1. ANÁLISIS EXPLORATORIO METEO

June 8, 2023

1 Análisis explotario del dataset METEO

[1]:
```
import pandas as pd
import numpy as np
```

[2]:
```
# Almacenamos los datos en un dataframe
df = pd.read_csv('DATOS_METEO.txt', delimiter='|')
#Visualizamos las primeras instancias
df
```

[2]:
```
                validTimeUtc  precip1Hour  precip6Hour  precip24Hour  \
0        2015-06-29 16:20:00          0.0          0.0           0.0
1        2015-06-29 17:20:00          0.0          0.0           0.0
2        2015-06-29 18:20:00          0.0          0.0           0.0
3        2015-06-29 19:20:00          0.0          0.0           0.0
4        2015-06-29 20:20:00          0.0          0.0           0.0
…                        …          …            …             …
1223655  2022-06-30 19:20:00          0.0          0.0           0.0
1223656  2022-06-30 20:20:00          0.0          0.0           0.0
1223657  2022-06-30 21:20:00          0.0          0.0           0.0
1223658  2022-06-30 22:20:00          0.0          0.0           0.0
1223659  2022-06-30 23:20:00          0.0          0.0           0.0

         precip2Day  precip3Day  precip7Day  precipMtd  precipYtd  \
0               NaN         NaN         NaN        NaN        NaN
1               NaN         NaN         NaN        NaN        NaN
2               NaN         NaN         NaN        NaN        NaN
3               NaN         NaN         NaN        NaN        NaN
4               NaN         NaN         NaN        NaN        NaN
…               …           …           …          …          …
1223655         0.0         0.0         2.0        9.0      238.0
1223656         0.0         0.0         2.0        9.0      238.0
1223657         0.0         0.0         2.0        9.0      238.0
1223658         0.0         0.0         2.0        9.0      238.0
1223659         0.0         0.0         2.0        9.0      238.0

         pressureChange  …  temperatureMax24Hour  temperatureMin24Hour  \
```

```
0                -1.4  …                36.3                17.9
1                -1.0  …                35.0                17.9
2                -0.3  …                34.7                17.9
3                 0.3  …                34.7                17.9
4                 0.9  …                34.7                17.9
…                 …    …                …                   …
1223655           2.7  …                32.3                15.1
1223656           3.7  …                32.3                15.1
1223657           3.8  …                32.3                15.1
1223658           3.1  …                32.3                15.1
1223659           2.0  …                32.3                15.1

         temperatureDewPoint  temperatureFeelsLike  uvIndex  visibility  \
0                       12.8                  34.5      2.0       16.09
1                       12.3                  34.3      1.0       16.09
2                       12.4                  32.8      0.0       16.09
3                       12.9                  31.0      0.0       16.09
4                       13.9                  28.0      0.0       16.09
…                       …                     …         …         …
1223655                 12.9                  25.1      0.0       13.55
1223656                 13.9                  22.9      0.0       13.59
1223657                 14.4                  21.0      0.0       13.85
1223658                 15.4                  19.9      0.0       13.17
1223659                 17.2                  18.7      0.0        9.51

         windDirection  windGust  windSpeed  ID_ESTACION
0                  NaN       NaN       18.7           13
1                  NaN       NaN       18.0           13
2                  NaN       NaN       16.6           13
3                  NaN       NaN       15.1           13
4                  NaN       NaN       10.1           13
…                  …         …         …              …
1223655          110.0       NaN       10.8            8
1223656           80.0       NaN        9.0            8
1223657           80.0       NaN        7.9            8
1223658           50.0       NaN        8.3            8
1223659           50.0       NaN       10.4            8

[1223660 rows x 33 columns]
```

[3]:
```
#Obtenemos información sobre la base de datos
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1223660 entries, 0 to 1223659
Data columns (total 33 columns):
 #   Column                  Non-Null Count    Dtype
---  ------                  --------------    -----
```

```
 0   validTimeUtc             1223660 non-null  object
 1   precip1Hour              1223640 non-null  float64
 2   precip6Hour              1223140 non-null  float64
 3   precip24Hour             1223540 non-null  float64
 4   precip2Day               750600 non-null   float64
 5   precip3Day               750600 non-null   float64
 6   precip7Day               750600 non-null   float64
 7   precipMtd                750600 non-null   float64
 8   precipYtd                750600 non-null   float64
 9   pressureChange           1223640 non-null  float64
 10  pressureMeanSeaLevel     867520 non-null   float64
 11  relativeHumidity         1223660 non-null  float64
 12  snow1Hour                1223640 non-null  float64
 13  snow6Hour                1223140 non-null  float64
 14  snow24Hour               1223540 non-null  float64
 15  snow2Day                 750600 non-null   float64
 16  snow3Day                 750600 non-null   float64
 17  snow7Day                 750600 non-null   float64
 18  snowMtd                  750600 non-null   float64
 19  snowSeason               750600 non-null   float64
 20  snowYtd                  750600 non-null   float64
 21  temperature              1223640 non-null  float64
 22  temperatureChange24Hour  1223280 non-null  float64
 23  temperatureMax24Hour     1223560 non-null  float64
 24  temperatureMin24Hour     1223560 non-null  float64
 25  temperatureDewPoint      1223640 non-null  float64
 26  temperatureFeelsLike     1223640 non-null  float64
 27  uvIndex                  1223640 non-null  float64
 28  visibility               1223640 non-null  float64
 29  windDirection            867520 non-null   float64
 30  windGust                 125679 non-null   float64
 31  windSpeed                1223660 non-null  float64
 32  ID_ESTACION              1223660 non-null  int64
dtypes: float64(31), int64(1), object(1)
memory usage: 308.1+ MB
```

[4]:
```
#Obtnemos información sobre los datos nulos
df.isnull().sum()
```

[4]:
```
validTimeUtc                  0
precip1Hour                  20
precip6Hour                 520
precip24Hour                120
precip2Day                473060
precip3Day                473060
precip7Day                473060
precipMtd                 473060
```

```
precipYtd                   473060
pressureChange                  20
pressureMeanSeaLevel        356140
relativeHumidity                 0
snow1Hour                       20
snow6Hour                      520
snow24Hour                     120
snow2Day                    473060
snow3Day                    473060
snow7Day                    473060
snowMtd                     473060
snowSeason                  473060
snowYtd                     473060
temperature                     20
temperatureChange24Hour        380
temperatureMax24Hour           100
temperatureMin24Hour           100
temperatureDewPoint             20
temperatureFeelsLike            20
uvIndex                         20
visibility                      20
windDirection               356140
windGust                   1097981
windSpeed                        0
ID_ESTACION                      0
dtype: int64
```

[5]:
```
# Creamos columnas separadas para el año, mes, día y hora
df['validTimeUtc'] = pd.to_datetime(df['validTimeUtc'])

df['año'] = df['validTimeUtc'].dt.year
df['mes'] = df['validTimeUtc'].dt.month
df['dia'] = df['validTimeUtc'].dt.day
df['hora'] = df['validTimeUtc'].dt.hour

df
```

[5]:

	validTimeUtc	precip1Hour	precip6Hour	precip24Hour	\
0	2015-06-29 16:20:00	0.0	0.0	0.0	
1	2015-06-29 17:20:00	0.0	0.0	0.0	
2	2015-06-29 18:20:00	0.0	0.0	0.0	
3	2015-06-29 19:20:00	0.0	0.0	0.0	
4	2015-06-29 20:20:00	0.0	0.0	0.0	
…	…	…	…	…	
1223655	2022-06-30 19:20:00	0.0	0.0	0.0	
1223656	2022-06-30 20:20:00	0.0	0.0	0.0	
1223657	2022-06-30 21:20:00	0.0	0.0	0.0	

```
1223658 2022-06-30 22:20:00          0.0          0.0           0.0
1223659 2022-06-30 23:20:00          0.0          0.0           0.0

         precip2Day  precip3Day  precip7Day  precipMtd  precipYtd  \
0               NaN         NaN         NaN        NaN        NaN
1               NaN         NaN         NaN        NaN        NaN
2               NaN         NaN         NaN        NaN        NaN
3               NaN         NaN         NaN        NaN        NaN
4               NaN         NaN         NaN        NaN        NaN
…               …           …           …          …          …
1223655         0.0         0.0         2.0        9.0      238.0
1223656         0.0         0.0         2.0        9.0      238.0
1223657         0.0         0.0         2.0        9.0      238.0
1223658         0.0         0.0         2.0        9.0      238.0
1223659         0.0         0.0         2.0        9.0      238.0

         pressureChange  …  uvIndex  visibility  windDirection  windGust  \
0                  -1.4  …      2.0       16.09            NaN       NaN
1                  -1.0  …      1.0       16.09            NaN       NaN
2                  -0.3  …      0.0       16.09            NaN       NaN
3                   0.3  …      0.0       16.09            NaN       NaN
4                   0.9  …      0.0       16.09            NaN       NaN
…                   …    …      …         …                …         …
1223655             2.7  …      0.0       13.55          110.0       NaN
1223656             3.7  …      0.0       13.59           80.0       NaN
1223657             3.8  …      0.0       13.85           80.0       NaN
1223658             3.1  …      0.0       13.17           50.0       NaN
1223659             2.0  …      0.0        9.51           50.0       NaN

         windSpeed  ID_ESTACION   año  mes  dia  hora
0             18.7           13  2015    6   29    16
1             18.0           13  2015    6   29    17
2             16.6           13  2015    6   29    18
3             15.1           13  2015    6   29    19
4             10.1           13  2015    6   29    20
…             …              …   …  …   …    …
1223655       10.8            8  2022    6   30    19
1223656        9.0            8  2022    6   30    20
1223657        7.9            8  2022    6   30    21
1223658        8.3            8  2022    6   30    22
1223659       10.4            8  2022    6   30    23

[1223660 rows x 37 columns]
```

2 Obtención de las variables meteorológicas

En esta sección vamos a crear una serie de funciones que nos permitirán obtener variables meteorológicas por mes para cada ID_ESTACION. Estas funciones nos devolverán listas de listas, donde cada sublista corresponderá a una estación meteorológica y cada elemento de cada sublista a un mes.

En primer lugar, vamos crear dos variables que recogerán respectivamente la cantidad de lluvia y nieve registrada en los distintos meses para cada año para cada ID_ESTACION.

[6]:
```
#Creamos una lista con los diferentes valores de ID_ESTACION
Id_estacion=df.ID_ESTACION.values
Id_estacion_no_dup = list(set(Id_estacion))
Id_estacion_no_dup
```

[6]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

2.0.1 Precipitaciones y nieve por mes

[7]:
```
#Localizamos los datos no nulos de la variable "precipMtd"
df1=df[np.isnan(df['precipMtd']) == False]
df1[["hora","dia","mes","año"]]
#Parece que a partir de marzo de 2018 esta variable no presenta datos nulos.
```

[7]:

	hora	dia	mes	año
23544	16	14	3	2018
23545	17	14	3	2018
23546	18	14	3	2018
23547	19	14	3	2018
23548	20	14	3	2018
…	…	…	…	…
1223655	19	30	6	2022
1223656	20	30	6	2022
1223657	21	30	6	2022
1223658	22	30	6	2022
1223659	23	30	6	2022

[750600 rows x 4 columns]

[8]:
```
#Localizamos los datos no nulos de la variable "snowMtd"
df2=df[np.isnan(df['snowMtd']) == False]
df2[["hora","dia","mes","año"]]
#Parece que a partir de marzo de 2018 esta variable no presenta datos nulos.
```

[8]:

	hora	dia	mes	año
23544	16	14	3	2018
23545	17	14	3	2018
23546	18	14	3	2018

```
23547       19   14    3  2018
23548       20   14    3  2018
…         …   …   …   …
1223655     19   30    6  2022
1223656     20   30    6  2022
1223657     21   30    6  2022
1223658     22   30    6  2022
1223659     23   30    6  2022

[750600 rows x 4 columns]
```

Así, a partir de marzo de 2018, podemos calcular cuanto llueve/nieva por mes para cada ID_ESTACION mediante las variables "precipMtd" y "snowMtd". Para ello, tomaremos el valor que toma el último día de cada mes a última hora del día, esto es, a las 23 de la noche. Empleamos las siguientes funciones:

[9]:
```
def obtener_precip_snow(df,variable,año):
    Mes_18=range(3,13)#2018 datos desde marzo
    Mes_22=range(1,7)#2022 datos hasta junio
    Mes=range(1,13)
    Mes_30=[4,6,9,11]
    Mes_31=[1,3,5,7,8,10,12]
    Dia1=range(1,31)
    Dia2=range(1,32)
    Dia3=range(1,29)
    lista_año=[]
    if año==2018:
        for i in Id_estacion_no_dup:
            for m in Mes_18:
                #tomamos el valor que toma el último día de cada mes a última␣
↪hora del día, esto es, a las 23 de la noche
                if m in Mes_30:
                    a=(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
↪(df['dia']==30) & (df['ID_ESTACION']==i) & (df['hora']==23), variable]).
↪values
                    lista_año.append(a)
                elif m in Mes_31:
                    #tomamos el valor que toma el último día de cada mes a␣
↪última hora del día, esto es, a las 23 de la noche
                    a=(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
↪(df['dia']==31) & (df['ID_ESTACION']==i) & (df['hora']==23), variable]).
↪values
                    lista_año.append(a)
        #Aplanamos la lista
        lista_año= [valor for array in lista_año for valor in np.ndarray.
↪flatten(array)]
        #Dividimos en sublistas para cada ID
```

```
        sublists_año = []
        for i in range(0, len(lista_año), 10):
            sublists_año.append(lista_año[i:i+10])
    elif año==2022:
        for i in Id_estacion_no_dup:
            for m in Mes_22:
                if m in Mes_30:
                    #tomamos el valor que toma el último día de cada mes a␣
↪última hora del día, esto es, a las 23 de la noche
                    a=(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
↪(df['dia']==30) & (df['ID_ESTACION']==i) & (df['hora']==23), variable]).
↪values
                    lista_año.append(a)
                elif m in Mes_31:
                    #tomamos el valor que toma el último día de cada mes a␣
↪última hora del día, esto es, a las 23 de la noche
                    a=(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
↪(df['dia']==31) & (df['ID_ESTACION']==i) & (df['hora']==23), variable]).
↪values
                    lista_año.append(a)
                else:#para febrero
                    #tomamos el valor que toma el último día de cada mes a␣
↪última hora del día, esto es, a las 23 de la noche
                    a=(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
↪(df['dia']==28) & (df['ID_ESTACION']==i) & (df['hora']==23), variable]).
↪values
                    lista_año.append(a)
        #Aplanamos la lista
        lista_año= [valor for array in lista_año for valor in np.ndarray.
↪flatten(array)]
        #Dividimos en sublistas para cada ID
        sublists_año = []
        for i in range(0, len(lista_año), 6):
            sublists_año.append(lista_año[i:i+6])
    else:
        for i in Id_estacion_no_dup:
            for m in Mes:
                if m in Mes_30:
                    #tomamos el valor que toma el último día de cada mes a␣
↪última hora del día, esto es, a las 23 de la noche
                    a=(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
↪(df['dia']==30) & (df['ID_ESTACION']==i) & (df['hora']==23), variable]).
↪values
                    lista_año.append(a)
                elif m in Mes_31:
```

```
                    #tomamos el valor que toma el último día de cada mes a␣
↪última hora del día, esto es, a las 23 de la noche
                    a=(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
↪(df['dia']==31) & (df['ID_ESTACION']==i) & (df['hora']==23), variable]).
↪values
                    lista_año.append(a)
                else:#para febrero
                    #tomamos el valor que toma el último día de cada mes a␣
↪última hora del día, esto es, a las 23 de la noche
                    a=(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
↪(df['dia']==28) & (df['ID_ESTACION']==i) & (df['hora']==23), variable]).
↪values
                    lista_año.append(a)
        #Aplanamos la lista
        lista_año= [valor for array in lista_año for valor in np.ndarray.
↪flatten(array)]
        #Dividimos en sublistas para cada ID
        sublists_año = []
        for i in range(0, len(lista_año), 12):
            sublists_año.append(lista_año[i:i+12])
    return sublists_año
```

[10]:
```
precip_mes_18=obtener_precip_snow(df,'precipMtd',2018)
precip_mes_19=obtener_precip_snow(df,'precipMtd',2019)
precip_mes_20=obtener_precip_snow(df,'precipMtd',2020)
precip_mes_21=obtener_precip_snow(df,'precipMtd',2021)
precip_mes_22=obtener_precip_snow(df,'precipMtd',2022)
```

Tomando ahora la variable 'snowMtd':

[11]:
```
snow_mes_18=obtener_precip_snow(df,'snowMtd',2018)


snow_mes_19=obtener_precip_snow(df,'snowMtd',2019)


snow_mes_20=obtener_precip_snow(df,'snowMtd',2020)


snow_mes_21=obtener_precip_snow(df,'snowMtd',2021)


snow_mes_22=obtener_precip_snow(df,'snowMtd',2022)
```

Para los años 2015,2016 y 2017 debemos de usar la variables "precip24Hour" y "snow24Hour" pues no tenemos datos de "precipMtd" y "snowMtd".

[12]:
```
#Localizamos los datos nulos de dichas variables:
df3=df[np.isnan(df['precip24Hour']) == True]
df3[["hora","dia","mes","año"]]
```

[12]:

	hora	dia	mes	año
2830	17	27	10	2015
2831	18	27	10	2015
2832	19	27	10	2015
2833	20	27	10	2015
2834	21	27	10	2015
…	…	…	…	…
1165308	18	27	10	2015
1165309	19	27	10	2015
1165310	20	27	10	2015
1165311	21	27	10	2015
1167307	11	19	1	2016

[120 rows x 4 columns]

[13]:
```
df4=df[np.isnan(df['precip24Hour']) == True]
df4[["hora","dia","mes","año"]]
```

[13]:

	hora	dia	mes	año
2830	17	27	10	2015
2831	18	27	10	2015
2832	19	27	10	2015
2833	20	27	10	2015
2834	21	27	10	2015
…	…	…	…	…
1165308	18	27	10	2015
1165309	19	27	10	2015
1165310	20	27	10	2015
1165311	21	27	10	2015
1167307	11	19	1	2016

[120 rows x 4 columns]

Los nulos, en ambos casos, no corresonden a ninguna última hora de ningún día

[14]:
```
#Notar que (df.loc[(df['año'] == 2015) & (df['mes'] == m) &
 ↪(df['ID_ESTACION']==i) & (df['hora']==23), 'precip24Hour']) será
#una lista de 30 o 31 valores (depende de los dias que tenga ese mes),
#donde cada valor corresponde a cuanto se registró que llovío el día que indica
 ↪la posición a las
# 11 de la noche dado el ID_ESTACION=i y el mes=m.
#Ej:
print(df.loc[(df['año'] == 2015) & (df['mes'] == 7) & (df['ID_ESTACION']==0) &
 ↪(df['hora']==23),'precip24Hour'])
```

```
print(len(df.loc[(df['año'] == 2015) & (df['mes'] == 7) &␣
 ↪(df['ID_ESTACION']==0) & (df['hora']==23),'precip24Hour']))
#Análogo para la variable snow24Hour
print(df.loc[(df['año'] == 2015) & (df['mes'] == 7) & (df['ID_ESTACION']==0) &␣
 ↪(df['hora']==23),'snow24Hour'])
print(len(df.loc[(df['año'] == 2015) & (df['mes'] == 7) &␣
 ↪(df['ID_ESTACION']==0) & (df['hora']==23),'snow24Hour']))
```

```
917800    0.0
917824    0.0
917848    0.0
917870    0.0
917894    0.0
917918    0.0
917942    0.0
917965    0.0
917989    0.0
918011    0.0
918035    0.0
918059    0.0
918083    0.0
918107    0.0
918131    0.0
918155    0.0
918179    0.0
918203    0.0
918227    0.0
918251    3.1
918275    0.0
918299    0.0
918323    0.3
918347    0.0
918371    0.0
918395    0.0
918418    0.0
918442    0.2
918465    0.0
918489    8.8
918513    0.0
Name: precip24Hour, dtype: float64
31
917800    0.0
917824    0.0
917848    0.0
917870    0.0
917894    0.0
917918    0.0
```

```
917942    0.0
917965    0.0
917989    0.0
918011    0.0
918035    0.0
918059    0.0
918083    0.0
918107    0.0
918131    0.0
918155    0.0
918179    0.0
918203    0.0
918227    0.0
918251    0.0
918275    0.0
918299    0.0
918323    0.0
918347    0.0
918371    0.0
918395    0.0
918418    0.0
918442    0.0
918465    0.0
918489    0.0
918513    0.0
Name: snow24Hour, dtype: float64
31
```

Lo que haremos mediante el uso de la siguiente función es,dados los parámetros de interés, calculamos, para cada mes de cada ID_ESTACION, la media de los registros de cuanto llovío/nevó por día utilizando para ello el dato de la última hora (esto es de las 23 de la noche) de cada uno de los días, y que multiplicar los valores obtenidos por los días de cada mes.

[15]:
```python
def precip_snow_24h(df,variable,año):
    Mes=range(1,13)
    Mes_15=range(7,13)#Omitimos Junio de 2015 pues solo tenemos dos datos
    lista_día=[]
    if año==2015:
        for i in Id_estacion_no_dup:
            for m in Mes_15:
                    #utilizamos el dato de la ultima hora del dia
                    a=np.mean(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
 ↪(df['ID_ESTACION']==i) & (df['hora']==23), variable])
                    lista_día.append(a)
        #Vamos a crear sublistas para cada ID:
        sublists_día = []
        for i in range(0, len(lista_día), 6):
            sublists_día.append(lista_día[i:i+6])
```

```
        #Ahora los datos están en cantidad lluvia/día. Tendriamos que␣
↪multiplicar por los días de cada mes.
        mult1=30
        mult2=31
        for i in Id_estacion_no_dup:
            for j in range(0,6):
                if j==0 or j==1 or j==3 or j==5: #Julio,Agosto, Octubre y␣
↪Diciembre
                    sublists_día[i][j]=sublists_día[i][j]*mult2
                else:
                    sublists_día[i][j]=sublists_día[i][j]*mult1
    else:
        for i in Id_estacion_no_dup:
            for m in Mes:
                    a=np.mean(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
↪(df['ID_ESTACION']==i) & (df['hora']==23), variable])
                    lista_día.append(a)
        #Vamos a crear sublistas para cada ID:
        sublists_día = []
        for i in range(0, len(lista_día), 12):
            sublists_día.append(lista_día[i:i+12])

        #Ahora los datos están en cantidad lluvia/día. Tendriamos que␣
↪multiplicar por los días de cada mes.
        mult1=30
        mult2=31
        mult3=29#2016 bisiesto
        mult4=28
        for i in Id_estacion_no_dup:
            for j in range(0,12):
                if j==0 or j==2 or j==4 or j==6 or j==7 or j==9 or j==11:
↪#Enero,Marzo,Julio,Agosto, Octubre y Diciembre
                    sublists_día[i][j]=sublists_día[i][j]*mult2
                elif j==1 and año == 2016: #Febero 2016
                    sublists_día[i][j]=sublists_día[i][j]*mult3
                elif j==1 and (año == 2017 or año==2018): #Febrero 2017 y 2018
                     sublists_día[i][j]=sublists_día[i][j]*mult4
                else:
                     sublists_día[i][j]=sublists_día[i][j]*mult1
    return sublists_día
```

[16]:
```
precip_mes_15=precip_snow_24h(df,'precip24Hour',2015)


precip_mes_16=precip_snow_24h(df,'precip24Hour',2016)
```

```
precip_mes_17=precip_snow_24h(df,'precip24Hour',2017)
```

[17]:
```
#Falta obtener enero y febrero  de 2018
precip_2018_24=precip_snow_24h(df,'precip24Hour',2018)
EyF_precip_2018=[]
for i in Id_estacion_no_dup:
    EyF_precip_2018.append(precip_2018_24[i][0:2])

#Para el año 2018, creamos una lista con todos los datos juntos porque tenemos␣
 ↪enero y febrero por un lado y los meses
#meses restantes por otro
precip_2018=[]
for i in Id_estacion_no_dup:
    precip_2018.append(EyF_precip_2018[i] + precip_mes_18[i])
precip_2018
precip_mes_18=precip_2018
```

Repetimos usando la variable 'snow24Hour'

[18]:
```
snow_mes_15=precip_snow_24h(df,'snow24Hour',2015)


snow_mes_16=precip_snow_24h(df,'snow24Hour',2016)


snow_mes_17=precip_snow_24h(df,'snow24Hour',2017)
```

[19]:
```
#Falta obtener enero y febrero  de 2018
snow_2018_24=precip_snow_24h(df,'snow24Hour',2018)
EyF_snow_2018=[]
for i in Id_estacion_no_dup:
    EyF_snow_2018.append(snow_2018_24[i][0:2])
#Para el año 2018, creamos una lista con todos los datos juntos porque teníamos␣
 ↪enero y febrero por un lado y
#meses restantes por otro
snow_2018=[]
for i in Id_estacion_no_dup:
    snow_2018.append(EyF_snow_2018[i] + snow_mes_18[i])
snow_mes_18=snow_2018
```

2.0.2 Temperatura mínima/máxima, humedad relativa media y fuerza del viento media por mes

Para la temperatura mínima y máxima de cada mes usaremos las variables 'temperatureMin24Hour' y 'temperatureMax24Hour'

[20]:
```
#Localizamos los datos nulos de dichas variables:
df5=df[np.isnan(df['temperatureMin24Hour']) == True]
```

```
print(df5[["hora","dia","mes","año"]])
df6=df[np.isnan(df['temperatureMax24Hour']) == True]
print(df6[["hora","dia","mes","año"]])
#Los nulos no corresonde a ninguna última hora de ningún día.
```

```
         hora  dia  mes   año
2830       17   27   10  2015
2831       18   27   10  2015
2832       19   27   10  2015
2833       20   27   10  2015
2834       21   27   10  2015
…         …  …  …  …   …
1165307    17   27   10  2015
1165308    18   27   10  2015
1165309    19   27   10  2015
1165310    20   27   10  2015
1165311    21   27   10  2015

[100 rows x 4 columns]
         hora  dia  mes   año
2830       17   27   10  2015
2831       18   27   10  2015
2832       19   27   10  2015
2833       20   27   10  2015
2834       21   27   10  2015
…         …  …  …  …   …
1165307    17   27   10  2015
1165308    18   27   10  2015
1165309    19   27   10  2015
1165310    20   27   10  2015
1165311    21   27   10  2015

[100 rows x 4 columns]
```

Para la fuerza del viendo media y la humedad realtiva por mes, usaremos las variables 'windSpeed' y 'relativeHumidity'

[21]:
```
#Recordemos que las variables windSpeed y relativeHumidity no presentaban datos␣
↪nulos.
print(df['windSpeed'].isnull().sum())
print(df['relativeHumidity'].isnull().sum())
```

```
0
0
```

Definirimos una funcion que dados los parámetros de interés calcule, para cada mes de cada ID_ESTACION, los registros de la temperatura mínima (máxima) por mes como el mínimo (máximo) de los datos de la última hora (esto es de las 23 de la noche) de cada uno de los días.

