16   
 9   geometry    44858 non-null  geometry
dtypes: float64(3), geometry(1), int16(3), object(3)
```

```
memory usage: 2.7+ MB
```

```
valor.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 152967 entries, 0 to 152966
Data columns (total 8 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   ID_POLYGON       152967 non-null  object 
 1   ID_COBERTURAS    152967 non-null  int16  
 2   ID_ANCESTROS     152967 non-null  object 
 3   INTER_ID         152967 non-null  int16  
 4   INTER_ANCESTROS  152967 non-null  object 
 5   ATRIBUTOS        152967 non-null  object 
 6   SUPERF_HA        152967 non-null  float64
 7   SUPERF_POR       152967 non-null  float64
dtypes: float64(2), int16(2), object(4)
memory usage: 7.6+ MB
```

Los recuentos de sus valores únicos: se puede usar

```
for col in trips.columns:
    print(f"Columna: {col}")
    print(trips[col].unique())
    print("-" * 50)
```

```
Columna: fecha
[nan '2023-02-01' '2023-02-02' '2023-02-03' '2023-02-04' '2023-02-05'
 '2023-02-06' '2023-02-07' '2023-02-08' '2023-02-09' '2023-02-10'
 '2023-02-11' '2023-02-12' '2023-02-13' '2023-02-14' '2023-02-15'
 '2023-02-16' '2023-02-17' '2023-02-18']
--------------------------------------------------
Columna: idBike
[  nan 7337. 5098. ... 7951. 7127. 5730.]
--------------------------------------------------
Columna: fleet
[nan  1.  2.]
--------------------------------------------------
Columna: trip_minutes
[  nan  5.52  0.32 ... 40.63 54.82 47.55]
--------------------------------------------------
Columna: geolocation_unlock
[nan "{'type': 'point', 'coordinates': [-3.6956178, 40.4132798]}"
 "{'type': 'point', 'coordinates': [-3.7022591, 40.4056107]}"
 "{'type': 'point', 'coordinates': [-3.6894193, 40.4166834]}"
 "{'type': 'point', 'coordinates': [-3.6840229, 40.4211802]}"
 "{'type': 'point', 'coordinates': [-3.6758555, 40.4323655]}"
 "{'type': 'point', 'coordinates': [-3.6758383, 40.4239447]}"
 "{'type': 'point', 'coordinates': [-3.6989925, 40.4607502]}"
 "{'type': 'point', 'coordinates': [-3.6689089, 40.4114475]}"
 "{'type': 'point', 'coordinates': [-3.6784838, 40.4082805]}"
 "{'type': 'point', 'coordinates': [-3.708439, 40.4338516]}"
 "{'type': 'point', 'coordinates': [-3.71183, 40.4168]}"
 "{'type': 'point', 'coordinates': [-3.6954615, 40.4192342]}"
 "{'type': 'point', 'coordinates': [-3.6780555, 40.4515833]}"
 "{'type': 'point', 'coordinates': [-3.6892368, 40.4355143]}"
 "{'type': 'point', 'coordinates': [-3.71019, 40.4364]}"
 "{'type': 'point', 'coordinates': [-3.7036825, 40.4146755]}"
 "{'type': 'point', 'coordinates': [-3.6996502, 40.4207773]}"
 "{'type': 'point', 'coordinates': [-3.7095084, 40.4156057]}"
 "{'type': 'point', 'coordinates': [-3.7020842, 40.4239757]}"
 "{'type': 'point', 'coordinates': [-3.6772222, 40.43725]}"
 "{'type': 'point', 'coordinates': [-3.6782777, 40.4290555]}"
 "{'type': 'point', 'coordinates': [-3.7007111, 40.4083061]}"
 "{'type': 'point', 'coordinates': [-3.7088337, 40.4112744]}"
 "{'type': 'point', 'coordinates': [-3.7040344, 40.4418402]}"
 "{'type': 'point', 'coordinates': [-3.678492, 40.4345973]}"
 "{'type': 'point', 'coordinates': [-3.688398, 40.419752]}"
 "{'type': 'point', 'coordinates': [-3.7036675, 40.4463667]}"
 "{'type': 'point', 'coordinates': [-3.7064516, 40.4035988]}"
 "{'type': 'point', 'coordinates': [-3.7043611, 40.401]}"
 "{'type': 'point', 'coordinates': [-3.7188898, 40.4309797]}"
 "{'type': 'point', 'coordinates': [-3.6887722, 40.4656779]}"
 "{'type': 'point', 'coordinates': [-3.7172213778842007, 40.40765396169372]}"
 "{'type': 'point', 'coordinates': [-3.66096, 40.43666]}"
 "{'type': 'point', 'coordinates': [-3.6738865, 40.4159569]}"
 "{'type': 'point', 'coordinates': [-3.6992759, 40.4060941]}"
 "{'type': 'point', 'coordinates': [-3.7110513, 40.4070358]}"
 "{'type': 'point', 'coordinates': [-3.7080833, 40.4411388]}"
 "{'type': 'point', 'coordinates': [-3.6751388, 40.4441388]}"
 "{'type': 'point', 'coordinates': [-3.7170448, 40.4253944]}"
 "{'type': 'point', 'coordinates': [-3.6752024, 40.4272582]}"
 "{'type': 'point', 'coordinates': [-3.6993465, 40.4309524]}"
 "{'type': 'point', 'coordinates': [-3.697895, 40.4222862]}"
```

```
for col in suelo.columns:
    print(f"Columna: {col}")
    print(suelo[col].unique())
    print("-" * 50)
```

```
Columna: ID_POLYGON
['98678c0f-70e2-49f6-85d0-40b7cb9292d4'
 'ae8c5627-affd-4a5d-8022-3b5e5ff31bbb'
 'cae746a8-2196-4f6e-9e4f-65d3ce3efec0' ...
 'b4a9dc2a-5536-4974-8b34-ad0347ac02e2'
 'bf0d01a8-ee01-411a-bdc0-b76cee6149e5'
 '98dc704f-23e9-4cb7-9691-101c1b8b56af']
--------------------------------------------------
Columna: SIOSE_CODE
['PMX(80ZEV_10SNE_05EDFnv_05VAP)' 'MTR' 'A(75MTR_25FDP)' ...
 'ECM(40OCT_30SNE_20VAP_10EDFea)' 'UDS(60ZAU_20EDFva_20SNE)'
 'UEN(65ZAU_10EDFva_10VAP_05LAA_05OCT_05SNE)']
--------------------------------------------------
Columna: SIOSE_XML
['<?xml version="1.0" encoding="UTF-8"?><POLIGONO Id="98678c0f-70e2-49f6-85d0-40b7cb9292d4" code="PMX(80ZEV_10SNE_05EDFnv_05VAP)"
 '<?xml version="1.0" encoding="UTF-8"?><POLIGONO Id="ae8c5627-affd-4a5d-8022-3b5e5ff31bbb" code="MTR" Sup_ha="2.8361"><COBERTURA
 '<?xml version="1.0" encoding="UTF-8"?><POLIGONO Id="cae746a8-2196-4f6e-9e4f-65d3ce3efec0" code="A(75MTR_25FDP)" Sup_ha="22.2027
 ...
 '<?xml version="1.0" encoding="UTF-8"?><POLIGONO Id="b4a9dc2a-5536-4974-8b34-ad0347ac02e2" code="A(60MTR_40CNF)" Sup_ha="4.9701"
 '<?xml version="1.0" encoding="UTF-8"?><POLIGONO Id="bf0d01a8-ee01-411a-bdc0-b76cee6149e5" code="A(50FDP_50MTR)" Sup_ha="3.895">
 '<?xml version="1.0" encoding="UTF-8"?><POLIGONO Id="98dc704f-23e9-4cb7-9691-101c1b8b56af" code="CHLsc" Sup_ha="24.6418"><COBERT
--------------------------------------------------
Columna: SUPERF_HA
[41.6024919   2.83609294 22.20272951 ...  4.97008837  3.89496853
 24.64184042]
--------------------------------------------------
Columna: CODIIGE
[123 330 340 113 311 320 140 161 210 312 112 172 234 250 260 111 313 114
 150 233 240 130 122 121 236 354 511 235 171 352 163 513 232 220 514 512
 411 162]
--------------------------------------------------
Columna: HILUCS
[130 631 500 660 610 110 120 330 632 310 410 200 430 620 340 140]
--------------------------------------------------
Columna: SELLADO
[ 10.    nan  25.    65.    64.5   60.    50.    85.    90.     5.
  40.    75.    80.   100.    70.    30.    20.    32.    55.    15.
  95.    67.    45.     6.    62.25  69.    92.5   64.    35.     7.
  27.    42.5   89.    48.    68.5   66.     1.    86.    19.25  93.
  32.75  84.75  87.     2.5   16.5   53.5   24.     3.75  16.25  91.
  42.    14.    33.75  77.    22.    31.25  74.5   10.5   13.75  79.5
   9.     3.    22.5   47.5    2.    12.    44.    96.     1.5   31.9
  84.6   83.    37.5   68.25  11.5   11.25  76.25   8.25  88.    75.5
  31.5   18.95  31.95  15.25  52.5   51.    82.5   49.    59.    33.5
   1.25  28.    56.65  22.75  77.5   56.    92.    19.    44.5    4.
  41.    90.2   59.5   52.    82.    96.5   51.05  40.5   31.75  13.5
  93.25  72.    48.75  29.75  29.    17.5   81.    74.    72.2   76.
  58.    50.75  41.85  18.    16.     6.75  94.25  55.5    9.75  79.
  39.    48.8   37.     2.25  82.1   15.75  57.5   36.    94.    84.
  33.    93.5   38.    73.    50.9   56.7    8.    11.9   11.    87.25
  17.    78.5   54.    68.    26.25   4.5   75.25  31.    43.    14.25
  76.6   65.05  98.     8.75   6.5   72.5   73.9   41.05  60.5   67.5
  18.25  92.35  62.    12.75  71.25  27.25  24.75  34.    85.75  71.5
   7.5   35.9   97.    48.9   29.25  13.    40.25  45.5   36.5   46.
  61.5   63.    78.75  67.75  62.5   23.    71.    15.5   81.5   26.
  19.75  57.75  31.15  63.6   42.25  61.    12.5   61.75  32.5   86.25
  94.9   50.5   21.    91.25  89.25  57.    66.75  48.25  39.8    9.9
```

```
for col in valor.columns:
    print(f"Columna: {col}")
    print(valor[col].unique())
    print("-" * 50)
```

```
Columna: ID_POLYGON
['00007cce-155b-40de-be0f-e0b4c841b720'
 '0001cccc-8b59-4f24-bce6-84582ca2f886'
 '00026eeb-8064-4d02-a427-a8bd89282162' ...
 'fff95dfc-50d8-47a7-988f-4577c7ed8e2c'
 'fffafa23-3a08-413b-9073-4cbb82a51784'
 '98dc704f-23e9-4cb7-9691-101c1b8b56af']
--------------------------------------------------
Columna: ID_COBERTURAS
[858 111 101 102 104 600 313 320 300 702 231 232 312 882 894 121 812 212
 131 103 316 813 860 290 352 881 831 333 823 833 841 854 701 851 821 511
 844 911 852 822 811 223 703 336 241 513 704 853 857 856 353 912 842 514
 922 897 884 832 859 411 921 843 892 900 896 855 834 883 222]
--------------------------------------------------
Columna: ID_ANCESTROS
['' '858' '600' '600,702' '882' '894' '812' '813' '860' '881' '831'
 '600,812' '600,813' '823' '833' '841' '854' '701' '851' '600,831' '821'
 '600,841' '600,823' '844' '600,911' '852' '822' '600,858' '811' '600,852'
 '600,854' '600,851' '600,881' '702' '703' '600,600' '600,704' '600,860'
```

```
 '600,853' '857' '856' '912' '842' '853' '911' '922' '600,897' '884'
 '600,857' '832' '600,703' '704' '600,856' '859' '600,842' '600,832' '921'
 '600,921' '600,822' '843' '892' '600,701' '600,922' '600,821' '900'
 '600,894' '600,811' '896' '855' '600,896' '600,884' '600,833' '897'
 '600,882' '600,859' '600,912' '600,892' '600,855' '600,600,702'
 '600,600,892' '600,843' '834' '883' '600,883' '600,900']
--------------------------------------------------
Columna: INTER_ID
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21]
--------------------------------------------------
Columna: INTER_ANCESTROS
['' '1' '1,2' '1,3' '1,6' '1,10' '1,7' '1,8' '1,5' '1,4' '1,9' '1,16'
 '1,11' '1,12' '1,13' '1,19' '1,14' '1,18' '1,6,7' '1,6,10' '1,15']
--------------------------------------------------
Columna: ATRIBUTOS
['' '21' '11' '12' '31' '23' '13' '24' '40' '22' '25' '41' '36,31' '28'
 '32' '46' '49' '33' '40,41' '47' '32,36' '44' '31,36' '48' '41,46' '45'
 '46,47' '32,35' '35,31' '31,35' '36,32']
--------------------------------------------------
Columna: SUPERF_HA
[ 1.511   0.68    0.4533 ...  2.3709  7.1083 24.6418]
--------------------------------------------------
Columna: SUPERF_POR
[100.    45.    30.    20.     5.    40.    15.    60.    42.    18.
  85.    10.    70.    55.    35.    50.    25.    80.    65.    75.
  90.    48.     6.     8.     2.    14.    95.    27.    12.     9.
  26.     4.     3.     2.5   17.5   56.    39.    22.75   3.25  32.9
   2.1    5.25   3.75   2.25   1.5    0.75  45.5   21.     3.5   22.5
   4.5   17.    16.    32.    24.    76.5   13.5    8.5   67.5   15.75
  12.25   7.    28.    42.75  28.5   19.     4.75   1.    34.    25.5
   4.25  10.5   58.5    6.5   16.25   8.75  52.    36.    24.5   22.
  81.    38.25  29.75   6.25   1.25  32.5    9.75  37.5   11.25   7.5
  44.    11.    63.    57.6   14.4   51.     0.5   35.75  13.     8.25
   5.5   12.5   38.    98.    34.3   29.4    9.8   71.25  14.25   9.5
  54.6    4.9   31.5   64.    42.25  99.    74.25  24.75  38.5   16.1
  11.9   29.75  97.    26.25  16.5   33.    55.25  21.25  54.    33.75
  18.75  30.25  19.25  31.5   27.5    6.75  49.5   19.5   52.5    1.75
  18.2    9.8    9.9    2.1   27.5   66.     8.1   82.    40.5   47.5
```

Frecuencia de cada columna

```
for col in trips.columns:
    print(f"Columna: {col}")
    print(trips[col].value_counts().to_frame())
    print("-" * 50)
```

```
Columna: fecha
            count
fecha
2023-02-01  11442
2023-02-02  11069
2023-02-09  10814
2023-02-15  10766
2023-02-10  10579
2023-02-08  10576
2023-02-14  10341
2023-02-03  10166
2023-02-06  10118
2023-02-17  10067
2023-02-13   9802
2023-02-16   9684
2023-02-07   8986
2023-02-04   8903
2023-02-11   8383
2023-02-05   8026
2023-02-12   7801
2023-02-18    971
--------------------------------------------------
Columna: idBike
        count
idBike
7949.0    178
7670.0    166
7166.0    161
6270.0    157
7971.0    150
...       ...
5307.0      1
5724.0      1
7951.0      1
7127.0      1
5730.0      1

[3134 rows x 1 columns]
--------------------------------------------------
Columna: fleet
        count
fleet
```

```
1.0    167692
2.0       802
--------------------------------------------------
Columna: trip_minutes
              count
trip_minutes
0.23           2929
0.25           2552
0.27           2435
0.15           2206
0.17           2170
...             ...
57.67             1
1472.18           1
1495.15           1
51.97             1
```

```
for col in suelo.columns:
    print(f"Columna: {col}")
    print(suelo[col].value_counts().to_frame())
    print("-" * 50)
```

```
Columna: ID_POLYGON
                                      count
ID_POLYGON
98dc704f-23e9-4cb7-9691-101c1b8b56af      1
98678c0f-70e2-49f6-85d0-40b7cb9292d4      1
ae8c5627-affd-4a5d-8022-3b5e5ff31bbb      1
cae746a8-2196-4f6e-9e4f-65d3ce3efec0      1
febe7109-4d88-4b87-b3b3-043bb22ea29f      1
...                                     ...
35ad3fe5-9ed6-46a3-b1c5-09aa10e49fa4      1
B4C9A55F-70CB-4473-AF1B-B5FA703142A2      1
58e3aea0-aa6c-4e20-9fbb-fac9cbbc1b75      1
0c6696da-403d-4b41-a04b-f7aae147ec91      1
31b05faf-32b6-49f0-bb38-425bb18bb261      1

[44858 rows x 1 columns]
--------------------------------------------------
Columna: SIOSE_CODE
                                      count
SIOSE_CODE
CHLsc                                  2374
LOLsc                                  1715
MTR                                    1519
PST                                    1232
LVIsc                                   828
...                                     ...
UEN(55EDFem_40SNE_05LAA)                  1
UEN(45EDFvd_30SNE_20VAP_05ZAU)            1
UEN(40EDFva_20VAP_15ZAU_15OCT_10SNE)      1
EDP(65OCT_30ZAU_05VAP)                    1
UEN(65EDFem_20ZEV_10ZAU_05VAP)            1

