Autor: Martín de las Heras

Práctica 3

Ejercicio 1

Se descargan los datos diarios de la empresa del IBEX35 asignada según lista adjunta. El periodo es 2020-Mayo-4 (Lunes) a 2023-Noviembre-10 (Viernes) y las variables son precio mínimo, precio máximo, precio de apertura, precio de cierre y volumen negociado.

Nuestro objetivo es "aplicar modelos de regresión de ML-AS para predecir los valores de esas variables el lunes 13 de Noviembre, así como los 4 días siguientes. Consideramos tres opciones para los conjuntos de entrenamiento (TR) y de test (TS):

Opción	TR-inicio	TR-fin	TS-inicio	TS-fin
A	2020-Mayo-4	2023-Enero-31	2023-Febrero-1	2023-Noviembre-10
B	2020-Mayo-4	2023-Agosto-31	2023-Septiembre-1	2023-Noviembre-10
C	2020-Mayo-4	2022-Agosto-31	2022-Septiembre-1	2023-Noviembre-10

En primer lugar, importamos las librerías que vamos a utilizar:

```
In [ ]:
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.ar_model import AutoReg
import statsmodels.api as sm
import seaborn as sns
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
import numpy as np
from sklearn.metrics import r2_score
```

A continuación, leemos los datos del fichero csv de la empresa

```
In [ ]:
df_acs = pd.read_csv('Datos históricos ACS.csv', index_col='Fecha', date_format=
df_acs['Vol.'] = df_acs['Vol.'].str.replace(',', '.')
df_acs['Vol.'] = (df_acs['Vol.'].replace(r'[KM]+$', '', regex=True).astype(float
    df_acs['Vol.'].str.extract(r'[\d\.]+([KM]+)', expand=False)
    .fillna(1)
    .replace(['K','M'], [10**3, 10**6]).astype(int))
df_acs['% var.'] = df_acs['% var.'].str.replace(',','.')
df_acs['% var.'] = df_acs['% var.'].str.replace('%','')
df_acs['% var.'] = df_acs['% var.'].astype(float)/100.
df_acs.sort_values(by='Fecha', inplace=True)
df_acs = df_acs.asfreq(freq='B', method='ffill')
df_acs
```

Out[ ]:

	Último	Apertura	Máximo	Mínimo	Vol.	% var.
Fecha						
2020-05-04	22.33	22.32	22.67	21.92	1960000.0	-0.0193
2020-05-05	23.01	22.80	23.12	22.25	2040000.0	0.0305
2020-05-06	23.33	22.91	23.70	22.61	2470000.0	0.0139
2020-05-07	23.42	23.40	23.66	23.38	1250000.0	0.0039
2020-05-08	23.74	23.70	23.95	23.54	1230000.0	0.0137
...	...	...	...	...	...	...
2023-11-06	33.50	33.89	33.89	33.39	270000.0	-0.0115
2023-11-07	33.18	33.30	33.47	33.10	254900.0	-0.0096
2023-11-08	33.26	33.09	33.30	32.93	663930.0	0.0024
2023-11-09	33.32	33.25	33.44	33.10	273070.0	0.0018
2023-11-10	33.26	33.22	33.47	33.16	231270.0	-0.0018

920 rows × 6 columns

a) Calcula el porcentaje de datos utilizados en cada opción para TR y TS.

En primer lugar creamos los datasets de cada opción, tanto las **X** como las **y** de TR y TS:

In [ ]:
```
train_a = df_acs[df_acs.index <= '31.01.2023']
test_a = df_acs[df_acs.index > '31.01.2023']
train_b = df_acs[df_acs.index <= '31.08.2023']
test_b = df_acs[df_acs.index > '31.08.2023']
train_c = df_acs[df_acs.index <= '31.08.2022']
test_c = df_acs[df_acs.index > '31.08.2022']

X_train_a = train_a.drop(['% var.'], axis=1)
X_test_a = test_a.drop(['% var.'], axis=1)
X_train_b = train_b.drop(['% var.'], axis=1)
X_test_b = test_b.drop(['% var.'], axis=1)
X_train_c = train_c.drop(['% var.'], axis=1)
X_test_c = test_c.drop(['% var.'], axis=1)

y_train_a = train_a['% var.']
y_test_a = test_a['% var.']
y_train_b = train_b['% var.']
y_test_b = test_b['% var.']
y_train_c = train_c['% var.']
y_test_c = test_c['% var.']
```

A continuación, sacamos los porcentajes de cada una de las opciones para train y test.

In [ ]:
```
train_a_pct = X_train_a.shape[0]/df_acs.shape[0]
test_a_pct = X_test_a.shape[0]/df_acs.shape[0]
train_b_pct = X_train_b.shape[0]/df_acs.shape[0]
test_b_pct = X_test_b.shape[0]/df_acs.shape[0]
train_c_pct = X_train_c.shape[0]/df_acs.shape[0]
```

```
test_c_pct = X_test_c.shape[0]/df_acs.shape[0]

print(train_a_pct)
print(test_a_pct)
print(train_b_pct)
print(test_b_pct)
print(train_c_pct)
print(test_c_pct)
```

```
0.7793478260869565
0.22065217391304348
0.9445652173913044
0.05543478260869565
0.6608695652173913
0.3391304347826087
```

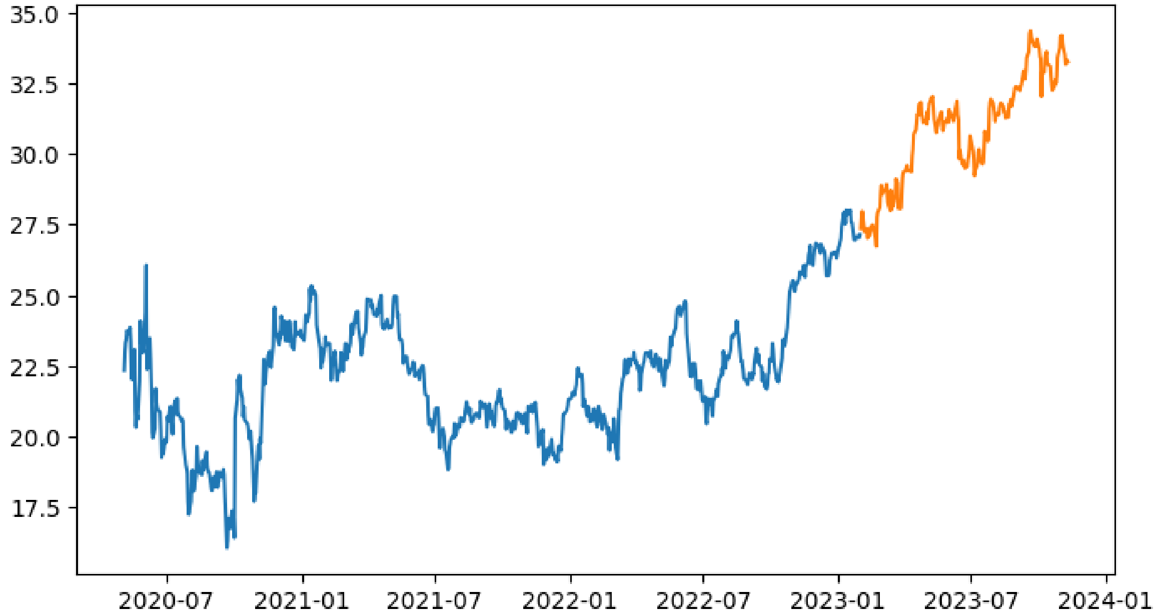
De esta manera nos quedan estos porcentajes para cada opción

Opción	TR	TS
A	0.78	0.22
B	0.94	0.06
C	0.66	0.34

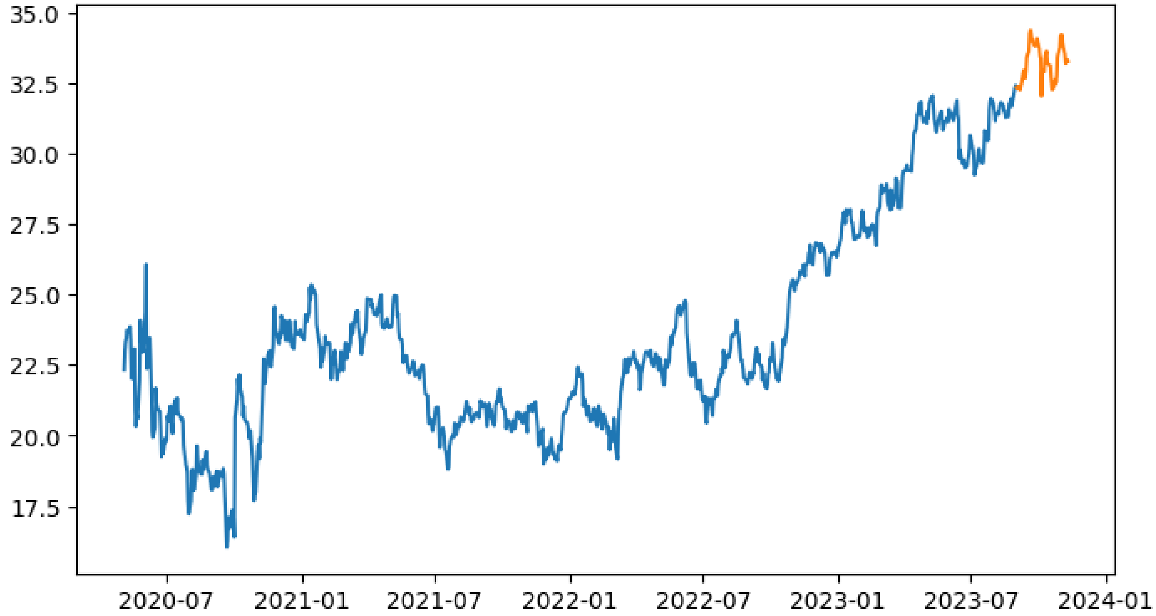
b) Muestra gráficamente las tres opciones en un gráfico de serie temporal para el precio de cierre.