[22]:
```
def calcula_t(df,variable,año,funcion):
    Mes_15=range(7,13) #Omitimos junio de 2015 pues solo tenemos información␣
 ↪para dos días
    Mes=range(1,13)
    Mes_22=range(1,7)
    hum=[]
    if año==2015:
        for i in Id_estacion_no_dup:
            for m in Mes_15:
                    a=funcion(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
 ↪(df['ID_ESTACION']==i) & (df['hora']==23), variable])
                    hum.append(a)
        #Vamos a crear sublistas para cada ID:
        sublists = []
        for i in range(0, len(hum), 6):
            sublists.append(hum[i:i+6])
    elif año==2022:
        for i in Id_estacion_no_dup:
            for m in Mes_22:
                    a=funcion(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
 ↪(df['ID_ESTACION']==i) & (df['hora']==23), variable])
                    hum.append(a)
        #Vamos a crear sublistas para cada ID:
        sublists = []
        for i in range(0, len(hum), 6):
            sublists.append(hum[i:i+6])
    else:
        for i in Id_estacion_no_dup:
            for m in Mes:
                    a=funcion(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
 ↪(df['ID_ESTACION']==i) & (df['hora']==23), variable])
                    hum.append(a)
        #Vamos a crear sublistas para cada ID:
        sublists = []
        for i in range(0, len(hum), 12):
            sublists.append(hum[i:i+12])
    return sublists
```

Para la fuerza de viento media en cada mes y humedad media relativa en cada mes definimos una función que dados los parámetros de interés, calcule, para cada mes de cada ID_ESTACION, la media de los registros de la fuerza del viento media y la humedad realitva media por día para cada ID_ESTACION.

[23]:
```
def calcula_h_v(df,variable,año,funcion):
    Mes_15=range(7,13) #Omitimos junio de 2015 pues solo tenemos información␣
 ↪para dos días
    Mes=range(1,13)
```

```
    Mes_22=range(1,7)
    hum=[]
    if año==2015:
        for i in Id_estacion_no_dup:
            for m in Mes_15:
                    a=funcion(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
↪(df['ID_ESTACION']==i), variable])
                    hum.append(a)
        #Vamos a crear sublistas para cada ID:
        sublists = []
        for i in range(0, len(hum), 6):
            sublists.append(hum[i:i+6])
    elif año==2022:
        for i in Id_estacion_no_dup:
            for m in Mes_22:
                    a=funcion(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
↪(df['ID_ESTACION']==i), variable])
                    hum.append(a)
        #Vamos a crear sublistas para cada ID:
        sublists = []
        for i in range(0, len(hum), 6):
            sublists.append(hum[i:i+6])
    else:
        for i in Id_estacion_no_dup:
            for m in Mes:
                    a=funcion(df.loc[(df['año'] == año) & (df['mes'] == m) &␣
↪(df['ID_ESTACION']==i), variable])
                    hum.append(a)
        #Vamos a crear sublistas para cada ID:
        sublists = []
        for i in range(0, len(hum), 12):
            sublists.append(hum[i:i+12])
    return sublists
```

[24]:
```
tempmin_mes_15=calcula_t(df,'temperatureMin24Hour',2015, min)

tempmin_mes_16=calcula_t(df,'temperatureMin24Hour',2016, min)

tempmin_mes_17=calcula_t(df,'temperatureMin24Hour',2017, min)

tempmin_mes_18=calcula_t(df,'temperatureMin24Hour',2018, min)

tempmin_mes_19=calcula_t(df,'temperatureMin24Hour',2019, min)

tempmin_mes_20=calcula_t(df,'temperatureMin24Hour',2020, min)

tempmin_mes_21=calcula_t(df,'temperatureMin24Hour',2021, min)
```

```
tempmin_mes_22=calcula_t(df,'temperatureMin24Hour',2022, min)
```

[25]:
```
tempmax_mes_15=calcula_t(df,'temperatureMax24Hour',2015, max)
tempmax_mes_16=calcula_t(df,'temperatureMax24Hour',2016, max)
tempmax_mes_17=calcula_t(df,'temperatureMax24Hour',2017, max)
tempmax_mes_18=calcula_t(df,'temperatureMax24Hour',2018, max)
tempmax_mes_19=calcula_t(df,'temperatureMax24Hour',2019, max)
tempmax_mes_20=calcula_t(df,'temperatureMax24Hour',2020, max)
tempmax_mes_21=calcula_t(df,'temperatureMax24Hour',2021, max)
tempmax_mes_22=calcula_t(df,'temperatureMax24Hour',2022, max)
```

[26]:
```
windspeed_mes_15=calcula_h_v(df,'windSpeed',2015,np.mean)
windspeed_mes_16=calcula_h_v(df,'windSpeed',2016,np.mean)
windspeed_mes_17=calcula_h_v(df,'windSpeed',2017,np.mean)
windspeed_mes_18=calcula_h_v(df,'windSpeed',2018,np.mean)
windspeed_mes_19=calcula_h_v(df,'windSpeed',2019,np.mean)
windspeed_mes_20=calcula_h_v(df,'windSpeed',2020,np.mean)
windspeed_mes_21=calcula_h_v(df,'windSpeed',2021,np.mean)
windspeed_mes_22=calcula_h_v(df,'windSpeed',2022,np.mean)
```

[27]:
```
rhum_mes_15=calcula_h_v(df,'relativeHumidity',2015,np.mean)
rhum_mes_16=calcula_h_v(df,'relativeHumidity',2016,np.mean)
rhum_mes_17=calcula_h_v(df,'relativeHumidity',2017,np.mean)
rhum_mes_18=calcula_h_v(df,'relativeHumidity',2018,np.mean)
rhum_mes_19=calcula_h_v(df,'relativeHumidity',2019,np.mean)
rhum_mes_20=calcula_h_v(df,'relativeHumidity',2020,np.mean)
```

```
rhum_mes_21=calcula_h_v(df,'relativeHumidity',2021,np.mean)

rhum_mes_22=calcula_h_v(df,'relativeHumidity',2022,np.mean)
```

Temperatura media por mes Utilizamos para calcularla la variable 'temperature'.

[28]:
```
#Estudiemos en primer lugar los datos nulos de dicha variable.
print(df.temperature.info())
print(df.temperature.isnull().sum())
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 1223660 entries, 0 to 1223659
Series name: temperature
Non-Null Count    Dtype
--------------    -----
1223640 non-null  float64
dtypes: float64(1)
memory usage: 9.3 MB
None
20
```

[29]:
```
df7=df[np.isnan(df['temperature']) == True]
df7[["hora","dia","mes","año"]]
```

[29]:

	hora	dia	mes	año
4830	11	19	1	2016
66013	11	19	1	2016
127196	11	19	1	2016
188379	11	19	1	2016
249562	11	19	1	2016
310745	11	19	1	2016
371928	11	19	1	2016
433111	11	19	1	2016
494294	11	19	1	2016
555477	11	19	1	2016
616660	11	19	1	2016
677843	11	19	1	2016
739026	11	19	1	2016
800209	11	19	1	2016
861392	11	19	1	2016
922575	11	19	1	2016
983758	11	19	1	2016
1044941	11	19	1	2016
1106124	11	19	1	2016
1167307	11	19	1	2016

Como podemos ver el número de nulos es muy pequeño en relación al número de información disponible que tenemos. En particular, no se obtienen registros de la temperatura de una hora de

un día de un mes de un año concreto, pero sí se tiene para el resto de días de ese mes para todas las horas registradas. Por tanto, vamos a omitir estos datos nulos y vamos a calcular la temperatura media de cada mes para cada ID_ESTACION empleando la funcion que definimos para calcular la humedad media y la fuerza del viento media anteriormente, introduciendo como parámetro la funcion np.nanmean en vez de np.mean.

[30]:
```
t_mes_15=calcula_h_v(df,'temperature',2015,np.nanmean)

t_mes_16=calcula_h_v(df,'temperature',2016,np.nanmean)


t_mes_17=calcula_h_v(df,'temperature',2017,np.nanmean)


t_mes_18=calcula_h_v(df,'temperature',2018,np.nanmean)


t_mes_19=calcula_h_v(df,'temperature',2019,np.nanmean)


t_mes_20=calcula_h_v(df,'temperature',2020,np.nanmean)


t_mes_21=calcula_h_v(df,'temperature',2021,np.nanmean)


t_mes_22=calcula_h_v(df,'temperature',2022,np.nanmean)
```

Nótese que todas estas funciones nos devolverán listas de listas, donde cada sublista representará una estación y cada elemento de de cada sublista un mes. Por ejemplo:

[31]:
```
precip_mes_16
#El primer elemento de la primera sublista corresponde a cuanto llovió en enero␣
 ↪de 2016 para ID_ESTACION = 0.
```

[31]:
```
[[13.399999999999999,
  11.1,
  25.900000000000002,
  118.8,
  23.944827586206895,
  2.896551724137931,
  3.2,
  15.000000000000002,
  48.1,
  79.9,
  88.89999999999999,
  371.58666666666664],
 [10.2,
```

```
 27.3,
 24.8,
 67.1,
 38.48275862068965,
 10.03448275862069,
 0.6,
 43.1,
 5.8,
 78.50000000000001,
 128.4,
 293.77666666666664],
[12.0,
 9.6,
 31.299999999999997,
 84.19999999999999,
 12.613793103448277,
 8.172413793103448,
 0.0,
 4.6,
 5.800000000000001,
 78.8,
 64.7,
 259.2633333333333],
[7.8999999999999995,
 12.9,
 30.0,
 65.3,
 32.17586206896552,
 4.344827586206897,
 0.0,
 79.0,
 20.200000000000003,
 62.0,
 108.69999999999999,
 319.71333333333337],
[9.899999999999999,
 18.099999999999998,
 48.3,
 85.30000000000001,
 26.937931034482762,
 8.586206896551724,
 0.0,
 36.5,
 20.3,
 83.2,
 96.39999999999998,
 287.88666666666666],
```

```
[10.9,
 13.499999999999998,
 29.9,
 84.80000000000001,
 34.74137931034484,
 3.93103448275862,
 0.3,
 61.0,
 28.1,
 61.199999999999996,
 85.2,
 319.92],
[5.3,
 23.5,
 25.5,
 94.0,
 43.18620689655173,
 1.2413793103448276,
 3.8,
 38.3,
 53.5,
 63.5,
 120.70000000000003,
 489.49000000000007],
[9.3,
 10.3,
 29.800000000000004,
 69.4,
 28.968965517241383,
 5.482758620689657,
 0.0,
 63.5,
 15.800000000000002,
 58.2,
 87.60000000000001,
 304.52333333333337],
[8.5,
 13.9,
 31.099999999999998,
 65.30000000000001,
 32.817241379310346,
 8.689655172413794,
 0.0,
 78.4,
 12.1,
 58.99999999999999,
 106.5,
```

```
 303.8],
[10.5,
 12.399999999999999,
 31.699999999999996,
 78.39999999999999,
 27.472413793103453,
 3.0,
 1.4000000000000001,
 66.49999999999999,
 44.400000000000006,
 63.6,
 92.1,
 332.01000000000005],
[12.9,
 12.4,
 25.900000000000002,
 102.7,
 21.165517241379312,
 3.827586206896551,
 1.8,
 19.5,
 47.599999999999994,
 76.60000000000001,
 85.70000000000002,
 392.3566666666667],
[9.0,
 10.600000000000001,
 34.5,
 81.69999999999999,
 14.324137931034484,
 7.96551724137931,
 0.0,
 35.8,
 7.5,
 74.4,
 67.4,
 234.77333333333334],
[9.1,
 10.9,
 30.0,
 65.7,
 30.46551724137931,
 2.4827586206896552,
 0.3,
 85.8,
 26.2,
 58.1,
```

```
 100.3,
 323.64],
[8.5,
 10.9,
 33.4,
 71.89999999999999,
 22.875862068965517,
 17.79310344827586,
 0.0,
 76.6,
 10.0,
 73.2,
 88.89999999999999,
 266.29],
[7.8999999999999995,
 14.399999999999999,
 31.0,
 65.39999999999999,
 32.60344827586208,
 7.137931034482759,
 0.0,
 79.7,
 15.899999999999997,
 60.900000000000006,
 105.59999999999998,
 308.76000000000005],
[8.0,
 11.7,
 30.499999999999996,
 64.40000000000002,
 30.572413793103454,
 8.172413793103448,
 0.0,
 77.7,
 13.8,
 64.5,
 93.00000000000001,
 296.25666666666666],
[10.6,
 12.5,
 28.200000000000003,
 97.69999999999999,
 26.617241379310343,
 7.655172413793102,
 0.0,
 55.5,
 14.2,
```

```
 64.0,
 81.5,
 281.06666666666666],
[10.7,
 21.799999999999997,
 23.400000000000002,
 65.69999999999999,
 36.665517241379305,
 9.620689655172415,
 1.2000000000000002,
 47.8,
 6.4,
 69.9,
 118.59999999999998,
 275.0733333333333],
[11.4,
 15.499999999999998,
 29.3,
 89.19999999999999,
 33.244827586206895,
 3.5172413793103448,
 0.3,
 45.6,
 36.8,
 66.4,
 83.39999999999999,
 353.5033333333334],
[10.4,
 10.8,
 30.2,
 77.50000000000001,
 29.71724137931034,
 3.93103448275862,
 0.1,
 63.199999999999996,
 20.599999999999998,
 56.2,
 84.50000000000001,
 308.5533333333334]]
```

