LIMPIEZA Y PREPROCESAMIENTO DE DATOS

Librerías a emplear:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib
from matplotlib.pyplot import figure
import seaborn as sns
%matplotlib inline
matplotlib.rcParams['figure.figsize'] = (12,8)
```

Lectura de los datos y primer análisis

Leemos los datos del csv estando este en el mismo directorio que el código.

In [2]:
```
df=pd.read_csv('9898-19430301-20221204.csv',sep=';')
df_original=pd.read_csv('9898-19430301-20221204.csv',sep=';')
```

Vamos a analizar el data frame que tenemos.

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

Out[3]:

	FECHA	INDICATIVO	NOMBRE	PROVINCIA	ALTITUD	TMEDIA	PRECIPITACION	TMIN	HORATMIN	TMAX
0	1943-03-01	9898	HUESCA AEROPUERTO	HUESCA	546	9.8	NaN	5.0	NaN	14.6
1	1943-03-02	9898	HUESCA AEROPUERTO	HUESCA	546	11.1	NaN	5.2	NaN	17.0
2	1943-03-03	9898	HUESCA AEROPUERTO	HUESCA	546	11.9	NaN	5.6	NaN	18.2
3	1943-03-04	9898	HUESCA AEROPUERTO	HUESCA	546	11.0	NaN	7.9	NaN	14.0
4	1943-03-05	9898	HUESCA AEROPUERTO	HUESCA	546	8.5	NaN	5.4	NaN	11.6

In [4]:
```
df.tail(2)
```

Out[4]:

	FECHA	INDICATIVO	NOMBRE	PROVINCIA	ALTITUD	TMEDIA	PRECIPITACION	TMIN	HORATMIN	T
28614	2022-12-03	9898	HUESCA, AEROPUERTO	HUESCA	546	1.4	0.0	-0.3	00:00	
28615	2022-12-04	9898	HUESCA, AEROPUERTO	HUESCA	546	3.8	0.0	0.7	06:40	

In [5]:
```
print(df.shape)
```

```
(28616, 20)
```

In [6]:
```
print(df.dtypes)
```

```
FECHA             object
INDICATIVO         int64
NOMBRE            object
PROVINCIA         object
ALTITUD            int64
TMEDIA           float64
PRECIPITACION     object
TMIN             float64
HORATMIN          object
TMAX             float64
HORATMAX          object
DIR              float64
VELMEDIA         float64
RACHA            float64
HORARACHA         object
SOL              float64
PRESMAX          float64
HORAPRESMAX       object
PRESMIN          float64
HORAPRESMIN       object
dtype: object
```

Vemos como aquellos campos que aceptan 'varios' se han cargado como tipo objeto. Lo primero que vamos a hacer es localizar esos valores y modificarlos. En primer lugar vamos a arreglar la columna de precipitaciones. Sabemos que cuando estas son muy leves se les da el valor de 'Ip' lo cual es un problema. Vamos a modificar esos valores a 0 y vamos a cambiar el tipo a float.

In [7]:
```
'Ip' in df['PRECIPITACION'].values
for index,dato in enumerate(df['PRECIPITACION']):
  if dato == 'Ip':
    df['PRECIPITACION'].iloc[index]=0.1
```

```
C:\Users\dmrbe\AppData\Local\Temp\ipykernel_15728\269649687.py:4: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df['PRECIPITACION'].iloc[index]=0.1
```

In [8]:
```
df['PRECIPITACION']=df['PRECIPITACION'].astype('float')
```

Se va a modificar el resto de atributos, en este momento vamos a transformar los vampos que sean 'Varias' a NaN y más adelante gestionaremos estos campos junto a los demás datos faltantes.

In [9]:
```
df['PRECIPITACION'].keys
```

Out[9]:
```
<bound method Series.keys of 0        NaN
1        NaN
2        NaN
3        NaN
4        NaN
        ...
28611    0.0
28612    0.0
28613    0.0
28614    0.0
28615    0.0
Name: PRECIPITACION, Length: 28616, dtype: float64>
```

In [10]:
```
for column in df.columns[7 : ]:
  for index,dato in df[column].items():
    if dato == 'Varias':
```

```
        #print("Se ha remplazado el valor numero {} del atributo {} con el valor de {} por
        df.loc[index,column]=np.NaN
```

In [11]:
```
df.tail(3)
```

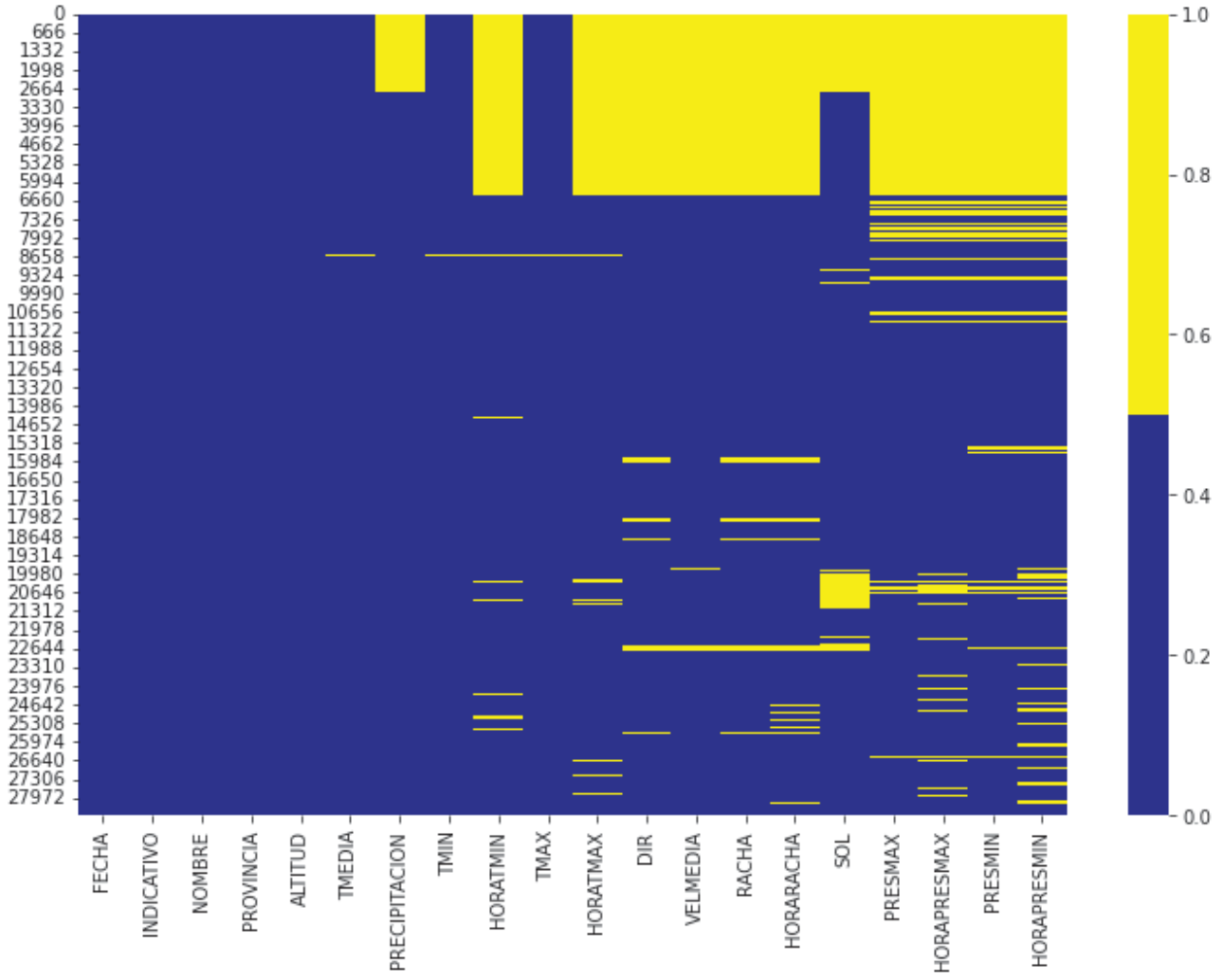
Out[11]:

	FECHA	INDICATIVO	NOMBRE	PROVINCIA	ALTITUD	TMEDIA	PRECIPITACION	TMIN	HORATMIN	T
28613	2022-12-02	9898	HUESCA, AEROPUERTO	HUESCA	546	3.0	0.0	-0.3	23:50	
28614	2022-12-03	9898	HUESCA, AEROPUERTO	HUESCA	546	1.4	0.0	-0.3	00:00	
28615	2022-12-04	9898	HUESCA, AEROPUERTO	HUESCA	546	3.8	0.0	0.7	06:40	

Analisis de valores nulos

In [12]:
```
cols = df.columns # first 30 columns
colours = ['#000099', '#ffff00'] # specify the colours - yellow is missing. blue is not
ylabels=df['FECHA'].values
ylabels=' '.join(ylabels)
sns.heatmap(df[cols].isnull(), cmap=sns.color_palette(colours))
```

Out[12]: <AxesSubplot:>



In [13]:
```
df.iloc[6500]
```

Out[13]:
```
FECHA                    1961-02-14
INDICATIVO                     9898
NOMBRE            HUESCA AEROPUERTO
PROVINCIA                    HUESCA
ALTITUD                         546
TMEDIA                          9.0
PRECIPITACION                   0.0
TMIN                            4.0
HORATMIN                      07:20
TMAX                           14.0
HORATMAX                      15:15
DIR                             9.0
VELMEDIA                        2.2
RACHA                           6.1
HORARACHA                     15:00
SOL                             9.0
PRESMAX                       963.2
HORAPRESMAX                      24
PRESMIN                       959.1
HORAPRESMIN                      05
Name: 6500, dtype: object
```

In [14]:
```
for col in df.columns:
    pct_missing = np.mean(df[col].isnull())
    print('{} - {}%'.format(col, round(pct_missing*100)))
```

