PRÁCTICA 6
ALMACÉNS E MINARÍA DE DATOS

Álex Baquero DomÍnguez
alex.baquero@rai.usc.es

Hugo Gómez Sabucedo
hugo.gomez.sabucedo@rai.usc.es

Alejandro Aybar Cifuentes
alejandro.aybar@rai.usc.es

Índice

1. Valoración de los valores atípicos en los archivos .csv

Aclaración: Según hemos hablado en clase el grupo y el profesor, hemos decidido dejar este primer ejercicio en R, al tratarse de un paso de limpieza de datos. Sin embargo, para el resto de ejercicios, acorde a lo decidido a nivel de clase, vamos a utilizar Python, que es el lenguaje más al alza para los análisis de aprendizaje automático.

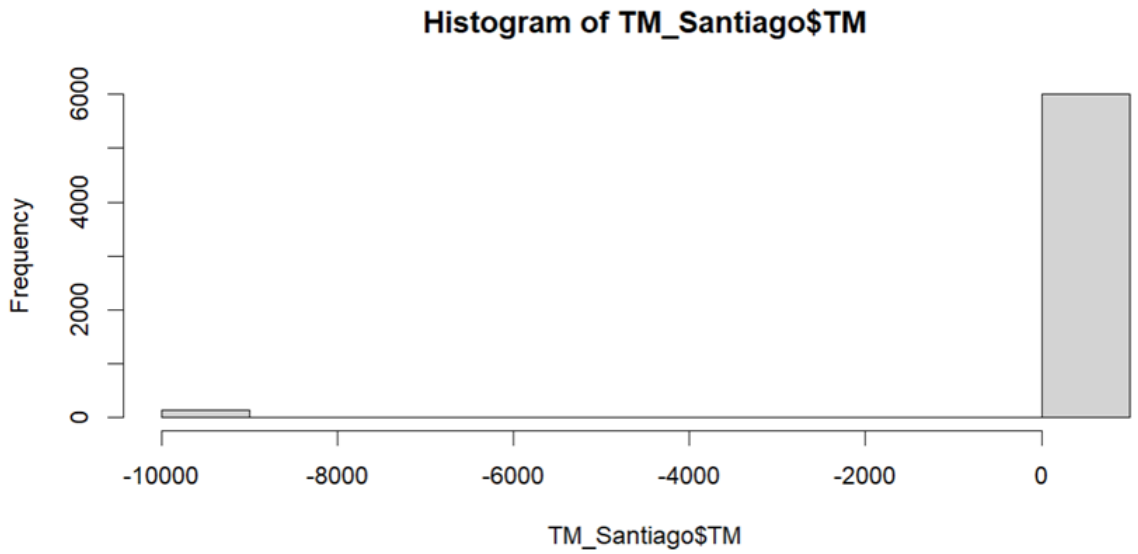
Conjunto de datos TM_Santiago

En primer lugar, instalamos la librería de readr y la importamos. Con ella, procedemos a importar el fichero CSV.

```r
# instalar y cargar paquetes
install.packages("readr")
library(readr)


# establecer directorio de trabajo (donde tienes los CSV)
setwd("C:/Users/alex1/Documents/UNIVERSIDAD/AMD/practicas/p6")

# leer CSV
TM_Santiago <- read_delim("./TM_Santiago.csv", delim = "|", escape_double = FALSE, trim_ws = TRUE)

```

Vamos a comenzar el análisis de los valores atípicos mirando el histograma del conjunto de datos.

```r
# ver histograma
hist(TM_Santiago$TM)
```



Mientras que lo normal es tener valores positivos, encontramos en el histograma valores situados en los -10000, lo cual nos hace pensar en la posible presencia de valores atípicos.

Otro análisis que podemos realizar sobre los datos es el resumen estadístico del conjunto de datos.

```
> summary(TM_Santiago$TM)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-9999.0    13.1    17.5  -206.7    22.6    40.0
>
```

A simple vista nos llama la atención la gran diferencia entre la mediana (17.5) y la media (-206.7). Esto nos hace pensar que hay valores negativos muy grandes que nos están llevando la media hacia el eje negativo.

El hecho de que la mediana sea 17.5 y el máximo sea 40.0, nos deja claro que estos son los valores por los que se debe mover la variable en condiciones normales. Por otro lado, el tener un mínimo de -9999.0 nos deja claro que existen valores atípicos en el modelo.

También podemos obtener información de valor a partir del rango intercuartílico (IQR). Este rango refleja el 50% de los valores, que se encuentran entre Q1 (13.1) y Q3 (22.6).

IQR = Q3 - Q1 = 22.6 - 13.1 = **9.5**

A nivel teórico, se denomina valor atípico leve a todos aquellos valores que distan 1,5 veces el rango intercuartílico por debajo de Q1 o por encima de Q3. Un valor atípico extremo es aquel que dista 3 veces el rango intercuartílico.

Vamos a calcular los umbrales de los valores atípicos leves y los umbrales de los valores atípicos extremos:

```
18  Q1 <- 13.1
19  Q3 <- 22.6
20  IQR <- Q3 - Q1
21
22  umbral_leve_arriba <- Q3 + 1.5 * IQR
23  umbral_leve_abajo <- Q1 - 1.5 * IQR
24
25  umbral_ext_arriba <- Q3 + 3 * IQR
26  umbral_ext_abajo <- Q1 - 3 * IQR
27
```

Values	
IQR	9.5
Q1	13.1
Q3	22.6
umbral_ext_abajo	-15.4
umbral_ext_arriba	51.1
umbral_leve_abajo	-1.15
umbral_leve_arri…	36.85

Con estos valores obtenidos, consideraremos como valor atípico leve:

valor_atípico_leve < -1.15 || valor_atípico_leve > 36.85

Y como valor atípico extremo:

valor_atípico_extremo < -15.4 || valor_atípico_extremo > 51.1

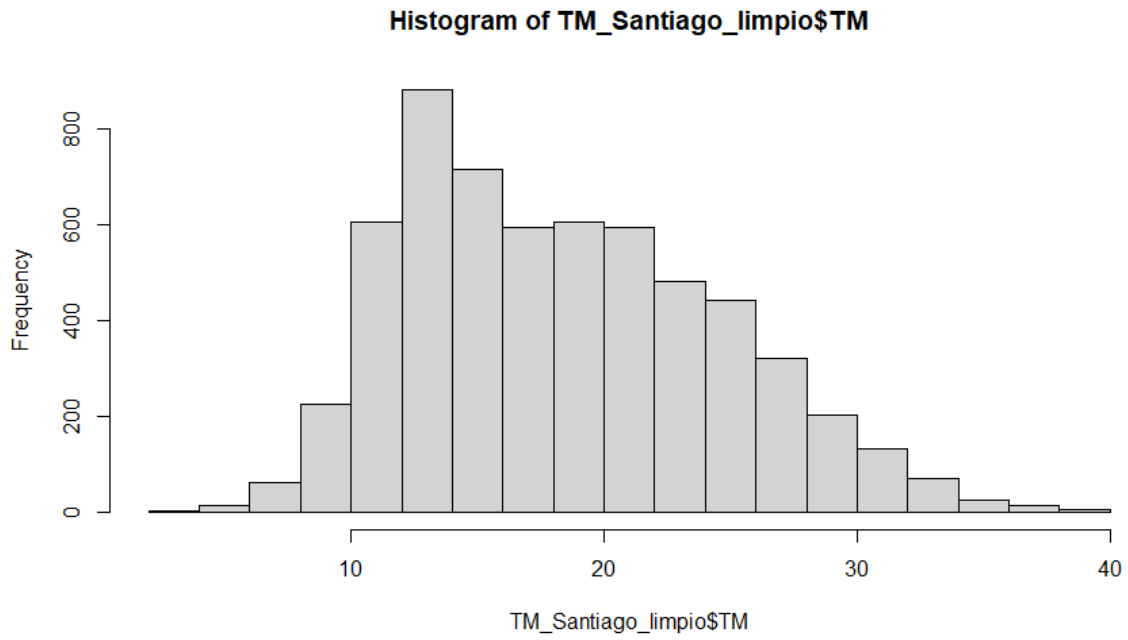
Hemos decidido quitar los valores atípicos extremos, ya que restan valor estadístico a nuestro estudio (ej. -9999.0), pero dejar los valores atípicos leves, ya que pueden deberse a condiciones especiales de un día concreto, pero ser valores correctamente registrados (ej. 37.5).

Para ello, filtramos el conjunto de datos, de manera que excluyamos los valores que estén por debajo del umbral extremo inferior y los que estén por encima del umbral extremo superior.

```
31
32  # eliminamos los valores atípicos extremos
33
34  TM_Santiago_limpio <- TM_Santiago[TM_Santiago$TM > umbral_ext_abajo &
35                                    TM_Santiago$TM < umbral_ext_arriba, c("Fecha", "TM")]
36
```

Consultamos el histograma del conjunto de datos filtrado:

```
35
36  #vemos histograma del conjunto limpio
37  hist(TM_Santiago_limpio$TM)
38
```



Como podemos observar, el histograma ahora está equilibrado, y se mueve en un rango de valores razonable. Por lo tanto, podemos confirmar que hemos eliminado los valores atípicos que perjudicaban nuestro análisis.

Si echamos un vistazo al resumen estadístico, vemos que los valores de mediana y media tienen una distancia adecuada, y bien ajustada con respecto a los valores mínimo y máximo del conjunto.

```
> summary(TM_Santiago_limpio$TM)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2.80   13.40   17.70   18.43   22.80   40.00
> |
```

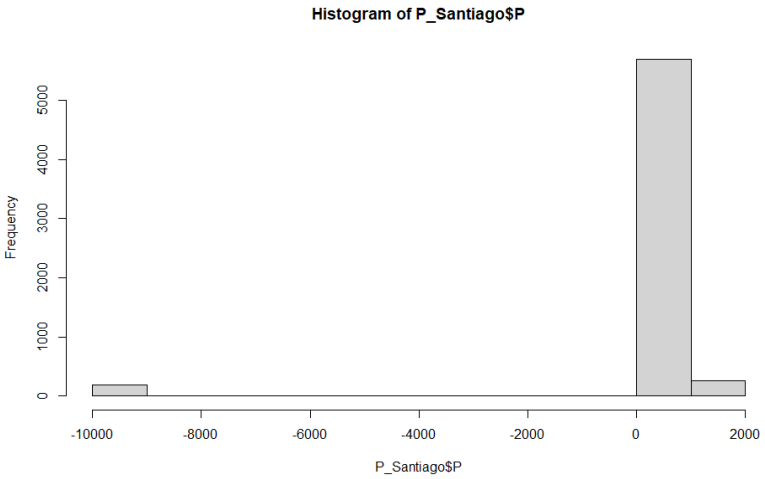
Conjunto de datos P_Santiago

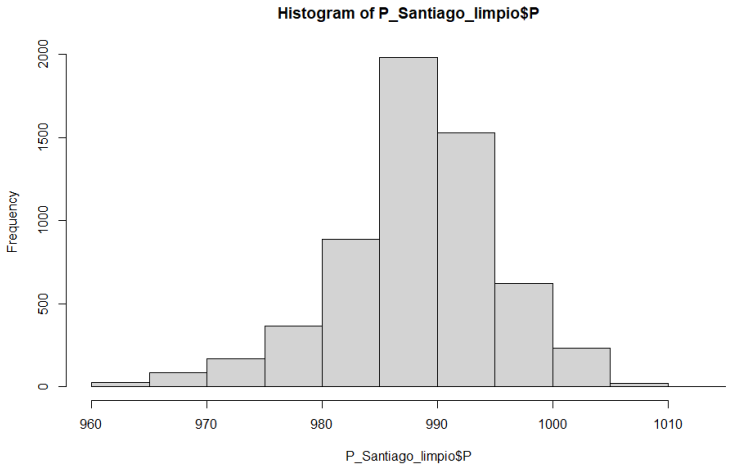
Dado que los datos tienen la misma estructura que en el conjunto TM_Santiago, aplicamos los mismos pasos que en el caso anterior.

```
# establecemos directorio de trabajo (donde tenemos los csv)
setwd("C:/Users/alex1/Documents/UNIVERSIDAD/AMD/practicas/p6")

# leer csv
P_Santiago <- read_delim("./P_Santiago.csv", delim = "|", escape_double = FALSE, trim_ws = TRUE)

# ajustamos el formato del campo fecha
P_Santiago$Fecha <- as.Date(P_Santiago$Fecha, format="%d-%m-%Y")

# vemos histograma
hist(P_Santiago$P)

# vemos resumen estadístico (media, mediana, cuantiles, etc)
summary(P_Santiago$P)
```

```r
# guardamos los cuartiles y calculamos el rango intercuartílico
P_Q1 <- 984.2
P_Q3 <- 992.3
P_IQR <- P_Q3 - P_Q1

# calculamos umbrales leves
P_umbral_leve_arriba <- P_Q3 + 1.5 * P_IQR
P_umbral_leve_abajo <- P_Q1 - 1.5 * P_IQR

# calculamos umbrales extremos
P_umbral_ext_arriba <- P_Q3 + 3 * P_IQR
P_umbral_ext_abajo <- P_Q1 - 3 * P_IQR

# eliminamos los valores atípicos extremos
P_Santiago_limpio <- P_Santiago[P_Santiago$P > P_umbral_ext_abajo &
                                P_Santiago$P < P_umbral_ext_arriba, c("Fecha", "P")]

#vemos histograma del conjunto limpio
hist(P_Santiago_limpio$P)

# vemos resumen estadístico del conjunto limpio
summary(P_Santiago_limpio$P)

#guardamos el conjunto limpio en un CSV
write.csv(P_Santiago_limpio, "./P_Santiago_limpio.csv", row.names=FALSE)
```

Comparamos el histograma antes y después de la limpieza:





Como podemos observar, la limpieza de datos se ha hecho también correctamente para este conjunto de datos.

