Trabajo Final de Master

Título: Integración de celdas h3 para la selección de áreas de factibilidad de proyectos de energía eólica mediante la aplicación de componentes principales y modelo de machine learning random forest.

Autor: Manuel E. Hinojosa

Fecha: 2 de Marzo de 2023

1. Carga de datos

La carga de los datos se hace mediante el uso de la libreria Pandas.

In [1]:
```
import pandas as pd
```

In [2]:
```
tfm_hex = pd.read_csv(r"D:\TFM_UCM\DATASET\TFM_HEX_MH.csv")
```

In [3]:
```
tfm_hex.head(5)
```

Out[3]:

	OID	Org_id	Max_WSP	Min_WSP	Mean_WSP	Calz_km	Calz_cnt	Calz_dsc	TTE_cnt	LTE_km	...	Inl
0	1	3	7.570377	3.145082	4.889567	0.0	0	NaN	0	0.0	...	
1	2	4	7.513499	3.203467	4.849657	0.0	0	NaN	0	0.0	...	
2	3	5	7.489709	3.150265	4.868490	0.0	0	NaN	0	0.0	...	
3	4	7	7.512815	3.142868	4.887665	0.0	0	NaN	0	0.0	...	
4	5	8	7.577927	3.231379	4.927116	0.0	0	NaN	0	0.0	...	

5 rows × 29 columns

1.1 Descripción de Variables

- Variable / Descripción*
- OID : Identificador automático del hexágono
- Org_id : Identificador adicional del hexágono
- Max_WSP : Velocidad máxima del viento *Min_WSP : Velocidad mínima del viento*Mean_WSP : Velocidad media del viento *Calz_km : Largo total de la calzada que se superpone con el hexágono*Calz_cnt : Cantidad total de tramos de calzada que se superponen con el hexágono

*NA_km2 : Superficie total de áreas naturales dentro del hexágono*NA_cnt : Áreas naturales individuales dentro del hexágono *Na_dsc : Descripción del área natural (bosques, praderas, arboles individuales, llanos, roqueríos)*TTE_cnt : Torre de alta tensión *LTE_km : Largo total de las líneas de transmisión eléctrica se superponen con el hexágono*LTE_cnt : Cantidad total de líneas de transmisión eléctrica que se superponen con el hexágono *UA_km2 : Superficie en kilómetros cuadrados de áreas urbanas o edificadas que están dentro del hexágono*UA_cnt : Cantidad de polígonos urbanos dentro del hexágono *Den_P : Densidad de población medidas en hab/km2 del municipio*Pend_m : Pendiente media que está dentro del hexágono *InEl_dsc : Descripción de las instalaciones eléctricas (Inst_E.OF: Instalación de otras fuentes; E.E: Energía Eléctrica; E.EO: Eólica;
E.HE: Hidroeléctrica; E.NU: Nuclear; E.SO: Solar; E.TM: Térmica; E.HC: Hidrocarburos)
*InEl_km2 : Área en km2 de las instalaciones eléctricas*Calz_dsc : Descripción del tipo de calzada *VxHx_left : Vértice izquierdo del hexágono*VxHx_top : Vértice superior del hexágono *VxHx_right: Vértice derecho del hexágono*VxHx_btm : Vértice inferior del hexágono *Cmrca : Nombre de la Comarca*Mun_nom : Nombre del Municipio *Con_kWh_hex: Consumo en kWh por hexágono*Con_kWh_Mun: Consumo total en kWh por el municipio *Area_Hex : Área en km2 del hexágono

2. Transformacion de datos

2.1 Columnas categóricas a columnas númericas

Para realizar la siguiente transformacion, se crea una lista llamada "col" que contiene todos los nombres de las columnas del dataframe **tfm_hex**. Una vez hecho esto, seleccionamos las columas categoricas que van a ser procesadas, que en este caso son: **'Calz_dsc'**, **'Na_dsc'**, **'InEl_dsc'**. Por otro lado, eliminaremos las columnas **'Cmrca'**, **'Mun_nom'**, que corresponden a la informacion politico administrativa que no genera aporte a la conversión.

Luego, generamos una iteración sobre dichas columnas, lo que convertira los valores de categóricos en columnas de variables binarias mediante la funcion *pd.get_dummies*. Finalmente agregamos cada una de estas nuevas columnas a nuestro dataframe mediante la funcion *pd.concat* y eliminamos la columna original de valores categoricos del dataframe **tfm_hex**.

Finalmente, este codigo busca mediante la transformacion de las columnas categóricas a columnas de variables binarias, obtener los datos de entrada para el modelo de *machine learning*

In [4]:
```
#Transformaciones
col = [x for x in tfm_hex.columns]
columns_to_process = ['Calz_dsc', 'Na_dsc', 'InEl_dsc']
tfm_hex.drop(['Cmrca', 'Mun_nom'], axis=1, inplace=True)
print(col)
for c in columns_to_process:
    print('Working with : ', c, tfm_hex[c].dtype)
    temp_name = f'tfm_hex_{c}'
    temp_name = pd.get_dummies(tfm_hex[c])
    tfm_hex = pd.concat([tfm_hex, temp_name], axis=1).reindex(tfm_hex.index)
    tfm_hex.drop(c, axis=1, inplace=True)
```

```
['OID', 'Org_id', 'Max_WSP', 'Min_WSP', 'Mean_WSP', 'Calz_km', 'Calz_cnt', 'Calz_dsc',
'TTE_cnt', 'LTE_km', 'LTE_cnt', 'Na_dsc', 'NA_km2', 'Na_cnt', 'UA_km2', 'UA_cnt', 'Den_
P', 'Pend_m', 'InEl_dsc', 'InEl_km2', 'VxHx_left', 'VxHx_top', 'VxHx_right', 'VxHx_btm',
'Cmrca', 'Mun_nom', 'Con_kWh_hex', 'Con_kWh_Mun', 'Area_Hex']
Working with :  Calz_dsc object
Working with :  Na_dsc object
Working with :  InEl_dsc object
```

In [5]:
```
tfm_hex['TTE_cnt'].unique()
```

Out[5]:
```
array([ 0,  2,  1,  3,  6,  4,  7,  5, 10,  8,  9, 12, 11, 17, 19, 20, 21,
       13, 23, 14, 15, 16, 59, 33, 26, 18], dtype=int64)
```

2.2 Analisis exploratorio inicial

Como parte del analisis exploratorio de datos, necesitamos conocer la estructura y el contenido del dataframe "tfm_hex", y verificar que los datos se esten cargando y procesando correctamente. Por lo tanto, con cada iteracion por medio del bucle "for" sobre cada elemento de la lista imprimiremos nombre de la columna y su tipo de datos que almacena.