```
FECHA - 0%
INDICATIVO - 0%
NOMBRE - 0%
PROVINCIA - 0%
ALTITUD - 0%
TMEDIA - 0%
PRECIPITACION - 10%
TMIN - 0%
HORATMIN - 24%
TMAX - 0%
HORATMAX - 24%
DIR - 27%
VELMEDIA - 23%
RACHA - 27%
HORARACHA - 28%
SOL - 17%
PRESMAX - 27%
HORAPRESMAX - 30%
PRESMIN - 27%
HORAPRESMIN - 32%
```

Vamos a seleccionar un subset donde todos los datos tengan menos de un 10% de datos perdidos. En primer lugar vamos a eliminar los registros correspondientes al año 2006 ya que la estación metereológica presenta algún problema y las medidas de sol y presión no son buenas. Para hacer esto puede ser interesante obtener los campos día, mes, año y las fechas anuales en formato datetime.

Fechas y horas

Vamos a canviar el tipo de dato de FECHA de "objetc" a "datetime", además vamos a añadir dia, mes, año, dia de la semana y semana del año como nuevas columnas del data frame.

In [15]:
```
df['FECHA_dt']=pd.to_datetime(df['FECHA'],format="%Y-%m-%d",exact=True)
```

In [16]:
```
df['YEAR'] = df['FECHA_dt'].dt.year
df['MONTH'] = df['FECHA_dt'].dt.month
```

```
df['DAY'] = df['FECHA_dt'].dt.day
df['WEEK']=df['FECHA_dt'].dt.isocalendar()['week']
```

(Estas celdas se ejecutarán más adelante ya que se van a aplicar estrategias para completar datos nulos y las tendríamos que volver a ejecutar. Se incluye aquí como referencia) Queremos ahora ajustar las horas de presión mínima para que sean del tipo Data Time, no obstante tenemos varios problemas. El primero es que el formato de fecha no es correcto, se tiene valor de 24 en vez de 00:00. Después el formato en si.

In [17]:
```
'''
for index,horas in df['HORAPRESMIN'].items():
  if str(horas) != 'nan':
    if int(horas)==24:
      df['HORAPRESMIN'].iloc[index]= '00:00'
    else:
      df['HORAPRESMIN'].iloc[index]= horas+':00'
'''
```

Out[17]:
```
"\nfor index,horas in df['HORAPRESMIN'].items():\n  if str(horas) != 'nan':\n    if int
(horas)==24:\n      df['HORAPRESMIN'].iloc[index]= '00:00'\n    else:\n      df['HORAPRE
SMIN'].iloc[index]= horas+':00'\n"
```

In [18]:
```
'''
for index,horas in df['HORAPRESMAX'].items():
  if str(horas) != 'nan':
    if int(horas)==24:
      df['HORAPRESMAX'].iloc[index]= '00:00'
    else:
      df['HORAPRESMAX'].iloc[index]= horas+':00'
'''
```

Out[18]:
```
"\nfor index,horas in df['HORAPRESMAX'].items():\n  if str(horas) != 'nan':\n    if int
(horas)==24:\n      df['HORAPRESMAX'].iloc[index]= '00:00'\n    else:\n      df['HORAPRE
SMAX'].iloc[index]= horas+':00'\n"
```

In [19]:
```
'''df['HORAPRESMIN_2']=pd.to_datetime(df['FECHA']+' '+df['HORAPRESMIN']+':00',infer_date
df['HORAPRESMIN_2']
'''
```

Out[19]:
```
"df['HORAPRESMIN_2']=pd.to_datetime(df['FECHA']+' '+df['HORAPRESMIN']+':00',infer_dateti
me_format=True)\ndf['HORAPRESMIN_2']\n"
```

In [20]:
```
'''df['HORAPRESMAX_2']=pd.to_datetime(df['FECHA']+' '+df['HORAPRESMIN']+':00',infer_date
```

Out[20]:
```
"df['HORAPRESMAX_2']=pd.to_datetime(df['FECHA']+' '+df['HORAPRESMIN']+':00',infer_dateti
me_format=True)"
```

Haremos lo equivalente con los campos HORATMIN y HORATMAX, el campo HORARACHA se eliminará más adelante ya que saber a qué hora del día se produce la máxima racha de viento sin datos de menor frecuencia no aporta valor dado el carácter aleatorio del viento.

In [21]:
```
'''df['HORATMIN_2']=pd.to_datetime(df['FECHA']+' '+df['HORATMIN']+':00',infer_datetime_f
df['HORATMAX_2']=pd.to_datetime(df['FECHA']+' '+df['HORATMAX']+':00',infer_datetime_form
```

Out[21]:
```
"df['HORATMIN_2']=pd.to_datetime(df['FECHA']+' '+df['HORATMIN']+':00',infer_datetime_for
mat=True)\ndf['HORATMAX_2']=pd.to_datetime(df['FECHA']+' '+df['HORATMAX']+':00',infer_da
tetime_format=True)"
```

Elección de un sub set de datos

In [22]:
```
missingByYear={}
missingInYear={}
dfmv=pd.DataFrame(index=[df['YEAR'].unique()],columns=['FECHA', 'INDICATIVO', 'NOMBRE',
```

In [23]:
```
for year in df['YEAR'].unique():
  dfs=df[df['YEAR'] == year]
  missingByYear[year]=np.mean(dfs.isnull())
  for col in dfs.columns:
    pct_missing = np.mean(dfs[col].isnull())
    dfmv.at[year,col]=pct_missing

dfmv.to_csv('miss')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:3438: FutureWarnin
g: In a future version, DataFrame.mean(axis=None) will return a scalar mean over the ent
ire DataFrame. To retain the old behavior, use 'frame.mean(axis=0)' or just 'frame.mean
()'
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```

In [24]:
```
df2=df.copy(deep=True)
df_1961=df[df['YEAR']>=1961]
```

In [25]:
```
df_1961=df_1961[df_1961['YEAR']!=2006]
```

In [26]:
```
df_1961['YEAR'].unique()
```

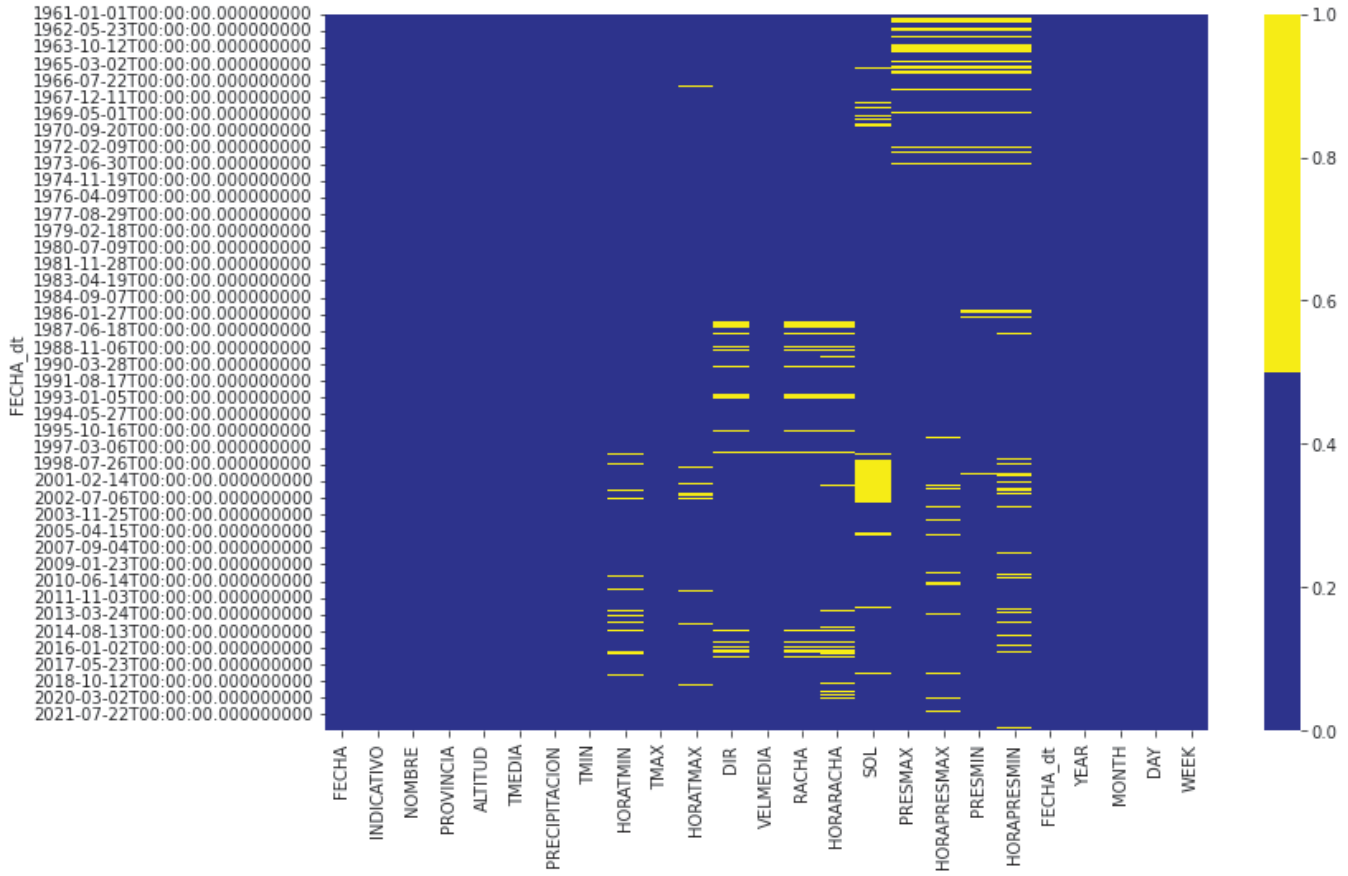
Out[26]:
```
array([1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971,
       1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982,
       1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993,
       1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004,
       2005, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016,
       2017, 2018, 2019, 2020, 2021, 2022], dtype=int64)
```