[18061 rows x 1 columns]
--------------------------------------------------
Columna: SIOSE_XML
                                                      count
SIOSE_XML
<?xml version="1.0" encoding="UTF-8"?><POLIGONO...      1
<?xml version="1.0" encoding="UTF-8"?><POLIGONO...      1
<?xml version="1.0" encoding="UTF-8"?><POLIGONO...      1
<?xml version="1.0" encoding="UTF-8"?><POLIGONO...      1
<?xml version="1.0" encoding="UTF-8"?><POLIGONO...      1
...                                                   ...
<?xml version="1.0" encoding="UTF-8"?><POLIGONO...      1
<?xml version="1.0" encoding="UTF-8"?><POLIGONO...      1
<?xml version="1.0" encoding="UTF-8"?><POLIGONO...      1
<?xml version="1.0" encoding="UTF-8"?><POLIGONO...      1
<?xml version="1.0" encoding="UTF-8"?><POLIGONO...      1

[44858 rows x 1 columns]
--------------------------------------------------
Columna: SUPERF_HA
             count
SUPERF_HA
24.641840        1
41.602492        1
2.836093         1
22.202730        1
```

```
for col in valor.columns:
    print(f"Columna: {col}")
    print(valor[col].value_counts().to_frame())
    print("-" * 50)
```

```
Columna: ID_POLYGON
                                     count
ID_POLYGON
d62d9129-03a4-4502-a54f-8d2096e5b944     21
e2e6aa53-d164-49bf-80ab-0e0bb719c7d5     21
3915dae2-4fb3-4fcc-8579-07d074e121a8     21
41b31f31-bc38-4b61-b6f8-2bb7b3e03237     21
61cb7169-4ff2-49d4-ba93-cca3d9542792     20
...                                     ...
250bbc8b-7eec-47e3-b2b9-f4e855c4ff46      1
250c71e8-871b-4d40-bb89-cc831772eef4      1
2514534a-9fd7-4c1f-8913-8565651397bd      1
25170ade-5c45-421c-b8c0-17ab743bf58e      1
d35010e2-3a1b-4de5-9d05-d4dd22210f75      1

[44858 rows x 1 columns]
--------------------------------------------------
Columna: ID_COBERTURAS
               count
ID_COBERTURAS
600            18305
101            14861
104            14781
320            13765
102            10786
...              ...
896                7
897                4
883                2
834                1
222                1

[69 rows x 1 columns]
--------------------------------------------------
Columna: ID_ANCESTROS
              count
ID_ANCESTROS
600           45094
              44858
812           19864
813            6000
854            3349
...             ...
600,897           2
600,859           2
600,600,702       2
600,883           2
600,900           2

[85 rows x 1 columns]
--------------------------------------------------
Columna: INTER_ID
          count
INTER_ID
1         44858
2         34278
3         33011
4         20587
```

P4. ¿Existen registros que necesitan limpieza?

trips alaprecer el documento csv fueron llenados intercalando entre datos y filas vacias, eliminaremos esas filas para ver los nan aleatorios que si seran un problema al procesar los datos.

```
Empieza a programar o a crear código con IA.
```

```
# Elimina filas donde *todas* las columnas son NaN
trips = trips.dropna(how='all').reset_index(drop=True)
```

```
trips.head(10)
```

	fecha	idBike	fleet	trip_minutes	geolocation_unlock	address_unlock	unlock_date	locktype	unlocktype	geolocation_lock	addr
0	2023-02-01	7337.0	1.0	5.52	{'type': 'point', 'coordinates': [-3.6956178, ...	'calle jesús nº 1'	2023-02-01T00:00:10	station	station	{'type': 'Point', 'coordinates': [-3.7088337, ...	'p ceba
1	2023-02-01	5098.0	1.0	0.32	{'type': 'point', 'coordinates': [-3.7022591, ...	'glorieta de embajadores nº 2'	2023-02-01T00:00:25	station	station	{'type': 'Point', 'coordinates': [-3.7022591, ...	'g emb
2	2023-02-01	6519.0	1.0	0.27	{'type': 'point', 'coordinates': [-3.6894193, ...	'calle antonio maura nº 15'	2023-02-01T00:00:36	station	station	{'type': 'Point', 'coordinates': [-3.6894193, ...	'call ma
3	2023-02-01	2551.0	1.0	8.58	{'type': 'point', 'coordinates': [-3.7022591, ...	'glorieta de embajadores nº 2'	2023-02-01T00:00:53	station	station	{'type': 'Point', 'coordinates': [-3.6991147, ...	'ca
4	2023-02-01	6519.0	1.0	0.20	{'type': 'point', 'coordinates': [-3.6894193, ...	'calle antonio maura nº 15'	2023-02-01T00:00:57	station	station	{'type': 'Point', 'coordinates': [-3.6894193, ...	'call ma
5	2023-02-01	7620.0	1.0	0.25	{'type': 'point', 'coordinates': [-3.6840229, ...	'calle alcalá nº 95'	2023-02-01T00:01:09	station	station	{'type': 'Point', 'coordinates': [-3.6840229, ...	'calle
6	2023-02-01	6355.0	1.0	23.07	{'type': 'point', 'coordinates': [-3.6758555, ...	'calle juan bravo nº 50'	2023-02-01T00:01:16	station	station	{'type': 'Point', 'coordinates': [-3.6758555, ...	' bra
7	2023-02-01	4911.0	1.0	7.83	{'type': 'point', 'coordinates': [-3.6758383, ...	'plaza de felipe ii'	2023-02-01T00:01:36	station	station	{'type': 'Point', 'coordinates': [-3.70239, 40...	dese
8	2023-02-01	7620.0	1.0	0.22	{'type': 'point', 'coordinates': [-3.6840229, ...	'calle alcalá nº 95'	2023-02-01T00:01:38	station	station	{'type': 'Point', 'coordinates': [-3.6840229, ...	'calle
9	2023-02-01	7780.0	1.0	9.12	{'type': 'point', 'coordinates': [-3.6989925, ...	'calle marqués de viana nº 3'	2023-02-01T00:01:40	station	station	{'type': 'Point', 'coordinates': [-3.7046666, ...	'g que

```
print(len(trips))
```

```
168494
```

```
nulos_trips = trips.isnull().sum()
nulos_trips
```

	0
fecha	0
idBike	0
fleet	0
trip_minutes	0
geolocation_unlock	0
address_unlock	0
unlock_date	0
locktype	0
unlocktype	0
geolocation_lock	0
address_lock	0
lock_date	0
station_unlock	380
dock_unlock	380
unlock_station_name	380
station_lock	476
dock_lock	476
lock_station_name	476

dtype: int64

Analizando nulos especificos en trips.

¿Están vacías en las mismas filas?

Grupo de columnas	Nulos completos	¿Qué significa?
`station_unlock`, `dock_unlock`, `unlock_station_name`	**380**	Hay 380 viajes que **no tienen información sobre la estación de inicio** del viaje.
`station_lock`, `dock_lock`, `lock_station_name`	**476**	Hay 476 viajes que **no tienen información sobre la estación de fin** del viaje.
Ambos grupos sin datos	**308**	Hay 308 viajes que **no tienen estación de inicio ni de fin** — es decir, no tienen puntos de anclaje registrad

```
# Ver si las columnas 'unlock' tienen nulos en las mismas filas
unlock_cols = ['station_unlock', 'dock_unlock', 'unlock_station_name']
nulos_unlock = trips[unlock_cols].isnull().all(axis=1)

# Ver si las columnas 'lock' tienen nulos en las mismas filas
lock_cols = ['station_lock', 'dock_lock', 'lock_station_name']
nulos_lock = trips[lock_cols].isnull().all(axis=1)

# Cuántas filas tienen el grupo unlock nulo completo
print("Filas con todos los datos de unlock nulos:", nulos_unlock.sum())

# Cuántas filas tienen el grupo lock nulo completo
print("Filas con todos los datos de lock nulos:", nulos_lock.sum())

# ¿Cuántas filas tienen ambos grupos nulos?
print("Filas con ambos grupos nulos:", (nulos_unlock & nulos_lock).sum())
```

```
Filas con todos los datos de unlock nulos: 380
Filas con todos los datos de lock nulos: 476
Filas con ambos grupos nulos: 308
```

Los 308 viajes que no tienen estación de inicio ni de fin (station_unlock, station_lock, etc.) sí tienen coordenadas de inicio y fin (geolocation_unlock y geolocation_lock).

```
# Extraer los viajes sin estación ni de inicio ni de fin
sin_estaciones = trips[nulos_unlock & nulos_lock]

# Ver cuántos tienen coordenadas en unlock y lock
con_coords_unlock = sin_estaciones['geolocation_unlock'].notnull().sum()
con_coords_lock = sin_estaciones['geolocation_lock'].notnull().sum()

print("Viajes sin estaciones pero con coordenadas de inicio:", con_coords_unlock)
print("Viajes sin estaciones pero con coordenadas de fin:", con_coords_lock)
```

```
Viajes sin estaciones pero con coordenadas de inicio: 308
Viajes sin estaciones pero con coordenadas de fin: 308
```

Analizando los que no tienen lock y unlock nulos al mismo tiempo

```
solo_nulo_unlock = (nulos_unlock & ~nulos_lock).sum()
solo_nulo_lock = (nulos_lock & ~nulos_unlock).sum()
```

Ambos casos tienen coordenadas utiles

```
# Filas con solo unlock nulo
solo_unlock_nulo = trips[nulos_unlock & ~nulos_lock]
coordenadas_lock_validas = solo_unlock_nulo['geolocation_lock'].notnull().sum()

# Filas con solo lock nulo
solo_lock_nulo = trips[nulos_lock & ~nulos_unlock]
coordenadas_unlock_validas = solo_lock_nulo['geolocation_unlock'].notnull().sum()

print("Viajes con unlock nulo pero coordenadas de fin:", coordenadas_lock_validas)
print("Viajes con lock nulo pero coordenadas de inicio:", coordenadas_unlock_validas)
```

```
Viajes con unlock nulo pero coordenadas de fin: 72
Viajes con lock nulo pero coordenadas de inicio: 168
```

```
coordenadas_lock_validas
```

```
np.int64(72)
```

RESUMEN

Tipo de nulo	Filas	¿Tiene coordenadas útiles?	¿Conservar?	Comentario
Ambos grupos nulos	308	✅ Sí (inicio y fin)	✔ **Sí**	No tienen estaciones, pero tienen geolocalización completa
Solo `unlock` nulo	72	✅ Sí (coordenada de fin)	✔ **Sí**	Falta estación de inicio, pero puede ubicarse espacialmente
Solo `lock` nulo	168	✅ Sí (coordenada de inicio)	✔ **Sí**	Falta estación de fin, pero se puede analizar origen
🚫 Total sin coordenadas	0	❌ No	❌ **No**	No hay registros que realmente debamos eliminar ahora

```
def clasificar_viaje(row):
    unlock = pd.notnull(row['station_unlock'])
    lock = pd.notnull(row['station_lock'])

    if unlock and lock:
        return "Estación inicio y fin"
    elif unlock or lock:
        return "Solo una estación"
    else:
        return "Sin estaciones"
trips_tmp=trips
trips_tmp['tipo_viaje'] = trips.apply(clasificar_viaje, axis=1)
```

```
trips_tmp
```

	unlock_date	locktype	unlocktype	geolocation_lock
	2023-02-01T00:00:10	station	station	{'type': 'Point', 'coordinates': [-3.7088337, ...
	2023-02-01T00:00:25	station	station	{'type': 'Point', 'coordinates': [-3.7022591, ...
	2023-02-01T00:00:36	station	station	{'type': 'Point', 'coordinates': [-3.6894193, ...
	2023-02-01T00:00:53	station	station	{'type': 'Point', 'coordinates': [-3.6991147, ...
	2023-02-01T00:00:57	station	station	{'type': 'Point', 'coordinates': [-3.6894193, ...
	...	...	...	...
	2023-02-18T04:25:31	station	station	{'type': 'Point', 'coordinates': [-3.7065376, ...
	2023-02-18T04:27:30	station	station	{'type': 'Point', 'coordinates': [-3.7074453, ...
	2023-02-18T04:39:38	station	station	{'type': 'Point', 'coordinates': [-3.6726019, ...
	2023-02-18T04:42:50	station	station	{'type': 'Point', 'coordinates': [-3.7064516, ...
	2023-02-18T07:22:48	station	station	{'type': 'Point', 'coordinates': [-3.6933498, ...

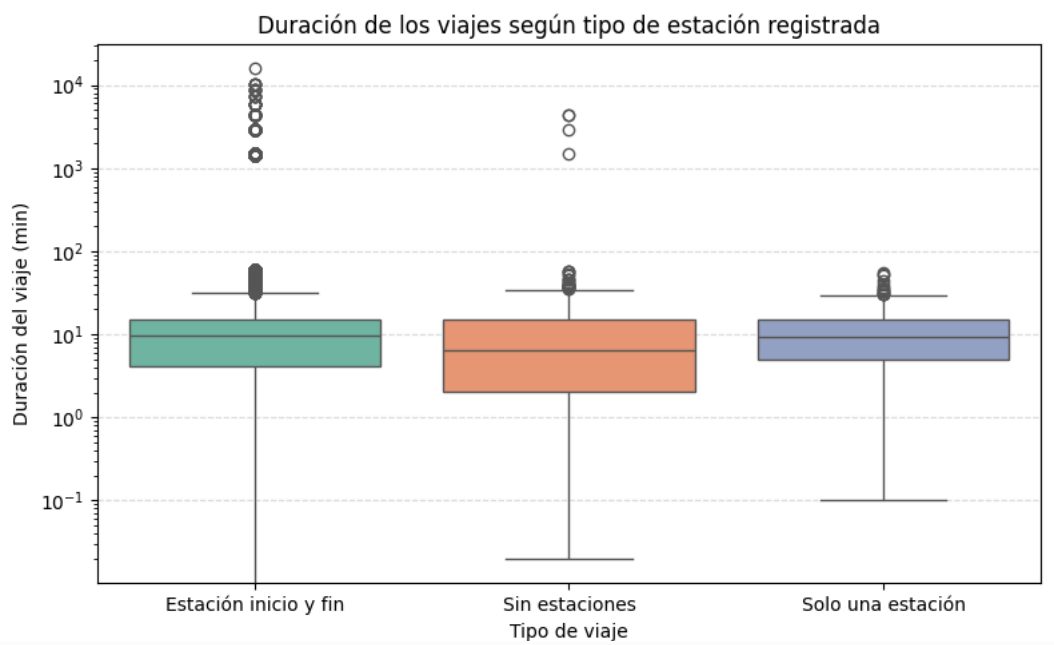
```
trips_tmp['tipo_viaje'].unique()
```

```
array(['Estación inicio y fin', 'Sin estaciones', 'Solo una estación'],
      dtype=object)
```

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 5))
sns.boxplot(data=trips_tmp, x='tipo_viaje', y='trip_minutes', palette='Set2')

plt.yscale('log')  # Escala logarítmica si hay valores extremos
plt.title("Duración de los viajes según tipo de estación registrada")
```

```
plt.xlabel("Tipo de viaje")
plt.ylabel("Duración del viaje (min)")
plt.grid(True, axis='y', linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()
```

```
<ipython-input-40-15ee69d9db79>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le
```



El tamaño de los nan de station_unlock, dock_unlock,unlock_station_name,station_lock,dock_lock,lock_station_name. Es pequeño comaprado con el total de datos asi que borraremos sus filas que las contienen

```
nulos_puntuales_promedio = nulos_trips['station_unlock'] + nulos_trips['dock_unlock']+nulos_trips['unlock_station_name']+nulos_trips['sta
cant_nulos_promedio = nulos_puntuales_promedio/6
cant_nulos_promedio
```

```
np.float64(428.0)
```

```
print(len(trips))
```

```
168494
```

```
trips_no_null = trips.dropna()
trips_no_null
```

	fecha	idBike	fleet	trip_minutes	geolocation_unlock	address_unlock	unlock_date	locktype	unlocktype	geolocation_lock
0	2023-02-01	7337.0	1.0	5.52	{'type': 'point', 'coordinates': [-3.6956178, ...	'calle jesús nº 1'	2023-02-01T00:00:10	station	station	{'type': 'Point', 'coordinates': [-3.7088337, ...
1	2023-02-01	5098.0	1.0	0.32	{'type': 'point', 'coordinates': [-3.7022591, ...	'glorieta de embajadores nº 2'	2023-02-01T00:00:25	station	station	{'type': 'Point', 'coordinates': [-3.7022591, ...
2	2023-02-01	6519.0	1.0	0.27	{'type': 'point', 'coordinates': [-3.6894193, ...	'calle antonio maura nº 15'	2023-02-01T00:00:36	station	station	{'type': 'Point', 'coordinates': [-3.6894193, ...
3	2023-02-01	2551.0	1.0	8.58	{'type': 'point', 'coordinates': [-3.7022591, ...	'glorieta de embajadores nº 2'	2023-02-01T00:00:53	station	station	{'type': 'Point', 'coordinates': [-3.6991147, ...
4	2023-02-01	6519.0	1.0	0.20	{'type': 'point', 'coordinates': [-3.6894193, ...	'calle antonio maura nº 15'	2023-02-01T00:00:57	station	station	{'type': 'Point', 'coordinates': [-3.6894193, ...
...	...	...	...	...	...	...	...	...	...	...
168489	2023-02-18	7492.0	1.0	1.33	{'type': 'point', 'coordinates': [-3.7065376, ...	'calle jacometrezo nº 3'	2023-02-18T04:25:31	station	station	{'type': 'Point', 'coordinates': [-3.7065376, ...
168490	2023-02-18	5538.0	1.0	9.48	{'type': 'point', 'coordinates': [-3.7065376, ...	'calle jacometrezo nº 3'	2023-02-18T04:27:30	station	station	{'type': 'Point', 'coordinates': [-3.7074453, ...
168491	2023-02-18	6473.0	1.0	0.47	{'type': 'point', 'coordinates': [-3.6726019, ...	'calle doctor esquerdo nº 191'	2023-02-18T04:39:38	station	station	{'type': 'Point', 'coordinates': [-3.6726019, ...
168492	2023-02-18	5252.0	1.0	7.43	{'type': 'point', 'coordinates': [-3.690358964...	'calle sodio nº 1b'	2023-02-18T04:42:50	station	station	{'type': 'Point', 'coordinates': [-3.7064516, ...
168493	2023-02-18	3077.0	1.0	0.12	{'type': 'point', 'coordinates': [-3.6933498, ...	'plaza cibeles'	2023-02-18T07:22:48	station	station	{'type': 'Point', 'coordinates': [-3.6933498, ...