In [6]:
```
col = [x for x in tfm_hex.columns]
print(col)
for c in col:
    print(c, ' : ', tfm_hex[c].dtype)
```

```
['OID', 'Org_id', 'Max_WSP', 'Min_WSP', 'Mean_WSP', 'Calz_km', 'Calz_cnt', 'TTE_cnt', 'L
TE_km', 'LTE_cnt', 'NA_km2', 'Na_cnt', 'UA_km2', 'UA_cnt', 'Den_P', 'Pend_m', 'InEl_km
2', 'VxHx_left', 'VxHx_top', 'VxHx_right', 'VxHx_btm', 'Con_kWh_hex', 'Con_kWh_Mun', 'Ar
ea_Hex', 'Autopista', 'Ferrocarril_Int_CONV', 'Ferrocarril_estr_vel_conv', 'Ferrocarril_
int_AVE', 'Ferrocarril_vel_conv', 'Via calzada unica', 'Via doble calzada', 'Via doble p
referente', 'Via unica preferente', 'Arboles aislados', 'Vertedero', 'arenas', 'bosque',
'humedal', 'pradera', 'roquerios', 'Inst_E.E', 'Inst_E.EO', 'Inst_E.HC', 'Inst_E.HE', 'I
nst_E.NU', 'Inst_E.OF', 'Inst_E.SO', 'Inst_E.TM']
OID  :  int64
Org_id  :  int64
Max_WSP  :  float64
Min_WSP  :  float64
Mean_WSP  :  float64
Calz_km  :  float64
Calz_cnt  :  int64
TTE_cnt  :  int64
LTE_km  :  float64
LTE_cnt  :  int64
NA_km2  :  float64
Na_cnt  :  int64
UA_km2  :  float64
UA_cnt  :  int64
Den_P  :  int64
Pend_m  :  float64
InEl_km2  :  float64
VxHx_left  :  float64
VxHx_top  :  float64
VxHx_right  :  float64
VxHx_btm  :  float64
Con_kWh_hex  :  float64
Con_kWh_Mun  :  float64
Area_Hex  :  float64
```

```
Autopista  :  uint8
Ferrocarril_Int_CONV  :  uint8
Ferrocarril_estr_vel_conv  :  uint8
Ferrocarril_int_AVE  :  uint8
Ferrocarril_vel_conv  :  uint8
Via calzada unica  :  uint8
Via doble calzada  :  uint8
Via doble preferente  :  uint8
Via unica preferente  :  uint8
Arboles aislados  :  uint8
Vertedero  :  uint8
arenas  :  uint8
bosque  :  uint8
humedal  :  uint8
pradera  :  uint8
roquerios  :  uint8
Inst_E.E  :  uint8
Inst_E.EO  :  uint8
Inst_E.HC  :  uint8
Inst_E.HE  :  uint8
Inst_E.NU  :  uint8
Inst_E.OF  :  uint8
Inst_E.SO  :  uint8
Inst_E.TM  :  uint8
```

Una vez obtenido la lista completa de las columnas con su tipo de datos, cargamos nuevamente el dataframe para visualizar el resultado

In [7]:
```
tfm_hex
```

Out[7]:

	OID	Org_id	Max_WSP	Min_WSP	Mean_WSP	Calz_km	Calz_cnt	TTE_cnt	LTE_km	LTE_cnt	.
0	1	3	7.570377	3.145082	4.889567	0.0	0	0	0.0	0	
1	2	4	7.513499	3.203467	4.849657	0.0	0	0	0.0	0	
2	3	5	7.489709	3.150265	4.868490	0.0	0	0	0.0	0	
3	4	7	7.512815	3.142868	4.887665	0.0	0	0	0.0	0	
4	5	8	7.577927	3.231379	4.927116	0.0	0	0	0.0	0	
...	...	...	...	...	...	...	...	...	...	...	...
37065	37066	37889	7.801652	4.573597	6.071446	0.0	0	0	0.0	0	
37066	37067	37892	7.720746	4.434027	5.970415	0.0	0	0	0.0	0	
37067	37068	37895	7.950208	4.594975	6.045334	0.0	0	0	0.0	0	
37068	37069	37896	8.017551	4.658286	6.054664	0.0	0	0	0.0	0	
37069	37070	37901	7.958786	4.750614	6.234726	0.0	0	0	0.0	0	

37070 rows × 48 columns

3. Normalización

3.1 Seleccion del modelo de normalizacion

La normalización del dataframe *tfm_hex* se hara mediante el uso de *MinMaxScaler* de la libreria *Scikit-learn*. La razón para el uso de *preprocessing.MinMaxScaler()* se debe a que el dataframe cuenta con muchos campos categóricos con valores [0, 1] y este es el modulo que mejor maneja este tipo de datos.

La lista [ ] "col" contiene los nombres de todas las columnas numéricas de "tfm_hex" (es decir, aquellas columnas cuyo tipo de dato es: "float64" o "int64"). También en el mismo código, especificamos algunas columnas que no deben normalizarse.

Creamos un objeto *scaler* donde definimos las caracteristicas del rango de salida, que en este caso debe ser entre (0,1). Hacemos una copia del dataframe en "tfm_hex_norm" y luego normalizamos las columnas que mediante el uso de *scaler.fit_transform*. Esto ajustará el rango de nuestras columnas por medio del objeto scaler que habiamos definido en la segunda linea, esto con el proposito de ajustar el rango de de las columnas en la lista "col" al rango establecido en scaler.

Por ultimo, guardamos los datos normalizados en *normalized_df* y asignamos el nombre de las columnas orginales del dataframe.

In [8]:
```
from sklearn import preprocessing
import numpy as np
```

In [9]:
```
col = [x for x in tfm_hex.columns if tfm_hex[x].dtype in ['float64', 'int64'] and x not
scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))

tfm_hex_norm = tfm_hex
dfn = scaler.fit_transform(tfm_hex_norm[col])
normalized_df = pd.DataFrame(dfn, columns = col)

normalized_df
```

Out[9]:

	Max_WSP	Min_WSP	Mean_WSP	Calz_km	Calz_cnt	TTE_cnt	LTE_km	LTE_cnt	NA_km2	Na_c
0	0.288057	0.289192	0.324866	0.0	0.0	0.0	0.0	0.0	0.661782	0.3333
1	0.278822	0.305923	0.313118	0.0	0.0	0.0	0.0	0.0	0.701793	0.5000
2	0.274959	0.290678	0.318662	0.0	0.0	0.0	0.0	0.0	0.828543	0.1666
3	0.278711	0.288558	0.324306	0.0	0.0	0.0	0.0	0.0	0.657581	0.3333
4	0.289283	0.313922	0.335918	0.0	0.0	0.0	0.0	0.0	0.894986	0.1666
...	...	...	...	...	...	...	...	...	...	
37065	0.325608	0.698552	0.672753	0.0	0.0	0.0	0.0	0.0	0.027000	0.1666
37066	0.312471	0.658556	0.643014	0.0	0.0	0.0	0.0	0.0	0.207841	0.5000
37067	0.349728	0.704678	0.665067	0.0	0.0	0.0	0.0	0.0	0.039438	0.3333
37068	0.360662	0.722821	0.667813	0.0	0.0	0.0	0.0	0.0	0.000051	0.1666
37069	0.351121	0.749278	0.720814	0.0	0.0	0.0	0.0	0.0	0.058336	0.3333

37070 rows × 17 columns

3.2 Eliminación y reemplazo de columnas no normalizadas

Ahora procederemos a eliminar del dataframe a a aquellas columnas que no fueron normalizadas y reemplazarlas con sus datos ya normalizados. Para ello, aplicamos el cambio directamente en el dataframe. Luego, con *pd.concat* concatenamos el dataframes *tfm_hex_norm* (que es el dataframe con las valores ya normalizados) sobre el dataframe final que es *normalized_df*. Conseguimos ordenar los indices del dataframe *tfm_hex_norm* a fin de que coincidan con con el dataframe original *tfm_hex*, esto se hace para garantizar que los datos normalizados en ambos dataframes se concatenen correctamente.

In [10]:
```
#Let's drop not normalized fields, then add normalized (scaled) version of fields
tfm_hex_norm.drop(col, axis=1, inplace=True)
tfm_hex_norm = pd.concat([tfm_hex_norm, normalized_df], axis=1).reindex(tfm_hex_norm.in

tfm_hex_norm.describe()
```

Out[10]:

	OID	Org_id	VxHx_left	VxHx_top	VxHx_right	VxHx_btm	Area
count	37070.000000	37070.000000	37070.000000	3.707000e+04	37070.000000	3.707000e+04	37070.00
mean	18535.500000	18990.618425	376835.886688	4.628837e+06	377990.587227	4.627837e+06	866025.40
std	10701.331576	10863.016857	64066.390814	5.352501e+04	64066.390814	5.352501e+04	0.01
min	1.000000	3.000000	261025.605800	4.489476e+06	262180.306300	4.488476e+06	866025.40
25%	9268.250000	9647.250000	324245.460300	4.590976e+06	325400.160800	4.589976e+06	866025.40
50%	18535.500000	18976.500000	366680.705100	4.631976e+06	367835.405600	4.630976e+06	866025.40
75%	27802.750000	28377.750000	425570.432500	4.669976e+06	426725.133100	4.668976e+06	866025.40
max	37070.000000	37901.000000	525163.354000	4.747976e+06	526318.054500	4.746976e+06	866025.45

8 rows × 48 columns

3.3 Eliminación y revisión de datos nulos

La siguiente línea de código busca determinar cuantos valores nulos *NaN* hay en cada columnas del dataframe, esto lo hace por medio de una interación en la lista *col_norm*. Cada iteración se imprime junto con su tipo de dato. Esta lista se utiliza para revisar cada columna e identificar donde se encuentran dichos valores nulos.