In [27]:
```
df=df_1961.copy(deep=True)
```

In [28]:
```
dfaux=df.set_index('FECHA_dt',drop=False)
```

In [29]:
```
cols = dfaux.columns # first 30 columns
colours = ['#000099', '#ffff00'] # specify the colours - yellow is missing. blue is not
sns.heatmap(dfaux[cols].isnull(), cmap=sns.color_palette(colours))
```

Out[29]:
```
<AxesSubplot:ylabel='FECHA_dt'>
```



Vemos cómo prescindir de estos campos mejora los datos, pero sigue habiendo una gran cantidad de NaN values. Estos aparecen en los campos correspondientes a las horas, sol, racha, y presiones máximas y mínimas.

Valores NaN

Ahora vamos a ver qué política se emplea para todos estos valores faltantes.
Dado que estamos en un problema donde la estacionalidad de los datos es muy marcada, para los campos de "HORAS" se va a emplear la hora que sea la moda estadística de los años anteriores ese mismo día. (En la entrega 1 se propone usar el valor anterior, pero esto creo que da mejores resultados y como no es una solución más sencilla que la planteada optó por esta.) Para los campos de presión, el campo de horas de sol y velocidad media se va a emplear la media de los valores de los años anteriores en el mismo día del mismo mes, para mantener la estacionalidad.

Los datos correspondientes a Dir y Racha se van a eliminar. Esto se debe a que en Aragón, zona geográfica donde se registran estos datos, se produce un fenómeno meteorológico denominado Cierzo. El cierzo es un tipo de viento racheado y de dirección cambiante. Por lo que tener únicamente los valores máximos y la dirección de estas rachas no aporta una información relevante relativa a la temperatura, o no por lo menos sin contar con datos de mayor frecuencia. La velocidad media se va a mantener.

In [30]:
```
df.head()
```

Out[30]:

	FECHA	INDICATIVO	NOMBRE	PROVINCIA	ALTITUD	TMEDIA	PRECIPITACION	TMIN	HORATMIN	TM
6456	1961-01-01	9898	HUESCA AEROPUERTO	HUESCA	546	6.4	0.0	3.4	05:50	
6457	1961-01-02	9898	HUESCA AEROPUERTO	HUESCA	546	5.5	5.6	1.6	00:20	

6458	1961-01-03	9898	HUESCA AEROPUERTO	HUESCA	546	6.9	4.0	4.4	05:00
6459	1961-01-04	9898	HUESCA AEROPUERTO	HUESCA	546	4.2	0.0	1.6	03:40
6460	1961-01-05	9898	HUESCA AEROPUERTO	HUESCA	546	4.1	0.0	0.2	05:10

5 rows × 25 columns

In [31]:
```
#df[df.MONTH.isin([1]) & df.DAY.isin([1])]
```

In [32]:
```
dfNaN=df.copy(deep=True)
```

In [33]:
```
from random import randint

def calculoMedias(df:pd.DataFrame,dia:int,mes:int,col:str)->float:

  dfaux=df[df.MONTH.isin([mes]) & df.DAY.isin([dia])]

  media= dfaux[col].median(skipna=True)
  return media
```

In [34]:
```
def calculoModa(df:pd.DataFrame,dia:int,mes:int,col:str)->float:

  dfaux=df[df.MONTH.isin([mes]) & df.DAY.isin([dia])]
  media= dfaux[col].mode(dropna=True)[0]
  return media
```

In [35]:
```
for col in df.columns.values:
  missing = df[col].isnull()
  num_missing = np.sum(missing)
  if num_missing >0:
    dfaux=df[df[col].isnull()]
    for index,valores in dfaux[col].items():
      if col in ['SOL','VELMEDIA','PRESMAX','PRESMIN']:
        dfNaN.loc[index,col]=calculoMedias(df,df['DAY'].loc[index],df['MONTH'].loc[index
        salida=calculoMedias(df,df['DAY'].loc[index],df['MONTH'].loc[index],col)
        print('insertado en el indice {}, de la columna: {}, el valor de: {}'.format(ind
      else:
        dfNaN.loc[index,col]=calculoModa(df,df['DAY'].loc[index],df['MONTH'].loc[index],
        salida=calculoModa(df,df['DAY'].loc[index],df['MONTH'].loc[index],col)
        print('insertado en el indice {}, de la columna: {}, el valor de: {}'.format(ind
```