167946 rows × 19 columns

```
nulos_trips_no_null = trips_no_null.isnull().sum()
nulos_trips_no_null
```

	0
fecha	0
idBike	0
fleet	0
trip_minutes	0
geolocation_unlock	0
address_unlock	0
unlock_date	0
locktype	0
unlocktype	0
geolocation_lock	0
address_lock	0
lock_date	0
station_unlock	0
dock_unlock	0
unlock_station_name	0
station_lock	0
dock_lock	0
lock_station_name	0
tipo_viaje	0

dtype: int64

```
print(len(trips_no_null))
```

```
167946
```

```
print(len(trips)-len(trips_no_null))
```

```
548
```

Los datos de localizacion deben extraerse la latitud y longitud

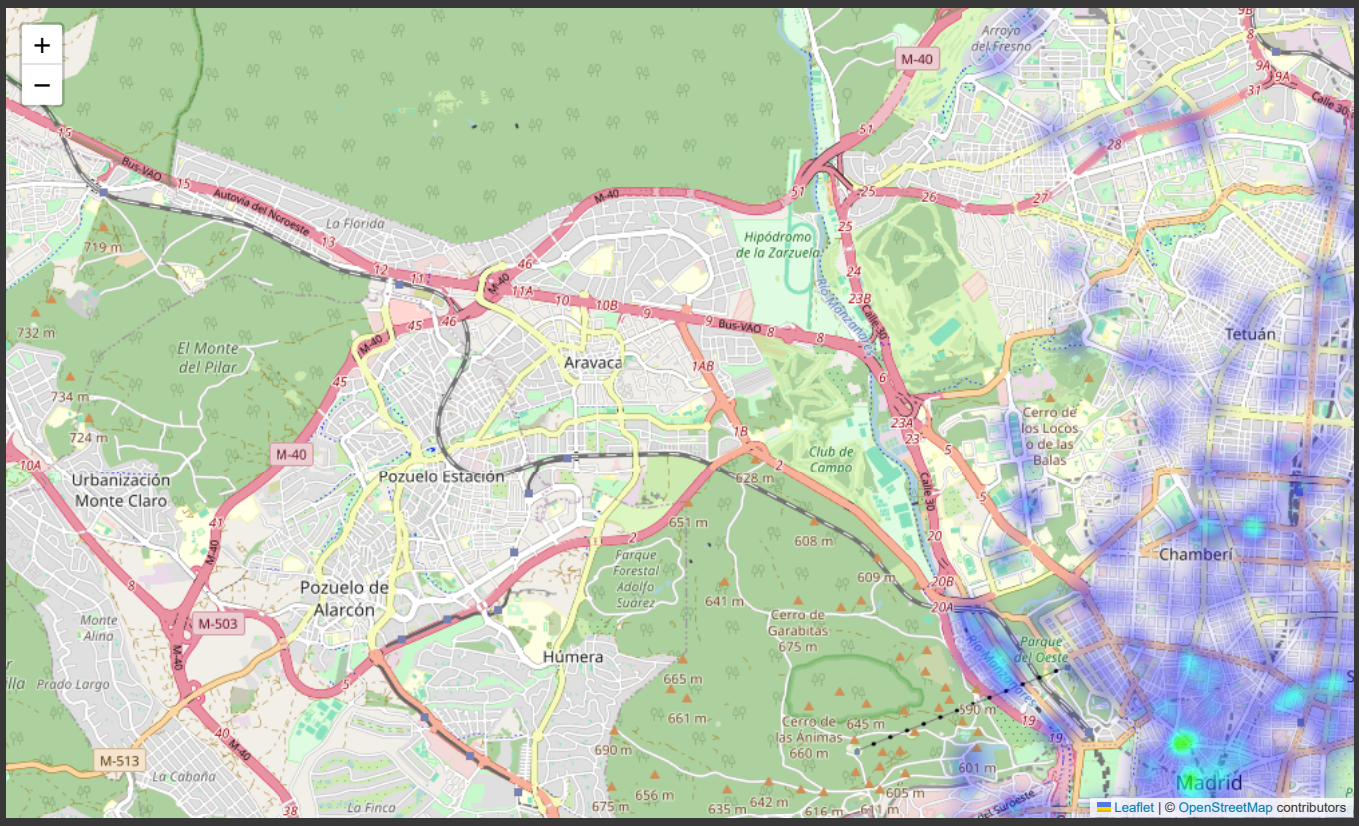
```
import folium
from folium.plugins import HeatMap
import ast

# 🔹 Paso 1: función para extraer lat/lon de geolocation_unlock
def extraer_coords(punto):
    try:
        geo = ast.literal_eval(punto) if isinstance(punto, str) else punto
        return pd.Series([geo["coordinates"][1], geo["coordinates"][0]])  # lat, lon
    except:
        return pd.Series([None, None])

# 🔹 Paso 2: aplicar la función y crear columnas lat/lon
trips[['lat_unlock', 'lon_unlock']] = trips['geolocation_unlock'].apply(extraer_coords)
```

```
trips
```

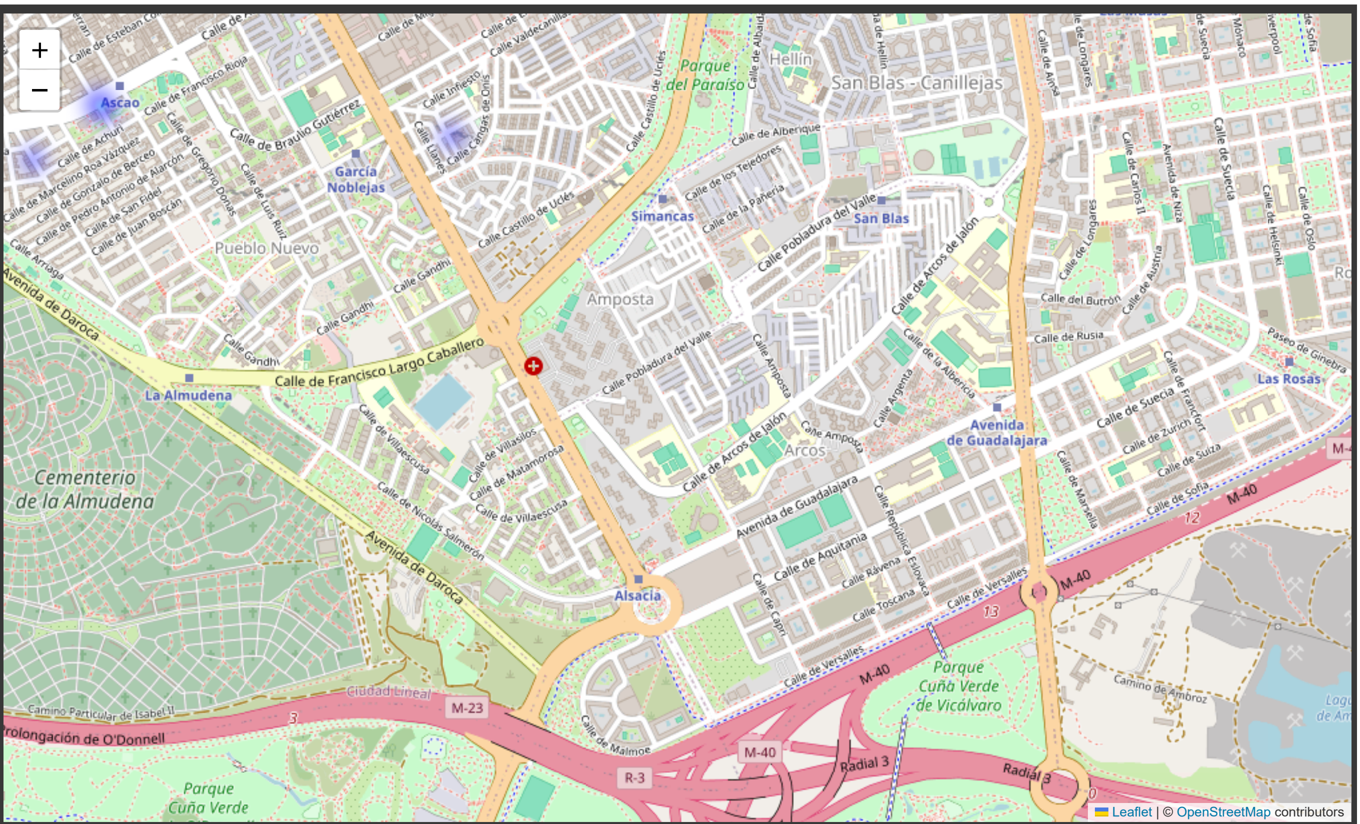
	locktype	unlocktype	geolocation_lock
	station	station	{'type': 'Point', 'coordinates': [-3.7088337, ...
	station	station	{'type': 'Point', 'coordinates': [-3.7022591, ...
	station	station	{'type': 'Point', 'coordinates': [-3.6894193, ...
	station	station	{'type': 'Point', 'coordinates': [-3.6991147, ...
	station	station	{'type': 'Point', 'coordinates': [-3.6894193, ...
	...	...	...
	station	station	{'type': 'Point', 'coordinates': [-3.7065376, ...
	station	station	{'type': 'Point', 'coordinates': [-3.7074453, ...
	station	station	{'type': 'Point', 'coordinates': [-3.6726019, ...
	station	station	{'type': 'Point', 'coordinates': [-3.7064516, ...
	station	station	{'type': 'Point', 'coordinates': [-3.6933498, ...

```
# 🔹 Paso 3: crear dataset para HeatMap
heat_df = trips[['lat_unlock', 'lon_unlock', 'trip_minutes']].dropna()
heat_points = heat_df.values.tolist()

# 🔹 Paso 4: crear mapa centrado en Madrid
mapa = folium.Map(location=[40.4168, -3.7038], zoom_start=13)  # Coordenadas de Madrid

# 🔹 Paso 5: añadir HeatMap (pesado por duración del viaje)
HeatMap(heat_points, radius=10, blur=15, max_zoom=1).add_to(mapa)
```

```
# 🔹 Mostrar el mapa
mapa
#{'type': 'point', 'coordinates': [-3.7022591, 40.4056107]}
#'type': 'Point', 'coordinates': [-3.6956178, 40.4132798]] LONGITUD,LATITUD / Formato estándar GeoJSON / GeoPandas / shapely
```




```
# Para mapa de destino
trips[['lat_lock', 'lon_lock']] = trips['geolocation_lock'].apply(extraer_coords)
heat_df_lock = trips[['lat_lock', 'lon_lock', 'trip_minutes']].dropna()
heat_points_lock = heat_df_lock.values.tolist()

# Nuevo mapa de cierre de viaje
mapa_lock = folium.Map(location=[40.4168, -3.7038], zoom_start=13)
HeatMap(heat_points_lock, radius=10, blur=15, max_zoom=1).add_to(mapa_lock)
mapa_lock
```


+
−
Ascao
García Noblejas
Pueblo Nuevo
Parque del Paraíso
Hellín
San Blas - Canillejas
Simancas
San Blas
Amposta
La Almudena
Calle de Francisco Largo Caballero
Cementerio de la Almudena
Arcos
Avenida de Guadalajara
Las Rosas
Avenida de Daroca
Alsacia
M-40
M-23
R-3
Radial 3
Parque Cuña Verde de Vicálvaro
Parque Cuña Verde
Prolongación de O'Donnell
Leaflet | © OpenStreetMap contributors

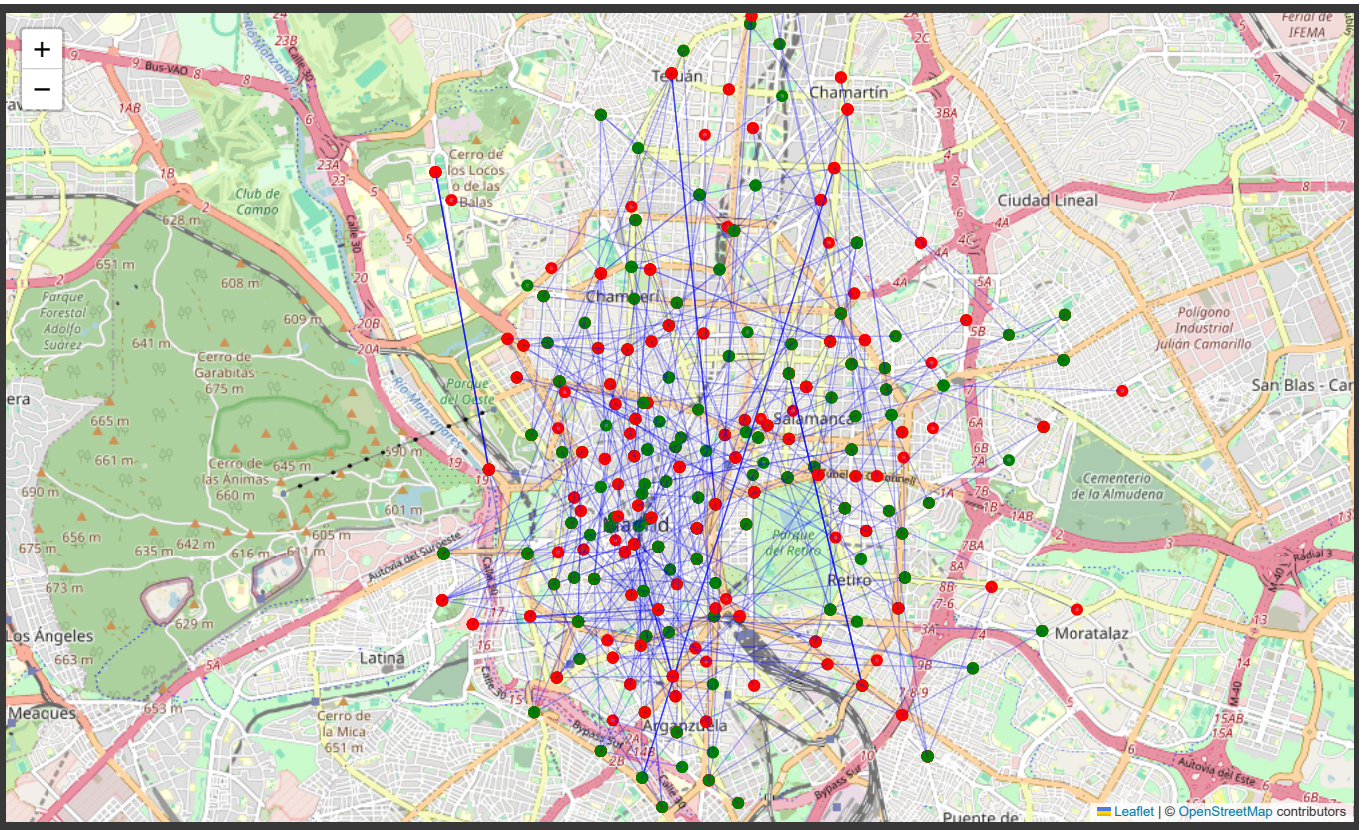
```
trips
```

k_date	locktype	unlocktype	geolocation_lock
023-02-0:00:10	station	station	{'type': 'Point', 'coordinates': [-3.7088337, ...
023-02-0:00:25	station	station	{'type': 'Point', 'coordinates': [-3.7022591, ...
023-02-0:00:36	station	station	{'type': 'Point', 'coordinates': [-3.6894193, ...
023-02-0:00:53	station	station	{'type': 'Point', 'coordinates': [-3.6991147, ...
023-02-0:00:57	station	station	{'type': 'Point', 'coordinates': [-3.6894193, ...
...	...	...	...
023-02-4:25:31	station	station	{'type': 'Point', 'coordinates': [-3.7065376, ...
023-02-4:27:30	station	station	{'type': 'Point', 'coordinates': [-3.7074453, ...
023-02-4:39:38	station	station	{'type': 'Point', 'coordinates': [-3.6726019, ...
023-02-4:42:50	station	station	{'type': 'Point', 'coordinates': [-3.7064516, ...
023-02-7:22:48	station	station	{'type': 'Point', 'coordinates': [-3.6933498, ...