In [ ]:
```
plt.figure(figsize=(8,15))
plt.subplot(3,1,1)
plt.plot(X_train_a['Último'])
plt.plot(X_test_a['Último'])
plt.title('Opción A')
plt.subplot(3,1,2)
plt.plot(X_train_b['Último'])
plt.plot(X_test_b['Último'])
plt.title('Opción B')
plt.subplot(3,1,3)
plt.plot(X_train_c['Último'])
plt.plot(X_test_c['Último'])
plt.title('Opción C');
```

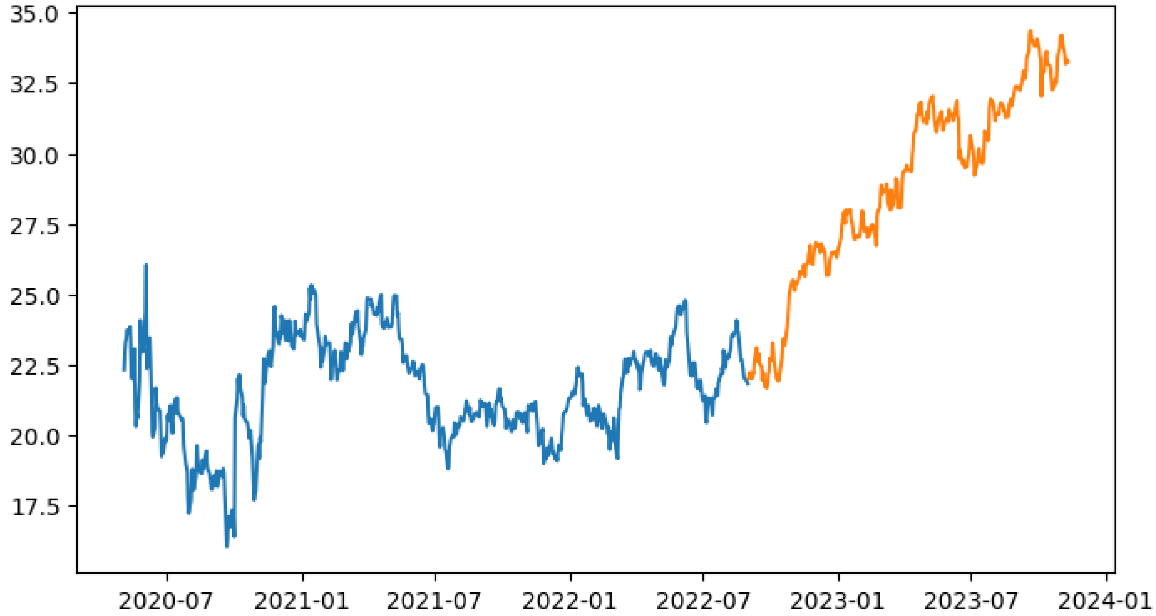
Opción A



Opción B



Opción C



Aquí podemos ver que la opción **B** es la que tiene el menor tamaño del conjunto de test, así como la opción **C** es la que mayor tamaño tiene.

c) ¿En qué opción los periodos TR y TS son más diferentes? ¿Y más semejantes? En ambos casos respecto al precio de cierre. Emplea medidas estadísticas y gráficos como el gráfico de cajas.

Comparaos la media y varianza de los valores de cierre de los datos de train contra los de test:

In [ ]:
```
print('Opción A')
print('Media TR A: %s\nVarianza TR A: %s' % (X_train_a['Último'].mean(), X_train
print('Media TS A: %s\nVarianza TS A: %s' % (X_test_a['Último'].mean(), X_test_a
print('Opción B')
print('Media TR B: %s\nVarianza TR B: %s' % (X_train_b['Último'].mean(), X_train
print('Media TS B: %s\nVarianza TS B: %s' % (X_test_b['Último'].mean(), X_test_b
print('Opción C')
print('Media TR C: %s\nVarianza TR C: %s' % (X_train_c['Último'].mean(), X_train
print('Media TS C: %s\nVarianza TS C: %s' % (X_test_c['Último'].mean(), X_test_c
```

```
Opción A
Media TR A: 22.15100139470014
Varianza TR A: 2.2288990289564414
Media TS A: 30.887911330049267
Varianza TS A: 1.9158360484070511
Opción B
Media TR B: 23.541477560414272
Varianza TR B: 3.6914550174825735
Media TS B: 33.23470588235294
Varianza TS B: 0.6436034584778939
Opción C
Media TR C: 21.627769736842104
Varianza TR C: 1.8220367813408005
Media TS C: 28.855224358974358
Varianza TS C: 3.397229928077857
```

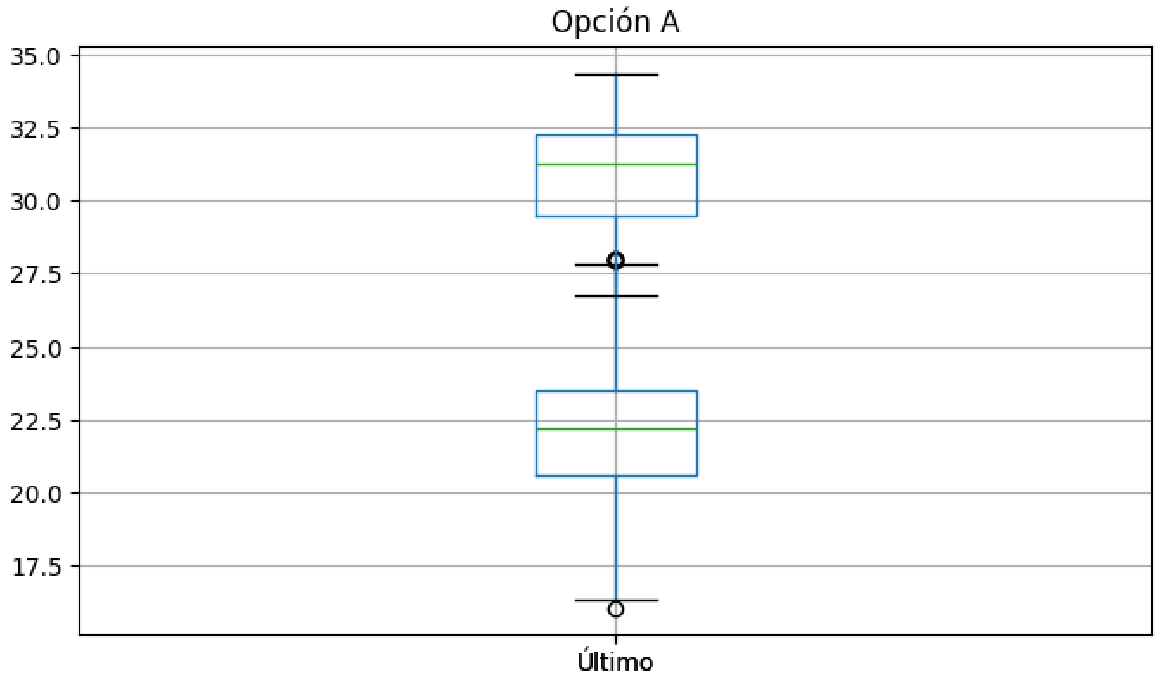
Aquí podemos ver que la media en las opciones **A** y **B** es muy distinta entre TR y TS. En la opción **A** tenemos que las varianzas son relativamente parecidas, al contrario que en la opción **B**.

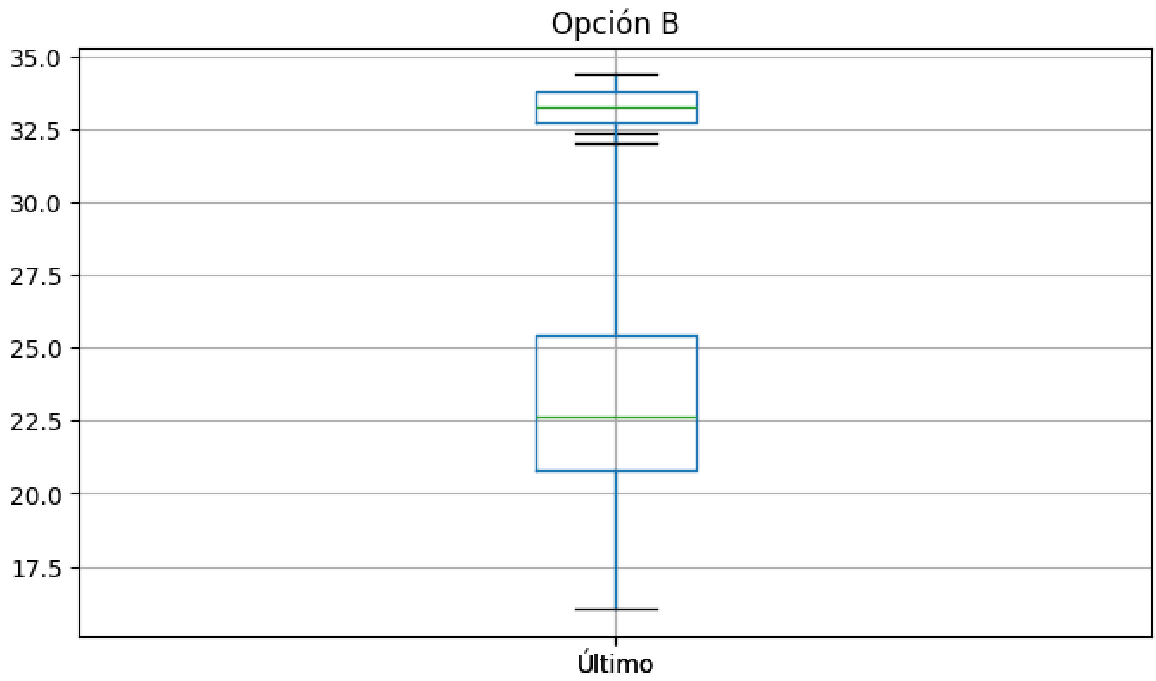
La opción que parece que tiene parámetros más estables es la **C**.