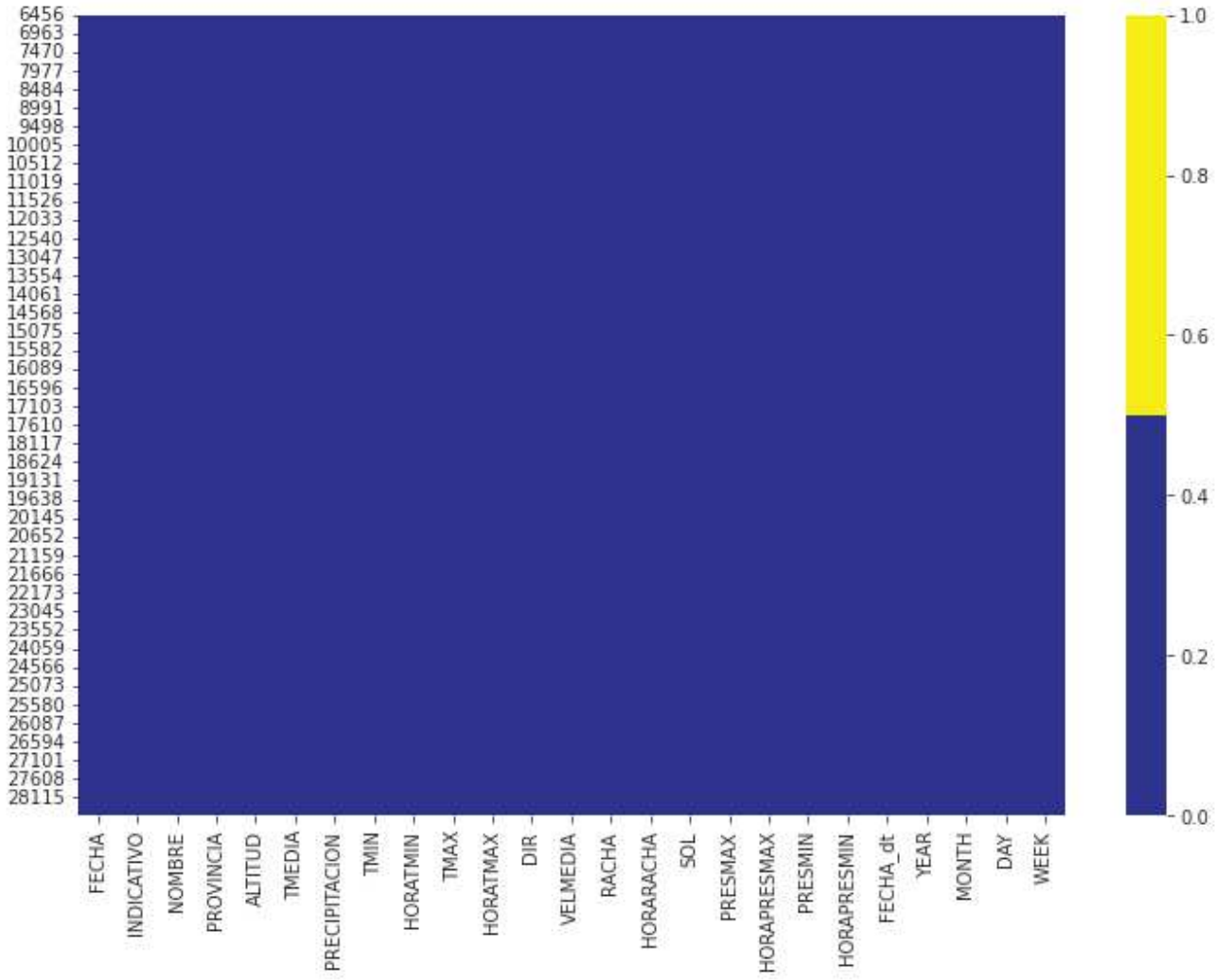
```
insertado en el indice 7922, de la columna: TMEDIA, el valor de: 5.6
insertado en el indice 8029, de la columna: TMEDIA, el valor de: 9.9
insertado en el indice 8042, de la columna: TMEDIA, el valor de: 14.4
insertado en el indice 8189, de la columna: TMEDIA, el valor de: 14.6
insertado en el indice 8260, de la columna: TMEDIA, el valor de: 7.6
insertado en el indice 8277, de la columna: TMEDIA, el valor de: 5.1
insertado en el indice 8398, de la columna: TMEDIA, el valor de: 11.6
insertado en el indice 8453, de la columna: TMEDIA, el valor de: 23.0
insertado en el indice 8523, de la columna: TMEDIA, el valor de: 22.4
insertado en el indice 8617, de la columna: TMEDIA, el valor de: 4.6
insertado en el indice 8619, de la columna: TMEDIA, el valor de: 6.1
insertado en el indice 8622, de la columna: TMEDIA, el valor de: 4.9
insertado en el indice 8637, de la columna: TMEDIA, el valor de: 3.5
insertado en el indice 8646, de la columna: TMEDIA, el valor de: 7.2
insertado en el indice 9539, de la columna: TMEDIA, el valor de: 17.6
insertado en el indice 12537, de la columna: TMEDIA, el valor de: 21.8
insertado en el indice 12542, de la columna: TMEDIA, el valor de: 17.0
insertado en el indice 15686, de la columna: TMEDIA, el valor de: 13.4
```

```
insertado en el indice 28054, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28060, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28061, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28067, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28070, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28079, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28084, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28094, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28107, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28110, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28126, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28127, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28129, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28154, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28160, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28164, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28174, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28178, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28186, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28190, de la columna: HORAPRESMIN, el valor de: 16
insertado en el indice 28194, de la columna: HORAPRESMIN, el valor de: 16
insertado en el indice 28208, de la columna: HORAPRESMIN, el valor de: 00
insertado en el indice 28209, de la columna: HORAPRESMIN, el valor de: 00
insertado en el indice 28220, de la columna: HORAPRESMIN, el valor de: 24
insertado en el indice 28234, de la columna: HORAPRESMIN, el valor de: 24
insertado en el indice 28238, de la columna: HORAPRESMIN, el valor de: 00
insertado en el indice 28240, de la columna: HORAPRESMIN, el valor de: 24
insertado en el indice 28243, de la columna: HORAPRESMIN, el valor de: 00
insertado en el indice 28256, de la columna: HORAPRESMIN, el valor de: 00
insertado en el indice 28276, de la columna: HORAPRESMIN, el valor de: 24
insertado en el indice 28281, de la columna: HORAPRESMIN, el valor de: 00
insertado en el indice 28295, de la columna: HORAPRESMIN, el valor de: 24
insertado en el indice 28297, de la columna: HORAPRESMIN, el valor de: 00
insertado en el indice 28299, de la columna: HORAPRESMIN, el valor de: 24
insertado en el indice 28311, de la columna: HORAPRESMIN, el valor de: 24
insertado en el indice 28313, de la columna: HORAPRESMIN, el valor de: 24
insertado en el indice 28318, de la columna: HORAPRESMIN, el valor de: 24
insertado en el indice 28320, de la columna: HORAPRESMIN, el valor de: 24
insertado en el indice 28325, de la columna: HORAPRESMIN, el valor de: 00
insertado en el indice 28330, de la columna: HORAPRESMIN, el valor de: 00
insertado en el indice 28332, de la columna: HORAPRESMIN, el valor de: 24
insertado en el indice 28358, de la columna: HORAPRESMIN, el valor de: 00
insertado en el indice 28362, de la columna: HORAPRESMIN, el valor de: 00
insertado en el indice 28363, de la columna: HORAPRESMIN, el valor de: 24
insertado en el indice 28368, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28390, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28405, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28427, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28442, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28472, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28476, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28485, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28512, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28519, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28523, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28529, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28535, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28540, de la columna: HORAPRESMIN, el valor de: 18
insertado en el indice 28552, de la columna: HORAPRESMIN, el valor de: 17
insertado en el indice 28558, de la columna: HORAPRESMIN, el valor de: 16
insertado en el indice 28582, de la columna: HORAPRESMIN, el valor de: 00
insertado en el indice 28605, de la columna: HORAPRESMIN, el valor de: 24
```

In [36]:
```
cols = dfNaN.columns # first 30 columns
colours = ['#000099', '#ffff00'] # specify the colours - yellow is missing. blue is not
```

```
sns.heatmap(dfNaN[cols].isnull(), cmap=sns.color_palette(colours),vmin=0,vmax=1)
```

Out[36]: <AxesSubplot:>



In [37]:
```
df=dfNaN.copy(deep=True)
```

Vemos cómo se han completado todos los valores NaN. Ahora vamos a ejecutar las celdas de Fechas y Horas que no habíamos ejecutados antes y que nos añadirán los campos de las horas con formato ISO. En primer lugar hemos de cambiar los calores de 24 por 00, además de añadir los minutos a todos los campos como ':00'

In [38]:
```
for index,horas in df['HORAPRESMIN'].items():
  if str(horas) != 'nan':
    if int(horas)==24:
      df.loc[index,'HORAPRESMIN']= '00:00'
    else:
      df.loc[index,'HORAPRESMIN']= horas+':00'
```

In [39]:
```
for index,horas in df['HORAPRESMAX'].items():
  if str(horas) != 'nan':
    if int(horas)==24:
      df.loc[index,'HORAPRESMIN']= '00:00'
    else:
      df.loc[index,'HORAPRESMIN']= horas+':00'
```

In [40]:
```
df['HORAPRESMIN_2']=pd.to_datetime(df['FECHA']+' '+df['HORAPRESMIN']+':00',infer_datetim
```

```
df['HORAPRESMAX_2']=pd.to_datetime(df['FECHA']+' '+df['HORAPRESMIN']+':00',infer_datetim
```

In [41]:
```
df['HORATMAX']
```

Out[41]:
```
6456     13:25
6457     14:00
6458     15:55
6459     13:10
6460     15:50
         ...
28611    13:10
28612    15:00
28613    12:00
28614    14:50
28615    13:00
Name: HORATMAX, Length: 21795, dtype: object
```

In [42]:
```
df2['HORATMAX'].to_csv('micsv')
```

Haremos lo equivalente con los campos HORATMIN y HORATMAX, el campo HORARACHA se eliminará más adelante ya que saber a qué hora del día se produce la máxima racha de viento sin datos de menor frecuencia no aporta valor dado el carácter aleatorio del viento.

In [43]:
```
df['HORATMIN_2']=pd.to_datetime(df['FECHA']+' '+df['HORATMIN']+':00',infer_datetime_form
```

In [44]:
```
df['HORATMAX_2']=pd.to_datetime(df['FECHA']+' '+df['HORATMAX']+':00',infer_datetime_form
```

Eliminación de columnas

Se van a eliminar aquellas columnas que no aportan valor, en este caso son "INDICATIVO", "NOMBRE", "PROVINCIA", "ALTITUD" y "HORARACHA". Como se ha comentado anteriormente RACHA y DIR también se pueden eliminar

In [45]:
```
dfaux=df.copy(deep=True)
```

In [46]:
```
df=df.drop(["INDICATIVO", "NOMBRE", "PROVINCIA", "ALTITUD", "HORARACHA","RACHA","DIR" ],
```

In [47]:
```
df.head()
```

Out[47]:

	FECHA	TMEDIA	PRECIPITACION	TMIN	HORATMIN	TMAX	HORATMAX	VELMEDIA	SOL	PRESMAX	...
6456	1961-01-01	6.4	0.0	3.4	05:50	9.4	13:25	3.9	4.9	954.0	...
6457	1961-01-02	5.5	5.6	1.6	00:20	9.4	14:00	3.1	4.2	955.6	...
6458	1961-01-03	6.9	4.0	4.4	05:00	9.4	15:55	7.8	2.4	947.1	...
6459	1961-01-04	4.2	0.0	1.6	03:40	6.8	13:10	10.6	3.7	948.4	...
6460	1961-01-05	4.1	0.0	0.2	05:10	8.0	15:50	4.7	8.2	957.9	...