Conjunto de datos IRRA

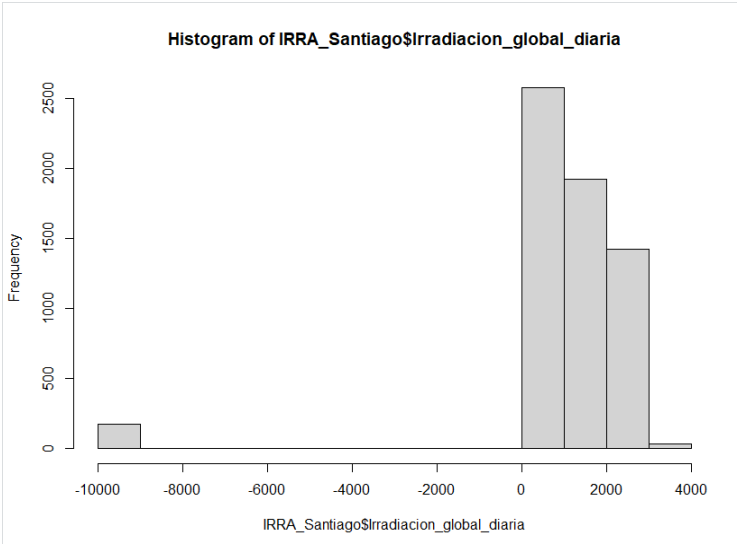
Para este conjunto, al tener la misma estructura que los otros dos, aplicamos los mismos pasos. Hemos tenido que corregir el nombre de las columnas, ya que tanto la tilde de

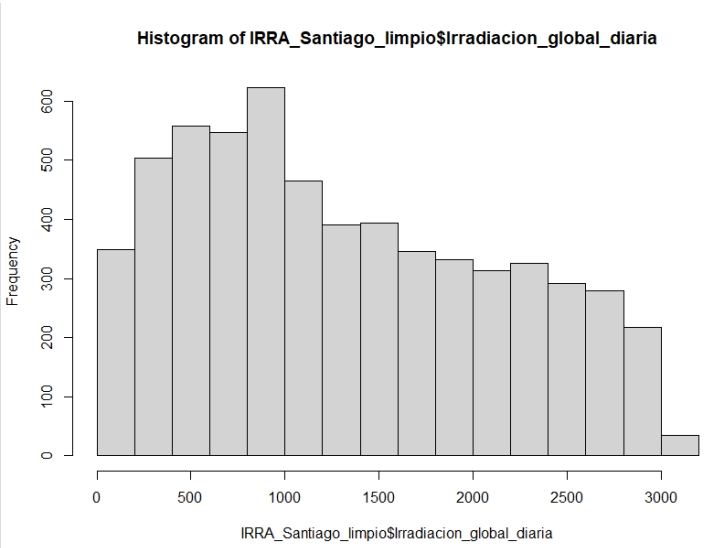
“Irradiación global diaria” como los espacios daban problemas a la hora de importar el CSV. Esto lo hemos solucionado sustituyendo los espacios por “_”, y quitando la tilde.

Por lo demás, el procedimiento es el mismo que en los dos conjuntos de datos anteriores.

```r
# establecemos directorio de trabajo (donde tenemos los csv)
setwd("C:/Users/alex1/Documents/UNIVERSIDAD/AMD/practicas/p6")

# leer csv
IRRA_Santiago <- read_delim("./IRRA_Santiago.csv", delim = "|", escape_double = FALSE, trim_ws = TRUE)

# ajustamos el formato del campo fecha
IRRA_Santiago$Instante_lectura <- as.Date(IRRA_Santiago$Instante_lectura, format="%d-%m-%Y")

# vemos histograma
hist(IRRA_Santiago$Irradiacion_global_diaria)

# vemos resumen estadístico (media, mediana, cuantiles, etc)
summary(IRRA_Santiago$Irradiacion_global_diaria)

# guardamos los cuartiles y calculamos el rango intercuartílico
IRRA_Q1 <- 583.0
IRRA_Q3 <- 1944.0
IRRA_IQR <- IRRA_Q3 - IRRA_Q1

# calculamos umbrales leves
IRRA_umbral_leve_arriba <- IRRA_Q3 + 1.5 * IRRA_IQR
IRRA_umbral_leve_abajo <- IRRA_Q1 - 1.5 * IRRA_IQR

# calculamos umbrales extremos
IRRA_umbral_ext_arriba <- IRRA_Q3 + 3 * IRRA_IQR
IRRA_umbral_ext_abajo <- IRRA_Q1 - 3 * IRRA_IQR


# eliminamos los valores atípicos extremos
IRRA_Santiago_limpio <- IRRA_Santiago[IRRA_Santiago$Irradiacion_global_diaria > IRRA_umbral_ext_abajo &
                                      IRRA_Santiago$Irradiacion_global_diaria < IRRA_umbral_ext_arriba,
                                      c("Instante_lectura", "Irradiacion_global_diaria")]


#vemos histograma del conjunto limpio
hist(IRRA_Santiago_limpio$Irradiacion_global_diaria)

# vemos resumen estadístico del conjunto limpio
summary(IRRA_Santiago_limpio$Irradiacion_global_diaria)

#guardamos el conjunto limpio en un csv
write.csv(IRRA_Santiago_limpio, "./IRRA_Santiago_limpio.csv", row.names=FALSE)

```

Volvemos a comparar los histogramas para ver cómo los valores atípicos han sido eliminados correctamente:





2. Creación de serie temporal. Primera y segunda diferencia y ACF. Descomposición en componentes estacionales y tendencia

Como hemos acordado, a partir de aquí vamos a trabajar en Python.

2.1. Variable TM

Comenzamos leyendo el fichero CSV con ayuda de la librería pandas. Utilizamos los comandos shape y describe a modo de resumen del contenido del dataframe.

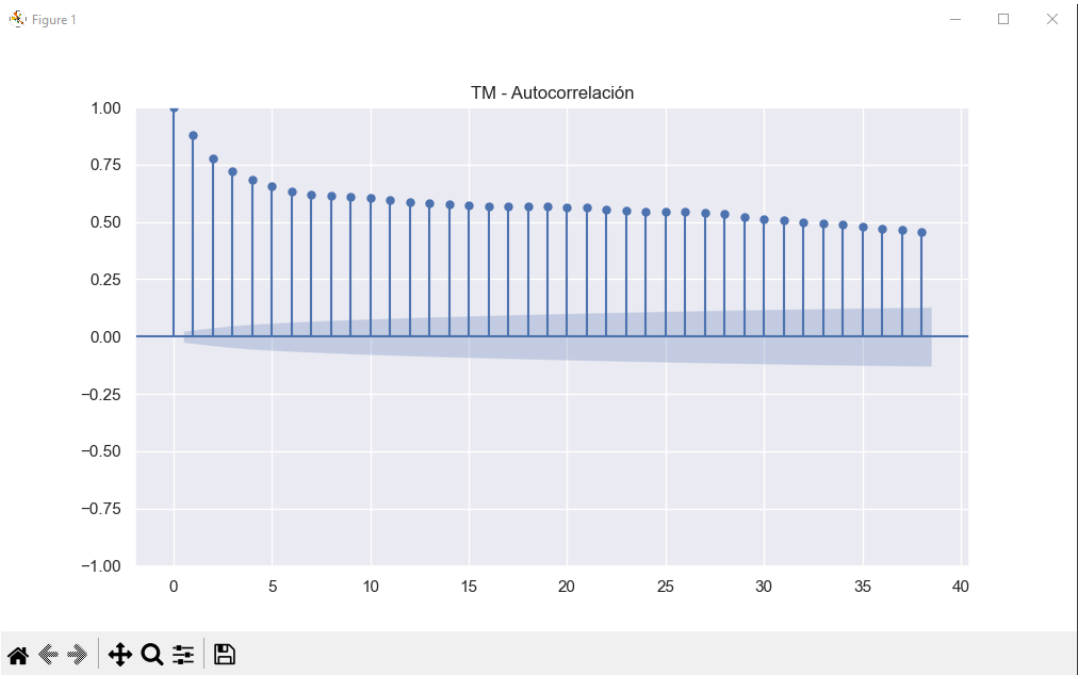
```python
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
import statsmodels.api as sm
sns.set()

# importar CSV
serieTM = pd.read_csv('./TM_Santiago_limpio.csv')

# ver dataframe
print(serieTM.head(4))
print(serieTM.tail(4))

# resumen dataframe
print(serieTM.shape)
print(serieTM.describe())
```

```
PS C:\Users\alex1\Documents\UNIVERSIDAD\AMD\practicas\p6>
        Fecha    TM
0  2006-01-01  11.1
1  2006-01-02  11.6
2  2006-01-03  13.6
3  2006-01-04  14.3
          Fecha    TM
5999  2022-10-21  17.3
6000  2022-10-22  17.0
6001  2022-10-23  17.0
6002  2022-10-24  14.3
(6003, 2)
                TM
count  6003.000000
mean     18.426637
std       6.174549
min       2.800000
25%      13.400000
50%      17.700000
75%      22.800000
max      40.000000
```

Como podemos observar, tenemos 6003 observaciones. Además, el resumen estadístico coincide con el que obtuvimos cuando limpiamos los valores atípicos en el ejercicio anterior.

Moviéndonos ya a un análisis de la serie temporal en sí, vamos a comprobar si nuestra serie es estacionaria o no. Para ello, analizamos la autocorrelación de la variable TM mediante una gráfica ACF.

```
# AFC normal
TM = serieTM.loc[:,"TM"]
sm.graphics.tsa.plot_acf(TM, title="TM - Autocorrelación")

plt.show()
```



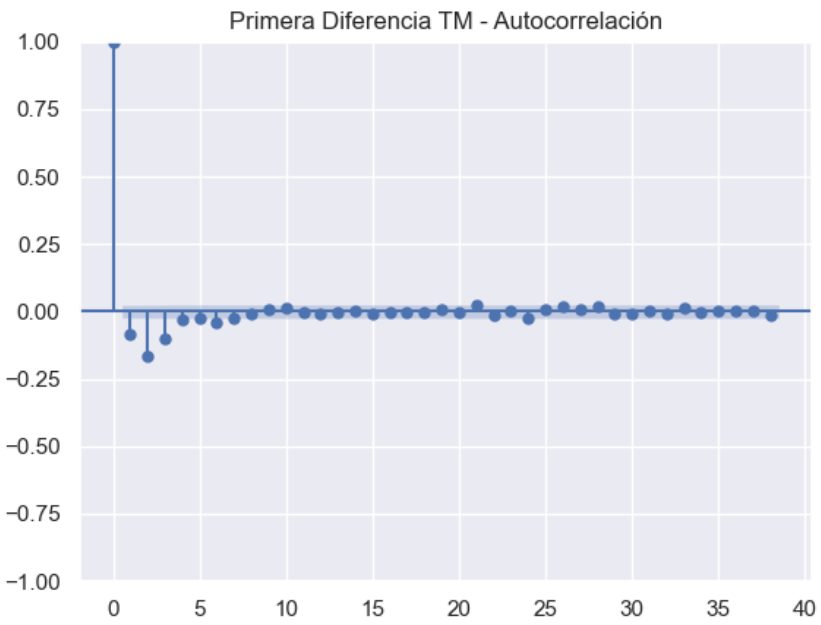
Como podemos observar, la gráfica AFC decae lentamente. Confirmamos, por tanto, que la variable TM describe una serie no estacionaria.