In [11]:
```
# Then we need to check data for NaNs and process it

col_norm = [x for x in tfm_hex_norm.columns]
for c in col_norm:
    print(f'{tfm_hex_norm[c].isnull().sum()} NANs in -> {c} column, column datatype : {
```

```
0 NANs in -> OID column, column datatype : int64
0 NANs in -> Org_id column, column datatype : int64
0 NANs in -> VxHx_left column, column datatype : float64
0 NANs in -> VxHx_top column, column datatype : float64
0 NANs in -> VxHx_right column, column datatype : float64
0 NANs in -> VxHx_btm column, column datatype : float64
0 NANs in -> Area_Hex column, column datatype : float64
0 NANs in -> Autopista column, column datatype : uint8
0 NANs in -> Ferrocarril_Int_CONV column, column datatype : uint8
0 NANs in -> Ferrocarril_estr_vel_conv column, column datatype : uint8
0 NANs in -> Ferrocarril_int_AVE column, column datatype : uint8
0 NANs in -> Ferrocarril_vel_conv column, column datatype : uint8
0 NANs in -> Via calzada unica column, column datatype : uint8
0 NANs in -> Via doble calzada column, column datatype : uint8
0 NANs in -> Via doble preferente column, column datatype : uint8
0 NANs in -> Via unica preferente column, column datatype : uint8
0 NANs in -> Arboles aislados column, column datatype : uint8
0 NANs in -> Vertedero column, column datatype : uint8
0 NANs in -> arenas column, column datatype : uint8
0 NANs in -> bosque column, column datatype : uint8
0 NANs in -> humedal column, column datatype : uint8
0 NANs in -> pradera column, column datatype : uint8
0 NANs in -> roquerios column, column datatype : uint8
0 NANs in -> Inst_E.E column, column datatype : uint8
0 NANs in -> Inst_E.EO column, column datatype : uint8
0 NANs in -> Inst_E.HC column, column datatype : uint8
0 NANs in -> Inst_E.HE column, column datatype : uint8
0 NANs in -> Inst_E.NU column, column datatype : uint8
0 NANs in -> Inst_E.OF column, column datatype : uint8
0 NANs in -> Inst_E.SO column, column datatype : uint8
0 NANs in -> Inst_E.TM column, column datatype : uint8
0 NANs in -> Max_WSP column, column datatype : float64
0 NANs in -> Min_WSP column, column datatype : float64
0 NANs in -> Mean_WSP column, column datatype : float64
0 NANs in -> Calz_km column, column datatype : float64
0 NANs in -> Calz_cnt column, column datatype : float64
0 NANs in -> TTE_cnt column, column datatype : float64
0 NANs in -> LTE_km column, column datatype : float64
0 NANs in -> LTE_cnt column, column datatype : float64
0 NANs in -> NA_km2 column, column datatype : float64
0 NANs in -> Na_cnt column, column datatype : float64
0 NANs in -> UA_km2 column, column datatype : float64
0 NANs in -> UA_cnt column, column datatype : float64
0 NANs in -> Den_P column, column datatype : float64
0 NANs in -> Pend_m column, column datatype : float64
36086 NANs in -> InEl_km2 column, column datatype : float64
0 NANs in -> Con_kWh_hex column, column datatype : float64
0 NANs in -> Con_kWh_Mun column, column datatype : float64
```

Como podemos ver en la lista de arriba, los valores nulos corresponden a 36086 y se encuentra en la columna InEl_km2. Entonces, reemplazaremos los valores nulos en dicha columna mediante la función *fillna()*, tal como se muestra con la siguiente línea de código:

In [12]:
```
tfm_hex_norm['InEl_km2'] = tfm_hex_norm['InEl_km2'].fillna(0)
```

Ya encontrados los valores nulos y su reemplazo por el valor cero, la siguiente línea de código crea una lista *col_norm* que incluye todas las columnas del dataframe "tfm_hex_norm" y se comprobará

nuevamente si hay valores nulos. Se imprimirá la lista todas las columnas con su respectivo data type y el el total de valores nulos si hubiese.

In [13]:
```
col_norm = [x for x in tfm_hex_norm.columns]
for c in col_norm:
    print(f'{tfm_hex_norm[c].isnull().sum()} NANs in -> {c} column, column datatype : {
```

```
0 NANs in -> OID column, column datatype : int64
0 NANs in -> Org_id column, column datatype : int64
0 NANs in -> VxHx_left column, column datatype : float64
0 NANs in -> VxHx_top column, column datatype : float64
0 NANs in -> VxHx_right column, column datatype : float64
0 NANs in -> VxHx_btm column, column datatype : float64
0 NANs in -> Area_Hex column, column datatype : float64
0 NANs in -> Autopista column, column datatype : uint8
0 NANs in -> Ferrocarril_Int_CONV column, column datatype : uint8
0 NANs in -> Ferrocarril_estr_vel_conv column, column datatype : uint8
0 NANs in -> Ferrocarril_int_AVE column, column datatype : uint8
0 NANs in -> Ferrocarril_vel_conv column, column datatype : uint8
0 NANs in -> Via calzada unica column, column datatype : uint8
0 NANs in -> Via doble calzada column, column datatype : uint8
0 NANs in -> Via doble preferente column, column datatype : uint8
0 NANs in -> Via unica preferente column, column datatype : uint8
0 NANs in -> Arboles aislados column, column datatype : uint8
0 NANs in -> Vertedero column, column datatype : uint8
0 NANs in -> arenas column, column datatype : uint8
0 NANs in -> bosque column, column datatype : uint8
0 NANs in -> humedal column, column datatype : uint8
0 NANs in -> pradera column, column datatype : uint8
0 NANs in -> roquerios column, column datatype : uint8
0 NANs in -> Inst_E.E column, column datatype : uint8
0 NANs in -> Inst_E.EO column, column datatype : uint8
0 NANs in -> Inst_E.HC column, column datatype : uint8
0 NANs in -> Inst_E.HE column, column datatype : uint8
0 NANs in -> Inst_E.NU column, column datatype : uint8
0 NANs in -> Inst_E.OF column, column datatype : uint8
0 NANs in -> Inst_E.SO column, column datatype : uint8
0 NANs in -> Inst_E.TM column, column datatype : uint8
0 NANs in -> Max_WSP column, column datatype : float64
0 NANs in -> Min_WSP column, column datatype : float64
0 NANs in -> Mean_WSP column, column datatype : float64
0 NANs in -> Calz_km column, column datatype : float64
0 NANs in -> Calz_cnt column, column datatype : float64
0 NANs in -> TTE_cnt column, column datatype : float64
0 NANs in -> LTE_km column, column datatype : float64
0 NANs in -> LTE_cnt column, column datatype : float64
0 NANs in -> NA_km2 column, column datatype : float64
0 NANs in -> Na_cnt column, column datatype : float64
0 NANs in -> UA_km2 column, column datatype : float64
0 NANs in -> UA_cnt column, column datatype : float64
0 NANs in -> Den_P column, column datatype : float64
0 NANs in -> Pend_m column, column datatype : float64
0 NANs in -> InEl_km2 column, column datatype : float64
0 NANs in -> Con_kWh_hex column, column datatype : float64
0 NANs in -> Con_kWh_Mun column, column datatype : float64
```

4. Selección de hexágonos para el modelo de entrenamiento y test

Debido que el conjunto de datos contiene información de tipo geoespacial debidamente georeferenciada, es importante considerar las coordenadas de los centroides o punto centro del hexágonos para la visualización de los resultados del modelo en un mapa. Por lo tanto, explicaremos la forma de obtener estos puntos centrales de ellos. También determinaremos el conjunto de datos de entrenamiento y test aplicando para ellos condiciones sobre la columna objetivo, que en este caso es *Inst_E.EO*.