[32]:
```
t_mes_15
#El primer elemento de la primera sublista corresponde a la temperatura media␣
 ↪que se registró de 2016
#en julio de 2015 para ID_ESTACION = 0 (recordemos que de 2015 solo tenemos␣
 ↪datos a partir del 29 de junio)
```

```
[32]: [[26.89606512890095,
        24.55402777777778,
        19.77394366197183,
        16.608719346049046,
        12.093016759776537,
        10.219891745602165],
       [26.1842605156038,
        23.71472222222222,
        18.572957746478874,
        15.396730245231609,
        10.979469273743018,
        8.905006765899865],
       [27.03405698778833,
        24.554305555555555,
        19.350281690140847,
        16.115395095367845,
        10.799720670391062,
        9.256968876860622],
       [26.88303934871099,
        24.47055555555556,
        19.340140845070422,
        16.066212534059943,
        11.444413407821228,
        9.733288227334235],
       [26.761194029850746,
        24.531527777777775,
        19.190985915492956,
        15.851498637602177,
        11.44441340782123,
        9.171583220568335],
       [26.43459972862958,
        24.07847222222222,
        19.022535211267606,
        15.825749318801089,
        10.988268156424581,
        9.428416779431664],
       [26.67069199457259,
        24.145833333333332,
        19.090845070422535,
        15.846321525885559,
        11.378491620111731,
        9.406901217861975],
       [26.649525101763903,
        24.281805555555557,
        19.25112676056338,
        16.024523160762943,
        11.255167597765363,
```

```
 9.510825439783492],
[26.752645861601085,
 24.278611111111108,
 19.177464788732394,
 15.932970027247956,
 11.37290502793296,
 9.515832205683356],
[26.792672998643148,
 24.43916666666666,
 19.607323943661974,
 16.36798365122616,
 11.738966480446924,
 9.861569688768608],
[27.406648575305287,
 25.071944444444448,
 20.18676056338028,
 17.04073569482289,
 12.442039106145252,
 10.588633288227335],
[26.789552238805967,
 24.26861111111111,
 18.999859154929577,
 15.770027247956405,
 10.366899441340783,
 8.854127198917455],
[27.025644504748985,
 24.672916666666666,
 19.675352112676055,
 16.431062670299728,
 11.711871508379886,
 9.883220568335588],
[26.5696065128901,
 24.3225,
 19.115211267605634,
 15.865531335149866,
 11.291480446927373,
 9.269959404600812],
[26.12550881953867,
 23.692638888888887,
 18.529154929577466,
 15.253678474114443,
 10.657402234636873,
 8.890527740189444],
[26.354409769335142,
 24.023472222222217,
 18.839859154929577,
 15.580790190735694,
```

```
 10.925279329608939,
 9.20108254397835],
[26.674355495251017,
 24.33930555555556,
 19.17154929577465,
 15.935286103542234,
 11.182821229050278,
 9.53680649526387],
[26.0640434192673,
 23.577500000000004,
 18.43084507042254,
 15.245095367847414,
 10.85223463687151,
 8.804736129905278],
[26.340162822252374,
 23.996666666666666,
 18.960985915492955,
 15.763760217983648,
 10.95377094972067,
 9.389309878213803],
[26.93514246947083,
 24.598194444444445,
 19.587605633802816,
 16.370572207084468,
 11.565642458100559,
 9.85385656292287]]
```

3 ANÁLISIS GRÁFICO

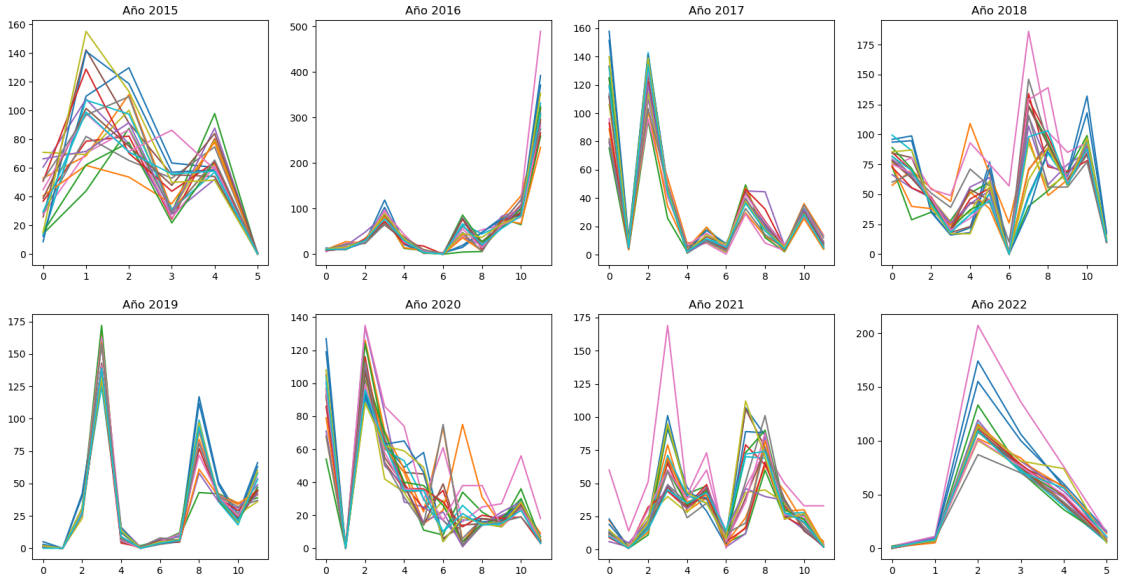
Nuestro siguiente objetivo es intentar identificar los años que son parecidos en términos meteorológicos a 2022. Para ello, comenzaremos haciendo una exploración gráfica a partir de las variables que obtuvimos en el apartado anterior. Los siguientes gráficos presentan la siguiente estructura: 1. En el eje x tendremos los distintos meses. El valor "0" indica el mes enero, el "1" el mes febrero y así sucesivamente hasta el "11" que indica el mes diciembre. En el eje y tendremos los valores que toman las variables en el año en cuestión para cada ID_ESTACION. 2. Nótese que en 2015 hemos obtenido las variables a partir del mes de julio, por lo que, para este año, el "0" indicará julio, el "1" agosto y así hasta el "5" que indicará diciembre. 3. En 2022 solo tenemos datos hasta junio.

[33]:
```
import matplotlib.pyplot as plt
```

3.0.1 Precipitaciones por mes

[34]:
```
# Definimos una lista con las variables precipitaciones
variables = [precip_mes_15, precip_mes_16, precip_mes_17, precip_mes_18,␣
 ↪precip_mes_19, precip_mes_20, precip_mes_21, precip_mes_22]

# Creamos la figura y los subplots
```

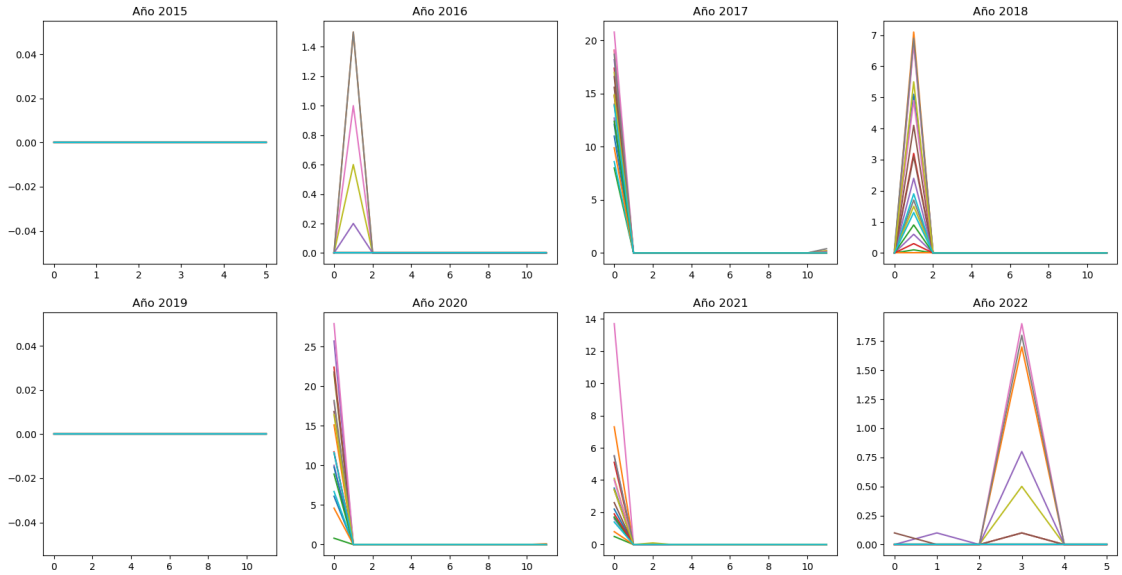
```
fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))

# Iteramos sobre cada variable y su posición en los subplots
for i, var in enumerate(variables):
    row = i // 4
    col = i % 4

    # Graficamos la variable en su subplot correspondiente
    for j in var:
        axs[row, col].plot(j)
    axs[row, col].set_title('Año {}'.format(2015+i))
# Mostramos la figura
plt.show()
```



3.0.2 Nevadas por mes

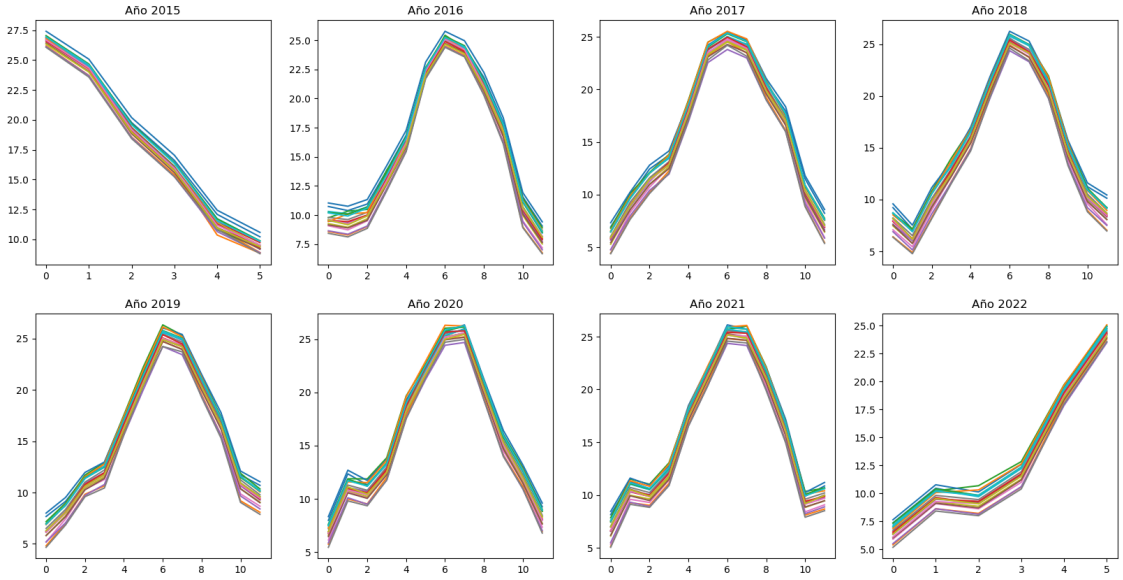
[35]:
```
# Definimos la lista de variables
variables = [snow_mes_15, snow_mes_16, snow_mes_17, snow_mes_18, snow_mes_19,␣
 ↪snow_mes_20, snow_mes_21, snow_mes_22]

# Creamos la figura y los subplots
fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))

# Iteramos sobre cada variable y su posición en los subplots
for i, var in enumerate(variables):
    row = i // 4
    col = i % 4
```

```
    # Graficamos la variable en su subplot correspondiente
    for j in var:
        axs[row, col].plot(j)
    axs[row, col].set_title('Año {}'.format(2015+i))
# Mostramos la figura
plt.show()
```