Primera diferencia

Como buscamos una serie estacionaria, vamos a calcular la primera diferencia de la variable TM, ayudándonos de la función diff() de la librería numpy.

```
# AFC normal
TM = serieTM.loc[:,"TM"]
sm.graphics.tsa.plot_acf(TM, title="TM - Autocorrelación")

# AFC de la primera diferencia
TM_diff_1 = np.diff(TM)
sm.graphics.tsa.plot_acf(TM_diff_1, title="Primera Diferencia TM - Autocorrelación")

plt.show()
```

Vamos a analizar el diagrama ACF de la primera diferencia:



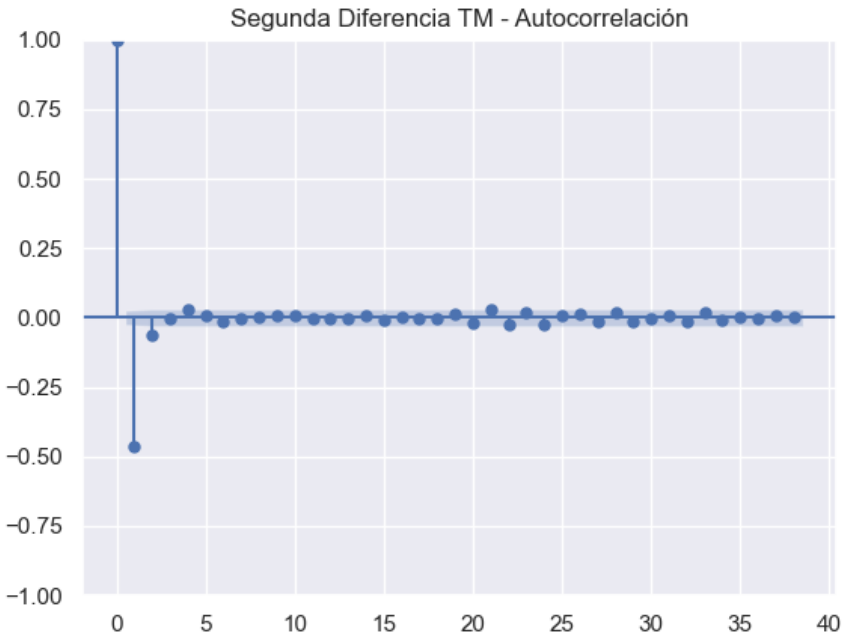
Observamos que en este caso la gráfica no decae lentamente. Por lo tanto, podemos determinar que la primera diferencia de la variable TM es una serie temporal estacionaria.

Segunda Diferencia

Para calcular la segunda diferencia, volvemos a aplicar la función diff(), en este caso sobre la primera diferencia.

```
19  # AFC normal
20  TM = serieTM.loc[:,"TM"]
21  sm.graphics.tsa.plot_acf(TM, title="TM - Autocorrelación")
22
23  # AFC de la primera diferencia
24  TM_diff_1 = np.diff(TM)
25  sm.graphics.tsa.plot_acf(TM_diff_1, title="Primera Diferencia TM - Autocorrelación")
26
27  # AFC de la segunda diferencia
28  TM_diff_2 = np.diff(TM_diff_1)
29  sm.graphics.tsa.plot_acf(TM_diff_2, title="Segunda Diferencia TM - Autocorrelación")
30
31  plt.show()
32
```

Observamos el diagrama AFC para la segunda diferencia:



Como podemos ver, la segunda diferencia también podría ser usada como serie temporal estacionaria.

Descomposición en Componentes Estacionales

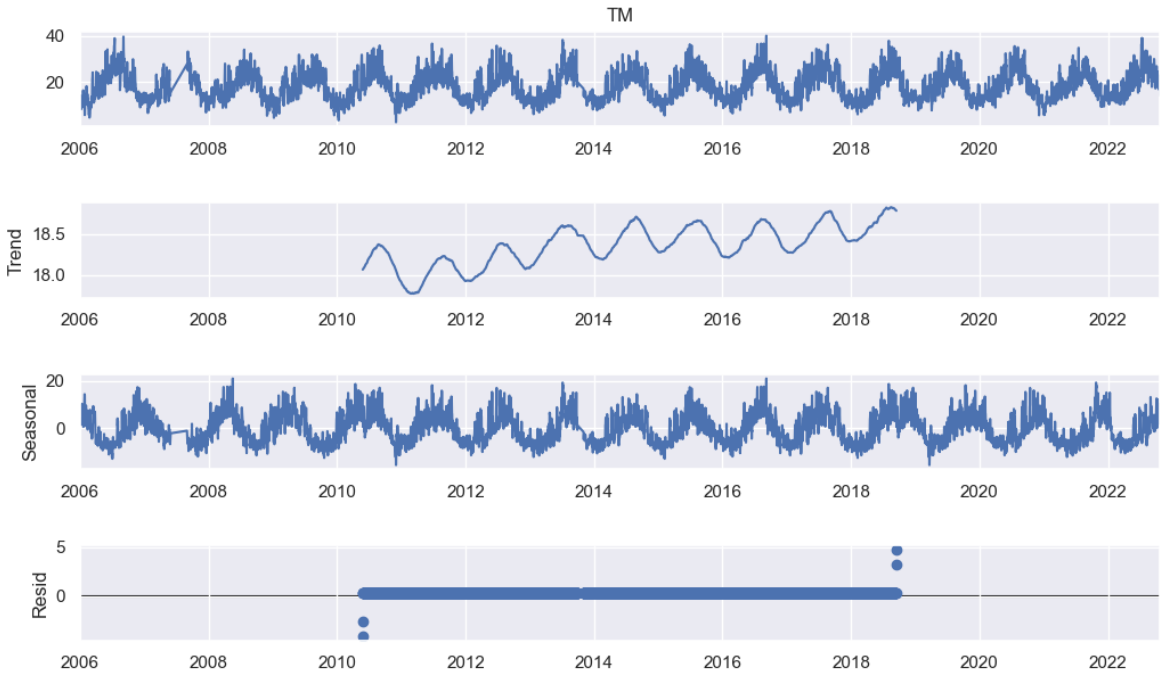
Vamos a descomponer la serie temporal en componentes estacionales, tendencia y residuos. Para ello, nos ayudamos de la función seasonal_decompose() de la librería statsmodel.

Para que la función nos reconozca la frecuencia adecuadamente, hemos tenido que realizar dos modificaciones en el dataframe. En primer lugar, hemos tenido que transformar la columna 'Fecha' con la función 'to_datetime' de pandas, para que pase a ser de un string a un timestamp. Por otro lado, hemos indicado que el índice del dataframe sea el campo 'Fecha' con set_index().

```
8   # importar CSV
9   serieTM = pd.read_csv('./TM_Santiago_limpio.csv', parse_dates = True)
10
11  # campo Fecha como índice
12  serieTM['Fecha'] = pd.to_datetime(serieTM['Fecha'])
13  serieTM = serieTM.set_index('Fecha')
14
15  # ver dataframe
16  print(serieTM.head(4))
17  print(serieTM.tail(4))
18
19  # resumen dataframe
20  print(serieTM.shape)
21  print(serieTM.describe())
22
```

Ahora sí, podemos ejecutar la función seasonal_decompose(). Como no sabemos exactamente qué período utilizar, empezamos indicando period=3001, para que tengamos dos ciclos completos, teniendo en cuenta que el número de observaciones es 6003.

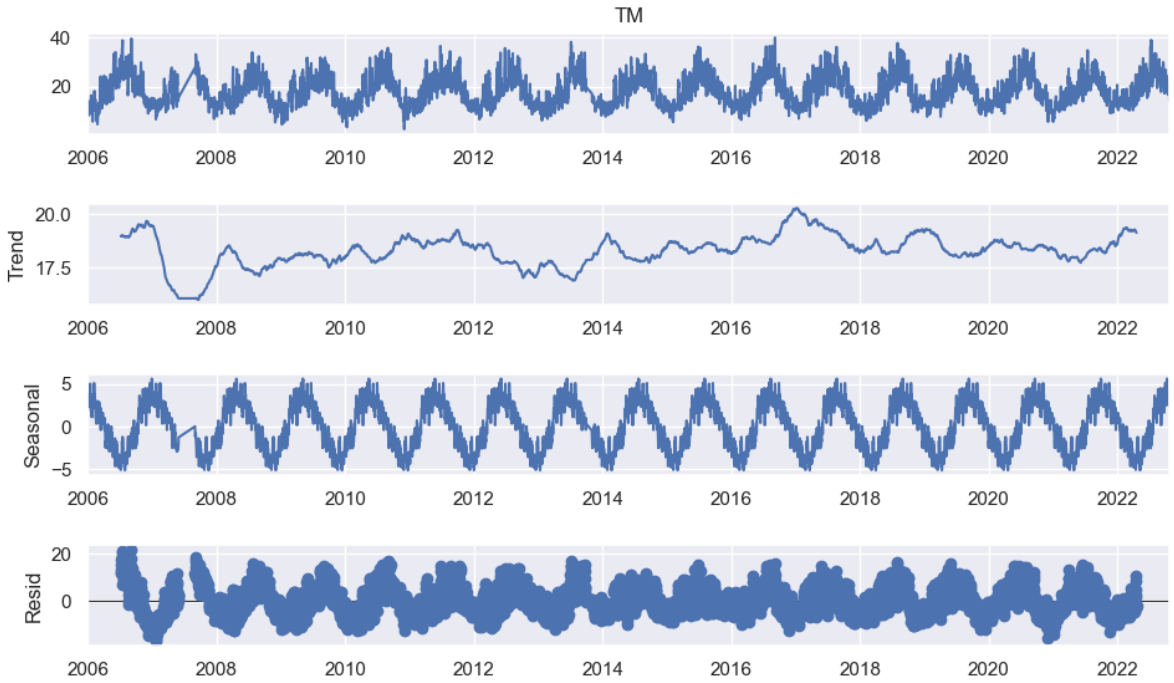
```
38
39  # descomponer en componentes estacionales, tendencia y residuos
40  descomposicion_TM = sm.tsa.seasonal_decompose(serieTM.TM, model='additive', period=6003)
41  descomposicion_TM.plot()
42
43  plt.show()
44
```



Como podemos observar, la tendencia describe un ascenso. Sin embargo, no se cubre el espectro temporal completo de la serie, ya que la tendencia y los residuos solo existen entre 2010 y 2018.

Si observamos las gráficas de observaciones y de componentes estacionales, podemos observar cómo se describe un ciclo por año. Como tenemos datos desde 2006 hasta 2022, habrá 16 ciclos. Por lo tanto, el periodo será de 6003 / 16 = 375.

```
38
39  # descomponer en componentes estacionales, tendencia y residuos
40  descomposicion_TM = sm.tsa.seasonal_decompose(serieTM.TM, model='additive', period=375)
41  descomposicion_TM.plot()
42
43  plt.show()
44
```



Por lo tanto, esta será la descomposición en componentes estacionales y tendencia más precisa para la variable TM.

2.2. Variable P

Para empezar, empezamos importando todas las librerias que vamos a necesitar para este ejercicio. Hacemos un .set y leemos el archivo .csv P_Santiago limpio, es decir, que tiene los valores atipicos eliminados. Ejecutamos por pantalla el dataframe.

```python
import statsmodels.api as sm
import pandas as pnd
import seaborn as sns
import matplotlib.pyplot as mplt
import numpy as np

#empezamos lendo o .csv
sns.set()
serieP = pnd.read_csv('P_Santiago.csv')

#ver dataframe e resumen do datafram
print(serieP.head(4));
print(serieP.tail(4));
print(serieP.shape);
print(serieP.describe())
```

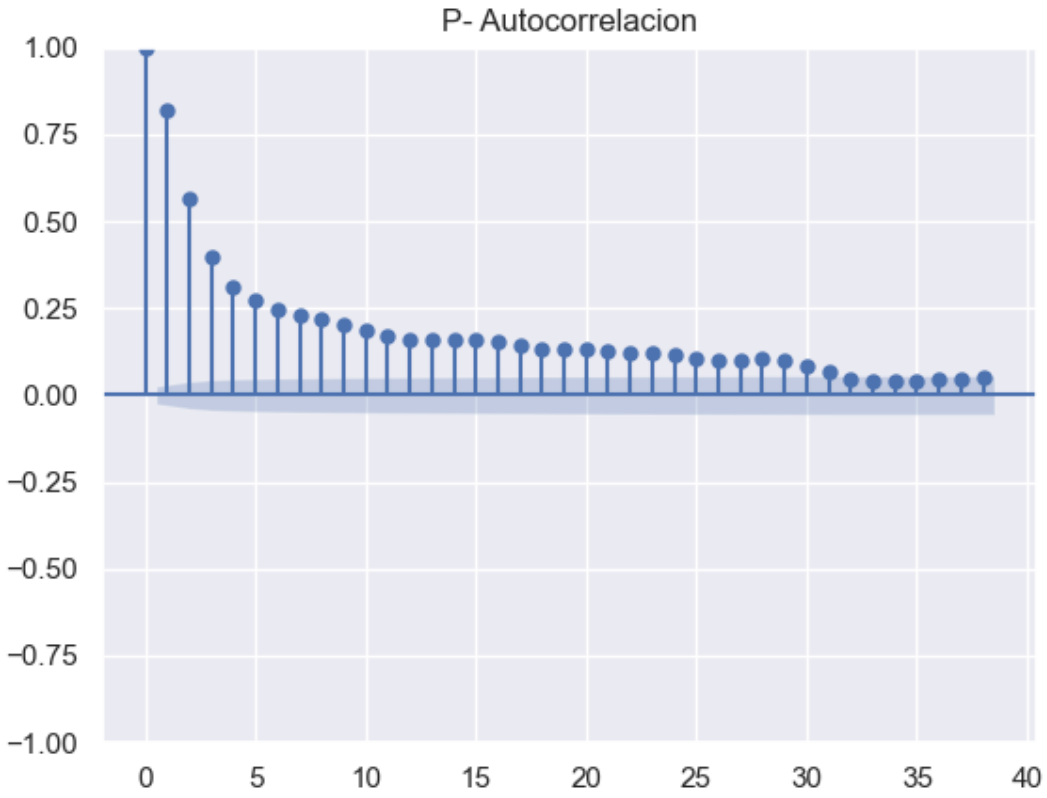
Tenemos 5930 observaciones. Este número en el resumen estadístico coincide con el count.

```
        Fecha       P
0  01/01/2006   997.6
1  02/01/2006  1002.8
2  03/01/2006  1002.9
3  04/01/2006   997.1
          Fecha      P
5926  21/10/2022  978.6
5927  22/10/2022  975.2
5928  23/10/2022  976.6
5929  24/10/2022  983.1
(5930, 2)
                P
count  5930.000000
mean    988.345177
std       7.060993
min     960.200000
25%     984.800000
50%     988.800000
75%     992.500000
max    1011.600000
```