4.1 Cálculo de centroide de los hexágonos

Se utilizan dos librerias. La primera de ellas es *shapely* que la utilizamos para trabajar con la geometría de los hexágonos y que nos proporciona metodos de calculo de distancias, interseccíon de lineas y áreas. La siguiente es pyproj que nos permite trabajar con projecciones cartográficas.

In [13]:
```
import shapely
import pyproj
```

Crearemos dos columnas nuevas, 'Centerpoint_X' y 'Centerpoint_Y'; que son las coordenadas del par ordenado del punto centro del hexágono. Estas las agregaremos al dataframe *tfm_hex_norm*. Estas columnas tiene las coordenadas del centro del hexágono y el calculo se realiza a través del cálculo del valor medio de entre los vértices izquierdo y derecho para el centroide X, en tanto que para la variable Y, se obtiene con el valor medio de las coordenadas superiores e inferiores del hexágono.

In [14]:
```
#Centerpoint creation

tfm_hex_norm['Centerpoint_X'] = (tfm_hex_norm['VxHx_right'] + tfm_hex_norm['VxHx_left']
tfm_hex_norm['Centerpoint_Y'] = (tfm_hex_norm['VxHx_top'] + tfm_hex_norm['VxHx_btm'])/2
tfm_hex_norm
```

Out[14]:

	OID	Org_id	VxHx_left	VxHx_top	VxHx_right	VxHx_btm	Area_Hex	Autopista	Ferroca
0	1	3	261025.6058	4515475.523	262180.3063	4514475.523	866025.4	0	
1	2	4	261025.6058	4514475.523	262180.3063	4513475.523	866025.4	0	
2	3	5	261025.6058	4513475.523	262180.3063	4512475.523	866025.4	0	
3	4	7	261891.6312	4514975.523	263046.3317	4513975.523	866025.4	0	
4	5	8	261891.6312	4513975.523	263046.3317	4512975.523	866025.4	0	
...	...	...	...	...	...	...	...	...	
37065	37066	37889	523431.3032	4684975.523	524586.0037	4683975.523	866025.4	0	
37066	37067	37892	523431.3032	4681975.523	524586.0037	4680975.523	866025.4	0	
37067	37068	37895	524297.3286	4686475.523	525452.0291	4685475.523	866025.4	0	
37068	37069	37896	524297.3286	4685475.523	525452.0291	4684475.523	866025.4	0	
37069	37070	37901	525163.3540	4685975.523	526318.0545	4684975.523	866025.4	0	

37070 rows × 50 columns

4.1 Creación del conjunto de datos de entrenamiento

Creamos el objeto "to_train" que será el conjunto de datos de entrenamiento, que se hace a partir de dos subconjuntos que se obtienen de tfm_hex_norm cuya condicionante esta en la columna "*Inst_E.EO*". El primer subconjunto se hace por medio de la selección de los registros con valores de 1. El segundo se hace sobre los registros con valor 0 y con selección aleatroria muestreal de 2000 registros.

Combinamos ambos subconjuntos utilizando la función *append()*. Este sera el conjunto de datos que se utilizará como el conjunto de datos de entrenamiento para el modelo.

In [15]:
```
to_train = tfm_hex_norm[tfm_hex_norm["Inst_E.EO"] == 1].append(tfm_hex_norm[tfm_hex_nor
to_train
```

```
C:\Users\manhi\AppData\Local\Temp/ipykernel_55592/360738906.py:1: FutureWarning: The fra
me.append method is deprecated and will be removed from pandas in a future version. Use
pandas.concat instead.
  to_train = tfm_hex_norm[tfm_hex_norm["Inst_E.EO"] == 1].append(tfm_hex_norm[tfm_hex_no
rm["Inst_E.EO"] != 1].sample(n = 2000))
```

Out[15]:

	OID	Org_id	VxHx_left	VxHx_top	VxHx_right	VxHx_btm	Area_Hex	Autopista	Ferroca
141	142	192	269685.8598	4550475.523	270840.5604	4549475.523	866025.40	0	
186	187	246	270551.8852	4550975.523	271706.5858	4549975.523	866025.40	0	
187	188	247	270551.8852	4549975.523	271706.5858	4548975.523	866025.40	0	
189	190	249	270551.8852	4547975.523	271706.5858	4546975.523	866025.40	0	
190	191	250	270551.8852	4546975.523	271706.5858	4545975.523	866025.40	0	
...	...	...	...	...	...	...	...	...	
10006	10007	10391	326843.5365	4599475.523	327998.2370	4598475.523	866025.40	0	
35114	35115	35822	492254.3886	4686975.523	493409.0892	4685975.523	866025.45	0	
19889	19890	20372	374474.9337	4615975.523	375629.6342	4614975.523	866025.40	0	
24790	24791	25318	404785.8228	4636475.523	405940.5234	4635475.523	866025.40	0	
35309	35310	36020	493986.4394	4659975.523	495141.1400	4658975.523	866025.40	0	

2315 rows × 50 columns

una vez creado el conjunto de datos de entrenamiento, eliminaremos las columnas que no son necesarias y que están identificadas en la lista *columns*.

In [16]:
```
columns = ['VxHx_left', 'VxHx_top', 'VxHx_right', 'VxHx_btm', 'OID', 'Org_id', 'Area_He
train_data = to_train.drop(columns,  axis=1, inplace=False)
train_data
```

Out[16]:

	Autopista	Ferrocarril_Int_CONV	Ferrocarril_estr_vel_conv	Ferrocarril_int_AVE	Ferrocarril_vel_conv
141	0	0	0	0	0
186	0	0	0	0	0
187	0	0	0	0	0
189	0	0	0	0	0
190	0	0	0	0	0
...	...	...	...	...	...
10006	0	0	0	0	0
35114	0	0	0	1	0
19889	0	0	0	0	0
24790	0	0	0	0	0
35309	0	0	0	0	0

2315 rows × 41 columns

4.1 Creación de variables dependientes e independientes

Con *train_test_split() de Scikit-learn* se dividirá el conjunto de datos para entrenamiento y test. Esta división se realiza para entrenar el modelo en los datos de entrenamiento y evaluar el modelo en los datos test y verificar su rendimiento.

Por consiguiente, separaremos los datos en variables de entrada (X) y variable de salida (y). X contiene las variables independientes que se utilizarán para predecir la variable dependiente (y). En este caso, la variable dependiente es *Inst_E.EO*.

In [17]:
```
from sklearn.model_selection import train_test_split
```

In [18]:
```
X = train_data.drop(['Inst_E.EO'], axis=1, inplace=False)
y = train_data['Inst_E.EO']
```

In [19]:
```
X
```

Out[19]:

	Autopista	Ferrocarril_Int_CONV	Ferrocarril_estr_vel_conv	Ferrocarril_int_AVE	Ferrocarril_vel_conv
141	0	0	0	0	0
186	0	0	0	0	0
187	0	0	0	0	0

	Autopista	Ferrocarril_Int_CONV	Ferrocarril_estr_vel_conv	Ferrocarril_int_AVE	Ferrocarril_vel_conv
189	0	0	0	0	0
190	0	0	0	0	0
...	...	...	...	...	...
10006	0	0	0	0	0
35114	0	0	0	1	0
19889	0	0	0	0	0
24790	0	0	0	0	0
35309	0	0	0	0	0

2315 rows × 40 columns

ya con los conjuntos de datos de entrenamiento y test; y establecemos un *test_size* de un 25% con un *random_state* de 42.