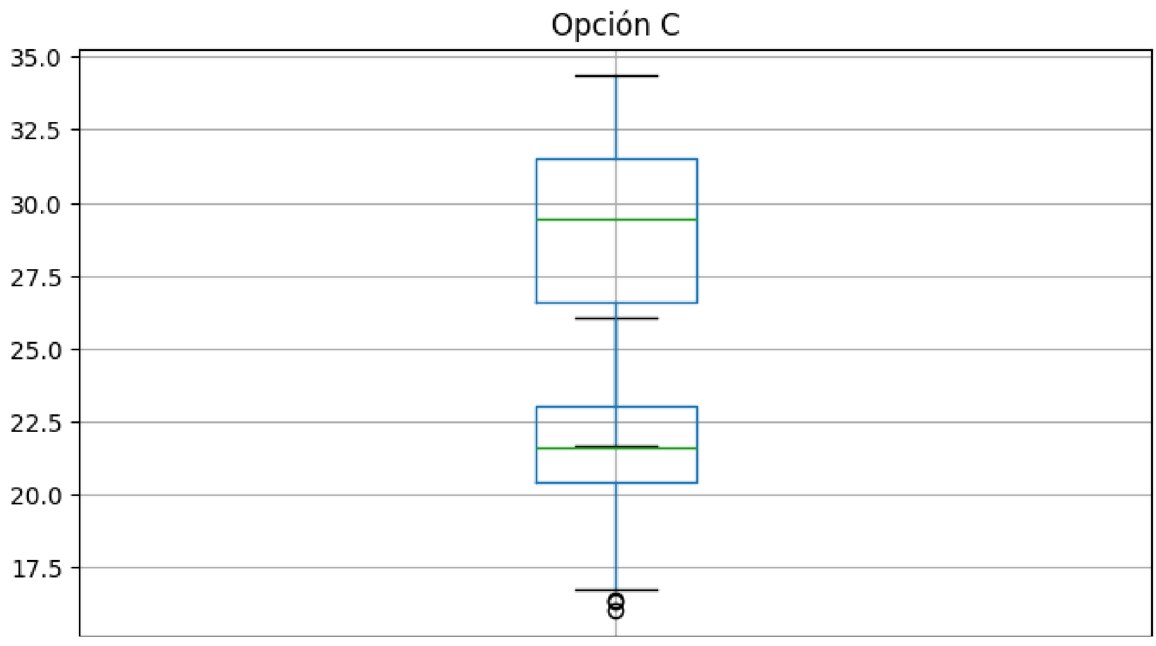
Continuamos con el gráfico de cajas:

In [ ]:
```
plt.figure(figsize=(8,15))
plt.subplot(3,1,1)
X_train_a.boxplot(column=['Último'])
X_test_a.boxplot(column=['Último'])
plt.title('Opción A')
plt.subplot(3,1,2)
X_train_b.boxplot(column=['Último'])
X_test_b.boxplot(column=['Último'])
plt.title('Opción B')
plt.subplot(3,1,3)
X_train_c.boxplot(column=['Último'])
```

```
X_test_c.boxplot(column=['Último'])
plt.title('Opción C');
```


Opción A
35.0
32.5
30.0
27.5
25.0
22.5
20.0
17.5
Último


Opción B
35.0
32.5
30.0
27.5
25.0
22.5
20.0
17.5
Último


Opción C
35.0
32.5
30.0
27.5
25.0
22.5
20.0
17.5

Último

Aquí podemos ver en los diagramas de cajas las comparaciones entre las opciones **A**, **B** y **C**, donde podemos verificar lo dicho con los valores de la media y la varianza.

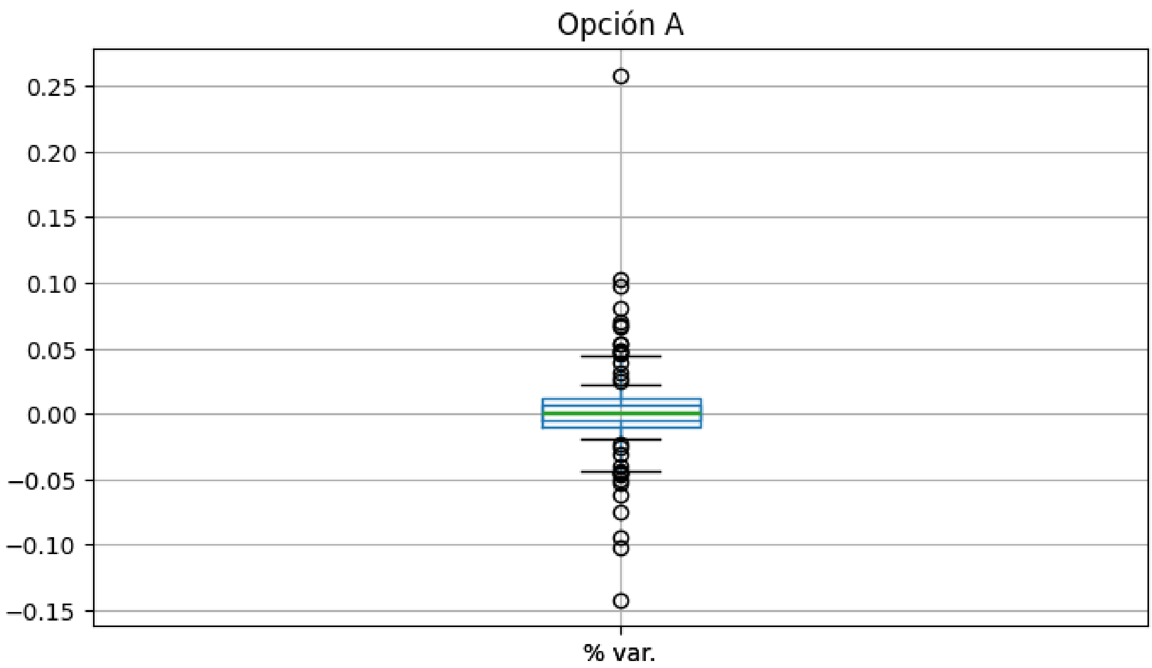
d) En qué opción los periodos TR y TS son más diferentes? ¿Y más semejantes? En ambos casos respecto a la variación diaria del precio de cierre. Emplea medidas estadísticas y gráficos como el gráfico de cajas.

Comparamos la media y la varianza de la columna *% var.* y graficamos los resultados en un diagrama de cajas.

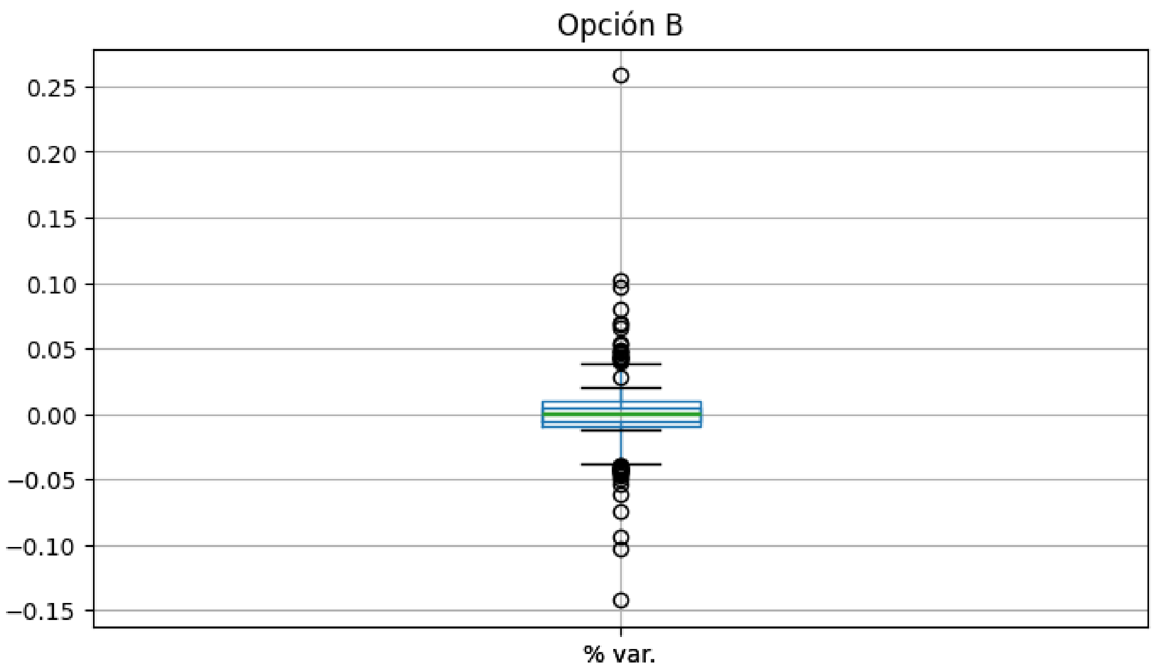
In [ ]:
```
print('Opción A')
print('Media TR A: %s\nVarianza TR A: %s' % (y_train_a.mean(), y_train_a.std()))
print('Media TS A: %s\nVarianza TS A: %s' % (y_test_a.mean(), y_test_a.std()))
print('Opción B')
print('Media TR B: %s\nVarianza TR B: %s' % (y_train_b.mean(), y_train_b.std()))
print('Media TS B: %s\nVarianza TS B: %s' % (y_test_b.mean(), y_test_b.std()))
print('Opción C')
print('Media TR C: %s\nVarianza TR C: %s' % (y_train_c.mean(), y_train_c.std()))
print('Media TS C: %s\nVarianza TS C: %s' % (y_test_c.mean(), y_test_c.std()))
```

```
Opción A
Media TR A: 0.000520781032078103
Varianza TR A: 0.022581729289696997
Media TS A: 0.0010167487684729062
Varianza TS A: 0.010863938679622142
Opción B
Media TR B: 0.0006336018411967776
Varianza TR B: 0.0210193738486367
Media TS B: 0.0005725490196078428
Varianza TS B: 0.010486468965888805
Opción C
Media TR C: 0.00023404605263157888
Varianza TR C: 0.02405586870573025
Media TS C: 0.0014022435897435895
Varianza TS C: 0.010973518024954906
```

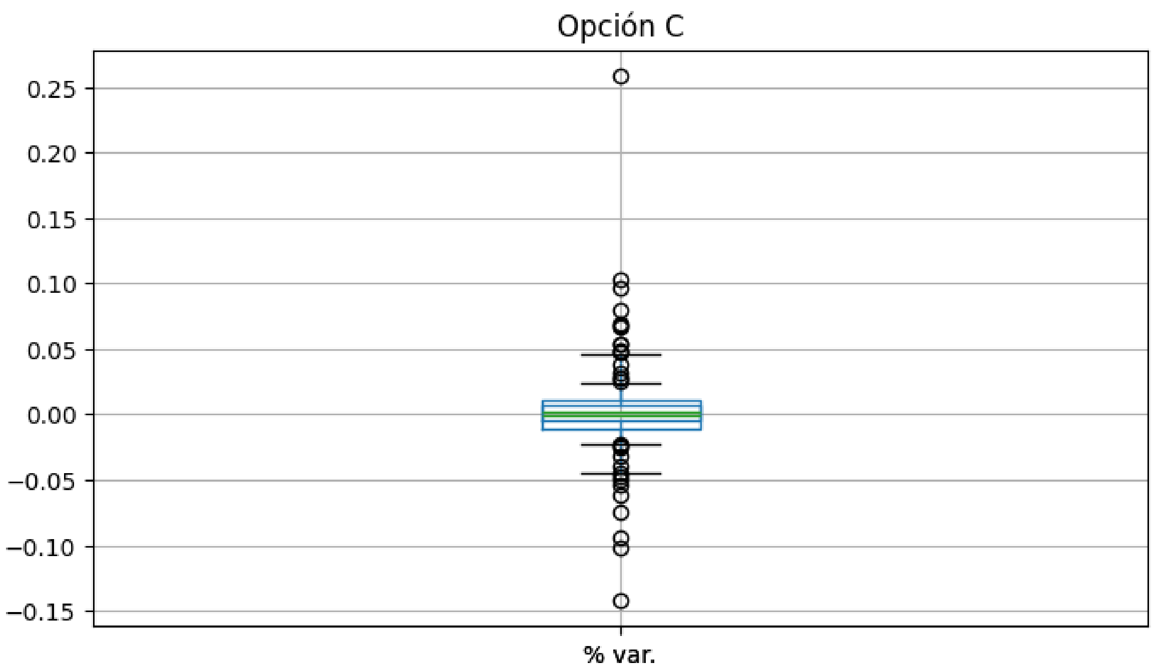
In [ ]:
```
plt.figure(figsize=(8,15))
plt.subplot(3,1,1)
pd.DataFrame(y_train_a).boxplot(column='% var.')
pd.DataFrame(y_test_a).boxplot(column='% var.')
plt.title('Opción A')
plt.subplot(3,1,2)
pd.DataFrame(y_train_b).boxplot(column='% var.')
pd.DataFrame(y_test_b).boxplot(column='% var.')
plt.title('Opción B')
plt.subplot(3,1,3)
pd.DataFrame(y_train_c).boxplot(column='% var.')
pd.DataFrame(y_test_c).boxplot(column='% var.')
plt.title('Opción C');
```



Opción A
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
% var.



Opción B
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
% var.



Opción C
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
% var.

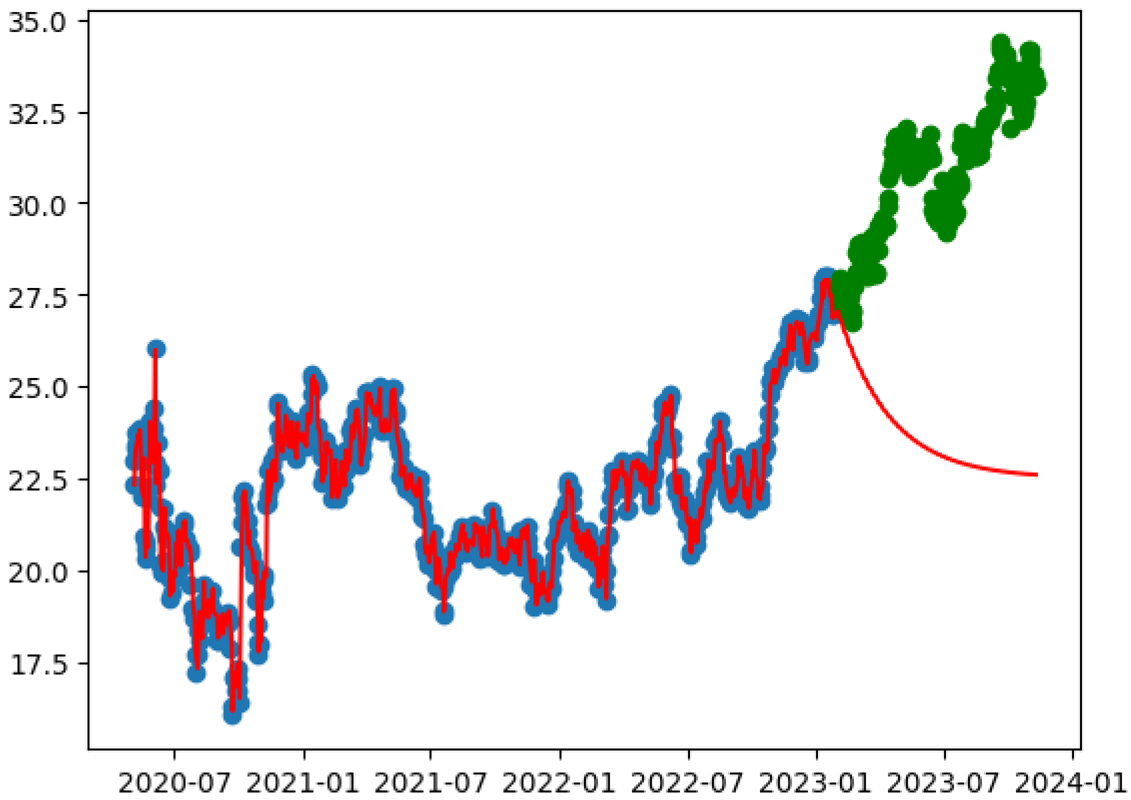
En este caso conviene mirarlo mejor por los valores que por la gráfica ya que por temas de la escala no se aprecian bien las diferencias entre los valores.

Vemos que en el caso de la opción **A**, se duplican tanto la media como la varianza, en la opción **B** las medias son bastante parejas, pero la varianza otra vez se duplica. Por último, la opción **C**, la media se dispara de TR a TS y la varianza varía también en un factor de 2.

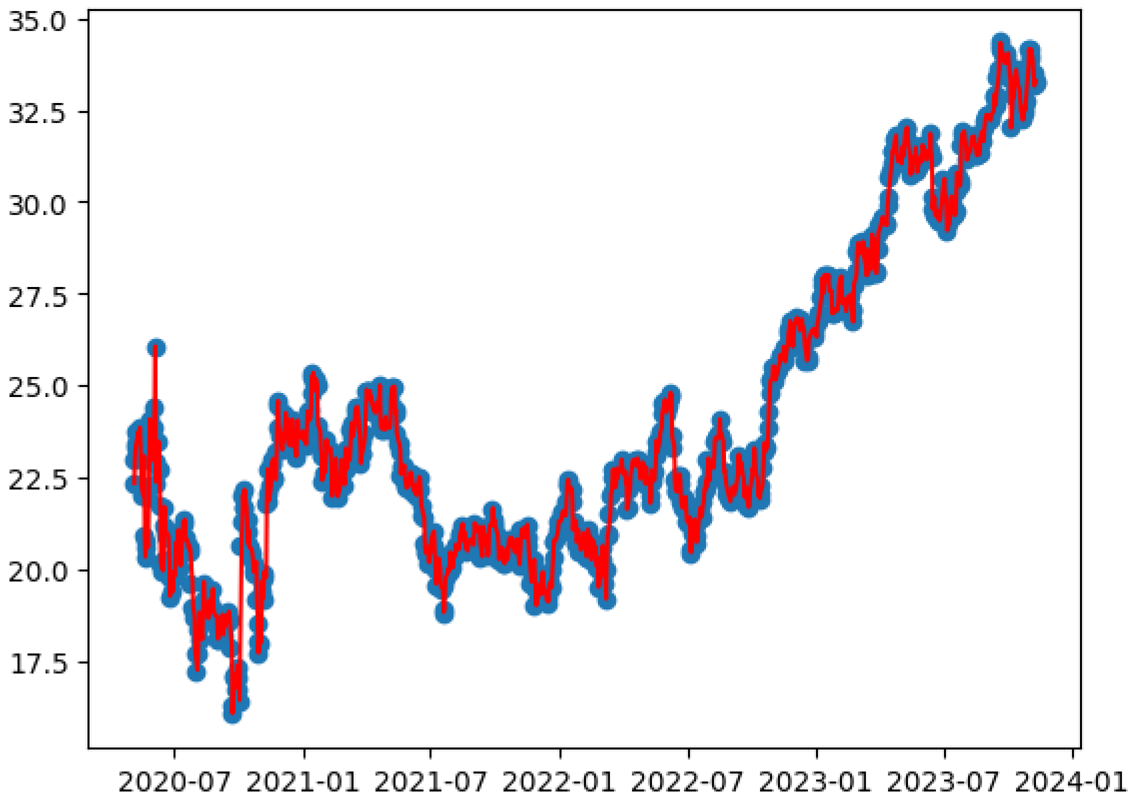
La relación del precio (variación) en un día con el precio (variación) en el día anterior se podría aprender y supervisar o modelar por la regresión lineal, a través del llamado modelo autorregresivo de orden 1 y notado por AR(1).

e) Dibuja el diagrama de dispersión con la recta de regresión ajustada para los precios de cierre con el modelo AR(1) en la opción A. Idem tomando todo el conjunto de datos como TR. Extraer conclusiones.

In [ ]:
```
ar_model_train = AutoReg(X_train_a['Último'], lags=1).fit()
pred_train = ar_model_train.predict(start=df_acs.index.min(), end=df_acs.index.m

plt.plot(pred_train, color='red')
plt.scatter(X_train_a.index, X_train_a['Último'])
plt.scatter(X_test_a.index, X_test_a['Último'], color='green');
```



In [ ]:
```
ar_model_whole = AutoReg(df_acs['Último'], lags=1).fit()
pred_whole = ar_model_whole.predict(start=df_acs.index.min(), end=df_acs.index.m

plt.plot(pred_whole, color='red')
plt.scatter(df_acs.index, df_acs['Último']);
```



Al haber una subida inesperadamente larga, el modelo que está solo entrenado con los datos de entrenamiento se espera una bajada, por lo que la predicción es mala. Sin embargo, el modelo entrenado con todos los datos encaja bien todos los datos de entrenamiento.

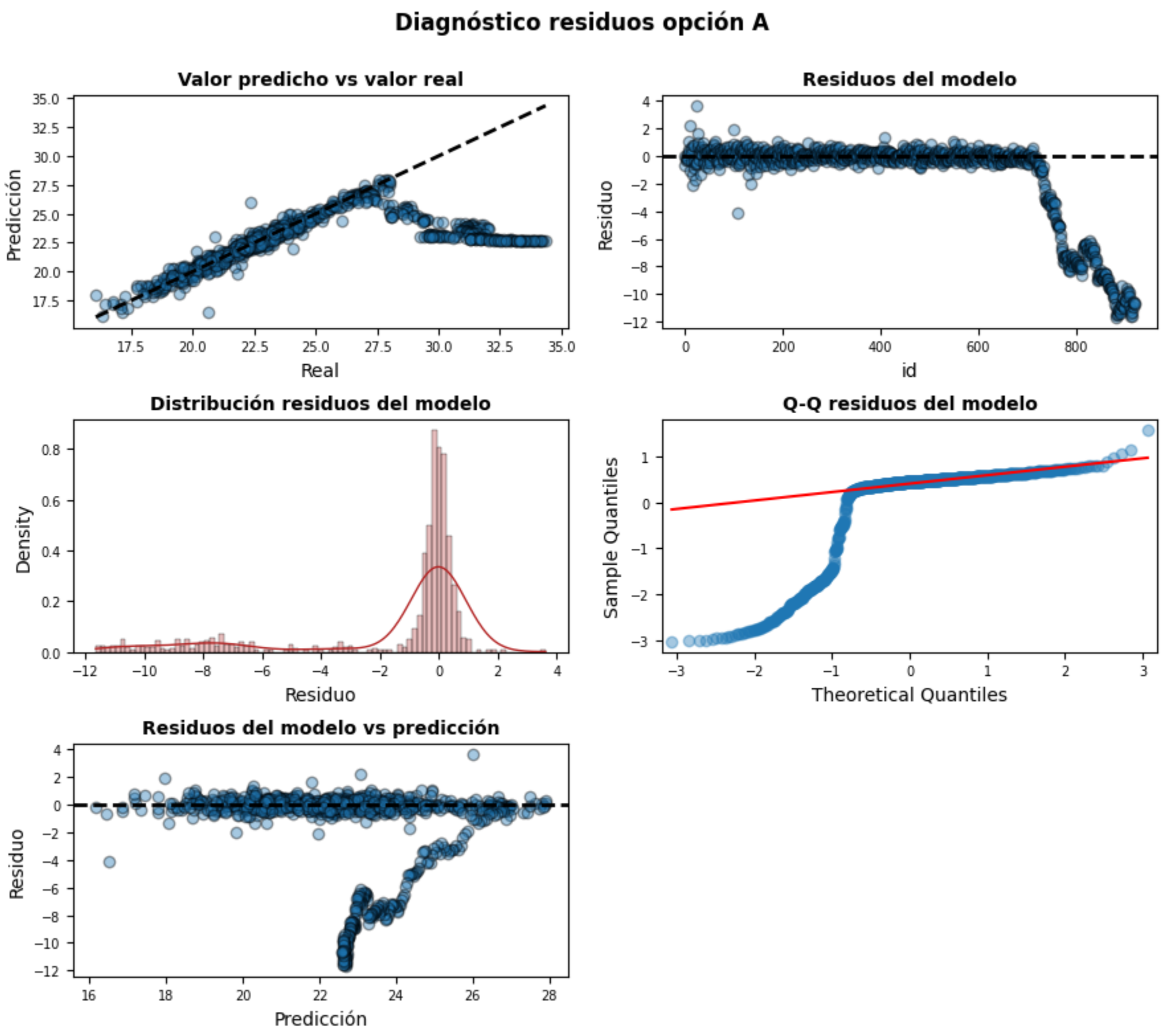
f) Obtener los residuos y proceder a un análisis gráfico de los mismos en las dos hipótesis de **e)**. Extraer conclusiones sobre el modelo AS-RLS basado en normalidad, aleatoriedad y homocedasticidad de los errores.

In [ ]:

```
# Diagnóstico errores (residuos) de las predicciones de entrenamiento
# ==============================================================================
pred_train.iloc[0] = pred_train.iloc[1]
residuos_train    = pred_train - df_acs['Último']

# Gráficos
# ==============================================================================
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(9, 8))

axes[0, 0].scatter(df_acs['Último'], pred_train, edgecolors=(0, 0, 0), alpha = 0
axes[0, 0].plot([min(df_acs['Último']), max(df_acs['Último'])], [min(df_acs['Últ
                'k--', lw=2)
axes[0, 0].set_title('Valor predicho vs valor real', fontsize = 10, fontweight =
axes[0, 0].set_xlabel('Real')
axes[0, 0].set_ylabel('Predicción')
axes[0, 0].tick_params(labelsize = 7)

axes[0, 1].scatter(list(range(len(df_acs['Último']))), residuos_train,
                   edgecolors=(0, 0, 0), alpha = 0.4)
axes[0, 1].axhline(y = 0, linestyle = '--', color = 'black', lw=2)
axes[0, 1].set_title('Residuos del modelo', fontsize = 10, fontweight = "bold")
axes[0, 1].set_xlabel('id')
axes[0, 1].set_ylabel('Residuo')
```

```
axes[0, 1].tick_params(labelsize = 7)

sns.histplot(
    data     = residuos_train,
    stat     = "density",
    kde      = True,
    line_kws= {'linewidth': 1},
    color    = "firebrick",
    alpha    = 0.3,
    ax       = axes[1, 0]
)

axes[1, 0].set_title('Distribución residuos del modelo', fontsize = 10,
                     fontweight = "bold")
axes[1, 0].set_xlabel("Residuo")
axes[1, 0].tick_params(labelsize = 7)


sm.qqplot(
    residuos_train,
    fit   = True,
    line  = 'q',
    ax    = axes[1, 1],
    color = 'firebrick',
    alpha = 0.4,
    lw    = 2
)
axes[1, 1].set_title('Q-Q residuos del modelo', fontsize = 10, fontweight = "bol
axes[1, 1].tick_params(labelsize = 7)

axes[2, 0].scatter(pred_train, residuos_train,
                   edgecolors=(0, 0, 0), alpha = 0.4)
axes[2, 0].axhline(y = 0, linestyle = '--', color = 'black', lw=2)
axes[2, 0].set_title('Residuos del modelo vs predicción', fontsize = 10, fontwei
axes[2, 0].set_xlabel('Predicción')
axes[2, 0].set_ylabel('Residuo')
axes[2, 0].tick_params(labelsize = 7)

# Se eliminan los axes vacíos
fig.delaxes(axes[2,1])

fig.tight_layout()
plt.subplots_adjust(top=0.9)
fig.suptitle('Diagnóstico residuos opción A', fontsize = 12, fontweight = "bold"
```

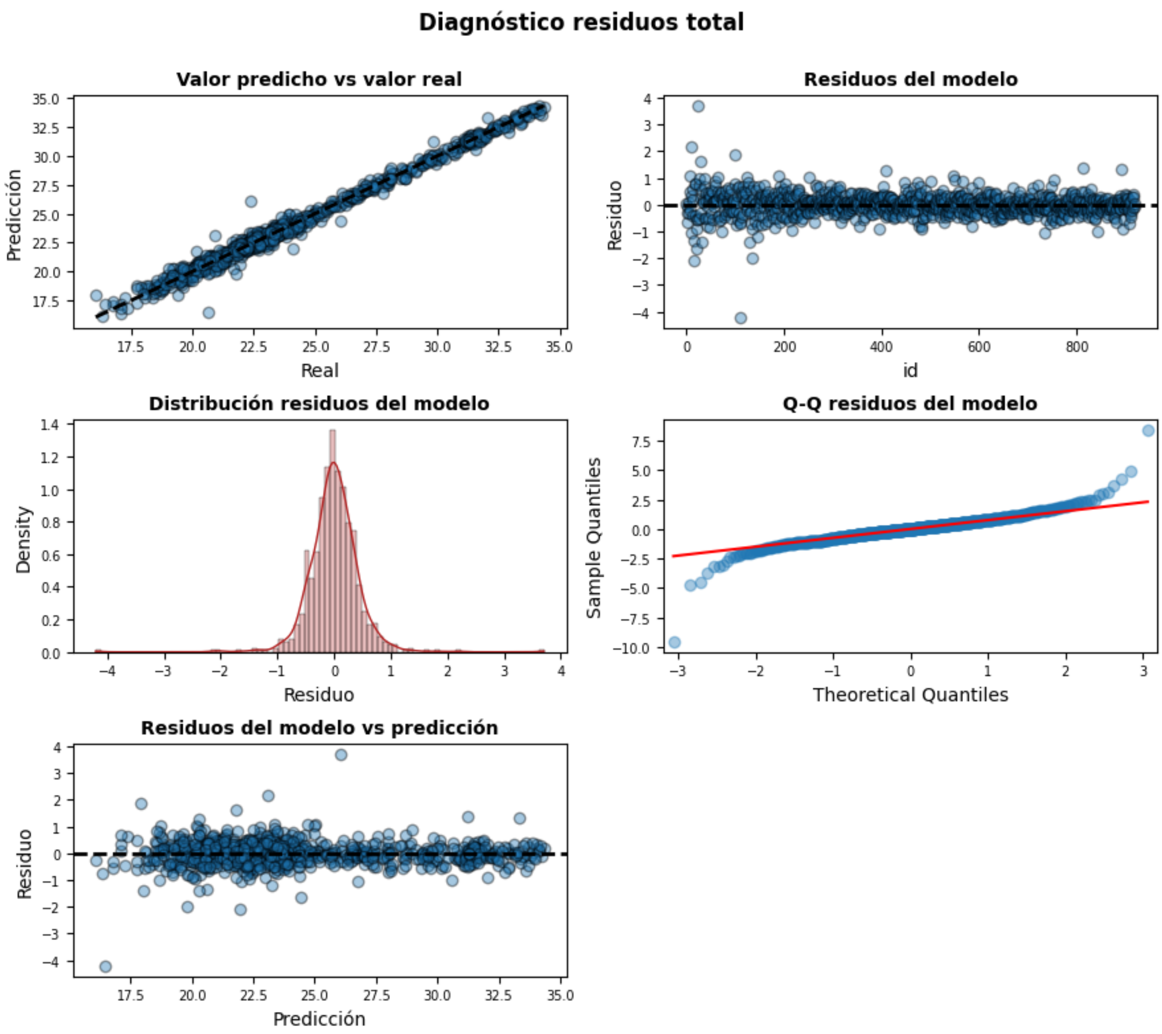
```
C:\Users\Marti\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2
kfra8p0\LocalCache\local-packages\Python311\site-packages\statsmodels\graphics\go
fplots.py:1045: UserWarning: color is redundantly defined by the 'color' keyword
argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword argume
nt will take precedence.
  ax.plot(x, y, fmt, **plot_style)
```




```
In [ ]: # Diagnóstico errores (residuos) de las predicciones de entrenamiento
        # ==============================================================================
        pred_whole.iloc[0] = pred_whole.iloc[1]
        residuos_train    = pred_whole - df_acs['Último']

        # Gráficos
        # ==============================================================================
        fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(9, 8))

        axes[0, 0].scatter(df_acs['Último'], pred_whole, edgecolors=(0, 0, 0), alpha = 0
        axes[0, 0].plot([min(df_acs['Último']), max(df_acs['Último'])], [min(df_acs['Últ
                        'k--', lw=2)
        axes[0, 0].set_title('Valor predicho vs valor real', fontsize = 10, fontweight =
        axes[0, 0].set_xlabel('Real')
        axes[0, 0].set_ylabel('Predicción')
        axes[0, 0].tick_params(labelsize = 7)

        axes[0, 1].scatter(list(range(len(df_acs['Último']))), residuos_train,
                           edgecolors=(0, 0, 0), alpha = 0.4)
        axes[0, 1].axhline(y = 0, linestyle = '--', color = 'black', lw=2)
        axes[0, 1].set_title('Residuos del modelo', fontsize = 10, fontweight = "bold")
        axes[0, 1].set_xlabel('id')
        axes[0, 1].set_ylabel('Residuo')
        axes[0, 1].tick_params(labelsize = 7)

        sns.histplot(
            data     = residuos_train,
            stat     = "density",
```

```python
    kde      = True,
    line_kws= {'linewidth': 1},
    color    = "firebrick",
    alpha    = 0.3,
    ax       = axes[1, 0]
)

axes[1, 0].set_title('Distribución residuos del modelo', fontsize = 10,
                     fontweight = "bold")
axes[1, 0].set_xlabel("Residuo")
axes[1, 0].tick_params(labelsize = 7)


sm.qqplot(
    residuos_train,
    fit   = True,
    line  = 'q',
    ax    = axes[1, 1],
    color = 'firebrick',
    alpha = 0.4,
    lw    = 2
)
axes[1, 1].set_title('Q-Q residuos del modelo', fontsize = 10, fontweight = "bol
axes[1, 1].tick_params(labelsize = 7)

axes[2, 0].scatter(pred_whole, residuos_train,
                   edgecolors=(0, 0, 0), alpha = 0.4)
axes[2, 0].axhline(y = 0, linestyle = '--', color = 'black', lw=2)
axes[2, 0].set_title('Residuos del modelo vs predicción', fontsize = 10, fontwei
axes[2, 0].set_xlabel('Predicción')
axes[2, 0].set_ylabel('Residuo')
axes[2, 0].tick_params(labelsize = 7)

# Se eliminan los axes vacíos
fig.delaxes(axes[2,1])

fig.tight_layout()
plt.subplots_adjust(top=0.9)
fig.suptitle('Diagnóstico residuos total', fontsize = 12, fontweight = "bold");
```

```
C:\Users\Marti\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2
kfra8p0\LocalCache\local-packages\Python311\site-packages\statsmodels\graphics\go
fplots.py:1045: UserWarning: color is redundantly defined by the 'color' keyword
argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword argume
nt will take precedence.
  ax.plot(x, y, fmt, **plot_style)
```



En estos casos se puede observar lo que hemos comentado en el apartado anterior:

- Para la opción **A**, se ve que los errores aumentan repentinamente cuando pasa de TR a TS, esto se debe a que no ha visto previamente los datos, por lo que estima de una manera razonable dentro de lo que cabe en base a los datos previos que ha tenido. Se puede ver que los residuos tienen una distribución generalmente normal y heterocedástica.
- Para la opción de entrenamiento con todo el conjunto, nos sale una distribución normal de los residuos, homocedástica y regular en todo el conjunto.

g) Comparar el MSE y R cuadrado en TR y TS en la opción **A**

In [ ]:
```
mse_train = mean_squared_error(X_train_a['Último'], pred_train.loc[:'31.01.2023'
mse_test = mean_squared_error(X_test_a['Último'], pred_train.loc['02.01.2023':])

r2_train = X_train_a['Último'].corr(pred_train.loc[:'31.01.2023'])**2
r2_test = X_test_a['Último'].corr(pred_train.loc['02.01.2023':])**2

print('MSE TR: %f' % mse_train)
print('MSE TS: %f' % mse_test)

print('R2 TR: %f' % r2_train)
print('R2 TS: %f' % r2_test)
```

```
MSE TR: 0.217709
MSE TS: 60.905249
R2 TR: 0.956116
R2 TS: 0.723127
```

Claramente se puede ver que el error en TS es mucho más grande que en TR, y por otro lado que los valores de R2 también son mucho mejores para TR que para TS. Esto se corresponde con lo que hemos visto en el análisis de los errores y en las gráficas del AR.

h) ¿Qué información se puede obtener sobre la predicción del precio de cierre del lunes 13 de Noviembre a un 95 %?

Para ello tomamos el modelo que ha sido entrenado con todos los datos, ya que nos dará una mejor predicción.

In [ ]:
```
ar_model_whole.predict(start='13.11.2023', end='13.11.2023')
prediction = ar_model_whole.get_prediction(start='13.11.2023', end='13.11.2023')
prediction.predicted_mean
```

Out[ ]:
```
2023-11-13    33.244211
Freq: B, Name: predicted_mean, dtype: float64
```

Nos realiza la predicción de que el precio de cierre el lunes 13 de Noviembre será de 33.244211. Comprobamos ahora el intervalo de confianza que nos da para la predicción:

In [ ]:
```
prediction.conf_int(alpha=0.05)
```

Out[ ]:

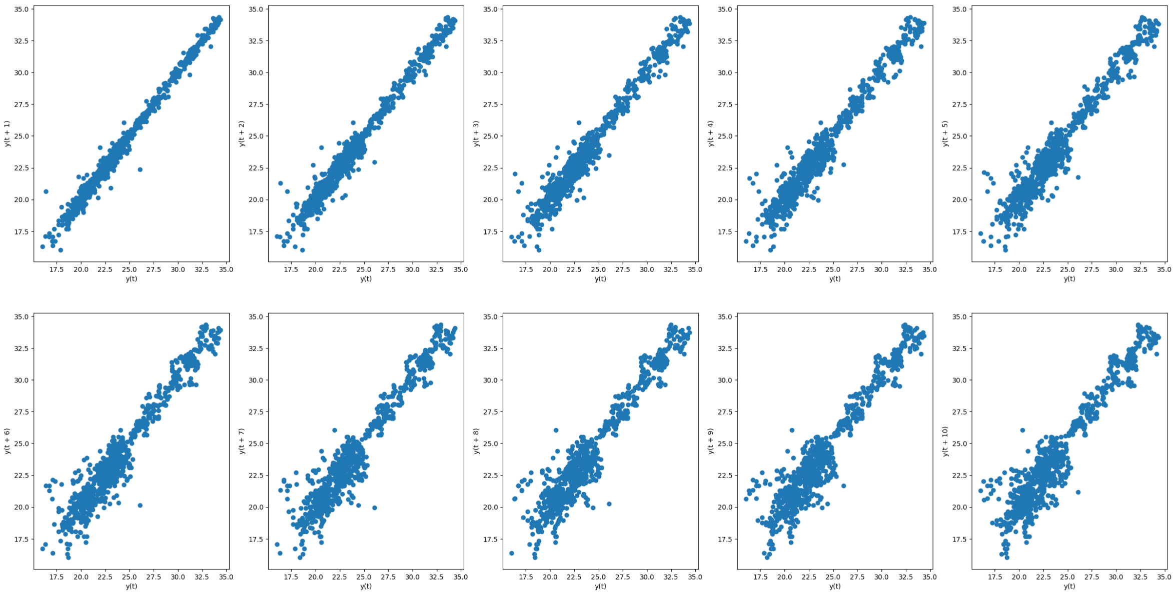
	lower	upper
2023-11-13	32.378472	34.109951

Ejercicio 2

A veces el precio de cierre (variación) diario se relaciona con el precio de cierre (variación) de varios días atrás. Si es con 2 días sería el modelo AR(2), si es con p días atrás sería el modelo AR(p).

a) Dibuja el gráfico matricial del precio de cierre de un día con el día anterior, dos días atrás,....y hasta 10 días atrás. Calcula la matriz de correlaciones. Extrae conclusiones sobre el “orden” del modelo AR.

In [ ]:
```
from pandas.plotting import lag_plot
plt.figure(figsize=(30,15))
for i in range(10):
    plt.subplot(2,5,i+1)
    lag_plot(df_acs['Último'], lag=i+1)
```

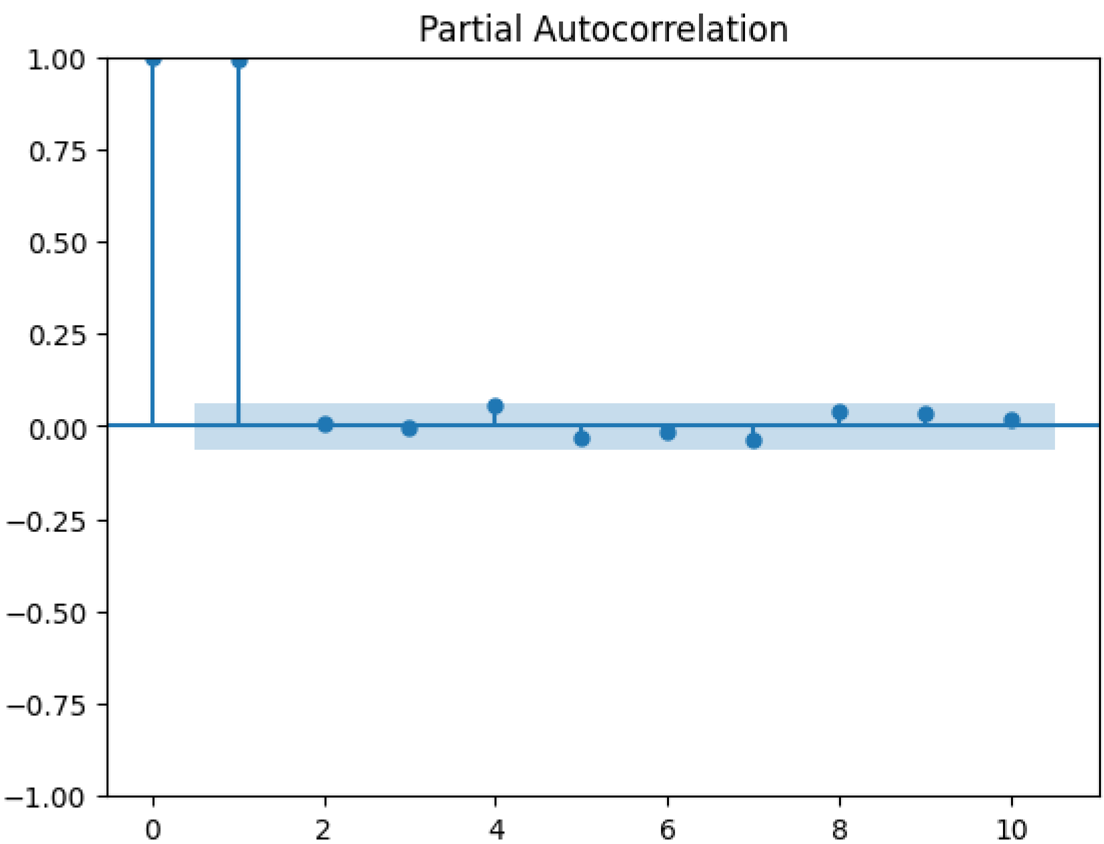


b) Para responder al apartado anterior se utilizan los gráficos de autocorrelación y autocorrelación parcial. Obtenerlos para el precio de cierre. Como ayuda ver el video ACF & PACF Code Example : Time Series Talk

https://www.youtube.com/watch?v=y8opUEd05Dg

Sacamos el gráfico de autocorrelación parcial:

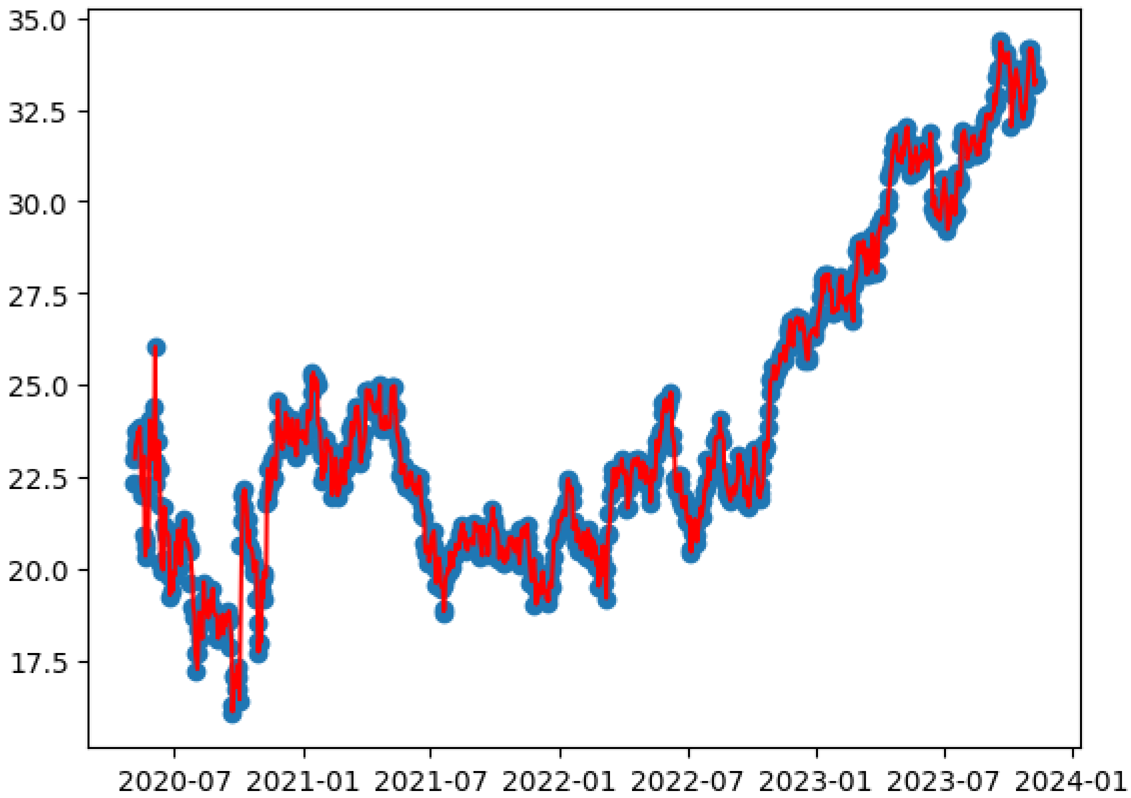
In [ ]:
```
from statsmodels.graphics.tsaplots import plot_pacf
pacf = plot_pacf(df_acs['Último'], lags=10)
```



Vemos aquí que a partir de *lag=2* la autocorrelación se mantiene, por lo que será el *lag* que utilizaremos.

c) Calcula de manera razonada el modelo AS-RLM del precio diario de cierre en función de los precios de cierre en los días anteriores. Es decir, el orden del modelo AR y la ecuación del modelo AR.

In [ ]:
```
ar_model_whole_2 = AutoReg(df_acs['Último'], lags=2).fit()
pred_whole = ar_model_whole_2.predict(start=df_acs.index.min(), end=df_acs.index

plt.plot(pred_whole, color='red')
plt.scatter(df_acs.index, df_acs['Último']);
```



d) Comparar lo anterior con las indicaciones de la web https://dataaspirant.com/stepwise-regression/ para proporcionar el modelo AS-RLM con selección de variables autorregresivas que utilizarías en el futuro. Interpreta dicho modelo.

En nuestro caso, lo que estamos haciendo más que un stepwise regression, es un análisis previo que nos permite saber el número de días hacia atrás que tenemos que mirar para poder estimar mejor el modelo. En este caso hemos visto claramente que el número idóneo es 2.

e) ¿Qué información se puede obtener sobre la predicción del precio de cierre del lunes 13 de Noviembre a un 95 % según la un modelo AS-RLM de tipo autoregresión o modelo AR?

In [ ]:
```
ar_model_whole_2.predict(start='13.11.2023', end='13.11.2023')
prediction = ar_model_whole_2.get_prediction(start='13.11.2023', end='13.11.2023
prediction.predicted_mean
```

Out[ ]:
```
2023-11-13    33.246561
Freq: B, Name: predicted_mean, dtype: float64
```

Nos realiza la predicción de que el precio de cierre el lunes 13 de Noviembre será de 33.246561. Comprobamos ahora el intervalo de confianza que nos da para la predicción:

In [ ]:
```
prediction.conf_int(alpha=0.05)
```

Out[ ]:

	lower	upper
2023-11-13	32.381532	34.111589

Ejercicio 3

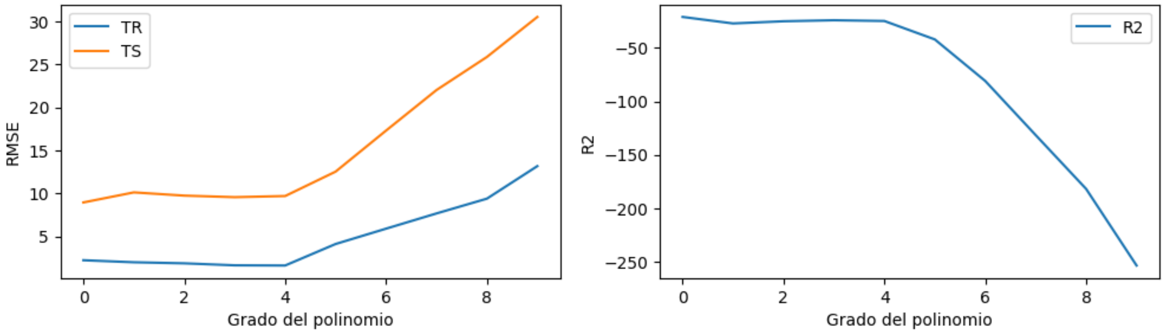
Ahora nos planteamos establecer una relación polinómica entre el precio de cierre (variación) diario y el precio de cierre (variación) del día anterior. Para decidir el orden del polinomio a utilizar se puede consultar

https://rohanmandrekar.netlify.app/post/overfitting-using-higher-order-linear-regression/

a) Adjuntar los gráficos y salidas correspondientes para tomar la decisión anterior

In [ ]:
```
batch_size=10
rmse_train = [None]*batch_size
rmse_test = [None]*batch_size
r2 = [None]*batch_size

X_train_a['DiasDesdeInicio'] = (X_train_a.index - X_train_a.index.min()).days
X_train = X_train_a['DiasDesdeInicio'].values.reshape(-1, 1)

X_test_a['DiasDesdeInicio'] = (X_test_a.index - X_test_a.index.min()).days
X_test = X_test_a['DiasDesdeInicio'].values.reshape(-1, 1)

for i in range(batch_size):
    polynomial_features= PolynomialFeatures(degree=i)
    xp_train = polynomial_features.fit_transform(X_train)

    modelo = sm.OLS(endog=X_train_a['Último'], exog=xp_train,).fit()

    xp_test = polynomial_features.fit_transform(X_test)
    predicciones_train = modelo.get_prediction(exog=xp_train).summary_frame(alph
    predicciones_test = modelo.get_prediction(exog=xp_test).summary_frame(alpha=
    rmse_train[i]=np.sqrt(mean_squared_error(X_train_a['Último'],predicciones_tr
    rmse_test[i]=np.sqrt(mean_squared_error(X_test_a['Último'],predicciones_test
    r2[i]=r2_score(X_test_a['Último'],predicciones_test['mean'])

plt.figure(figsize=(12,3))
plt.subplot(1,2,1)
plt.plot(rmse_train, label='TR')
plt.plot(rmse_test, label='TS')
plt.xlabel('Grado del polinomio')
plt.ylabel('RMSE')
plt.legend()
plt.subplot(1,2,2)
plt.plot(r2, label='R2')
plt.xlabel('Grado del polinomio')
```

```
plt.ylabel('R2')
plt.legend();
```



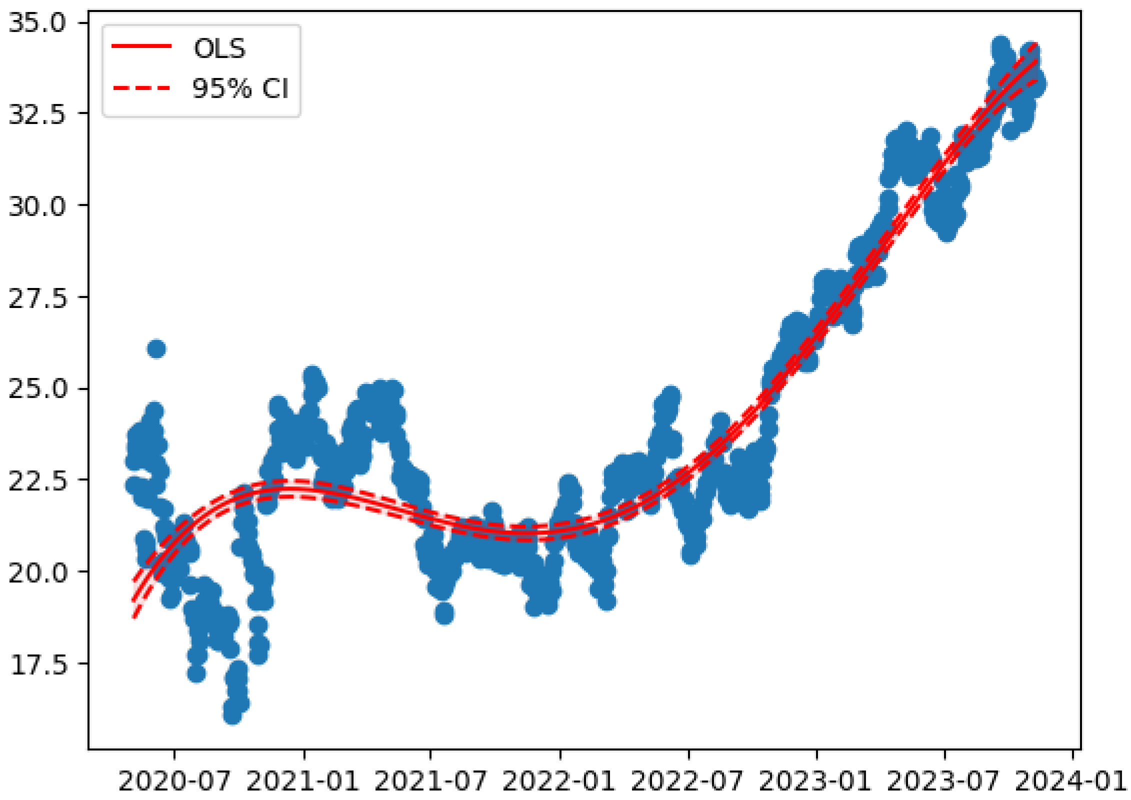
Con estas gráficas podemos ver cómo el grado óptimo es claramente 4, ya que ahí es cuando los errores de **TR** y **TS** son más bajos en conjunto. También hemos graficado el error R2 para comprobarlo y vemo que también desciende a medida que aumenta el grado del polinomio.

b) ¿Qué información se puede obtener sobre la predicción del precio de cierre del lunes 13 de Noviembre a un 95 % según la regresión polinómica?

In [ ]:

```
polynomial_features = PolynomialFeatures(degree=4)
df_acs['DiasDesdeInicio'] = (df_acs.index - df_acs.index.min()).days
X = df_acs['DiasDesdeInicio'].values.reshape(-1, 1)
xp = polynomial_features.fit_transform(X)

modelo = sm.OLS(endog=df_acs['Último'], exog=xp,).fit()
print(modelo.summary())
xp = polynomial_features.fit_transform(np.append(X,max(X)+1).reshape(-1,1))
predicciones = modelo.get_prediction(exog=xp).summary_frame(alpha=0.05)
predicciones['x'] = np.append(df_acs.index.values,max(df_acs.index.values)+1)
predicciones = predicciones.sort_values('x')
modelo.conf_int(alpha=0.05)

plt.scatter(df_acs.index,df_acs['Último'])
plt.plot(predicciones['x'], predicciones["mean"], linestyle='-', label="OLS", cc
plt.plot(predicciones['x'], predicciones["mean_ci_lower"], linestyle='--', color
plt.plot(predicciones['x'], predicciones["mean_ci_upper"], linestyle='--', color
plt.fill_between(predicciones['x'], predicciones["mean_ci_lower"], predicciones[
plt.legend();
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Último   R-squared:                       0.862
Model:                            OLS   Adj. R-squared:                  0.861
Method:                 Least Squares   F-statistic:                     1429.
Date:                Sat, 18 Nov 2023   Prob (F-statistic):               0.00
Time:                        19:24:32   Log-Likelihood:                -1718.7
No. Observations:                 920   AIC:                             3447.
Df Residuals:                     915   BIC:                             3471.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         19.2000      0.255     75.155      0.000      18.699      19.701
x1             0.0331      0.003     11.971      0.000       0.028       0.038
x2            -0.0001   8.75e-06    -13.196      0.000      -0.000   -9.83e-05
x3          1.379e-07   1.02e-08     13.461      0.000    1.18e-07    1.58e-07
x4         -4.752e-11   3.95e-12    -12.018      0.000   -5.53e-11   -3.98e-11
==============================================================================
Omnibus:                       28.949   Durbin-Watson:                   0.079
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               48.220
Skew:                          -0.248   Prob(JB):                     3.38e-11
Kurtosis:                       4.006   Cond. No.                     4.50e+12
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 4.5e+12. This might indicate that there are
strong multicollinearity or other numerical problems.
```




```
In [ ]: predicciones.iloc[-1]
```

```
Out[ ]: mean                                 31.005065
        mean_se                               0.105918
        mean_ci_lower                        30.797197
        mean_ci_upper                        31.212934
        obs_ci_lower                         26.056063
        obs_ci_upper                         35.954068
        x                2023-11-10 00:00:00.000000001
        Name: 920, dtype: object
```

Miramos la última predicción y podemos ver los valores de la estimación, así como la cota superior e inferior del intervalo de confianza.