Procedemos a investigar si esta serie es estacionaria o no. Para ello, analizamos la autocorrelación de la variable P mediante una gráfica ACF:

```python
P = serieP.loc[:,"P"]
sm.graphics.tsa.plot_acf(P,title="P- Autocorrelacion")
mplt.show()
```

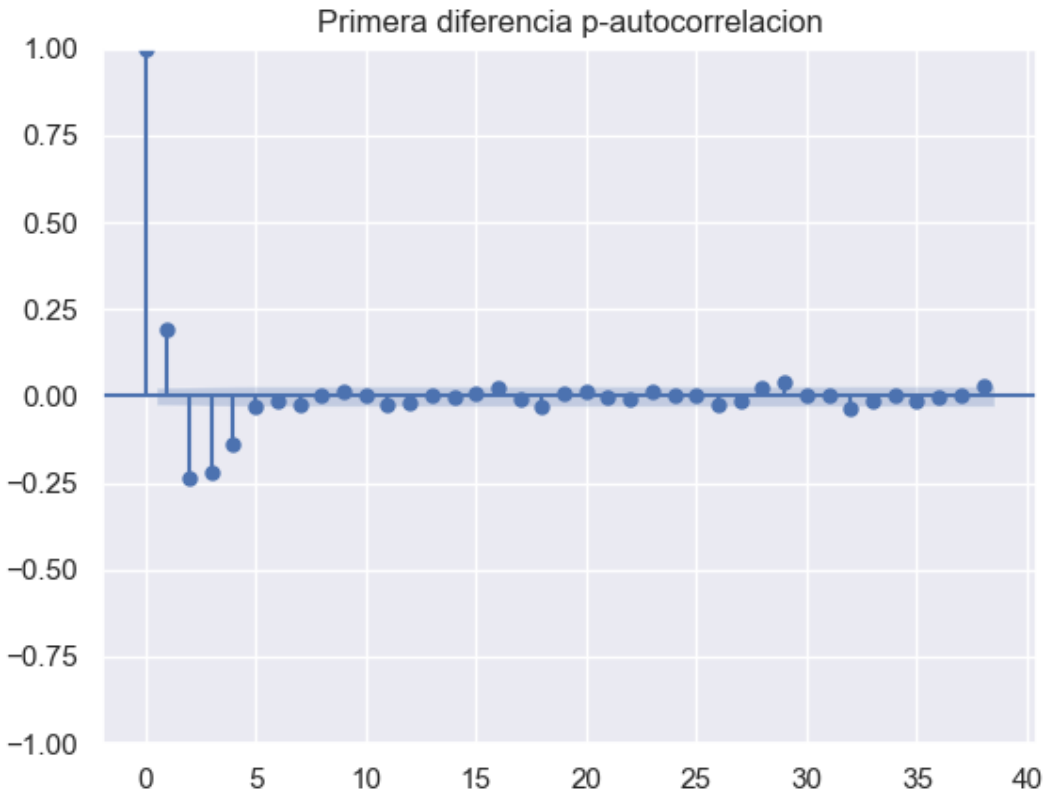
Podemos ver que realmente nuestra variable si describe una serie no estacionaria pues la gráfica AFC va cayendo poco a poco.

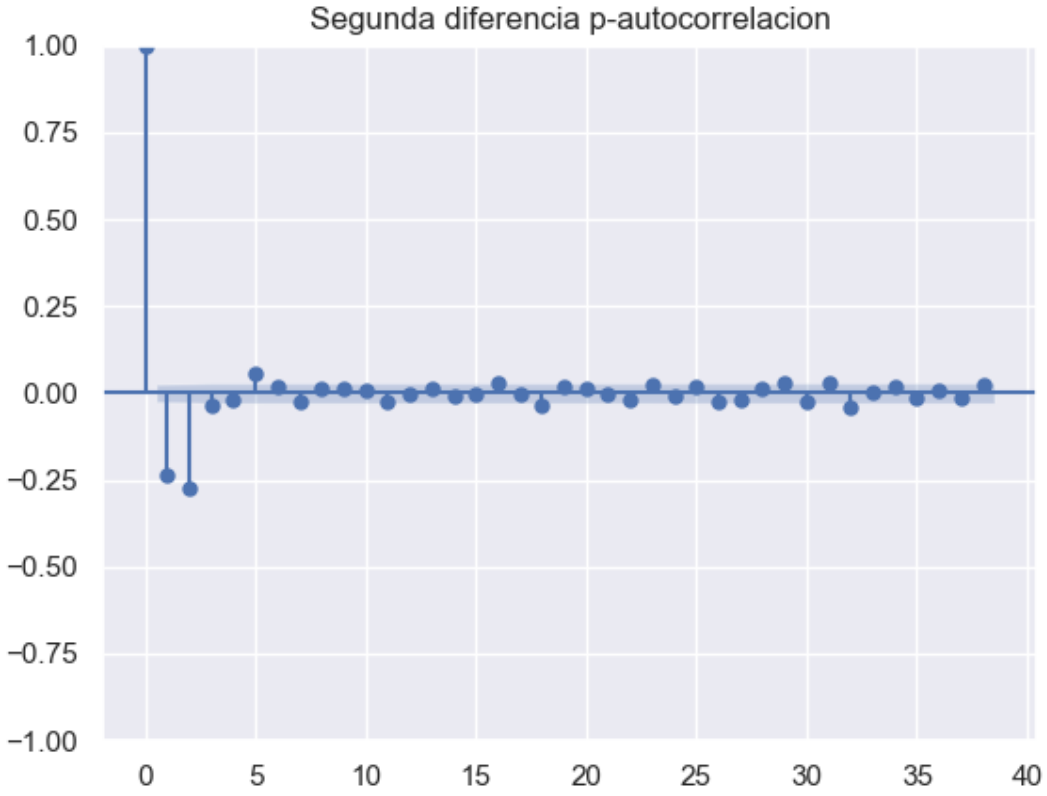


Ahora procedemos a calcular la primera y la segunda diferencia de la variable P para encontrar una serie estacionaria. Para ello, utilizaremos la función diff()de la librería numpy.

```
P_diff_1= np.diff(P)
sm.graphics.tsa.plot_acf(P_diff_1,title="Primera diferencia p-autocorrelacion")
mplt.show()

P_diff_2= np.diff(P_diff_1)
sm.graphics.tsa.plot_acf(P_diff_2,title="Segunda diferencia p-autocorrelacion")
mplt.show()
```

Podemos ver que ninguna gráfica decae poco a poco por lo que podemos decir que ambas son una serie temporal estacionaria de la variable P.


Primera diferencia p-autocorrelacion
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40


Segunda diferencia p-autocorrelacion
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40

Procedemos a descomponer la serie en componentes estacionales y tendencia.

Para conseguirlo realizaremos lo siguiente:

- Importar la libreria statsmodel y usar función seasonal_decompose().
- Importar la librería pandas y usar la función to_Datetime para pasar los datos de la columna fecha de un string a un tipo timestamp
- Utilizar set_index para especificar que el índice de dataframe sea el campo Fecha

```
import statsmodels.api as sm
import pandas as pnd
import seaborn as sns
import matplotlib.pyplot as mplt
import numpy as np

#empezamos lendo o .csv
sns.set()

serieP = pnd.read_csv('P_Santiago_limpio.csv', parse_dates= True)
print(serieP.head(4));
print(serieP.tail(4));
print(serieP.shape);
print(serieP.describe())
serieP['Fecha'] = pnd.to_datetime(serieP['Fecha'])
serieP = serieP.set_index('Fecha')

P = serieP.loc[:,"P"]
```

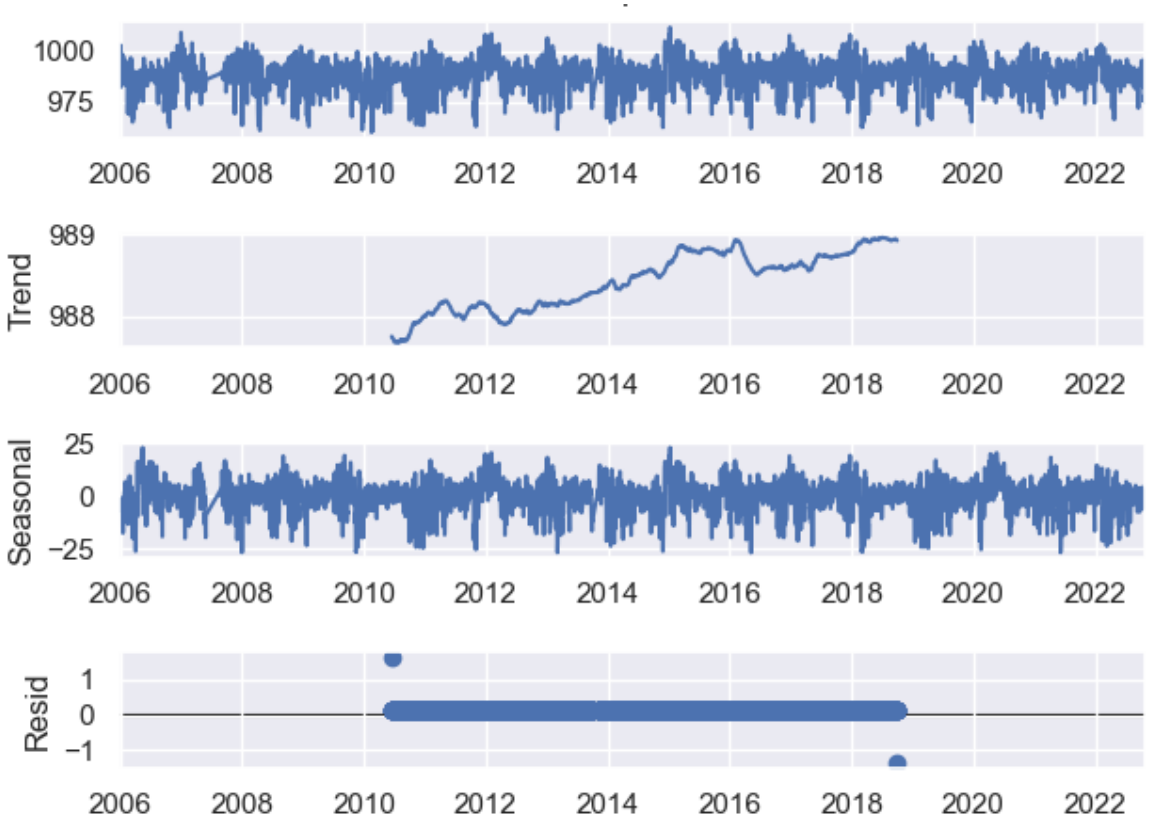
Hacemos dos gráficas utilizando dos veces el seasonal_descompose especificando dos periodos distintos:

- El número de observaciones es de 5930 y necesitamos dos ciclos completos.Así que, hacemos 5390/2= 2965.
- El número de años de nuestro .csv es de 16 años (2006-2022). Necesitamos observar un ciclo por año para observar las gráficas de componentes estacionales así que especificamos 5930 entre 16, es decir, 371.

```
descomposicion_P = sm.tsa.seasonal_decompose(serieP.P, model= 'additive',period=2965)
descomposicion_P.plot()
mplt.show()


descomposicion_P = sm.tsa.seasonal_decompose(serieP.P,model= 'additive',period=371)
descomposicion_P.plot()
mplt.show()
```

Generamos la gráfica con .plot():

Periodo = 2965



Periodo = 371

Se puede observar que la variable P con el periodo 371 es mejor que el periodo 2965.