In [20]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=
```

In [21]:
```
X_train.columns
```

Out[21]:
```
Index(['Autopista', 'Ferrocarril_Int_CONV', 'Ferrocarril_estr_vel_conv',
       'Ferrocarril_int_AVE', 'Ferrocarril_vel_conv', 'Via calzada unica',
       'Via doble calzada', 'Via doble preferente', 'Via unica preferente',
       'Arboles aislados', 'Vertedero', 'arenas', 'bosque', 'humedal',
       'pradera', 'roquerios', 'Inst_E.E', 'Inst_E.HC', 'Inst_E.HE',
       'Inst_E.NU', 'Inst_E.OF', 'Inst_E.SO', 'Inst_E.TM', 'Max_WSP',
       'Min_WSP', 'Mean_WSP', 'Calz_km', 'Calz_cnt', 'TTE_cnt', 'LTE_km',
       'LTE_cnt', 'NA_km2', 'Na_cnt', 'UA_km2', 'UA_cnt', 'Den_P', 'Pend_m',
       'InEl_km2', 'Con_kWh_hex', 'Con_kWh_Mun'],
      dtype='object')
```

5. Análisis de Componentes principales (PCA)

Con el conjunto de datos de entrenamiento (X_train), procedemos a realizar un análisis de componentes principales (PCA) cuya finalidad es reducir la dimensionalidad de los datos del dataframe. Para ello, utilizaremos la función PCA de la librería *sklearn.decomposition*.

5.1 Determinación del numero de componentes principales

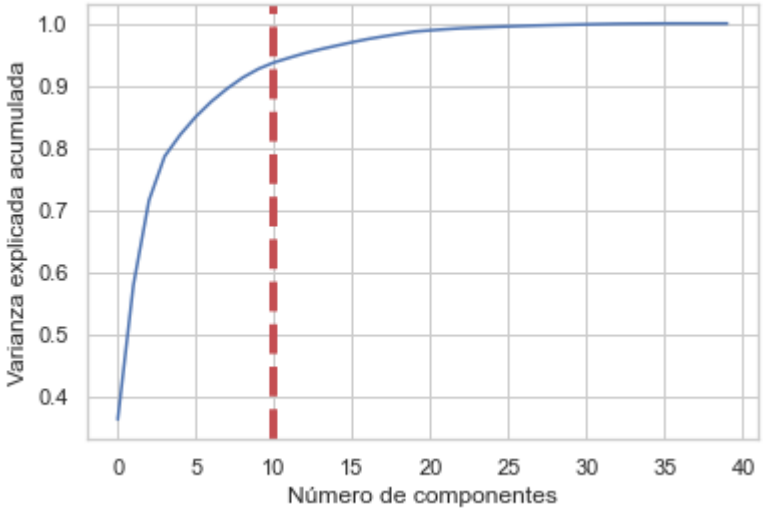
Mediante la aplicación de PCA, intentaremos determinar el número de componentes para explicar al menos el 95% de varianza y construiremos un gráfico para visualizar la varianza explicada acumulada (con una línea horizontal que nos indica el umbral del 95%)

In [22]:
```
from sklearn.decomposition import PCA
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

In [23]:

```
pca_test = PCA(n_components=40)
pca_test.fit(X_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('Número de componentes')
plt.ylabel('Varianza explicada acumulada')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
display(plt.show())
evr = pca_test.explained_variance_ratio_
cvr = np.cumsum(pca_test.explained_variance_ratio_)
pca_df = pd.DataFrame()
pca_df['Tasa de Varianza Acumulada'] = cvr
pca_df['Tasa de Varianza Explicada'] = evr
display(pca_df.head(10))
```



None

	Tasa de Varianza Acumulada	Tasa de Varianza Explicada
0	0.362896	0.362896
1	0.578891	0.215995
2	0.715567	0.136676
3	0.786196	0.070629
4	0.821084	0.034888
5	0.850184	0.029100
6	0.874591	0.024407
7	0.895191	0.020600
8	0.912958	0.017767
9	0.926893	0.013935

El modelo de PCA ajustó los datos de entrenamiento donde el gráfico muestra la relación entre el número de componentes principales y la varianza acumulada explicada por esos componentes. Como se aprecia en el grafico, a partir del componente número 10, obtenemos una varianza

explicada acumulada del 95%. Eso si, para alcanzar un valor del 100% se alcanza con 25 componentes. La línea vertical de color rojo indica la posición de diez componentes principales.

Por otro lado, la siguiente línea de código, crea un objeto PCA con los 10 componentes principales (n_components=10) obtenidos del primer ánalisis. Se ajustan los datos de entrenamiento (pca.fit(X_train)) y se aplican a ambos conjuntos de datos (pca.transform(X_train) y pca.transform(X_test)). Esto para obtener los nuevos conjuntos de datos con la dimensionalidad reducida.

In [24]:
```
pca = PCA(n_components=10)
pca.fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```

6. Ajuste (*Tuning*) del modelo.

6.1 Optimización de hiperparámetros mediante GridSearchCV

Para encontrar los mejores parámetros para el modelo, realizaremos un un proceso de optimización o *tuning*, donde se va a ajustar un modelo de Random Forest Classifier (rfc) utilizando la técnica de validación cruzada y la búsqueda en cuadrícula (GridSearchCV).

In [25]:
```
from sklearn.model_selection import GridSearchCV
```

In [26]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
```

A continuación definiremos la cuadrícula de posibles valores para los hiperparámetros del modelo: *(n_estimators, max_depth, min_samples_split, min_samples_leaf)* donde buscaremos evaluar todas las combinaciones posibles y entrenar el modelo con los datos del conjunto de entrenamiento (X_train_pca) que transformamos mediante el análisis de componentes principales.

En tanto que, *GridSearchCV* buscará entre todas las posibles de las combinaciones los hiperparámetros para *RandomForestClassifier* y nos devolverá el modelo que mejor se ajusta a los datos de entrenamiento (X_train_pca).