3.0.3 Temperatura media por mes

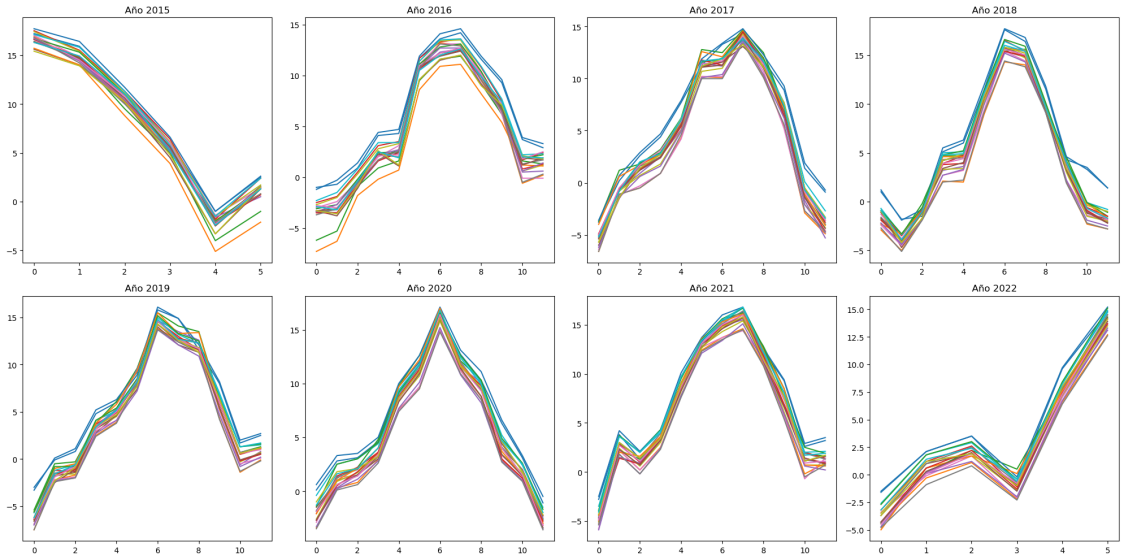
[36]:
```
# Definimos una lista con todas las variables
variables = [t_mes_15, t_mes_16, t_mes_17, t_mes_18, t_mes_19, t_mes_20,␣
 ↪t_mes_21, t_mes_22]

# Creamos la figura y los subplots
fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))

# Iteramos sobre cada variable y su posición en los subplots
for i, var in enumerate(variables):
    row = i // 4
    col = i % 4

    # Graficamos la variable en su subplot correspondiente
    for j in var:
        axs[row, col].plot(j)
    axs[row, col].set_title('Año {}'.format(2015+i))
# Mostramos la figura
plt.show()
```



3.0.4 Temperatura mínima de cada mes

[37]:
```
# Definimos una lista con todas las variables
variables = [tempmin_mes_15, tempmin_mes_16, tempmin_mes_17, tempmin_mes_18,␣
 ↪tempmin_mes_19, tempmin_mes_20,
             tempmin_mes_21, tempmin_mes_22]

# Creamos la figura y los subplots
```

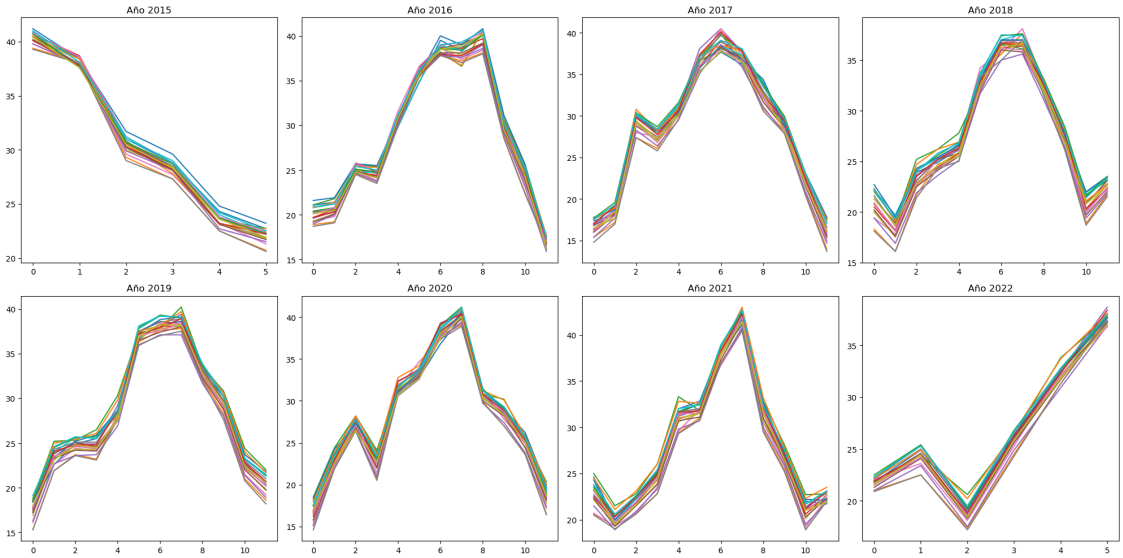
```
fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))

# Iteramos sobre cada variable y su posición en los subplots
for i, var in enumerate(variables):
    row = i // 4
    col = i % 4

    # Graficamos la variable en su subplot correspondiente
    for j in var:
        axs[row, col].plot(j)

    axs[row, col].set_title('Año {}'.format(2015+i))

# Ajustamos la separación entre subplots
fig.tight_layout()

# Mostramos la figura
plt.show()
```



3.0.5 Temperaturas máximas de cada mes

[38]:
```
variables = [tempmax_mes_15, tempmax_mes_16, tempmax_mes_17, tempmax_mes_18,␣
 ↪tempmax_mes_19,
             tempmax_mes_20, tempmax_mes_21, tempmax_mes_22]

# Creamos la figura y los subplots
fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))
```

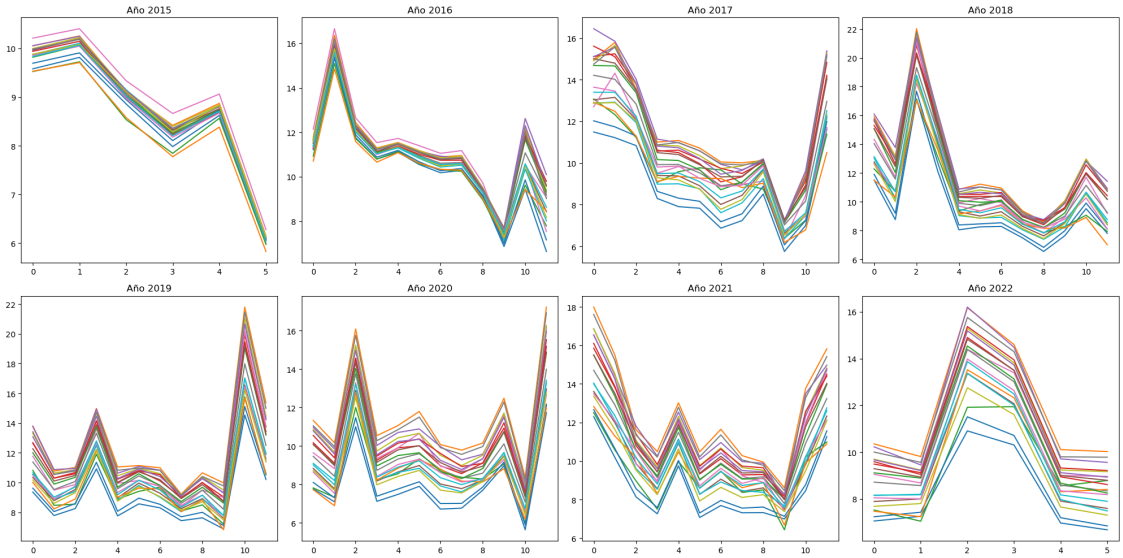
```
# Iteramos sobre cada variable y su posición en los subplots
for i, var in enumerate(variables):
    row = i // 4
    col = i % 4

   # Graficamos la variable en su subplot correspondiente
    for j in var:
        axs[row, col].plot(j)

    axs[row, col].set_title('Año {}'.format(2015+i))

# Ajustamos la separación entre subplots
fig.tight_layout()

# Mostramos la figura
plt.show()
```



3.0.6 Fuerza del viento media por mes

[39]:
```
# Definimos una lista con todas las variables
variables = [windspeed_mes_15, windspeed_mes_16, windspeed_mes_17,␣
 ↪windspeed_mes_18,
             windspeed_mes_19, windspeed_mes_20, windspeed_mes_21,␣
 ↪windspeed_mes_22]

# Creamos la figura y los subplots
fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))
```

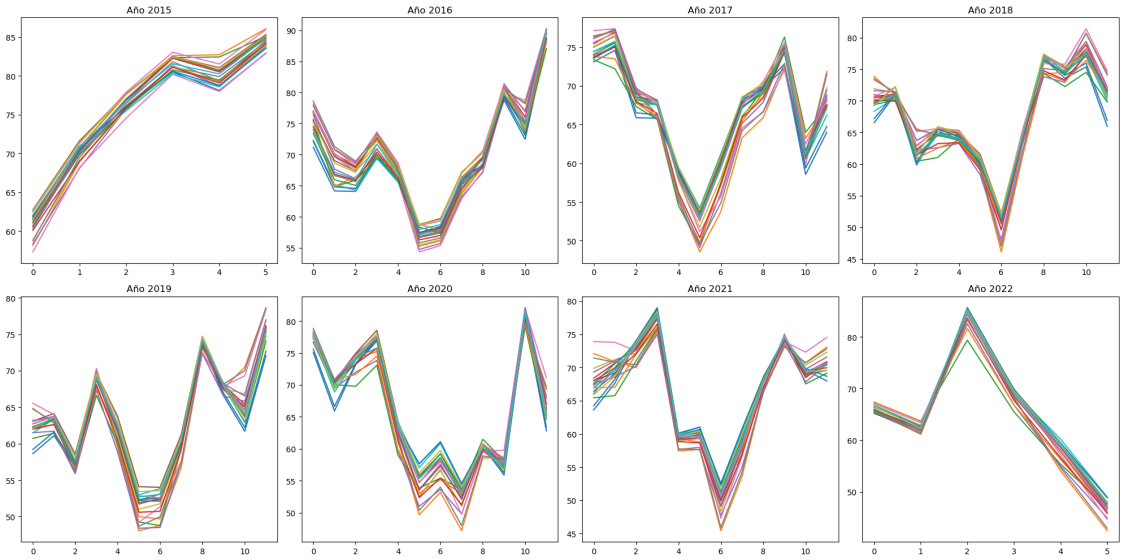
```
# Iteramos sobre cada variable y su posición en los subplots
for i, var in enumerate(variables):
    row = i // 4
    col = i % 4

   # Graficamos la variable en su subplot correspondiente
    for j in var:
        axs[row, col].plot(j)

    axs[row, col].set_title('Año {}'.format(2015+i))

# Ajustamos la separación entre subplots
fig.tight_layout()

# Mostramos la figura
plt.show()
```



3.0.7 Humedad relativa media por mes

[40]:
```
# Definimos una lista con todas las variables
variables = [rhum_mes_15, rhum_mes_16, rhum_mes_17, rhum_mes_18,
             rhum_mes_19, rhum_mes_20, rhum_mes_21, rhum_mes_22]

# Creamos la figura y los subplots
fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))

# Iteramos sobre cada variable y su posición en los subplots
for i, var in enumerate(variables):
```

```
    row = i // 4
    col = i % 4

    # Graficamos la variable en su subplot correspondiente
    for j in var:
        axs[row, col].plot(j)
    axs[row, col].set_title('Año {}'.format(2015+i))

# Ajustamos la separación entre subplots
fig.tight_layout()

# Mostramos la figura
plt.show()
```



4 Años más próximos a 2022 en términos meteorológicos

En la primera sección obtuvimos a partir de la base de datos original el valor de diversas variables meteorológicas por mes para cada estación. En la segunda sección graficamos esas variables con el objetivo de hacernos una idea de qué años se pueden parecer más en términos meteorológicos a 2022. Por ejemplo, las gráficas anteriores muestran que las temperaturas medias por mes y las precipitaciones por mes de 2021 y 2022 se parecen. El objetivo de esta sección es el de ver analíticamente cuáles son los años más cercanos a 2022 teniendo en cuenta exclusivamente variables meteorológicas. Además, buscamos que los resultados sean robustos.

Así, decidimos emplear métodos de clustering. Lo que haremos será crear un modelo de clustering para cada año comprendido entre 2016 y 2021. Cada uno de ellos clasificará cada ID_ESTACION en distintos clusters, a partir de los valores que tomen las estaciones en una serie de variables meteorológicas para ese año. Entrenaremos cada modelo creado para cada año y predecimos en qué clusters caen cada estación de 2022. Por último, para cada modelo, calcularemos la distancia

de cada estación de 2022 al cluster en el que ha caído. Almacenaremos la información en un diccionario.

Las variables meteorológicas que usaremos para realizar este análisis serán temperatura media por mes, precipitaciones por mes y humedad relativa por mes.

Lo primero será crear el dataframe de train que denominaremos df_clust que contenga la información correspondiente a estas variables.