2.3. Variable IRRA

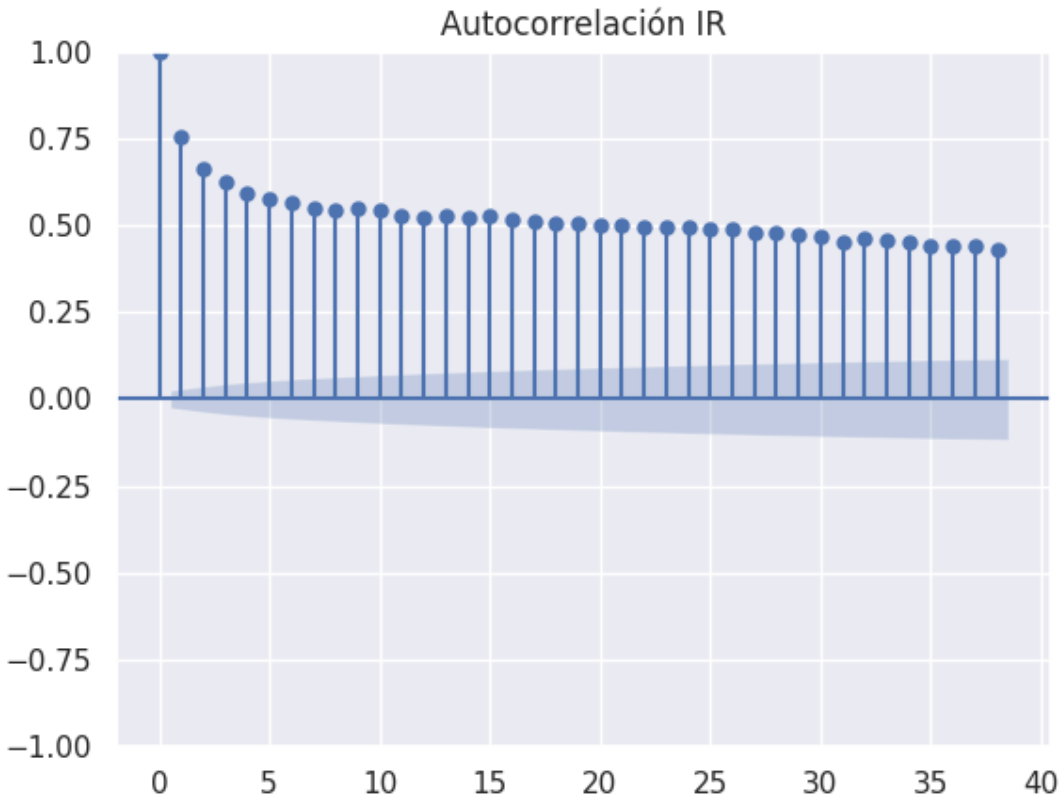
En este caso, procederemos de forma similar a como lo hicimos con las anteriores variables. Para importar los datos, haremos uso de la función read_csv del paquete pandas. Una vez importado, usamos las funciones head y tail para ver el principio y fin de los datos, y comprobar que se hayan importado correctamente, así como shape y describe, para obtener información sobre ellos.

```
import pandas as pd
import numpy as np
import seaborn as sb
from matplotlib import pyplot as plt
from statsmodels import api as st
sb.set()

#Importamos el csv
serie=pd.read_csv('IRRA_Santiago_limpio.csv')

#Vemos el dataframe y un resumen
print(serie.head(4))
print(serie.tail(4))
print(serie.shape)
print(serie.describe())
```

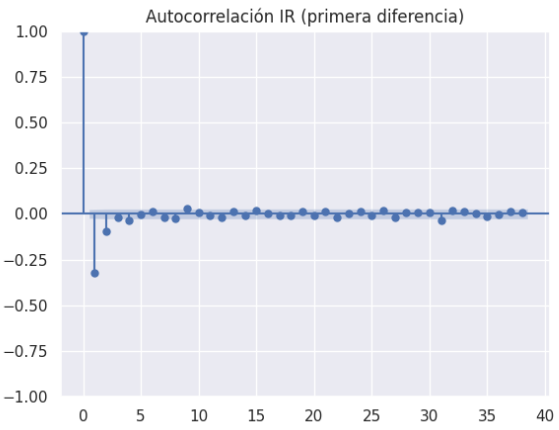
```
        Fecha   IR
0  2006-01-01  477
1  2006-01-02  195
2  2006-01-03  725
3  2006-01-04  870
           Fecha   IR
5959  2022-10-21  798
5960  2022-10-22  321
5961  2022-10-23  468
5962  2022-10-24    5
(5963, 2)
                IR
count  5963.000000
mean   1319.000168
std     819.800843
min       2.000000
25%     632.000000
50%    1168.000000
75%    1976.000000
max    3123.000000
```

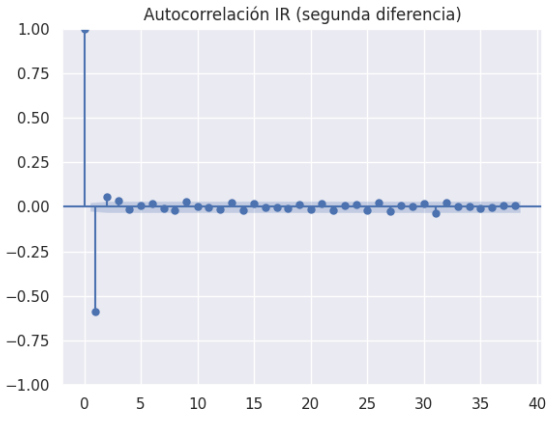
```
#Obtenemos la AFC de la serie, la primera y la segunda diferencia
IR=serie.loc[:,"IR"]
st.graphics.tsa.plot_acf(IR,title='Autocorrelación IR')

IRd1=np.diff(IR)
st.graphics.tsa.plot_acf(IRd1,title='Autocorrelación IR (primera diferencia)')

IRd2=np.diff(IRd1)
st.graphics.tsa.plot_acf(IRd2,title='Autocorrelación IR (segunda diferencia)')

plt.show()
```

A continuación, obtenemos la primera y segunda diferencia de la serie, y calculamos la autocorrelación tanto para la serie, como para la primera y la segunda diferencia. Para calcular esto, tenemos que usar la columna que contiene los datos en sí, que se denomina IR en el csv. Esto lo guardamos en una variable. Para obtener la primera diferencia, simplemente aplicamos la función diff de numpy; para obtener la segunda, aplicamos de nuevo dicha función al resultado anterior.



Así, la autocorrelación de la serie es una propiedad de las series de datos, basada en que los elementos más próximos entre sí (en nuestro caso, en el tiempo) se parecen más entre ellos que elementos más alejados. En nuestro caso, podemos ver que la serie es una serie no estacionaria, pues la gráfica decae lentamente con el tiempo.



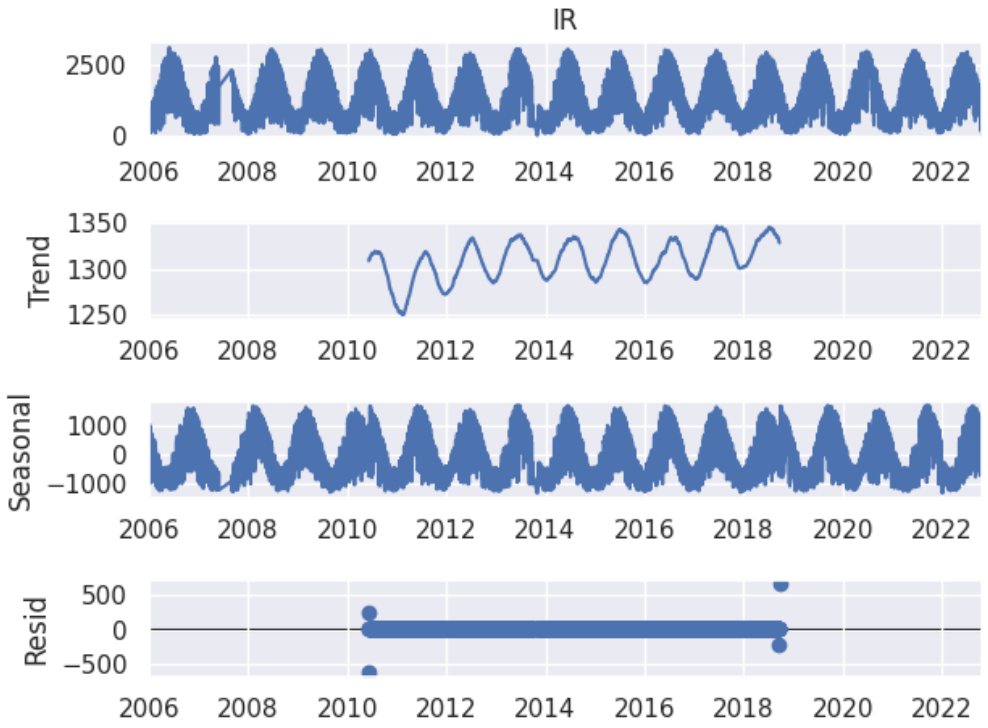


En lo que respecta a la primera y segunda diferencias, podemos ver que no siguen el mismo patrón que la serie: ambas tienen altos y bajos, no decaen lentamente. Por lo tanto, en ambos casos, estaríamos hablando de series temporales estacionarias.

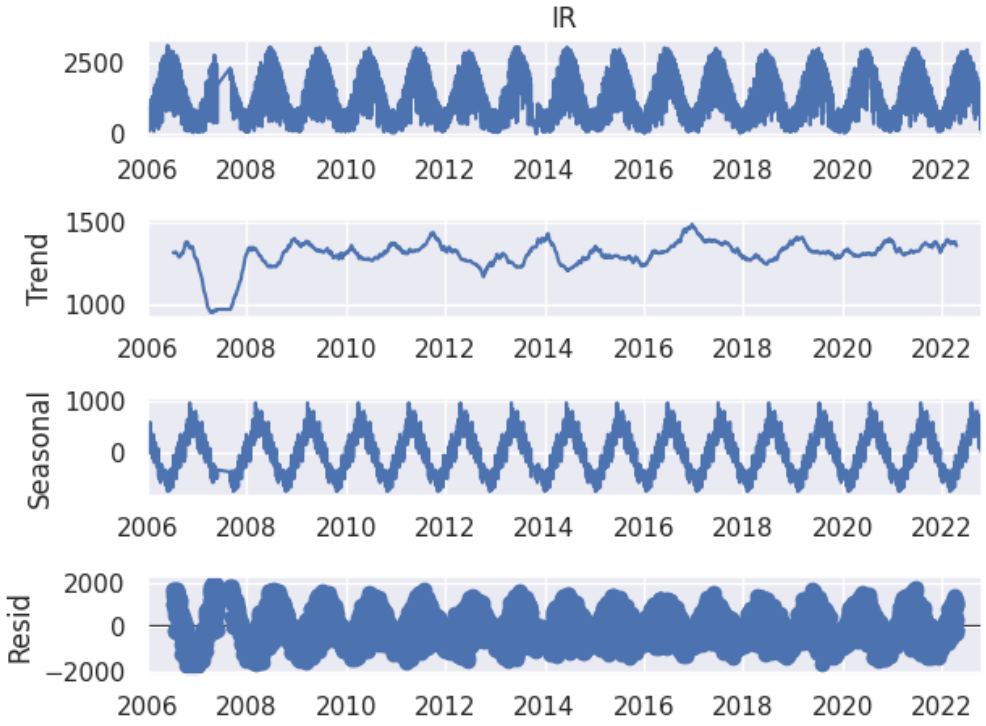
```
#Volvemos a importar el csv parseando las fechas y estableciendo el índice
serie=pd.read_csv('IRRA_Santiago_limpio.csv',index_col=0,parse_dates=['Fecha'])

#Descomponemos la serie
desc=st.tsa.seasonal_decompose(serie.IR,model='additive', period=2981)
desc.plot()

desc=st.tsa.seasonal_decompose(serie.IR,model='additive', period=372)
desc.plot()

plt.show()
```

A continuación, descomponemos la serie temporal en sus componentes estacionales y la tendencia. Previamente a realizar esto, debemos reimportar el csv, para parsear el campo de la fecha a un timestamp, ya que en nuestros datos se considerará como un string. Además, definimos esta columna (la columna 0) como el índice de la serie de datos.

Para descomponer la serie, comenzaremos realizando una aproximación. Como se ve en el primer caso, usamos la descomposición aditiva, representando la serie por la suma de sus componentes. Estos son la serie observada, la tendencia, el componente cíclico o estacional, y el residuo. La tendencia representa el crecimiento o decrecimiento de los valores en general de la serie, y se calcula usando la media móvil de los datos de la serie. El componente estacional, por su parte, representa un patrón observado en intervalos regulares de tiempo. Se calcula eliminando el componente de la tendencia de la serie y, con estos datos, calculando las medias para cada período de tiempo, construyendo el componente temporal a partir de estos datos. Por último, el residuo es el componente irregular que queda tras descomponer la serie.



En este primer caso, para elegir el período, haremos una aproximación, de forma que tengamos dos ciclos completos de datos. Ya que tenemos un total de 5963 observaciones, elegiremos un período de 2981. Se puede observar que la tendencia, pese a seguir un patrón en cierta forma cíclico, aumenta ligeramente con el tiempo, aunque tanto esta como el residuo sólamente contienen datos entre, aproximadamente, mediados de 2010 y mediados de 2018. Sin embargo, tanto en la serie como en el componente estacional, podemos ver que siguen un patrón cíclico que se repite cada año. Por lo tanto, haremos una segunda aproximación al período.