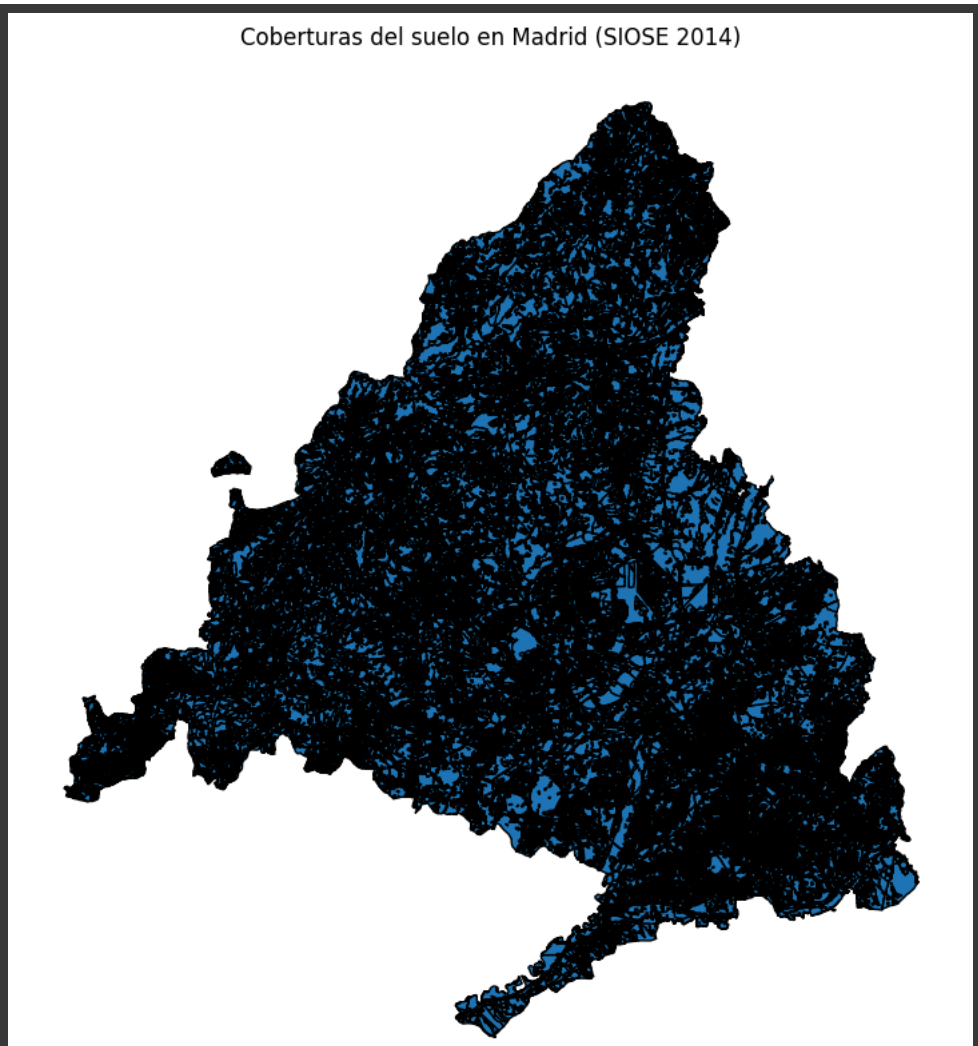
```
# Crear mapa base centrado en Madrid
mapa_ambos = folium.Map(location=[40.4168, -3.7038], zoom_start=13)
```

```
# Mostrar máximo 500 puntos para no sobrecargar (puedes ajustar)
subset = trips.dropna(subset=['lat_unlock', 'lon_unlock', 'lat_lock', 'lon_lock']).head(500)

# Añadir marcadores
for _, row in subset.iterrows():
    # Inicio (verde)
    folium.CircleMarker(
        location=[row['lat_unlock'], row['lon_unlock']],
        radius=3,
        color='green',
        fill=True,
        fill_opacity=0.7
    ).add_to(mapa_ambos)

    # Fin (rojo)
    folium.CircleMarker(
        location=[row['lat_lock'], row['lon_lock']],
        radius=3,
        color='red',
        fill=True,
        fill_opacity=0.7
    ).add_to(mapa_ambos)

    # Línea entre ambos
    folium.PolyLine(
        locations=[[row['lat_unlock'], row['lon_unlock']], [row['lat_lock'], row['lon_lock']]],
        color='blue',
        weight=1,
        opacity=0.3
    ).add_to(mapa_ambos)
```

```
mapa_ambos
```




```
import matplotlib.pyplot as plt

# Graficar todo el shapefile
suelo.plot(figsize=(10, 10), edgecolor='black')
plt.title("Coberturas del suelo en Madrid (SIOSE 2014)")
plt.axis('off')
plt.show()
```



Dar un foramto correcto a las fechas apra su uso

```
#2023-02-01T00:00:10
```

```
import pandas as pd

# Asegúrate de que tus columnas están en formato de cadena (string)
# Luego conviértelas al tipo datetime
trips['unlock_date'] = pd.to_datetime(trips['unlock_date'])
trips['lock_date'] = pd.to_datetime(trips['lock_date'])
```

```
#trips.drop(columns=['duration'], inplace=True)
```

```
# Calcular duración del viaje
trips_duration = trips['lock_date'] - trips['unlock_date'] #trips['duration'] = trips['lock_date'] - trips['unlock_date']

# Filtrar por fecha
trips_january = trips[trips['unlock_date'].dt.month == 1]
```

```
trips_duration
```

	0
0	0 days 00:05:31
1	0 days 00:00:19
2	0 days 00:00:16
3	0 days 00:08:35
4	0 days 00:00:12
...	...
168489	0 days 00:01:20
168490	0 days 00:09:29
168491	0 days 00:00:28
168492	0 days 00:07:26
168493	0 days 00:00:07

168494 rows × 1 columns

dtype: timedelta64[ns]

```
trips
```

	fecha	idBike	fleet	trip_minutes	geolocation_unlock	address_unlock	unlock_date	locktype	unlocktype	geolocation_lock
0	2023-02-01	7337.0	1.0	5.52	{'type': 'point', 'coordinates': [-3.6956178, ...	'calle jesús nº 1'	2023-02-01 00:00:10	station	station	{'type': 'Point', 'coordinates': [-3.7088337, ...
1	2023-02-01	5098.0	1.0	0.32	{'type': 'point', 'coordinates': [-3.7022591, ...	'glorieta de embajadores nº 2'	2023-02-01 00:00:25	station	station	{'type': 'Point', 'coordinates': [-3.7022591, ...
2	2023-02-01	6519.0	1.0	0.27	{'type': 'point', 'coordinates': [-3.6894193, ...	'calle antonio maura nº 15'	2023-02-01 00:00:36	station	station	{'type': 'Point', 'coordinates': [-3.6894193, ...
3	2023-02-01	2551.0	1.0	8.58	{'type': 'point', 'coordinates': [-3.7022591, ...	'glorieta de embajadores nº 2'	2023-02-01 00:00:53	station	station	{'type': 'Point', 'coordinates': [-3.6991147, ...
4	2023-02-01	6519.0	1.0	0.20	{'type': 'point', 'coordinates': [-3.6894193, ...	'calle antonio maura nº 15'	2023-02-01 00:00:57	station	station	{'type': 'Point', 'coordinates': [-3.6894193, ...
...	...	...	...	...	...	...	...	...	...	...
168489	2023-02-18	7492.0	1.0	1.33	{'type': 'point', 'coordinates': [-3.7065376, ...	'calle jacometrezo nº 3'	2023-02-18 04:25:31	station	station	{'type': 'Point', 'coordinates': [-3.7065376, ...
168490	2023-02-18	5538.0	1.0	9.48	{'type': 'point', 'coordinates': [-3.7065376, ...	'calle jacometrezo nº 3'	2023-02-18 04:27:30	station	station	{'type': 'Point', 'coordinates': [-3.7074453, ...
168491	2023-02-18	6473.0	1.0	0.47	{'type': 'point', 'coordinates': [-3.6726019, ...	'calle doctor esquerdo nº 191'	2023-02-18 04:39:38	station	station	{'type': 'Point', 'coordinates': [-3.6726019, ...
168492	2023-02-18	5252.0	1.0	7.43	{'type': 'point', 'coordinates': [-3.690358964...	'calle sodio nº 1b'	2023-02-18 04:42:50	station	station	{'type': 'Point', 'coordinates': [-3.7064516, ...
168493	2023-02-18	3077.0	1.0	0.12	{'type': 'point', 'coordinates': [-3.6933498, ...	'plaza cibeles'	2023-02-18 07:22:48	station	station	{'type': 'Point', 'coordinates': [-3.6933498, ...

168494 rows × 23 columns

CONVERSION DE UNIDADES

Archivo suelo

```
#USOS DEL SUELO
hilucs_dict = {
    110: "1_1_Agriculture",
    120: "1_2_Forestry",
    130: "1_3_MiningAndQuarrying",
    140: "1_4_AquacultureAndFishing",
    200: "2_SecondaryProduction",
    210: "2_1_CommercialServices"
```

```
    330: "3_3_CommunityServices",
    340: "3_4_CulturalEntertainmentAndRecreationalServices",
    410: "4_1_TransportNetworks",
    430: "4_3_Utilities",
    500: "5_ResidentialUse",
    610: "6_1_TransitionalAreas",
    620: "6_2_AbandonedAreas",
    631: "6_3_1_LandAreasNotInOtherEconomicUse",
    632: "6_3_2_WaterAreasNotInOtherEconomicUse",
    660: "6_6_NotKnownUse"
}
```

```
suelo["HILUCS"] = suelo["HILUCS"].replace(hilucs_dict)
suelo
```

	ID_POLYGON	SIOSE_CODE	SIOSE_XML	SUPERF_HA	CODIIGE	HILUC
0	98678c0f-70e2-49f6-85d0-40b7cb9292d4	PMX(80ZEV_10SNE_05EDFnv_05VAP)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	41.602492	123	1_3_MiningAndQuarryin
1	ae8c5627-affd-4a5d-8022-3b5e5ff31bbb	MTR	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	2.836093	330	6_3_1_LandAreasNotInOtherEconomicUs
2	cae746a8-2196-4f6e-9e4f-65d3ce3efec0	A(75MTR_25FDP)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	22.202730	340	6_3_1_LandAreasNotInOtherEconomicUs
3	febe7109-4d88-4b87-b3b3-043bb22ea29f	UDS(65ZAU_15EDFva_10VAP_05LAA_05SNE)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	1.938984	113	5_ResidentialUs
4	443f3abb-daa3-4b60-9ec0-a50496f88ab1	DHS(60PST_40FDP)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	41.033844	311	6_3_1_LandAreasNotInOtherEconomicUs
...	...	...	...	...	...	.
44853	d0306707-600d-45ea-a2c8-4aff6c7bf7b2	A(55MTR_45CNFpl)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	11.888392	312	6_3_1_LandAreasNotInOtherEconomicUs
44854	94ba6c60-3f23-4007-b5bb-43c48b641f55	LVIsc	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	20.698669	233	1_1_Agricultur
44855	b4a9dc2a-5536-4974-8b34-ad0347ac02e2	A(60MTR_40CNF)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	4.970088	312	6_3_1_LandAreasNotInOtherEconomicUs
	bf0d01a8-ee01-411a-		<?xml version="1.0"			

```
#CUBIERTAS DEL SUELO
codiige_dict = {
    111: "Casco",
    112: "Ensanche",
    113: "Discontinuo",
    114: "Zona verde urbana",
    121: "Instalación agrícola y/o ganadera",
    122: "Instalación forestal",
    123: "Extracción minera",
    130: "Industrial",
    140: "Servicio dotacional",
    150: "Asentamiento agrícola y huerta",
    161: "Red viaria o ferroviaria",
    162: "Puerto",
    163: "Aeropuerto",
```

```
    171: "Infraestructura de suministro",
    172: "Infraestructura de residuos",
    210: "Cultivo herbáceo",
    220: "Invernadero",
    231: "Frutal cítrico",
    232: "Frutal no cítrico",
    233: "Viñedo",
    234: "Olivar",
    235: "Otros cultivos leñosos",
    236: "Combinación de cultivos leñosos",
    240: "Prado",
    250: "Combinación de cultivos",
    260: "Combinación de cultivos con vegetación",
    311: "Bosque de frondosas",
    312: "Bosque de coníferas",
    313: "Bosque mixto",
    320: "Pastizal o herbazal",
    330: "Matorral",
    340: "Combinación de vegetación",
    351: "Playa, duna o arenal",
    352: "Roquedo",
    353: "Temporalmente desarbolado por incendios",
    354: "Suelo desnudo",
    411: "Zona húmeda y pantanosa",
    412: "Turbera",
    413: "Marisma",
    414: "Salina",
    511: "Curso de agua",
    512: "Lago o laguna",
    513: "Embalse",
    514: "Lámina de agua artificial",
    515: "Mar",
    516: "Glaciar y/o nieve perpetua"
}
```

```
suelo["CODIIGE"] = suelo["CODIIGE"].replace(codiige_dict)
suelo
```