5 rows × 22 columns

In [48]:
```
df.shape
```

Out[48]:
```
(21795, 22)
```

In [49]:
```
print(df.columns.values)
```

```
['FECHA' 'TMEDIA' 'PRECIPITACION' 'TMIN' 'HORATMIN' 'TMAX' 'HORATMAX'
 'VELMEDIA' 'SOL' 'PRESMAX' 'HORAPRESMAX' 'PRESMIN' 'HORAPRESMIN'
 'FECHA_dt' 'YEAR' 'MONTH' 'DAY' 'WEEK' 'HORAPRESMIN_2' 'HORAPRESMAX_2'
 'HORATMIN_2' 'HORATMAX_2']
```

Adición de los campos gradP y temperaturas en kelvin

Se van a añadir las variaciones de presión diarias máximas calculadas como las diferencias de la Pmáx y Pmín. Además de la temperatura en grados kelvin. Podremos eliminar las columnas cuya temperatura se expresa en Celsius.

In [50]:
```
df['TMEDIA_K']=df['TMEDIA']+273.15
df['TMAX_K']=df['TMAX']+273.15
df['TMIN_K']=df['TMIN']+273.15
```

In [51]:
```
df['grad_P']=df['PRESMAX']-df['PRESMIN']
```

In [52]:
```
df.drop(['TMEDIA','TMIN','TMAX'],axis=1)
```

Out[52]:

	FECHA	PRECIPITACION	HORATMIN	HORATMAX	VELMEDIA	SOL	PRESMAX	HORAPRESMAX	PRESMIN
6456	1961-01-01	0.0	05:50	13:25	3.9	4.9	954.0	23	946.6
6457	1961-01-02	5.6	00:20	14:00	3.1	4.2	955.6	11	947.2
6458	1961-01-03	4.0	05:00	15:55	7.8	2.4	947.1	00	937.0
6459	1961-01-04	0.0	03:40	13:10	10.6	3.7	948.4	22	939.9
6460	1961-01-05	0.0	05:10	15:50	4.7	8.2	957.9	23	948.3
...	...	...	...	...	...	...	...	...	...
28611	2022-11-30	0.0	07:40	13:10	1.7	5.9	952.6	10	950.5
28612	2022-12-01	0.0	03:50	15:00	1.4	3.6	952.9	10	950.9
28613	2022-12-02	0.0	23:50	12:00	1.1	1.6	952.3	00	946.5
28614	2022-12-03	0.0	00:00	14:50	1.4	0.0	946.5	00	942.4
28615	2022-12-04	0.0	06:40	13:00	1.1	5.0	951.1	24	943.8

21795 rows × 23 columns

In [53]:
```
df.columns
```

Out[53]:
```
Index(['FECHA', 'TMEDIA', 'PRECIPITACION', 'TMIN', 'HORATMIN', 'TMAX',
       'HORATMAX', 'VELMEDIA', 'SOL', 'PRESMAX', 'HORAPRESMAX', 'PRESMIN',
       'HORAPRESMIN', 'FECHA_dt', 'YEAR', 'MONTH', 'DAY', 'WEEK',
       'HORAPRESMIN_2', 'HORAPRESMAX_2', 'HORATMIN_2', 'HORATMAX_2',
       'TMEDIA_K', 'TMAX_K', 'TMIN_K', 'grad_P'],
      dtype='object')
```

In [54]:
```
df.to_csv('dfForRepresentation')
```

Graficación de los datos y detección de outliers.

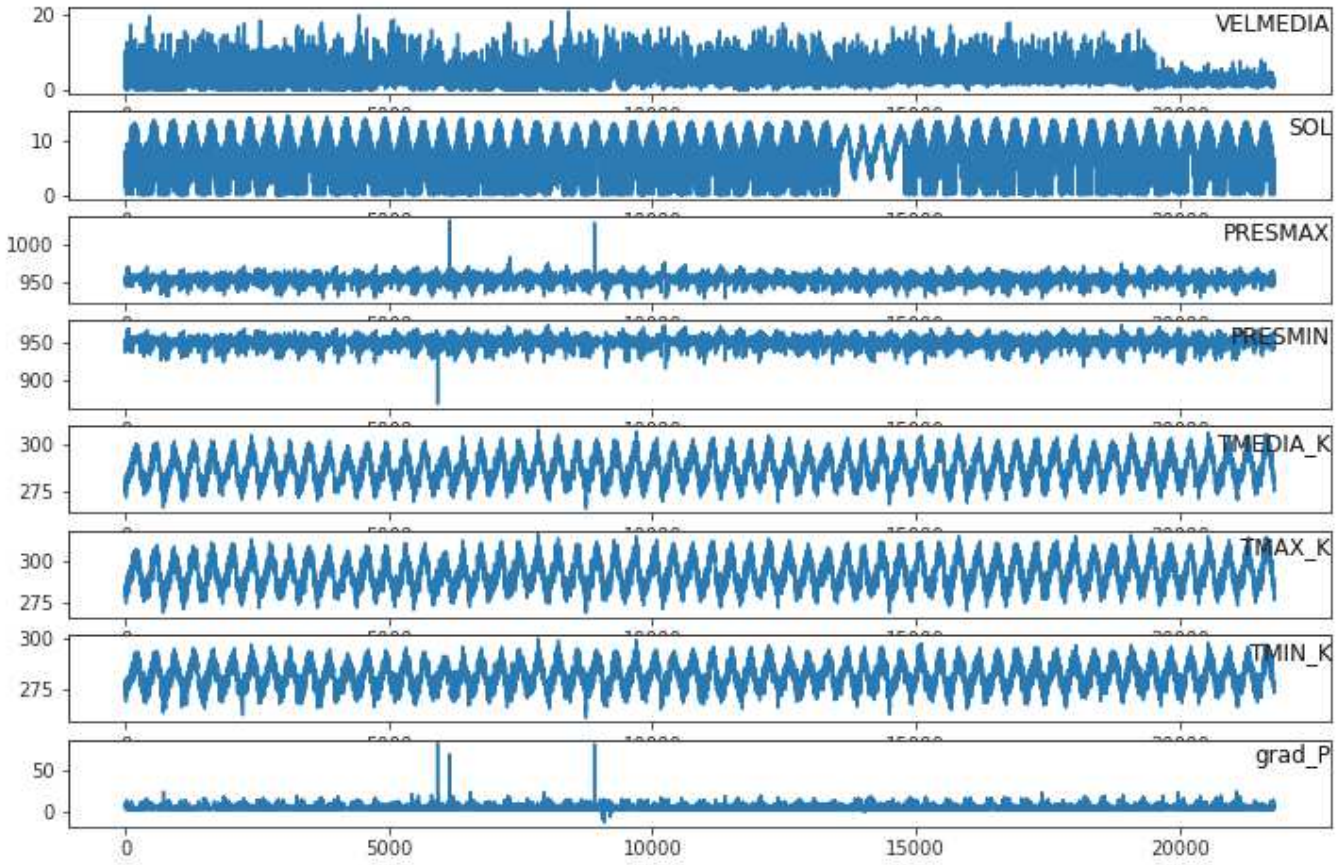
In [55]:
```
df=pd.read_csv('dfForRepresentation',index_col=1)
```

In [56]:
```
df.columns
```

Out[56]:
```
Index(['Unnamed: 0', 'TMEDIA', 'PRECIPITACION', 'TMIN', 'HORATMIN', 'TMAX',
       'HORATMAX', 'VELMEDIA', 'SOL', 'PRESMAX', 'HORAPRESMAX', 'PRESMIN',
       'HORAPRESMIN', 'FECHA_dt', 'YEAR', 'MONTH', 'DAY', 'WEEK',
       'HORAPRESMIN_2', 'HORAPRESMAX_2', 'HORATMIN_2', 'HORATMAX_2',
       'TMEDIA_K', 'TMAX_K', 'TMIN_K', 'grad_P'],
      dtype='object')
```

In [57]:
```
values=df.values

col2plot=[7,8,9,11,22,23,24,25]
i = 1
# plot each column
matplotlib.pyplot.figure()
for group in col2plot:
 matplotlib.pyplot.subplot(len(col2plot), 1, i)
 matplotlib.pyplot.plot(values[:, group])
 matplotlib.pyplot.title(df.columns[group], y=0.6, loc='right')
 i += 1
matplotlib.pyplot.show()
```



