Contents

Econometría Aplicada. Lección 2

Marcos Bujosa

June 19, 2024

Carga de algunos módulos de python

```python
# Para trabajar con los datos y dibujarlos necesitamos cargar algunos módulos de python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib as mpl
import matplotlib.pyplot as plt   # data visualization
mpl.rc('text', usetex=True)
mpl.rc('text.latex', preamble=r'\usepackage{amsmath}')
import dataframe_image as dfi
import statsmodels.api as sm
```

```python
from sympy.printing.preview import preview

def repr_png(tex, ImgFile):
    preamble = "\\documentclass[preview]{standalone}\n" \
        "\\usepackage{booktabs,amsmath,amsfonts}\\begin{document}"
    preview(tex, filename=ImgFile, viewer='file', preamble=preamble, dvioptions=['-D','250'])
    #return open(ImgFile,'rb').read()
```

0.1 Internat. airline passengers: monthly totals in thousands. Jan 49 – Dec 60

Leemos los datos de un fichero `csv` y generamos un `dataframe` de pandas.

```python
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())
```

Vamos a crear un nuevo `dataframe` con los datos originales y varias transformaciones de los datos

```python
TransformedData = OrigData.copy()
TransformedData['dataLog'] = np.log(OrigData['Passengers'])
TransformedData['dataLogDiff'] = TransformedData['dataLog'].diff(1)
TransformedData['dataLogDiffDiff12'] = TransformedData['dataLogDiff'].diff(12)
```

1 Descomposición estructural de una serie temporal

Una estrategia para analizar series temporales es transformar los datos para

1. primero lograr que sean "***estacionarios***" y

2. después, mediante más transformaciones, lograr una secuencia de "**datos *i.i.d***" (este segundo paso aún no lo hemos abordado)

(*recuerde que las expresiones "datos estacionarios" o "datos i.i.d." son un abuso del lenguaje*).

Pero existe otro enfoque que pretende descomponer la serie temporal en los siguientes componentes *"no observables"* (o 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 las oscilaciones periódicas que se repiten regularmente en ciclos estacionales (de año en año, o de semana en semana, 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, ya que debe cumplir la siguiente identidad: $\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{t} - \boldsymbol{c} - \boldsymbol{s}$.

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

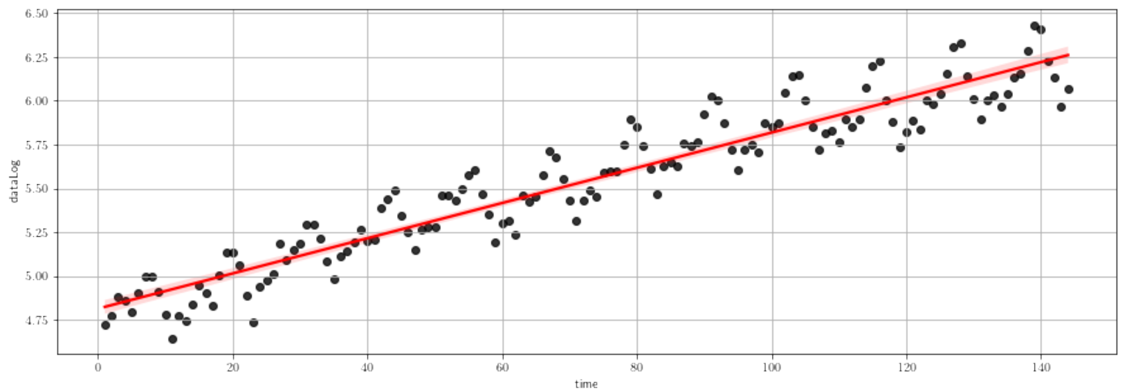
1.1 Tendencia determinista *lineal*

```python
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
import seaborn as sns
from matplotlib import rcParams
rcParams['figure.figsize'] = 15,5
plt.grid()
ax = sns.regplot(x="time", y="dataLog", data=datosModelo1,
                 scatter_kws={"color": "black"}, line_kws={"color": "red"})
fig = ax.get_figure()
#fig.savefig("./img/airlinepass+linearTrend.png")
```

El modelo de tendencia más simple es la recta de regresión donde el regresor es el propio índice t que indica el instante del tiempo de cada dato:

$$\ln y_t = \beta_1 \cdot 1 + \beta_2 \cdot t + e_t; \quad t = 1:114$$




```python
round(results1.params['cte'],4)
```

```python
round(results1.params['time'],4)
```

```
$$\widehat{\ln{y_t}}=<<Cte-ajuste-tendencia-lineal()>>+<<Pte-ajuste-tendencia-lineal()>>\cdot\big(t\big), \qquad t=1:114$$
```

$$\widehat{\ln{y_t}}=4.8137+0.01\cdot\big(t\big), \qquad t=1:114$$

```python
# print(results.summary()) Esta es la forma habitual de ver los resultados
repr_png(results1.summary().as_latex(),  "./img/resultsModel1.png") # pero emplearé esta para importar los resultados como image
```

Dep. Variable:	dataLog	**R-squared:**	0.902
Model:	OLS	**Adj. R-squared:**	0.901
Method:	Least Squares	**F-statistic:**	1300.
Date:	Wed, 19 Jun 2024	**Prob (F-statistic):**	2.41e-73
Time:	10:47:03	**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.

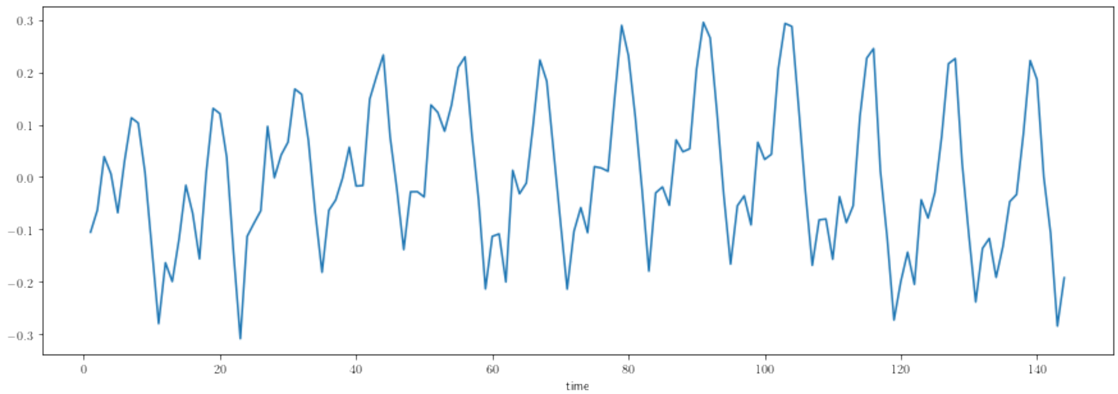
cte 4.813668 time 0.010048 dtype: float64

```python
ax = sns.lineplot(data=datosModelo1, x="time", y=results1.resid)
fig = ax.get_figure()
#fig.savefig("./img/airlinepass+irreg.png")
```

En este caso, el modelo

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

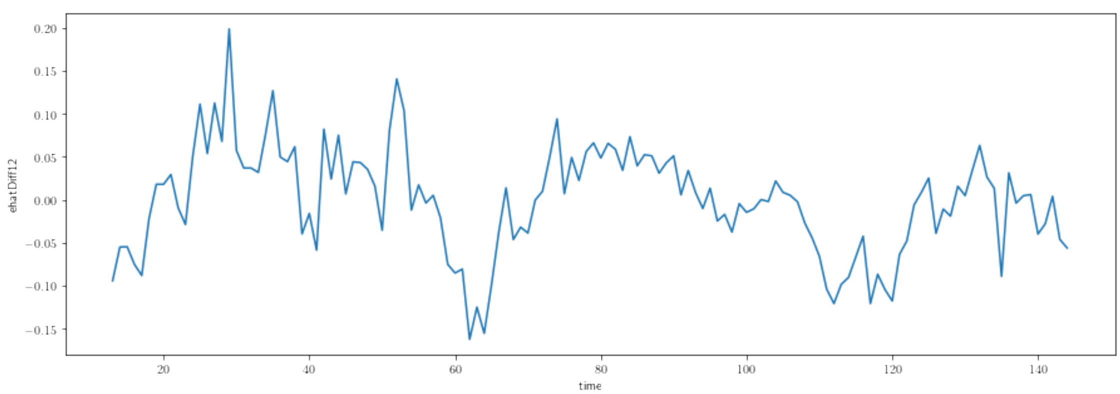
donde $\boldsymbol{t}$ es una tendencia lineal no es un ajuste satisfactorio, pues el componente irregular $\boldsymbol{e}$ no parece la realización de un proceso estacionario.




```
datosModelo1['yhat'] = datosModelo1['cte']*results1.params['cte']+datosModelo1['time']*results1.params['time']
datosModelo1['ehat'] = results1.resid
datosModelo1['ehatDiff12'] = datosModelo1['ehat'].diff(12)
```

```
ax = sns.lineplot(data=datosModelo1, x="time", y=datosModelo1['ehatDiff12'])
fig = ax.get_figure()
#fig.savefig("./img/airlinepass+irregDiff12.png")
```

Diferencia de orden 12 del componente irregular parece mostrar un componente cíclico.



Probemos con una tendencia cuadrática

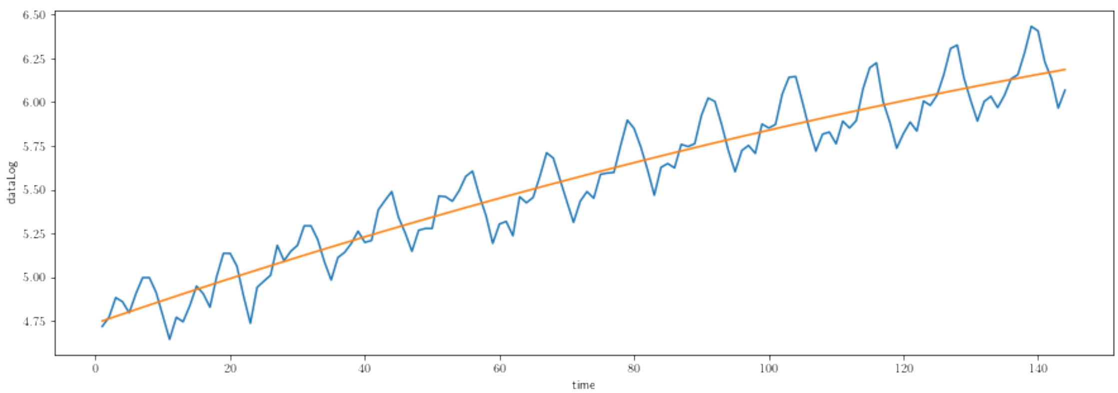
1.2 Tendencia determinista *cuadrática*

```
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']]
model2 = sm.OLS(datosModelo1['dataLog'], datosModelo2[['cte', 'time', 'sq_time']])
results2 = model2.fit()
```

```python
datosModelo2['yhat'] = datosModelo2['cte']*results2.params['cte']+datosModelo2['time']*results2.params['time']+datosModelo2['sq_
datosModelo2['ehat'] = results2.resid
datosModelo2['ehatDiff12'] = datosModelo2['ehat'].diff(12)
```

```python
ax = sns.lineplot(data=datosModelo2, x="time", y="dataLog")
ax = sns.lineplot(data=datosModelo2, x="time", y="yhat")
fig = ax.get_figure()
#fig.savefig("./img/airlinepass+quadraticTrend.png")
```

$$\ln y_t = \beta_1 \cdot 1 + \beta_2 \cdot t + \beta_3 \cdot t^2 + e_t; \quad t = 1:114$$




```python
# print(results.summary()) Esta es la forma habitual de ver los resultados
repr_png(results2.summary().as_latex(), "./img/resultsModel2.png") # pero usaré esta
```

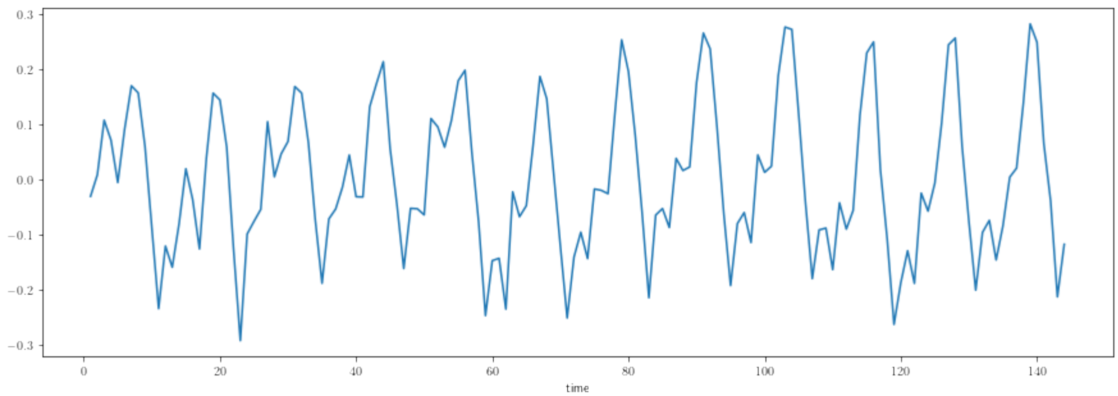
Dep. Variable:	dataLog	**R-squared:**	0.907
Model:	OLS	**Adj. R-squared:**	0.906
Method:	Least Squares	**F-statistic:**	691.0
Date:	Wed, 19 Jun 2024	**Prob (F-statistic):**	1.37e-73
Time:	10:47:04	**Log-Likelihood:**	85.260
No. Observations:	144	**AIC:**	-164.5
Df Residuals:	141	**BIC:**	-155.6
Df Model:	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.

```python
ax = sns.lineplot(data=datosModelo2, x="time", y=results2.resid)
fig = ax.get_figure()
#fig.savefig("./img/airlinepass+irreg2.png")
```



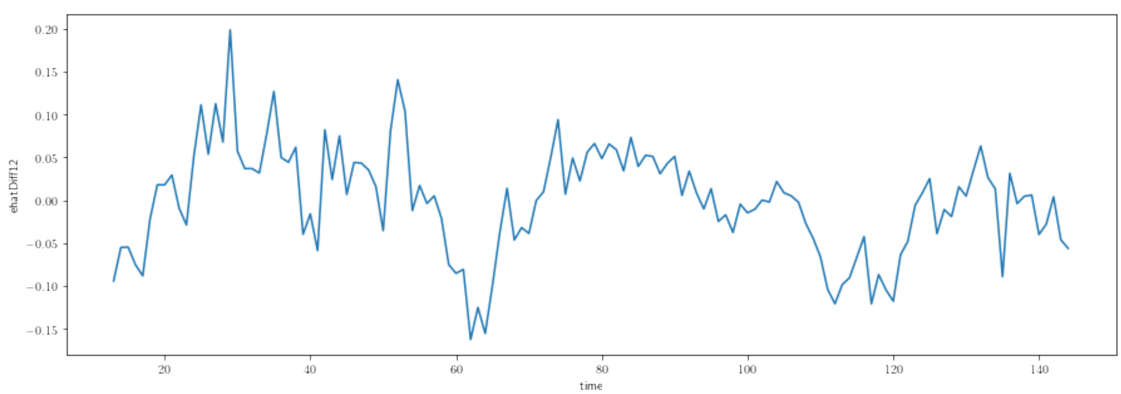
En este caso, el modelo

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

donde $\boldsymbol{t}$ es una tendencia cuadrática tampoco es un ajuste satisfactorio, pues el componente irregular $\boldsymbol{e}$ sigue sin parecer la realización de un proceso estacionario.

```
ax = sns.lineplot(data=datosModelo2, x="time", y=datosModelo2['ehatDiff12'])
fig = ax.get_figure()
#fig.savefig("./img/airlinepass+irregDiff12-2.png")
```

La diferencia de orden 12 del componente irregular de este segundo modelo sigue mostrando un componente cíclico.



Para obtener una *tendencia-ciclo* que capte este ciclo, necesitamos procedimientos más sofisticados (TRAMO-SEATS, X13-ARIMA, STAMP, LDHR, etc.)… y que estiman tendencias estocásticas (en lugar de tendencias deterministas con en los dos ejemplos vistos).

Pasemos a estimar un componente estacional

1.3 Componente estacional determinista mediante *dummies*

```
datosModelo3 = TransformedData[['dataLog']].copy()
nsample = len(datosModelo1)
datosModelo3['cte'] = [1]*nsample
datosModelo3['time'] = np.linspace(1, nsample, nsample)
datosModelo3['sq_time'] = [t**2 for t in datosModelo3['time']]
```

Creamos las *dummies* estacionales

```
from statsmodels.datasets import sunspots
from statsmodels.tsa.deterministic import Seasonality
seas_gen = Seasonality(12, initial_period=1)
seas_gen.in_sample(datosModelo3.index)
```

```
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']]
model2 = sm.OLS(datosModelo1['dataLog'], datosModelo2[['cte', 'time', 'sq_time']])
results2 = model2.fit()
```

```
datosModelo2['yhat'] = datosModelo2['cte']*results2.params['cte']+datosModelo2['time']*results2.params['time']+datosModelo2['sq_
datosModelo2['ehat'] = results2.resid
datosModelo2['ehatDiff12'] = datosModelo2['ehat'].diff(12)
```