Para esta segunda aproximación, ya que tenemos 5963 observaciones y un total de 16 años (pues nuestros datos van desde 2006 hasta 2022), elegiremos un período de 372. En este caso, obtenemos el resultado de la imagen anterior. Ahora, podemos ver que la tendencia y el componente estacional son mucho más precisas que en el caso anterior. Además, podemos observar un descenso pronunciado de la tendencia en torno al año 2007. Sin embargo, si observamos la serie temporal, esto se debe a la presencia de un gran número de valores atípicos que, al eliminarlos en el primer punto, hacen que para ese intervalo de tiempo no haya datos, y por tanto la tendencia tienda a acercarse a 0.

3. Interpolación lineal del trend

Vamos a interpolar linealmente las tendencias obtenidas en la descomposición de la serie. Dichas interpolaciones las utilizaremos para obtener una línea de regresión y su pendiente media.

3.1. Variable TM

En primer lugar (zona roja del código), interpolamos linealmente la tendencia con las funciones polyfit y poly1d de la librería numpy.

```
47
48  # interpolacion lineal de la tendencia
49  np_Tendencia = np.array(descomposicion_TM.trend)
50  finite_indexes = np.isfinite(np_Tendencia)
51  coeficientes = np.polyfit(np.array(pd.to_datetime(descomposicion_TM.trend.index).astype(int) / 10**18)[finite_indexes],
52                            np_Tendencia[finite_indexes], 1)
53  funcion_regresion = np.poly1d(coeficientes)
54
55  # insercion de columnas tendencia y regresion en el dataframe
56  serieTM.insert(1, 'Tendencia', descomposicion_TM.trend.interpolate(method="linear"))
57  serieTM.insert(2, 'Regresion', funcion_regresion(np.array(pd.to_datetime(serieTM.index).astype(int) / 10**18)))
58
59  # gráfica: interpolacion lineal con regresion lineal
60  plt.plot(serieTM.index, serieTM.Tendencia, serieTM.Regresion)
61
62  #  pendiente de la regresion lineal
63  print(coeficientes)
64
65  plt.show()
```

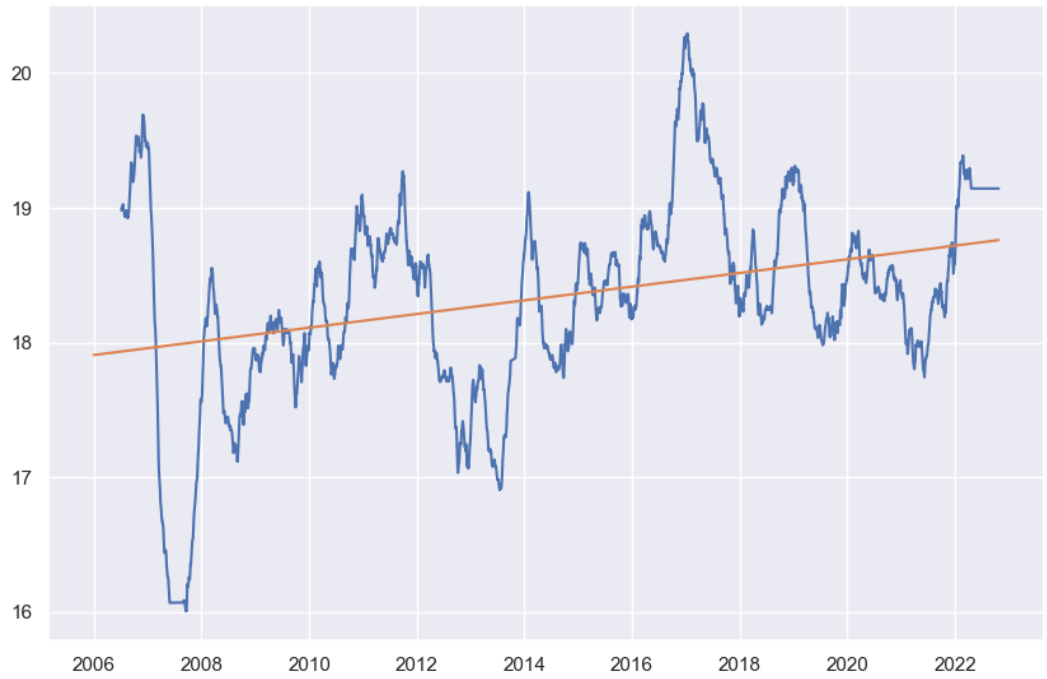
polyfit nos devolverá los coeficientes de la regresión, mientras que poly1d nos devolverá la función de regresión en sí. Pasamos como argumentos la Fecha en modo numérico para el eje X, y los valores de TM para el eje Y, ambos como un vector de numpy.

Para que polyfit funcionase correctamente, hemos tenido que filtrar ambos ejes de manera que no se cogieran los valores NaN. Para ello, hemos obtenido los índices finitos (que no son NaN) en la línea 50, y los hemos aplicado a los ejes en la función polyfit en la línea 51.

A continuación (zona azul del código), hemos insertado dos nuevas columnas en el dataframe, que contienen la tendencia obtenida en la descomposición y la regresión obtenida en la interpolación.

Con ellas, dibujamos un gráfico donde vemos la tendencia junto con la línea de regresión obtenida.

Veamos el gráfico obtenido:



Como podemos ver, el gráfico nos muestra de manera ampliada la tendencia que obtuvimos en la descomposición de la serie temporal, a la par que la línea de regresión que hemos generado.

En la línea de regresión podemos ver cómo la tendencia describe una dirección ascendente según se avanza en el tiempo. Es decir, podemos obtener como conclusión que la variable TM ha tendido a crecer a lo largo de los años.

Vamos a observar (zona verde del código) los coeficientes de la regresión:

```
[ 1.60864345 16.07974316]
```

Si la fórmula la dibujamos como y = a*x + b, el coeficiente a tiene un valor de 1.608 y el coeficiente b tiene un valor de 16.079.

El valor que más nos interesa es el del coeficiente a=1.608, ya que describe la pendiente de la línea de regresión. Al tener un valor positivo, nos confirma lo que habíamos intuido visualmente: la tendencia de la variable TM es ascendente.

3.2. Variable P

```
#Interpolacion linea da tendencia
tend  = np.array(descomposicion_P.trend)
indexes= np.isfinite(tend)
coefs = np.polyfit(np.array(pnd.to_datetime(descomposicion_P.trend.index).astype(int)/10**18)[indexes],tend[indexes],1)
func = np.poly1d(coefs)
```

A través del vector numpy creamos una tendencia de la descomposición estacional quedándonos con aquellos valores que no contenta el valor NaN.

Obtenemos los coeficientes de regresión a través polyfit usando en el eje X la fecha y en el eje los valores de la serie temporal.

Creamos la función de regresión usando poly1d con los coeficientes:

```
#Inserción das columnas da tendencia e regresion do dataframe
serieP.insert(1,'Tendencia', descomposicion_P.trend.interpolate(method='linear'))
serieP.insert(2, 'Regresion', func(np.array(pnd.to_datetime(serieP.index).astype(int)/10**18)))
```

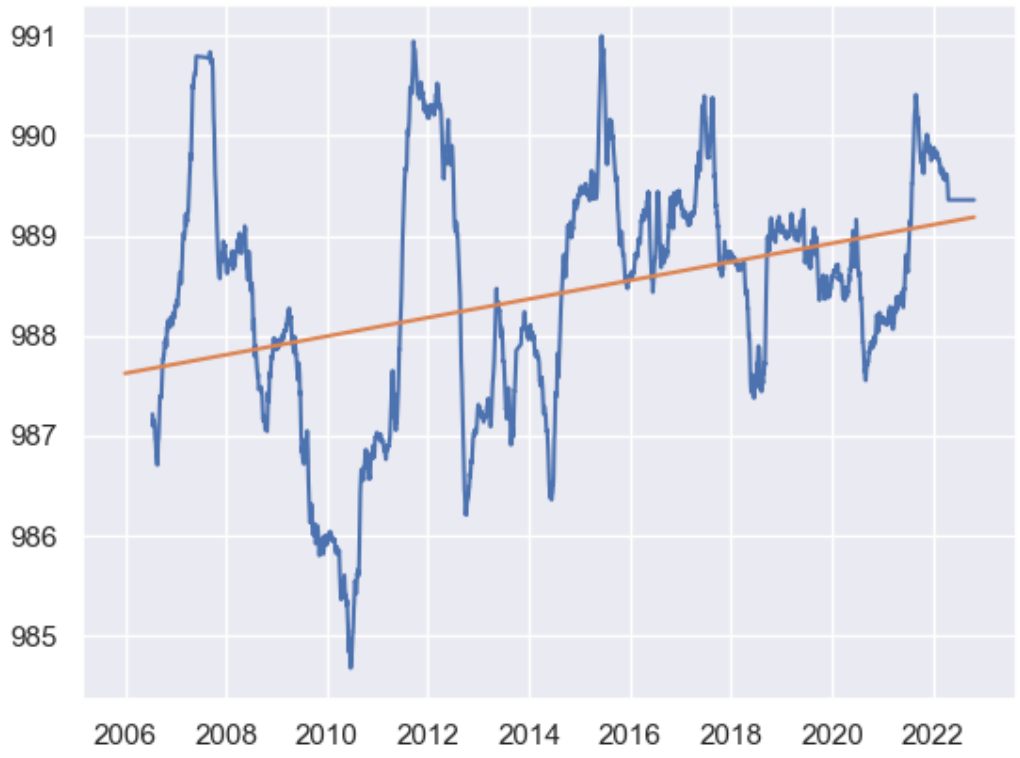
Insertamos los coeficientes y la función como dos nuevas columnas en el dataframe:

```
mplt.plot(serieP.index,serieP.Tendencia, serieP.Regresion)
print(coefs)
mplt.show()
```

Imprimimos los coeficientes:

```
[  2.94297265 984.28168141]
```

Ejecutamos el gráfico para ver la tendencia y la regresión:



El gráfico nos muestra la tendencia de la descomposición de la serie temporal que generamos anteriormente y la línea de regresión.

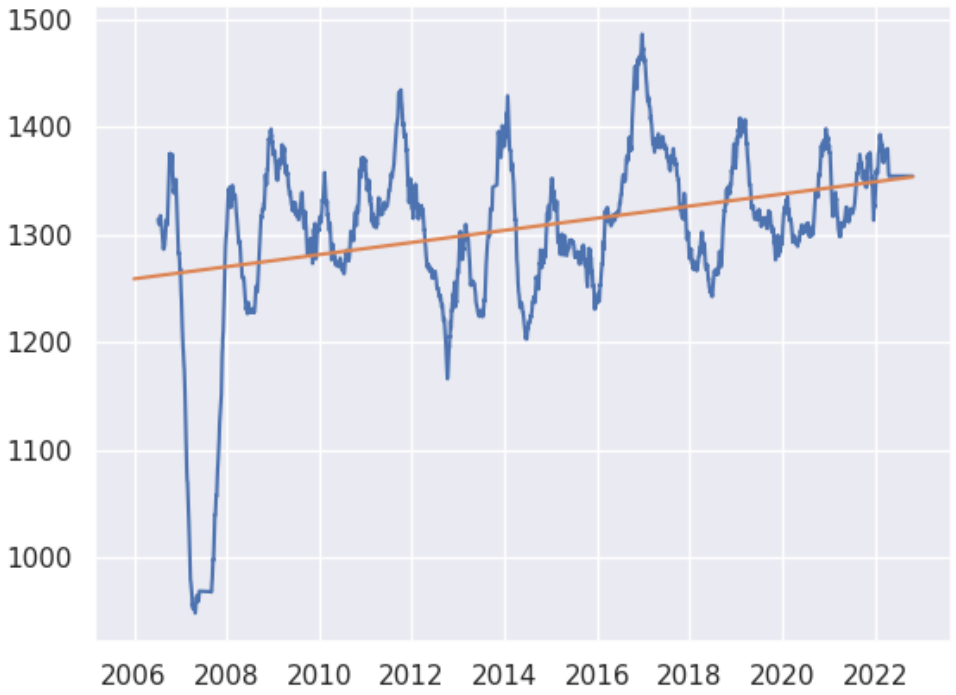
A través de la línea de regresión, podemos ver que la variable P crece a lo largo de los años, de 2006 hasta el 2022, pues existe una pendiente ascendente a lo largo del tiempo.

3.3. Variable IRRA

```
#Hacemos la interpolación linear de la tendencia
tend=np.array(desc.trend)
indexes=np.isfinite(tend)
coefs=np.polyfit(np.array(pd.to_datetime(desc.trend.index).astype(int)/10**18)[indexes],tend[indexes],1)
func=np.poly1d(coefs)


#Insertamos las columnas de tendencia y regresion al dataframe
serie.insert(1,'Tendencia',desc.trend.interpolate(method="linear"))
serie.insert(2,'Regresion',func(np.array(pd.to_datetime(serie.index).astype(int)/10**18)))

plt.plot(serie.index, serie.Tendencia, serie.Regresion)
print(coefs)
plt.show()
```

En este caso, seguimos un proceso similar a los anteriores. En primer lugar, obtenemos la tendencia de la descomposición estacional como un vector de numpy, y filtramos los valores para quedarnos sólo con aquellos que no son NaN. A continuación, usamos la función polyfit para obtener los coeficientes de la regresión, a la cual le pasamos la fecha, en modo numérico, en el eje X; y los valores de la

serie temporal, para el eje Y. Con los coeficientes, y usando la función poly1d, obtenemos la función de regresión. A continuación, insertamos los coeficientes y la función como dos nuevas columnas en el dataframe, y creamos un gráfico que represente la tendencia y la regresión, así como imprimir los coeficientes.