	ID_POLYGON	SIOSE_CODE	SIOSE_XML	SUPERF_HA	CODIIGE	HI
0	98678c0f-70e2-49f6-85d0-40b7cb9292d4	PMX(80ZEV_10SNE_05EDFnv_05VAP)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	41.602492	Extracción minera	1_3_MiningAndQua
1	ae8c5627-affd-4a5d-8022-3b5e5ff31bbb	MTR	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	2.836093	Matorral	6_3_1_LandAreasNotInOtherEconom
2	cae746a8-2196-4f6e-9e4f-65d3ce3efec0	A(75MTR_25FDP)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	22.202730	Combinación de vegetación	6_3_1_LandAreasNotInOtherEconom
3	febe7109-4d88-4b87-b3b3-043bb22ea29f	UDS(65ZAU_15EDFva_10VAP_05LAA_05SNE)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	1.938984	Discontinuo	5_Residenti
4	443f3abb-daa3-4b60-9ec0-a50496f88ab1	DHS(60PST_40FDP)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	41.033844	Bosque de frondosas	6_3_1_LandAreasNotInOtherEconom
...	...	...	...	...	...	
44853	d0306707-600d-45ea-a2c8-4aff6c7bf7b2	A(55MTR_45CNFpl)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	11.888392	Bosque de coníferas	6_3_1_LandAreasNotInOtherEconom
44854	94ba6c60-3f23-4007-b5bb-43c48b641f55	LVIsc	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	20.698669	Viñedo	1_1_Agric
44855	b4a9dc2a-5536-4974-8b34-ad0347ac02e2	A(60MTR_40CNF)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	4.970088	Bosque de coníferas	6_3_1_LandAreasNotInOtherEconom
	bf0d01a8-ee01-411a-		<?xml version="1.0"		Bosque de	

```
COBERTURAS = {
    "EDF": "Edificación",
    "ZAU": "Zona verde artificial y arbolado urbano",
    "VAP": "Vial, aparcamiento o zona peatonal sin vegetación",
    "OCT": "Otras construcciones",
    "SNE": "Suelo no edificado",
    "ZEV": "Zonas de extracción o vertido",
    "CHL": "Cultivos herbáceos distintos de arroz",
    "CHA": "Arroz",
    "LFC": "Frutales cítricos",
    "LFN": "Frutales no cítricos",
    "LVI": "Viñedo",
    "LOL": "Olivo",
    "LAL": "Otros cultivos leñosos",
    "PRD": "Prado",
    "PST": "Pastizal",
    "FDC": "Frondosas caducifolias",
    "FDP": "Frondosas perennifolias",
    "CNF": "Coníferas",
    "MTR": "Matorral",
    "SDN": "Playas, dunas y arenales",
    "RMB": "Roquedos",
    "ACM": "Acantilado marino",
    "ARR": "Afloramientos rocosos",
    "ACH": "Canchales",
    "CLC": "Coladas lávicas",
    "HTU": "Zonas pantanosas",
    "HPA": "Humedales interiores",
    "HSA": "Humedales marinos",
    "HMA": "Marismas",
    "HSM": "Salinas",
    "AUC": "Aguas continentales",
```

```
    "LAA": "Láminas de agua artificial",
    "ALG": "Lagos y lagunas",
    "AEM": "Embalses",
    "ALC": "Aguas litorales",
    "AMO": "Mares y océanos",
    "GMP": "Glaciares"
}
```

```
CODIGOS_COBERTURAS = {
    "EDF": 101,
    "ZAU": 102,
    "LAA": 103,
    "VAP": 104,
    "OCT": 111,
    "SNE": 121,
    "ZEV": 131,
    "CHA": 211,
    "CHL": 212,
    "LFC": 222,
    "LFN": 223,
    "LVI": 231,
    "LOL": 232,
    "LAL": 241,
    "PRD": 290,
    "PST": 300,
    "FDC": 312,
    "FDP": 313,
    "CNF": 316,
    "MTR": 320,
    "PDA": 331,
    "SDN": 333,
    "ZQM": 334,
    "GMP": 335,
    "RMB": 336,
    "ACM": 351,
    "ARR": 352,
    "ACH": 353,
    "CLC": 354,
    "HPA": 411,
    "HTU": 412,
    "HSA": 413,
    "HMA": 421,
    "HSM": 422,
    "AUC": 511,
    "ALG": 513,
    "AEM": 514,
    "ALC": 521,
    "AMO": 523
}
```

```
COBERTURAS_COMPUESTAS = {
    "DHS": "Dehesas",
    "OVD": "Olivar - Viñedo",
    "AAR": "Asentamiento Agrícola Residencial",
    "UER": "Huerta Familiar",
    "UCS": "Urbano Mixto",
    "UEN": "Ensanche Urbano",
    "UDS": "Urbano Discontinuo",
    "IPO": "Polígono Industrial Ordenado",
    "IPS": "Polígono Industrial Sin Ordenar",
    "IAS": "Industria Aislada",
    "PAG": "Primario Agrícola / Ganadero",
    "PFT": "Primario Forestal",
    "PMX": "Primario Minero Extractivo",
    "PPS": "Piscifactoría",
    "TCO": "Comercial y Oficinas",
    "TCH": "Complejo Hotelero",
    "TPR": "Parque Recreativo",
    "TCG": "Camping",
    "EAI": "Administrativo Institucional",
    "ESN": "Sanitario",
    "ECM": "Cementerio",
    "EDU": "Educación",
    "EPN": "Penitenciario",
    "ERG": "Religioso",
    "ECL": "Cultural",
    "EDP": "Deportivo",
    "ECG": "Campo de Golf",
    "EPU": "Parque Urbano",
```

```
    "NRV": "Red Viaria",
    "NRF": "Red Ferroviaria",
    "NPO": "Puerto",
    "NAP": "Aeropuerto",
    "NEO": "Energía Eólica",
    "NSL": "Energía Solar",
    "NCL": "Energía Nuclear",
    "NEL": "Energía Eléctrica",
    "NTM": "Energía Térmica",
    "NHD": "Energía Hidroeléctrica",
    "NGO": "Gaseoducto / Oleoducto",
    "NTC": "Telecomunicaciones",
    "NDP": "Depuradoras y Potabilizadoras",
    "NCC": "Conducciones y Canales",
    "NDS": "Desalinizadoras",
    "NVE": "Vertederos y Escombreras",
    "NPT": "Plantas de Tratamiento"
}
```

```
CODIGOS_COBERTURAS_COMPUESTAS = {
    "DHS": 701,
    "OVD": 702,
    "AAR": 703,
    "UER": 704,
    "UCS": 810,
    "UEN": 812,
    "UDS": 813,
    "IPO": 821,
    "IPS": 822,
    "IAS": 823,
    "PAG": 831,
    "PFT": 832,
    "PMX": 833,
    "PPS": 834,
    "TCO": 841,
    "TCH": 842,
    "TPR": 843,
    "TCG": 844,
    "EAI": 851,
    "ESN": 852,
    "ECM": 853,
    "EDU": 854,
    "EPN": 855,
    "ERG": 856,
    "ECL": 857,
    "EDP": 858,
    "ECG": 859,
    "EPU": 860,
    "NRV": 881,
    "NRF": 882,
    "NPO": 883,
    "NAP": 884,
    "NEO": 891,
    "NSL": 892,
    "NCL": 893,
    "NEL": 894,
    "NTM": 895,
    "NHD": 896,
    "NGO": 897,
    "NTC": 900,
    "NDP": 911,
    "NCC": 912,
    "NDS": 913,
    "NVE": 921,
    "NPT": 922
}
```

```
ATRIBUTOS = {
    "ea": "edificio aislado",
    "em": "edificio entre medianeras",
    "va": "vivienda unifamiliar aislada",
    "vd": "vivienda unifamiliar adosada",
    "nv": "nave",
    "ec": "en construcción",
    "sc": "secano",
    "rr": "regadío regado",
    "rn": "regadío no regado",
    "ab": "abancalado",
    "fz": "forzado",
```

```
    "pl": "plantación",
    "fr": "formación de ribera",
    "fc": "función cortafuegos",
    "ct": "cortas",
    "pc": "procedencia de cultivos",
    "am": "alta montaña",
    "ra": "roturados no agrícolas",
    "ze": "zonas erosionadas",
    "cu": "cuaternarias"
}
```

```
import re

def decodificar_etiqueta(etiqueta):
    # Separar porcentaje (si existe) + cobertura + atributos
    match = re.match(r"(\d{2})?([A-Z]{3})([a-z]*)", etiqueta)
    if not match:
        return etiqueta  # Retornar sin cambios si no es reconocible

    porcentaje, cobertura, attrs = match.groups()
    descripcion = COBERTURAS.get(cobertura, cobertura)
    atributos = [ATRIBUTOS.get(a, a) for a in re.findall(r'[a-z]{2}', attrs)]

    texto = f"{porcentaje + '% ' if porcentaje else ''}{descripcion}"
    if atributos:
        texto += f" con {' y '.join(atributos)}"
    return texto

def decodificar_siose_code(code):
    # Coberturas compuestas (A, M, I, o predefinidas tipo UER, DHS, etc.)
    if re.match(r"^[A-Z]{1,3}\(.*\)$", code):
        tipo = code[:code.index("(")]
        contenido = code[code.index("(")+1:-1]
        partes = contenido.split("_")
        descripcion = [decodificar_etiqueta(p) for p in partes]
        return f"{tipo}: " + ", ".join(descripcion)
    else:
        return decodificar_etiqueta(code)
```

```
print(decodificar_siose_code("FDC"))
# Frondosas caducifolias

print(decodificar_siose_code("50CNF"))
# 50% Coníferas

print(decodificar_siose_code("CHLscfz"))
# Cultivos herbáceos distintos de arroz con secano y forzado

print(decodificar_siose_code("A(45MTR_35CNF_20PST)"))
# A: 45% Matorral, 35% Coníferas, 20% Pastizal

print(decodificar_siose_code("UER(30LFCfzsc_25EDFva_45PST)"))
# UER: 30% Frutales cítricos con forzado y secano, 25% Edificación con vivienda unifamiliar aislada, 45% Pastizal
```

```
Frondosas caducifolias
50% Coníferas
Cultivos herbáceos distintos de arroz con secano y forzado
A: 45% Matorral, 35% Coníferas, 20% Pastizal
UER: 30% Frutales cítricos con forzado y secano, 25% Edificación con vivienda unifamiliar aislada, 45% Pastizal
```

```
suelo["SIOSE_DESC"] = suelo["SIOSE_CODE"].apply(decodificar_siose_code)
suelo
```

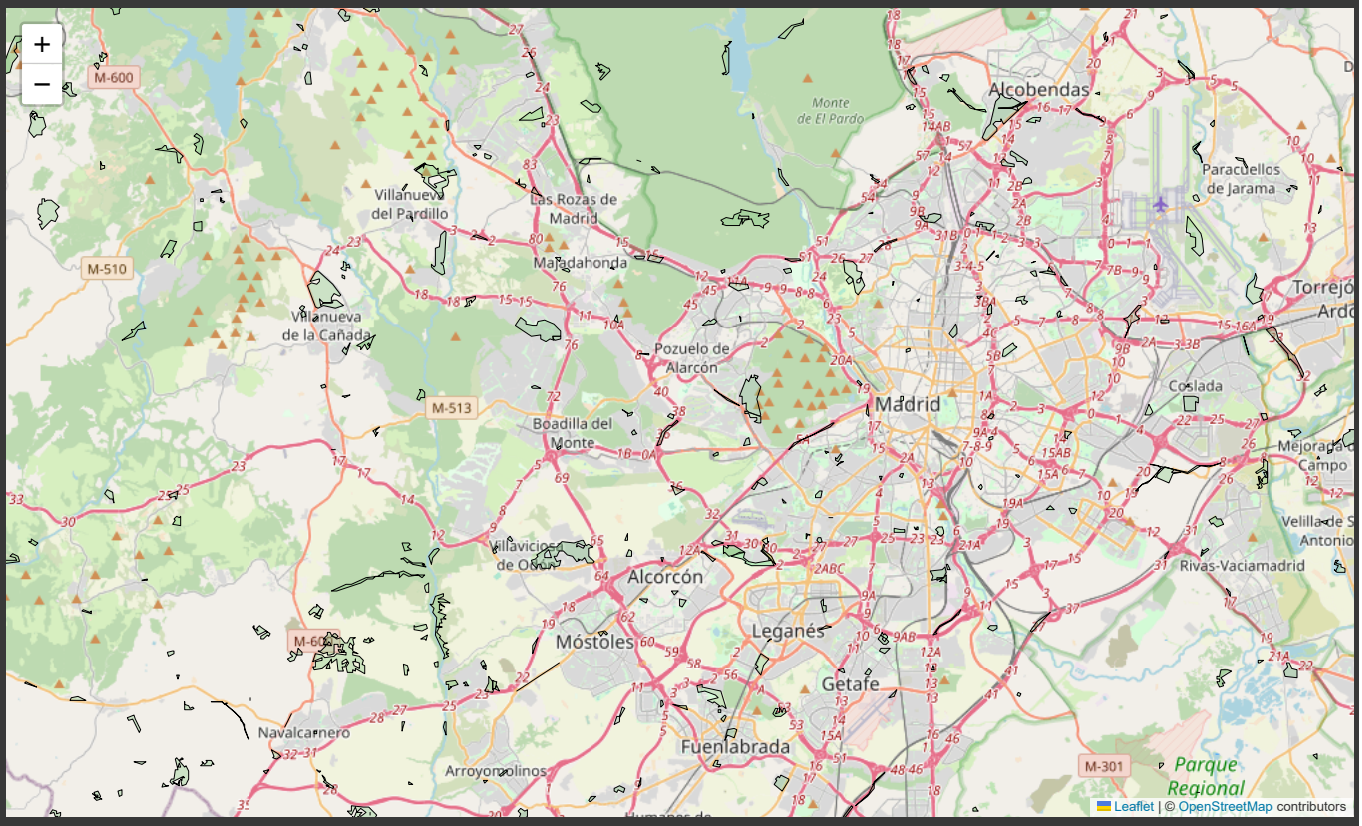
	ID_POLYGON	SIOSE_CODE	SIOSE_XML	SUPERF_HA	CODIIGE	
0	98678c0f-70e2-49f6-85d0-40b7cb9292d4	PMX(80ZEV_10SNE_05EDFnv_05VAP)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	41.602492	Extracción minera	1_3_MiningAndC
1	ae8c5627-affd-4a5d-8022-3b5e5ff31bbb	MTR	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	2.836093	Matorral	6_3_1_LandAreasNotInOtherEcon
2	cae746a8-2196-4f6e-9e4f-65d3ce3efec0	A(75MTR_25FDP)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	22.202730	Combinación de vegetación	6_3_1_LandAreasNotInOtherEcon
3	febe7109-4d88-4b87-b3b3-043bb22ea29f	UDS(65ZAU_15EDFva_10VAP_05LAA_05SNE)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	1.938984	Discontinuo	5_Reside
4	443f3abb-daa3-4b60-9ec0-a50496f88ab1	DHS(60PST_40FDP)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	41.033844	Bosque de frondosas	6_3_1_LandAreasNotInOtherEcon
...	...	...	...	...	...	
44853	d0306707-600d-45ea-a2c8-4aff6c7bf7b2	A(55MTR_45CNFpl)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	11.888392	Bosque de coníferas	6_3_1_LandAreasNotInOtherEcon
44854	94ba6c60-3f23-4007-b5bb-43c48b641f55	LVIsc	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	20.698669	Viñedo	1_1_Ag
44855	b4a9dc2a-5536-4974-8b34-ad0347ac02e2	A(60MTR_40CNF)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	4.970088	Bosque de coníferas	6_3_1_LandAreasNotInOtherEcon
44856	bf0d01a8-ee01-411a-bdc0-b76cee6149e5	A(50FDP_50MTR)	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	3.894969	Bosque de frondosas	6_3_1_LandAreasNotInOtherEcon
44857	98dc704f-23e9-4cb7-9691-101c1b8b56af	CHLsc	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	24.641840	Cultivo herbáceo	1_1_Ag

44858 rows × 11 columns

Mostrar solo los primeros N polígonos (ej. 1000)

```
import folium
# Mostrar máximo 500 puntos para no sobrecargar (puedes ajustar)

# Convertir CRS a WGS84 para usar en folium
suelo_wgs84 = suelo.head(1000).to_crs(epsg=4326)

# Crear mapa base
m = folium.Map(location=[40.4168, -3.7038], zoom_start=11)

# Agregar polígonos al mapa
folium.GeoJson(
    suelo_wgs84,
    name='Coberturas del Suelo',
    tooltip=folium.GeoJsonTooltip(fields=['SIOSE_CODE', 'CODIIGE', 'HILUCS','SIOSE_DESC']),
    style_function=lambda x: {
        'fillColor': '#74c476',
        'color': 'black',
        'weight': 0.5,
```

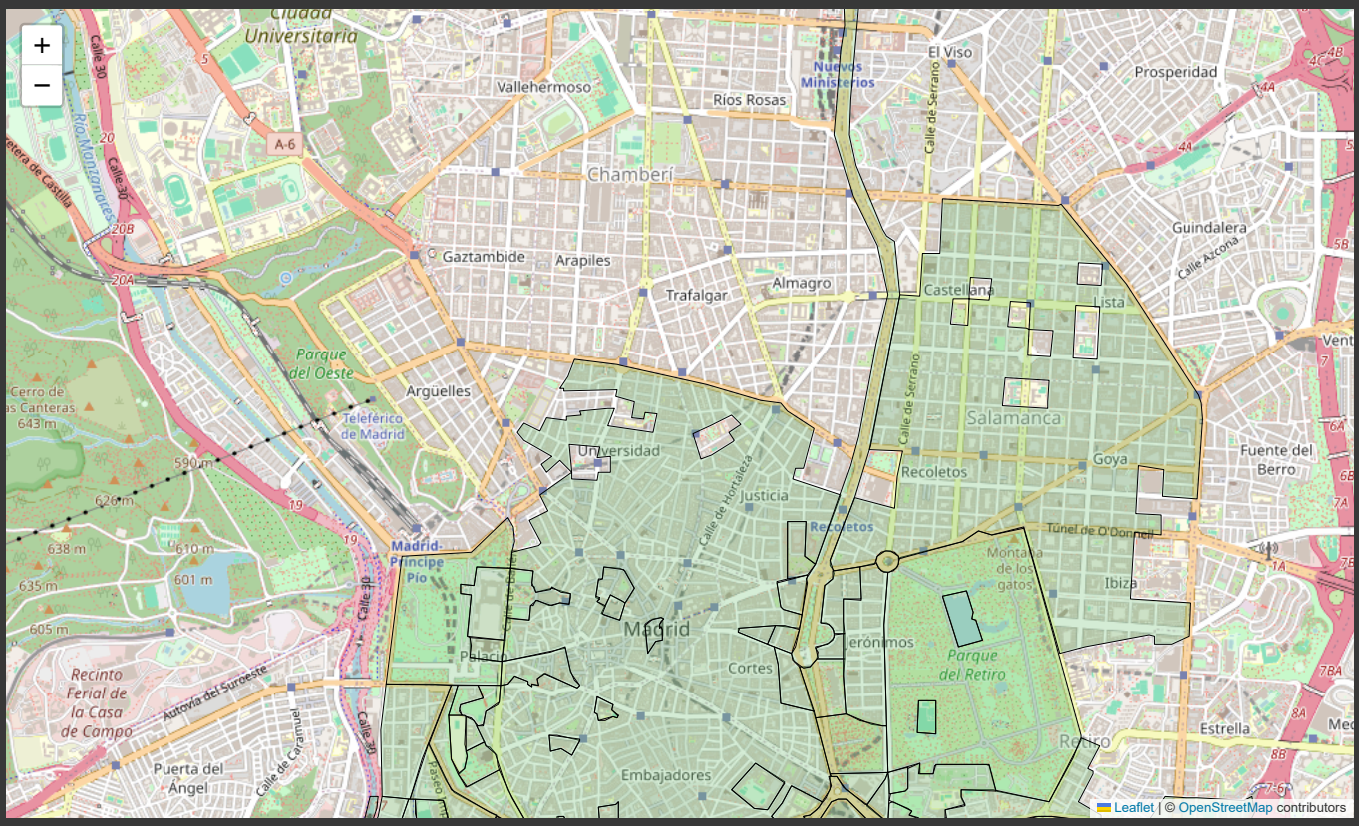
```
        'fillOpacity': 0.3
    }
).add_to(m)

# Mostrar mapa
m
```