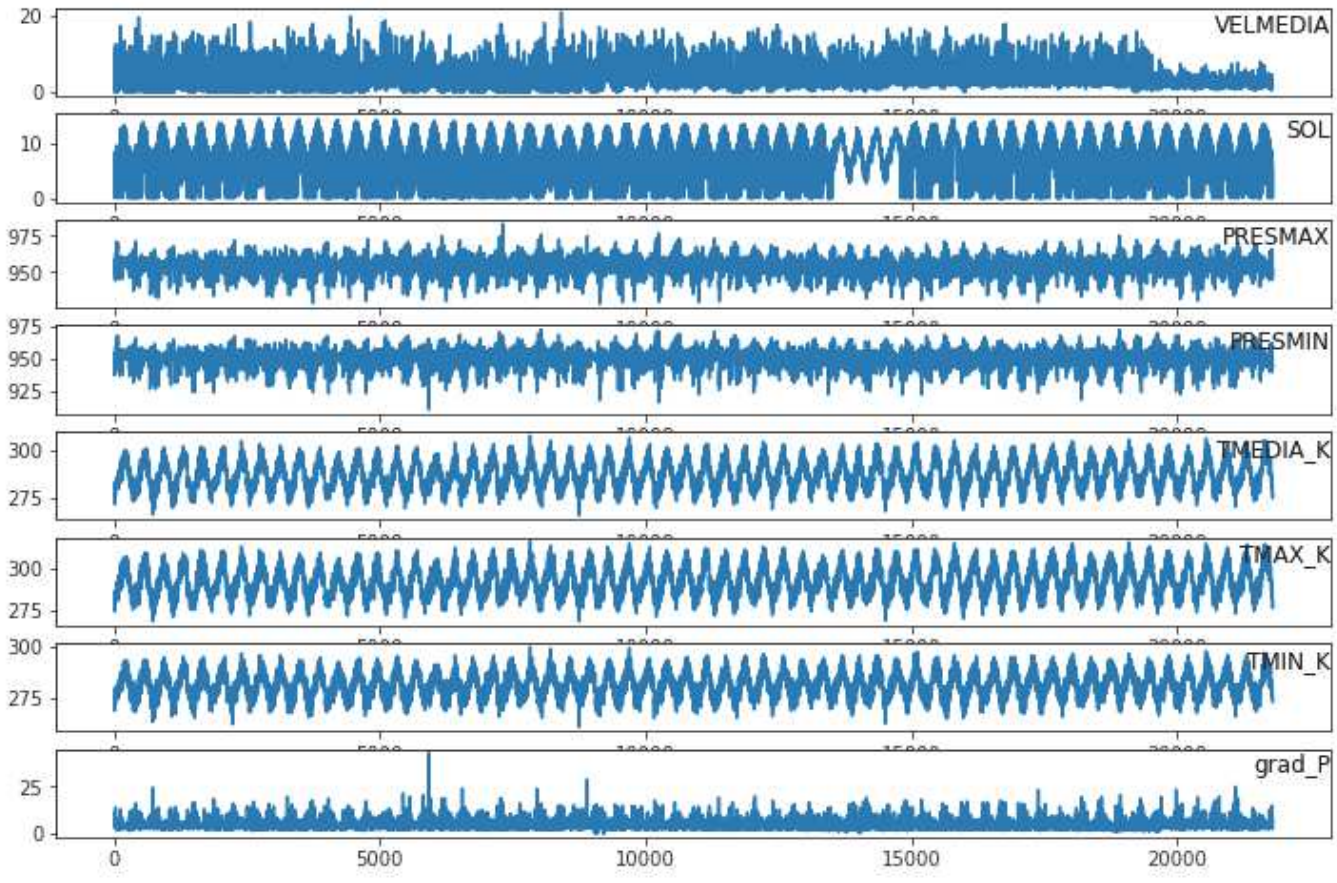
A la vista de estos valores vemos como los valores de temperaturas no presentan anomalías, no obstante los valores de presión presentan algún valor "outlier" y en el caso de sol los datos introducidos no han sido todo lo bueno que se desearía. Se van a filtrar estos valores que tienen una presión mayor a 1000 y menor a 900 suponiendo un mal funcionamiento del sensor. Además cuando el gradiente de presión sea negativo se dará el mismo valor a la presión máxima que a la mínima dado que este fenómeno es prueba de un error seguramente al calcular los valores NAN.

In [58]:
```
for col in ['PRESMAX','PRESMIN']:
  for index,valores in df[col].items():
    if df['grad_P'][index]>0:
      if valores>1000:
        df.loc[index,col]=975
      if valores<900:
        df.loc[index,col]=910
    else:
      df.loc[index,'PRESMAX']=df['PRESMIN'][index]

df['grad_P']=df['PRESMAX']-df['PRESMIN']
```

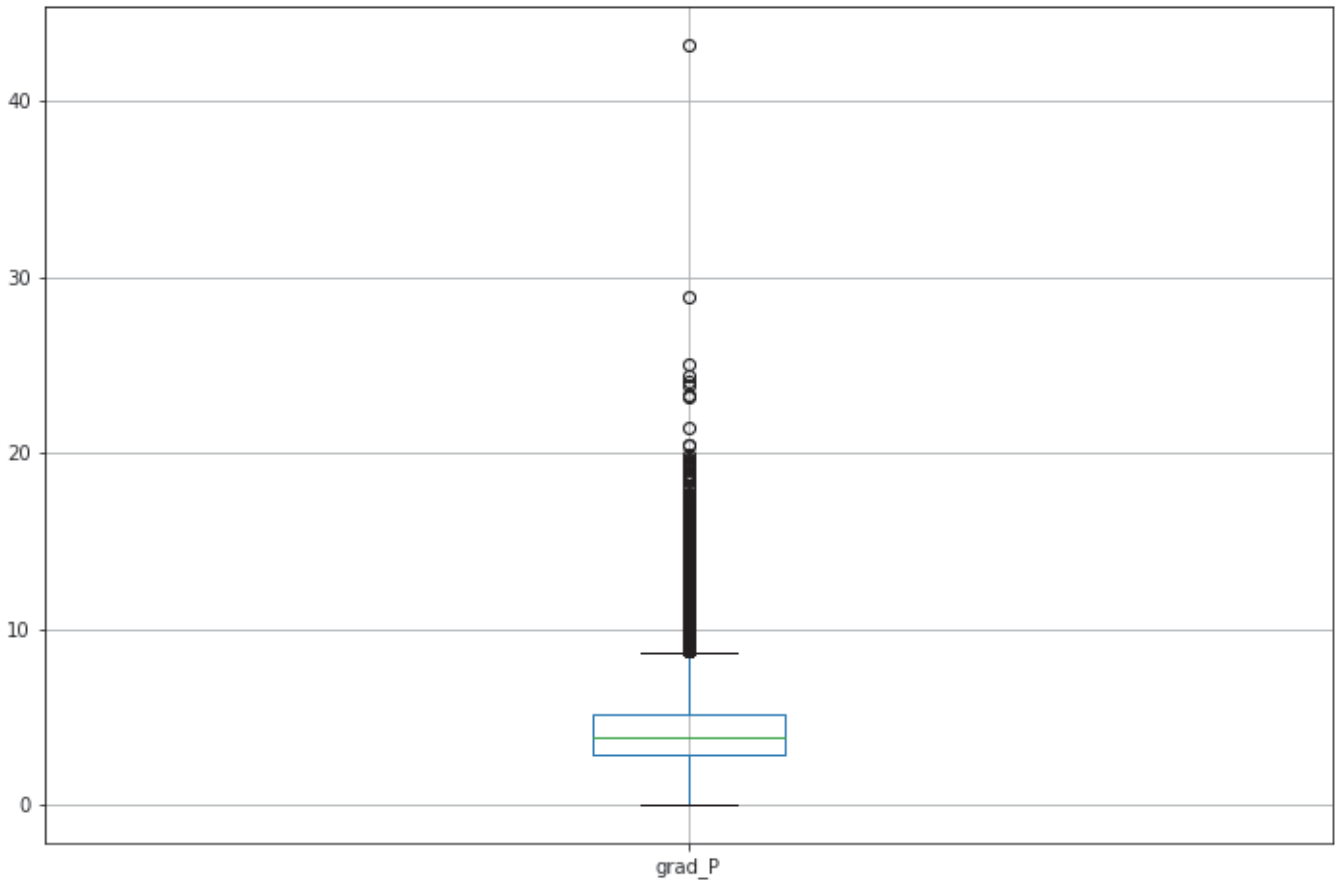
In [59]:
```
values=df.values

col2plot=[7,8,9,11,22,23,24,25]
i = 1
# plot each column
matplotlib.pyplot.figure()
for group in col2plot:
 matplotlib.pyplot.subplot(len(col2plot), 1, i)
 matplotlib.pyplot.plot(values[:, group])
 matplotlib.pyplot.title(df.columns[group], y=0.6, loc='right')
 i += 1
matplotlib.pyplot.show()
```