En el gráfico, tenemos, ampliada, la tendencia que obtuvimos en la descomposición anterior, así como la línea de regresión. Podemos ver que, claramente, la tendencia es ascendente, lo cual indica que la irradiación global ha aumentado a lo largo de los años. Esto lo podemos ver también imprimiendo los coeficientes de la regresión. En este caso, el primero, que es el que representa la pendiente de la regresión, es positivo, con lo cual sabemos que la tendencia será creciente.

4. Cálculo de la correlación cruzada entre las series temporales

Vamos a calcular la correlación cruzada entre las series temporales. Lo haremos para las 3 combinaciones de dos variables: TM-IRRA, TM-P e IRRA-P.

El análisis de la correlación cruzada nos permitirá saber si las variaciones en el tiempo de dos series temporales están relacionadas entre sí o no. El valor de la correlación cruzada se encuentra en el rango [-1,1], siendo mayor la correlación según los valores sean más cercanos a 1.

Eso sí, debemos saber que el análisis de la correlación cruzada nos dará información sobre el pasado, pero no podrá usarse nunca para predecir el futuro.

Importación de los dataframe

Comenzamos importando los 3 dataframes. Vamos a establecer el campo fecha como índice de cada serie, a la par que le cambiamos el tipo a timestamp.

```
 9  #-------------------------------------------------------------------------
10  # CREACION DE LOS DATAFRAMES
11
12  # importar CSV
13  serieTM = pd.read_csv('./TM_Santiago_limpio.csv', parse_dates = True)
14  serieP = pd.read_csv('./P_Santiago_limpio.csv', parse_dates = True)
15  serieIRRA = pd.read_csv('./IRRA_Santiago_limpio.csv', parse_dates = True)
16
17  # campo Fecha como índice
18  serieTM['Fecha'] = pd.to_datetime(serieTM['Fecha'])
19  serieTM = serieTM.set_index('Fecha')
20
21  serieP['Fecha'] = pd.to_datetime(serieP['Fecha'])
22  serieP = serieP.set_index('Fecha')
23
24  serieIRRA['Instante_lectura'] = pd.to_datetime(serieIRRA['Instante_lectura'])
25  serieIRRA = serieIRRA.set_index('Instante_lectura')
26
27  # creamos dataframe conjunto
28  df_conjunto = serieTM
29  df_conjunto.insert(0, 'P', serieP.P)
30  df_conjunto.insert(1, 'IRRA', serieIRRA.Irradiacion_global_diaria)
31
32  # ver dataframe
33  print(df_conjunto.head(4))
34  print(df_conjunto.tail(4))
35  print("\n")
36
```

Comprobamos que los datos se han guardado correctamente en los dataframes:

```
PS C:\Users\alex1\Documents\UNIVERSIDAD\AMD\practicas\p6> python .\correlacion_cruzada.py
               P   IRRA    TM
Fecha
2006-01-01   997.6  477.0  11.1
2006-01-02  1002.8  195.0  11.6
2006-01-03  1002.9  725.0  13.6
2006-01-04   997.1  870.0  14.3
               P   IRRA    TM
Fecha
2022-10-21  978.6  798.0  17.3
2022-10-22  975.2  321.0  17.0
2022-10-23  976.6  468.0  17.0
2022-10-24  983.1    5.0  14.3


PS C:\Users\alex1\Documents\UNIVERSIDAD\AMD\practicas\p6>
```

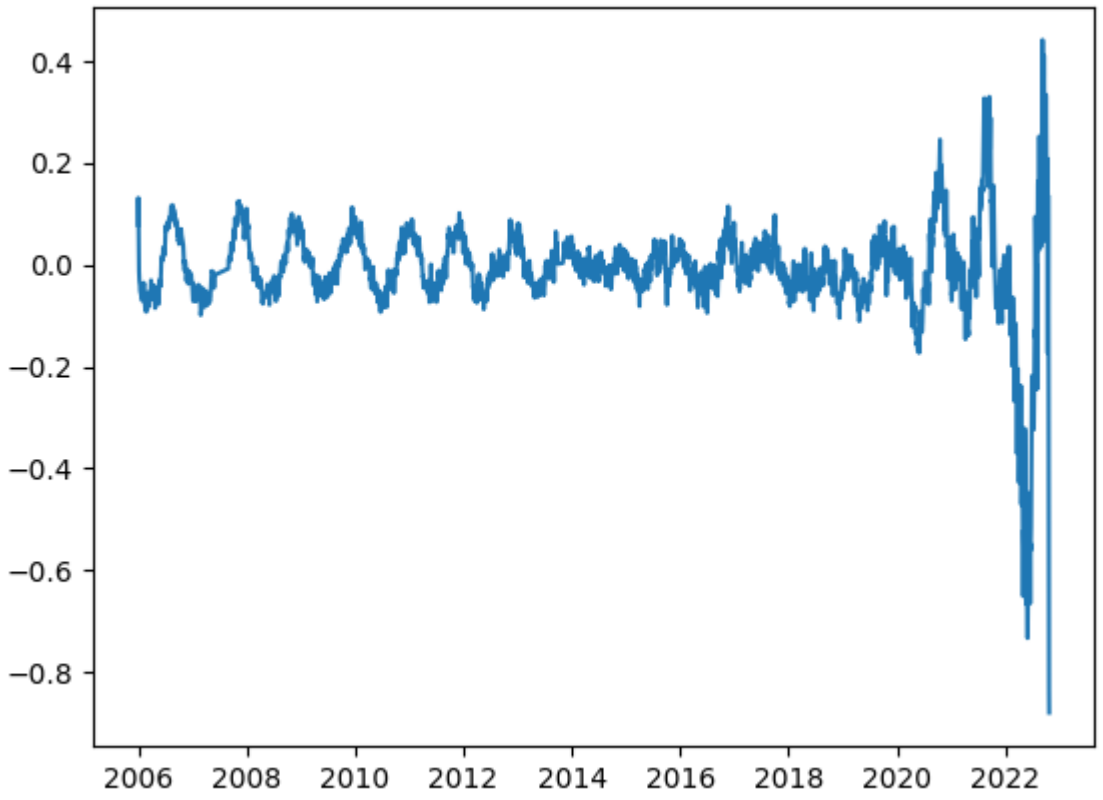
Correlación TM-P

Para la correlación cruzada entre las series de TM y P, obtenemos los índices donde no haya valores NaN en ninguna de las series.

Utilizamos la función ccf de statsmodels, pasando las series filtradas por los índices que hemos obtenido previamente.

```
37  #-------------------------------------------------------------------------
38  # CORRELACIONES CRUZADAS
39
40  # índices sin NaN
41  indexes_TM_P = np.isfinite(df_conjunto.TM) & np.isfinite(df_conjunto.P)
42
43  # cálculo de correlaciones cruzadas
44  ccf_TM_P = sm.tsa.stattools.ccf(df_conjunto.TM[indexes_TM_P], df_conjunto.P[indexes_TM_P])
45
46  # graficamos la correlación cruzada
47  plt.plot(ccf_TM_P)
48
49  plt.show()
50
```

Vemos el gráfico de la correlación cruzada:



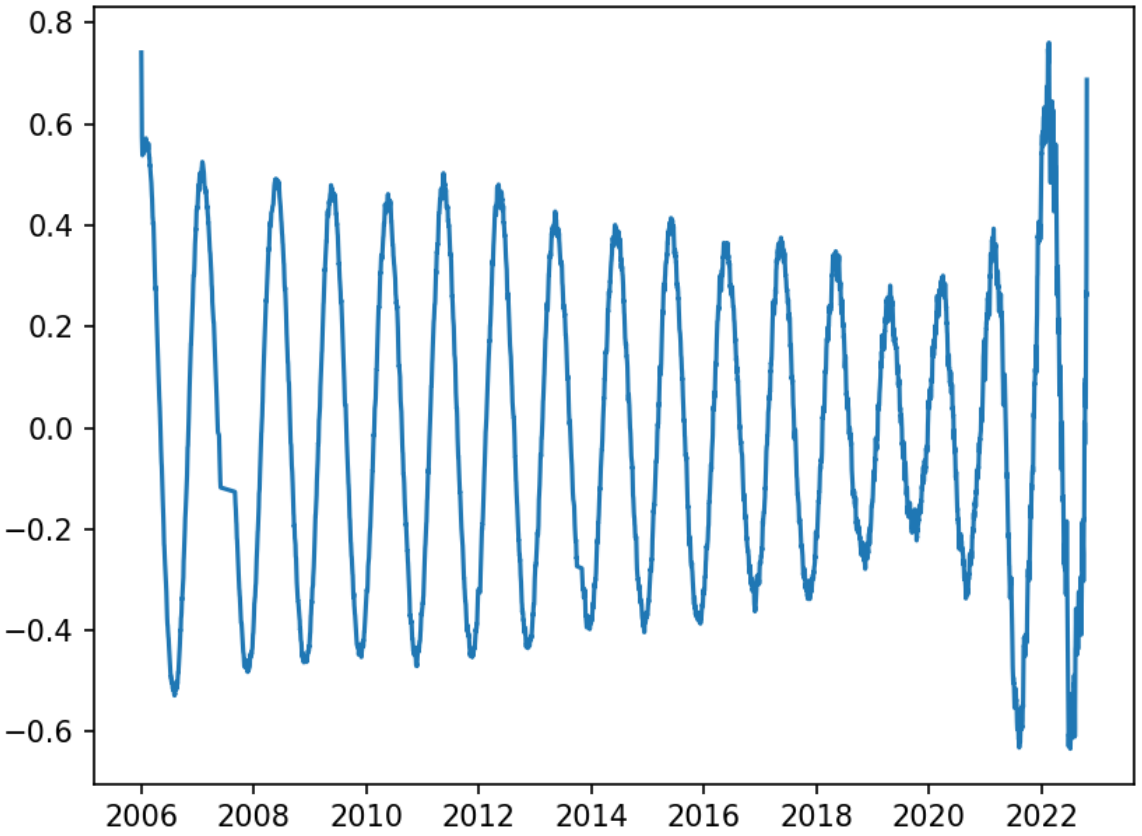
Como podemos observar, la gráfica nos muestra cómo la correlación entre las series temporales ha tendido a ser prácticamente nula desde 2006 hasta 2020. Sin embargo, en estos últimos dos años, se ha vivido un incremento en la oscilación de la correlación, haciendo que las variables estén más relacionadas entre sí en las épocas donde tienden a estar relacionadas, y menos relacionadas en las épocas donde tienden a no estar relacionadas.

Correlación cruzada TM-IRRA y P-IRRA

Hacemos el mismo proceso para la correlación cruzada entre las series de TM e IRRA y las de P e IRRA.

```
#-------------------------------------------------------------------------
# CORRELACIONES CRUZADAS

# índices sin NaN
indexes_TM_P = np.isfinite(df_conjunto.TM) & np.isfinite(df_conjunto.P)
indexes_TM_IRRA = np.isfinite(df_conjunto.TM) & np.isfinite(df_conjunto.IRRA)
indexes_P_IRRA = np.isfinite(df_conjunto.P) & np.isfinite(df_conjunto.IRRA)


# cálculo de correlaciones cruzadas
ccf_TM_P = sm.tsa.stattools.ccf(df_conjunto.TM[indexes_TM_P], df_conjunto.P[indexes_TM_P])
ccf_TM_IRRA = sm.tsa.stattools.ccf(df_conjunto.TM[indexes_TM_IRRA], df_conjunto.IRRA[indexes_TM_IRRA])
ccf_P_IRRA = sm.tsa.stattools.ccf(df_conjunto.P[indexes_TM_P], df_conjunto.IRRA[indexes_TM_P])

# graficamos la correlación cruzada
plt.plot(ccf_TM_P)
plt.plot(ccf_TM_IRRA)
plt.plot(ccf_P_IRRA)

plt.show()
```

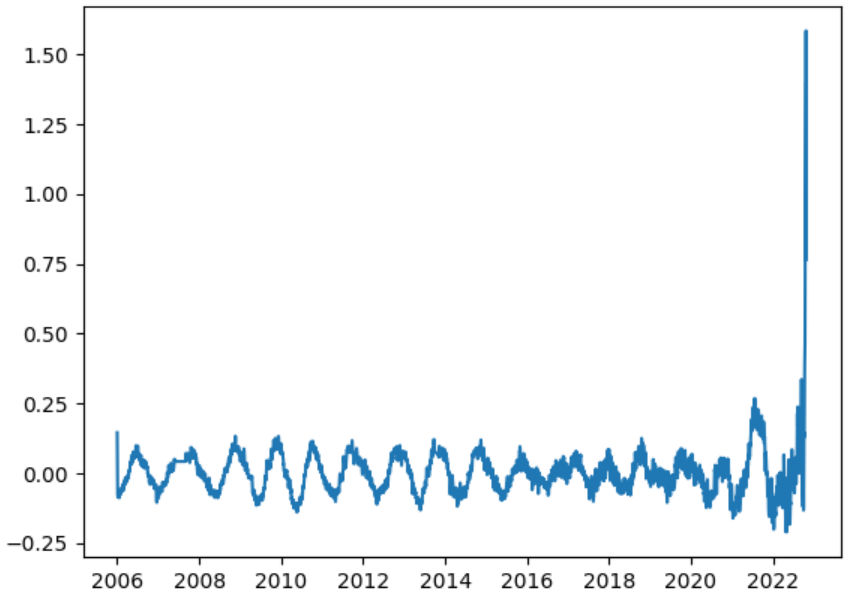
Observamos la gráfica de correlación cruzada entre TM e IRRA:



En esta ocasión, la correlación es más acentuada que en el caso anterior. Aquí podemos ver cómo desde 2006 hasta 2020 la correlación, pese a ir disminuyendo muy ligeramente, se mantenía en un valor razonable. En 2020 el rango se había reducido ya de [-0.6,0.6] a [-0.4,0.4]. Sin embargo, en estos últimos años, la correlación se ha disparado, de manera

que se ha alcanzado un rango de [-0.8,0.8]. Esto quiere decir que, en la época del año donde las variables tienden a estar correlacionadas, lo hacen de manera bastante fuerte, y en la época donde las variables están inversamente correlacionadas, dicha correlación inversa es también bastante notable.

