In [543…
```
# Librerías básicas
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set()
```

El módulo Statsmodels

Statsmodels es un módulo de Python que ofrece clases y funciones de varios modelos estadísticos, así como pruebas de hipótesis y análisis exploratorio de datos. La documentación se encuentra en statsmodels.org.

In [544…
```
import statsmodels.graphics.tsaplots as sgt
import statsmodels.tsa.stattools as sts
from statsmodels.tsa.seasonal import seasonal_decompose
```

Fuentes de datos interesantes

API de Yahoo finance

Yahoo Finance tiene una API que permite descargar información financiera para realizar análisis. La página web original de la documentación ya no existe. Sin embargo, existen algunas páginas de documentación no oficial como este repositorio en GitHub, o esta página de documentación the ReadTheDocs .

In [545…
```
# Instalación de yfinance
%pip install yfinance
```

```
Defaulting to user installation because normal site-packages is not writeableNote: yo
u may need to restart the kernel to use updated packages.

Requirement already satisfied: yfinance in c:\users\lenovo\appdata\roaming\python\pyt
hon39\site-packages (0.2.43)
Requirement already satisfied: pandas>=1.3.0 in c:\programdata\anaconda3\lib\site-pac
kages (from yfinance) (1.4.4)
Requirement already satisfied: platformdirs>=2.0.0 in c:\programdata\anaconda3\lib\si
te-packages (from yfinance) (2.5.2)
Requirement already satisfied: multitasking>=0.0.7 in c:\users\lenovo\appdata\roaming
\python\python39\site-packages (from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in c:\programdata\anaconda3\lib\site-packa
ges (from yfinance) (4.9.1)
Requirement already satisfied: numpy>=1.16.5 in c:\users\lenovo\appdata\roaming\pytho
n\python39\site-packages (from yfinance) (1.24.3)
Requirement already satisfied: pytz>=2022.5 in c:\users\lenovo\appdata\roaming\python
\python39\site-packages (from yfinance) (2024.1)
Requirement already satisfied: frozendict>=2.3.4 in c:\users\lenovo\appdata\roaming\p
ython\python39\site-packages (from yfinance) (2.4.4)
Requirement already satisfied: html5lib>=1.1 in c:\users\lenovo\appdata\roaming\pytho
n\python39\site-packages (from yfinance) (1.1)
Requirement already satisfied: requests>=2.31 in c:\users\lenovo\appdata\roaming\pyth
on\python39\site-packages (from yfinance) (2.32.3)
Requirement already satisfied: peewee>=3.16.2 in c:\users\lenovo\appdata\roaming\pyth
on\python39\site-packages (from yfinance) (3.17.6)
Requirement already satisfied: beautifulsoup4>=4.11.1 in c:\programdata\anaconda3\lib
\site-packages (from yfinance) (4.11.1)
Requirement already satisfied: soupsieve>1.2 in c:\programdata\anaconda3\lib\site-pac
kages (from beautifulsoup4>=4.11.1->yfinance) (2.3.1)
Requirement already satisfied: six>=1.9 in c:\programdata\anaconda3\lib\site-packages
(from html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in c:\programdata\anaconda3\lib\site-pack
ages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\programdata\anaconda3\lib
\site-packages (from pandas>=1.3.0->yfinance) (2.8.2)
Requirement already satisfied: idna<4,>=2.5 in c:\programdata\anaconda3\lib\site-pack
ages (from requests>=2.31->yfinance) (3.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\programdata\anaconda3\l
ib\site-packages (from requests>=2.31->yfinance) (2.0.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\programdata\anaconda3\lib\sit
e-packages (from requests>=2.31->yfinance) (1.26.11)
Requirement already satisfied: certifi>=2017.4.17 in c:\programdata\anaconda3\lib\sit
e-packages (from requests>=2.31->yfinance) (2023.7.22)
```

In [ ]:

In [546…

```
# Se importa la librería yfinance
import yfinance
```

In [547…

```
# Se descargan los datos S&P500 y Nikkei225
df_yfinance_raw = yfinance.download(tickers = "^GSPC ^N225", #Las series de tiempo de
                                    start = "1994-01-07", #Fecha inicial
                                    end = "2024-08-27", #Fecha final
                                    interval = "1d", #Frecuencia.
                                    group_by = 'ticker', #Criterio de agrupación. Usualmente
                                    auto_adjust = True#,
                                    #threads = True
                                     ) #.
```

```
[**********100%**********]  2 of 2 completed
```

In [548…
```
df_yfinance_raw.head()
```

Out[548]:

Ticker					^N225		
Price	Open	High	Low	Close	Volume	Open	H
Date							
1994-01-07 00:00:00+00:00	17842.980469	18131.410156	17787.480469	18124.009766	0.0	467.089996	470.260
1994-01-10 00:00:00+00:00	18186.519531	18567.060547	18186.519531	18443.439453	0.0	469.899994	475.269
1994-01-11 00:00:00+00:00	18481.849609	18671.669922	18373.039062	18485.250000	0.0	475.269989	475.279
1994-01-12 00:00:00+00:00	18447.339844	18807.080078	18301.929688	18793.880859	0.0	474.130005	475.059
1994-01-13 00:00:00+00:00	18770.380859	18823.380859	18548.750000	18577.259766	0.0	474.170013	474.170

In [549…
```
# Se crea una copia para modificar
df_yfinance = df_yfinance_raw.copy()
```

In [550…
```
# Se agregan columnas para el nivel de cierre de los índices S&P500 y Nikkei255
df_yfinance['spx'] = df_yfinance['^GSPC'].Close
df_yfinance['nikkei'] = df_yfinance['^N225'].Close
```

In [551…
```
#f_comp = df_comp.iloc[1:] # Removing the first elements, since we always start 1 peri
del df_yfinance['^N225'] # Se retiran los grupos de columnas '^N225' y '^GSPC'
del df_yfinance['^GSPC']
df_yfinance=df_yfinance.asfreq('b') # Se establece la frecuencia de los datos
df_yfinance=df_yfinance.fillna(method='ffill') # Se rellenan los datos faltantes con e
```

In [552…
```
df_yfinance.head()
```

Out[552]:

Ticker	spx	nikkei
Price		
Date		
1994-01-07 00:00:00+00:00	469.899994	18124.009766
1994-01-10 00:00:00+00:00	475.269989	18443.439453
1994-01-11 00:00:00+00:00	474.130005	18485.250000
1994-01-12 00:00:00+00:00	474.170013	18793.880859
1994-01-13 00:00:00+00:00	472.470001	18577.259766

In [553…
```
df_yfinance.columns
```

Out[553]:
```
MultiIndex([(   'spx', ''),
            ('nikkei', '')],
           names=['Ticker', 'Price'])
```

In [554…
```
df_yfinance.columns=['spx','nikkei']
```

In [555…
```
df_yfinance.columns
```

Out[555]:
```
Index(['spx', 'nikkei'], dtype='object')
```

In [556…
```
df_yfinance.head()
```

Out[556]:

Date	spx	nikkei
1994-01-07 00:00:00+00:00	469.899994	18124.009766
1994-01-10 00:00:00+00:00	475.269989	18443.439453
1994-01-11 00:00:00+00:00	474.130005	18485.250000
1994-01-12 00:00:00+00:00	474.170013	18793.880859
1994-01-13 00:00:00+00:00	472.470001	18577.259766

In [557…
```
df_yfinance.tail() # Making sure of the last day we're including in the series
```

Out[557]:

Date	spx	nikkei
2024-08-20 00:00:00+00:00	5597.120117	38062.921875
2024-08-21 00:00:00+00:00	5620.850098	37951.800781
2024-08-22 00:00:00+00:00	5570.640137	38211.011719
2024-08-23 00:00:00+00:00	5634.609863	38364.269531
2024-08-26 00:00:00+00:00	5616.839844	38110.218750

In [ ]:
```
```

In [558…
```
#df_comp.date = pd.to_datetime(df_comp.date, dayfirst = True)
#df_comp.set_index("date", inplace=True)
df_yfinance=df_yfinance.asfreq('b')
df_yfinance=df_yfinance.fillna(method='ffill')
```

In [559…
```
df_yfinance.head()
```

Out[559]:

Date	spx	nikkei
1994-01-07 00:00:00+00:00	469.899994	18124.009766
1994-01-10 00:00:00+00:00	475.269989	18443.439453
1994-01-11 00:00:00+00:00	474.130005	18485.250000
1994-01-12 00:00:00+00:00	474.170013	18793.880859
1994-01-13 00:00:00+00:00	472.470001	18577.259766

Algunos archivos csv con datos

En este repositorio de GitHub se encuentran muchos archivos .csv con datos apropiados para ejercicios de aprendizaje de máquina y series de tiempo. En este notebook utilizaremos el archivo 'airline_passengers.csv'.

In [560…
```
df_airline = pd.read_csv('airline_passengers.csv',index_col='Month',parse_dates=True)
```

In [561…
```
df_airline.head()
```

Out[561]:

Month	Passengers
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121

Datos que vienen con la librería statsmodels

La librería statsmodels tiene varios conjuntos de datos incorporados. En este notebook, utilizaremos el dataset 'macrodata', que trae datos macroeconómicos de EEUU.

In [562…
```
#import pandas as pd
import statsmodels.api as sm
df_macrodata = sm.datasets.macrodata.load_pandas().data
df_macrodata.index = pd.Index(sm.tsa.datetools.dates_from_range('1959Q1', '2009Q3'))
print(sm.datasets.macrodata.NOTE)
```

```
::
    Number of Observations - 203

    Number of Variables - 14

    Variable name definitions::

        year      - 1959q1 - 2009q3
        quarter   - 1-4
        realgdp   - Real gross domestic product (Bil. of chained 2005 US$,
                    seasonally adjusted annual rate)
        realcons  - Real personal consumption expenditures (Bil. of chained
                    2005 US$, seasonally adjusted annual rate)
        realinv   - Real gross private domestic investment (Bil. of chained
                    2005 US$, seasonally adjusted annual rate)
        realgovt  - Real federal consumption expenditures & gross investment
                    (Bil. of chained 2005 US$, seasonally adjusted annual rate)
        realdpi   - Real private disposable income (Bil. of chained 2005
                    US$, seasonally adjusted annual rate)
        cpi       - End of the quarter consumer price index for all urban
                    consumers: all items (1982-84 = 100, seasonally adjusted).
        m1        - End of the quarter M1 nominal money stock (Seasonally
                    adjusted)
        tbilrate  - Quarterly monthly average of the monthly 3-month
                    treasury bill: secondary market rate
        unemp     - Seasonally adjusted unemployment rate (%)
        pop       - End of the quarter total population: all ages incl. armed
                    forces over seas
        infl      - Inflation rate (ln(cpi_{t}/cpi_{t-1}) * 400)
        realint   - Real interest rate (tbilrate - infl)
```

In [563…

```
df_macrodata.head()
```

Out[563]:

	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	cpi	m1	tbilrate	unemp
1959-03-31	1959.0	1.0	2710.349	1707.4	286.898	470.045	1886.9	28.98	139.7	2.82	5.8
1959-06-30	1959.0	2.0	2778.801	1733.7	310.859	481.301	1919.7	29.15	141.7	3.08	5.1
1959-09-30	1959.0	3.0	2775.488	1751.8	289.226	491.260	1916.4	29.35	140.5	3.82	5.3
1959-12-31	1959.0	4.0	2785.204	1753.7	299.356	484.052	1931.3	29.37	140.0	4.33	5.6
1960-03-31	1960.0	1.0	2847.699	1770.5	331.722	462.199	1955.5	29.54	139.6	3.50	5.2

Datos de simulados de ventas

In [564…

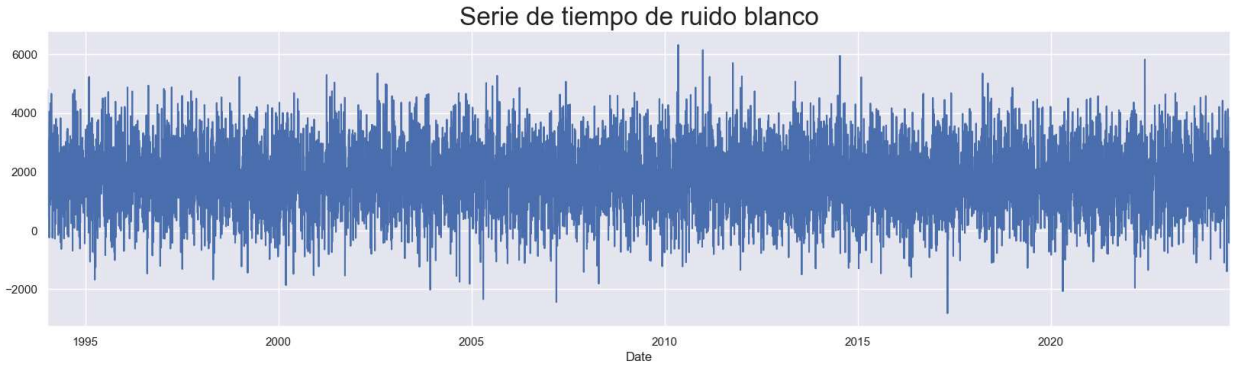
```
df_ventas = pd.read_excel('Datos_clase_1.xlsx',index_col = 'Fecha', parse_dates = True
```

Ruido blanco

In [565...
```
wn = np.random.normal(loc = df_yfinance['spx'].mean(), scale = df_yfinance['spx'].std(
```

In [566...
```
df_yfinance['wn']=wn
```

In [567...
```
df_yfinance.wn.plot(figsize = (20,5))
plt.title("Serie de tiempo de ruido blanco", size= 24)
plt.show()
```



Caminata aleatoria

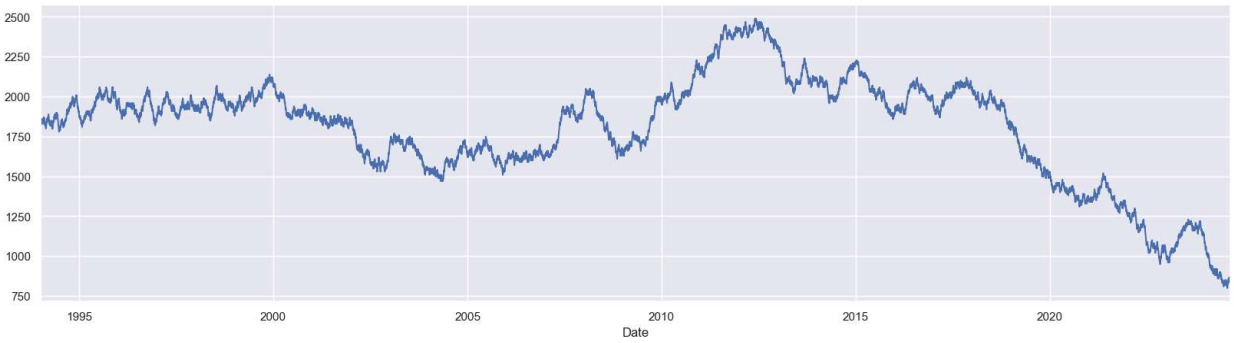
In [568...
```
steps = np.random.choice([-10, 10], size = len(df_yfinance))
```

In [569...
```
steps[0]=0
```

In [570...
```
rw = df_yfinance['spx'].mean()+np.cumsum(steps)
```

In [571...
```
df_yfinance['rw']=rw
```

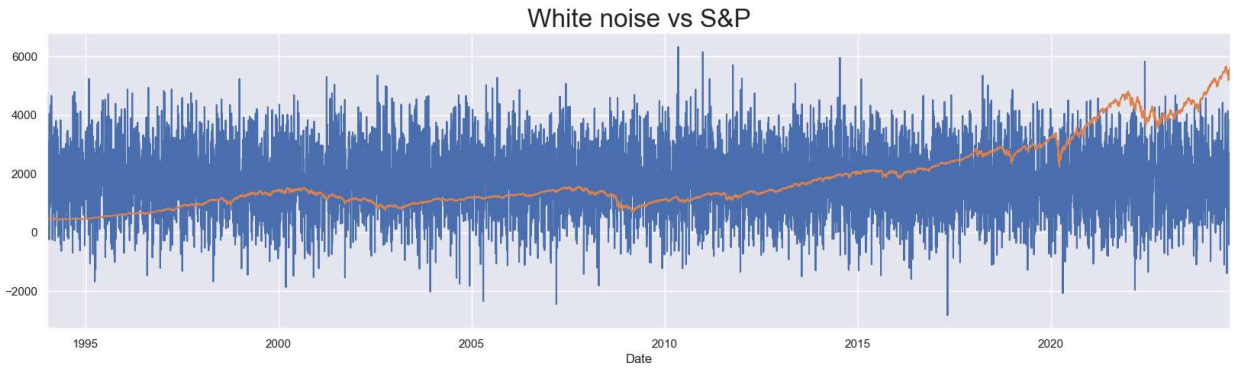
In [572...
```
df_yfinance['rw'].plot(figsize = (20,5));
```



In [573...
```
df_yfinance['spx'].plot(figsize=(20,5))
plt.title("S&P Prices", size = 24)
#plt.ylim(0,2300)
plt.show()
```

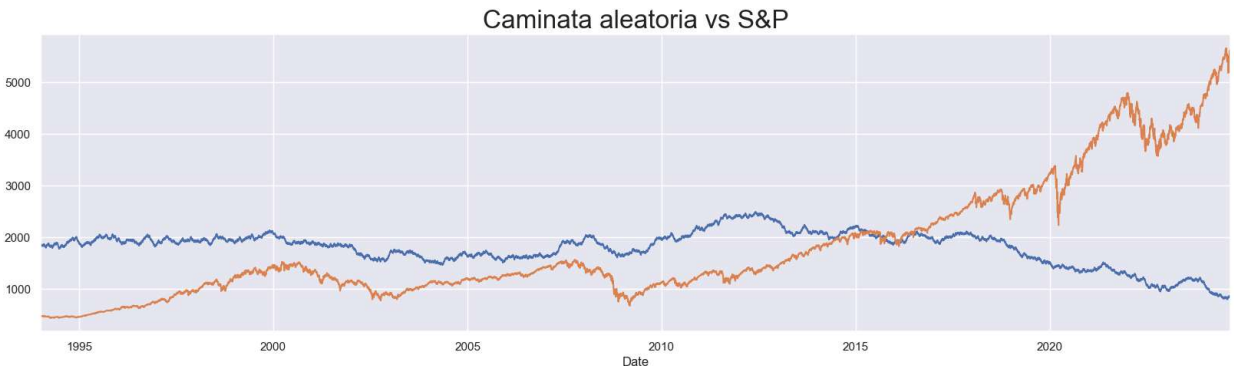
In [574…

```
df_yfinance.wn.plot(figsize = (20,5))
df_yfinance['spx'].plot()
plt.title("White noise vs S&P", size = 24)
plt.show()
```



In [575…

```
df_yfinance.rw.plot(figsize = (20,5))
df_yfinance['spx'].plot()
plt.title("Caminata aleatoria vs S&P", size = 24)
plt.show()
```



La descomposición de Hodrick-Prescott

La descomposición de Hodrick-Prescott descompone una serie de tiempo y_t en una componente de tendencia τ_t y una componente cíclica c_t

$$y_t = \tau_t + c_t$$

Estas componentes se determinan minimizando la siguiente función cuadrática, donde λ es un parámetro de suavizamiento:

$$\min_{\tau} \sum_{t=1}^{T} c_t^2 + \lambda \sum_{t=1}^{T} \left[(\tau_t - \tau_{t-1}) - (\tau_{t-1} - \tau_{t-2}) \right]^2$$

En este ejercicio, se aplicará la descomposición de Hodrick-Prescott al data frame: df_macrodata

In [576…]
```
ax = df_macrodata['realgdp'].plot()
ax.autoscale(axis='x',tight=True)
ax.set(ylabel='REAL GDP');
```

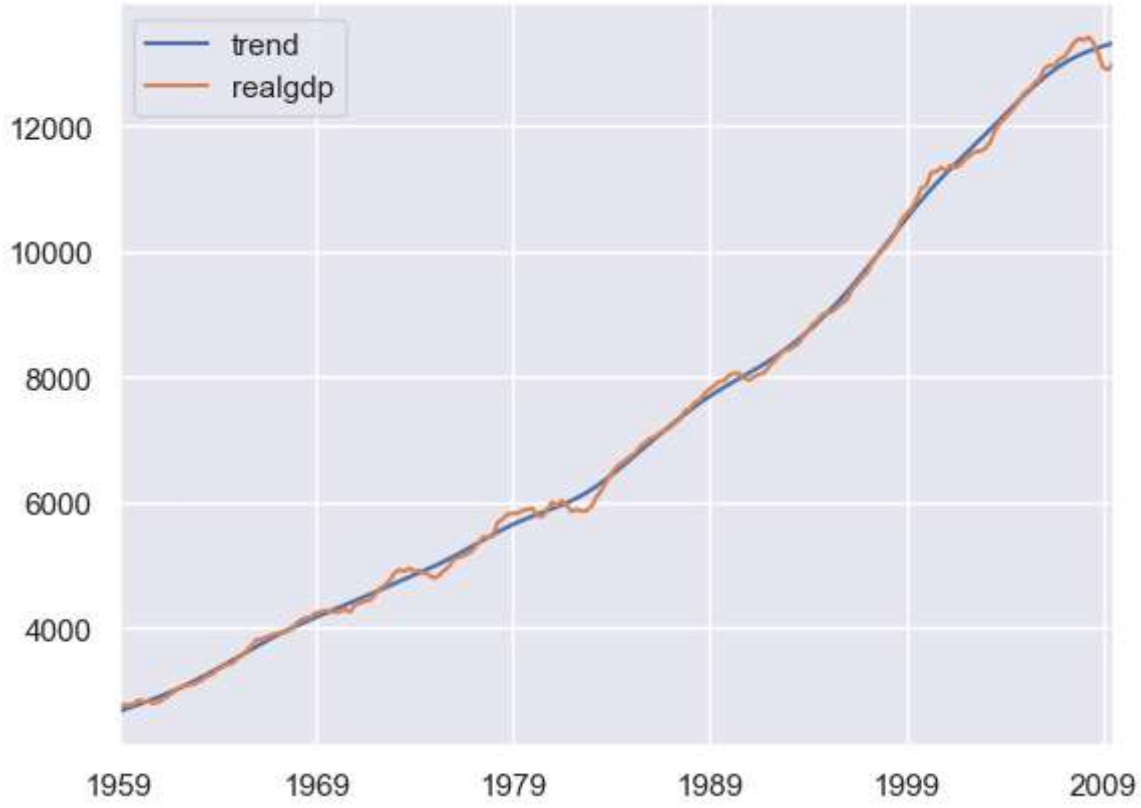
In [577…]
```
from statsmodels.tsa.filters.hp_filter import hpfilter

# Se separa la descomposición
gdp_cycle, gdp_trend = hpfilter(df_macrodata['realgdp'], lamb=1600)
```

In [578…]
```
gdp_cycle.head()
```

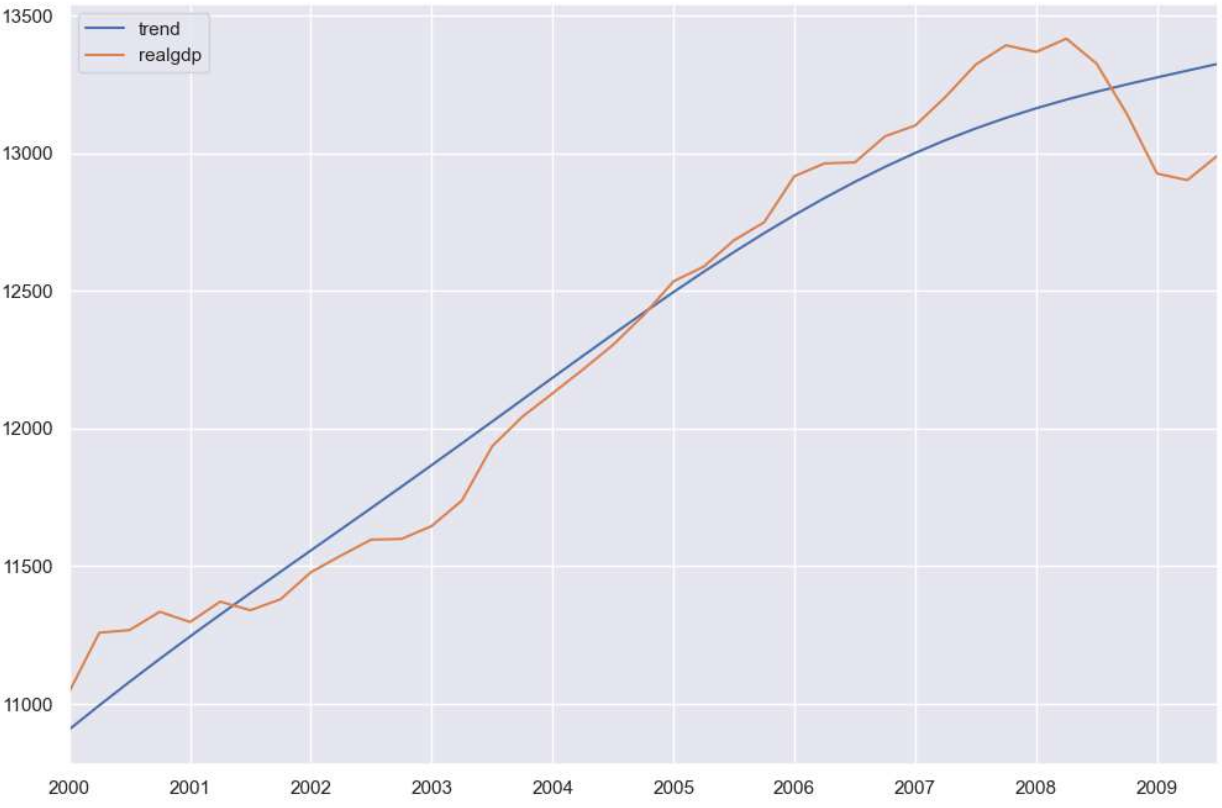
Out[578]:
```
1959-03-31    39.511915
1959-06-30    80.088532
1959-09-30    48.875455
1959-12-31    30.591933
1960-03-31    64.882667
Name: realgdp_cycle, dtype: float64
```

In [579…]
```
df_macrodata['trend'] = gdp_trend
```

In [580…]
```
df_macrodata[['trend','realgdp']].plot().autoscale(axis='x',tight=True);
```

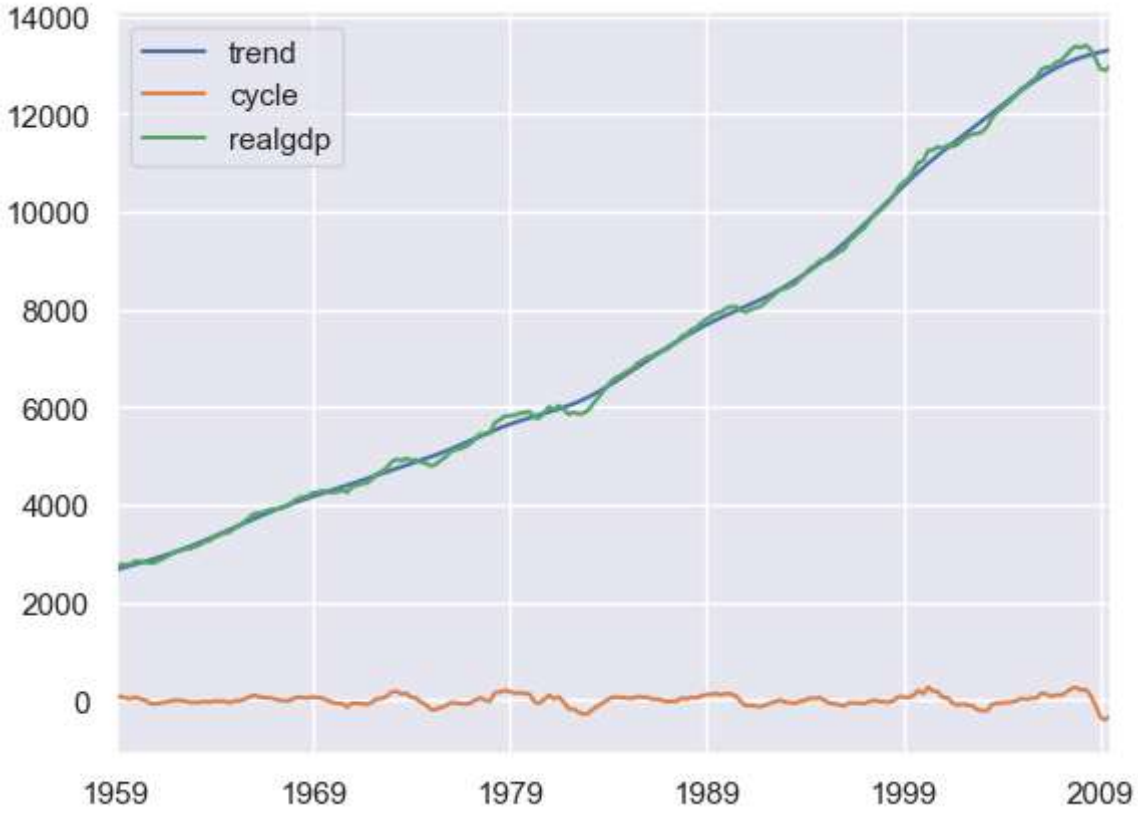


In [581…
```
df_macrodata[['trend','realgdp']]['2000-03-31':].plot(figsize=(12,8)).autoscale(axis='
```



In [582…
```
df_macrodata['cycle'] = gdp_cycle
```

In [583…
```
df_macrodata[['trend','cycle','realgdp']].plot().autoscale(axis='x',tight=True);
```



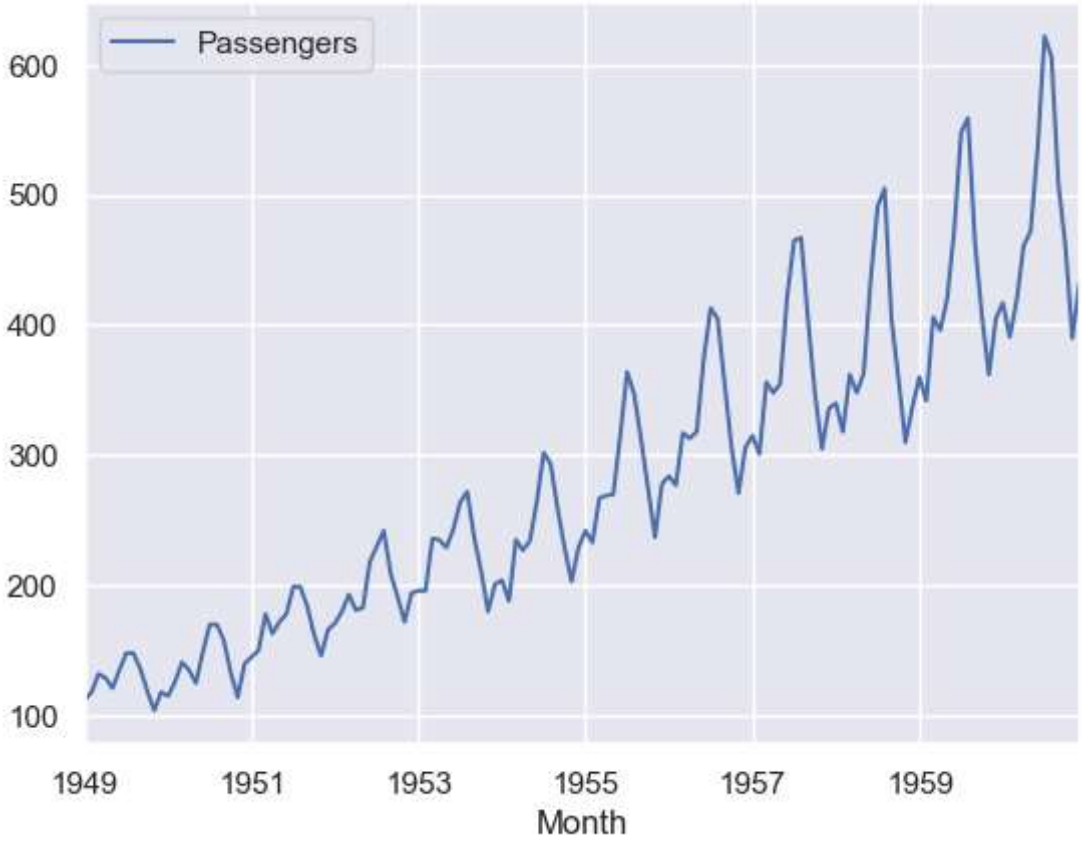
Descomposición ETS (Error-Trend-Seasonality)

Similar a la descomposición de Hodrick-Prescott, descompone una serie de tiempo en una componente de tendencia, una componente estacional y una componente de error. Si el modelo es aditivo, estas componentes se suman. Si el modelo es multiplicativo, las componentes se multiplican.

El modelo aditivo se usa cuando se aprecia un crecimiento lineal. Si el crecimiento (o decrecimiento) no es lineal, se suele usar el modelo multiplicativo.

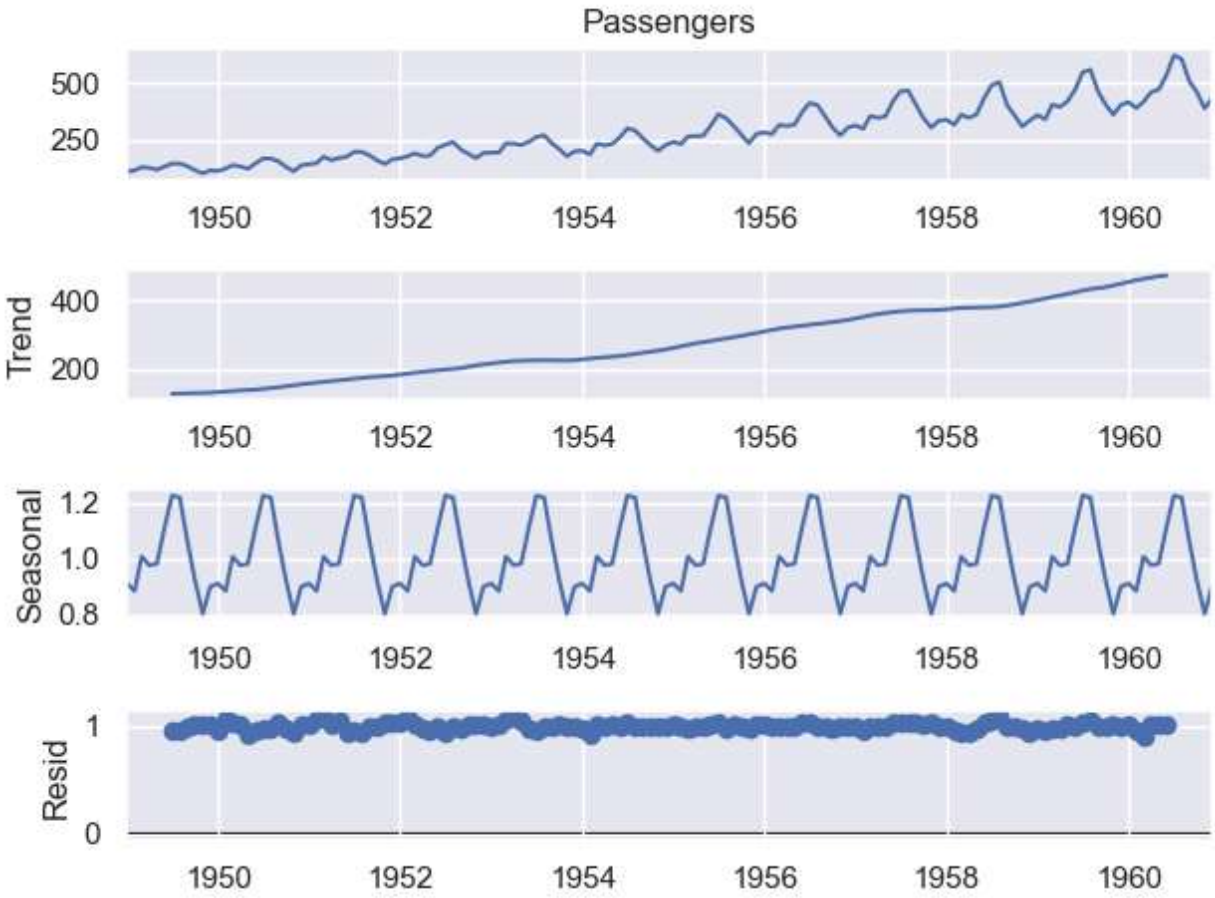
In [584…

```
df_airline.plot();
```



In [585…

```
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(df_airline['Passengers'], model='multiplicative')  # model
result.plot();
```



SMA (Simple Moving Average)

Es la media móvil que se calcula con los cálculos de ventana móvil

In [586…
```
df_airline['6-month-SMA'] = df_airline['Passengers'].rolling(window=6).mean()
df_airline['12-month-SMA'] = df_airline['Passengers'].rolling(window=12).mean()
```

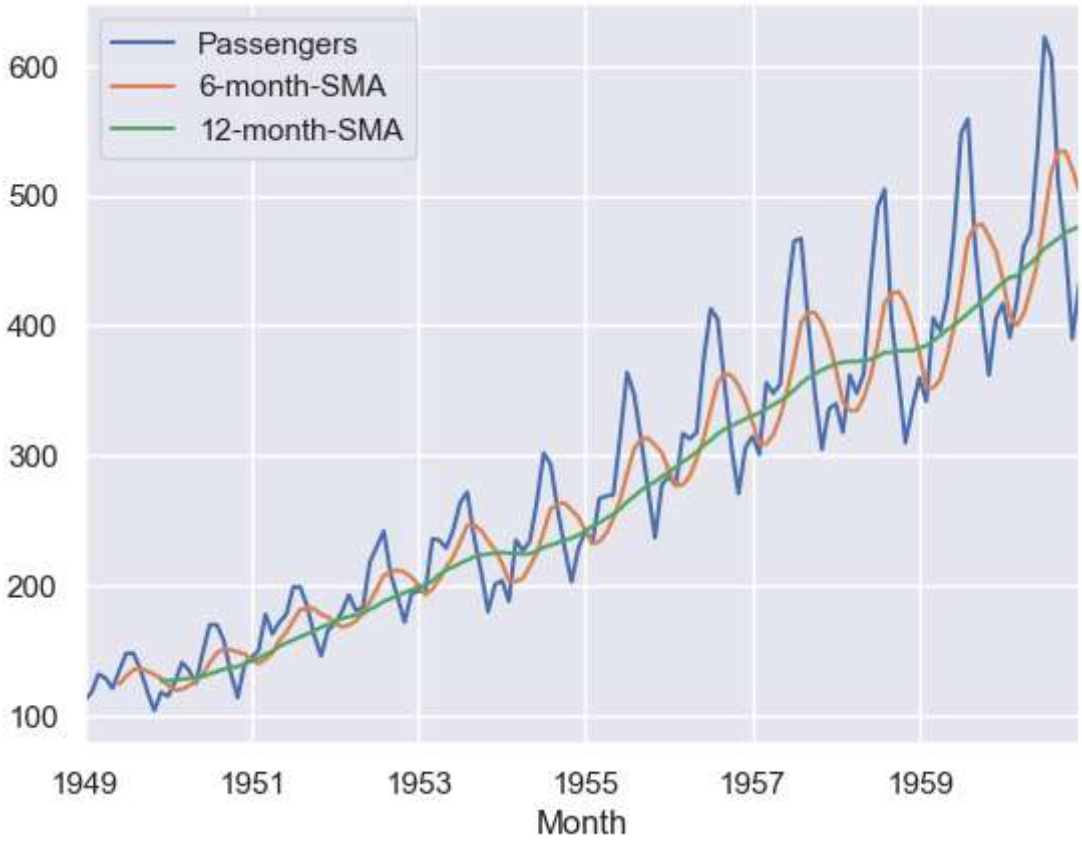
In [587…
```
df_airline.head(15)
```

Out[587]:

Month	Passengers	6-month-SMA	12-month-SMA
1949-01-01	112	NaN	NaN
1949-02-01	118	NaN	NaN
1949-03-01	132	NaN	NaN
1949-04-01	129	NaN	NaN
1949-05-01	121	NaN	NaN
1949-06-01	135	124.500000	NaN
1949-07-01	148	130.500000	NaN
1949-08-01	148	135.500000	NaN
1949-09-01	136	136.166667	NaN
1949-10-01	119	134.500000	NaN
1949-11-01	104	131.666667	NaN
1949-12-01	118	128.833333	126.666667
1950-01-01	115	123.333333	126.916667
1950-02-01	126	119.666667	127.583333
1950-03-01	141	120.500000	128.333333

In [588…

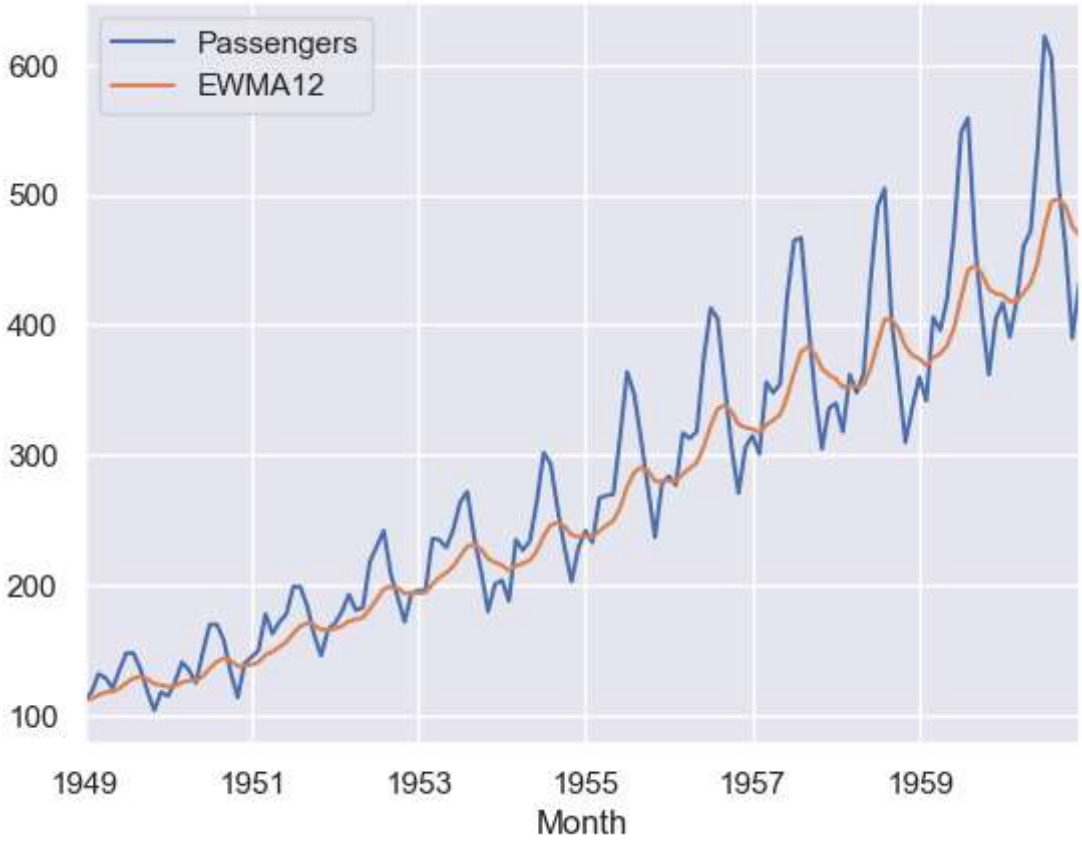
```
df_airline.plot();
```



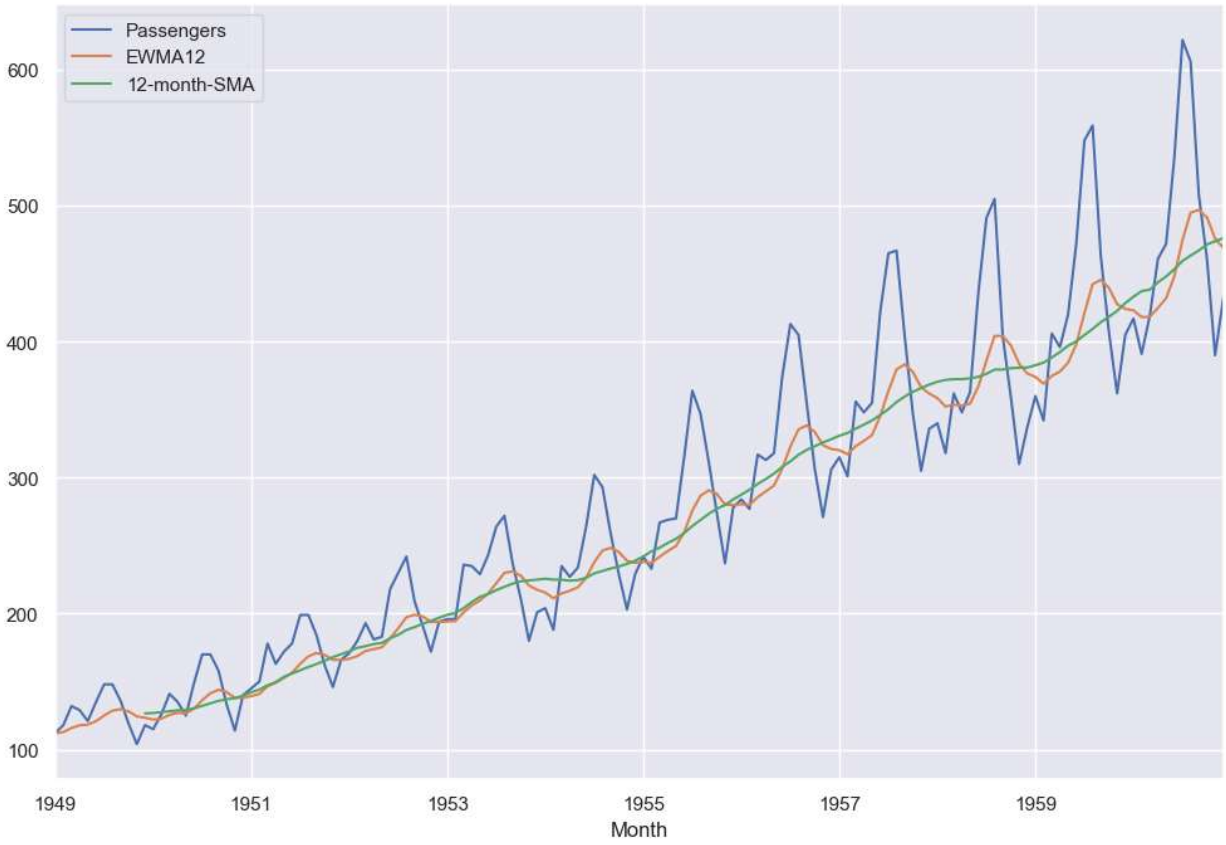
EWMA Exponentially Weighted Moving Average

In [589…
```
df_airline['EWMA12'] = df_airline['Passengers'].ewm(span=12,adjust=False).mean()
```

In [590…
```
df_airline[['Passengers','EWMA12']].plot();
```



In [591…
```
df_airline[['Passengers','EWMA12','12-month-SMA']].plot(figsize=(12,8)).autoscale(axis
```



Métodos de Holt-Winters

SES (Simple Exponential Smooting)

In [592…

```
# df_airline.index.freq = 'MS'
```

In [593…

```
from statsmodels.tsa.holtwinters import SimpleExpSmoothing

span = 12
alpha = 2/(span+1)

df_airline['EWMA12'] = df_airline['Passengers'].ewm(alpha=alpha,adjust=False).mean()
df_airline['SES12']=SimpleExpSmoothing(df_airline['Passengers']).fit(smoothing_level=a
df_airline.head()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency MS will be us
ed.
  self._init_dates(dates, freq)
```

Out[593]:

Month	Passengers	6-month-SMA	12-month-SMA	EWMA12	SES12
1949-01-01	112	NaN	NaN	112.000000	112.000000
1949-02-01	118	NaN	NaN	112.923077	112.923077
1949-03-01	132	NaN	NaN	115.857988	115.857988
1949-04-01	129	NaN	NaN	117.879836	117.879836
1949-05-01	121	NaN	NaN	118.359861	118.359861

DES (Double Exponential Smoothing)

In [609…

```
from statsmodels.tsa.holtwinters import ExponentialSmoothing

df_airline['DESadd12'] = ExponentialSmoothing(df_airline['Passengers'], trend='add').f
df_airline.head()
```

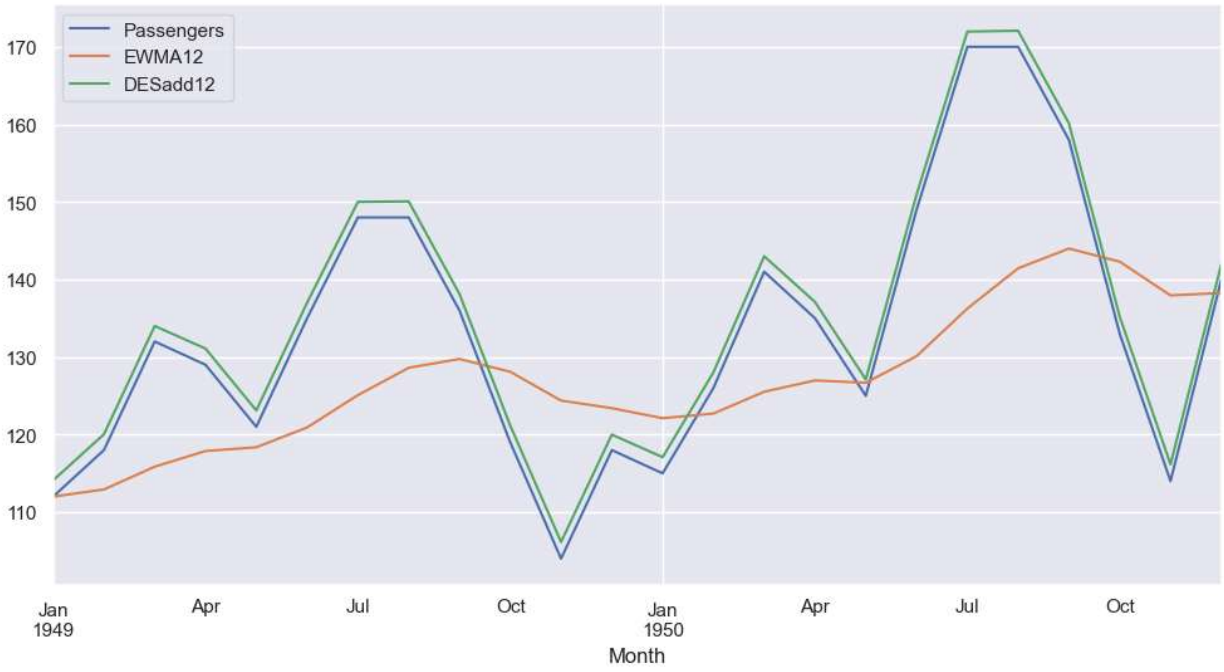
```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency MS will be us
ed.
  self._init_dates(dates, freq)
```

Out[609]:

Month	Passengers	6-month-SMA	12-month-SMA	EWMA12	SES12	DESadd12
1949-01-01	112	NaN	NaN	112.000000	112.000000	114.102394
1949-02-01	118	NaN	NaN	112.923077	112.923077	120.040657
1949-03-01	132	NaN	NaN	115.857988	115.857988	134.001539
1949-04-01	129	NaN	NaN	117.879836	117.879836	131.085845
1949-05-01	121	NaN	NaN	118.359861	118.359861	123.110263

In [610…

```
df_airline[['Passengers','EWMA12','DESadd12']].iloc[:24].plot(figsize=(12,6)).autoscal
```



In [611…

```
df_airline['DESmul12'] = ExponentialSmoothing(df_airline['Passengers'], trend='mul').f
df_airline.head()
```

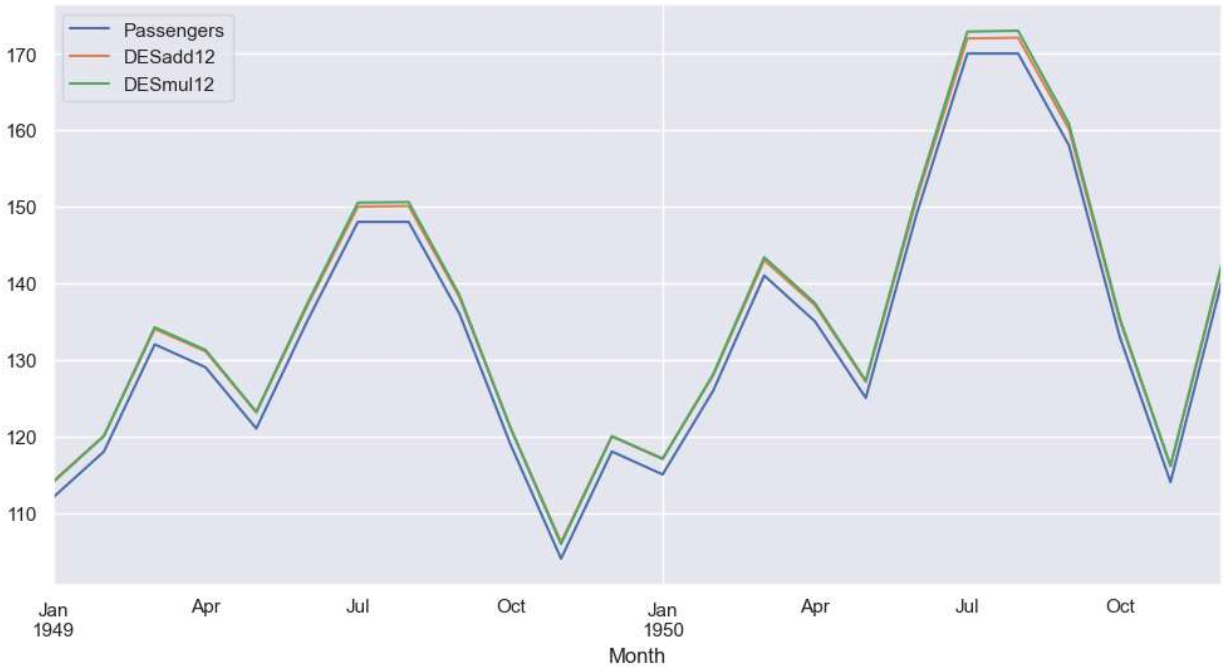
```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency MS will be us
ed.
  self._init_dates(dates, freq)
```

Out[611]:

Month	Passengers	6-month-SMA	12-month-SMA	EWMA12	SES12	DESadd12	DESmul12
1949-01-01	112	NaN	NaN	112.000000	112.000000	114.102394	113.990701
1949-02-01	118	NaN	NaN	112.923077	112.923077	120.040657	120.031669
1949-03-01	132	NaN	NaN	115.857988	115.857988	134.001539	134.235979
1949-04-01	129	NaN	NaN	117.879836	117.879836	131.085845	131.270786
1949-05-01	121	NaN	NaN	118.359861	118.359861	123.110263	123.156267

In [612…

```
df_airline[['Passengers','DESadd12','DESmul12']].iloc[:24].plot(figsize=(12,6)).autosc
```



TES (Triple Exponential Smoothing)

In [613…

```
df_airline['TESadd12'] = ExponentialSmoothing(df_airline['Passengers'],trend='add',sea
df_airline.head()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency MS will be us
ed.
  self._init_dates(dates, freq)
```

Out[613]:

Month	Passengers	6-month-SMA	12-month-SMA	EWMA12	SES12	DESadd12	DESmul12	TESadd12
1949-01-01	112	NaN	NaN	112.000000	112.000000	114.102394	113.990701	111.959998
1949-02-01	118	NaN	NaN	112.923077	112.923077	120.040657	120.031669	120.193337
1949-03-01	132	NaN	NaN	115.857988	115.857988	134.001539	134.235979	134.676835
1949-04-01	129	NaN	NaN	117.879836	117.879836	131.085845	131.270786	131.407263
1949-05-01	121	NaN	NaN	118.359861	118.359861	123.110263	123.156267	124.643743

In [614…

```
df_airline['TESmul12'] = ExponentialSmoothing(df_airline['Passengers'],trend='mul',sea
df_airline.head()
```

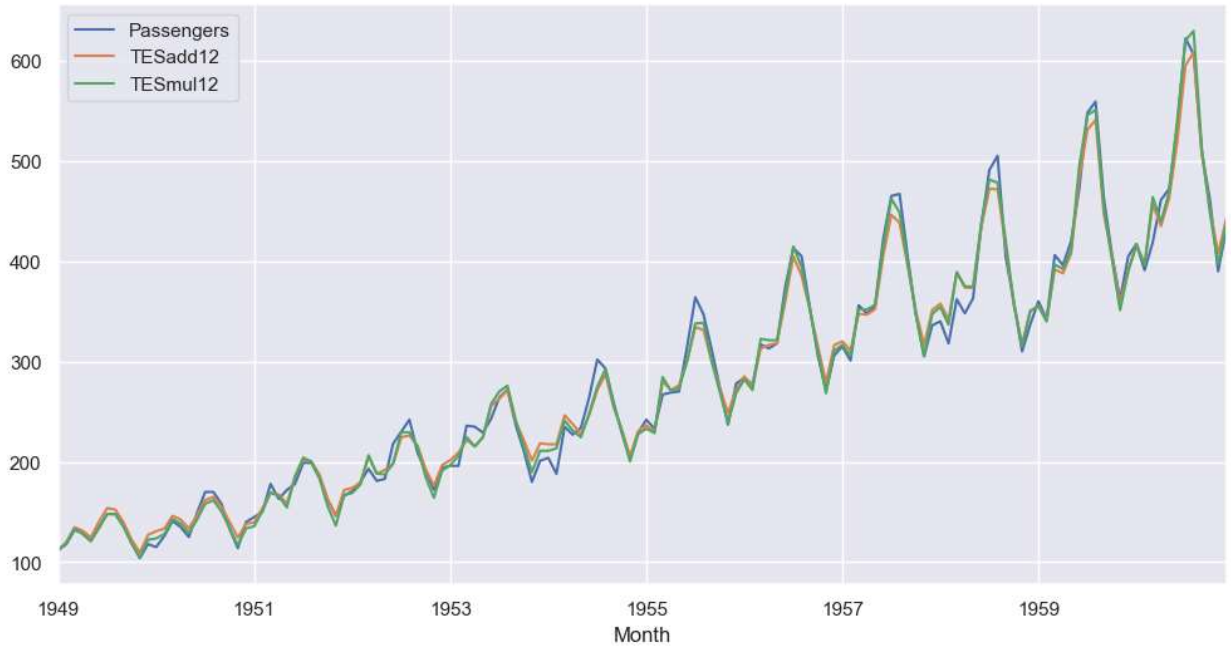
```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency MS will be us
ed.
  self._init_dates(dates, freq)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\holtwinters\model.py:83: R
untimeWarning: overflow encountered in matmul
  return err.T @ err
```

Out[614]:

Month	Passengers	6-month-SMA	12-month-SMA	EWMA12	SES12	DESadd12	DESmul12	TESadd12	TE
1949-01-01	112	NaN	NaN	112.000000	112.000000	114.102394	113.990701	111.959998	11
1949-02-01	118	NaN	NaN	112.923077	112.923077	120.040657	120.031669	120.193337	11
1949-03-01	132	NaN	NaN	115.857988	115.857988	134.001539	134.235979	134.676835	13
1949-04-01	129	NaN	NaN	117.879836	117.879836	131.085845	131.270786	131.407263	12
1949-05-01	121	NaN	NaN	118.359861	118.359861	123.110263	123.156267	124.643743	12

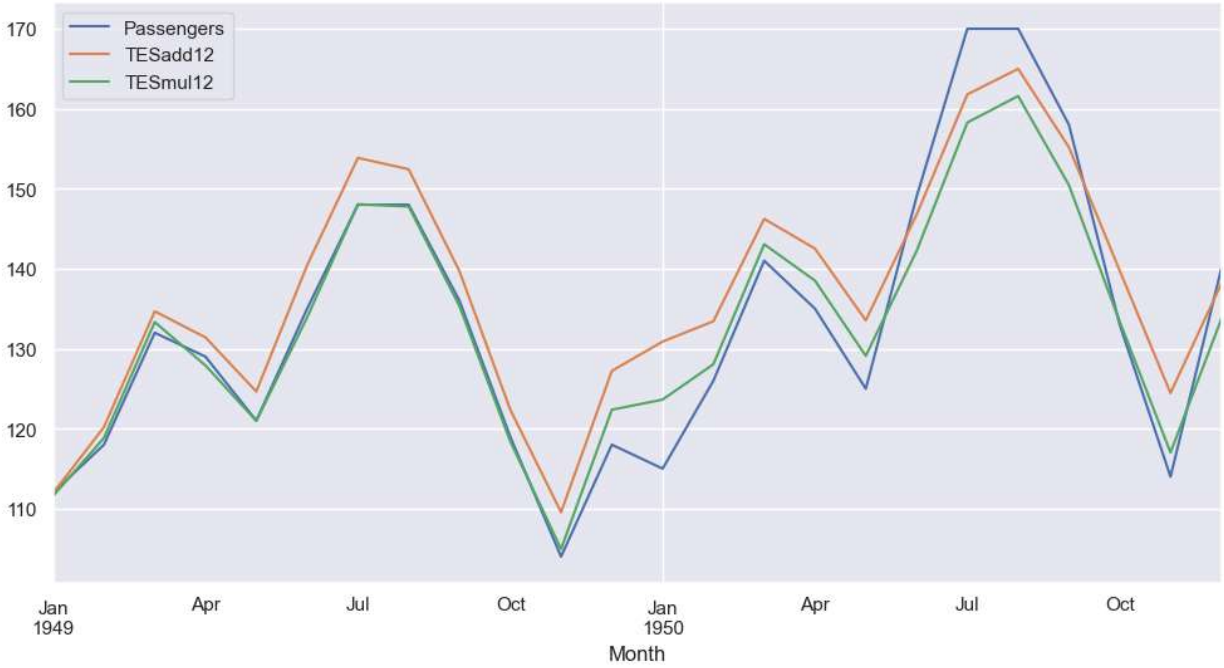
In [615…

```
df_airline[['Passengers','TESadd12','TESmul12']].plot(figsize=(12,6)).autoscale(axis='
```



In [616…

```
df_airline[['Passengers','TESadd12','TESmul12']].iloc[:24].plot(figsize=(12,6)).autosc
```



Pronósticos (Forecasting)

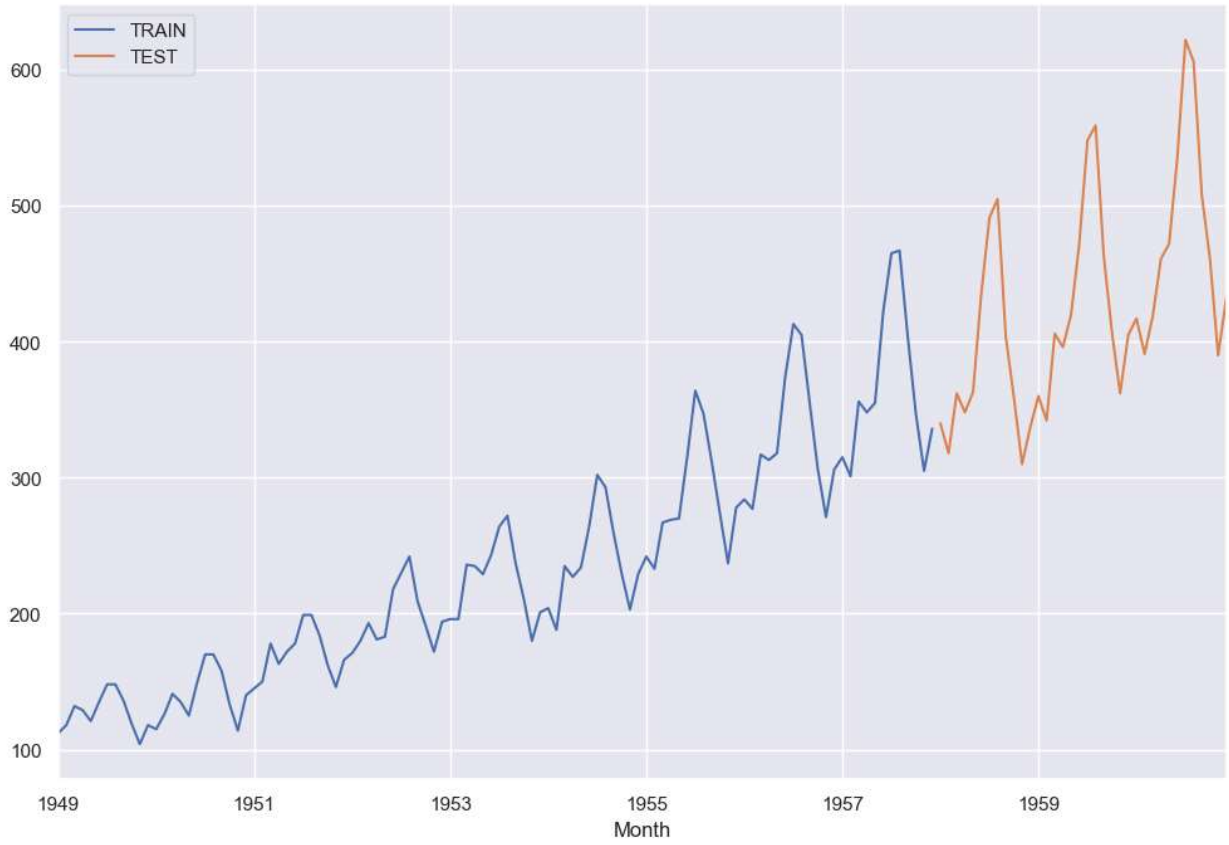
In [618…
```
# Descomposición en conjunto de entrenamiento y conjunto de prueba
df_airline_train_data = df_airline.iloc[:108]
df_airline_test_data = df_airline.iloc[108:]
```

In [619…
```
# Ajuste del modelo (Exponential Smoothing)
from statsmodels.tsa.holtwinters import ExponentialSmoothing

df_airline_fitted_model = ExponentialSmoothing(df_airline_train_data['Passengers'],tre
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency MS will be us
ed.
  self._init_dates(dates, freq)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\holtwinters\model.py:83: R
untimeWarning: overflow encountered in matmul
  return err.T @ err
```

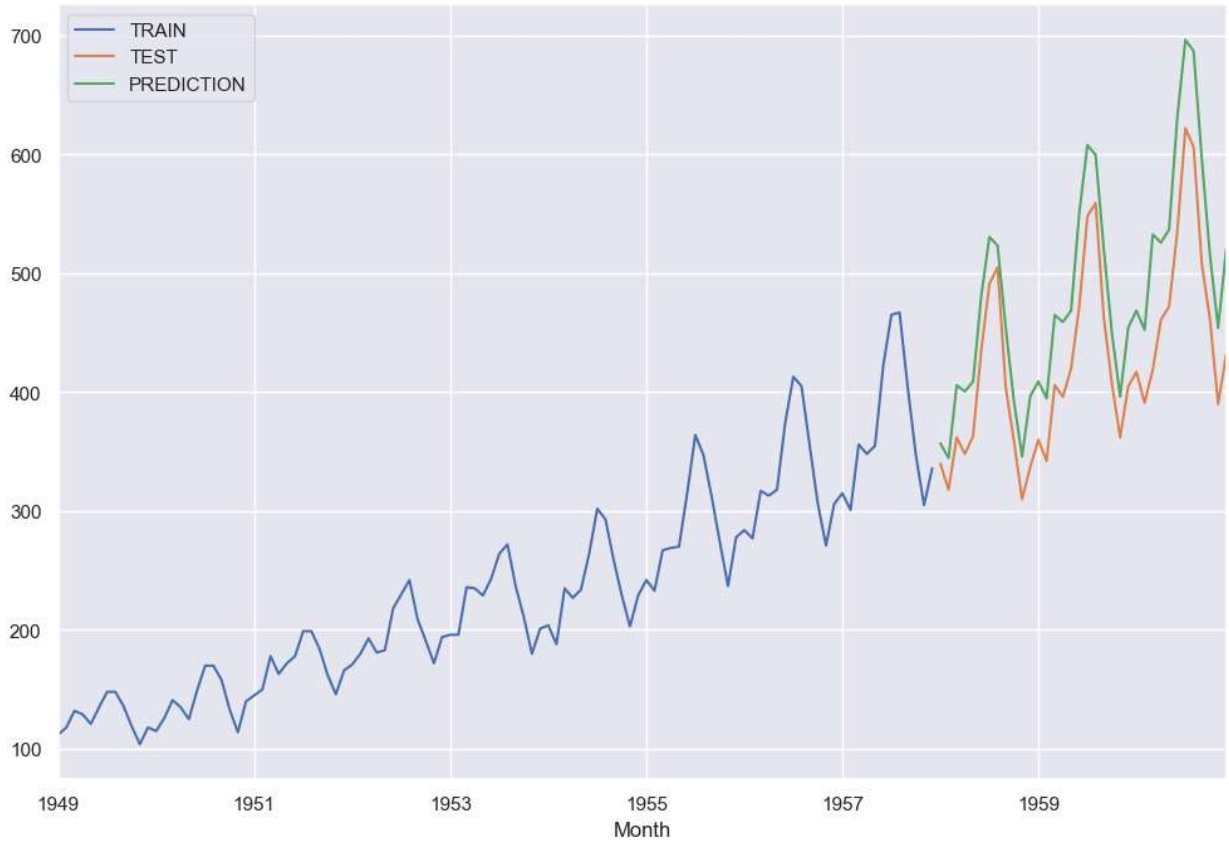
In [621…
```
# Evaluación del modelo
df_airline_test_predictions = df_airline_fitted_model.forecast(36).rename('HW Forecast
```

In [622…
```
df_airline_train_data['Passengers'].plot(legend=True,label='TRAIN')
df_airline_test_data['Passengers'].plot(legend=True,label='TEST',figsize=(12,8));
```

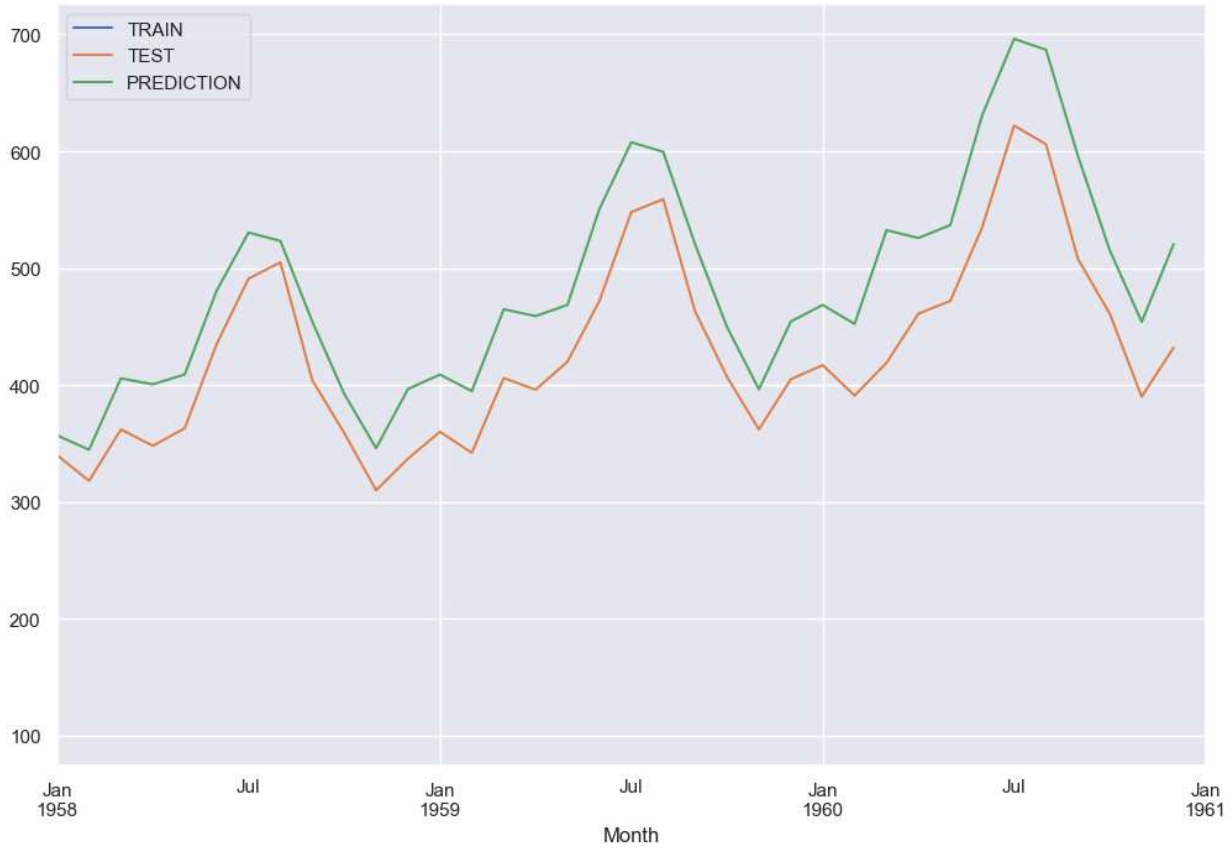


In [623…

```
df_airline_train_data['Passengers'].plot(legend=True,label='TRAIN')
df_airline_test_data['Passengers'].plot(legend=True,label='TEST',figsize=(12,8))
df_airline_test_predictions.plot(legend=True,label='PREDICTION');
```



In [624…
```
df_airline_train_data['Passengers'].plot(legend=True,label='TRAIN')
df_airline_test_data['Passengers'].plot(legend=True,label='TEST',figsize=(12,8))
df_airline_test_predictions.plot(legend=True,label='PREDICTION',xlim=['1958-01-01','19
```



Métricas de evaluación

In [625…
```
from sklearn.metrics import mean_squared_error,mean_absolute_error
```

In [639…
```
df_airline_test_data.drop(['6-month-SMA','12-month-SMA','EWMA12','SES12','DESadd12','D
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_26928\2008869839.py:1: SettingWithCopyWa
rning:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df_airline_test_data.drop(['6-month-SMA','12-month-SMA','EWMA12','SES12','DESadd1
2','DESmul12','TESadd12','TESmul12'], axis='columns',inplace=True)
```

In [640…
```
df_airline_test_data.head()
```

Out[640]:

Month	Passengers
1958-01-01	340
1958-02-01	318
1958-03-01	362
1958-04-01	348
1958-05-01	363

In [641…
```
df_airline_test_predictions.head()
```

Out[641]:
```
1958-01-01    356.968658
1958-02-01    344.588831
1958-03-01    405.718358
1958-04-01    400.610839
1958-05-01    409.001684
Freq: MS, Name: HW Forecast, dtype: float64
```

In [642…
```
mean_absolute_error(df_airline_test_data,df_airline_test_predictions)
```

Out[642]:
```
55.698326986164496
```

In [643…
```
mean_squared_error(df_airline_test_data,df_airline_test_predictions)
```

Out[643]:
```
3525.9322806129094
```

In [644…
```
np.sqrt(mean_squared_error(df_airline_test_data,df_airline_test_predictions))
```

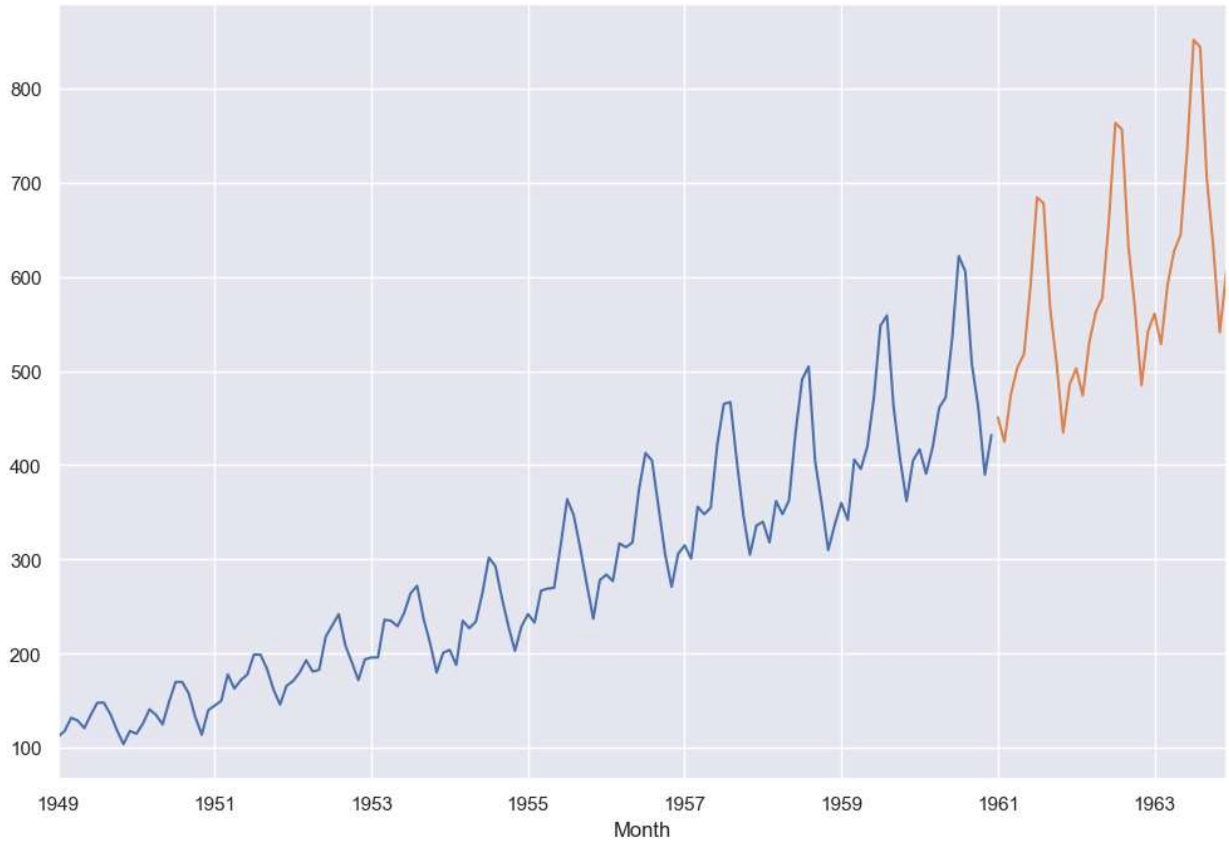
Out[644]:
```
59.3795611352333
```

In [645…
```
# Modelo final
final_model = ExponentialSmoothing(df_airline['Passengers'],trend='mul',seasonal='mul'
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency MS will be us
ed.
  self._init_dates(dates, freq)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\holtwinters\model.py:83: R
untimeWarning: overflow encountered in matmul
  return err.T @ err
```

In [646…
```
forecast_predictions = final_model.forecast(36)
```

In [647…
```
df_airline['Passengers'].plot(figsize=(12,8))
forecast_predictions.plot();
```



Estacionaridad

In [594…
```
sts.adfuller(df_yfinance['spx'])
```

Out[594]:
```
(2.495421902147278,
 0.9990483594146493,
 32,
 7959,
 {'1%': -3.4311718884675066,
  '5%': -2.861903215555782,
  '10%': -2.566963334958385},
 72367.34132821791)
```

In [595…
```
sts.adfuller(df_yfinance['wn'])
```

Out[595]:
```
(-89.74058595566777,
 0.0,
 0,
 7991,
 {'1%': -3.431168596152813,
  '5%': -2.861901760790028,
  '10%': -2.5669625605706994},
 134771.9239369095)
```

In [596…
```
sts.adfuller(df_yfinance['rw'])
```

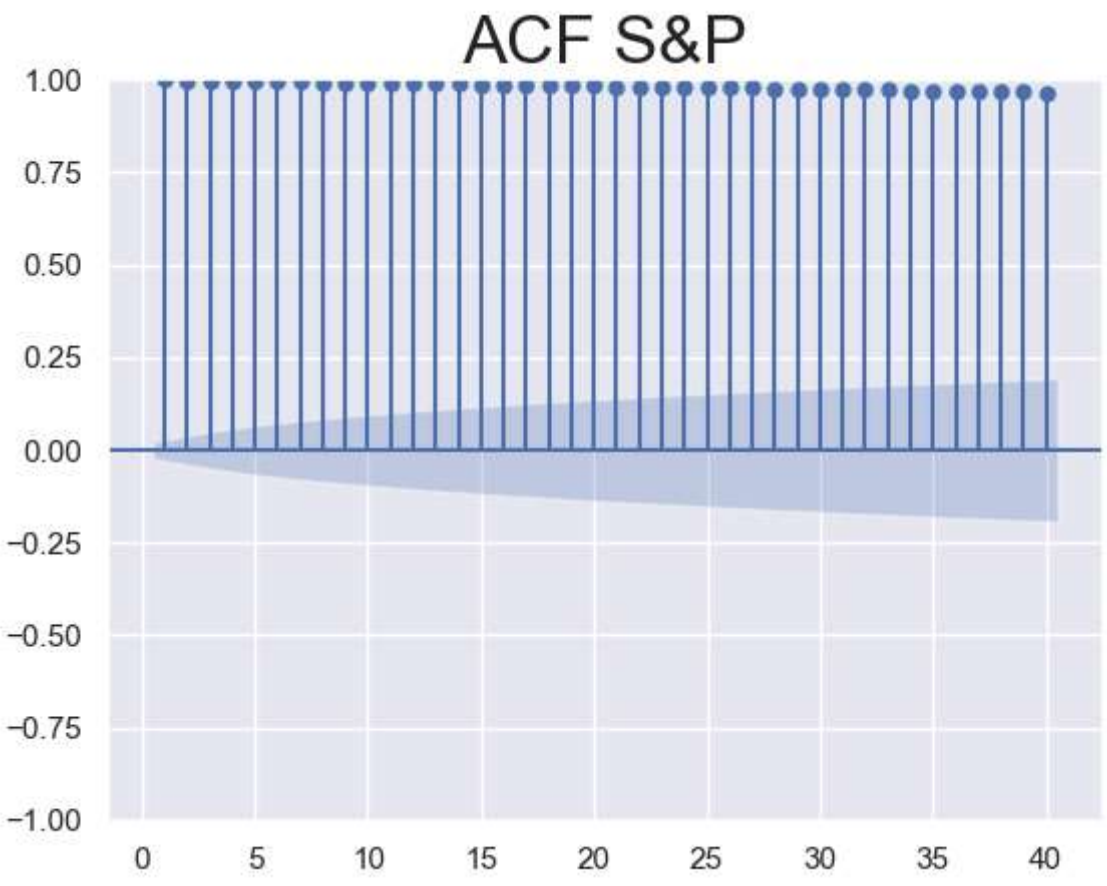
Out[596]:
```
(0.1859099015889541,
 0.9714601579831695,
 0,
 7991,
 {'1%': -3.431168596152813,
  '5%': -2.861901760790028,
  '10%': -2.5669625605706994},
 59212.22135391413)
```

In [597…
```
sts.adfuller(df_ventas['Ventas'])
```

Out[597]:
```
(-2.853928653615534,
 0.05097376463402474,
 13,
 351,
 {'1%': -3.44911857009962,
  '5%': -2.8698097654570507,
  '10%': -2.5711757061225153},
 6365.4145809406855)
```

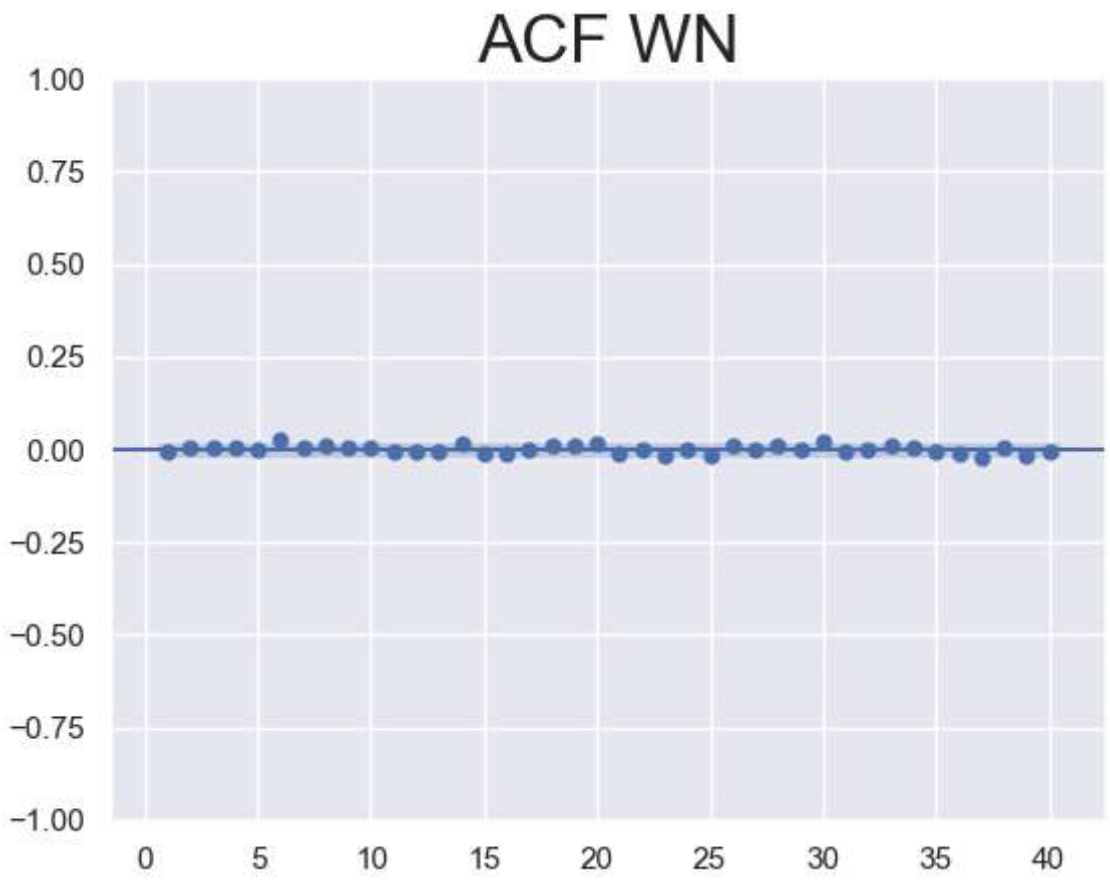
Funciones de autocorrelación y autocorrelación parcial

In [601…
```
sgt.plot_acf(df_yfinance['spx'], lags = 40, zero = False)
plt.title("ACF S&P", size = 24)
plt.show()
```



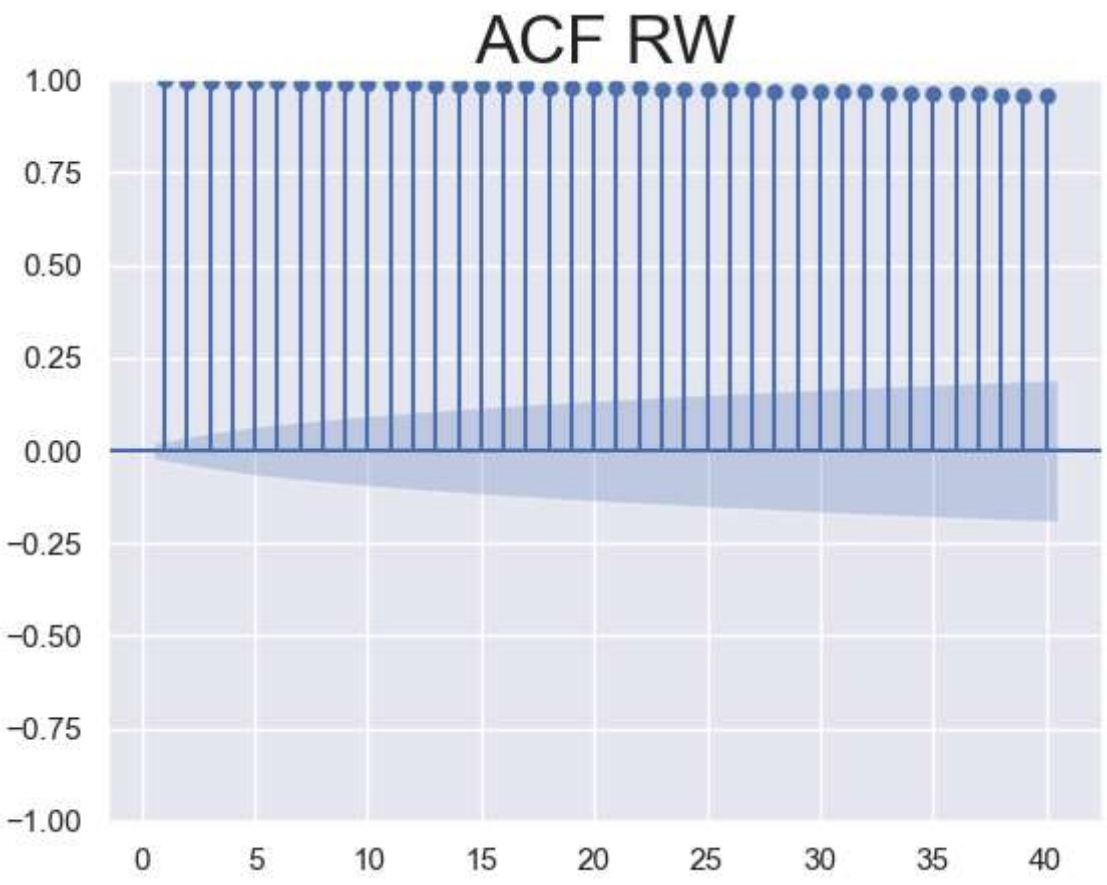
In [602…
```
sgt.plot_acf(df_yfinance['wn'], lags = 40, zero = False)
plt.title("ACF WN", size = 24)
```

```
plt.show()
```



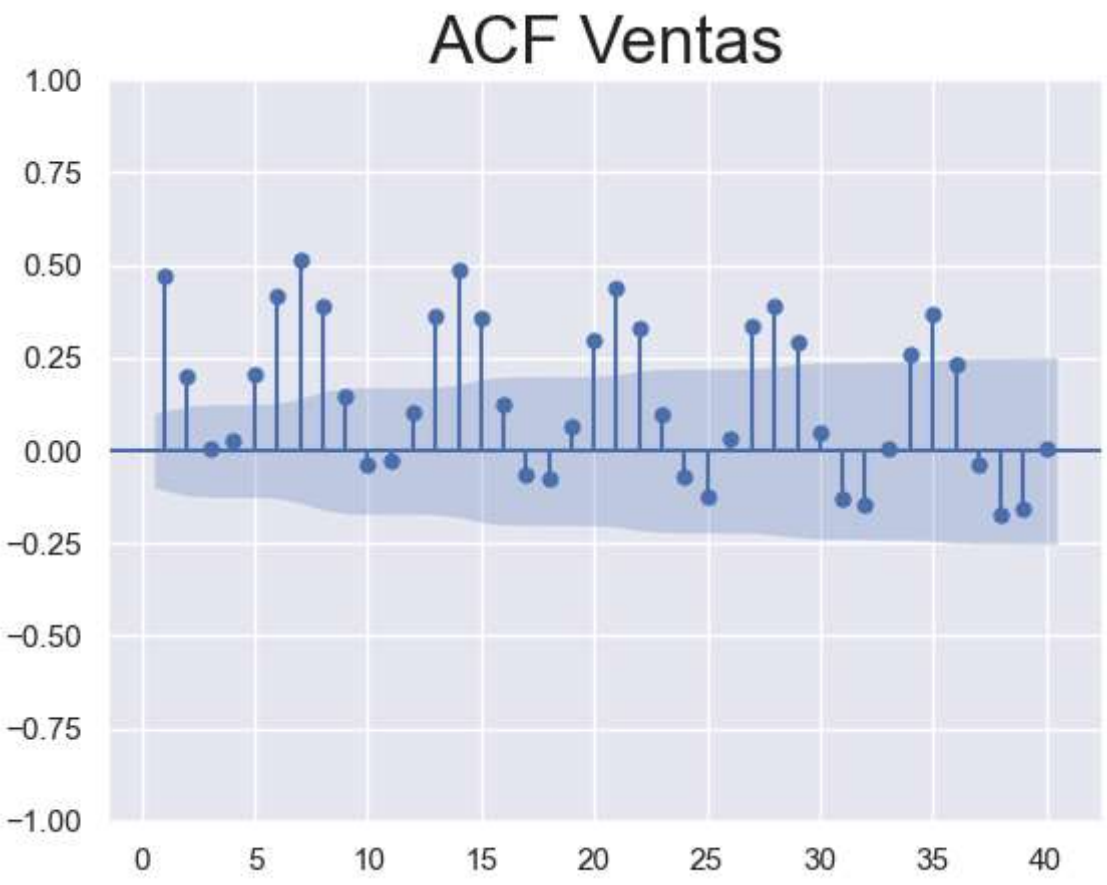
In [603…

```
sgt.plot_acf(df_yfinance['rw'], lags = 40, zero = False)
plt.title("ACF RW", size = 24)
plt.show()
```



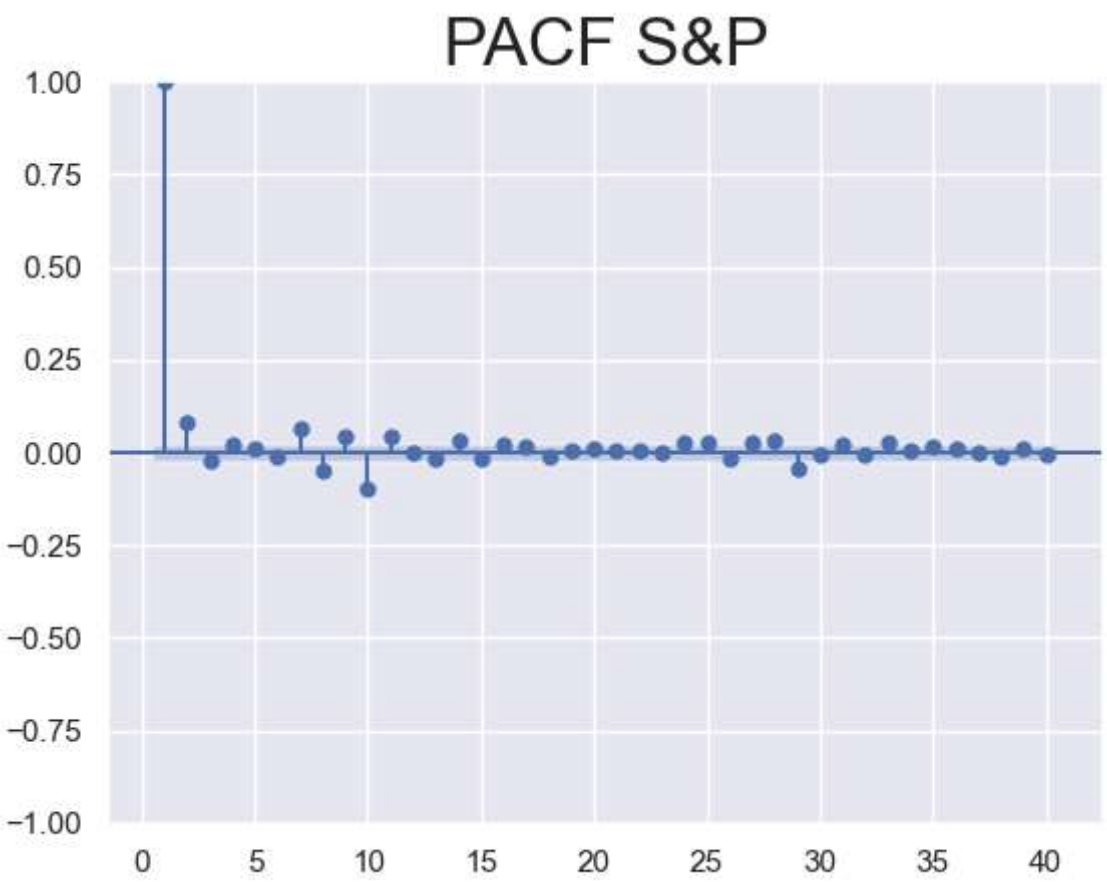
In [604…

```
sgt.plot_acf(df_ventas['Ventas'], lags = 40, zero = False)
plt.title("ACF Ventas", size = 24)
plt.show()
```



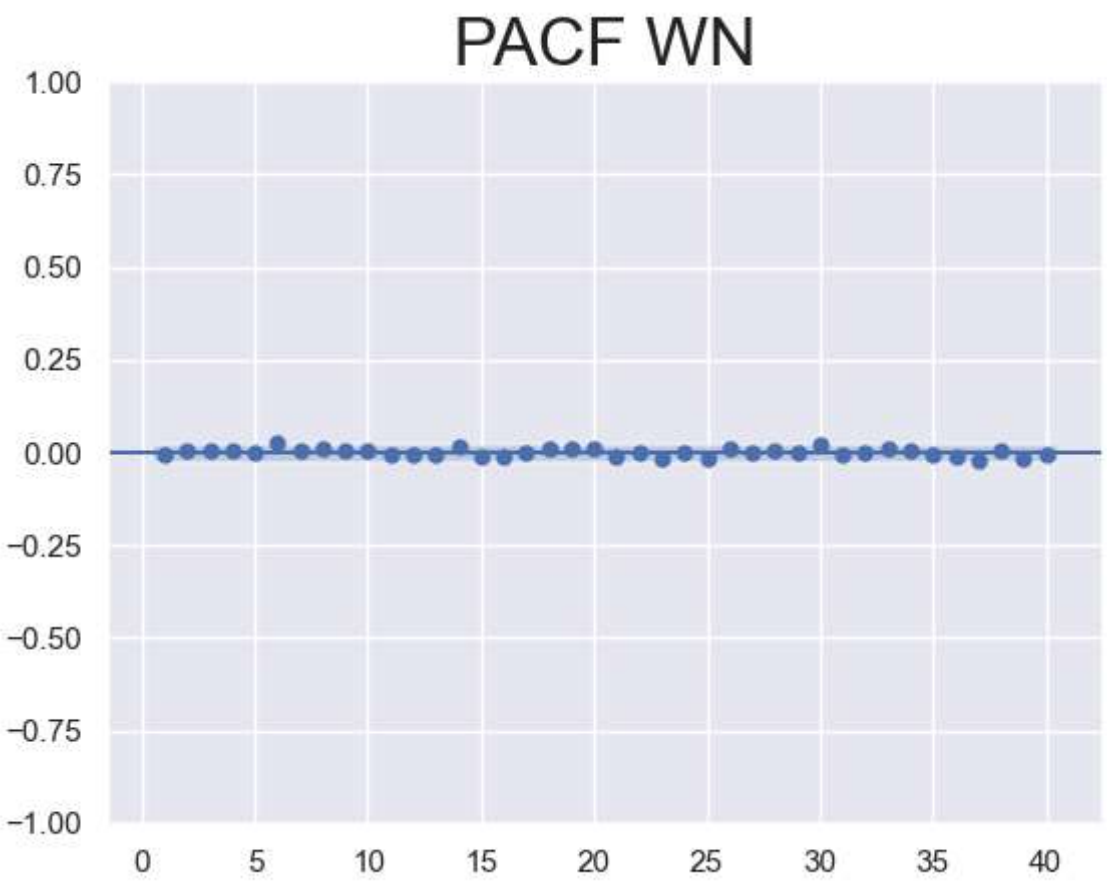
In [605…

```
sgt.plot_pacf(df_yfinance['spx'], lags = 40, zero = False, method = ('ols'))
plt.title("PACF S&P", size = 24)
plt.show()
```



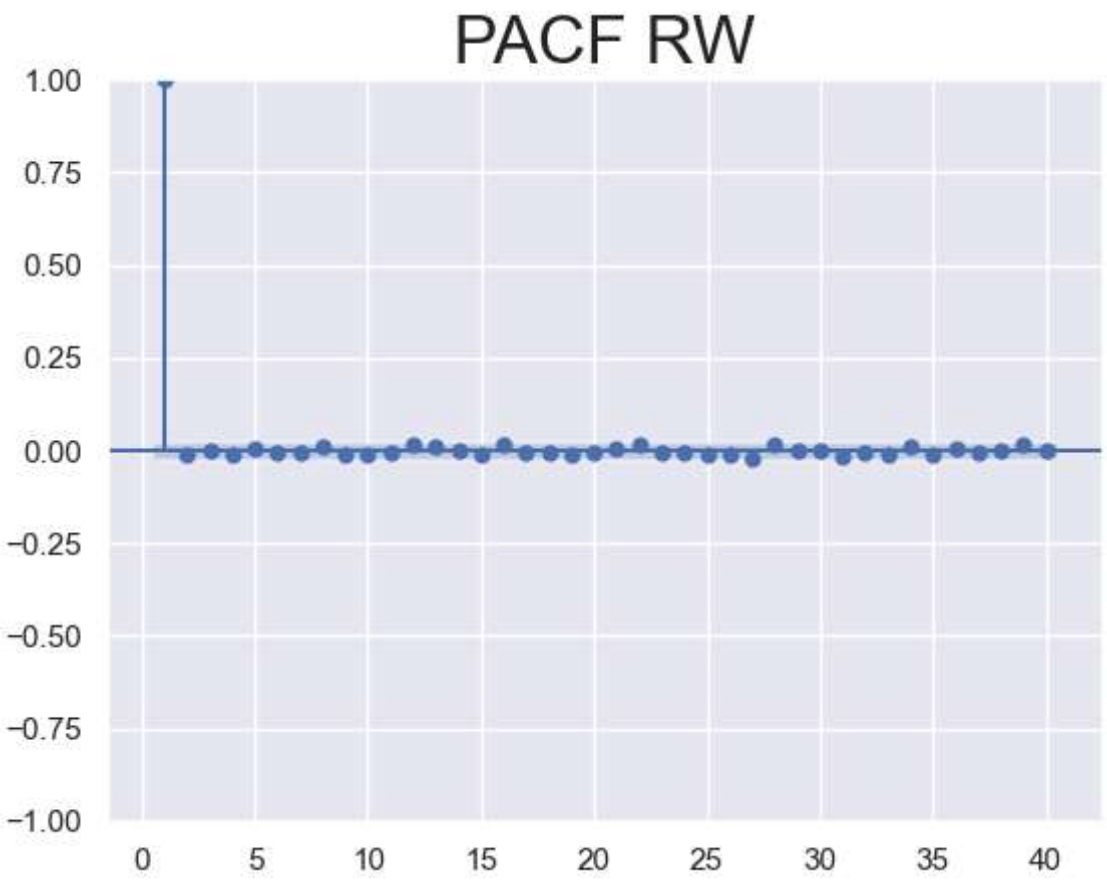
In [606…

```
sgt.plot_pacf(df_yfinance['wn'], lags = 40, zero = False, method = ('ols'))
plt.title("PACF WN", size = 24)
plt.show()
```



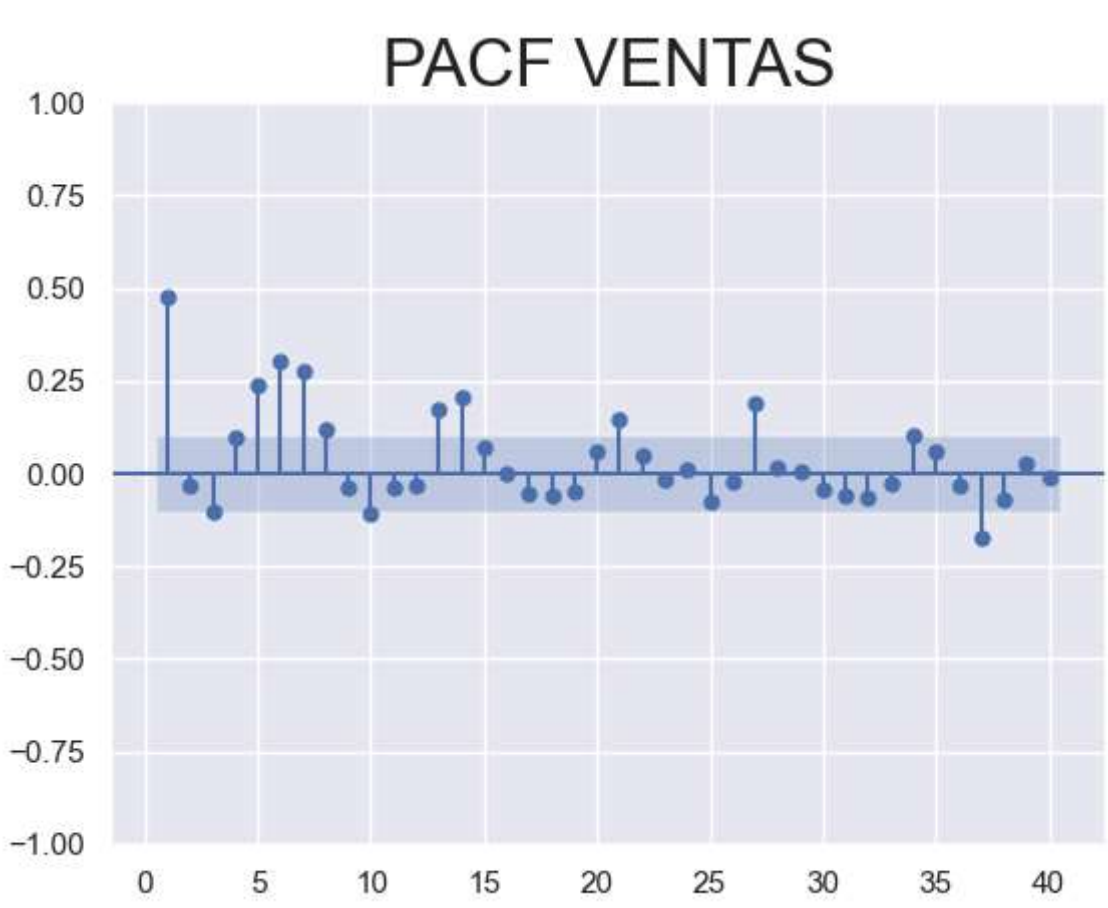
In [607…

```
sgt.plot_pacf(df_yfinance['rw'], lags = 40, zero = False, method = ('ols'))
plt.title("PACF RW", size = 24)
plt.show()
```



In [608…

```
sgt.plot_pacf(df_ventas['Ventas'], lags = 40, zero = False, method = ('ols'))
plt.title("PACF VENTAS", size = 24)
plt.show()
```


PACF VENTAS
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