Filtrar por zona geográfica (bounding box)

```
import folium
# Mostrar máximo 500 puntos para no sobrecargar (puedes ajustar)
xmin, xmax = -3.72, -3.68
ymin, ymax = 40.40, 40.42

# Convertir CRS antes de filtrar por bounding box
suelo_wgs84 = suelo.to_crs(epsg=4326)
suelo_zona = suelo_wgs84.cx[xmin:xmax, ymin:ymax]


# Crear mapa base
m1 = folium.Map(location=[40.4168, -3.7038], zoom_start=11)

# Agregar polígonos al mapa
folium.GeoJson(
    suelo_zona,
    name='Coberturas del Suelo',
    tooltip=folium.GeoJsonTooltip(fields=['SIOSE_CODE', 'CODIIGE', 'HILUCS','SIOSE_DESC']),
    style_function=lambda x: {
        'fillColor': '#74c476',
        'color': 'black',
        'weight': 0.5,
        'fillOpacity': 0.3
    }
).add_to(m1)
```

```
).add_to(m1)

# Mostrar mapa
m1
```



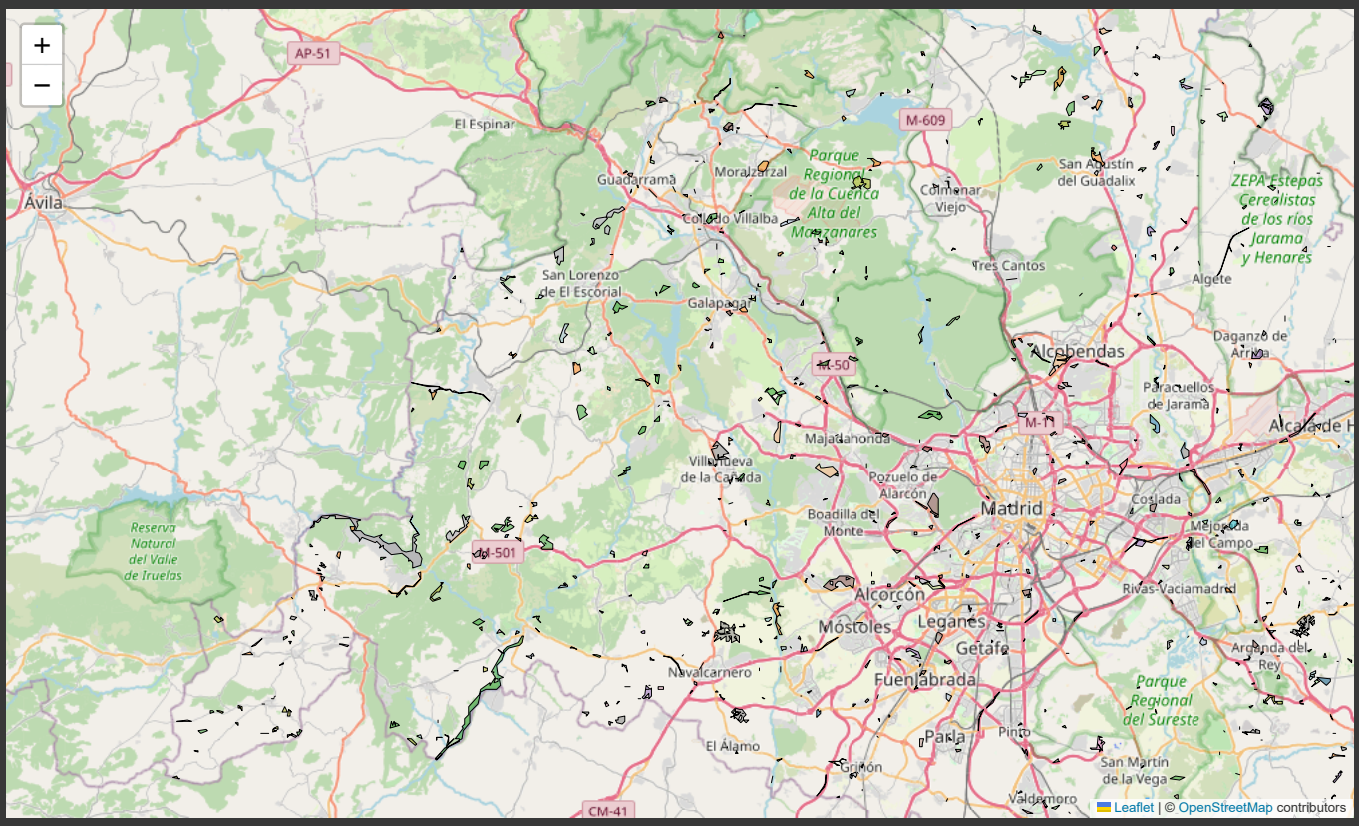
Asignamos colores

```
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors

# Crear un diccionario de colores únicos para cada código de cobertura
categorias = suelo['CODIIGE'].unique()
colores = plt.cm.tab20.colors  # Paleta de 20 colores
color_map = {cat: mcolors.to_hex(colores[i % len(colores)]) for i, cat in enumerate(categorias)}
```

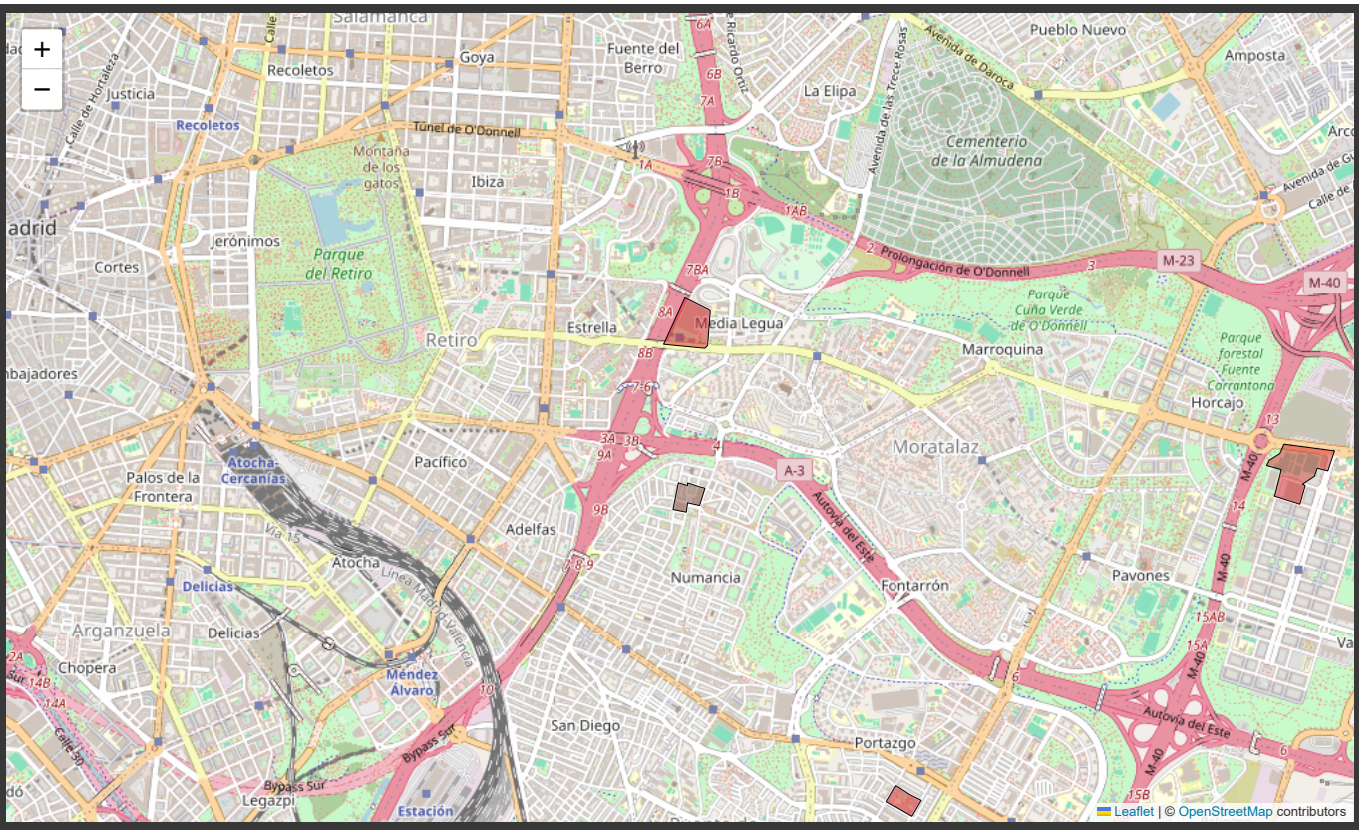
```
def estilo_por_cobertura(feature):
    codiige = feature['properties']['CODIIGE']
    color = color_map.get(codiige, '#cccccc')  # gris si no está
    return {
        'fillColor': color,
        'color': 'black',
        'weight': 0.3,
        'fillOpacity': 0.5
    }
```

```
suelo_wgs84 = suelo.head(1000).to_crs(epsg=4326)
m2 = folium.Map(location=[40.4168, -3.7038], zoom_start=13)

folium.GeoJson(
    suelo_wgs84,  # o suelo_grande si estás usando ese
    name='Coberturas del Suelo por CODIIGE',
```

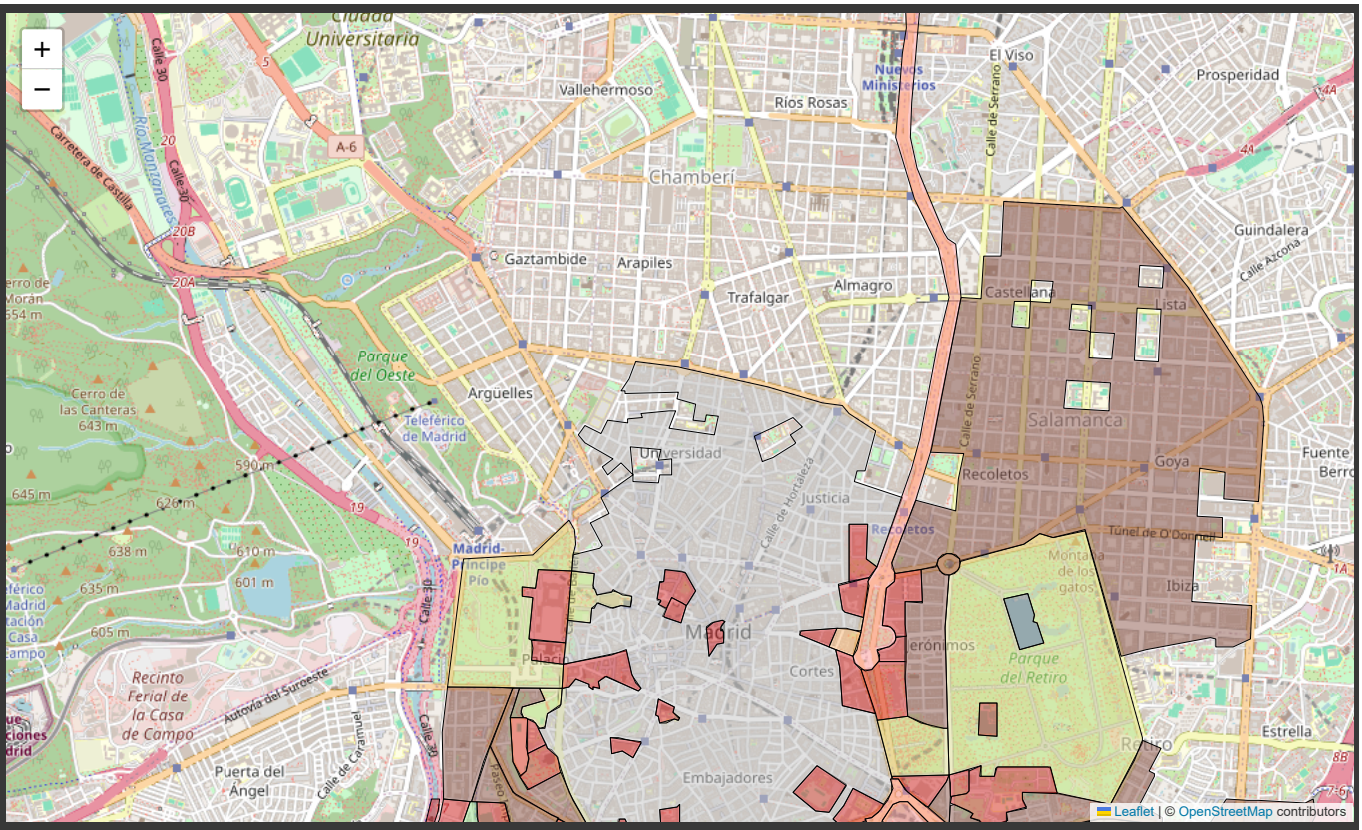
```
    style_function=estilo_por_cobertura,
    tooltip=folium.GeoJsonTooltip(fields=['SIOSE_CODE', 'CODIIGE', 'HILUCS'])
).add_to(m2)

m2
```




```
# Suponiendo que SUPERF_HA es el área en hectáreas
suelo_grande = suelo[suelo['SUPERF_HA'] > 5].head(100).to_crs(epsg=4326)

#suelo_wgs84 = suelo.head(1000).to_crs(epsg=4326)
mx = folium.Map(location=[40.4168, -3.7038], zoom_start=13)
folium.GeoJson(
    suelo_wgs84,  # o suelo_grande si estás usando ese
    name='Coberturas del Suelo por CODIIGE',
    style_function=estilo_por_cobertura,
    tooltip=folium.GeoJsonTooltip(fields=['SIOSE_CODE', 'CODIIGE', 'HILUCS'])
).add_to(mx)

mx
```

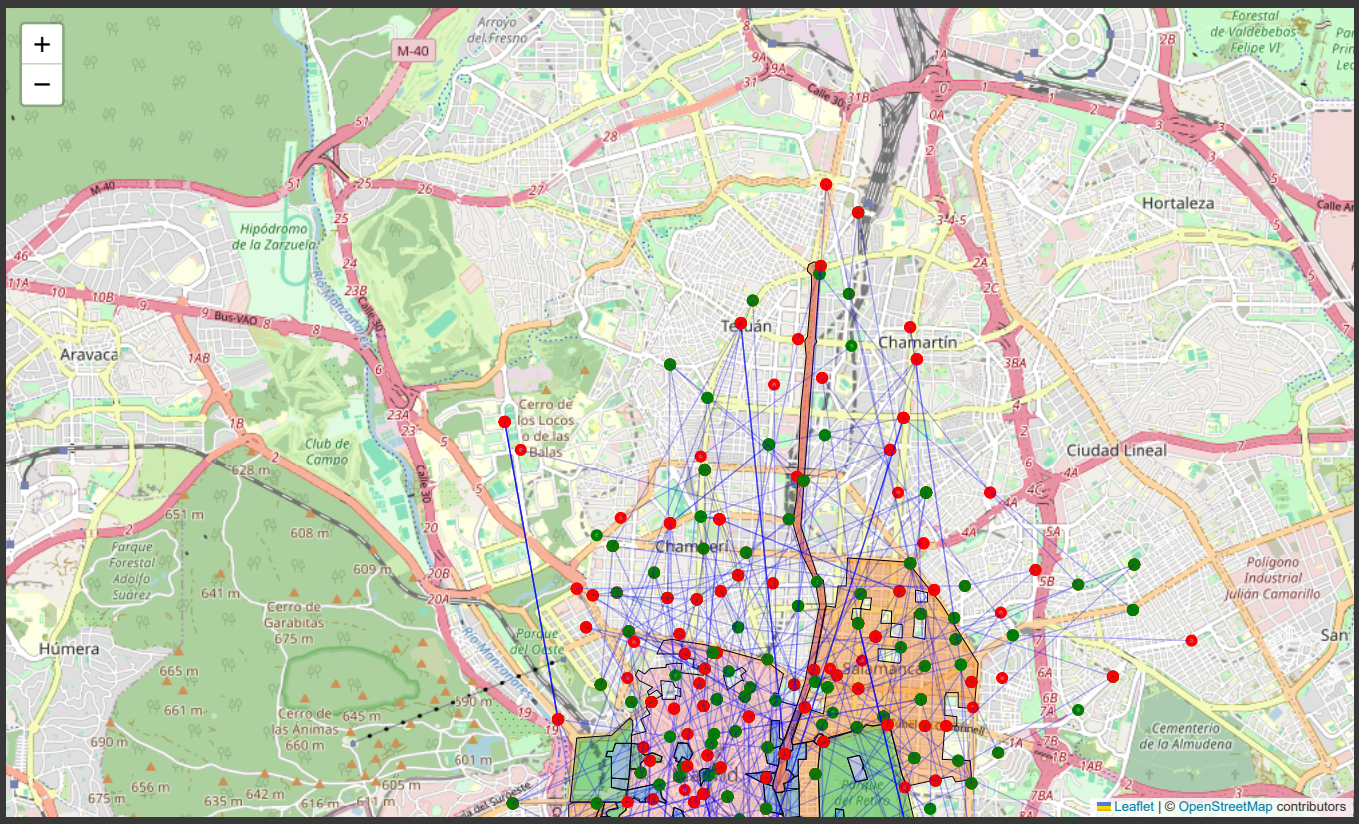



```
xmin, xmax = -3.72, -3.68
ymin, ymax = 40.40, 40.42

# Convertir CRS antes de filtrar por bounding box
suelo_wgs84 = suelo.to_crs(epsg=4326)
suelo_zona = suelo_wgs84.cx[xmin:xmax, ymin:ymax]

m3 = folium.Map(location=[40.4168, -3.7038], zoom_start=13)

folium.GeoJson(
    suelo_zona,  # o suelo_grande si estás usando ese
    name='Coberturas del Suelo por CODIIGE',
    style_function=estilo_por_cobertura,
    tooltip=folium.GeoJsonTooltip(fields=['SIOSE_CODE', 'CODIIGE', 'HILUCS'])
).add_to(m3)

m3
```




```
import folium
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors

# 1. Convertir CRS del suelo
suelo_wgs84 = suelo.to_crs(epsg=4326)

# 2. Filtrar por bounding box
xmin, xmax = -3.72, -3.68
ymin, ymax = 40.40, 40.42
suelo_zona = suelo_wgs84.cx[xmin:xmax, ymin:ymax]

# 3. Crear diccionario de colores por CODIIGE
categorias = suelo_zona['CODIIGE'].unique()
colores = plt.cm.tab20.colors
color_map = {cat: mcolors.to_hex(colores[i % len(colores)]) for i, cat in enumerate(categorias)}

# 4. Función de estilo por CODIIGE
def estilo_por_cobertura(feature):
    codiige = feature['properties']['CODIIGE']
    color = color_map.get(codiige, '#cccccc')
    return {
        'fillColor': color,
        'color': 'black',
        'weight': 0.3,
        'fillOpacity': 0.4
    }

# 5. Crear mapa base
m_comb = folium.Map(location=[40.4168, -3.7038], zoom_start=14)

# 6. Añadir polígonos del suelo
folium.GeoJson(
    suelo_zona,
    name='Coberturas del Suelo por CODIIGE',
    style_function=estilo_por_cobertura,
```

```
    tooltip=folium.GeoJsonTooltip(fields=['SIOSE_CODE', 'CODIIGE', 'HILUCS','SIOSE_DESC'])
).add_to(m_comb)

# 7. Añadir puntos y trayectos de viajes
subset = trips.dropna(subset=['lat_unlock', 'lon_unlock', 'lat_lock', 'lon_lock']).head(500)

for _, row in subset.iterrows():
    # Inicio (verde)
    folium.CircleMarker(
        location=[row['lat_unlock'], row['lon_unlock']],
        radius=3,
        color='green',
        fill=True,
        fill_opacity=0.7
    ).add_to(m_comb)

    # Fin (rojo)
    folium.CircleMarker(
        location=[row['lat_lock'], row['lon_lock']],
        radius=3,
        color='red',
        fill=True,
        fill_opacity=0.7
    ).add_to(m_comb)

    # Línea (azul)
    folium.PolyLine(
        locations=[[row['lat_unlock'], row['lon_unlock']], [row['lat_lock'], row['lon_lock']]],
        color='blue',
        weight=1,
        opacity=0.3
    ).add_to(m_comb)

# 8. Mostrar mapa combinado
m_comb
```



```
import pandas as pd

# Extraer coordenadas de inicio (unlock)
trips['lon_unlock'] = trips['geolocation_unlock'].apply(
    lambda x: eval(x)['coordinates'][0] if pd.notnull(x) else None
)
trips['lat_unlock'] = trips['geolocation_unlock'].apply(
    lambda x: eval(x)['coordinates'][1] if pd.notnull(x) else None
)

# Extraer coordenadas de fin (lock)
trips['lon_lock'] = trips['geolocation_lock'].apply(
    lambda x: eval(x)['coordinates'][0] if pd.notnull(x) else None
)
trips['lat_lock'] = trips['geolocation_lock'].apply(
    lambda x: eval(x)['coordinates'][1] if pd.notnull(x) else None
)
```

```
import geopandas as gpd
from shapely.geometry import Point

# Viajes válidos con ubicación de inicio
viajes_inicio = trips.dropna(subset=['lon_unlock', 'lat_unlock']).copy()
gdf_trips = gpd.GeoDataFrame(
    viajes_inicio,  # df con columnas lat/lon
    geometry=gpd.points_from_xy(viajes_inicio['lon_unlock'], viajes_inicio['lat_unlock']),
    crs='EPSG:4326'
)

viajes_inicio['geometry'] = [Point(xy) for xy in zip(viajes_inicio['lon_unlock'], viajes_inicio['lat_unlock'])]
gdf_inicio = gpd.GeoDataFrame(viajes_inicio, geometry='geometry', crs='EPSG:4326')
gdf_inicio = gdf_inicio.to_crs(suelo.crs)

# Viajes válidos con ubicación de fin
viajes_fin = trips.dropna(subset=['lon_lock', 'lat_lock']).copy()
gdf_trips = gpd.GeoDataFrame(
    viajes_fin,  # df con columnas lat/lon
    geometry=gpd.points_from_xy(viajes_fin['lon_unlock'], viajes_fin['lat_unlock']),
    crs='EPSG:4326'
)
viajes_fin['geometry'] = [Point(xy) for xy in zip(viajes_fin['lon_lock'], viajes_fin['lat_lock'])]
gdf_fin = gpd.GeoDataFrame(viajes_fin, geometry='geometry', crs='EPSG:4326')
gdf_fin = gdf_fin.to_crs(suelo.crs)
```

```
# JOIN para origen (inicio del viaje)
join_inicio = gpd.sjoin(gdf_inicio, suelo[['geometry', 'SIOSE_CODE', 'CODIIGE']], how='left', predicate='within')
join_inicio.rename(columns={'SIOSE_CODE': 'suelo_inicio', 'CODIIGE': 'codiige_inicio'}, inplace=True)

# JOIN para destino (fin del viaje)
join_fin = gpd.sjoin(gdf_fin, suelo[['geometry', 'SIOSE_CODE', 'CODIIGE']], how='left', predicate='within')
join_fin.rename(columns={'SIOSE_CODE': 'suelo_fin', 'CODIIGE': 'codiige_fin'}, inplace=True)
```

```
# Unimos por índice o por algún ID común (idBike + unlock_date si no hay ID único)
# Aquí se muestra una fusión simple basada en index (puedes adaptar según convenga)
viajes_completo = join_inicio[['unlock_date', 'trip_minutes', 'suelo_inicio', 'codiige_inicio']].copy()
viajes_completo['suelo_fin'] = join_fin['suelo_fin']
viajes_completo['codiige_fin'] = join_fin['codiige_fin']
```

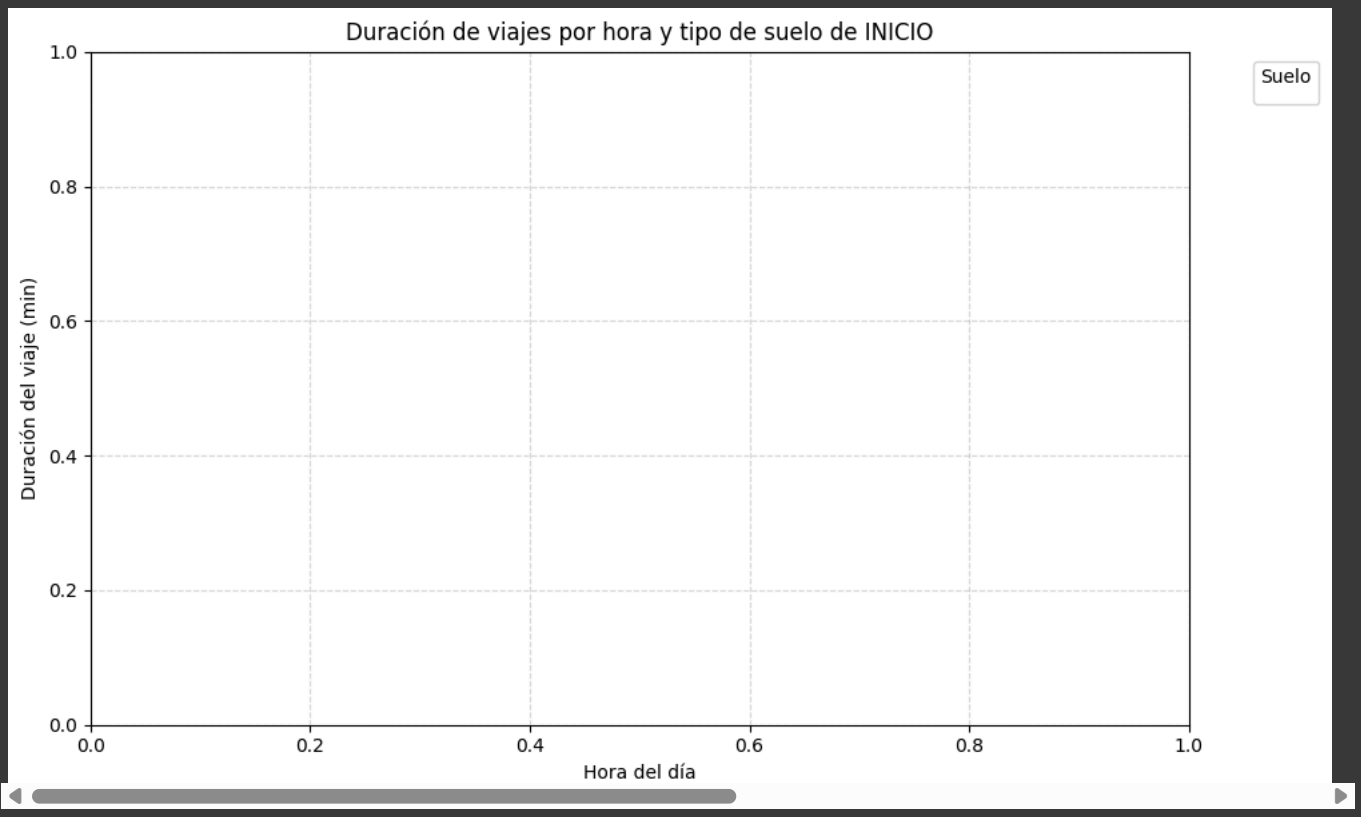
VISUALIZACION DE DATA

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Extraer la hora de inicio
viajes_completo['hora'] = pd.to_datetime(viajes_completo['unlock_date']).dt.hour
```

```
# Filtrar los 5 tipos de suelo más frecuentes en el inicio
top5_inicio = viajes_completo['codiige_inicio'].value_counts().head().index.tolist()
df_inicio = viajes_completo[viajes_completo['codiige_inicio'].isin(top5_inicio)].copy()
df_inicio['Suelo de inicio'] = df_inicio['codiige_inicio'].map(codiige_dict)
```

```
# Graficar
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=df_inicio,
    x='hora',
    y='trip_minutes',
    hue='Suelo de inicio',
    palette='Set1',
    alpha=0.6
)
plt.title("Duración de viajes por hora y tipo de suelo de INICIO")
plt.xlabel("Hora del día")
plt.ylabel("Duración del viaje (min)")
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend(title="Suelo", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

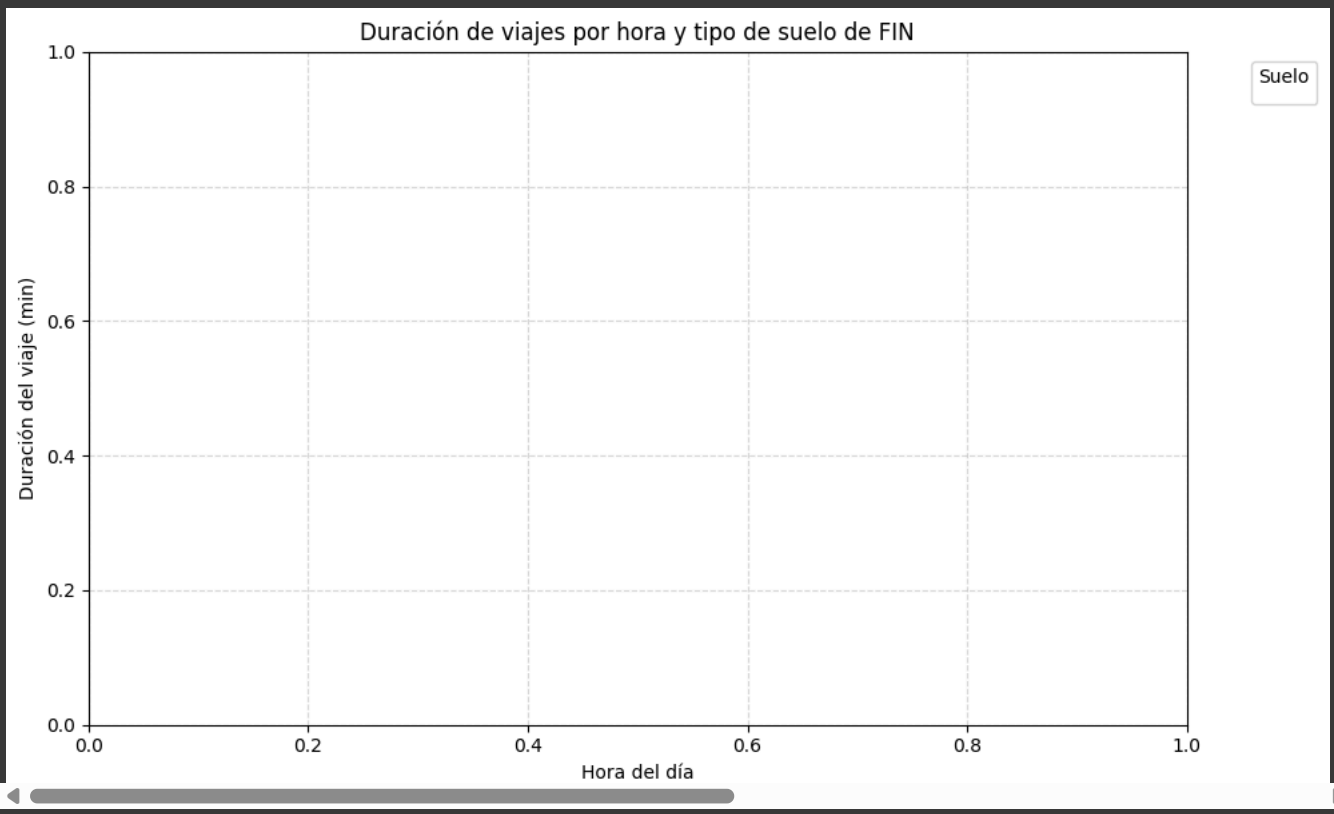
```
<ipython-input-111-743dc82c22f1>:8: UserWarning:

Ignoring `palette` because no `hue` variable has been assigned.

<ipython-input-111-743dc82c22f1>:20: UserWarning:

No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is
```




```
# Filtrar los 5 tipos de suelo más frecuentes en el fin
top5_fin = viajes_completo['codiige_fin'].value_counts().head(5).index.tolist()
df_fin = viajes_completo[viajes_completo['codiige_fin'].isin(top5_fin)].copy()
df_fin['Suelo de fin'] = df_fin['codiige_fin'].map(codiige_dict)

# Graficar
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=df_fin,
    x='hora',
    y='trip_minutes',
    hue='Suelo de fin',
    palette='Set2',
    alpha=0.6
)
plt.title("Duración de viajes por hora y tipo de suelo de FIN")
plt.xlabel("Hora del día")
plt.ylabel("Duración del viaje (min)")
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend(title="Suelo", bbox_to_anchor=(1.05, 1), loc='upper left')
```

```
plt.tight_layout()
plt.show()
```

```
<ipython-input-112-d3cc28276cdb>:8: UserWarning:

Ignoring `palette` because no `hue` variable has been assigned.

<ipython-input-112-d3cc28276cdb>:20: UserWarning:

No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is
```




```
# Extraer solo las filas que tienen coordenadas de inicio válidas
trips_validos = trips.dropna(subset=['lon_unlock', 'lat_unlock']).copy()
```

```
# Para puntos de fin
trips_validos_lock = trips.dropna(subset=['lon_lock', 'lat_lock']).copy()
```

```
# Convertir trips a GeoDataFrame con CRS igual al del suelo
gdf_trips = gpd.GeoDataFrame(
    trips_validos,  # df con columnas lat/lon
    geometry=gpd.points_from_xy(trips_validos['lon_unlock'], trips_validos['lat_unlock']),
    crs='EPSG:4326'
)

# Proyectar al CRS del suelo
gdf_trips_proj = gdf_trips.to_crs(suelo.crs)

# Ahora se puede hacer spatial join y verificaciones espaciales correctamente
resultado_join = gpd.sjoin(gdf_trips_proj, suelo, predicate='within', how='left')
```

```
# Tomamos un resultado del join
ejemplo = resultado_join.iloc[0]

# Obtenemos el punto y el polígono exactos (mismo índice)
punto = ejemplo.geometry
id_poligono = ejemplo['index_right']
poligono = suelo.loc[id_poligono].geometry

# Verificación precisa
print(punto.within(poligono))  # Debería ser True
```

```
True
```

Eliminamos los registroincorrectos

P5. Resumen de los datos

```
trips_no_null.describe()
```

	idBike	fleet	trip_minutes	station_unlock	dock_unlock	station_lock	dock_lock
count	167946.000000	167946.000000	167946.00000	167946.000000	167946.000000	167946.000000	167946.000000
mean	6003.277280	1.001512	19.13966	128.208424	12.426024	128.621884	12.427870
std	2007.063799	0.038860	161.89681	75.515541	7.598833	75.495462	7.626967
min	1.000000	1.000000	-53.25000	1.000000	1.000000	1.000000	1.000000
25%	5052.000000	1.000000	4.07000	59.000000	5.000000	59.000000	5.000000
50%	6596.000000	1.000000	9.57000	128.000000	12.000000	128.000000	12.000000
75%	7551.000000	1.000000	15.32000	191.000000	19.000000	192.000000	19.000000
max	8090.000000	2.000000	15849.53000	270.000000	30.000000	270.000000	30.000000

```
suelo.describe(include='object')
```

	ID_POLYGON	SIOSE_CODE	SIOSE_XML	CODIIGE	HILUCS	SIOSE_DESC
count	44858	44858	44858	44858	44858	44858
unique	44858	18061	44858	38	16	17375
top	98dc704f-23e9-4cb7-9691-101c1b8b56af	CHLsc	<?xml version="1.0" encoding="UTF-8"?> <POLIGON...	Ensanche	6_3_1_LandAreasNotInOtherEconomicUse	Cultivos herbáceos distintos de arroz con secano

```
valor.describe()
```

	ID_COBERTURAS	INTER_ID	SUPERF_HA	SUPERF_POR
count	152967.000000	152967.000000	152967.000000	152967.000000
mean	319.478548	2.718279	8.979810	52.586532
std	244.267945	1.726595	43.817243	37.017606
min	101.000000	1.000000	0.000600	0.100000
25%	104.000000	1.000000	0.737500	16.000000
50%	300.000000	2.000000	2.196000	40.000000
75%	352.000000	4.000000	6.061000	100.000000
max	922.000000	21.000000	3622.417700	100.000000

P6. ¿Qué representan top y Freq?