[41]:
```
#Para cada año, tenemos 20 estaciones
campaña = [16]*20 + [17]*20 + [18]*20 + [19]*20 + [20]*20 + [21]*20
id_estacion = list(range(20))*6
data = {'CAMPAÑA': campaña, 'ID_ESTACION': id_estacion}
df_clust = pd.DataFrame(data)
df_clust
```

[41]:

	CAMPAÑA	ID_ESTACION
0	16	0
1	16	1
2	16	2
3	16	3
4	16	4
..	…	…
115	21	15
116	21	16
117	21	17
118	21	18
119	21	19

[120 rows x 2 columns]

[42]:
```
#Definimos una funcion para meter las variables meteorológicas de interés
def metervariables(df):
    meses=['enero', 'febrero', 'marzo', 'abril', 'mayo', 'junio']
    for m in range(len(meses)):
        precip_col = f"precip_{meses[m]}"
        df[precip_col] = np.nan

    # Rellenamos los valores de precip_mes en función de CAMPAÑA y ID_ESTACION
    for c in range(16, 23):
        for e in range(20):
            for m in range(6):
                precip_mes = globals()[f"precip_mes_{c}"][e][m]
                mes_index = meses[m]
                precip_col = f"precip_{mes_index}"
                df.loc[(df['CAMPAÑA'] == c) & (df['ID_ESTACION'] == e),␣
 ↪precip_col] = precip_mes
```

```
    # Rellenamos los valores de temp_mes en función de CAMPAÑA y ID_ESTACION
    for m in range(len(meses)):
        temp_col = f"t_mes_{meses[m]}"
        df[temp_col] = np.nan
    for c in range(16, 23):
        for e in range(20):
            for m in range(6):
                temp_mes = globals()[f"t_mes_{c}"][e][m]
                mes_index = meses[m]
                temp_col = f"t_mes_{mes_index}"
                df.loc[(df['CAMPAÑA'] == c) & (df['ID_ESTACION'] == e),␣
↪temp_col] = temp_mes
    for m in range(len(meses)):
        rhum_col = f"rhum_mes_{meses[m]}"
        df[rhum_col] = np.nan
    for c in range(16, 23):
        for e in range(20):
            for m in range(6):
                rhum_mes = globals()[f"rhum_mes_{c}"][e][m]
                mes_index = meses[m]
                rhum_col = f"rhum_mes_{mes_index}"
                df.loc[(df['CAMPAÑA'] == c) & (df['ID_ESTACION'] == e),␣
↪rhum_col] = rhum_mes
    return df
```

[43]:
```
metervariables(df_clust)
```

[43]:
```
     CAMPAÑA  ID_ESTACION  precip_enero  precip_febrero  precip_marzo  \
0         16            0          13.4            11.1          25.9
1         16            1          10.2            27.3          24.8
2         16            2          12.0             9.6          31.3
3         16            3           7.9            12.9          30.0
4         16            4           9.9            18.1          48.3
..       …           …           …              …           …
115       21           15          10.0             2.0          16.0
116       21           16          11.0             2.0          30.0
117       21           17          14.0             1.0          16.0
118       21           18          15.0             1.0          21.0
119       21           19          10.0             1.0          15.0

     precip_abril  precip_mayo  precip_junio  t_mes_enero  t_mes_febrero  \
0           118.8    23.944828      2.896552    10.733469      10.382577
1            67.1    38.482759     10.034483     8.581978       8.285212
2            84.2    12.613793      8.172414     9.765312      10.330454
3            65.3    32.175862      4.344828     9.553794       9.266618
4            85.3    26.937931      8.586207     9.475610       9.385944
..            …           …             …           …            …
```

```
115          47.0    32.000000     46.000000     6.187854       9.921610
116          60.0    38.000000     60.000000     6.671795      10.382712
117          59.0    24.000000     35.000000     5.048178       9.118629
118          95.0    43.000000     35.000000     6.975709      10.497914
119          45.0    35.000000     43.000000     7.542240      11.160507

     t_mes_marzo  t_mes_abril  t_mes_mayo  t_mes_junio  rhum_mes_enero  \
0      10.945479    13.823876   16.847880    22.626928       71.168606
1       9.016575    12.235955   15.570725    21.825666       78.102030
2      10.481096    13.848876   16.396990    22.428471       75.648850
3       9.963425    13.091433   16.354446    22.535063       77.028687
4       9.981507    13.190449   16.251436    22.440813       76.380785
..           …           …           …           …              …
115     9.475941    11.477361   17.062903    20.947917       67.990013
116     9.848118    11.864167   17.399731    21.345278       67.225371
117     8.831452    10.892361   16.519892    20.369306       71.430094
118     9.878629    11.783056   17.459946    21.222361       65.899190
119    10.625806    12.582778   18.180511    22.012222       66.894332

     rhum_mes_febrero  rhum_mes_marzo  rhum_mes_abril  rhum_mes_mayo  \
0           64.152269       64.085753       69.486376      65.571956
1           70.770425       68.398356       72.279073      67.085226
2           64.841874       65.896849       69.263764      67.112585
3           70.092679       68.068082       72.643961      67.588919
4           67.669985       66.099452       70.019522      65.744460
..                …               …               …              …
115         69.949180       72.709274       77.391389      59.184946
116         68.766319       73.329032       77.730417      59.583199
117         70.783159       70.402823       75.747500      57.661290
118         68.677645       73.788575       78.442222      59.723118
119         69.478539       73.519758       77.977500      59.674866

     rhum_mes_junio
0         57.439411
1         55.262693
2         58.328752
3         56.808415
4         54.754278
..              …
115       59.340139
116       59.383194
117       57.973194
118       60.423056
119       60.125694

[120 rows x 20 columns]
```

Creamos el dataframe de test. Vamos a usar las estaciones que aparecen en la campaña 22 en la

base de datos de TRAIN.

[44]:
```
df_BASETRAIN = pd.read_csv('UH_2023_TRAIN.txt', delimiter='|')
```

[45]:
```
#Creamos un dataframe con los datos sobre los que queremos estimar la␣
 ↪producción utilizando un procedimiento similar al anterior
estaciones_22=list(set(df_BASETRAIN.
 ↪loc[df_BASETRAIN['CAMPAÑA']==22,'ID_ESTACION']))
len(estaciones_22)
```

[45]:
```
16
```

[46]:
```
campaña = [22]*16
id_estacion = estaciones_22
data = {'CAMPAÑA': campaña, 'ID_ESTACION': id_estacion}
df_pred = pd.DataFrame(data)
metervariables(df_pred)
df_pred
```

[46]:

	CAMPAÑA	ID_ESTACION	precip_enero	precip_febrero	precip_marzo
0	22	2	1.0	6.0	133.0
1	22	3	1.0	7.0	114.0
2	22	5	1.0	7.0	108.0
3	22	6	2.0	11.0	207.0
4	22	7	1.0	7.0	110.0
5	22	8	0.0	8.0	100.0
6	22	9	1.0	6.0	116.0
7	22	10	0.0	9.0	174.0
8	22	11	1.0	5.0	115.0
9	22	12	2.0	7.0	109.0
10	22	13	0.0	7.0	112.0
11	22	14	1.0	10.0	113.0
12	22	15	1.0	7.0	110.0
13	22	16	1.0	8.0	101.0
14	22	18	1.0	7.0	113.0
15	22	19	1.0	8.0	109.0

	precip_abril	precip_mayo	precip_junio	t_mes_enero	t_mes_febrero
0	73.0	49.0	10.0	6.995833	10.230952
1	76.0	43.0	9.0	6.575806	9.637500
2	76.0	58.0	7.0	6.375806	9.547768
3	136.0	76.0	15.0	5.916398	9.082887
4	73.0	43.0	6.0	6.704839	9.816964
5	75.0	42.0	9.0	6.093548	9.231548
6	73.0	39.0	6.0	7.167204	10.319643
7	106.0	57.0	10.0	7.644758	10.769345
8	81.0	57.0	14.0	6.868952	10.069940
9	71.0	36.0	10.0	7.342742	10.398214

```
10          78.0         49.0           7.0     6.458065       9.518452
11          81.0         53.0          15.0     5.514247       8.623512
12          75.0         41.0           7.0     6.022581       9.121875
13          76.0         50.0          10.0     6.107527       9.267560
14          81.0         74.0           5.0     6.359946       9.542113
15          71.0         56.0          11.0     7.023118      10.172173

    t_mes_marzo  t_mes_abril  t_mes_mayo  t_mes_junio  rhum_mes_enero  \
0     10.685618    12.836667   19.709274    24.724200       67.347581
1      9.067608    11.557639   18.733468    24.257997       65.180376
2      9.157930    11.629583   18.696505    24.189847       66.743280
3      8.622446    10.988889   18.623253    24.313630       65.891129
4      9.442339    11.936528   19.027957    24.536857       66.344624
5      8.845699    11.278056   18.601075    24.195410       65.994892
6      9.703360    12.224722   19.172715    24.650070       65.914785
7     10.127957    12.585139   19.589247    24.895271       65.959677
8     10.308468    12.623889   19.661156    25.050209       67.368011
9      9.832930    12.344722   19.403091    24.936857       65.321371
10     9.281317    11.748889   18.984677    24.367594       65.645296
11     8.136290    10.604861   17.860349    23.465925       65.612366
12     8.705780    11.193889   18.382661    23.889708       65.824866
13     9.084946    11.529861   18.709274    24.133797       67.090054
14     9.070027    11.532222   18.600806    24.050070       66.509812
15     9.830108    12.321806   19.364516    24.868567       66.779032

    rhum_mes_febrero  rhum_mes_marzo  rhum_mes_abril  rhum_mes_mayo  \
0          63.702232       79.392742       65.484028      55.320968
1          61.775149       84.836694       69.003194      58.563306
2          62.667560       85.106989       69.396389      59.267339
3          61.544048       84.821505       69.971111      57.110753
4          62.702381       84.715995       68.871667      58.741667
5          62.245387       84.567473       69.087639      57.663441
6          61.848214       85.303629       69.331806      60.165995
7          61.819345       85.706586       69.838611      59.501613
8          63.660714       81.647849       66.796667      56.249731
9          61.879167       84.984005       69.057500      59.241532
10         62.488393       83.689516       67.701667      56.569355
11         61.973512       84.745027       69.021250      58.055914
12         62.283631       84.387231       68.530139      57.865323
13         63.364435       84.587231       68.908194      57.978763
14         62.339286       85.242070       69.530000      59.324328
15         62.843601       84.922581       69.131111      59.189247

    rhum_mes_junio
0        46.928512
1        46.939777
2        47.784701
```

```
3        44.599166
4        47.309040
5        45.664673
6        48.942837
7        48.811544
8        46.039777
9        47.610709
10       45.871766
11       46.052017
12       46.535744
13       46.742142
14       48.015716
15       47.706954
```

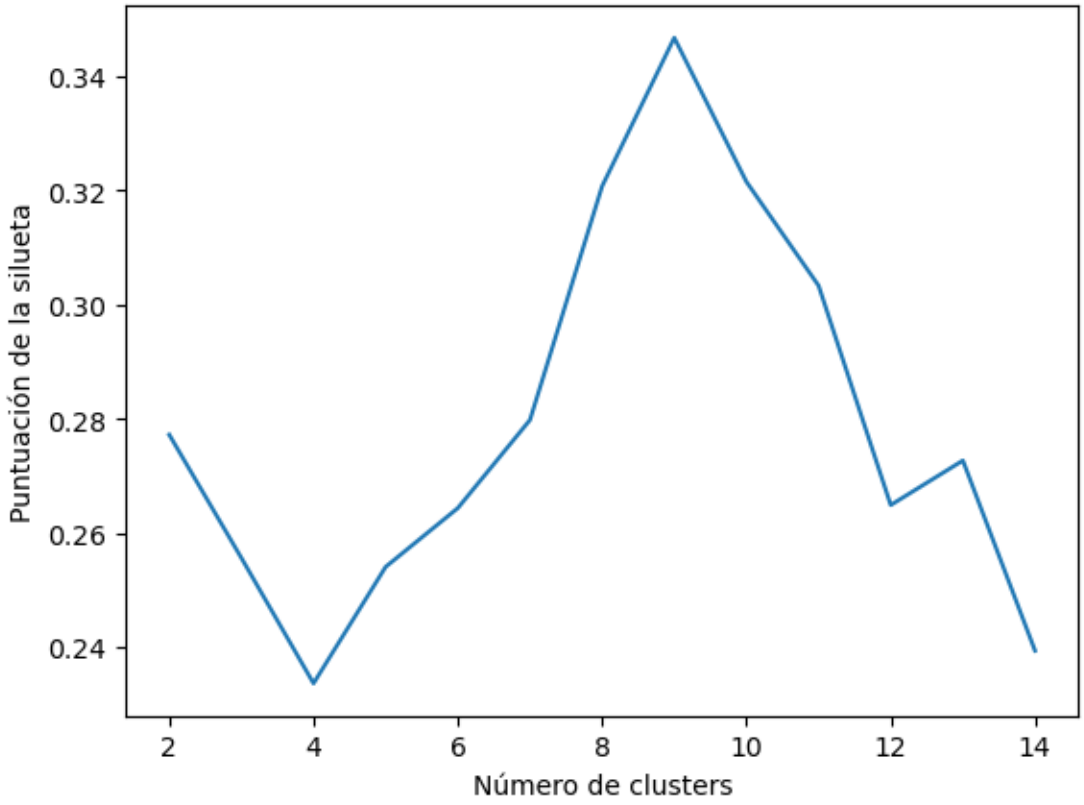
[47]:
```
#Renombramos por comodidad
df_train=df_clust
```

