

```
[1] 1 from google.colab import drive
2
3 drive.mount("/content/gdrive")

Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).

1 %cd "/content/gdrive/MyDrive/Bloque 2 IA/Estadística/Entregable 2"
2 !ls

/content/gdrive/MyDrive/Bloque 2 IA/Estadística/Entregable 2
'Entregable 2.ipynb'  Entregable2.ipynb  IGAE.csv  Indicadores.xlsx

[3] 1 import pandas as pd
2 import numpy as np
3 from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

Entregable 2

Lectura de datos

```
1 df = pd.read_csv('IGAE.csv')
2 df
```

	Periodo	Valor
0	2022/08	113.320302
1	2022/07	111.268587
2	2022/06	112.797501
3	2022/05	114.442424
4	2022/04	109.529866
...
351	1993/05	63.612900
352	1993/04	61.865984
353	1993/03	63.943249
354	1993/02	61.022521
355	1993/01	60.407691

356 rows x 2 columns

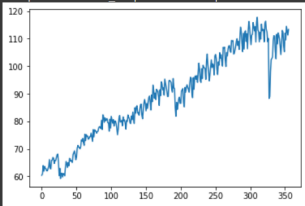
Sorteamos los valores según el período, recetamos y dropeamos los index para poder reordenar de forma periódica los datos

```
[5] 1 df = df.sort_values(by='Periodo')
2 df = df.reset_index(drop=True)
```

Realizar pruebas estadísticas de estacionariedad y decide el tipo de transformación de la variable

Podemos ver según el plot de 'Valor' que nuestra variable es no estacionaria, ya que tiene un crecimiento visible

```
1 df['Valor'].plot()
```



<matplotlib.axes._subplots.AxesSubplot at 0x7ff45f1d4950>

Calculamos la diferencia entre el logaritmo de los valores anuales del crecimiento índice de volumen físico base, esto con el fin de estacionarizar los datos

```
1 df['lnair'] = np.log(df['Valor']) - np.log(df['Valor']).shift(12)
2 df
```

	Periodo	Valor	lnair
0	1993/01	60.407691	NaN
1	1993/02	61.022521	NaN
2	1993/03	63.943249	NaN
3	1993/04	61.865984	NaN
4	1993/05	63.612900	NaN
...
351	2022/04	109.529866	0.013414
352	2022/05	114.442424	0.021001
353	2022/06	112.797501	0.014522
354	2022/07	111.268587	0.012578
355	2022/08	113.320302	0.055340

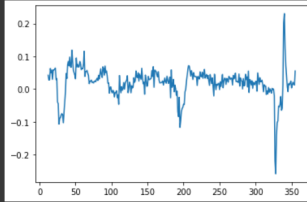
356 rows x 3 columns

Dropeamos nans

```
[8] 1 df2 = df.dropna().copy()
```

```
[9] 1 df2['lnair'].plot()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7ff45f0cd350>



Con este grafico, podemos ver cierta tendencia a estacionaridad, pero para poder comprobar esto, usaremos el Dickey-Fuller test

<https://www.statology.org/dickey-fuller-test-python/>

```
[10] 1 from statsmodels.tsa.stattools import adfuller
2
3 p_value = adfuller(df2['lnair'])
4 p_value[1]
```

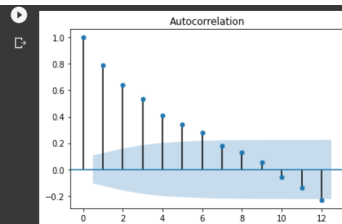
0.003199714148110552

Podemos ver que tenemos un p-value de 0.00320, por lo que posemos usar los valores de $D = 1$ y $d = 0$ para calcular el porcentaje de cambio anual segun el mes, debido a que nuestra variable ya es estacionaria y con esto proseguimos a usar un modelo ARIMA-SARIMA

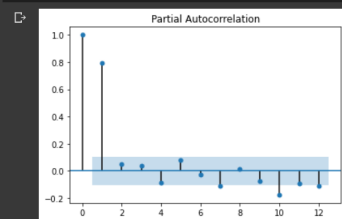
Calibra un modelo ARIMA-SARIMA basado en pruebas y gráficos de autocorrelaciones . Explica claramente el proceso de calibración y la razón de cómo va modificando los parámetros.

Realizamos el grafico de autocorrelaciones (ACF) y una autocorrelacion parcial de los datos(PACF) para conocer la auccorrelacion de los datos con sus lags

```
[11] 1 plot_acf(df2['lnair'], lags = 12)
```



```
1 plot_pacf(df2['lnair'], lags = 12)
```



En el grafico de autocorrelacion, podemos ver que esta va descendiendo, entre mayor el lag, y nos damos cuenta que los lags anteriores a 7 se encuentran por encima del 95% de significancia.

Por otro lado, en la autocorrelacion parcial unicamente se ve una autocorrelacion explictiva en el primer lag.

Debido a esto, tomaremos un valor de $p = 1$ y $q = 0$

Estimacion del modelo ARIMA-SARIMA

Variables para el modelo ARIMA-SARIMA

Modelo 1

```
[13] 1 #Arima(1,0,0)
2 p = 1
3 q = 0
4 d = 0
5
6 #SARIMA(0,0,1,12)
7 P = 0
8 Q = 0
9 D = 1
10 num_per = 12
```

```
1 import statsmodels.api as sm
2
3 df2['y'] = np.log(df2['Valor'])
4 model1 = sm.tsa.statespace.SARIMAX(df2['y'], order = (p, d, q), seasonal_order = (P, D, Q, num_per), trend = 'c', simple_differencing = True)
5 aux = model1.fit()
6 print(aux.summary())
```

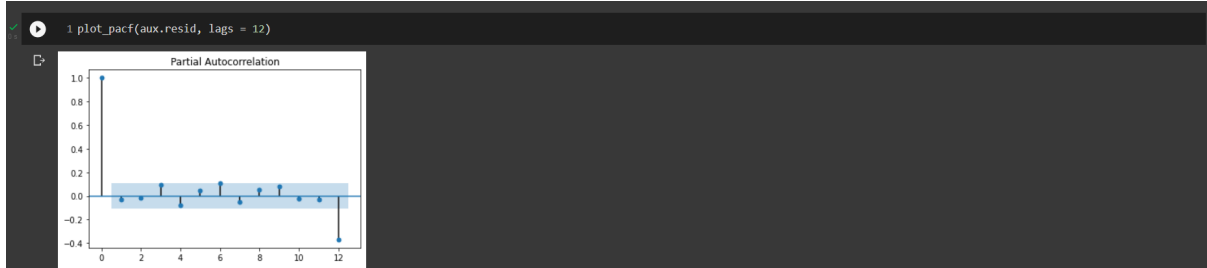
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:579: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:579: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
ignored when e.g. forecasting.', ValueWarning)

SARIMAX Results

Dep. Variable:	DS12.y	No. Observations:	332
Model:	SARIMAX(1, 0, 0)x(0, 0, 0, 12)	Log Likelihood	714.064
Date:	Sat, 26 Nov 2022	AIC	-1422.127
Time:	04:11:40	BIC	-1410.712
Sample:	0	HQIC	-1417.575
	- 332		
Covariance Type:	opg		

	coef	std err	z	P> z	[0.025	0.975]
intercept	0.0041	0.002	2.626	0.009	0.001	0.007
ar.L1	0.7878	0.022	35.739	0.000	0.745	0.831
sigma2	0.0008	2.19e-05	36.118	0.000	0.001	0.001

Ljung-Box (L1) (Q): 0.28 Jarque-Bera (JB): 2885.82
Prob(Q): 0.60 Prob(JB): 0.00
Heteroskedasticity (H): 2.23 Skew: -0.43
Prob(H) (two-sided): 0.00 Kurtosis: 17.42



Como podemos ver en las graficas, tenemos ruido en el lag 12, yendo este hacia abajo, por o que cambiaremos el parametro Q de 0 a 1 para tratar de corregir este error

Modelo 2

```
[17] 1 Q = 1
```

```
1 modelo2 = sm.tsa.statespace.SARIMAX(df2['y'], order = (p, d, q), seasonal_order = (P, D, Q, num_per), trend = 'c', simple_differencing = True)
2 aux2 = modelo2.fit()
3 print(aux2.summary())
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:579: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:579: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
ignored when e.g. forecasting.', ValueWarning)

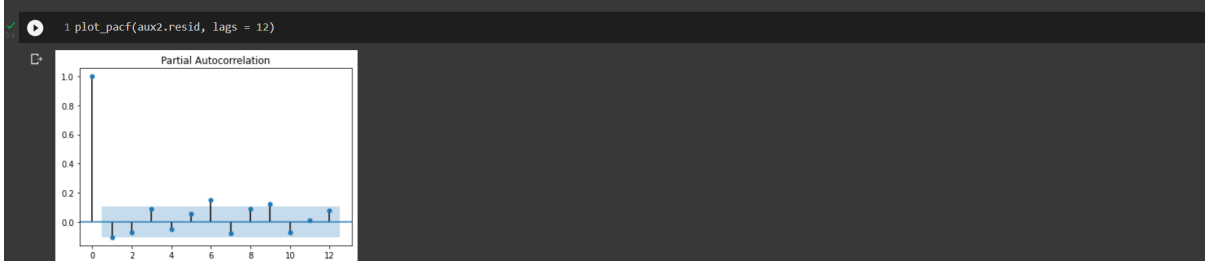
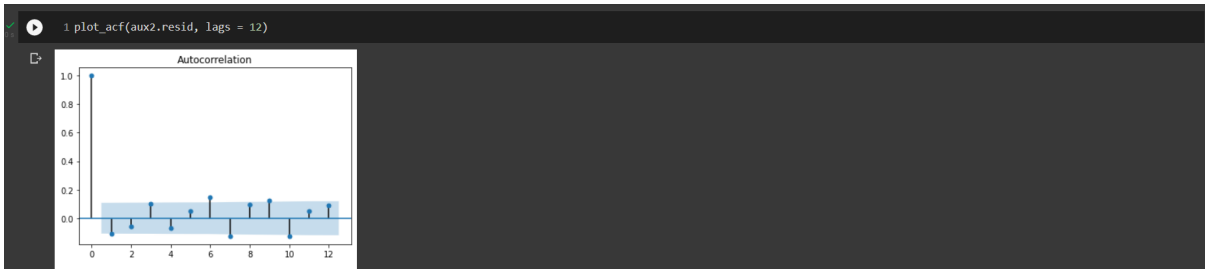
SARIMAX Results

Dep. Variable:	DS12.y	No. Observations:	332
Model:	SARIMAX(1, 0, 0)x(0, 0, [1], 12)	Log Likelihood	777.666
Date:	Sat, 26 Nov 2022	AIC	-1547.332
Time:	04:11:44	BIC	-1532.111
Sample:	0	HQIC	-1541.262
	- 332		
Covariance Type:	opg		

	coef	std err	z	P> z	[0.025	0.975]
intercept	0.0025	0.000	6.158	0.000	0.002	0.003
ar.L1	0.8785	0.019	45.539	0.000	0.841	0.916
ma.S112	-0.0465	0.049	-17.267	0.000	-0.943	-0.750
sigma2	0.0005	1.38e-05	37.428	0.000	0.000	0.001

Ljung-Box (L1) (Q): 4.09 Jarque-Bera (JB): 7595.76
Prob(Q): 0.04 Prob(JB): 0.00
Heteroskedasticity (H): 2.06 Skew: -2.33
Prob(H) (two-sided): 0.00 Kurtosis: 25.96

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).



Se puede mejorar la calibración ya que en el último gráficos de AC-PACS se ve ligeramente que la autocorrelación con lag 1 es negativa y sig.
Por lo que cambiamos de $q=0$ a $q=1$

Modelo 3

```
[21] 1 q = 1
```

```
1 modelo3 = sm.tsa.statespace.SARIMAX(df2['y'], order = (p, d, q), seasonal_order = (P, D, Q, num_per), trend = 'c', simple_differencing = True)
2 aux3 = modelo3.fit()
3 print(aux3.summary())
```

```
' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:579: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
' ignored when e.g. forecasting.', ValueWarning)
```

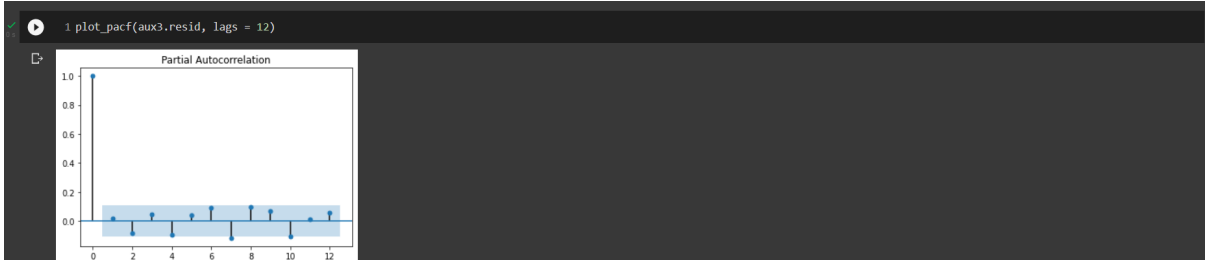
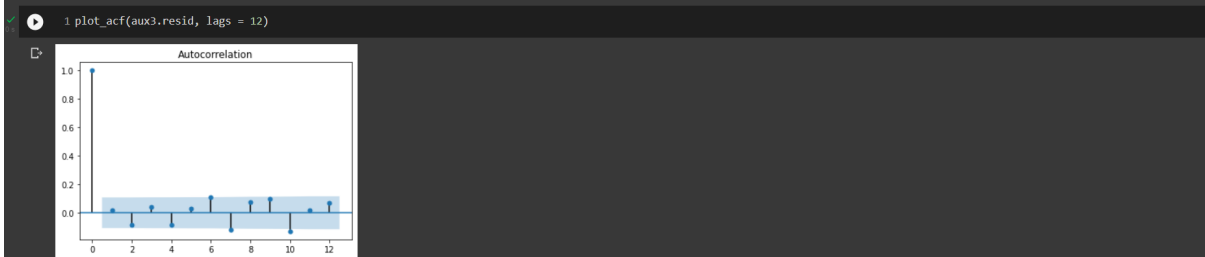
SARIMAX Results

```
=====
Dep. Variable:          DS12-y      No. Observations:          332
Model:             SARIMAX(1, 0, 1)x(0, 0, 1, 12)  Log Likelihood          781.071
Date:               Sat, 26 Nov 2022              AIC                  -1552.142
Time:               04:11:50                     BIC                  -1533.116
Sample:             - 0                      HQIC                  -1544.554
Covariance Type:    opg

=====
              coef    std err          z      P>|z|      [0.025    0.975]
-----
intercept    0.0016     0.000       3.363     0.001     0.001     0.003
ar.L1        0.9211     0.023    40.693     0.000     0.877     0.966
ma.L1       -0.1867     0.038    -4.951     0.000    -0.261    -0.113
ma.S.L12     -0.8465     0.048   -17.706     0.000    -0.940    -0.753
sigma2       0.0005    1.37e-05    37.011     0.000     0.000     0.001
=====
```

Ljung-Box (L1) (Q): 0.06 Jarque-Bera (JB): 9914.59
Prob(Q): 0.80 Prob(JB): 0.00
Heteroskedasticity (H): 2.31 Skew: -2.73
Prob(H) (two-sided): 0.00 Kurtosis: 29.21
=====

Warnings:



Como podemos ver en estos graficos, los lags de la variable ya se enciendan todas por debajo del 95%, tanto en la autocorrelacion parcial como en la normal, por lo que ya no generan ruido en nuestro modelo.

En cuanto al p-value de nuestras variables, todos tienen un valor muy bajo, por lo que las variables pueden ser consideradas como relevantes.

Gracias a esto, ya podemos generar un modelo, el cual seria:

$$y_t = 0.0016 + 0.9211 * y_{t-1} - 0.1867 * E_{t-1} - 0.8465 * E_{t-12} + 0.0005(\text{Desviacion estandar})$$

Descripción de las variables

Con esto, nos damos cuenta que el valor de 'y' anterior tiene una significancia del 92.11% para la 'y' actual, también, podemos ver que el error por temporada tiene una relación de 84.65% de forma negativa y el error anterior de 18.67%, también de forma negativa

+ Código + Texto

Predicción

Declaramos las variables de p, d, q, P, D, Q que se usaran para el modelo SARIMA

```
[25] 1 #Arima(1,1,0)
      2 p = 1
      3 q = 1
      4 d = 0
      5
      6 #SARIMA(0,1,1,12)
      7 P = 0
      8 Q = 1
      9 D = 1
     10 num_per = 12

[26] 1 modelo4 = sm.tsa.statespace.SARIMAX(df['Valor'], order = (p, d, q), seasonal_order = (P, D, Q, num_per), trend = 'c')
      2 aux4 = modelo4.fit()
```

Usando el metodo de predict del modelo indicamos el inicio y fin para la prediccion

```
1 y = aux4.predict(start=0, end=(len(df) + 24))

[28] 1 df = df[['Valor', 'Periodo']]
```

Insertamos filas para los siguientes 24 meses en los que realizaremos la prediccion

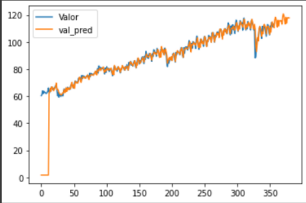
```
[29] 1 import math
      2 df = df.append(pd.Series([math.nan, '2022/09'], index=['Valor', 'Periodo']), ignore_index=True)
      3 df = df.append(pd.Series([math.nan, '2022/10'], index=['Valor', 'Periodo']), ignore_index=True)
      4 df = df.append(pd.Series([math.nan, '2022/11'], index=['Valor', 'Periodo']), ignore_index=True)
      5 df = df.append(pd.Series([math.nan, '2022/12'], index=['Valor', 'Periodo']), ignore_index=True)
      6 df = df.append(pd.Series([math.nan, '2023/01'], index=['Valor', 'Periodo']), ignore_index=True)
      7 df = df.append(pd.Series([math.nan, '2023/02'], index=['Valor', 'Periodo']), ignore_index=True)
      8 df = df.append(pd.Series([math.nan, '2023/03'], index=['Valor', 'Periodo']), ignore_index=True)
      9 df = df.append(pd.Series([math.nan, '2023/04'], index=['Valor', 'Periodo']), ignore_index=True)
     10 df = df.append(pd.Series([math.nan, '2023/05'], index=['Valor', 'Periodo']), ignore_index=True)
     11 df = df.append(pd.Series([math.nan, '2023/06'], index=['Valor', 'Periodo']), ignore_index=True)
     12 df = df.append(pd.Series([math.nan, '2023/07'], index=['Valor', 'Periodo']), ignore_index=True)
     13 df = df.append(pd.Series([math.nan, '2023/08'], index=['Valor', 'Periodo']), ignore_index=True)
     14 df = df.append(pd.Series([math.nan, '2023/09'], index=['Valor', 'Periodo']), ignore_index=True)
     15 df = df.append(pd.Series([math.nan, '2023/10'], index=['Valor', 'Periodo']), ignore_index=True)
     16 df = df.append(pd.Series([math.nan, '2023/11'], index=['Valor', 'Periodo']), ignore_index=True)
     17 df = df.append(pd.Series([math.nan, '2023/12'], index=['Valor', 'Periodo']), ignore_index=True)
     18 df = df.append(pd.Series([math.nan, '2024/01'], index=['Valor', 'Periodo']), ignore_index=True)
     19 df = df.append(pd.Series([math.nan, '2024/02'], index=['Valor', 'Periodo']), ignore_index=True)
     20 df = df.append(pd.Series([math.nan, '2024/03'], index=['Valor', 'Periodo']), ignore_index=True)
     21 df = df.append(pd.Series([math.nan, '2024/04'], index=['Valor', 'Periodo']), ignore_index=True)
     22 df = df.append(pd.Series([math.nan, '2024/05'], index=['Valor', 'Periodo']), ignore_index=True)
     23 df = df.append(pd.Series([math.nan, '2024/06'], index=['Valor', 'Periodo']), ignore_index=True)
     24 df = df.append(pd.Series([math.nan, '2024/07'], index=['Valor', 'Periodo']), ignore_index=True)
     25 df = df.append(pd.Series([math.nan, '2024/08'], index=['Valor', 'Periodo']), ignore_index=True)
```

```
[30] 1 df['val_pred'] = y
```

Comparamos las y prededidas con el modelo con las y reales

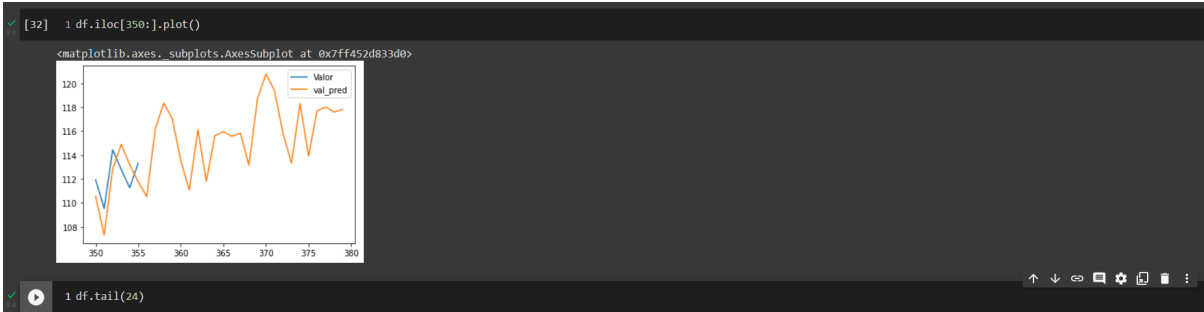
```
[31] 1 df.iloc[:].plot()

<matplotlib.axes._subplots.AxesSubplot at 0x7ff453124250>
```



Vemos la prediccion futura en una grafica

```
[32] 1 df.iloc[350:].plot()
```



	Valor	Periodo	val_pred
356	NaN	2022/09	110.526705
357	NaN	2022/10	116.144827
358	NaN	2022/11	118.350126
359	NaN	2022/12	117.040367
360	NaN	2023/01	113.530171
361	NaN	2023/02	111.082870
362	NaN	2023/03	116.123995
363	NaN	2023/04	111.806114
364	NaN	2023/05	115.622564
365	NaN	2023/06	115.957012
366	NaN	2023/07	115.580637
367	NaN	2023/08	115.828451
368	NaN	2023/09	113.148850
369	NaN	2023/10	118.669124
370	NaN	2023/11	120.787815
371	NaN	2023/12	119.401398
372	NaN	2024/01	115.823348

373	NaN	2024/02	113.315989
374	NaN	2024/03	118.303955
375	NaN	2024/04	113.939021
376	NaN	2024/05	117.713822
377	NaN	2024/06	118.011406
378	NaN	2024/07	117.602402
379	NaN	2024/08	117.821335

Podemos ver que las y_predecidas siguen la misma tendencia de las anteriores, y en este caso podemos ver que va aumentando