- La mayoria de tiempos de viaje largos solo tienen frecuencia de 1.
- La mayoría de las tierras usadas son Cultivos herbaceos distintos de arroz con secano con una frecuencia de 2374.
- Las hectareas tienen un comprotamiento similar.

```
trips_tmp_no_null_trips_minutes_count = trips_tmp['trip_minutes'].value_counts().sort_values(ascending=True).to_frame().head(20)
trips_tmp_no_null_trips_minutes_count
```

trip_minutes	count
40.63	1
49.07	1
1469.10	1
59.48	1
39.42	1
53.60	1
51.13	1
1480.67	1
1494.70	1
-31.90	1
1461.05	1
1472.12	1
-53.25	1
1448.70	1
43.93	1
-32.90	1
51.50	1
54.63	1
1486.97	1
1442.00	1

```
trips_no_null_trips_minutes_count = trips_no_null['trip_minutes'].value_counts().sort_values(ascending=False).to_frame().head(20)
trips_no_null_trips_minutes_count
```

trip_minutes	count
0.23	2924
0.25	2549
0.27	2434
0.15	2205
0.17	2167
0.18	2059
0.13	1858
0.28	1787
0.20	1721
0.22	1643
0.30	1274
0.12	1036
0.32	1028
0.33	809
0.35	585
0.37	583
0.08	537
0.10	483
0.38	464
0.40	426

```
suelo_SIOSE_DESC_count = suelo['SIOSE_DESC'].value_counts().sort_values(ascending=False).to_frame().head(20)
suelo_SIOSE_DESC_count
```

SIOSE_DESC	count
Cultivos herbáceos distintos de arroz con secano	2374
Olivo con secano	1715
Matorral	1519
Pastizal	1232
Viñedo con secano	828
NRV: Vial, aparcamiento o zona peatonal sin vegetación	505
UEN con en construcción	497
Cultivos herbáceos distintos de arroz con regadío regado	365
Prado con secano	338
Pastizal con procedencia de cultivos	275
Frondosas caducifolias con formación de ribera	206
Suelo no edificado	203
Zonas de extracción o vertido	183
A: 85% Matorral, 15% Frondosas perennifolias	180
A: 80% Matorral, 20% Frondosas perennifolias	167
UEN: Suelo no edificado	164
Coníferas con plantación	140
Matorral con formación de ribera	139
Matorral con procedencia de cultivos	130
NRF: Otras construcciones	129

```
valor_SUPERF_HA_count = valor['SUPERF_HA'].value_counts().sort_values(ascending=False).to_frame().head(20)
valor_SUPERF_HA_count
```

SUPERF_HA	count
0.1520	20
0.2450	20
0.1470	19
0.1675	18
0.4158	18
0.1513	18
0.0520	18
0.1589	17
0.5040	17
0.1023	17
0.0579	17
0.4043	17
0.1003	16
0.4213	16
0.2121	16
0.1594	16
0.1303	16
0.4156	16
0.1095	16
0.2619	16

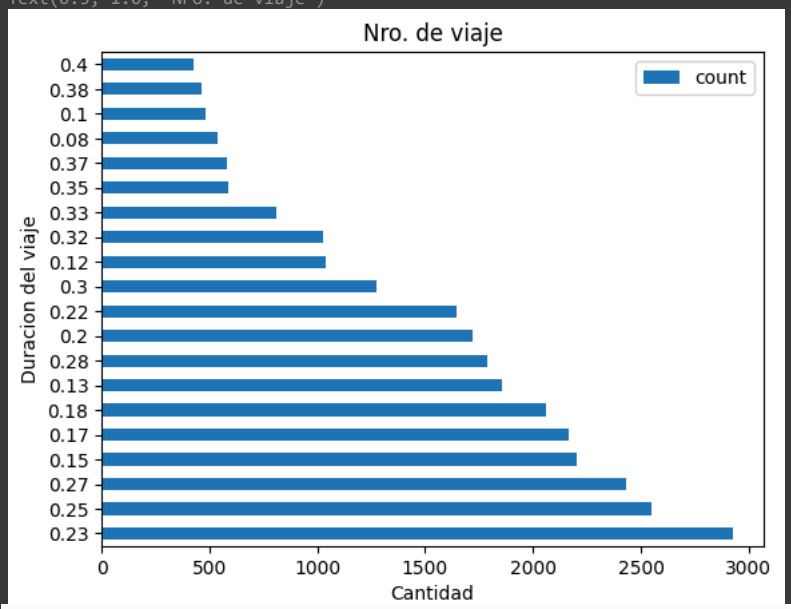
P7. Visualizaciones para dar sentido a los datos.

- Elemento de lista

- Elemento de lista

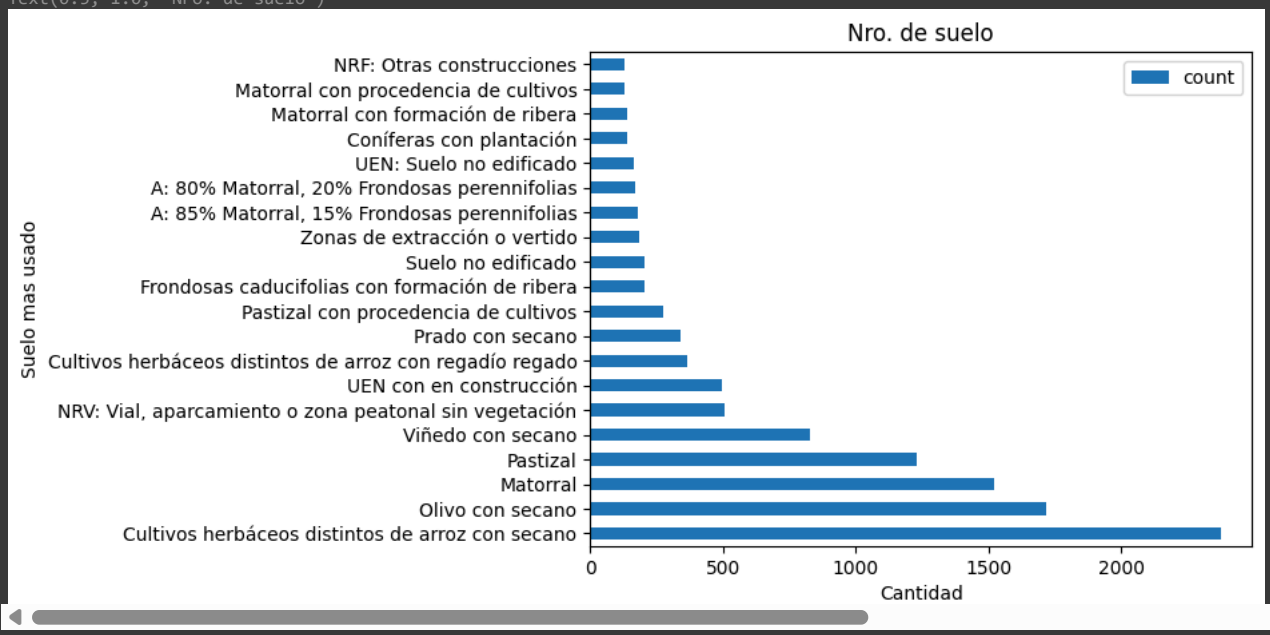
```
trips_no_null_trips_minutes_count.plot(kind = 'barh')
plt.xlabel('Cantidad') # definir el nombre del eje X
plt.ylabel('Duracion del viaje') # definir el nombre del eje Y
plt.title('Nro. de viaje')
```

```
Text(0.5, 1.0, 'Nro. de viaje')
```



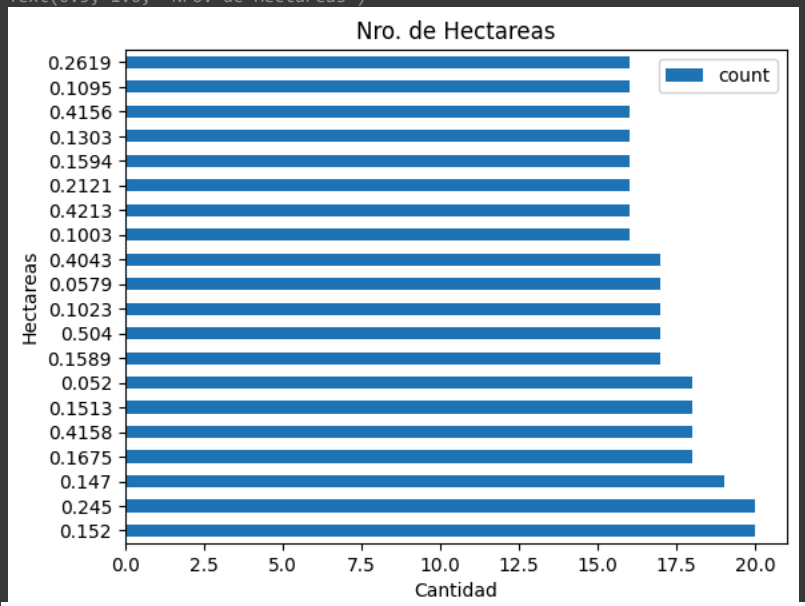

```
suelo_SIOSE_DESC_count.plot(kind = 'barh')
plt.xlabel('Cantidad') # definir el nombre del eje X
plt.ylabel('Suelo mas usado') # definir el nombre del eje Y
plt.title('Nro. de suelo')
```

```
Text(0.5, 1.0, 'Nro. de suelo')
```




```
valor_SUPERF_HA_count.plot(kind = 'barh')
plt.xlabel('Cantidad') # definir el nombre del eje X
plt.ylabel('Hectareas') # definir el nombre del eje Y
plt.title('Nro. de Hectareas')
```

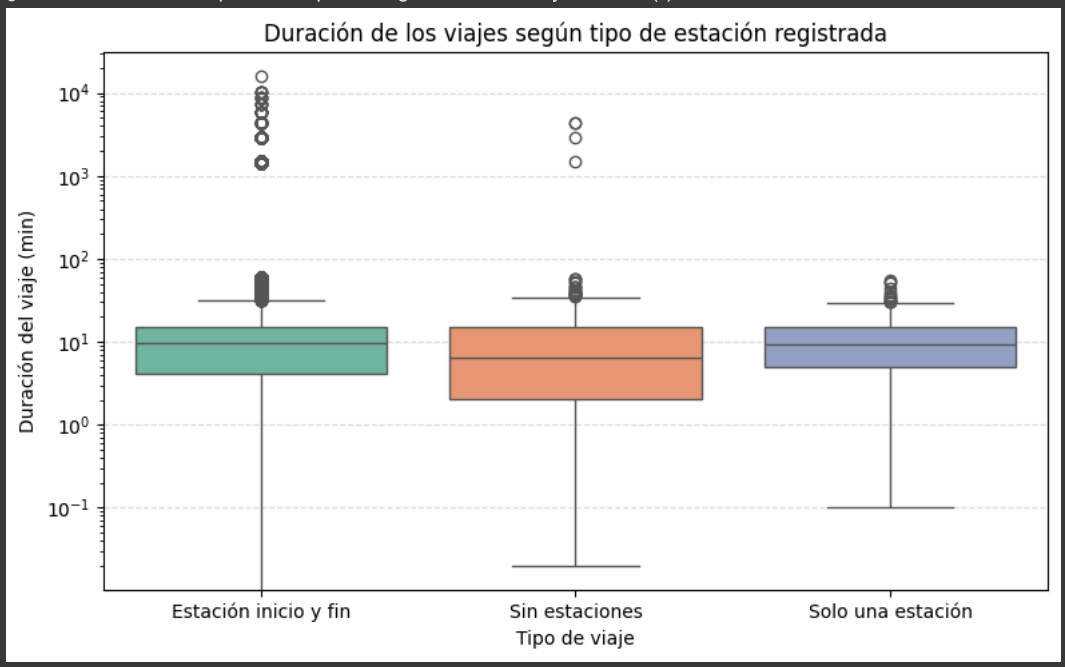
```
Text(0.5, 1.0, 'Nro. de Hectareas')
```



4. Entendiendo el mundo

P8. Realizar preguntas de acuerdo al problema, objetivos y/o contexto actual.

¿Que cantidad de tiempo es los que no registran sus idas y salidas? (*).



Oscilan entre 10^1.5 y 10^4

La amyoria de viajes tienen su registro completo

```
trips_no_null['tipo_viaje'].value_counts().sort_values(ascending=False).to_frame().head(20)
```

```
                    count
tipo_viaje
Estación inicio y fin  167946
```

```
trips_tmp['tipo_viaje'].value_counts().sort_values(ascending=False).to_frame().head(20)
```

```
                count
tipo_viaje
Estación inicio y fin   167946
Sin estaciones          308
Solo una estación       240
```

PIPELINE CIENCIA DE DATOS

1. ¿QUÉ PROBLEMAS IDENTIFICAMOS EN EL DATASET?

Las coordenada debemos dar un formato que se peuda usar en python.

{'type': 'point', 'coordinates': [-3.7022591, 40.4056107]} 'type': 'Point', 'coordinates': [-3.6956178, 40.4132798]] LONGITUD,LATITUD / Formato estándar GeoJSON / GeoPandas / shapely.

```
import folium
from folium.plugins import HeatMap
import ast

# 🔹 Paso 1: función para extraer lat/lon de geolocation_unlock
def extraer_coords(punto):
    try:
        geo = ast.literal_eval(punto) if isinstance(punto, str) else punto
        return pd.Series([geo["coordinates"][1], geo["coordinates"][0]])  # lat, lon
    except:
        return pd.Series([None, None])

# 🔹 Paso 2: aplicar la función y crear columnas lat/lon
trips[['lat_unlock', 'lon_unlock']] = trips['geolocation_unlock'].apply(extraer_coords)
```

```
# 🔹 Paso 3: crear dataset para HeatMap
heat_df = trips[['lat_unlock', 'lon_unlock', 'trip_minutes']].dropna()
heat_points = heat_df.values.tolist()

# 🔹 Paso 4: crear mapa centrado en Madrid
mapa = folium.Map(location=[40.4168, -3.7038], zoom_start=13)  # Coordenadas de Madrid

# 🔹 Paso 5: añadir HeatMap (pesado por duración del viaje)
HeatMap(heat_points, radius=10, blur=15, max_zoom=1).add_to(mapa)

# 🔹 Mostrar el mapa
mapa
#{'type': 'point', 'coordinates': [-3.7022591, 40.4056107]}
#'type': 'Point', 'coordinates': [-3.6956178, 40.4132798]] LONGITUD,LATITUD / Formato estándar GeoJSON / GeoPandas / shapely
```

```
# Para mapa de destino
trips[['lat_lock', 'lon_lock']] = trips['geolocation_lock'].apply(extraer_coords)
heat_df_lock = trips[['lat_lock', 'lon_lock', 'trip_minutes']].dropna()
heat_points_lock = heat_df_lock.values.tolist()

# Nuevo mapa de cierre de viaje
mapa_lock = folium.Map(location=[40.4168, -3.7038], zoom_start=13)
HeatMap(heat_points_lock, radius=10, blur=15, max_zoom=1).add_to(mapa_lock)
mapa_lock
```

```
# Crear mapa base centrado en Madrid
mapa_ambos = folium.Map(location=[40.4168, -3.7038], zoom_start=13)
```

```
# Mostrar máximo 500 puntos para no sobrecargar (puedes ajustar)
```