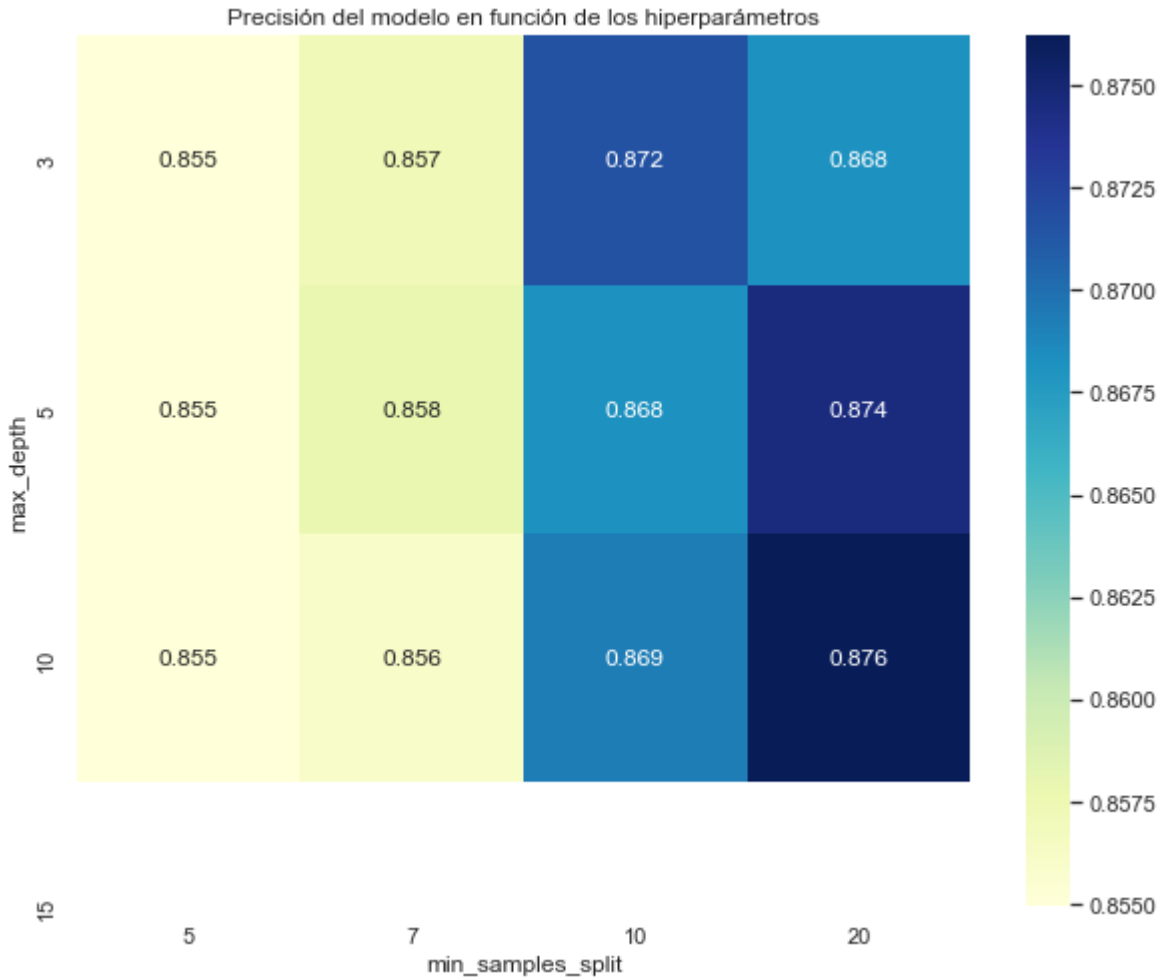
In [31]:
```
n_estimators = [50, 100, 200]
max_depth = [3,5,10,15]
min_samples_split = [5,7,10,20]
min_samples_leaf = [5,7,10]
param_grid = {'n_estimators': n_estimators,
              'max_depth': max_depth,
              'min_samples_split': min_samples_split,
              'min_samples_leaf': min_samples_leaf}
clf = GridSearchCV(rfc, param_grid, cv = 3)
clf.fit(X_train_pca, y_train)
est = clf.best_estimator_
clf.best_params_
```

Out[31]:
```
{'max_depth': 15,
 'min_samples_leaf': 5,
 'min_samples_split': 7,
 'n_estimators': 100}
```

6.2 Visualización de los valores de los hiperparámetros

A modo de visualizacion, utilizaremos la matriz de precisión media para la combinación de hiperparámetros e identificar los valores de *max_depth y min_samples_split* que producen la precisión media más alta en el modelo.

In [29]:
```
from sklearn.metrics import accuracy_score

scores = np.zeros((len(n_estimators), len(max_depth), len(min_samples_split), len(min_s
for i, n in enumerate(n_estimators):
    for j, d in enumerate(max_depth):
        for k, s in enumerate(min_samples_split):
            for l, m in enumerate(min_samples_leaf):
                rfc = RandomForestClassifier(n_estimators=n, max_depth=d, min_samples_s
                rfc.fit(X_train_pca, y_train)
                y_pred = rfc.predict(X_test_pca)
                score = accuracy_score(y_test, y_pred)
                scores[i, j, k, l] = score

# Crear la figura de la matriz de calor
fig, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(np.mean(scores, axis=3)[:,:,1], annot=True, fmt=".3f",
            xticklabels=min_samples_split, yticklabels=max_depth, cmap="YlGnBu", ax=ax)
ax.set_xlabel("min_samples_split")
ax.set_ylabel("max_depth")
plt.title("Precisión del modelo en función de los hiperparámetros")
plt.show()
```



Según lo anterior, *max_depth* entrega un valor de 15 y para *min_samples_split* es un valor de 7, por lo tanto, la precisión media del modelo con la combinación de hiperparámetros en el cuadrante con estos valores es de un 0.879. Esto significa que el modelo tiene en promedio una precisión del 87.9% para los datos de prueba.

7. Predicciones y resultados del modelo

7.1 Precision del modelo en el cojunto de datos de entrenamiento (X_test_pca)

Esta sección de código está relacionada con la evaluación y uso del modelo para hacer predicciones en datos nuevos. La función accuracy_score de la librería *Scikit-learn*, comparará los valores reales del conjunto "*y_test*" con las predicciones hechas por el modelo *(y_pred)*. El resultado se muestra como un porcentaje de precisiónm que equivale a un 87.74%.

In [33]:
```
y_pred = est.predict(X_test_pca)
print(f'Accuracy score: {round(accuracy_score(y_test, y_pred), 4)*100}%')
```

```
Accuracy score: 88.08%
```

En la siguiente linea de código, se preparán los datos para hacer nuevas predicciones con el modelo uya entrenado. Para ello, lista col_to_drop tiene los nombres de algunas columnas que se van a eliminar de los datos de entrada, luego filtraremos los datos de entrada *tfm_hex_norm* y eliminar las filas cuyos OID están en la variable *to_train*. Para terminar, eliminaremos las columnas de la variable

Org_id y pero esta se almacenará en la variable results. Ahora bien, se eliminarán las columnas de la lista *col_to_drop* y se almacenaná en la variable *to_predict*. Para verificar que todas las columnas inncesarias se han removido, aplicaremos la función print para desplegar los nombres de las columnas en to_predict.

In [34]:
```
col_to_drop = ['VxHx_left', 'VxHx_top', 'VxHx_right', 'VxHx_btm', 'OID', 'Org_id', 'Are
data_for_predict = tfm_hex_norm[tfm_hex_norm['OID'].isin(to_train['OID']) == False]
results = data_for_predict.drop('Org_id', 1)
to_predict = data_for_predict.drop(col_to_drop, 1)
to_predict.columns
```

```
C:\Users\manhi\AppData\Local\Temp/ipykernel_55592/285050031.py:3: FutureWarning: In a fu
ture version of pandas all arguments of DataFrame.drop except for the argument 'labels'
will be keyword-only.
  results = data_for_predict.drop('Org_id', 1)
C:\Users\manhi\AppData\Local\Temp/ipykernel_55592/285050031.py:4: FutureWarning: In a fu
ture version of pandas all arguments of DataFrame.drop except for the argument 'labels'
will be keyword-only.
  to_predict = data_for_predict.drop(col_to_drop, 1)
```

Out[34]:
```
Index(['Autopista', 'Ferrocarril_Int_CONV', 'Ferrocarril_estr_vel_conv',
       'Ferrocarril_int_AVE', 'Ferrocarril_vel_conv', 'Via calzada unica',
       'Via doble calzada', 'Via doble preferente', 'Via unica preferente',
       'Arboles aislados', 'Vertedero', 'arenas', 'bosque', 'humedal',
       'pradera', 'roquerios', 'Inst_E.E', 'Inst_E.HC', 'Inst_E.HE',
       'Inst_E.NU', 'Inst_E.OF', 'Inst_E.SO', 'Inst_E.TM', 'Max_WSP',
       'Min_WSP', 'Mean_WSP', 'Calz_km', 'Calz_cnt', 'TTE_cnt', 'LTE_km',
       'LTE_cnt', 'NA_km2', 'Na_cnt', 'UA_km2', 'UA_cnt', 'Den_P', 'Pend_m',
       'InEl_km2', 'Con_kWh_hex', 'Con_kWh_Mun'],
      dtype='object')
```

7.2 Predicción y Resultados

Para la predicción de resultados, se transforma el mismo conjunto de datos PCA que se uso para entrenar el modelo (X_predict_pca = pca.transform(to_predict)). Al utilizar el modelo entrenado (est) se podrá predecir la variable objetivo (y_predicted = est.predict(X_predict_pca)).

Finalmente, creamos la variable **Prediccion_Eolica** y ésta se agrega al dataFrame **results** que contiene todas las variables de entrada junto con la variable objetivo. La variable ***Prediccion_Eolica*** es la variable objetivo predicha por el modelo para los datos de entrada en to_predict.