[48]:
```
#Cálculo de las distancias usando modelos kmeans
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import pairwise_distances
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score
from sklearn.manifold import TSNE

# Creamos un diccionario para almacenar las distancias
distances = {}

# Iteraramos sobre los años (recordemos que hacemos un modelo por año)
for year in range(16, 22):
    # Seleccionamos los datos de entrenamiento y prueba para el año específico␣
 ↪y el año 22
    train_data = df_train[df_train['CAMPAÑA'] == year]
    test_data = df_pred[df_pred['CAMPAÑA'] == 22]

    # Seleccionamos las columnas de interés para el cálculo de distancia
    dist_cols = ['precip_enero', 'precip_febrero', 'precip_marzo',␣
 ↪'precip_abril', 'precip_mayo', 'precip_junio', 't_mes_enero',␣
 ↪'t_mes_febrero', 't_mes_marzo', 't_mes_abril', 't_mes_mayo', 't_mes_junio']


    # Prepararamos los datos de entrenamiento y prueba normalizando
    scaler = StandardScaler()
    X_train = scaler.fit_transform(train_data[dist_cols])
    X_test = scaler.transform(test_data[dist_cols])

    # Estudiamos el número de clusters por el método de la silueta
```

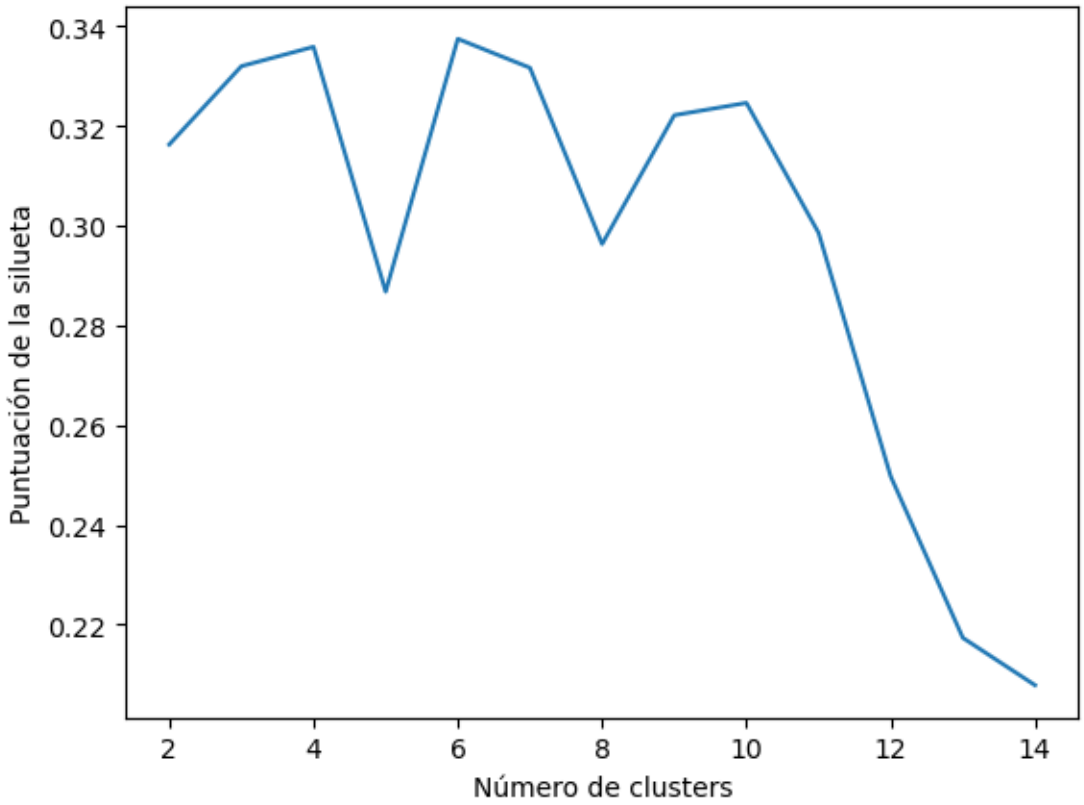
```
    range_n_clusters = range(2, 15)
    silhouette_scores = []
    for n_clusters in range_n_clusters:
        # Entrenamos
        clusterer = KMeans(n_clusters=n_clusters, random_state=42)
        cluster_labels = clusterer.fit_predict(X_train)
        # Calculamos la puntuación de la silueta promedio
        silhouette_avg = silhouette_score(X_train, cluster_labels)
        silhouette_scores.append(silhouette_avg)
        if n_clusters == 2:
            print("Puntuación para 2 clusters:", silhouette_avg)

    # Encontramos la puntuación máxima y el número de clusters correspondiente
    max_score = max(silhouette_scores)
    best_n_clusters = silhouette_scores.index(max_score) + 2 # añadimos 2␣
↪porque empezamos el rango en 2
    print("La puntuación máxima es:", max_score, "con", best_n_clusters,␣
↪"clusters")

    # Graficamos
    plt.plot(range_n_clusters, silhouette_scores)
    plt.xlabel('Número de clusters')
    plt.ylabel('Puntuación de la silueta')
    plt.show()


    #Elegimos 2 clusters. Nótese que necesitamos que todos los años tengan el␣
↪mismo número de clusters y es la máxima
    #puntuación en la mayoría de los modelos (y no es alejada de la máxima␣
↪puntuación del que no lo es).

    # Entrenamos el modelo KMeans
    kmeans = KMeans(n_clusters=2, random_state=42)
    kmeans.fit(X_train)

    # Predecimos las etiquetas de clúster para los datos de prueba
    test_labels = kmeans.predict(X_test)

    # Calculamos las distancias entre los datos de prueba y los centros de␣
↪clúster
    cluster_centers = kmeans.cluster_centers_
    distances[year] = {}

    for i in range(len(test_data)):
        row = test_data.loc[test_data.index[i]]
        station = row['ID_ESTACION']
```

```
        if station not in train_data['ID_ESTACION'].unique() or station not in␣
↪test_data['ID_ESTACION'].unique():
            continue
        distance = pairwise_distances([X_test[i]],␣
↪[cluster_centers[test_labels[i]]], metric='euclidean')[0][0]
        distances[year][station]=distance
```

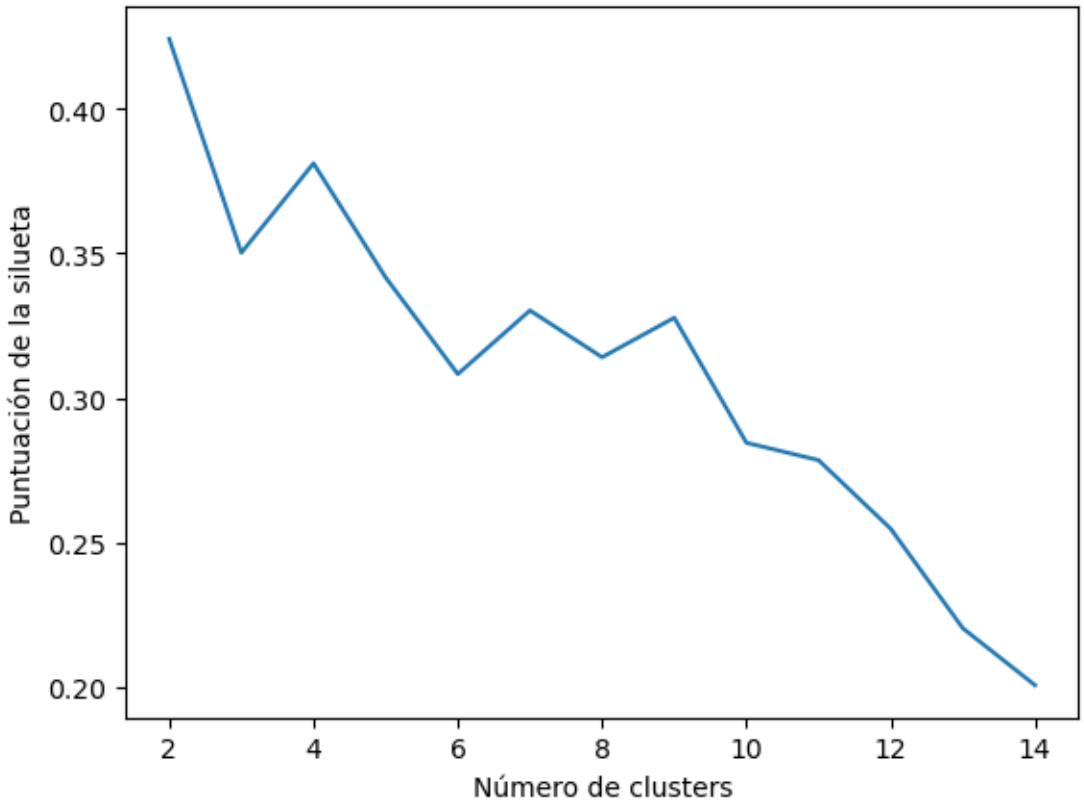
```
Puntuación para 2 clusters: 0.2772000481524276
La puntuación máxima es: 0.34673093389035897 con 9 clusters
```



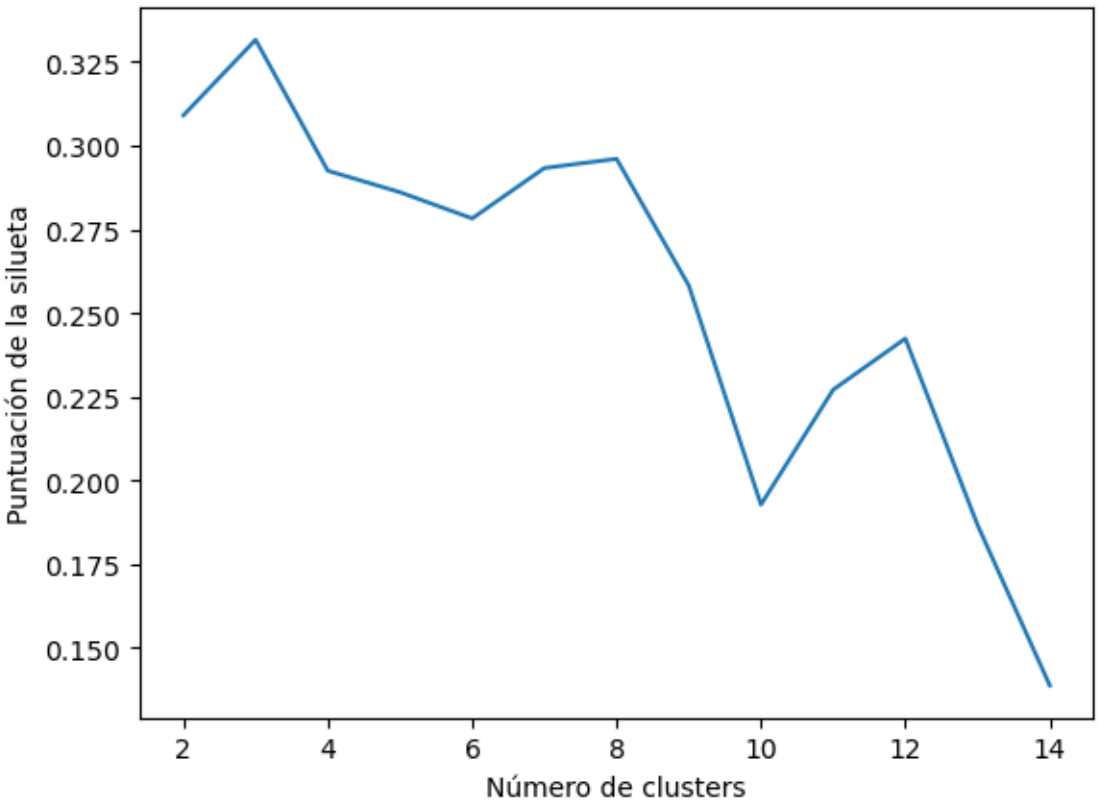

```
Puntuación para 2 clusters: 0.3162871938155161
La puntuación máxima es: 0.33748785877761656 con 6 clusters
```



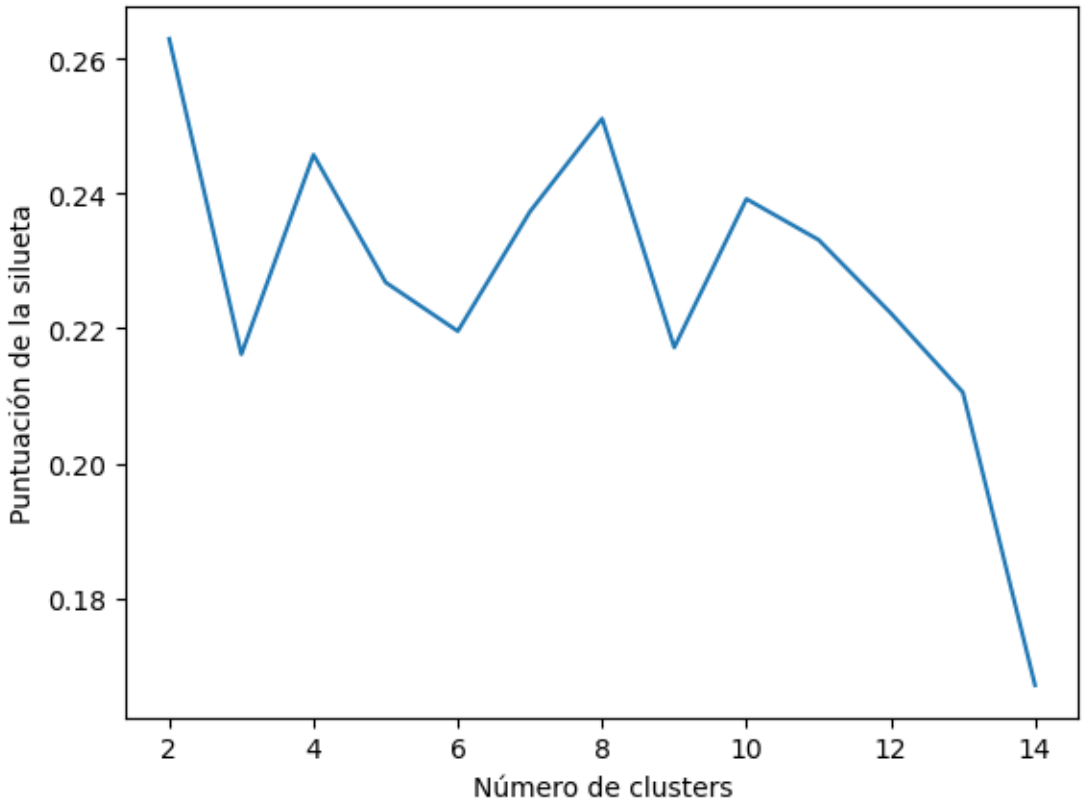

```
Puntuación para 2 clusters: 0.42423278074372084
La puntuación máxima es: 0.42423278074372084 con 2 clusters
```



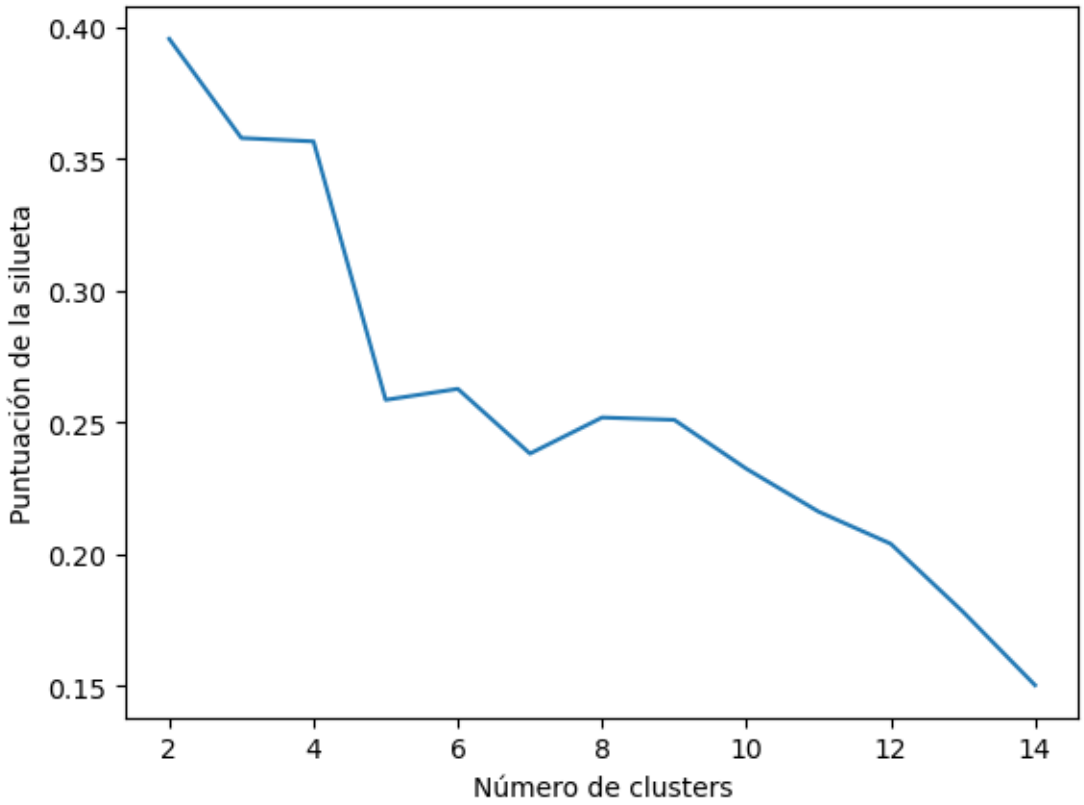

```
Puntuación para 2 clusters: 0.3090833453604511
La puntuación máxima es: 0.3316354468405625 con 3 clusters
```




```
Puntuación para 2 clusters: 0.26298900008473675
La puntuación máxima es: 0.26298900008473675 con 2 clusters
```




```
Puntuación para 2 clusters: 0.3953601059981465
La puntuación máxima es: 0.3953601059981465 con 2 clusters
```



Cabe destacar que, como el objetivo del clustering no era el de clasificar patrones meteorlógicos sino que consistía en compararlos para distintos años y ver qué años están más cercanos del año del que queremos predecir la producción (2022), una baja puntuación de silueta no indica que las distancias que obtengamos no sean fiables.

[49]:
```
# Creo diccionario invertido para ver los resultados más claros
distances_invertido = {}
for year, subdiccionario in distances.items():
    for estacion, valor in subdiccionario.items():
        if estacion not in distances_invertido:
            distances_invertido[estacion] = {}
        distances_invertido[estacion][year] = valor
```

[50]:
```
distances_invertido
```

[50]:
```
{2.0: {16: 24.1986225940076,
  17: 23.83210771793543,
  18: 21.753245029596673,
  19: 31.720731393357358,
  20: 17.064859802272284,
  21: 15.434276200757369},
```

```
3.0: {16: 20.131909398020866,
 17: 20.98699150688409,
 18: 18.460708690293067,
 19: 27.429777409772253,
 20: 18.66838247845296,
 21: 13.183496910541656},
5.0: {16: 19.359760977505104,
 17: 27.956387504395444,
 18: 17.74577177363968,
 19: 26.23949049993456,
 20: 18.655715635561453,
 21: 13.05493481985757},
6.0: {16: 37.86205653622091,
 17: 39.81517627277692,
 18: 32.624689987291546,
 19: 51.17980739151033,
 20: 30.028634738434643,
 21: 24.26866750926029},
7.0: {16: 19.87426294329744,
 17: 20.944533498053488,
 18: 18.481231188321946,
 19: 24.50911740277533,
 20: 18.836348136181723,
 21: 12.989340390433624},
8.0: {16: 18.06904526523752,
 17: 20.53820215461611,
 18: 16.803247961426038,
 19: 25.729917349002637,
 20: 20.985395597075495,
 21: 12.078372263712524},
9.0: {16: 20.829923860268014,
 17: 19.252125840581034,
 18: 19.53997148586427,
 19: 24.98608705575054,
 20: 16.791346582378257,
 21: 13.494892358553175},
10.0: {16: 31.661421761615447,
 17: 29.38879253057695,
 18: 28.061890717860283,
 19: 39.814625184317116,
 20: 24.593913298965603,
 21: 20.07006236384966},
11.0: {16: 21.891837595033582,
 17: 27.89098429432176,
 18: 19.914931660430568,
 19: 33.80039018374633,
 20: 15.40689152569788,
```

```
 21: 14.015097057707004},
12.0: {16: 19.894991551723685,
 17: 17.97181934037503,
 18: 18.81005671409308,
 19: 27.53439980671823,
 20: 19.190766712798244,
 21: 13.014948448121663},
13.0: {16: 20.27411221926527,
 17: 23.840053670225288,
 18: 18.51448225613617,
 19: 25.996746026246786,
 20: 18.803141637337827,
 21: 13.255931438561042},
14.0: {16: 19.793296856243458,
 17: 25.991381593455678,
 18: 17.137722227158108,
 19: 35.31482592055187,
 20: 25.581840682528174,
 21: 13.112129310987731},
15.0: {16: 19.175170140655545,
 17: 20.026890345710456,
 18: 17.47989310183366,
 19: 24.923486429318988,
 20: 18.554151357339936,
 21: 12.85452396031953},
16.0: {16: 18.141343058562207,
 17: 24.177074614376586,
 18: 16.850123391361596,
 19: 27.409275788711245,
 20: 20.88710505335131,
 21: 12.148885565990353},
18.0: {16: 20.522802403457305,
 17: 35.76119562339416,
 18: 18.288855449425913,
 19: 27.717101830672046,
 20: 18.78077753754534,
 21: 14.488589516826597},
19.0: {16: 20.486691143327377,
 17: 27.053882034896223,
 18: 18.730471961774715,
 19: 30.315694453623717,
 20: 21.337227332093033,
 21: 13.37762505392678}}
```

Como podemos ver, los resultados nos indican que el año más cercano a 2022 es 2021, seguido de 2020.

[51]:
```
# Guardamos los datos de interés para que puedan ser utilizados en otros␣
 ↪notebooks
import pickle
with open('precip_mes_15.pkl', 'wb') as f:
    pickle.dump(precip_mes_15, f)
with open('precip_mes_16.pkl', 'wb') as f:
    pickle.dump(precip_mes_16, f)
with open('precip_mes_17.pkl', 'wb') as f:
    pickle.dump(precip_mes_17, f)
with open('precip_mes_18.pkl', 'wb') as f:
    pickle.dump(precip_mes_18, f)
with open('precip_mes_19.pkl', 'wb') as f:
    pickle.dump(precip_mes_19, f)
with open('precip_mes_20.pkl', 'wb') as f:
    pickle.dump(precip_mes_20, f)
with open('precip_mes_21.pkl', 'wb') as f:
    pickle.dump(precip_mes_21, f)
with open('precip_mes_22.pkl', 'wb') as f:
    pickle.dump(precip_mes_22, f)
```

[52]:
```
with open('t_mes_15.pkl', 'wb') as f:
    pickle.dump(t_mes_16, f)
with open('t_mes_16.pkl', 'wb') as f:
    pickle.dump(t_mes_16, f)
with open('t_mes_17.pkl', 'wb') as f:
    pickle.dump(t_mes_17, f)
with open('t_mes_18.pkl', 'wb') as f:
    pickle.dump(t_mes_18, f)
with open('t_mes_19.pkl', 'wb') as f:
    pickle.dump(t_mes_19, f)
with open('t_mes_20.pkl', 'wb') as f:
    pickle.dump(t_mes_20, f)
with open('t_mes_21.pkl', 'wb') as f:
    pickle.dump(t_mes_21, f)
with open('t_mes_22.pkl', 'wb') as f:
    pickle.dump(t_mes_22, f)
```

[53]:
```
with open('windspeed_mes_15.pkl', 'wb') as f:
    pickle.dump(windspeed_mes_15, f)
with open('windspeed_mes_16.pkl', 'wb') as f:
    pickle.dump(windspeed_mes_16, f)
with open('windspeed_mes_17.pkl', 'wb') as f:
    pickle.dump(windspeed_mes_17, f)
with open('windspeed_mes_18.pkl', 'wb') as f:
    pickle.dump(windspeed_mes_18, f)
with open('windspeed_mes_19.pkl', 'wb') as f:
    pickle.dump(windspeed_mes_19, f)
```

```
with open('windspeed_mes_20.pkl', 'wb') as f:
    pickle.dump(windspeed_mes_20, f)
with open('windspeed_mes_21.pkl', 'wb') as f:
    pickle.dump(windspeed_mes_21, f)
with open('windspeed_mes_22.pkl', 'wb') as f:
    pickle.dump(windspeed_mes_22, f)
```

[54]:
```
with open('rhum_mes_15.pkl', 'wb') as f:
    pickle.dump(rhum_mes_15, f)
with open('rhum_mes_16.pkl', 'wb') as f:
    pickle.dump(rhum_mes_16, f)
with open('rhum_mes_17.pkl', 'wb') as f:
    pickle.dump(rhum_mes_17, f)
with open('rhum_mes_18.pkl', 'wb') as f:
    pickle.dump(rhum_mes_18, f)
with open('rhum_mes_19.pkl', 'wb') as f:
    pickle.dump(rhum_mes_19, f)
with open('rhum_mes_20.pkl', 'wb') as f:
    pickle.dump(rhum_mes_20, f)
with open('rhum_mes_21.pkl', 'wb') as f:
    pickle.dump(rhum_mes_21, f)
with open('rhum_mes_22.pkl', 'wb') as f:
    pickle.dump(rhum_mes_22, f)
```

[55]:
```
with open('distances_invertido.pickle', 'wb') as handle:
    pickle.dump(distances_invertido, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

[ ]:
```
```

[ ]:
```
```