Por último, vamos a ver la gráfica de correlación cruzada entre las series P e IRRA:



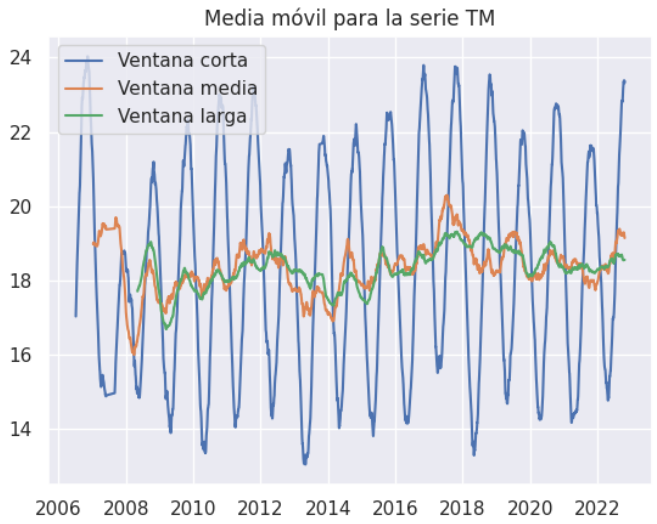
En este caso, vemos cómo la correlación entre las variables ha sido prácticamente nula durante todos los años. Es cierto que en 2021 y 2022 se ha acentuado ligeramente la oscilación, sin llegar a ser una correlación notable en ninguno de los sentidos. También podemos ver cómo para el último tramo de este año se ha alcanzado una correlación de 1.50, lo cual se deba seguramente a algún error de cálculo o de ausencia de datos en alguna de las series.

5. Filtro en tendencia con una media móvil

Para filtrar los datos con la media móvil, primero tenemos que definir las ventanas de tiempo con las que trabajaremos. En nuestro caso, teníamos un período de 375, porque tenemos 6003 observaciones a lo largo de 16 años. Por tanto, la primera de las ventanas que definiremos será correspondiente a la mitad del período (es decir, medio año). La segunda será el período entero, y la tercera, el doble del período. De esta forma, las ventanas quedan definidas como en la imagen siguiente.

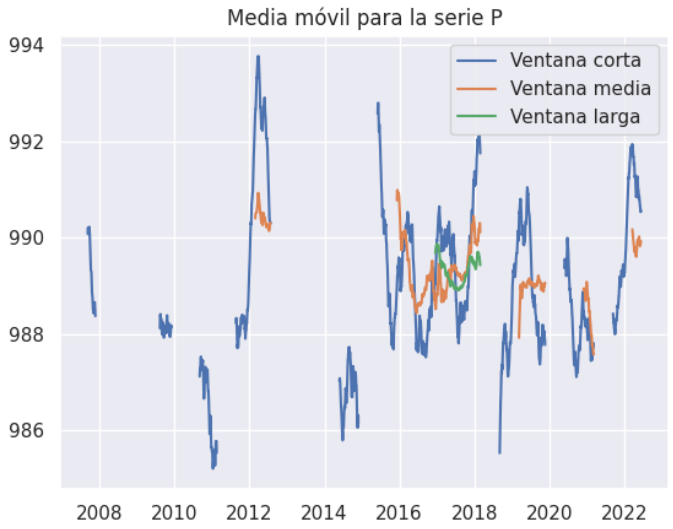
```
#-------------------------------------------------------
#---------Filtro en tendencia con media móvil---------
#Establecemos las ventanas para la media móvil
ventana0=math.floor(375/2)
ventana1=375
ventana2=375*2

#Calculamos las medias para la primera variable, con las tres ventanas
meanTM0=df_conjunto.TM.rolling(ventana0).mean()
meanTM1=df_conjunto.TM.rolling(ventana1).mean()
meanTM2=df_conjunto.TM.rolling(ventana2).mean()

#Graficamos las medias
plt.title("Media móvil para la serie TM")
plt.plot(meanTM0, label='Ventana corta')
plt.plot(meanTM1, label='Ventana media')
plt.plot(meanTM2, label='Ventana larga')
plt.legend()
plt.show()
```

Por lo tanto, para calcular las medias, simplemente aplicaremos la función `.rolling(ventana).mean()` al conjunto de datos, donde sustituiremos ventana por cada una de las tres ventanas que hemos definido. Con estos datos, los graficamos, representando en la misma gráfica la media obtenida con las tres ventanas. De esta forma, obtenemos el siguiente gráfico para la media de la serie TM.



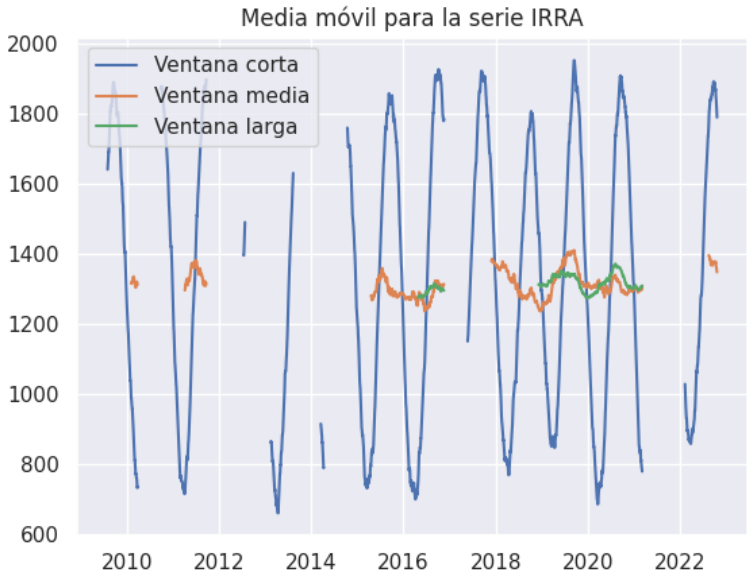
Para la serie P, el proceso seguido es similar: calculamos la media para las tres ventanas, y la graficamos. En el caso de esta gráfica, vemos que la media falla en bastantes valores intermedios. Esto se debe a la presencia de bastantes valores atípicos que, al eliminarlos, y al elegir unas ventanas bastante amplias para la media, provocan que falle en algunos datos.

```
#Calculamos las medias para la tercera variable, con las tres ventanas
meanP0=df_conjunto.P.rolling(ventana0).mean()
meanP1=df_conjunto.P.rolling(ventana1).mean()
meanP2=df_conjunto.P.rolling(ventana2).mean()

#Graficamos las medias
plt.title("Media móvil para la serie P")
plt.plot(meanP0, label='Ventana corta')
plt.plot(meanP1, label='Ventana media')
plt.plot(meanP2, label='Ventana larga')
plt.legend()
plt.show()
```



Para la serie IRRA, igual que en las otras dos, hacemos el mismo proceso. Al igual que en el caso anterior, también vemos que la media, debido a la presencia de valores atípicos, falla en algunos casos.

```
#Calculamos las medias para la segunda variable, con las tres ventanas
meanIRRA0=df_conjunto.IRRA.rolling(ventana0).mean()
meanIRRA1=df_conjunto.IRRA.rolling(ventana1).mean()
meanIRRA2=df_conjunto.IRRA.rolling(ventana2).mean()

#Graficamos las medias
plt.title("Media móvil para la serie IRRA")
plt.plot(meanIRRA0, label='Ventana corta')
plt.plot(meanIRRA1, label='Ventana media')
plt.plot(meanIRRA2, label='Ventana larga')
plt.legend()
plt.show()
```


Media móvil para la serie IRRA
Ventana corta
Ventana media
Ventana larga
2000
1800
1600
1400
1200
1000
800
600
2010
2012
2014
2016
2018
2020
2022

6. Realización de predicción

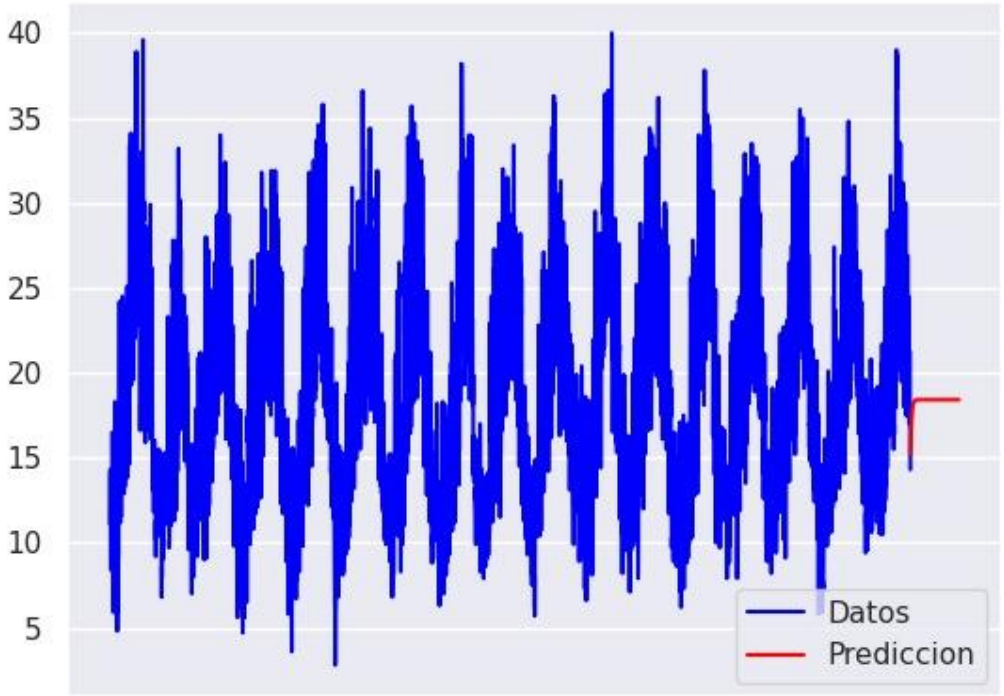
Para realizar La predicion vamos a utilizar la libreria SARIMAX.

```
#-------------------------------------------------------
#----------------------Predicción-----------------------
#Realizamos la predicción para la serie TM
modTM=sm.tsa.SARIMAX(df_conjunto.TM, order=(1,0,0), trend='c')
resTM=modTM.fit()
fcstTM=resTM.forecast(steps=365*2)
df_final_TM=pd.concat([df_conjunto.TM,fcstTM],ignore_index=True)

df_final_TM.iloc[0:6003].plot(color='blue',label='Datos')
df_final_TM.iloc[6004:6368].plot(color='red',label='Prediccion')
plt.gca().get_xaxis().set_visible(False)
plt.legend()
plt.show()

print(df_final_TM)
```

Usamos la función SARIMAX definiendo un order 1,0,0, trend c y usando los datos TM. Luego, metemos en el df_final los valores de la prediccion (dataframa) Los resultados finales de la predicción son los siguientes:

```
6004    15.233924
6005    15.618101
6006    15.955877
6007    16.252858
6008    16.513971
           ...
6363    18.414787
6364    18.414787
6365    18.414787
6366    18.414787
6367    18.414787
```

Creamos una gráfica df_final definiendo dos líneas. La primera va a definir los datos en color azul y la segunda va definir la predicción en color rojo. Finalmente, mostramos la gráfica añadiendo una leyenda.


40
35
30
25
20
15
10
5
Datos
Prediccion