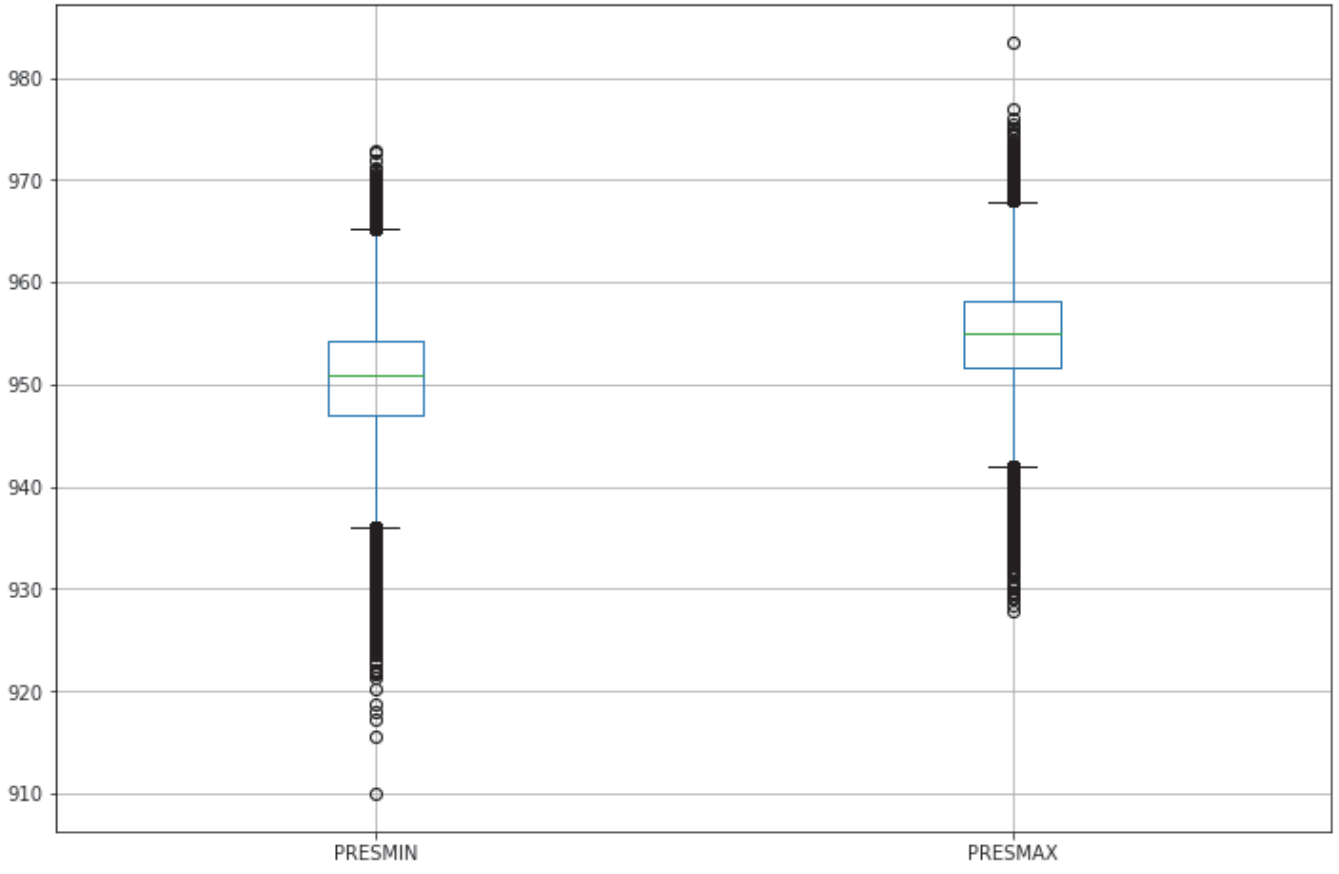
In [60]:
```
df.boxplot(column=['grad_P'])
```

Out[60]:
```
<AxesSubplot:>
```



In [61]:
```
df.boxplot(column=['PRESMIN','PRESMAX'])
```

Out[61]:
```
<AxesSubplot:>
```

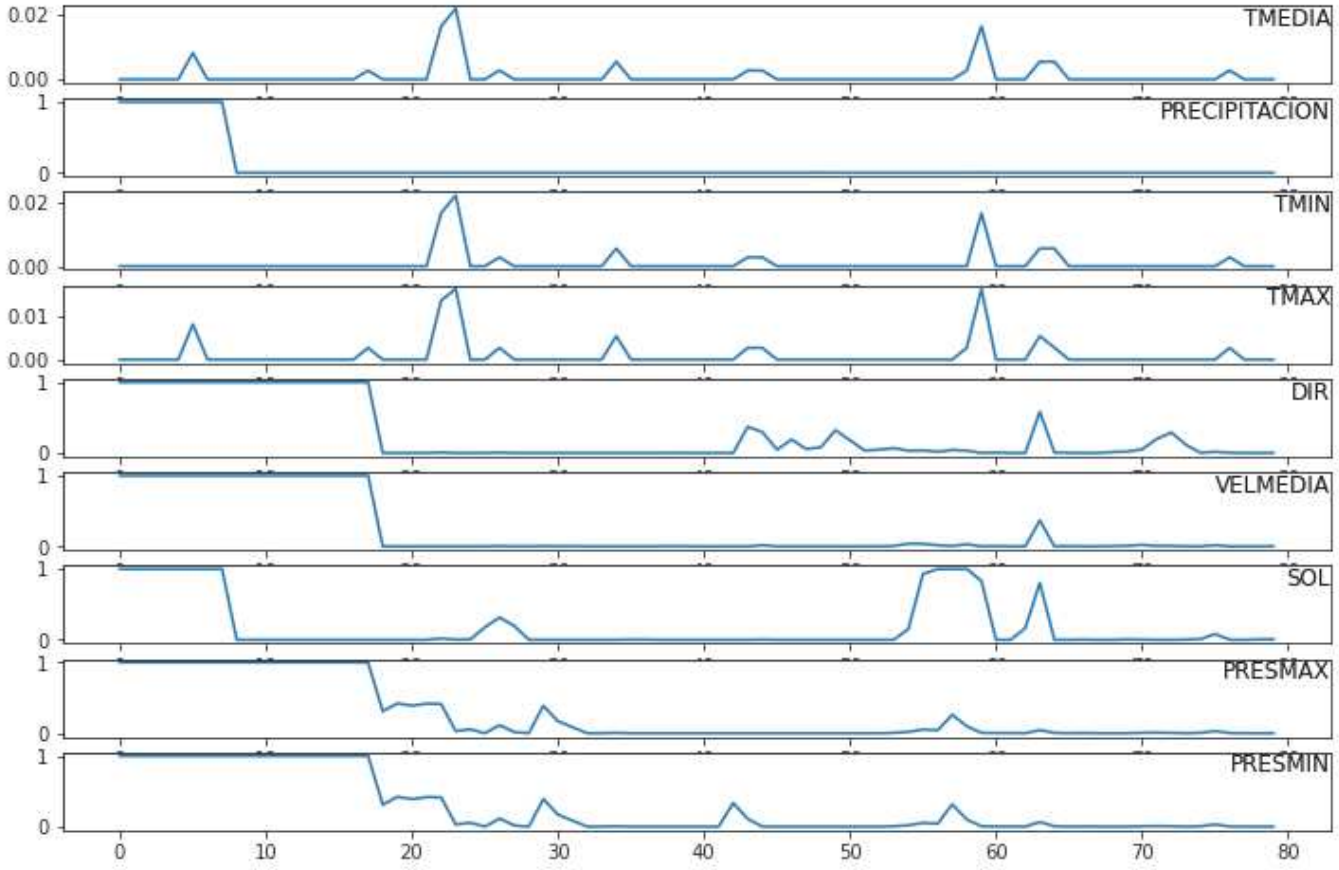


A la vista de estos tres gráficos no nos atrevferiamos a decir que el valor de grad_P de 40 sea un outlier. Bien podría haberse debido a un fenómeno meteorológico concreto en la zona que causará un valor de presión muy bajo o muy alto en un momento puntual, por ejemplo un "tornado". Que si bien son muy poco frecuentes y de intensidad muy moderada (nada comparable a los estadounidenses) se han dado de forma muy puntual en la peninsula y en la región concreta.

Se ha graficado la distribución de los valores nulos parqa poder saber si anomalías en las gráficas se debían a los datos originales o a los datos generados. Nos ha permitido mejorar las funciones empleadas para la adición de estos datos en un proceso iterativo que no se recoge en este documento.

In [62]:
```
values=dfmv.values

col2plot=[5,6,7,9,11,12,15,16,18]
i = 1
# plot each column
matplotlib.pyplot.figure()
for group in col2plot:
 matplotlib.pyplot.subplot(len(col2plot), 1, i)
 matplotlib.pyplot.plot(values[:, group])
 matplotlib.pyplot.title(dfmv.columns[group], y=0.6, loc='right')
 i += 1
matplotlib.pyplot.show()
```



Normalización de los datos

In [63]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
```

In [64]:
```
scaler=MinMaxScaler(feature_range=(0,1))
```

```
dfNorm=df.copy(deep=True)
dfNorm=dfNorm.drop(['Unnamed: 0', 'TMEDIA',  'TMIN', 'HORATMIN', 'TMAX','HORATMAX',  'HO

dfNorm.columns
```

Out[64]:
```
Index(['PRECIPITACION', 'VELMEDIA', 'SOL', 'PRESMAX', 'PRESMIN', 'TMEDIA_K',
       'TMAX_K', 'TMIN_K', 'grad_P'],
      dtype='object')
```

In [65]:
```
#https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/


from pandas import DataFrame
from pandas import concat

def series_to_supervised(data, n_in=1, n_out=1, dropnan=True,):
  """
  Frame a time series as a supervised learning dataset.
  Arguments:
  data: Sequence of observations as a list or NumPy array.
  n_in: Number of lag observations as input (X).
  n_out: Number of observations as output (y).
  dropnan: Boolean whether or not to drop rows with NaN values.
  Returns:
  Pandas DataFrame of series framed for supervised learning.
  """
  n_vars = 1 if type(data) is list else data.shape[1]
  df = DataFrame(data)
  cols, names = list(), list()
  # input sequence (t-n, ... t-1)
  for i in range(n_in, 0, -1):
    cols.append(df.shift(i))
    names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
  # forecast sequence (t, t+1, ... t+n)
  for i in range(0, n_out):

    cols.append(df.shift(-i))
    if i == 0:
      names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
    else:
      names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
  # put it all together
  agg = concat(cols, axis=1)
  agg.columns = names
  # drop rows with NaN values
  if dropnan:
    agg.dropna(inplace=True)
  return agg
```

Esta función se ha cogido de un artículo cuyo enlace encabeza la celda. Nos permite dada una serie temporal obtener los valores t-1 en el horizonte deseado con la salida a predecir en la forma t+1

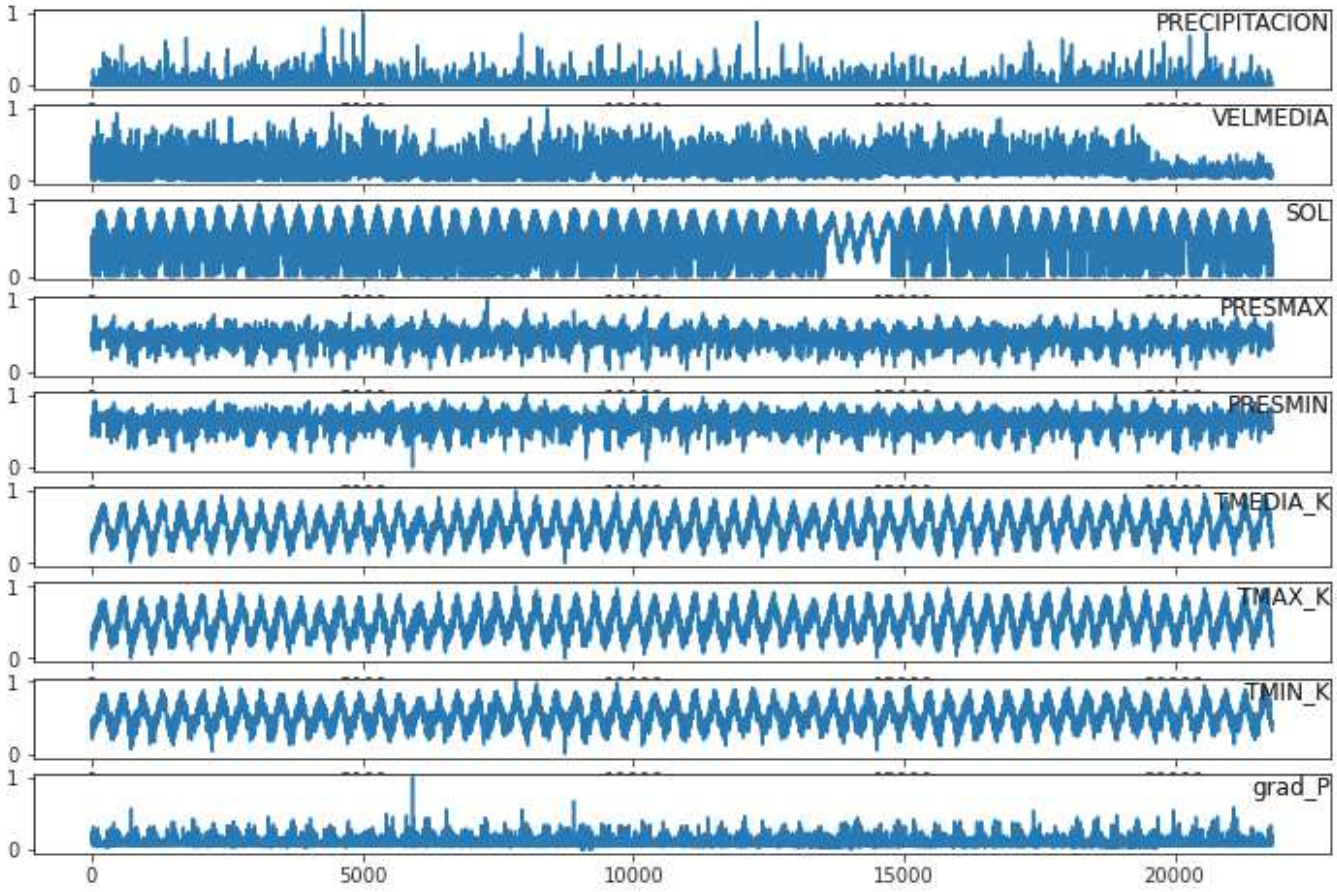
In [66]:
```
Valores = dfNorm.values[:,:]
# integer encode direction
Valores
```

Out[66]:
```
array([[  0.  ,   3.9 ,   4.9 , ..., 282.55, 276.55,   7.4 ],
       [  5.6 ,   3.1 ,   4.2 , ..., 282.55, 274.75,   8.4 ],
       [  4.  ,   7.8 ,   2.4 , ..., 282.55, 277.55,  10.1 ],
       ...,
       [  0.  ,   1.1 ,   1.6 , ..., 279.55, 272.85,   5.8 ],
       [  0.  ,   1.4 ,   0.  , ..., 276.15, 272.85,   4.1 ],
       [  0.  ,   1.1 ,   5.  , ..., 280.05, 273.85,   7.3 ]])
```

In [67]:
```
# ensure all data is float
Valores = Valores.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(Valores)
```

In [68]:
```
dfscaled=pd.DataFrame(scaled,columns=['PRECIPITACION', 'VELMEDIA', 'SOL', 'PRESMAX', 'PR
```

In [69]:
```
dfscaled.values
col2plot=[0,1,2,3,4,5,6,7,8]
i = 1
# plot each column
matplotlib.pyplot.figure()
for group in col2plot:
 matplotlib.pyplot.subplot(len(col2plot), 1, i)
 matplotlib.pyplot.plot(dfscaled.values[:, group])
 matplotlib.pyplot.title(dfscaled.columns[group], y=0.6, loc='right')
 i += 1
matplotlib.pyplot.show()
```



Podemos ver cómo se han normalizado los datos entre 0 y 1. No obstante no se elimina la componente de estacionalidad que podría ser necesaria en función del algoritmo propuesto. Sin embargo, esto queda pendiente y se hará en caso de ser necesario.

In [70]:
```
reframed = series_to_supervised(scaled, 60, 20)
print(reframed.head())
```

```
    var1(t-60)  var2(t-60)  var3(t-60)  var4(t-60)  var5(t-60)  var6(t-60)  \
60    0.000000    0.184834    0.331081    0.469425    0.582802    0.334117
61    0.056169    0.146919    0.283784    0.498201    0.592357    0.312941
62    0.040120    0.369668    0.162162    0.345324    0.429936    0.345882
63    0.000000    0.502370    0.250000    0.368706    0.476114    0.282353
64    0.000000    0.222749    0.554054    0.539570    0.609872    0.280000
```

```
    var7(t-60)  var8(t-60)  var9(t-60)  var1(t-59)  ...  var9(t+18)  \
60    0.299578    0.406091    0.171296    0.056169  ...    0.057870
61    0.299578    0.360406    0.194444    0.040120  ...    0.094907
62    0.299578    0.431472    0.233796    0.000000  ...    0.162037
63    0.244726    0.360406    0.196759    0.000000  ...    0.101852
64    0.270042    0.324873    0.222222    0.000000  ...    0.071759

    var1(t+19)  var2(t+19)  var3(t+19)  var4(t+19)  var5(t+19)  var6(t+19)  \
60         0.0    0.170616    0.412162    0.359713    0.538216    0.421176
61         0.0    0.052133    0.608108    0.480215    0.598725    0.378824
62         0.0    0.170616    0.472973    0.553957    0.705414    0.449412
63         0.0    0.090047    0.614865    0.575541    0.745222    0.428235
64         0.0    0.014218    0.614865    0.573742    0.754777    0.425882

    var7(t+19)  var8(t+19)  var9(t+19)
60    0.413502    0.456852    0.094907
61    0.417721    0.360406    0.162037
62    0.430380    0.497462    0.101852
63    0.421941    0.461929    0.071759
64    0.434599    0.441625    0.055556

[5 rows x 720 columns]
```

Con esto tendríamos los datos preparados para el entrenamiento. Quedaría eso si eliminar las variables que no queramos estimar. Pero esto lo dejo para la implementación final.

In [ ]:

In [ ]: