Índice

Econometría Aplicada. Lección 2

Marcos Bujosa

3 de abril de 2025

En esta lección veremos algunos modelos de regresión con series temporales; en particular la estimación de componentes (no observables) con modelos deterministas. También los efectos de la autocorrelación en las perturbaciones y como lidiar con ellos.

- lección en html
- lección en mybinder

Carga de algunos módulos de python

```
# Importamos algunos módulos de python
import numpy as np # linear algebra
import pandas as pd # dataframe processing
import statsmodels.api as sm  # modelos estadísticos
import matplotlib as mpl
import matplotlib.pyplot as plt # data visualization
# definimos parámetros para mejorar los gráficos
mpl.rc('text', usetex=False)
from matplotlib import rcParams
rcParams['figure.figsize'] = 15,5
```

- Lectura datos: Internat. airline passengers. Monthly totals in thousands. Jan 49 – Dec 60

```
# Leemos los datos de un fichero csv y generamos un dataframe de pandas cuyo índice es el tiempo
OrigData = pd.read_csv('./database/Datasets-master/airline-passengers.csv')
OrigData['Month'] = pd.to_datetime(OrigData['Month'])
OrigData = OrigData.set_index(['Month'])
print(OrigData.head())
```

```
# Creamos un dataframe con el mismo índice temporal de los datos originales pero con los datos en logaritmos
TransformedData = pd.DataFrame(index=OrigData.index)
TransformedData['dataLog'] = np.log(OrigData['Passengers'])
print(TransformedData.head())
```

1. Descomposición estructural de una serie temporal

En la lección anterior vimos que una estrategia para analizar series temporales es transformar los datos para

1. primero lograr que sean "***estacionarios***"
2. después, mediante más transformaciones, lograr una secuencia de "***ruido blanco***"(este segundo paso lo veremos cuando veamos los modelos ARIMA)

(recuerde que las expresiones *datos estacionarios* y *secuencia de ruido blanco* son un abuso del lenguaje).

Pero existe otro enfoque que pretende descomponer la serie temporal en los siguientes componentes *no observables* (o en un subconjunto de ellos):

$$\boldsymbol{y} = \boldsymbol{t} + \boldsymbol{c} + \boldsymbol{s} + \boldsymbol{e}$$

donde:

La tendencia "$\boldsymbol{t}$" recoge la lenta evolución de la media a *largo plazo*.

El componente estacional "$\boldsymbol{s}$" recoge oscilaciones periódicas que se repiten regularmente en ciclos estacionales (o semanales, u horarios, etc.).

El componente cíclico "$\boldsymbol{c}$" Cuando aparece explícitamente en el modelo, $\boldsymbol{c}$ recoge las oscilaciones a medio plazo. Es decir, aquellas de un plazo más largo que las oscilaciones estacionales, pero más corto que la tendencia de largo plazo. Si está ausente, dichas oscilaciones suelen aparecer en el componente de la tendencia, que entonces también podemos denominar *tendencia-ciclo*.

El componente irregular "$\boldsymbol{e}$" recoge las oscilaciones no captadas por el resto de componentes, pues: $\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{t} - \boldsymbol{c} - \boldsymbol{s}$.

Ajuste aceptable si (como poco) el componente irregular $\boldsymbol{e}$ parece "*estacionario*".

1.1. Tendencia determinista *lineal*

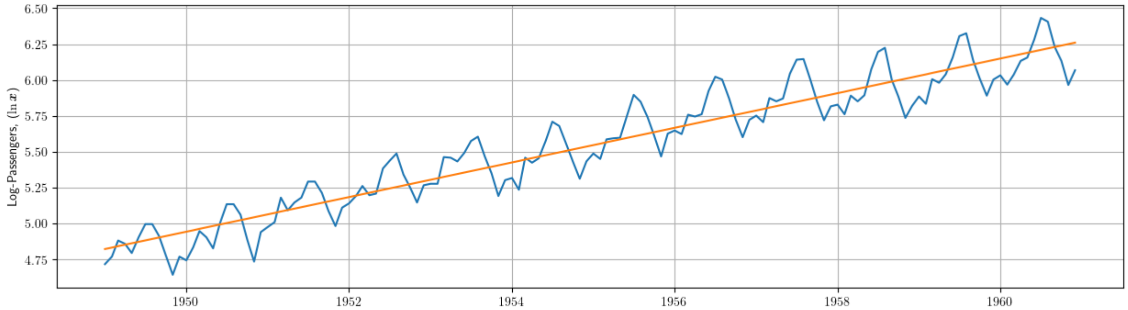
```python
# Ajustamos por MCO una tendencia linea.
# Para ello, primero creamos un DataFrame con el regresando y los regresores del modelo
datosModelo1 = TransformedData[['dataLog']].copy()
nsample = len(datosModelo1)
datosModelo1['cte'] = [1]*nsample
datosModelo1['time'] = np.linspace(1, nsample, nsample)
model1 = sm.OLS(datosModelo1['dataLog'], datosModelo1[['cte', 'time']])
results1 = model1.fit()
```

```python
#Añadimos al DataFrame =datosModelo1= la tendencia ajustada, los residuos y la diferencia estacional de los residuos.
datosModelo1['yhat'] = datosModelo1['cte']*results1.params['cte']+datosModelo1['time']*results1.params['time']
datosModelo1['ehat'] = results1.resid
datosModelo1['ehatDiff12'] = datosModelo1['ehat'].diff(12)
```

```python
# Dibujamos los datos junto a la tendencia estimada
plt.plot(datosModelo1['dataLog'])
plt.plot(results1.fittedvalues)
plt.grid()
plt.ylabel(r"Log-Passengers, ($\ln\boldsymbol{x}$) ")
```

El modelo de tendencia más simple es la recta de regresión (donde el regresor no constante es el índice t):

$$\ln y_t = \underbrace{\beta_1 + \beta_2 \cdot t}_{\text{tendencia}} + e_t; \quad t = 1 : 114$$



$$\widehat{\ln y_t} = 4{,}8137 + 0{,}01 \cdot (t), \qquad t = 1 : 114$$

```python
print(results1.summary())
```

Dep. Variable:	dataLog	**R-squared:**	0.902
Model:	OLS	**Adj. R-squared:**	0.901
Method:	Least Squares	**F-statistic:**	1300.
Date:	Thu, 03 Apr 2025	**Prob (F-statistic):**	2.41e-73
Time:	20:33:37	**Log-Likelihood:**	80.794
No. Observations:	144	**AIC:**	-157.6
Df Residuals:	142	**BIC:**	-151.6
Df Model:	1		
Covariance Type:	nonrobust		

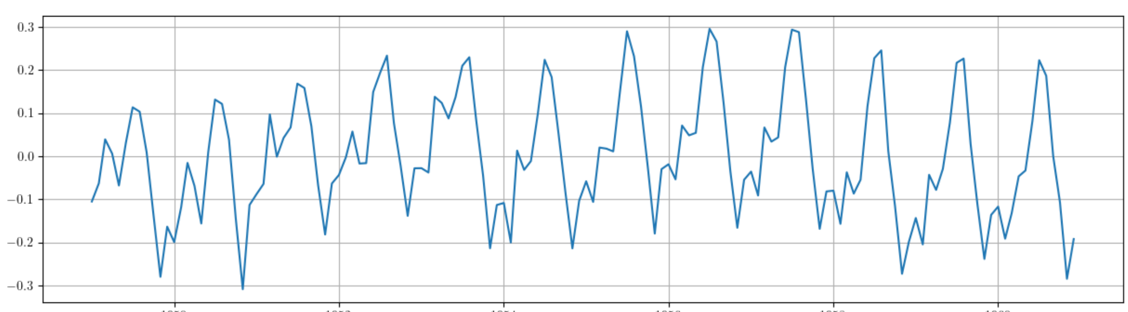
	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
cte	4.8137	0.023	206.648	0.000	4.768	4.860
time	0.0100	0.000	36.050	0.000	0.009	0.011

Omnibus:	3.750	**Durbin-Watson:**	0.587
Prob(Omnibus):	0.153	**Jarque-Bera (JB):**	2.722
Skew:	0.184	**Prob(JB):**	0.256
Kurtosis:	2.436	**Cond. No.**	168.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Componente irregular

```python
# Gráfico de los residuos del ajuste.
plt.grid()
plt.plot(results1.resid)
```



En este caso, el ajuste del modelo

$$\boldsymbol{y} = \boldsymbol{t} + \boldsymbol{e},$$

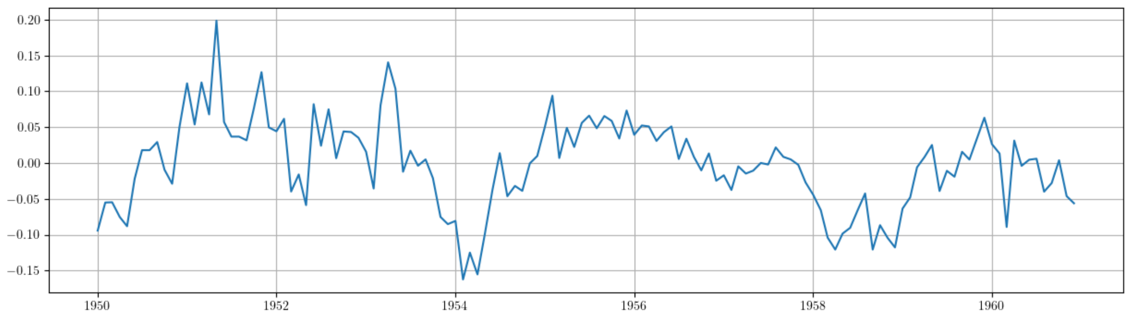
(donde $\boldsymbol{t}$ es una tendencia lineal) no es satisfactorio, ya que el *componente irregular*

$$\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{t}$$

no tiene la apariencia de realización de un proceso estacionario.

```
# Gráfico de la diferencia estacional de los residuos del ajuste.
plt.grid()
plt.plot(datosModelo1['ehatDiff12'])
```

Adicionalmente podemos ver que la diferencia de orden 12 del componente irregular parece mostrar un componente cíclico con un periodo de unos 4 años.



En el siguiente ejercicio probaremos con una tendencia cuadrática. . .

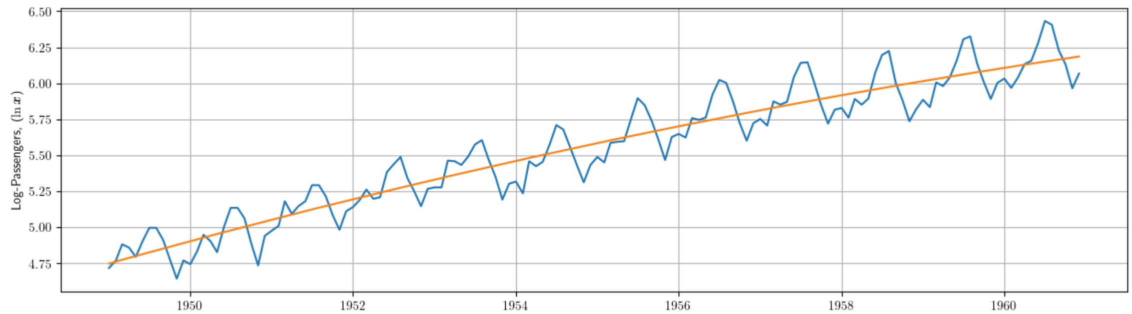
1.2. Tendencia determinista *cuadrática*

```
# creamos un DataFrame con el regresando y los regresores del modelo :results silent.
datosModelo2 = TransformedData[['dataLog']].copy()
nsample = len(datosModelo1)
datosModelo2['cte'] = [1]*nsample
datosModelo2['time'] = np.linspace(1, nsample, nsample)
datosModelo2['sq_time'] = [t**2 for t in datosModelo2['time']]
# Ajustamos por MCO una tendencia cuadrática a los datos.
model2 = sm.OLS(datosModelo1['dataLog'], datosModelo2[['cte', 'time', 'sq_time']])
results2 = model2.fit()
```

```
# Añadimos al DataFrame 'datosModelo2' la tendencia ajustada, los residuos y la diferencia estacional de los residuos.
datosModelo2['yhat'] = results2.fittedvalues
datosModelo2['ehat'] = results2.resid
datosModelo2['ehatDiff12'] = datosModelo2['ehat'].diff(12)
```

```
# Dibujamos los datos junto a la tendencia estimada.
plt.plot(datosModelo1['dataLog'])
plt.plot(results2.fittedvalues)
plt.grid()
plt.ylabel(r"Log-Passengers, ($\ln\boldsymbol{x}$) ")
```

$$\ln y_t = \underbrace{\beta_1 + \beta_2 \cdot t + \beta_3 \cdot t^2}_{\text{tendencia}} + e_t; \quad t = 1:114$$



$$\widehat{\ln y_t} = 4{,}7364 + (0{,}0132) \cdot t + (-2{,}191e - 05) \cdot t^2, \qquad t = 1:114$$

```python
print(results2.summary())
```

Dep. Variable:	dataLog	**R-squared:**	0.907
Model:	OLS	**Adj. R-squared:**	0.906
Method:	Least Squares	**F-statistic:**	691.0
Date:	Thu, 03 Apr 2025	**Prob (F-statistic):**	1.37e-73
Time:	20:33:38	**Log-Likelihood:**	85.260
No. Observations:	144	**AIC:**	-164.5
Df Residuals:	141	**BIC:**	-155.6
Df Model:	2		
Covariance Type:	nonrobust		

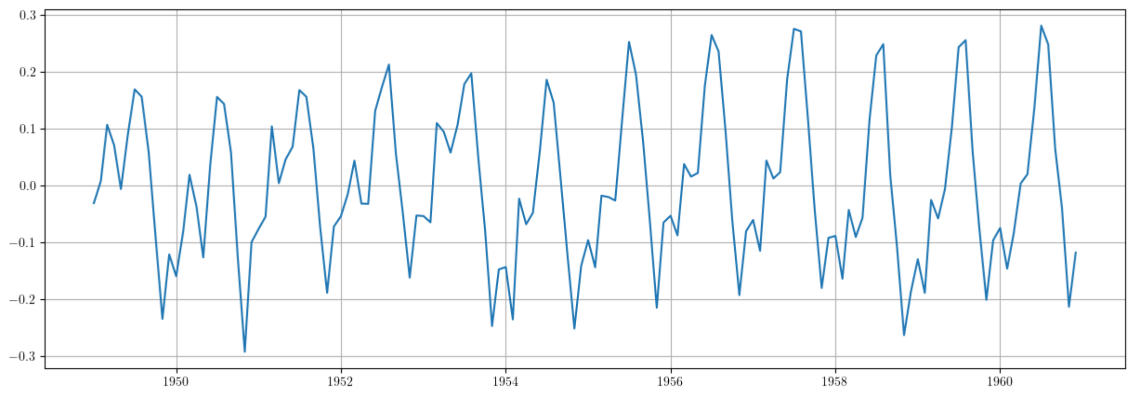
	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
cte	4.7364	0.034	138.117	0.000	4.669	4.804
time	0.0132	0.001	12.112	0.000	0.011	0.015
sq_time	-2.191e-05	7.29e-06	-3.004	0.003	-3.63e-05	-7.49e-06

Omnibus:	4.978	**Durbin-Watson:**	0.624
Prob(Omnibus):	0.083	**Jarque-Bera (JB):**	3.317
Skew:	0.205	**Prob(JB):**	0.190
Kurtosis:	2.380	**Cond. No.**	2.85e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.85e+04. This might indicate that there are strong multicollinearity or other numerical problems.

Componente irregular

```python
plt.grid()
plt.plot(results2.resid)
```



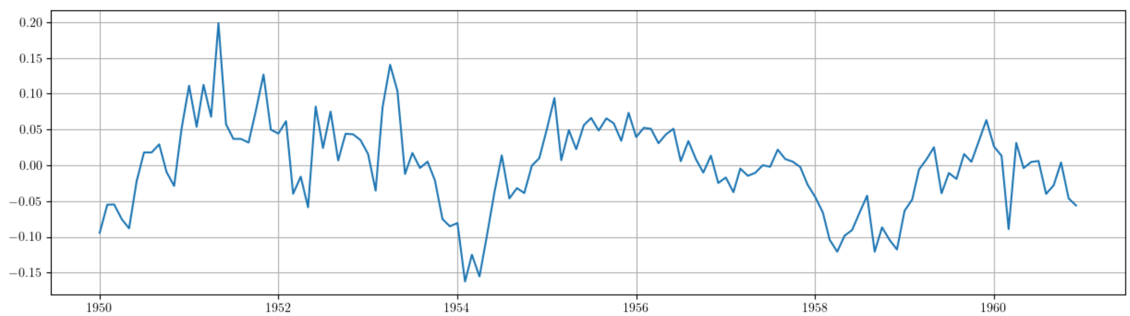
De manera análoga al caso anterior, el ajuste del modelo

$$\boldsymbol{y} = \boldsymbol{t} + \boldsymbol{e}$$

(donde $\boldsymbol{t}$ ahora es una *tendencia cuadrática*) tampoco es satisfactorio, puesto que el componente irregular $\boldsymbol{e}$ sigue sin parecerse a la realización de un proceso estacionario.

```
plt.grid()
plt.plot(datosModelo2['ehatDiff12'])
```

También en este modelo la diferencia de orden 12 del componente irregular muestra un componente cíclico con un periodo de unos 4 años.



Para obtener una *tendencia-ciclo* que capte este ciclo son necesarios procedimientos más sofisticados (por ejemplo TRAMO-SEATS, o X13-ARIMA, o STAMP, o LDHR, o E4, etc.) que estiman tendencias y componentes estacionales estocásticos.

En el siguiente ejercicio estimaremos un **componente estacional determinista** (junto a una tendencia cuadrática determinista).

1.3. Tendencia cuadrática más estacionalidad determinista mediante *dummies*

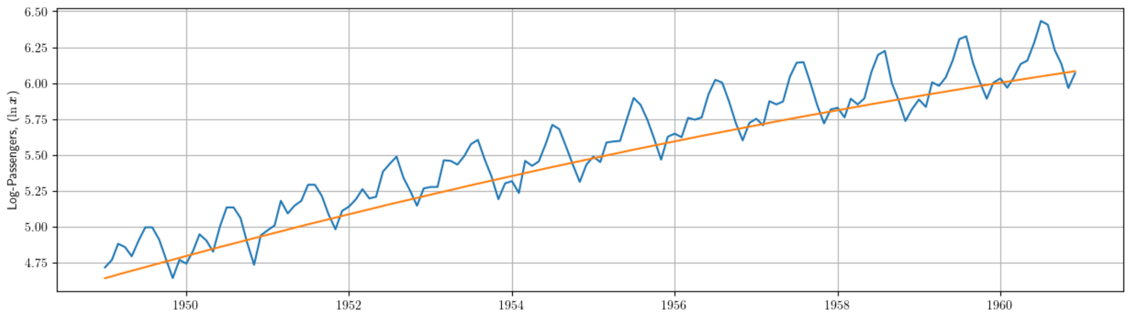
```
# Creamos un dataframe con los datos y los regresores 'cte', 't' y ' :results silentt^2'
df = TransformedData[['dataLog']].copy()
nsample = len(df)
df['cte']     = [1]*nsample
df['time']    = np.linspace(1, nsample, nsample)
df['sq_time'] = [t**2 for t in df['time']]
```

```python
# Creamos las /dummies/ estacionales
from statsmodels.tsa.deterministic import Seasonality
seas_gen = Seasonality(12, initial_period=1)
seasonalDummies = seas_gen.in_sample(df.index)
```

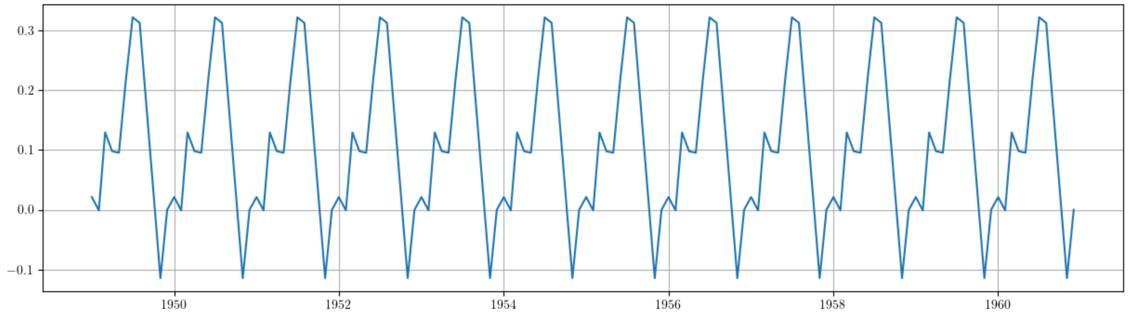
```python
# Creamos un dataframe con el regresando y todos los regresores del modelo
datosModelo3 = pd.concat([df, seasonalDummies],axis=1)
# realizamos la regresión de la primera columna ('dataLog') sobre el resto de columnas del dataframe.
model3 = sm.OLS(datosModelo3['dataLog'], datosModelo3.iloc[:,1:-1])
results3 = model3.fit()
```

```python
# La combinación lineal de los regresores 'cte', 'time' y 'sq_time' usando los correspondientes
# parámetros estimados nos da el componente de tendencia (determinista) estimado.
TrendComp = datosModelo3[['cte','time','sq_time']].dot(results3.params[['cte','time','sq_time']])
```

```python
rcParams['figure.figsize'] = 15,4
plt.plot(datosModelo1['dataLog'])
plt.plot(TrendComp)
plt.grid()
plt.ylabel(r"Log-Passengers, ($\ln\boldsymbol{x}$) ")
```

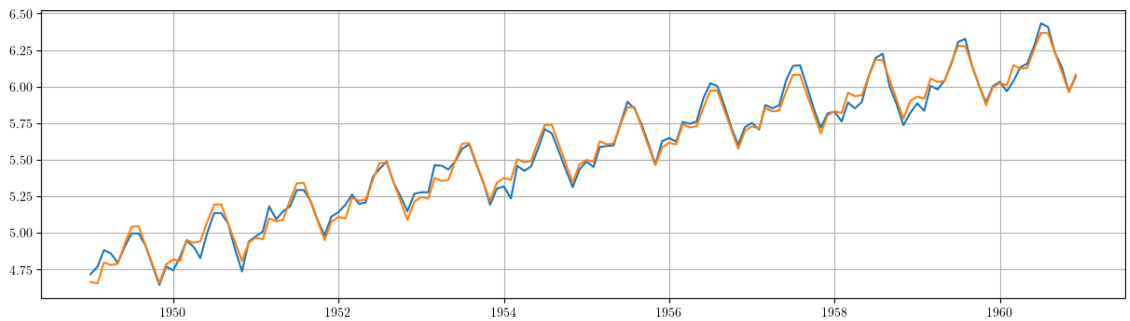



```python
SeasonalComp = (seasonalDummies.iloc[:,:-1]).dot(results3.params[3:])
plt.grid()
plt.plot(SeasonalComp)
```

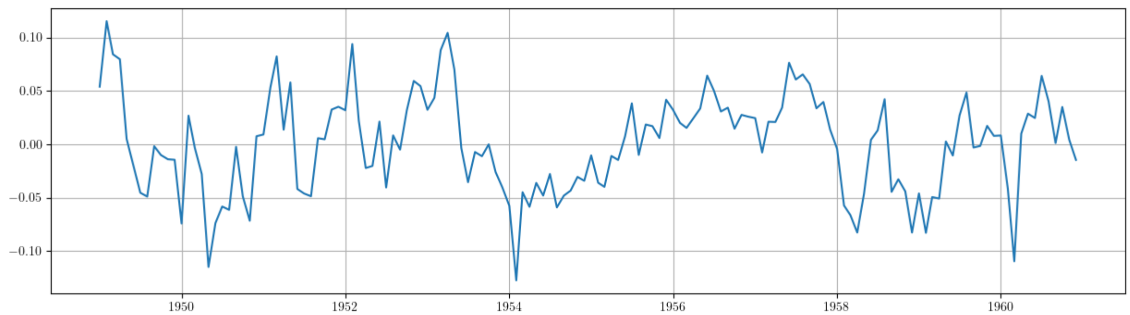


1.3.1. Ajuste y componente irregular $e = y - t - s$

```python
plt.grid()
plt.plot(datosModelo3['dataLog'])
plt.plot(TrendComp + SeasonalComp)
```




```
plt.grid()
plt.plot(results3.resid)
```



1.3.2. Valoración de modelos con componentes deterministas

- Estos modelos resultan útiles para realizar un análisis descriptivo.
- Pero suelen funcionar bastante mal como herramienta de predicción:
  - no tienen en cuenta la dependencia inter-temporal de los datos (se estiman mediante regresión, es decir, como si los datos fueran de sección cruzada)
  - Por ejemplo, a la hora de prever el dato de enero de 1961, en este modelo pesa tanto el dato de enero de 1949 como el dato de enero de 1960.

En general, para que los modelos funcionen bien en predicción deben *dar un mayor peso a los datos recientes* frente a los datos alejados en el tiempo.

Pero sigamos explorando este modelo. . .

Hay parámetros no significativos. . . (p-valores para dummies enero, febrero y octubre).

Dep. Variable:	dataLog	**R-squared:**	0.989
Model:	OLS	**Adj. R-squared:**	0.988
Method:	Least Squares	**F-statistic:**	912.7
Date:	Thu, 03 Apr 2025	**Prob (F-statistic):**	7.45e-121
Time:	20:33:38	**Log-Likelihood:**	239.70
No. Observations:	144	**AIC:**	-451.4
Df Residuals:	130	**BIC:**	-409.8
Df Model:	13		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
cte	4.6301	0.018	253.331	0.000	4.594	4.666
time	0.0132	0.000	33.877	0.000	0.012	0.014
sq_time	-2.148e-05	2.6e-06	-8.265	0.000	-2.66e-05	-1.63e-05
s(1,12)	0.0213	0.020	1.082	0.281	-0.018	0.060
s(2,12)	-0.0009	0.020	-0.048	0.962	-0.040	0.038
s(3,12)	0.1291	0.020	6.555	0.000	0.090	0.168
s(4,12)	0.0977	0.020	4.962	0.000	0.059	0.137
s(5,12)	0.0953	0.020	4.838	0.000	0.056	0.134
s(6,12)	0.2174	0.020	11.041	0.000	0.178	0.256
s(7,12)	0.3213	0.020	16.323	0.000	0.282	0.360
s(8,12)	0.3120	0.020	15.855	0.000	0.273	0.351
s(9,12)	0.1675	0.020	8.511	0.000	0.129	0.206
s(10,12)	0.0295	0.020	1.497	0.137	-0.009	0.068
s(11,12)	-0.1141	0.020	-5.797	0.000	-0.153	-0.075

Omnibus:	0.334	**Durbin-Watson:**	0.648
Prob(Omnibus):	0.846	**Jarque-Bera (JB):**	0.430
Skew:	-0.108	**Prob(JB):**	0.806
Kurtosis:	2.843	**Cond. No.**	1.17e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.17e+05. This might indicate that there are strong multicollinearity or other numerical problems.

podemos eliminarlos secuencialmente (quitando cada vez la variable de mayor p-valor)

```
import operator
def remove_most_insignificant(df, results):
    # use operator to find the key which belongs to the maximum value in the dictionary:
    max_p_value = max(results.pvalues.items(), key=operator.itemgetter(1))[0]
    # this is the feature you want to drop:
    df.drop(columns = max_p_value, inplace = True)
    return df
```

```
y = datosModelo3['dataLog']
X = datosModelo3.iloc[:,1:-1]
significacion = 0.05
insignificant_feature = True
while insignificant_feature:
        model4 = sm.OLS(y, X)
        results4 = model4.fit()
        significant = [p_value < significacion for p_value in results4.pvalues]
        if all(significant):
            insignificant_feature = False
        else:
            if X.shape[1] == 1:  # if there's only one insignificant variable left
                print('No significant features found')
                results4 = None
                insignificant_feature = False
            else:
                X = remove_most_insignificant(X, results4)

print(results4.summary())
```

Dep. Variable:	dataLog	**R-squared:**	0.989
Model:	OLS	**Adj. R-squared:**	0.988
Method:	Least Squares	**F-statistic:**	1181.
Date:	Thu, 03 Apr 2025	**Prob (F-statistic):**	1.19e-124
Time:	20:33:39	**Log-Likelihood:**	237.72
No. Observations:	144	**AIC:**	-453.4
Df Residuals:	133	**BIC:**	-420.8
Df Model:	10		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
cte	4.6425	0.013	344.431	0.000	4.616	4.669
time	0.0132	0.000	33.805	0.000	0.012	0.014
sq_time	-2.149e-05	2.61e-06	-8.248	0.000	-2.66e-05	-1.63e-05
s(3,12)	0.1166	0.016	7.479	0.000	0.086	0.147
s(4,12)	0.0852	0.016	5.467	0.000	0.054	0.116
s(5,12)	0.0828	0.016	5.309	0.000	0.052	0.114
s(6,12)	0.2049	0.016	13.140	0.000	0.174	0.236
s(7,12)	0.3088	0.016	19.805	0.000	0.278	0.340
s(8,12)	0.2996	0.016	19.211	0.000	0.269	0.330
s(9,12)	0.1550	0.016	9.941	0.000	0.124	0.186
s(11,12)	-0.1265	0.016	-8.111	0.000	-0.157	-0.096

Omnibus:	1.502	**Durbin-Watson:**	0.691
Prob(Omnibus):	0.472	**Jarque-Bera (JB):**	1.504
Skew:	-0.241	**Prob(JB):**	0.471
Kurtosis:	2.867	**Cond. No.**	5.81e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.81e+04. This might indicate that there are strong multicollinearity or other numerical problems.

Pero esta inferencia es incorrecta. En presencia de auto-correlación **la estimación por defecto de las desviaciones típicas del estimador MCO es incorrecta**. Veámoslo:

2. Perturbaciones no esféricas

Considere el modelo $\boldsymbol{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{U}$. Bajo los supuestos habituales

$$E(\boldsymbol{U} \mid \mathbf{X}) = \mathbf{0}, \quad Var(\boldsymbol{U} \mid \mathbf{X}) = \sigma^2\mathbf{I} \quad \text{y} \quad E(\mathbf{X'X}) \text{ es invertible}$$

el estimador $\widehat{\boldsymbol{\beta}} = (\mathbf{X'X})^{-1}\mathbf{X'}\boldsymbol{Y}$ es insesgado y eficiente, con varianza

$$Var(\widehat{\boldsymbol{\beta}} \mid \mathbf{X}) = \sigma^2(\mathbf{X'X})^{-1}$$

Pero si las perturbaciones $\boldsymbol{U}$ del modelo son heterocedásticas y/o autocorreladas

$$Var(\boldsymbol{U} \mid \mathbf{X}) = \boldsymbol{\Sigma} \neq \sigma^2\mathbf{I}$$

entonces el estimador $\widehat{\boldsymbol{\beta}}$, aunque insesgado, ya no es eficiente; y su varianza es

$$Var(\widehat{\boldsymbol{\beta}} \mid \mathbf{X}) = Var(\widehat{\boldsymbol{\beta}} - \mathbf{I}\boldsymbol{\beta} \mid \mathbf{X}) = (\mathbf{X'X})^{-1}\mathbf{X'}\boldsymbol{\Sigma}\mathbf{X}(\mathbf{X'X})^{-1}.$$

2.1. Test de autocorrelación de Breusch y Godfrey

El tests Breusch-Godfrey (y el Durbin-Watson) contrasta la H_0 de *no autocorrelación*. Considere el *modelo de regresión lineal*

$$Y_t = \beta_1 + \beta_2 X_{t,1} + \cdots + \beta_k X_{t,k+1} + U_t \tag{1}$$

donde las perturbaciones $\boldsymbol{U}$ quizá siguen un esquema auto-regresivo $AR(p)$:

$$U_t = \rho_1 U_{t-1} + \rho_2 U_{t-2} + \cdots + \rho_p U_{t-p} + \varepsilon_t$$

- **Paso 1**. Obtener los errores $\hat{\boldsymbol{e}}$ de ajuste MCO de (1) (muestra de tamaño T)
- **Paso 2**. Calcular el R^2 de la *regresión auxiliar* de los errores $\hat{\boldsymbol{e}}$ sobre los regresores del modelo original (1) y sobre los p primeros retardos de $\hat{\boldsymbol{e}}$.

$$\hat{e}_t = \alpha_0 + \alpha_1 X_{t,1} + \cdots \alpha_k X_{t,k} + \rho_1 \hat{e}_{t-1} + \rho_2 \hat{e}_{t-2} + \cdots + \rho_p \hat{e}_{t-p} + \varepsilon_t$$

Asintóticamente y bajo la H_0 de *no autocorrelación*: $\rho_i = 0$ para todo i

$$nR^2 \sim \chi^2_p,$$

donde R^2 es el coeficiente de determinación de la regresión auxiliar y $n = T - p$.

```python
import statsmodels.stats.diagnostic as dg
#perform Breusch-Godfrey t :results silentest of order p = 3
arbg = dg.acorr_breusch_godfrey(results4, nlags=3, store=True)
arbg[:1]
```

62.71194311498567

Dep. Variable:	y	**R-squared:**	0.435
Model:	OLS	**Adj. R-squared:**	0.379
Method:	Least Squares	**F-statistic:**	7.715
Date:	Thu, 03 Apr 2025	**Prob (F-statistic):**	3.54e-11
Time:	20:33:39	**Log-Likelihood:**	278.89
No. Observations:	144	**AIC:**	-529.8
Df Residuals:	130	**BIC:**	-488.2
Df Model:	13		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
const	-0.0001	0.005	-0.027	0.979	-0.010	0.010
x1	3.05e-05	0.000	0.103	0.918	-0.001	0.001
x2	-2.318e-07	1.98e-06	-0.117	0.907	-4.15e-06	3.69e-06
x3	0.0058	0.012	0.477	0.634	-0.018	0.030
x4	0.0024	0.012	0.199	0.843	-0.021	0.026
x5	-0.0017	0.012	-0.144	0.886	-0.025	0.022
x6	-0.0003	0.012	-0.027	0.978	-0.024	0.023
x7	-0.0003	0.012	-0.027	0.979	-0.024	0.023
x8	-0.0003	0.012	-0.026	0.979	-0.024	0.023
x9	-0.0003	0.012	-0.026	0.979	-0.024	0.023
x10	-0.0109	0.012	-0.908	0.366	-0.035	0.013
x11	-0.0001	0.005	-0.027	0.979	-0.010	0.010
x12	0.6214	0.089	6.973	0.000	0.445	0.798
x13	0.1333	0.105	1.274	0.205	-0.074	0.340
x14	-0.1042	0.090	-1.160	0.248	-0.282	0.074

Omnibus:	4.932	**Durbin-Watson:**	2.013
Prob(Omnibus):	0.085	**Jarque-Bera (JB):**	4.703
Skew:	-0.442	**Prob(JB):**	0.0952
Kurtosis:	3.062	**Cond. No.**	4.82e+19

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 5.41e-30. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

- Valor del estadístico: 62,7119 (p-valor: $1{,}55e-13$)
- x_{12} corresponde al primer retardo en la regresión auxiliar y es muy significativo

2.2. Test de Durbin-Watson

El test de Durbin-Watson contrasta la autocorrelación de orden uno. Para muestras grandes, el test es aproximadamente igual a $2(1-\hat{\rho})$, donde $\hat{\rho}$ es la autocorrelación de orden uno de los residuos. Por tanto, valores del test próximos a 2 indican no autocorrelación, valores próximos a 0 indican fuerte autocorrelación positiva y valores próximos a 4 indican fuerte autocorrelación negativa.

2.3. Errores estándar robustos

Un procedimiento adecuado en presencia de autocorrelación y muestras grandes consiste en usar errores estándar *robustos* (**HAC** - heteroscedasticity and autocorrelation robust covariance matrix) al realizar inferencia con la estimación de los parámetros.

1. las estimaciones serán insesgadas, consistentes pero ineficientes,
2. los residuos son los mismos y, por tanto, estarán autocorrelados,
3. pero la inferencia se realizará a partir de errores estándar robustos.

```python
y = datosModelo3['dataLog']
X = datosModelo3.iloc[:,1:-1]
model5 = sm.OLS(y, X)
results5 = model5.fit()
print(results5.get_robustcov_results(cov_type='HAC', maxlags=3, use_correction=True).summary())
```

Dep. Variable:	dataLog	**R-squared:**	0.989
Model:	OLS	**Adj. R-squared:**	0.988
Method:	Least Squares	**F-statistic:**	388.9
Date:	Thu, 03 Apr 2025	**Prob (F-statistic):**	3.26e-97
Time:	20:33:39	**Log-Likelihood:**	239.70
No. Observations:	144	**AIC:**	-451.4
Df Residuals:	130	**BIC:**	-409.8
Df Model:	13		
Covariance Type:	HAC		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
cte	4.6301	0.023	200.015	0.000	4.584	4.676
time	0.0132	0.001	19.130	0.000	0.012	0.015
sq_time	-2.148e-05	4.32e-06	-4.970	0.000	-3e-05	-1.29e-05
s(1,12)	0.0213	0.011	1.982	0.050	3.75e-05	0.043
s(2,12)	-0.0009	0.020	-0.046	0.963	-0.041	0.040
s(3,12)	0.1291	0.021	6.264	0.000	0.088	0.170
s(4,12)	0.0977	0.019	5.025	0.000	0.059	0.136
s(5,12)	0.0953	0.019	5.134	0.000	0.059	0.132
s(6,12)	0.2174	0.017	12.734	0.000	0.184	0.251
s(7,12)	0.3213	0.018	18.110	0.000	0.286	0.356
s(8,12)	0.3120	0.018	17.772	0.000	0.277	0.347
s(9,12)	0.1675	0.013	12.821	0.000	0.142	0.193
s(10,12)	0.0295	0.012	2.413	0.017	0.005	0.054
s(11,12)	-0.1141	0.011	-10.272	0.000	-0.136	-0.092

Omnibus:	0.334	**Durbin-Watson:**	0.648
Prob(Omnibus):	0.846	**Jarque-Bera (JB):**	0.430
Skew:	-0.108	**Prob(JB):**	0.806
Kurtosis:	2.843	**Cond. No.**	1.17e+05

Notes:
[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using 3 lags and with small sample correction
[2] The condition number is large, 1.17e+05. This might indicate that there are strong multicollinearity or other numerical problems.

- **Covariance type**: HAC (heteroscedasticity and autocorrelation robust covariance matrix)

Empleando errores estándar robustos (HAC), podemos reducir el modelo de manera más cuidadosa usando desviaciones típicas robustas. El modelo reducido es…

```python
y = datosModelo3['dataLog']
X = datosModelo3.iloc[:,1:-1]

significacion = 0.05

insignificant_feature = True
while insignificant_feature:
        results6      = sm.OLS(y, X).fit()
        robustResults = results6.get_robustcov_results(cov_type='HAC', maxlags=3, use_correction=True)
        robustPvalues = pd.Series(index=results6.pvalues.index, data=robustResults.pvalues)

        significant = [p_value < significacion for p_value in robustPvalues]


        if all(significant):
            insignificant_feature = False
        else:
            if X.shape[1] == 1:  # if there's only one insignificant variable left
```

```
            print('No significant features found')
            results6 = None
            insignificant_feature = False
        else:
            X = remove_most_insignificant(X, results6)
print(robustResults.summary())
```

Dep. Variable:	dataLog	**R-squared:**	0.989
Model:	OLS	**Adj. R-squared:**	0.988
Method:	Least Squares	**F-statistic:**	418.9
Date:	Thu, 03 Apr 2025	**Prob (F-statistic):**	3.59e-98
Time:	20:33:40	**Log-Likelihood:**	239.70
No. Observations:	144	**AIC:**	-453.4
Df Residuals:	131	**BIC:**	-414.8
Df Model:	12		
Covariance Type:	HAC		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
cte	4.6296	0.026	179.310	0.000	4.578	4.681
time	0.0132	0.001	19.195	0.000	0.012	0.015
sq_time	-2.148e-05	4.3e-06	-4.992	0.000	-3e-05	-1.3e-05
s(1,12)	0.0218	0.011	1.983	0.049	5.42e-05	0.044
s(3,12)	0.1296	0.016	7.867	0.000	0.097	0.162
s(4,12)	0.0982	0.018	5.496	0.000	0.063	0.134
s(5,12)	0.0957	0.019	4.917	0.000	0.057	0.134
s(6,12)	0.2178	0.018	11.837	0.000	0.181	0.254
s(7,12)	0.3218	0.019	16.955	0.000	0.284	0.359
s(8,12)	0.3125	0.019	16.603	0.000	0.275	0.350
s(9,12)	0.1680	0.015	11.071	0.000	0.138	0.198
s(10,12)	0.0299	0.015	2.014	0.046	0.001	0.059
s(11,12)	-0.1136	0.015	-7.616	0.000	-0.143	-0.084

Omnibus:	0.357	**Durbin-Watson:**	0.648
Prob(Omnibus):	0.837	**Jarque-Bera (JB):**	0.451
Skew:	-0.112	**Prob(JB):**	0.798
Kurtosis:	2.843	**Cond. No.**	8.24e+04

Notes:
[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using 3 lags and with small sample correction
[2] The condition number is large, 8.24e+04. This might indicate that there are strong multicollinearity or other numerical problems.

- Nótese que con HAC se aprecia que enero y octubre son significativos al 5 %
- Pero la estimación MCO no es eficiente en presencia de auto-correlación

2.4. Modelo del error

En el modelo $\boldsymbol{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{U}$, cuando las perturbaciones presentan heterocedasticidad y/o auto-correlación
$$Var(\boldsymbol{U} \mid \mathbf{X}) = \boldsymbol{\Sigma} \neq \sigma^2\mathbf{I},$$
por tanto, el Teorema de Gauss-Markov ya no es válido. En esta situación es posible explotar la estructura de la matriz $\boldsymbol{\Sigma}$ para minimizar la varianza del estimador.

En particular, el estimador lineal de mínima varianza es el estimador MCG (mínimos cuadrados generalizados)

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X'}\boldsymbol{\Sigma}^{-1}\mathbf{X})^{-1}\mathbf{X'}\boldsymbol{\Sigma}^{-1}\boldsymbol{y}$$

El problema es que, en general, la matriz $\boldsymbol{\Sigma}$ es desconocida.

Una solución es aplicar un procedimiento iterativo en el que con los errores de ajuste de una primera regresión que se estima la matriz $\boldsymbol{\Sigma}$. Con dicha matriz $\widehat{\boldsymbol{\Sigma}}$ se re-estima el modelo por MCG…

con los nuevos errores se re-estima $\mathbf{\Sigma}$... que se usa para re-estimar el modelo por MCG... y vuelta a empezar: con los nuevos errores...

El algoritmo se detiene cuando las estimaciones convergen a valores estables.

Cuando hicimos el Test de Breusch-Godfrey vimos que en la regresión auxiliar el primer retardo de los errores era significativo. Por tanto, vamos a indicar que las perturbaciones siguen un proceso AR(1). El decir, vamos a estimar el modelo

$$\ln y_t = \underbrace{\beta_1 + \beta_2 \cdot t + \beta_3 \cdot t^2}_{\text{tendencia}} + \underbrace{\alpha_1 S_{t1} + \alpha_3 S_{t3} + \cdots + \alpha_1 1 S_{t11}}_{\text{comp. estacional}} + \epsilon_t$$

donde las perturbaciones $\boldsymbol{\epsilon} = \{\epsilon_t\}$ siguen el modelo

$$\epsilon_t = \rho_1 \epsilon_{t-1} + e_t$$

(*en este caso la estimación (**GLSAR**) converge en 7 iteraciones*)

```
model = sm.GLSAR(y, X, rho=1) # :results silent rho=1 indica autocorrelación de orden uno
for i in range(7):
    results = model.fit()
    print("AR coefficients: {0}".format(model.rho))
    rho, sigma = sm.regression.yule_walker(results.resid,
                                           order=model.order)
    model = sm.GLSAR(y, X, rho)
```

```
print(results.summary())
```

Dep. Variable:	dataLog	**R-squared:**	0.958
Model:	GLSAR	**Adj. R-squared:**	0.954
Method:	Least Squares	**F-statistic:**	246.4
Date:	Thu, 03 Apr 2025	**Prob (F-statistic):**	3.79e-83
Time:	20:33:40	**Log-Likelihood:**	281.79
No. Observations:	143	**AIC:**	-537.6
Df Residuals:	130	**BIC:**	-499.1
Df Model:	12		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
cte	4.6157	0.031	146.687	0.000	4.553	4.678
time	0.0136	0.001	14.633	0.000	0.012	0.015
sq_time	-2.366e-05	5.99e-06	-3.951	0.000	-3.55e-05	-1.18e-05
s(1,12)	0.0179	0.009	2.044	0.043	0.001	0.035
s(3,12)	0.1306	0.011	12.014	0.000	0.109	0.152
s(4,12)	0.0995	0.014	7.228	0.000	0.072	0.127
s(5,12)	0.0973	0.015	6.389	0.000	0.067	0.127
s(6,12)	0.2194	0.016	13.773	0.000	0.188	0.251
s(7,12)	0.3233	0.016	20.029	0.000	0.291	0.355
s(8,12)	0.3140	0.016	19.716	0.000	0.282	0.345
s(9,12)	0.1692	0.015	11.123	0.000	0.139	0.199
s(10,12)	0.0308	0.014	2.238	0.027	0.004	0.058
s(11,12)	-0.1133	0.011	-10.427	0.000	-0.135	-0.092

Omnibus:	1.864	**Durbin-Watson:**	2.102
Prob(Omnibus):	0.394	**Jarque-Bera (JB):**	1.420
Skew:	-0.213	**Prob(JB):**	0.492
Kurtosis:	3.238	**Cond. No.**	3.85e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.85e+04. This might indicate that there are strong multicollinearity or other numerical problems.

```python
# este código realiza las mismas iteraciones que bloque de código de más arriba
model2 = sm.GLSAR(y, X, rho=1)
res = model2.iterative_fit(maxiter=7)
model2.rho
print(model2.fit().summary())
```