In [35]:
```
X_predict_pca = pca.transform(to_predict)
```

In [36]:
```
y_predicted = est.predict(X_predict_pca)
y_predicted
```

Out[36]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=uint8)
```

In [37]:
```
results['Prediccion_Eolica'] = y_predicted

results
```

Out[37]:

	OID	VxHx_left	VxHx_top	VxHx_right	VxHx_btm	Area_Hex	Autopista	Ferrocarril_Int_C
0	1	261025.6058	4515475.523	262180.3063	4514475.523	866025.4	0	
1	2	261025.6058	4514475.523	262180.3063	4513475.523	866025.4	0	
2	3	261025.6058	4513475.523	262180.3063	4512475.523	866025.4	0	
3	4	261891.6312	4514975.523	263046.3317	4513975.523	866025.4	0	
4	5	261891.6312	4513975.523	263046.3317	4512975.523	866025.4	0	
...	...	...	...	...	...	...	...	...
37065	37066	523431.3032	4684975.523	524586.0037	4683975.523	866025.4	0	
37066	37067	523431.3032	4681975.523	524586.0037	4680975.523	866025.4	0	
37067	37068	524297.3286	4686475.523	525452.0291	4685475.523	866025.4	0	
37068	37069	524297.3286	4685475.523	525452.0291	4684475.523	866025.4	0	
37069	37070	525163.3540	4685975.523	526318.0545	4684975.523	866025.4	0	

34755 rows × 50 columns

Ya con los datos de predicción en guardados en un dataframe, procedemos a exportar un nuevo conjunto de datos que tiene las coordenadas de los lugares que son factibles para el emplazamiento de parques eólicos en la comunidad autonoma de Cataluña.

In [38]:
```
rslts = results[results["Prediccion_Eolica"] == 1]
rslts.to_csv(r"D:\TFM_UCM\RESULTADOS\results_TFM.csv")
```

In [39]:
```
rslts
```

Out[39]:

	OID	VxHx_left	VxHx_top	VxHx_right	VxHx_btm	Area_Hex	Autopista	Ferrocarril_Int_C
13	14	264489.7074	4554475.523	265644.4080	4553475.523	866025.4	0	
15	16	264489.7074	4552475.523	265644.4080	4551475.523	866025.4	0	
20	21	265355.7328	4556975.523	266510.4334	4555975.523	866025.4	0	
21	22	265355.7328	4555975.523	266510.4334	4554975.523	866025.4	0	
22	23	265355.7328	4554975.523	266510.4334	4553975.523	866025.4	0	
...	...	...	...	...	...	...	...	...
36168	36169	503512.7189	4674475.523	504667.4194	4673475.523	866025.4	0	
36241	36242	504378.7443	4673975.523	505533.4448	4672975.523	866025.4	0	
36314	36315	505244.7697	4672475.523	506399.4702	4671475.523	866025.4	0	
36452	36453	506976.8205	4669475.523	508131.5210	4668475.523	866025.4	0	
36515	36516	507842.8459	4671975.523	508997.5464	4670975.523	866025.4	0	

830 rows × 50 columns

8. Productivización

8.1 Despliege de resultados en un mapa

Con la finalidad de desplegar el resultado del modelo de manera visual en un mapa, utilizamos la libreria GeoPandas que permite la integración de información georeferenciada en python. Para ello, debemos utilizar coberturas en formato *shp* de ESRI. Una vez cargadas las coberturas, procedemos a generar las líneas de código que desplieguen la información de referencia. Las coberturas que se utilizan son:

1. **Catalunya_munis.shp**: Polígono de la división administrativa de la comunidad de cataluña. Cada area representa un municipio
2. **Catalunya_pland_eol.shp**: Se de puntos cuyas coordenadas indican donde se planea construir un parque eólico. Cabe destacar que esta información no fue incorporada al modelo, debido a que todas estan en tramitación ambiental.
3. **Catalunya_eol_now.shp**: Polígono que muestra las áreas donde existen parques eólicos dentro de la comunidad de Cataluña. La existencia de esta informacion esta contenida en la columna 'Inst_E.EO'.
4. **Eolica_model_result.shp** : Cobertura de puntos que muestra el resultado del modelo analizado. Estos son los lugares que el modelo predíjo y donde los hexagonos cuentan con un factibilidad sobre un 85%.

In [40]:
```
import geopandas as gpd
from shapely.geometry import Point
```

In [43]:
```
# Creación de una columna nueva con los datos de coordenadas en formato POINT
geometry = [Point(xy) for xy in zip(rslts.Centerpoint_X, rslts.Centerpoint_Y)]
```

In [44]:
```
# Creación de un objeto GeoDataFrame con la columna de geometría
gdf = gpd.GeoDataFrame(rslts, geometry=geometry)
```

In [45]:
```
# Remover las columnas de Centerpoint_X y Centerpoint
gdf.drop(['Centerpoint_X', 'Centerpoint_Y'], axis=1, inplace=True)
```

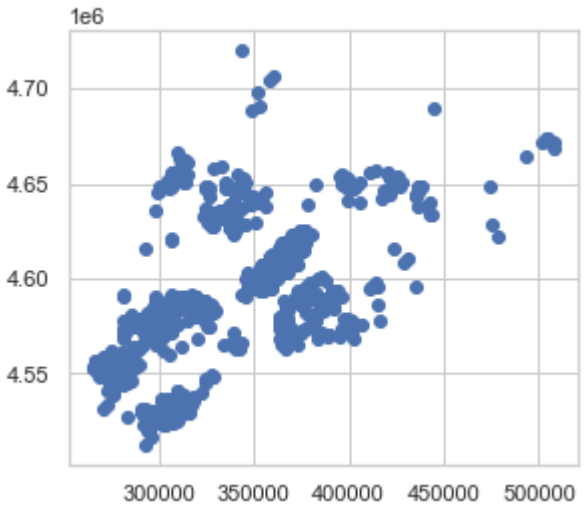
In [46]:
```
# Ver las primeras filas del GeoDataFrame resultante
print(gdf.head())
```

```
    OID    VxHx_left     VxHx_top   VxHx_right     VxHx_btm  Area_Hex  \
13   14  264489.7074  4554475.523  265644.4080  4553475.523  866025.4
15   16  264489.7074  4552475.523  265644.4080  4551475.523  866025.4
20   21  265355.7328  4556975.523  266510.4334  4555975.523  866025.4
21   22  265355.7328  4555975.523  266510.4334  4554975.523  866025.4
22   23  265355.7328  4554975.523  266510.4334  4553975.523  866025.4
```

```
    Autopista  Ferrocarril_Int_CONV  Ferrocarril_estr_vel_conv  \
13          0                     0                          0
15          0                     0                          0
20          0                     0                          0
21          0                     0                          0
22          0                     0                          0

    Ferrocarril_int_AVE  ...    Na_cnt  UA_km2  UA_cnt     Den_P    Pend_m  \
13                    0  ...  0.333333     0.0     0.0  0.000154  0.136810
15                    0  ...  0.333333     0.0     0.0  0.000154  0.178728
20                    0  ...  0.333333     0.0     0.0  0.000154  0.052166
21                    0  ...  0.333333     0.0     0.0  0.000154  0.074103
22                    0  ...  0.166667     0.0     0.0  0.000154  0.035333

    InEl_km2  Con_kWh_hex  Con_kWh_Mun  Prediccion_Eolica  \
13       0.0     0.000062     0.002339                  1
15       0.0     0.000062     0.002339                  1
20       0.0     0.000062     0.002339                  1
21       0.0     0.000062     0.002339                  1
22       0.0     0.000062     0.002339                  1

                        geometry
13  POINT (265067.058 4553975.523)
15  POINT (265067.058 4551975.523)
20  POINT (265933.083 4556475.523)
21  POINT (265933.083 4555475.523)
22  POINT (265933.083 4554475.523)

[5 rows x 49 columns]
```

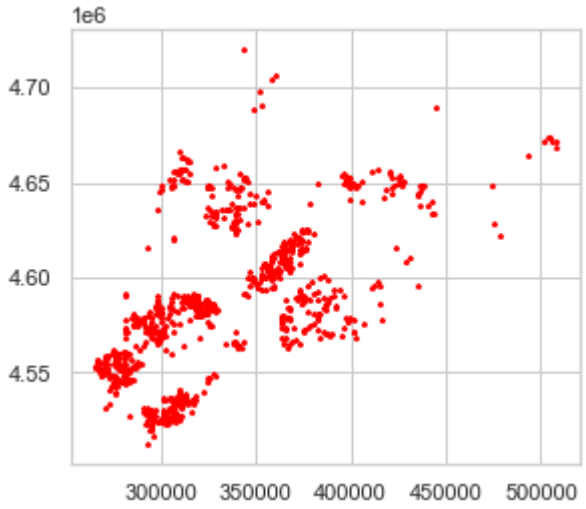
In [47]:
```
gdf.plot()
```

Out[47]:
```
<AxesSubplot:>
```



In [48]:
```
gdf.plot(marker='*', color='red', markersize=5)
```

Out[48]:
```
<AxesSubplot:>
```



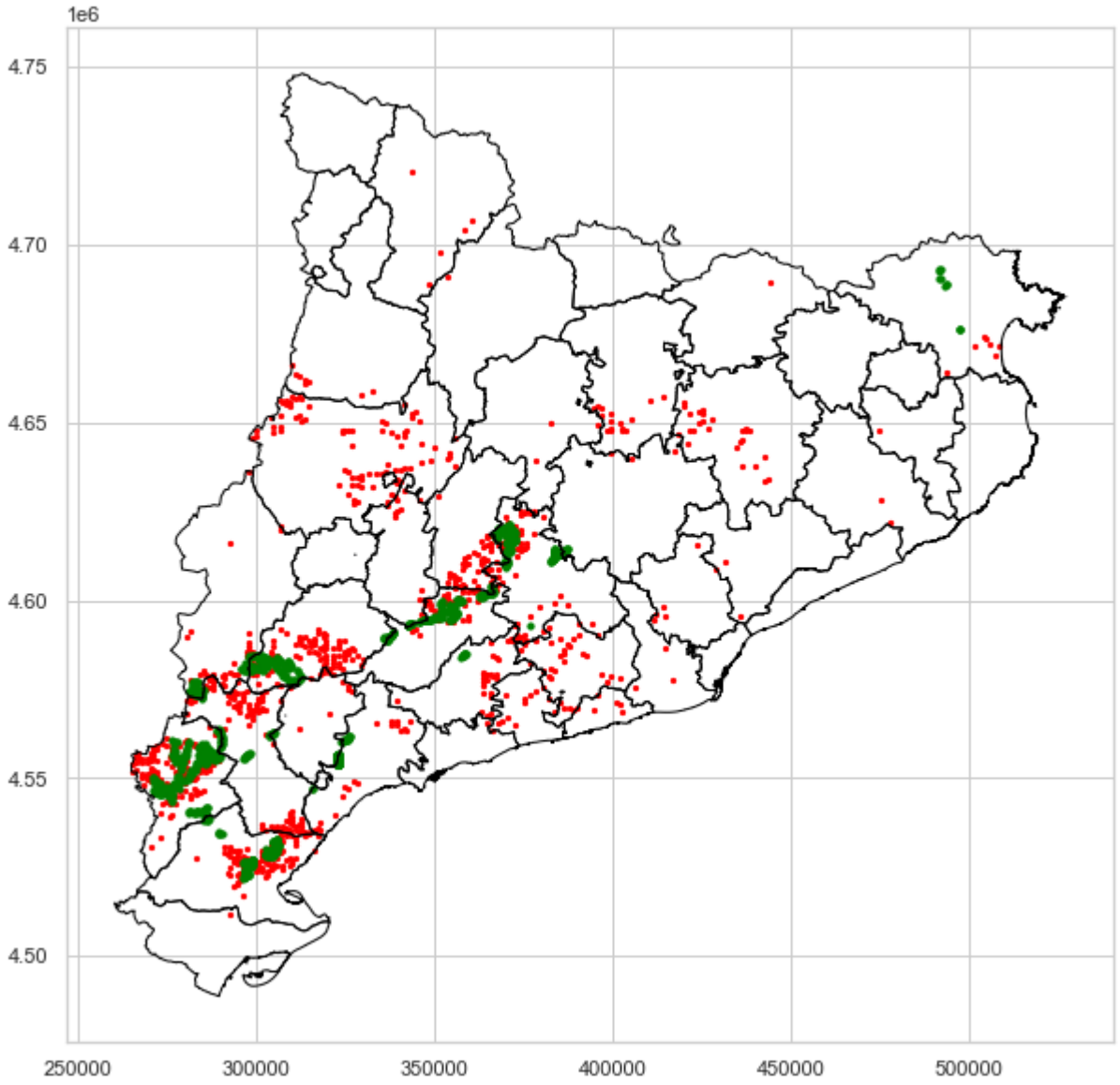
Las coberturas que estamos desplegando son:

Comunidad de catalunya, límites administrativos a nivel de comarcas y municipios Localización de parques eolicos en etapa de tramitacion ambiental Localización de los actuales parques eólicos en la comunidad Coordenadas de los centroides de los hexagonos que se obtuvieron en los procesos anteriores.

In [55]:
```
#Importar shapefile #Nota: El Sistema de projeccion oficial para Catalunya es ETRS 1989
shp_comarc = gpd.read_file(r'D:\TFM_UCM\TFM_SHP\Catalunya_admin_Comarcas.shp') #Poly
#shp_muni = gpd.read_file(r'D:\TFM_UCM\TFM_SHP\Catalunya_admin_Municipios.shp') #Poly
shp_plnd = gpd.read_file(r'D:\TFM_UCM\TFM_SHP\Catalunya_Parcs_Planned.shp') #Point
shp_parcs = gpd.read_file(r'D:\TFM_UCM\TFM_SHP\Catalunya_Parcs_Actuales.shp') #Poly
shp_rslt = gpd.read_file(r'D:\TFM_UCM\TFM_SHP\Catalunya_Resultado_Modelo.shp') #Point
```

In [56]:
```
# Creacion de un cuadrante de fondo y un eje para el mapa
fig, ax = plt.subplots(figsize=(10, 10))
gdf.plot(ax=ax,marker='o', color='red', markersize=5)
shp_parcs.plot(ax=ax, facecolor='blue', edgecolor='black')
#shp_rslt.plot(ax=ax, marker='o', color='red', markersize=5)
shp_plnd.plot(ax=ax, marker='o', color='green', markersize=10)
#shp_muni.plot(ax=ax, facecolor='lightyellow', edgecolor='grey')
shp_comarc.plot(ax=ax, facecolor='none', edgecolor='black')
```

Out[56]:
```
<AxesSubplot:>
```



8.2 Resultados

El modelo presenta un "accuracy" de un 88.08%, y eso se comprueba al desplegar los resultados en el mapa, puesto que al integrar dichos resultados en su extensión geoespacial, las coordenadas de estos registros coinciden con la ubicación de futuros proyectos que estan en tramitación ambiental por parte de la generalita de catalunya, es importante señalar que dicha información no fui incluida en el conjunto de datos del modelo ni tampoco se incluyeron en ningún proceso previo. Por lo tanto, el visualizar que dichos proyectos en estado de tramitación, cuya area fue seleccionada mediante métodos tradicionales (segun lo descrito por las DIA de los proyectos: Medición en terreno, campañas y rastreo de datos) coinciden con los hexágonos que fueron entrenados, confirma que la predicción hecha por el modelo escogido tiene una alta exactitud.

In [ ]: